OFweek 电子工程网 创新移动开发系列

Android

基础及典型案例开发指南

Android基础篇+经典开发案例：

- Android系统架构、系统移植、Debug调试、多线程、TCP/UDP、消息机制 、API
- 3D游戏开发、蓝牙通信、软硬件开发、SQLite详解、重力系统开发、模拟信号示波器、Tab标签传感器和语音识别

OFweek电子工程网 创新设计系列电子书

——基础篇——

——Android 简介

Android 一词的本义指“机器人”，同时也是 Google 于 2007 年 11 月 5 日宣布的基于 Linux 平台的开源手机操作系统的名称，该平台由操作系统、中间件、用户界面和应用软件组成，号称是首个为移动终端打造的真正开放和完整的移动软件。目前最好的是 Android2.0 的摩托罗拉 Droid。

Android 公司介绍

LOGO 国家：美国

业务：手机软件，操作系统

成立于：2003 年

创办人：Andy Rubin，Andy McFadden、Richard Miner Chris White

中文名：安致

注：Google 2005 年 8 月 17 日收购美国 Android 公司得到手机巨头摩托罗拉的支持，摩托罗拉 2010 年放弃其他的操作系统（包括自家的 Linux 和 UIQ）只支持 Android。Android 的特色业务是手机软件，之后 Google 在其基础上发展了 Android 业务。2010 年 Google 和摩托罗拉发布 Android2.2 手机——NEXUS TWO，在 Android 发展的过程中，摩托罗拉付出的是核心代码，Google 付出的是公关和品牌效应，当然还有它的 google app，但是 Google 掌握了 Android Market 以及通过 android google apps 获得的大量用户。

Android 操作系统简介

Android 是 Google 于 2007 年 11 月 05 日宣布的基于 Linux 平台的开源手机操作系统的名称，该平台由操作系统、中间件、用户界面和应用软件组成。它采用软件堆层（Software Stack，又名软件叠层）的架构，主要分为三部分。底层以 Linux 内核工作为基础，由 C 语言开发，只提供基本功能；中间层包括函数库 Library 和虚拟机 Virtual Machine，由 C++开发。最上层是各种应用软件，包括通话程序，短信程序等，应用软件则由各公司自行开发，以 Java 作为编写程序的一部分。不存在任何以往阻碍移动产业创新的专有权障碍，号称是首个为移动终端打造的真正开放和完整的移动软件。

Google 通过与软、硬件开发商、设备制造商、电信运营商等其他有 关各方结成深层次的合作伙伴关系，希望借助建立标准化、开放式的移动电话软件平台，在移动产业内形成一个开放式的生态系统。

Android 作为 Google 企业战略的重要组成部分，将进一步推进“随时随地为每个人提供信息”这一企业目标的实现。全球为数众多的移动电话用户正在使用各种基于 Android 的电话。谷歌的目标是让（移动通讯）不依赖于设备甚至平台。出于这个目的，Android 将补充，而不会替代谷歌长期以来奉行的移动发展战略：通过与全球各地的手机制造商和移动运营商结成合作伙伴，开发既有用又有吸引力的移动服务，并推广这些产品。

开放手机联盟

为了推广 Android 平台技术，Google 和几十个手机相关企业建立了开放手机联盟（Open Handset Alliance）。

联盟成员，包括摩托罗拉 （Motorola）、HTC、Samsung、LG、Intel、nVdia、SiRF、HP、Skype、KUPA Map 以及中国移动在内的 34 家技术和无线应用的领军企业，都将基于该平台开发手机的新型业务，应用之间的通用性和互联性将在最大程度上得到保持。34 家相关企业的加盟，也将大大降低新型手机设备的研发成本，完全整合的“全移动功能性产品”成为“开放手机联盟”的最终目标。

这 34 家企业中并不包含把持 Symbian 的 Nokia 公司，以及凭借着 iPhone 风光正在的 Apple 公司，美国运营商 AT&T 和 Verizon，当然微软没有加入，独树一帜的加拿大 RIM 和他们的 Blackberry 也被挡在门外。

“开放手机联盟”表示，Android 平台可以促使移动设备的创新，让用户体验到最优越的移动服务，同时，开发商也将得到一个新的开放级别，更方便的进行协同合作，从而保障新型移动设备的研发速度。

特性

- 应用程序框架 支持组件的重用与替换
- Dalvik 虚拟机 专门为移动设备做了优化
- 内部集成浏览器 该浏览器基于开源的 WebKit 引擎
- 优化的图形库 包括 2D 和 3D 图形库，3D 图形库基于 OpenGL ES 1.0 （硬件加速可选）
- #SQLite 用作结构化的数据存储
- 多媒体支持 包括常见的音频、视频和静态印象文件格式 （如 MPEG4， H.264， MP3， AAC， AMR， JPG， PNG， GIF）
- GSM 电话 （依赖于硬件）
- 蓝牙 Bluetooth， EDGE， 3G， and WiFi （依赖于硬件）
- 照相机，GPS，指南针，和加速度计 （依赖于硬件）
- 丰富的开发环境 包括设备模拟器，调试工具，内存及性能分析图表，和 Eclipse 集成开发环境插件

架构

下图显示的是 Android 操作系统的主要组件。每一部分将会在下面具体描述。

Android 架构

应用程序

Android 会同一个核心应用程序包一起发布，该应用程序包包括 email 客户端，SMS 短消息程序，日历，地图，浏览器，联系人管理程序等。所有的应用程序都是用 JAVA 编写的。

Android 应用程序框架开发者也完全可以访问核心应用程序所使用的 API 框架。该应用程序架构用来简化组件软件的重用；任何一个应用程序都可以发布它的功能块并且任何其它的应用程序都可以使用其所发布的功能块（不过得遵循框架的安全性限制）。该应用程序重用机制使得组件可以被用户替换。

以下所有的应用程序都由一系列的服务和系统组成，包括：

• 一个可扩展的视图（Views ）可以用来建应用程序，包括列表（lists），网格（grids），文本框（text boxes），按钮（buttons），甚至包括一个可嵌入的 web 浏览器

• 内容管理器（Content Providers ）使得应用程序可以访问另一个应用程序的数据（如联系人数据库），或者共享它们自己的数据。

• 一个资源管理器 （Resource Manager）提供非代码资源的访问，如本地字符串，图形，和分层文件（ layout files ）。

• 一个通知管理器 （Notification Manager）使得应用程序可以在状态栏中显示客户通知信息。

• 一个活动类管理器 （Activity Manager）用来管理应用程序生命周期并提供常用的导航回退功能。

有关更多的细节和怎样从头写一个应用程序，请参考 写一个 Android 应用程序 部分。

Android 程序库Android 包括一个被 Android 系统中各种不同组件所使用的 C/C++ 库集。该库通过 Android 应用程序框架为开发者提供服务。以下是一些主要的核心库：

系统 C 库 - 一个从 BSD 继承来的标准 C 系统函数库（ libc ），专门为基于 embedded linux 的设备定制。 媒体库 - 基于 PacketVideo OpenCORE；该库支持录放，并且可以录制许多流行的音频视频格式，还有静态印像文件包括 MPEG4， H.264， MP3， AAC， AMR， JPG， PNG。 Surface Manager - 对显示子系统的管理，并且为多个应用程序提供 2D 和 3D 图层的无缝融合。 LibWebCore - 一个最新的 web 浏览器引擎用来支持 Android 浏览器和一个可嵌入的 web 视图。 SGL - 一个内置的 2D 图形引擎 3D libraries - 基于 OpenGL ES 1.0 APIs 实现；该库可以使用硬件 3D 加速（如

果可用）或者使用高度优化的 3D 软加速。 FreeType - 位图（bitmap）和向量（vector）字体显示。 SQLite - 一个对于所有应用程序可用，功能强劲的轻型关系型数据库引擎。

Android 运行库

Android 包括了一个核心库，该核心库提供了 JAVA 编程语言核心库的大多数功能。

每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。Dalvik 是针对于同时高效地运行多个 VMs 来实现的。Dalvik 虚拟机执行.dex 的 Dalvik 可执行文件，该格式文件针对最小内存使用做了优化。该虚拟机是基于寄存器的，所有的类都经由 JAVA 汇编器编译，然后通过 SDK 中的 dx 工具转化成.dex 格式由虚拟机执行。

Dalvik 虚拟机依赖于 linux 的一些功能，比如线程机制和底层内存管理机制。

Linux 内核 Android 的核心系统服务依赖于 Linux 2.6 内核，如安全性，内存管理，进程管理，网络协议栈和驱动模型。 Linux 内核也同时作为硬件和软件堆栈之间的硬件抽象层。未来发展

老牌智能手机软件平台制造商 Symbian 发言人则表示：Google 的 android 只不过是另一个 linux，symbian 对其它软件与其形成的竞争并不感到担心。除了北美之外，Symbian 在其它地区智能手机市场都占有大部分市场份额。

与 iPhone 相似，Android 采用 WebKit 浏览器引擎，具备触摸屏、高级图形显示和上网功能，用户能够在手机上查看电子邮件、搜索网址和观看视频节目等，比 iPhone 等其他手机更强调搜索功能，界面更强大，可以说是一种融入全部 Web 应用的单一平台。

但其最震撼人心之处在于 Android 手机系统的开放性和服务免费。Android 是一个对第三方软件完全开放的平台，开发者在为其开发程序时拥有更大的自由度，突破了 iPhone 等只能添加为数不多的固定软件的枷锁；同时与 Windows Mobile、Symbian 等厂商不同，Android 操作系统免费向开发人员提供，这样可节省近三成成本… 继续阅读文章

——怎样使用 Eclipse 来开发 Android 源码

- 用 eclipse＋ADT 作为 android 开发工具，可以说是很方便的，在 HelloActivity 小程序里我们就感觉到 eclipse 功能的强大。那么，我们可以用 eclipse 来开发 android 源码吗？如果我们直接把 android 源码里一个工程导入 eclipse，一般来说都会出现错误，说许多类库（包）找不到。

今天找到关于怎样使用 eclipse 来开发 android 源码的官方文档：

https://sites.google.com/a/android.com/opensource/using-eclipse

从该文档和实践可以总结出几点：

1、可以使用 eclipse 来编辑 JAVA 程序、检查错误（主要是类库包含和语法方面），但是不能在 eclipse 上编译运行 android 源码，还是得在 shell 中 make（或 mm 或 mmm）

2、android 源码文件夹里提供有一些 eclipse 配置文件，

.claapath：eclipse 工程的配置文件，方便我们直接把 android 源码相应的文件和 JAVA 包导入工程

android-formatting.xml 和 android.importorder：这个很重要，主要是用来规范我们的编码风格，更容易使我们的代码风格一致

3、把 android 源码作为一个工程导入 eclipse 时，必须注意两点

1）、新建的工程必须是 java project，不能是 android project，否则会破坏 android 源码（一般是多添加文件/文件夹）

2）、导入前最好检查.classpath 里的文件在 android 源码中是否有相应的文件（文件夹），否则也会破坏 android 源码（一般是多添加文件/文件夹）

总的来说：

1、用 eclipse 来编辑代码、检查错误

2、不在 eclipse 上编译、运行 android 源码程序，只能在命令行通过 make（或 mm 或 mmm）编译 android 源码

3、可以在 eclipse 上调试 android 源码程序（原理：eclipse 通过 ddms 服务器在 emulator 上进行调试），并可以单步调试、断点调试。

下面，从官方文档总结出具体怎样用 eclipse 来开发 android 源码

1、建立基本的 android 开发环境

请参考官方文档或＜android 模拟器在 ubuntu8.10 的安装＞

2、编译 android 源码

android 源码根目录下通过 make 进行编译，请注意一些配置，具体可参考＜android 源码的编译＞

3、把 eclipse 工程配置文件复制到 android 源码根目录下

cp development/ide/eclipse/.classpath 。/

chmod u+w .classpath # Make the copy writable

4、修改 eclipse 程序的配置

1）、修改 eclipse 缓存设置

把 eclipse.ini（在 eclipse 软件的安装目录下）的 3 个值改为下面的值：

-Xms128m

-Xmx512m

-XX:MaxPermSize=256m

2）、把 android-formatting.xml 和 android.importorder 导入 eclipse

android-formatting.xml、.classpath 和 android.importorder 都放在 development/ide/eclipse/下

android-formatting.xml 用来配置 eclipse 编辑器的代码风格；android.importorder 用来配置 eclipse 的 import 的顺序和结构。

在 window-》preferences-》java-》Code style-》Formatter 中导入 android-formatting.xml

在 window-》preferences-》java-》Code style-》Organize Imports 中导入 android.importorder

3）、安装 anyedit 插件（可选）

在 http://andrei.gmxhome.de/anyedit/下载并导入 eclipse 中

5、把 android 源码作为一个工程导入 eclipse

导入前先检查.classpath 里的文件在 android 源码中是否有相应的文件（文件夹），否则也会破坏 android 源码（一般是多添加文件/文件夹），.classpath 里多余的路径可删除

新建 Java Project（不是 android project，否则会破坏 android 源码），选择从已存在的工程导入，工程名任意，完成。

导入时，eclipse 要 build 工程，比较慢。导完后，一般都没有错误。

6、eclipse 上调试 android 里的程序…

继续阅读文章

——Android 系统的移植要做的两个工作

- Android 系统的移植的主要目的是为了能在特定的硬件上运行 Android 系统。而在移植的过程中，一个重要的方面就是把握关键要点，减少工作量。从工作的角度，首先要熟悉硬件抽象层的接口，其次要集成和复用已有的驱动程序，主要的工作量在硬件抽象层的实现中。为了更好地理解和调试系统，也应该适当地了解上层对硬件抽象层的调用情况。

  移植方面主要的工作有两个部分：

  Linux 驱动

  Android 系统硬件抽象层

  Linux 中的驱动工作在内核空间，Android 系统硬件抽象层工作在用户空间，有了这两个部分的结合，就可以让庞大的

Android 系统运行在特定的硬件平台上。

Android 移植的主要工作如图 1 所示。

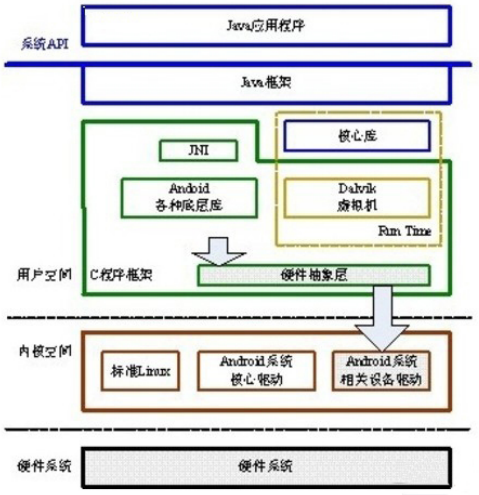


图 1 Android 移植的主要工作

在具有了特定的硬件系统之后，通常在 Linux 中需要实现其驱动程序，这些驱动程序通常是 Linux 的标准驱动程序，在 Android 平台和其他 Linux 平台基本上是相同的。主要的实现方面是 Android 系统中的硬件抽象层（HardwareAbstract Layer），硬件抽象层对下调用 Linux 中的驱动程序，对上提供接口，以供 Android 系统的其他部分（通常为 Android 本地框架层）调用。

提示：Android 硬件抽象层的接口是本地移植层的接口，不属于标准 API，不具有向前或者向后兼容性。

在 Android 系统需要移植的内容，主要包含了以下的各个部分：

显示部分（Display）

包括 framebuffer 驱动+Gralloc 模块（可选择是否实现）

用户输入部分（Input）

包括 Event 驱动+EventHub（Android 标准内容）

多媒体编解码（Codec）

包括硬件 Codec 驱动+Codec 插件（如 OpenMax）

3D 加速器部分（3D Accelerator）

包括硬件 OpenGL 驱动+OpenGL 插件

音频部分（Audio）

包括 Audio 驱动+Audio 硬件抽象层

视频输出部分（Video Out）

包括视频显示驱动+Overlay 硬件抽象层

摄像头部分（Camera）

包括 Camera 驱动（通常是 v4l2）+Camera 硬件抽象层

电话部分（Phone）

Modem 驱动程序+RIL 库

全球定位系统部分（GPS）

包括 GPS 驱动（通常为串口）+GPS 硬件抽象层

无线局域网部分（WIFI）

包括 Wlan 驱动和协议+WIFI 的适配层（Android 标准内容）

蓝牙部分（Blue Tooth）

包括 BT 驱动和协议+BT 的适配层（Android 标准内容）

传感器部分（Sensor）

包括 Sensor 驱动+Sensor 硬件抽象层

震动器部分（Vibrator）

包括 Vibrator 驱动+Vibrator 硬件抽象层（Android 标准内容）

背光部分（Light）

包括 Light 驱动+ Light 硬件抽象层

警告器部分（Alarm）

包括 Alarm 驱动和 RTC 系统+用户空间调用（Android 标准内容）

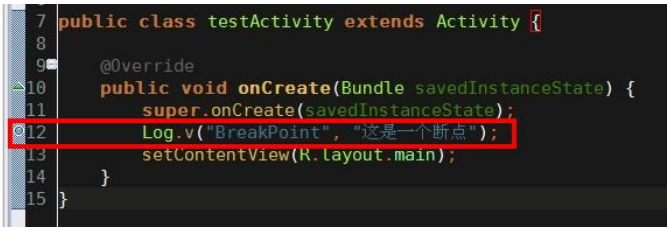
电池部分（Battery）

包括电池部分驱动+电池的硬件抽象层（Android 标准内容） 继续阅读文章

——Android 开发之 Eclipse Debug 调试详解

- **1.在程序中添加一个断点**

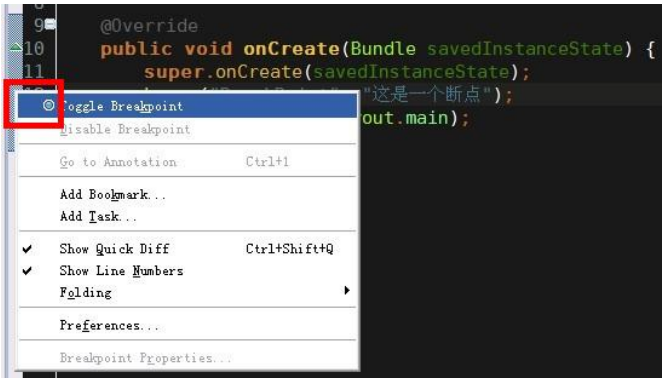
如果所示：在 Eclipse 中添加了一个程序断点



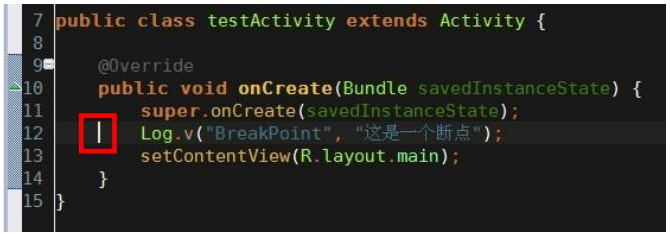
在 Eclipse 中一共有三种添加断点的方法

第一种： 在红框区域右键出现菜单后点击第一项 Toggle Breakpoint 将会在你右键代码的哪一行添加一个程序断点（同样的操作方可取消程序断点）

第二种： 在红框区域双击鼠标左键将会在你双击代码的哪一行添加一个程序断点 （同样的操作方可取消程序断点）



第三种 ：在光标停留的地方使用快捷键 Ctrl + Shift + B 将会在光标停留的这一行添加一个程序断点 （同样的操作方可取消程序断点）

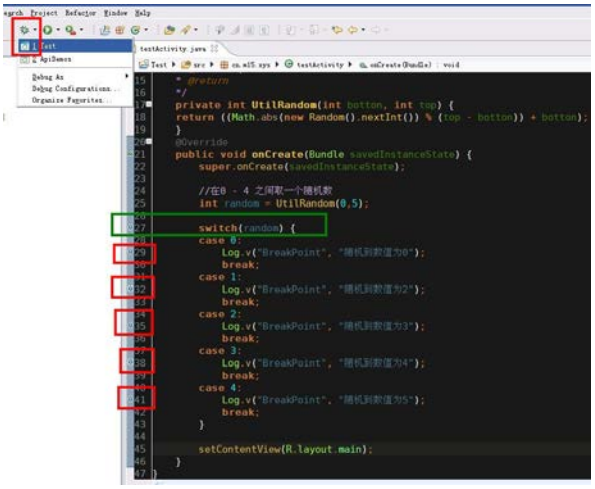


2.运行 Debug 调试 让程序停留在添加的断点上

如下图所示，在红框内点击下拉菜单选中需要调试的项目 则开始运行 Debug 调试

如果不在下拉表中选直接点击表示 Debug 运行默认项目（默认项目为上一次运行的项目）

Debug 调试 快捷键为单击 F11



分析一下如何科学的添加程序断点， 上图中我为了加断点查看生成出来随机数的值我一共添加了 6 个程序断点，绿框表示最为科学的断点位置 ，红框表示不科学的位置。 我们分析一下为什么， 如果 switch case 中的代码片段过长 或者 case 的数量过多 如果采用红框的方式来添加程序断点，程序员须要添加很多程序断点万一有疏漏 所以会很难快速定位代码执行到了那里 ，如果使用绿框的方式添加程序断点，程序员只须要在断点出按 F6 单步跳过这一行代码就会走进正确的 case 中方便继续调试。

Debug 调试运行后，程序停在了红框处，按 F6 单步跳过 发现随机数为 4 程序停留在了绿框中，程序员可以迅速定位 random 的值为 4

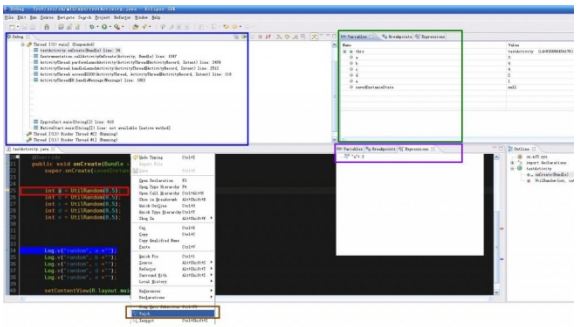
- **3.程序停留后查看变量的数值**

蓝框中的内容表示为断点的入口方法， 就好比你的断点是从那个方法进来的，学会看这个真的非常重要， 好比我现在明确知道我的一个方法在被调用的时候方法中会出现错误，但是这个方法在程序中 100 个地方都在调用，我可能断定实在那里调用的时候出的错误，我不可能在 100 个调用它的地方都加一个断点，我可以在方法中添加程序断点 然后在篮框中查看程序是从那个地方走进这个方法的，便可以快速定位问题所在。

绿框中可以查看当前方法中所有变量的值，但是如果变量非常多在这里看就比较麻烦，可以使用红框的方法查看。

红框中可以右键变量名点击咖啡框中的 watch 后 在紫框中 Expressions 就可以看到变量的数值了。

BreakPoints 中会记录程序中添加过多少程序断点。



4 分享一些 Eclipse 中 Debug 的一些小技巧

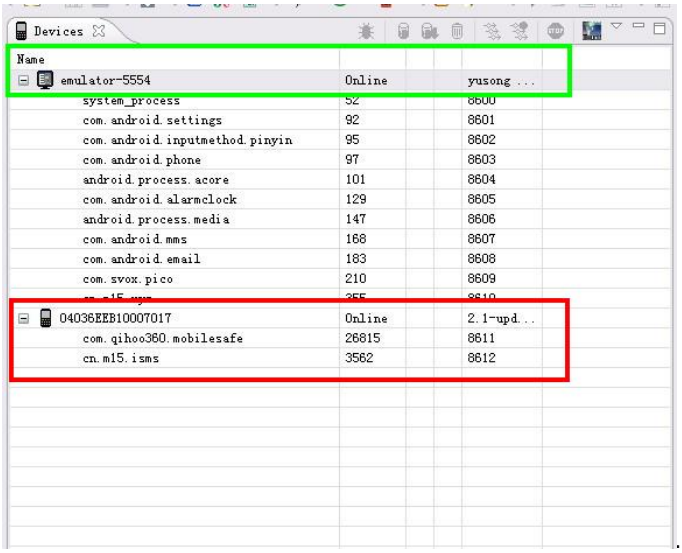
watch 过的变量 和我们自己加的程序断点不会被 Eclipse 自动删除 除非我们手动删除否则会一直留在紫框中，这些数值会拖慢 Eclipse 开发工具，如果过多的话很可能会造成 Eclipse 崩溃（有可能是 Eclipse 的 BUG），让开发变得非常痛苦，所以雨松 MOMO 在这里建议大家在每次 Debug 调试的时候将紫框中之前 加的程序断点 和 watch 过的变量 全不手动清空，只添加这一次调试须要的断点就可以了，这样的话 Eclipse 就不会被这些拖慢进程的东西所导致崩溃。

5.连接真机调试

第一步 打开自己的手机在设置中选择应用程序 然后选择开发 然后选中 USB 调试。

第二步 用 USB 线连接手机到电脑，一般情况会自动安装驱动，如果无法安装驱动的话 就去下载一个豌豆荚 或者 91 助手，让它帮我们手机自动安装驱动 很方便的。

第三步 驱动安装成功后会在 Device 中看到真机（红框中） 绿框中为 android 电脑模拟器



运行项目后弹出设备选择窗口 第一个为模拟器 第二个红框内的为我连接电脑的真机 MOTO 的里程碑，选择完后点击 OK 就可以通过真机来调试程序了，简单吧？

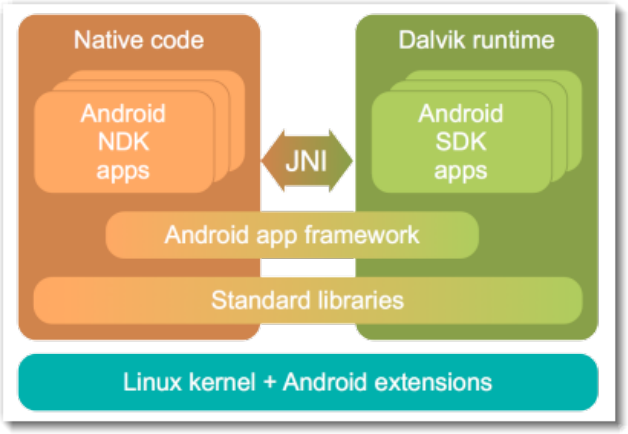
继续阅读文章

——Google 工程师多图详解 Android 系统架构

- 近日，Google 的一位工程师 Sans Serif 发布了一篇博文非常清楚的描述了 Android 系统架构，中国移动通信研究院院长黄晓庆在新浪微博上推荐了该文，并认为文中对 Android 的介绍很好，您可以看一下 Google 工程师眼中的 Android 系统架构是什么样的。以下为 Sans Serif 博文的译文：

Andriod 是什么？

首先，就像 Android 开源和兼容性技术负责人 Dan Morrill 在 Android 开发手册兼容性部分所解释的，“Android 并不是传统的 Linux 风格的一个规范或分发版本，也不是一系列可重用的组件集成，Android 是一个用于连接设备的软件块。”



Andriod 是什么？

Linux

所有东西的底层是一个稳定的保持更新的 Linux 内核（我现在用的 Nexus 手机所用的就是 2.6.32 版的内核），以及我们精心打造的能源管理组件；当然还有将它们整合至上层 Linux 代码的扩展和公共组件。

Dalvik

Android 另一个重要的部分，包括虚拟机和一组重要的运行环境。它的设计非常巧妙，是个很好的一个手机终端的底层应用。

代码如何生成？

Dalvik 虚拟机只执行.dex 的可执行文件。当 Java 程序通过编译，最后还需要通过 SDK 中的工具转化成.dex 格式才能在虚拟机上执行。

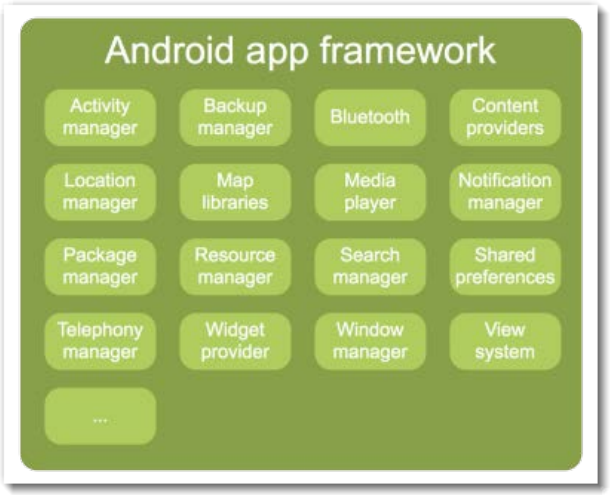
我需要强调的是，Android 应用本身就可视作可在平台上运行并调用 APIs 的代码，所以对代码如何生成不需特别看重。

特别的 Apps

在图中有些基于 Dalvik 虚拟机的 Apps 看起来像是 Android 的一部分，其实是由 Google 提供，这些应用包括 Dialer、Contact、Calendar、Gmail 和 Chat 等。它们中的绝大部分是开源并可复用的。只有少部分例外，比如 Google Maps 和 Android Market.

开源那些事

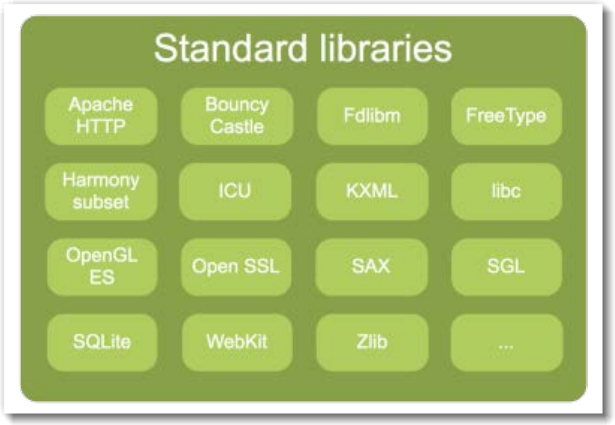
在下面的图中，绿色的大部分组件是基于 Apache 许可证开源，其余基于 GPL、LGPL 和 BSD。



开源的 Android

Android 框架

在 Android 开发者网（developer.android.com）上已有不少篇幅来帮助你使用它，在此就不再累述。



Android 框架

标准库

在这里“标准”是指“开发者在开源环境中一般可以使用的”。

App 里面是什么

一个 Android App 包含在一个我们称之为 APK 的压缩文件夹中，APK 并没有什么可说的，需要注意的是 Android Manifest——介于 App 和 Android System 的接口。 继续阅读文章

——如何在 Android 中用好多线程

- 首先我们思考几个问题，在Android应用中为什么要用多线程？为了解决哪些问题？或者为了实现哪些功能？有哪些好处？请先思考一分钟，再继续往下看。

  学习而不思考就像吃东西而不嚼，要么无法下咽，要么尝不出味道，同时都会影响消化吸收。控制一下你那脱缰野马一样的好奇心吧，先思考再往下看。

1. 为什么要用多线程

这里列出几个原因：

a） 提高用户体验或者避免 ANR

在事件处理代码中需要使用多线程，否则会出现 ANR（Application is not responding），或者因为响应较慢导致用户体验很差。

图 1 ANR 对话框

b） 异步

应用中有些情况下并不一定需要同步阻塞去等待返回结果，可以通过多线程来实现异步，例如：上一点中提到的，你的应用中的某个 Activity 需要从云端获取一些图片，加载图片比较耗时，这时需要使用异步加载，加载完成一个图片刷新一个，见下面图 2、图 3 。

c） 多任务

例如多线程下载。

后两点与 Java 中的多线程应用没有太大区别，不细说。

下面重点说明第一点，即如何减少事件响应的时间从而提高用户体验，以及如何避免 ANR。

2. 为什么通过多线程可以提高用户体验、避免 ANR

大家还记得我在群里说过的移动开发的“三不要”原则吗？即：不要让我想、不要让我等、不要让我烦。响应慢了用户需要等，等的次数多了就会烦，你的应用离被卸载不远了。

首先我们来了解一下 Android 应用程序的 main 线程，它负责处理 UI 的绘制，Android 系统为了防止应用程序反应较慢导致系统无法正常运行做了一个处理，一种情况是当用户输入事件在 5 秒内无法得到响应，那么系统会弹出 ANR 对话框，由用户决定继续等待还是强制结束应用程序（另一种情况是 BroadcastReciever 超过 10 秒没执行完也会弹出 ANR 对话框）。

即使你的程序中某个事件响应不超过 5 秒钟，人眼可以分辨的时间是 0.1 秒，小于 0.1 秒基本感觉不出来，超过 0.2 秒用户就能感觉到有点儿卡了，俗称打嗝现象，2 秒以上就很慢了，用户体验会很差。有同学说我可以用进度条啊，但你的程序中不能到处都是进度条，否则那个圈圈会把用户转晕的，好像在对用户说，画个圈圈烦死你……

比如某些应用，它要显示很多图片，还好它是异步的，不过在图片加载完成前每个图片的位置上都有一个圈圈，让人看了很烦。你可以变通一下，图片加载成功之前显示一个默认的图片，加载成功后再刷新一下即可，何必弄那么多进度条呢？

图 2 加载图片完成前显示默认图片，加载完成后再刷新

图 3 加载图片完成前显示默认图片，加载完成后再刷新

——深入理解 Android

- **序**

携来百侣曾游，忆往昔峥嵘岁月稠。 -- 《沁园春·长沙》

对于 Android，我也算是老人了，所谓，有文有真想。正由于这段玩票经历，使得我在毕业后，鬼使神差的成为移动平台的一名码工，再次有机会放肆的拥抱 Android。

2010 开年，手上突然有了一把闲散时间，有机会进一步总结和学习 Android。于是想再一次为 Android 写一系列的东西，这些东西来自于一些开发经验，对源码的学习和对 Android 的浅薄认识，也算是鞭笞自己学习的一种手段。

其下所有内容，预计有十数篇，抑或更多。基本和技术相关，也许会配有一些其他相关比较闲淡的话题。可能会有一些具象实例，但更多的可能是自己的一些理解和认知。所有一切，源自于妄自挖掘，难免有疏漏或误解，观者淡定。

以此为序，有心者，望共勉。

Android 简史

人生若只初识，何事秋风悲画扇。 -- 《木兰辞》

要说当今移动平台的当红辣子鸡，Android 说它是第二，也许没有别家敢认这个第一（好吧，iPhone，有意见就说。..）。了解 Android 开发平台的过去和现状，除了往下看，另有便捷的方式就是在 WikiPedia 中键入 Android，在这里，特此鸣谢 GFW 友情放生。

诞生

早在 2005 年 7 月，Google 舞动着手中的美刀，收下了由 Andy Rubin（传说中的 Android 之父。..）等人创立的一家小公司，他们当时做的就是基于 Linux 内核的手机操作系统，也就是小时候的 Android。经过 Google 多年打磨，Android 在 07 年末，正二八经的粉墨登场开门接客。

自打一出生，Android 便被钉板在富二代的角色上，不仅是因为老爹有钱的令人发指，也是因为其后有一帮金光闪耀的叔伯们保驾护航。这个叔伯群，便是响当当的开放手机联盟 OHA（Open Handset Alliance）。这个联盟涵盖了中国移动、T-Mobile、Sprint 这样的移动运营商，也包括 HTC、Motolora、三星这些的手机制造商，同时还有以 Google 为代表的手机软件商，以 Inter、Nvidia 为标志的底层硬件厂商和 Astonishing Tribe 这样的商业运作公司（这公司是啥我也不晓得。..）。作为后援团，他们理论上的任务，是各尽其长，全力捧红 Android，实际上的任务是齐心协力，借 Android 东风赚一个盆钵满响。

当然，Android 自所以被万众瞩目一炮走红，不仅仅是丫实在太有背景，同时，它也有这太多的新鲜的概念。Android 是一个开源的平台（恩，真正的全面开源，是在发布后很久以后了。..），它给那些捂着自家平台源码当宝的竞争对手们一记当头棒喝。Android 自行研发了一套 Java 虚拟机，当时仅提供 Java API 的支持（NDK 是更久以后的事情了。..），号称

为专为高端智能设计。Android 开发环境支持所有主流操作系统平台，包括 Windows，Linux，Mac，即便到今天，在手机开发中也是极其罕见的。Android 的带头人 Google，本身是做网络起家，Android 内嵌大量 Google 网络应用，听上去就显得很酷。这所有的一切，共同造就了 Android 那鹤立鸡群，不染风尘的少侠形象。

造势

推出伊始，Google 还有一个很震撼的推广举动，就是举行所谓的 Android 程序挑战赛（Android Developer Challenge，ADC）。整个比赛分成两场，第一场（ADC1）比赛，在没有任何真机问世，SDK 还是个雏形的状况下，便鸣金开锣了。

比赛套路是无差别的群殴，基本概念是无论你来自何方（还是要满足美国法律要求和避嫌要求的），无论你想做些什么，也不管你是光杆司令还是流氓团伙，只要提交一个能在模拟器上跑起来的程序，就可参赛。而比赛只是对你作品进行参观评比，作品的所有权依然放在开发者的口袋中。

当然，这还不算什么创新，NB 的是无比丰厚的奖金，整个 ADC 的奖金高达 1 千万美刀，每场各半，基本上首轮入围奖（前 50）已经超越了那时候一般程序比赛的头名奖金，这对很多小公司和个人而言，无疑是具有很强吸引力的。于是，各路打酱油好手蜂拥而至，各论坛、博客、网站也七嘴八舌的讨论开来，一时间，满城风雨。

ADC1 我也很厚颜无耻的参加了，结果当然可以预想，一毛钱都没摸上。回想整个过程，差距最大的并不是在技术上，而是认知上，我们玩的产品是人家几年前玩剩下的，说创新只是一抹笑谈。

当时觉着，Google 太 NB 了，ADC 这种车马未动粮草先行的招太华丽了，就这动静，不论比出来个啥结果，这 1 千万刀也掏值了。但时过境迁，现在回头来想，也许一切并不是看上去那么美。由于没有扎扎实实的真机摆出来，大家普遍抱着一种玩票的心理，真的敢不顾一切舍下身家性命押宝在 Android 上的尽在少数，这就锻造了 Android 平台很长一段时间的只见雷不见雨的局面。而等喧嚣过后，很多人热情消退，Google 真端出 Android 真机的时候，还需要重新热场再来一次，也许，真的有些得不偿失。

困境

所有的东西现在来说，都是事后诸葛亮，只能听个响不能当个真。而真实的状况是 ADC1 很快进入囧境，由于架构设计上的种种原因，Google 花了比预想多的多的时间做 Android 的优化工作，ADC1 比赛被迫不断延期，彻底沦为懒婆娘的裹脚布。各路曾热捧 Android 的媒体，也不失时机的倒戈，亲手在自己画上的感叹号后面，重重画上了个大大的问号。

祸不单行，同样是由于 Android 的性能问题，虽然各路高手把 Android 移植到了不同的手机平台上，但传说中的 GPhone 一直难产中，使得人们不免有了胎死腹中的猜测。

与此同时，其他对手可一点也没闲着。iPhone 很快宣布开放 SDK，以此来勾引纯情的开发人员。Symbian 被 Nokia 彻底收购，成为 Nokia 的自留地，开源计划也很快浮出水面。所有景象，对 Android 而言都犹如梦魇。

破茧

所有一切困境，都在 G1 发布后，渐渐消散了。2008 年 10 月，第一款搭配 Android 平台的真机，搭载着无限光荣与梦想的 HTC Dream 正式发售，这就是注定要载入史册的 G1。虽然比之当时绝代风华的 iPhone，粗陋的 G1 犹如村姑遇上公主一般，但无论如何，G1 让人们真真切切的看到了 Android。这就犹如你家买的跳票 N 久的期房，终于见着了个毛胚房，那种感觉，除了踏实，找不到更适合的词汇了。

好事当然也会成双，G1 它不是一个人在战斗。ADC1 总算是落下帷幕，Android Market 的也顺理成章的破茧而出，早期的应用，大都来自于 ADC1 的贡献。

Android 也结束了伪开源的历史旅程，正式开发 SDK 的源码实现，搭配的是 Apache 的 License，这种坦诚相见的感觉看上去很不错。

忠心耿耿的 HTC，更是再接再励，在 G1 后，陆续发布了 Magic（G2）和 Hero（G3）。尤其是 Hero 的现身，惹得一阵小惊艳，HTC 为 Hero 搭配的是基于 Android 改造 UI 的 Sense 系统，以华丽的界面风格赚足了眼球，也创了改造 Android 的先河。

在 HTC 高歌猛进的同时，猫在螳螂后的一群黄雀，也敌在动我也动了。摩托罗拉，三星，LG，华为，戴尔，联想等一干手机厂商纷纷跟进，各式各样的 Android 蜂拥而至。与此同时，其他嵌入式厂商也推陈出新，爱可视（Archos）发布了基于 Android 的平板设备，明基的 Android 上网本也是箭在弦上，而基于 Android 的手持电子书阅读设备也不断的被推出，庞大的 Android 联盟初现峥嵘。

为了避免同质化，各个厂商纷纷对 Android 进行的改造，摩托罗拉推出了 Cliq，打得是 SNS 整合牌，三星的新系统也是被广泛期待，而中移动的 OMS 丑媳妇也要见婆娘了，打着整合移动服务牌@_@的 OMS，以丑陋的外貌、低下的 SDK 版本和雷死人不偿命的宣传文案（绝口不提 Android，只说自己做了大量很 NB 的工作，其实。..，哎，咋就那么小家子气呢。..）也算是招来大量眼球。

而还是没能耐住寂寞的 Google，联手 HTC，一同推出了至今只为止最重量级的 Android 手机：Nexus One。江湖有云：天下武功，无快不破。搭载了全新的 Android 2.1，1G 的 CPU，史上最清晰的手机屏幕的 Nexus One，快的是一塌糊涂迅雷不掩耳盗铃小叮当，在单机层面，第一次使得 Android 手机与 iPhone 掰手腕的能力（之前与 iPhone 的比较，都需要依靠集团力量，三英战吕布。..）。

在各家厂商努力的同时，Android 本身也没有闲着，版本从 1.1，一步步进阶到了 2.1，SDK 的升级，伴随着大量性能、接口的改进，和功能的丰富，Android 变得越来越快，越来越省电，越来越丰富，越来越多 Google 服务被嵌入@_@。由于 Android SDK 是基于 Java 的，即便虚拟机做的很是 NB，在某些情况下，性能也是无法与原生的 C++代码相提并论，于是，从 1.5 版起，除了 SDK，Android 还拥有了 NDK（Native Develop Kit），它提供了一些 C++的库和编译环境（库是真的很少。..），开发人员可以基于 C++写底层库，用 Java 写上层逻辑，通过混编的方式，兼得鱼和熊掌。

Android Market 的发展也甚为迅猛，虽然和其鼻祖 App Store 相比，应用的规模和盈利能力还显得比较幼齿，但其涨势凶猛，发展趋势远胜于前辈。国内一些比较著名的手机软件，也纷纷拥有了 Android 版本的小弟。

起飞

种种迹象表明，2010，也许就是姗姗来迟的 Android 元年。三星，moto，LG，HTC 等多家手机制造厂商，都为 2010 年将推出的半数以上的手机搭配了 Android。在国内，移动的 OPhone，丑媳妇要正式揭开盖头了，惨烈是惨烈了一点，但聊胜于无，除了水货，2010 毕竟至少多了条购买 Android 手机的道路。

软件开发方面，大家也从抱着双臂冷眼旁观的状态，进入到了一种伺机而动的战略准备阶段。前不久召开的 moto 开发者大会，惊现国内各领域的公司，试水开始，可见一斑。国内各个山寨的 Market 的，也越来越货源充足，下载量稳步上升，升温，就在当下。

而随着 G2 为首的 Android 水机价格火速下调，身边路边地铁边，可以看到越来越多的人，把玩着各式各样的 Android 手机，状况尤为喜人。

所以，2010，如果你有心，就做好准备吧。

- **Android 架构和特征**

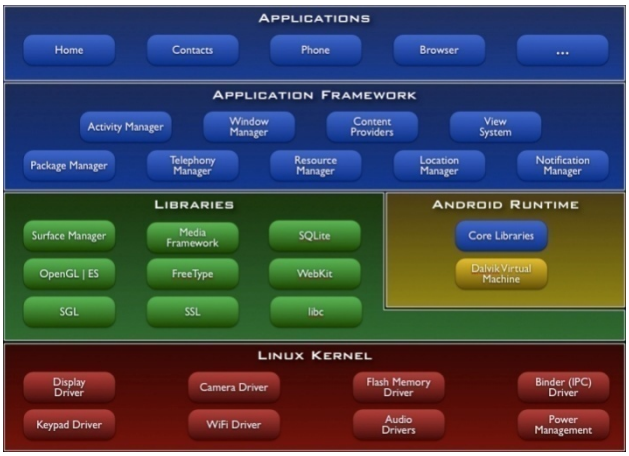
千呼万唤始出来，犹抱琵琶半遮。 -- 《琵琶行》

虽贵为富二代，但 Android 要是没任何可圈点的地方，开不过 70 迈，在玲琅满目的手机平台竞争中，充其量也就做几个俯卧撑打一桶酱油，然后被落的远远的。说到底，出来混，靠的还是技术。

架构

从 SDK 文档中，偷来一幅 Android 平台的架构图，如上。在整个架构最底层红彤彤的部分，是 Linux Kernel 在移动平台的一个移植，它隐藏了硬件、网络等相关的细节，为上层提供了一个相对纯洁的统一接口。除非要做的是 Android 到不同设备的移植工作，否则对于大部分普通开发者而言，基本上是远观而不必亵玩的。Google 一直强调，Android 的底层实现异常 NB，可移植性超强，暂没有功夫研读，实属遗憾。

靠上一层，是一些核心的和扩展的类库，它们都是原生的 C++实现。在这一层，你可以看到很多熟悉的面孔，一如 SQLite、WebKit、OpenGL，开源的力量与贡献由此可见。如果，该层类库需要被上层函数调用，就必须要通过 JNI 的导出相应的接口函数，否则就只能在层次内部自个把玩。



也是在这一层次上，还有为上层 Java 程序服务的运行时。Dalvik 虚拟机，是 Android 的 Java 虚拟机，之所以不采用 J2ME 的虚拟机，一方面是因为 J2ME 的设计是为了低端机器而优化，而 Dalvik 则是为了高端一些的机器进行优化，提供更好的性能。另一方面，从商业角度来看，必须绕开 J2ME 虚拟机，Android 才能彻底解放，想怎么开源就怎么开源，不再需要考虑 License 的问题。

再往上，终于有 Java 出没了。首先是框架层，这里包含所有开发所用的 SDK 类库，另外还有一些未公开接口的类库和实现，它们是整个 Android 平台核心机制的体现。

而在最上面，就是应用层了，系统的一些应用和第三方开发的所有应用都是位于这个层次上，也许要纠结两者的差别，就是系统应用会用一些隐藏的类，而第三方的应用，总是基于 SDK 提供的东西来搞。

一般来说，Android 开发，就是在 SDK 的基础上，吭哧吭哧用 Java 写应用。但自从有了 NDK，一切有了写小变化。NDK 的出现意味着，最上面应用层的内容，可以穿越 Java 部署的框架层，直接和底层暴露出来的，或者自行开发的 C++库直接对话，当然在这些库中需要包含 JNI 的接口。

人说，这就不是 Android 也可以用 C++开发应用么，但其实，这样的说法不够确切，纯 C++应用，是无法被接受的。

因为在 Android 中，大量的核心机制部署在框架层，它们都是用 Java 实现的，比如控件库，Activity 的调度之类的。因此，没了界面，没了调度，还是只用 C++做类库比较合适，否则一切都乱了套了。

特征

基于这样的架构，Android 有很多的设计显得很有意思。纵览整个 SDK 和核心机制的设计，工整漂亮，是 Android 给人的第一感觉。为了说明这一点，找一个反面教材是很有必要的，Symbian 同学毫无悬念的担当这个伟岸的角色。

写 Symbian 程序，感觉就像是在玩一个猜谜游戏。哪怕你是一个 Symbian 老手，当需要用到 Symbian 中某块陌生功能的时候，你可能依然束手无策。你往往需要猜并反复找寻，在这里我需要使用哪一种奇巧淫技呢，是该臆想某些事件，还是应该用一个神秘的 UID 寻找某个特定应用，诸如此类。

而做 Android 应用的时候，就像是做高考模拟试题，题看上去不一样，解答模式摸清楚，就一通百通，一了百了。监听某个系统事件，查一下 SDK 就好；访问某个应用的数据，看看它有没有提供 Content Provider 就可以。所有的一切，都是按套路出牌，只要你了解了套路，再陌生的牌也可以看得懂，出的顺。人说武林高手，都应该是无招胜有招，而一个好的应用框架，也应该做到举重若轻，可触类旁通。

而 Android 框架最文采飞扬的一点，就是引入了 Mash-Up 的思想。所谓 Mash-Up，就是把写应用搞成搭积木，要出效果的时候，东家一块西家一块现场拼起来就好。这里面关键有两点，一个是模块化，另一个就是动态性。所谓模块化，就是一个应用的功能要明确的被封成一个个边界清晰的功能点，每一个功能点都像是一个黑盒，由预先定义的规则描述出其交互方式；而动态性，就是这些独立的模块能够在运行的时候，按照需求描述，连接在一起，共同完成某项更大的功能。在这两点上，Android 都做得非常出色。

站在可 Mash-Up 构造应用这一点去看其他的一些 Android 中的核心功能设计，就显得很有章可循了。比如为什么要把文件私有化，为什么要让进程被托管，等等（当然也可以站在别的角度看出不同的效果，视角不同，视野自然不同。..）。

在 UI 机制方面，Android 也有很不错的表现。它采取 xml 格式的资源文件，描述所有界面相关的内容。资源文件不是什么新东西了，xml 格式也是老调重弹，但可贵的是 Android 做的更为的丰富和彻底，基本把界面相关的逻辑，全部从代码中剥离到了资源文件中，和 Symbian 那四不像的资源文件相比，真是强大了不知多少倍了。

继续阅读文章

——深入剖析 Android 消息机制

- 在 Android 中，线程内部或者线程之间进行信息交互时经常会使用消息，这些基础的东西如果我们熟悉其内部的原理，将会使我们容易、更好地架构系统，避免一些低级的错误。在学习 Android 中消息机制之前，我们先了解与消息有关的几个类：

1.Message

消息对象，顾名思义就是记录消息信息的类。这个类有几个比较重要的字段：

a.arg1 和 arg2：我们可以使用两个字段用来存放我们需要传递的整型值，在 Service 中，我们可以用来存放 Service

的 ID。

b.obj：该字段是 Object 类型，我们可以让该字段传递某个多项到消息的接受者中。

c.what：这个字段可以说是消息的标志，在消息处理中，我们可以根据这个字段的不同的值进行不同的处理，类似于我们在处理 Button 事件时，通过 switch（v.getId（））判断是点击了哪个按钮。

在使用 Message 时，我们可以通过 new Message（）创建一个 Message 实例，但是 Android 更推荐我们通过 Message.obtain（）或者 Handler.obtainMessage（）获取 Message 对象。这并不一定是直接创建一个新的实例，而是先从消息池中看有没有可用的 Message 实例，存在则直接取出并返回这个实例。反之如果消息池中没有可用的 Message 实例，则根据给定的参数 new 一个新 Message 对象。通过分析源码可得知，Android 系统默认情况下在消息池中实例化 10 个 Message 对象。

2.MessageQueue

消息队列，用来存放 Message 对象的数据结构，按照“先进先出”的原则存放消息。存放并非实际意义的保存，而是将 Message 对象以链表的方式串联起来的。MessageQueue 对象不需要我们自己创建，而是有 Looper 对象对其进行管理，一个线程最多只可以拥有一个 MessageQueue。我们可以通过 Looper.myQueue（）获取当前线程中的 MessageQueue。

3.Looper

MessageQueue 的管理者，在一个线程中，如果存在 Looper 对象，则必定存在 MessageQueue 对象，并且只存在一个 Looper 对象和一个 MessageQueue 对象。在 Android 系统中，除了主线程有默认的 Looper 对象，其它线程默认是没有 Looper 对象。如果想让我们新创建的线程拥有 Looper 对象时，我们首先应调用 Looper.prepare（）方法，然后再调用 Looper.loop（）方法。典型的用法如下：

```
view plaincopy to clipboardprint?
class LooperThread extends Thread
{
public Handler mHandler;
public void run（）
{
Looper.prepare（）;
//其它需要处理的操作
Looper.loop（）;
}
}
```

倘若我们的线程中存在 Looper 对象，则我们可以通过 Looper.myLooper（）获取，此外我们还可以通过 Looper.getMainLooper（）获取当前应用系统中主线程的 Looper 对象。在这个地方有一点需要注意，假如 Looper 对象位

于应用程 序主线程中，则 Looper.myLooper（）和 Looper.getMainLooper（）获取的是同一个对象。

- **4.Handler**

消息的处理者。通过 Handler 对象我们可以封装 Message 对象，然后通过 sendMessage（msg）把 Message 对象添加到 MessageQueue 中；当 MessageQueue 循环到该 Message 时，就会调用该 Message 对象对应的 handler 对象的 handleMessage（）方法对其进行处理。由于是在 handleMessage（）方法中处理消息，因此我们应该编写一个类继承自 Handler，然后在 handleMessage（）处理我们需要的操作。

view plaincopy to clipboardprint?

```
# public class MessageService extends Service
#
# {
# private static final String TAG = “MessageService”;
# private static final int KUKA = 0;
# private Looper looper;
# private ServiceHandler handler;
# /**
# * 由于处理消息是在 Handler 的 handleMessage（）方法中，因此我们需要自己编写类
# * 继承自 Handler 类，然后在 handleMessage（）中编写我们所需要的功能代码
# * @author coolszy
# *
# */
# private final class ServiceHandler extends Handler
# {
# public ServiceHandler（Looper looper）
# {
# super（looper）;
# }
#
# @Override
# public void handleMessage（Message msg）
```

```
# {
# // 根据 what 字段判断是哪个消息
# switch （msg.what）
# {
# case KUKA：
# //获取 msg 的 obj 字段。我们可在此编写我们所需要的功能代码
# Log.i（TAG， “The obj field of msg：” + msg.obj）；
# break;
# // other cases
# default：
# break;
# }
# // 如果我们 Service 已完成任务，则停止 Service
# stopSelf（msg.arg1）；
# }
# }
#
# @Override
# public void onCreate（）
# {
# Log.i（TAG， “MessageService-->onCreate（）”）；
# // 默认情况下 Service 是运行在主线程中，而服务一般又十分耗费时间，如果
# // 放在主线程中，将会影响程序与用户的交互，因此把 Service
# // 放在一个单独的线程中执行
# HandlerThread thread = new HandlerThread （ “MessageDemoThread” ，
Process.THREAD_PRIORITY_BACKGROUND）；
# thread.start（）；
# // 获取当前线程中的 looper 对象
```

```
# looper = thread.getLooper（）；
# //创建 Handler 对象，把 looper 传递过来使得 handler、
# //looper 和 messageQueue 三者建立联系
# handler = new ServiceHandler（looper）；
# }
#
# @Override
# public int onStartCommand（Intent intent， int flags， int startId）
# {
# Log.i（TAG， “MessageService-->onStartCommand（）”）；
#
# //从消息池中获取一个 Message 实例
# Message msg = handler.obtainMessage（）；
# // arg1 保存线程的 ID，在 handleMessage（）方法中
# // 我们可以通过 stopSelf（startId）方法，停止服务
# msg.arg1 = startId;
# // msg 的标志
# msg.what = KUKA;
# // 在这里我创建一个 date 对象，赋值给 obj 字段
# // 在实际中我们可以通过 obj 传递我们需要处理的对象
# Date date = new Date（）；
# msg.obj = date;
# // 把 msg 添加到 MessageQueue 中
# handler.sendMessage（msg）；
# return START_STICKY;
# }
#
# @Override
```

```
# public void onDestroy（）
#{
# Log.i（TAG， “MessageService-->onDestroy（）”）；
#}
#
# @Override
# public IBinder onBind（Intent intent）
#{
# return null;
#}
#}
```

注：在测试代码中我们使用了 HandlerThread 类，该类是 Thread 的子类，该类运行时将会创建 looper 对象，使用该类省去了我们自己编写 Thread 子类并且创建 Looper 的麻烦。

——Android 中实现 TCP 和 UDP 传输的方法

- TCP 和 UDP 在网络传输中非常重要，在 Android 开发中同样重要。

  首先我们来看一下什么是 TCP 和 UDP。

  什么是 TCP？

  TCP：Transmission Control Protocol 传输控制协议 TCP 是一种面向连接（连接导向）的、可靠的、基于字节流的运输层（Transport layer）通信协议，由 IETF 的 RFC 793 说明（specified）。在简化的计算机网络 OSI 模型中，它完成第四层传输层所指定的功能。应用层向 TCP 层发送用于网间传输的、用 8 位字节表示的数据流，然后 TCP 把数据流分割成适当长度的报文段（通常受该计算机连接的网络的数据链路层的最大传送单元（MTU）的限制）。之后 TCP 把结果包传给 IP 层，由它来通过网络将包传送给接收端实体的 TCP 层。TCP 为了保证不发生丢包，就给每个字节一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的字节发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据（假设丢失了）将会被重传。TCP 用一个校验和函数来检验数据是否有错误；在发送和接收时都要计算校验和。

  首先，TCP 建立连接之后，通信双方都同时可以进行数据的传输，其次，他是全双工的；在保证可靠性上，采用超时重传和捎带确认机制。

  在流量控制上，采用滑动窗口协议［1］，协议中规定，对于窗口内未经确认的分组需要重传。

在拥塞控制上，采用慢启动算法。

什么是 UDP？

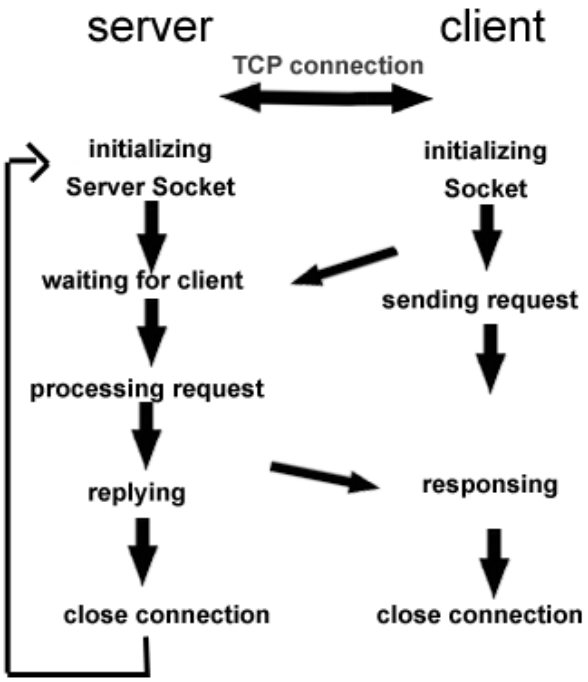
UDP 是 User Datagram Protocol 的简称，中文名是用户数据包协议，是 OSI 参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。它是 IETF RFC 768 是 UDP 的正式规范。在网络中它与 TCP 协议一样用于处理数据包。在 OSI 模型中，在第四层——传输层，处于 IP 协议的上一层。UDP 有不提供数据报分组、组装和不能对数据包的排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。 UDP 用来支持那些需要在计算机之间传输数据的网络应用。包括网络视频会议系统在内的众多的客户/服务器模式的网络应用都需要使用 UDP 协议。UDP 协议从问世至今已经被使用了很多年，虽然其最初的光彩已经被一些类似协议所掩盖，但是即使是在今天，UDP 仍然不失为一项非常实用和可行的网络传输层协议。

与所熟知的 TCP（传输控制协议）协议一样，UDP 协议直接位于 IP（网际协议）协议的顶层。根据 OSI（开放系统互连）参考模型，UDP 和 TCP 都属于传输层协议。

UDP 协议的主要作用是将网络数据流量压缩成数据报的形式。一个典型的数据报就是一个二进制数据的传输单位。每一个数据报的前 8 个字节用来包含报头信息，剩余字节则用来包含具体的传输数据。

TCP 和 UDP 在 android 中的使用和在 Java 里是完全一样的。

首先我们看看 TCP 连接，下图为 TCP 连接的一个示意图：



TCP 传输原理

是不是很好理解，这里就不多说了，直接看代码吧！实践出真知。

TCP 服务器端代码：

```
try {
Boolean endFlag = false;
ServerSocket ss = new ServerSocket（12345）；
while （！ endFlag） {
// 等待客户端连接
Socket s = ss.accept（）；
BufferedReader input = new BufferedReader（newInputStreamReader（s.getInputStream（））；
//注意第二个参数据为 true 将会自动 flush，否则需要需要手动操作 output.flush（）
PrintWriter output = newPrintWriter（s.getOutputStream（），true）；
String message = input.readLine（）；
Log.d（“Tcp Demo”， “message from Client：”+message）；
output.println（“message received！”）；
//output.flush（）；
if（“shutDown”.equals（message））{
endFlag=true;
}
s.close（）；
}
ss.close（）；
} catch （UnknownHostException e） {
e.printStackTrace（）；
} catch （IOException e） {
e.printStackTrace（）；
}
```

TCP 客户端代码：

```
try {
Socket s = new Socket（“localhost”， 12345）；
```

```
// outgoing stream redirect to socket
OutputStream out = s.getOutputStream（）；
// 注意第二个参数据为 true 将会自动 flush，否则需要需要手动操作 out.flush（）
PrintWriter output = new PrintWriter（out， true）；
output.println（“Hello IdeasAndroid！”）；
BufferedReader input = new BufferedReader（newInputStreamReader（s
.getInputStream（））；
// read line（s）
String message = input.readLine（）；
Log.d（“Tcp Demo”， “message From Server：” + message）；
s.close（）；
} catch （UnknownHostException e） {
e.printStackTrace（）；
} catch （IOException e） {
e.printStackTrace（）；
}
```

继续阅读文章

——典型案例开发篇——

——Android 上蓝牙通信功能开发：BluetoothChat 例程分析

- 1. 概述

Bluetooth 是几乎现在每部手机标准配备的功能，多用于耳机 mic 等设备与手机的连接，除此之外，还可以多部手机之间建立 bluetooth 通信，本文就通过 SDK 中带的一个聊天室的例程，来介绍一下 Android 上的 Bluetooth 的开发。

在 Android1.x 的时候，相关 API 非常不完善，还不能简单的使用 Bluetooth 开发，有一个开源项目可以帮助程序员使用、开发蓝牙，支持直接方法 bluetooth 协议栈。在 Android2 以后，框架提供了一些官方 API 来进行蓝牙的通信，但目前的程序也比较不完善。本文主要讨论 Android2 后的 Bluetooth 通信的 API 使用方法。

首先看聊天室的效果图：

2. Bluetooth 通信 API 介绍

2.1. Bluetooth 通信过程

（点击放大）

2.2. Bluetooth API 的主要方法

BluetoothAdapter 类

BluetoothAdapter.getDefaultAdapter（） ：得到本地默认的 BluetoothAdapter ，若返回为 null 则表示本地不支持蓝牙；

- isDiscovering（）：返回设备是否正在发现周围蓝牙设备；

  cancelDiscovery（）：取消正在发现远程蓝牙设备的过程；

  startDiscovery（）：开始发现过程；

  getScanMode（）：得到本地蓝牙设备的 Scan Mode ；

  getBondedDevices（）：得到已配对的设备；

  isEnabled（）：蓝牙功能是否启用。

  当发现蓝牙功能未启用时，如下调用设置启用蓝牙：

```
if (! mBluetoothAdapter .isEnabled()) {
    Intent enableIntent = new Intent(BluetoothAdapter. ACTION_REQUEST_ENABLE );
    startActivityForResult(enableIntent, REQUEST_ENABLE_BT );
}
```

（点击放大）

如果发现当前设备没有打开对外可见模式，则传递 Intent 来调用打开可发现模式，代码如下：

```
Intent discoverableIntent = new Intent(BluetoothAdapter. ACTION_REQUEST_DISCOVERABLE );
discoverableIntent.putExtra(BluetoothAdapter. EXTRA_DISCOVERABLE_DURATION , 300);
startActivity(discoverableIntent);
```

（点击放大）

BluetoothDevice 类，此为对应的远程蓝牙 Device

createRfcommSocketToServiceRecord（）：创建该 Device 的 socket 。

BluetoothSocket 类

connect（）：请求连接蓝牙。

getInputStream（）：得到输入流，用于接收远程方信息。

getOutputStream（）：得到输出流，发送给远程方的信息。

close（）：关闭蓝牙连接。

InputStream 类：

read（byte［］）：以阻塞方式读取输入流。

OutputStream 类：

write（byte［］）：将信息写入该输出流，发送给远程。 继续阅读文章

——Android 开发技巧：软硬件的巧妙整合

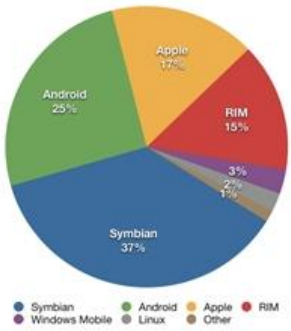
- 随着科技的快速演进，现代人对移动通信、无线上网与多媒体娱乐的需求更甚以往，所谓的智能手机（Smart Phone）便成了炙手可热的个人消费电子产品之一，从 Apple 不断推出 iPhone 企图颠覆消费者对手机的想象、RIM 推出主打商务功能的黑莓机、Google 的 Android 系统让众家手机厂商争食大饼，到微软屡败屡战的从 WinMo 一路开发到 WP7，智能手机的这块战场可说是打的如火如荼。然而在这些众家竞争者中，Android 可说是目前行情看俏的一套操作系统，以国际市调研究机构 Gartner 最新出炉 2010 年第三季的调查为例，采用 Android 操作系统的智能手机在过去一年以来成长幅度最高，光是市占率便是前一年同期的七倍之多，销售量更是达到 14 倍的成长，同时也一举从市占率排名的第六名窜升到第二名。而在今年一月份甫落幕的国际消费性电子展（CES），也处处可见各式各样采用 Android 操作系统的产品。

＊Gartner 2010 Q3 Worldwide Smartphone Sales

Worldwide Smartphone Sales to End Users by Operating System in 3Q10 (Thousands of Units)

Company	3Q10 Units	3Q10 Market Share (%)	3Q09 Units	3Q09 Market Share (%)
Symbian	29,480.1	36.6	18,314.8	44.6
Android	20,500.0	25.5	1,424.5	3.5
iOS	13,484.4	16.7	7,040.4	17.1
Research In Motion	11,908.3	14.8	8,522.7	20.7
Microsoft Windows Mobile	2,247.9	2.8	3,259.9	7.9
Linux	1,697.1	2.1	1,918.5	4.7
Other OS	1,214.8	1.5	612.5	1.5
Total	**80,532.6**	**100.0**	**41,093.3**	**100.0**

Source: Gartner (November 2010)



Android 在过去一直扮演后起之秀的角色，切入智能手机的速度似乎慢了苹果的 iOS 一步，但与 Apple 相同的是，它也成功的将其应用从手机移植到了平板电脑（Tablet PC）上。Android 开放源代码（Open Source）的特性，能轻易地提高厂商对自家产品的接受度，更不用提背后 Google 的强力撑腰能带来多大的经济效益。目前可见包括手机厂商 HTC、摩托罗拉（Motorola）、三星（SAMSUNG），以及电脑大厂惠普（HP）与戴尔（Dell）等皆投向 Android 的怀抱，Android 被广泛应用可说是势在必行。

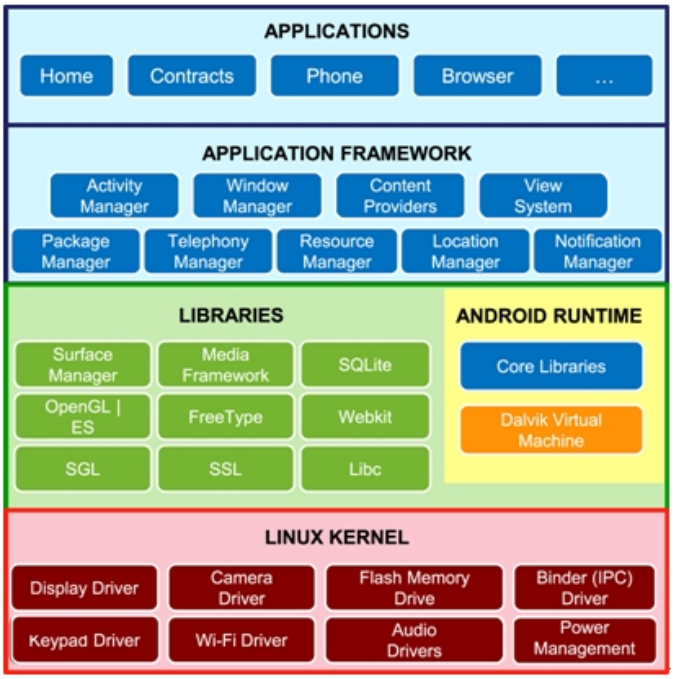
尽管 Android 系统的普及看似指日可待，但在实际的产品应用上，也有其可能产生的问题风险。Android 作为一个开放式的操作系统，是 Google 提供厂商的操作系统参考架构（reference design），厂商能有充足的发挥空间，以 Android

为基础向上开发设计自家产品，但也因为这样的开放性与自由性，让厂商在软硬件结合的这个环节必须下更大的功夫，像是如何挑选合适的硬件包括基频处理器、通信芯片、触控感应芯片、天线与存储器模组等，以及如何调整出最适当的软件设定等，更重要的是如何将软硬件整合，开发出差异化的产品。这中间所有的细节都会对产品最终样貌产生莫大的影响，像是其功能的完整度、使用接口的设计、效能表现（例如触控滑动画面、开启程序所需时间）、品质可靠度、甚至是后续的固件升级动作等等。在此百佳泰便试图以专业中立的测试实验室角度，来点出厂商应用 Android 于手机、平板电脑或其他设备时应注意的开发重点，以希冀作为一个有效的参考资讯。

- **解构 Android 基本技术架构**

首先我们先来看到 Android 的基本技术架构，Android 是以 Linux 为核心，并采用软件堆迭（software stack）的架构延伸发展的一套软件平台与操作系统。根据下图可以看出，其基本架构分为五层：

＊Android Structure by Google



·Linux 核心（Linux Kernel）：以 Linux 开发提供最底层的核心系统服务，包括安全性（Security）、存储器管理（Memory Management）、进程管理（Process Management）、网路堆迭（Network Stack）与驱动程序模型（Driver Model）。

·Android 执行环境（Android Runtime）：透过 Core Libraries（核心函式库）以及暂存器型态的 Dalvik Virtual Machine（Dalvik 虚拟机器）来执行程序。

·系统函式库（Library）：使用 C/C++函式库的系统组件以供呼叫使用，开发者可透过上层的应用程序框架来运用这些功能，这也是主要 Android 设备的效能关键。

·应用程序框架（Application Framework）：被设计来简化组件的再运用，开发者能完整存取使用与核心应用程序（Core Application）相同的 API，应用程序可以发布功能并为其它应用程序所使用（需受限于其安全性限制），开发者也可运用同样的机制来新增与置换组件。

·应用程序（Application）：所有 Android 应用程序皆是以 Java 程序语言编写，原始就会包含像是 Email、简讯、日历、地图、浏览器、联络人等其它应用程序，让用户一开始就拥有这些基本功能，开发者也可在此客制其使用接口。

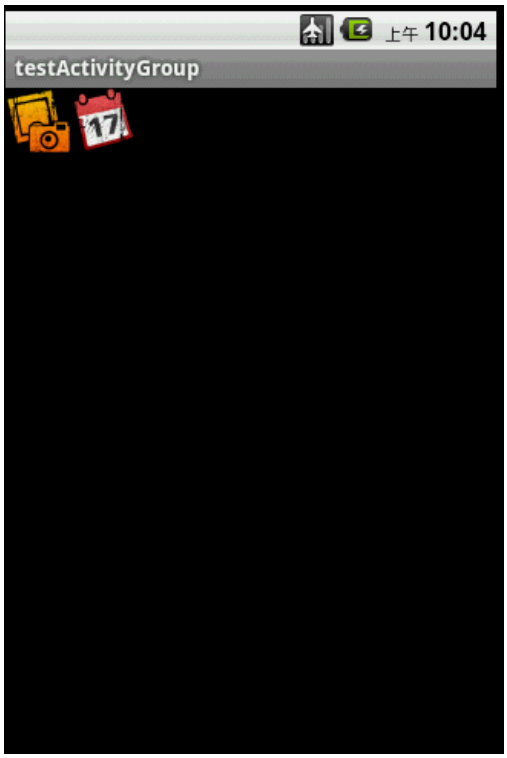
厂商越想要设计出与原始设定不同且增强效能的产品，便越需要对这五层架构进行修改。譬如像是多任务处理能力（multi-tasking），便可能需要修改包括 Linux 核心与应用程序框架的设计；而应用程序的开发者更可能需要针对应用程序与框架进行调整。由此可见，对 Android 设备而言，任何一个功能的置入或是对硬件设定的细微更动，都需要对 Android 系统进行从下到上的调整以达到最优化的效能，而这正是最为困难与需要验证的一环。

继续阅读文章

——Android 之 ActivityGroup 实现 Tab 分页标签

- 很多客户端软件和浏览器软件都喜欢用 Tab 分页标签来管理内容，除了可以用 TabHost 控件，还可以用 ImageButton + ActivityGroup 实现 Tab 分页标签。使用 ImageButton + ActivityGroup 实现 Tab 分页标签，主要是把一个 Sub Activity（子 Activity）的 Window 作为 View 添加到 ActivityGroup 所指定的容器中，本文使用 LinearLayout 作为容器装载 Sub Activity。

接下来贴出本例运行的效果图：



以下是切换时 Sub Activity 的生存周期的状态变化：

tag	Message
subActivity1	onCreate
subActivity1	onStart
subActivity1	onResume
subActivity1	onPause
subActivity1	onStop
subActivity1	onDestroy
subActivity2	onCreate
subActivity2	onStart
subActivity2	onResume
subActivity2	onPause
subActivity2	onStop
subActivity2	onDestroy
subActivity1	onCreate
subActivity1	onStart
subActivity1	onResume

从 subActivity1 切换到 subActivity2 的时候，会彻底释放 subActivity1 的资源。

主 Activity 的 main.xml 的源码如下：

view plaincopy to clipboardprint?

《? 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/LinearLayout01”

android:layout_height=“wrap_content” android:layout_width=“fill_parent”》

《ImageButton android:layout_width=“wrap_content”

android:layout_height=“wrap_content” android:id=“@+id/ibtnTab1”

android:background=“@drawable/png1298”》《/ImageButton》

《ImageButton android:layout_width=“wrap_content”

android:layout_height=“wrap_content” android:id=“@+id/ibtnTab2”

android:background=“@drawable/png1292”》《/ImageButton》

《/LinearLayout》

《LinearLayout android:id=“@+id/LinearLayout02”

android:layout_width=“fill_parent” android:layout_height=“fill_parent”》《/LinearLayout》

《/LinearLayout》

《？ 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/LinearLayout01”

android:layout_height=“wrap_content” android:layout_width=“fill_parent”》

《ImageButton android:layout_width=“wrap_content”

android:layout_height=“wrap_content” android:id=“@+id/ibtnTab1”

android:background=“@drawable/png1298”》《/ImageButton》

《ImageButton android:layout_width=“wrap_content”

android:layout_height=“wrap_content” android:id=“@+id/ibtnTab2”

android:background=“@drawable/png1292”》《/ImageButton》

《/LinearLayout》

《LinearLayout android:id=“@+id/LinearLayout02”

android:layout_width=“fill_parent” android:layout_height=“fill_parent”》《/LinearLayout》

《/LinearLayout》

Sub Activity 的 XML 源码（listview.xml）如下：

view plaincopy to clipboardprint？

《？ xml version=“1.0” encoding=“utf-8”？ 》

《LinearLayout android:id=“@+id/LinearLayout01”

xmlns:android=“http://schemas.android.com/apk/res/android”

android:layout_width=“fill_parent” android:layout_height=“fill_parent”》

《ListView android:id=“@+id/MyListView” android:layout_width=“fill_parent”

android:layout_height=“fill_parent”》

《/ListView》

《/LinearLayout》

《？ xml version=“1.0” encoding=“utf-8”？ 》

《LinearLayout android:id=“@+id/LinearLayout01”

xmlns:android=“http://schemas.android.com/apk/res/android”

android:layout_width=“fill_parent” android:layout_height=“fill_parent”》

《ListView android:id=“@+id/MyListView” android:layout_width=“fill_parent”

android:layout_height=“fill_parent”》

《/ListView》

《/LinearLayout》

- testActivityGroup.java 源码如下：

view plaincopy to clipboardprint?

```
package com.testActivityGroup;
import android.app.ActivityGroup;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.widget.ImageButton;
import android.widget.LinearLayout;
import android.widget.ListView;
public class testActivityGroup extends ActivityGroup {
/** Called when the activity is first created. */
LinearLayout container;//装载 sub Activity 的容器
ImageButton ibtnTab1，ibtnTab2;
@Override
public void onCreate（Bundle savedInstanceState） {
super.onCreate（savedInstanceState）;
setContentView（R.layout.main）;
container = （LinearLayout） findViewById（R.id.LinearLayout02）;
ibtnTab1=（ImageButton）this.findViewById（R.id.ibtnTab1）;
ibtnTab1.setOnClickListener（new ClickEvent（））;
```

```
ibtnTab2=（ImageButton）this.findViewById（R.id.ibtnTab2）；
ibtnTab2.setOnClickListener（new ClickEvent（））；
}
class ClickEvent implements View.OnClickListener{
@Override
public void onClick（View v） {
container.removeAllViews（）；
Intent intent=new Intent（testActivityGroup.this， subActivity.class）；
intent.addFlags（Intent.FLAG_ACTIVITY_CLEAR_TOP）；
String［］ str=new String［12］；
if（v==ibtnTab1）
{
for（int i=0;i《str.length;i++）
str［i］=“单选”+String.valueOf（i）；
intent.putExtra（“Name”， “subActivity1”）；
intent.putExtra（“Strings”， str）；
intent.putExtra（“ChoiceMode”， ListView.CHOICE_MODE_SINGLE）;//通过参数设置列表式样
}
else if（v==ibtnTab2）
{
for（int i=0;i《str.length;i++）
str［i］=“复选”+String.valueOf（i）；
intent.putExtra（“Name”， “subActivity2”）；
intent.putExtra（“Strings”， str）；
intent.putExtra（“ChoiceMode”， ListView.CHOICE_MODE_MULTIPLE）;//通过参数设置列表式样
}
Window subActivity=getLocalActivityManager（）.startActivity（“subActivity”，intent）；
container.addView（subActivity.getDecorView（））；
```

```
}
}
}
package com.testActivityGroup;
import android.app.ActivityGroup;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.widget.ImageButton;
import android.widget.LinearLayout;
import android.widget.ListView;
public class testActivityGroup extends ActivityGroup {
/** Called when the activity is first created. */
LinearLayout container;//装载 sub Activity 的容器
ImageButton ibtnTab1，ibtnTab2;
@Override
public void onCreate（Bundle savedInstanceState） {
super.onCreate（savedInstanceState）;
setContentView（R.layout.main）;
container = （LinearLayout） findViewById（R.id.LinearLayout02）;
ibtnTab1=（ImageButton）this.findViewById（R.id.ibtnTab1）;
ibtnTab1.setOnClickListener（new ClickEvent（））;
ibtnTab2=（ImageButton）this.findViewById（R.id.ibtnTab2）;
ibtnTab2.setOnClickListener（new ClickEvent（））;
}
class ClickEvent implements View.OnClickListener{
@Override
```

```
public void onClick（View v） {
container.removeAllViews（）；
Intent intent=new Intent（testActivityGroup.this， subActivity.class）；
intent.addFlags（Intent.FLAG_ACTIVITY_CLEAR_TOP）；
String［］ str=new String［12］；
if（v==ibtnTab1）
{
for（int i=0;i《str.length;i++）
str［i］=“单选”+String.valueOf（i）；
intent.putExtra（“Name”， “subActivity1”）；
intent.putExtra（“Strings”， str）；
intent.putExtra（“ChoiceMode”， ListView.CHOICE_MODE_SINGLE）;//通过参数设置列表式样
}
else if（v==ibtnTab2）
{
for（int i=0;i《str.length;i++）
str［i］=“复选”+String.valueOf（i）；
intent.putExtra（“Name”， “subActivity2”）；
intent.putExtra（“Strings”， str）；
intent.putExtra（“ChoiceMode”， ListView.CHOICE_MODE_MULTIPLE）;//通过参数设置列表式样
}
Window subActivity=getLocalActivityManager（）.startActivity（“subActivity”，intent）；
container.addView（subActivity.getDecorView（））；
}
}
}
```

继续阅读文章

——Android 深入浅出之 Surface 探讨

- 一 目的

本节的目的就是为了讲清楚 Android 中的 Surface 系统，大家耳熟能详的 SurfaceFlinger 到底是个什么东西，它的工作流程又是怎样的。当然，鉴于 SurfaceFlinger 的复杂性，我们依然将采用情景分析的办法，找到合适的切入点。

一个 Activity 是怎么在屏幕上显示出来的呢？我将首先把这个说清楚。

接着我们把其中的关键调用抽象在 Native 层，以这些函数调用为切入点来研究 SurfaceFlinger。好了，开始我们的征途吧。

二 Activity 是如何显示的

最初的想法就是，Activity 获得一块显存，然后在上面绘图，最后交给设备去显示。这个道理是没错，但是 Android 的 SurfaceFlinger 是在 System Server 进程中创建的，Activity 一般另有线程，这之间是如何。..如何挂上关系的呢？我可以先提前告诉大家，这个过程还比较复杂。呵呵。

好吧，我们从 Activity 最初的启动开始。代码在

framework/base/core/java/android/app/ActivityThread.java 中，这里有个函数叫 handleLaunchActivity

［----》ActivityThread：： handleLaunchActivity（）］

private final void handleLaunchActivity（ActivityRecord r， Intent customIntent） {

Activity a = performLaunchActivity（r， customIntent）；

if （a ！= null） {

r.createdConfig = new Configuration（mConfiguration）；

Bundle oldState = r.state;

handleResumeActivity（r.token， false， r.isForward）；

----》调用 handleResumeActivity

}

handleLaunchActivity 中会调用 handleResumeActivity。

［---》ActivityThread：： handleResumeActivity］

final void handleResumeActivity（IBinder token， boolean clearHide， boolean isForward） {

boolean willBeVisible = ！a.mStartedActivity;

```
if （r.window == null && ！a.mFinished && willBeVisible） {
r.window = r.activity.getWindow（）;
View decor = r.window.getDecorView（）;
decor.setVisibility（View.INVISIBLE）;
ViewManager wm = a.getWindowManager（）;
WindowManager.LayoutParams l = r.window.getAttributes（）;
a.mDecor = decor;
l.type = WindowManager.LayoutParams.TYPE_BASE_APPLICATION;
if （a.mVisibleFromClient） {
a.mWindowAdded = true;
wm.addView（decor， l）; //这个很关键。
}
```

- 上面 addView 那几行非常关键，它关系到咱们在 Activity 中 setContentView 后，整个 Window 到底都包含了些什么。我先告诉大家。所有你创建的 View 之上，还有一个 DecorView，这是一个 FrameLayout，另外还有一个 PhoneWindow。上面这些东西的代码在

  framework/Policies/Base/Phone/com/android/Internal/policy/impl。这些隐藏的 View 的创建都是由你在 Acitivty 的 onCreate 中调用 setContentView 导致的。

```
［----》PhoneWindow：： addContentView］
public void addContentView（View view， ViewGroup.LayoutParams params） {
if （mContentParent == null） { //刚创建的时候 mContentParent 为空
installDecor（）;
}
mContentParent.addView（view， params）;
final Callback cb = getCallback（）;
if （cb ！= null） {
cb.onContentChanged（）;
}
}
```

installDecor 将创建 mDecor 和 mContentParent。mDecor 是 DecorView 类型，

mContentParent 是 ViewGroup 类型

private void installDecor（） {

if （mDecor == null） {

mDecor = generateDecor（）;

mDecor.setDescendantFocusability（ViewGroup.FOCUS_AFTER_DESCENDANTS）;

mDecor.setIsRootNamespace（true）;

}

if （mContentParent == null） {

mContentParent = generateLayout（mDecor）;

那么，ViewManager wm = a.getWindowManager（）又返回什么呢？

PhoneWindow 从 Window 中派生，Acitivity 创建的时候会调用它的 setWindowManager。而这个函数由 Window 类实现。

代码在 framework/base/core/java/android/view/Window.java 中

public void setWindowManager（WindowManager wm，IBinder appToken， String appName） {

mAppToken = appToken;

mAppName = appName;

if （wm == null） {

wm = WindowManagerImpl.getDefault（）;

}

mWindowManager = new LocalWindowManager（wm）;

}

你看见没，分析 JAVA 代码这个东西真的很复杂。mWindowManager 的实现是 LocalWindowManager，但由通过 Bridge 模式把功能交给 WindowManagerImpl 去实现了。

真的很复杂！

好了，罗里罗嗦的，我们回到 wm.addView（decor， l）。最终会由 WindowManagerImpl 来完成

addView 操作，我们直接看它的实现好了。

代码在 framework/base/core/java/android/view/WindowManagerImpl.java

［----》addView］

```
private void addView（View view， ViewGroup.LayoutParams params， boolean nest）
{
ViewRoot root; //ViewRoot，我们的主人公终于登场！
synchronized （this） {
root = new ViewRoot（view.getContext（））；
root.mAddNesting = 1;
view.setLayoutParams（wparams）；
if （mViews == null） {
index = 1;
mViews = new View［1］；
mRoots = new ViewRoot［1］；
mParams = new WindowManager.LayoutParams［1］；
} else {
}
index--;
mViews［index］ = view;
mRoots［index］ = root;
mParams［index］ = wparams;
}
root.setView（view， wparams， panelParentView）；
}
```

- ViewRoot 是整个显示系统中最为关键的东西，看起来这个东西好像和 View 有那么点关系，其实它根本和 View 等 UI 关系不大，它不过是一个 Handler 罢了，唯一有关系的就是它其中有一个变量为 Surface 类型。我们看看它的定义。ViewRoot 代码在

  framework/base/core/java/android/view/ViewRoot.java 中

  public final class ViewRoot extends Handler implements ViewParent，

  View.AttachInfo.Callbacks

  {

private final Surface mSurface = new Surface（）；

}

它竟然从 handler 派生，而 ViewParent 不过定义了一些接口函数罢了。

看到 Surface 直觉上感到它和 SurfaceFlinger 有点关系。要不先去看看？

Surface 代码在 framework/base/core/java/android/view/Surface.java 中，我们调用的是无参构造函数。

public Surface（） {

mCanvas = new CompatibleCanvas（）; //就是创建一个 Canvas！

}

如果你有兴趣的话，看看 Surface 其他构造函数，最终都会调用 native 的实现，而这些 native 的实现将和 SurfaceFlinger 建立关系，但我们这里 ViewRoot 中的 mSurface 显然还没有到这一步。那它到底是怎么和 SurfaceFlinger 搞上的呢？这一切待会就会水落石出的。

另外，为什么 ViewRoot 是主人公呢？因为 ViewRoot 建立了客户端和 SystemServer 的关系。我们看看它的构造函数。

public ViewRoot（Context context） {

super（）；

。..。

getWindowSession（context.getMainLooper（））；

}

getWindowsession 将建立和 WindowManagerService 的关系。

ublic static IWindowSession getWindowSession（Looper mainLooper） {

synchronized （mStaticInit） {

if （！mInitialized） {

try {

//sWindowSession 是通过 Binder 机制创建的。终于让我们看到点希望了

InputMethodManager imm = InputMethodManager.getInstance（mainLooper）；

sWindowSession = IWindowManager.Stub.asInterface（

ServiceManager.getService（“window”））

.openSession（imm.getClient（）， imm.getInputContext（））；

mInitialized = true;

```
} catch （RemoteException e） {
}
}
return sWindowSession;
}
}
```

上面跨 Binder 的进程调用另一端是 WindowManagerService，代码在 framework/base/services/java/com/android/server/WindowManagerService.java 中。我们先不说这个。回过头来看看 ViewRoot 接下来的调用。

［--》ViewRoot:：setView（）]，这个函数很复杂，我们看其中关键几句。

```
public void setView（View view， WindowManager.LayoutParams attrs，
View panelParentView） {
synchronized （this） {
requestLayout（）;
try {
res = sWindowSession.add（mWindow， mWindowAttributes，
getHostVisibility（）， mAttachInfo.mContentInsets）;
}
}
```

requestLayout 实现很简单，就是往 handler 中发送了一个消息。

```
public void requestLayout（） {
checkThread（）;
mLayoutRequested = true;
scheduleTraversals（）; //发送 DO_TRAVERSAL 消息
}
public void scheduleTraversals（） {
if （！mTraversalScheduled） {
mTraversalScheduled = true;
```

sendEmptyMessage（DO_TRAVERSAL）；

}

}

- 我们看看跨进程的那个调用。sWindowSession.add。它的最终实现在 WindowManagerService 中。

［---》WindowSession：：add（）］

public int add（IWindow window， WindowManager.LayoutParams attrs，

int viewVisibility， Rect outContentInsets） {

return addWindow（this， window， attrs， viewVisibility， outContentInsets）；

}

WindowSession 是个内部类，会调用外部类的 addWindow

这个函数巨复杂无比，但是我们的核心目标是找到创建显示相关的部分。所以，最后精简的话就简单了。

［---》WindowManagerService：： addWindow］

public int addWindow（Session session， IWindow client，

WindowManager.LayoutParams attrs， int viewVisibility，

Rect outContentInsets） {

//创建一个 WindowState，这个又是什么玩意儿呢？

win = new WindowState（session， client， token，

attachedWindow， attrs， viewVisibility）；

win.attach（）；

return res;

}

WindowState 类中有一个和 Surface 相关的成员变量，叫 SurfaceSession。它会在

attach 函数中被创建。SurfaceSession 嘛，就和 SurfaceFlinger 有关系了。我们待会看。

好，我们知道 ViewRoot 创建及调用 add 后，我们客户端的 View 系统就和 WindowManagerService 建立了牢不可破的关系。

另外，我们知道 ViewRoot 是一个 handler，而且刚才我们调用了 requestLayout，所以接下来消息循环下一个将调用的就是 ViewRoot 的 handleMessage。

public void handleMessage（Message msg） {

switch （msg.what） {

case DO_TRAVERSAL：

performTraversals（）；

performTraversals 更加复杂无比，经过我仔细挑选，目标锁定为下面几个函数。当然，后面我们还会回到 performTraversals，不过我们现在更感兴趣的是 Surface 是如何创建的。

private void performTraversals（） {

// cache mView since it is used so much below.。.

final View host = mView;

boolean initialized = false;

boolean contentInsetsChanged = false;

boolean visibleInsetsChanged;

try {

//ViewRoot 也有一个 Surface 成员变量，叫 mSurface，这个就是代表 SurfaceFlinger 的客户端

//ViewRoot 在这个 Surface 上作画，最后将由 SurfaceFlinger 来合成显示。刚才说了 mSurface 还没有什么内容。

relayoutResult = relayoutWindow（params， viewVisibility， insetsPending）；

［----》ViewRoot：： relayoutWindow（）］

private int relayoutWindow（WindowManager.LayoutParams params， int viewVisibility，

boolean insetsPending） throws RemoteException {

//relayOut 是跨进程调用，mSurface 做为参数传进去了，看来离真相越来越近了呀！

int relayoutResult = sWindowSession.relayout（

mWindow， params，

（int） （mView.mMeasuredWidth * appScale + 0.5f），

（int） （mView.mMeasuredHeight * appScale + 0.5f），

viewVisibility， insetsPending， mWinFrame，

mPendingContentInsets， mPendingVisibleInsets，

mPendingConfiguration， mSurface）; mSurface 做为参数传进去了。

}

我们赶紧转到 WindowManagerService 去看看吧。、

```
public int relayoutWindow（Session session， IWindow client，
WindowManager.LayoutParams attrs， int requestedWidth，
int requestedHeight， int viewVisibility， boolean insetsPending，
Rect outFrame， Rect outContentInsets， Rect outVisibleInsets，
Configuration outConfig， Surface outSurface）{
。..。.
try {
//看到这里，我内心一阵狂喜，有戏，太有戏了！
//其中 win 是我们最初创建的 WindowState！
Surface surface = win.createSurfaceLocked（）；
if （surface ！= null） {
//先创建一个本地 surface，然后把传入的参数 outSurface copyFrom 一下
outSurface.copyFrom（surface）；
win.mReportDestroySurface = false;
win.mSurfacePendingDestroy = false;
} else {
outSurface.release（）；
}
}
}
［---》WindowState：：createSurfaceLocked］
Surface createSurfaceLocked（） {
try {
mSurface = new Surface（
mSession.mSurfaceSession， mSession.mPid，
mAttrs.getTitle（）.toString（），
0， w， h， mAttrs.format， flags）；
}
```

Surface.openTransaction（）；

这里使用了 Surface 的另外一个构造函数。

public Surface（SurfaceSession s，

int pid， String name， int display， int w， int h， int format， int flags）

throws OutOfResourcesException {

mCanvas = new CompatibleCanvas（）；

init（s，pid，name，display，w，h，format，flags）; ----》调用了 native 的 init 函数。

mName = name;

}

到这里，不进入 JNI 是不可能说清楚了。不过我们要先回顾下之前的关键步骤。 继续阅读文章

——Android 特色开发之传感器和语音识别

- **Android 特色开发**

Android 是一个面向应用程序开发的丰富平台，它拥有许多具有吸引力的用户界面元素、数据管理和网络应用等优秀的功能。Android 还提供了很多颇具特色的接口。本文我们将分别介绍这些吸引开发者眼球的特色开发，主要包括：传感器系统(Sensor)、语音识别技术(RecognizerIntent)、Google Map 和用来开发桌面的插件(Widget)。通过本文的学习，读者将对 Android 有一个更深入的了解，可以开发出一些有特色、有创意的应用程序。

一 传感器

据调查，2008 年全球传感器销售额为 506 亿美元，预计到 2010 年全球传感器销售额可达 600 亿美元以上。调查显示，东欧、亚太区和加拿大成为传感器市场增长最快的地区，而美国、德国、日本依旧是传感器市场分布最大的地区。就世界范围而言，传感器市场增长最快的领域依旧是汽车，占第二位的是过程控制，当然现在也被广泛应用于通信。那么，传感器的定义是什么呢?有哪些种类的传感器呢?Android 中提供了哪些传感器呢?

1.传感器的定义

传感器是一种物理装置或生物器官，能够探测、感受外界的信号、物理条件(如光、热、湿度)或化学组成(如烟雾)，并将探知的信息传递给其他装置或器官。国家标准 GB7665—87 对传感器的定义是：“能感受规定的被测量并按照一定的规律转换成可用信号的器件或装置，通常由敏感元件和转换元件组成”。传感器是一种检测装置，能感受被测量的信息，并能将检测的感受到的信息，按一定规律变换成为电信号或其他所需形式的信息输出，以满足信息的传输、处理、存储、显示、记录和控制等要求。它是实现自动检测和自动控制的首要环节。

2.传感器的种类

可以从不同的角度对传感器进行分类：转换原理(传感器工作的基本物理或化学效应);用途;输出信号类型以及制作材料和工艺等。

根据工作原理，传感器可分为物理传感器和化学传感器两大类。

物理传感器应用的是物理效应，诸如压电效应，磁致伸缩现象，离化、极化、热电、光电、磁电等效应。被测信号量的微小变化都将转换成电信号。

化学传感器包括那些以化学吸附、电化学反应等现象为因果关系的传感器，被测信号量的微小变化也将转换成电信号。

大多数传感器是以物理原理为基础运作的。化学传感器的技术问题较多，例如可靠性问题、规模生产的可能性、价格问题等，解决了这些问题，化学传感器的应用将会有巨大增长。而有些传感器既不能划分为物理类，也不能划分为化学类。

3.Android 中传感器的种类

Google Android 操作系统中内置了很多传感器，比如 G1 自带了一个非常实用的加速感应器(微型陀螺仪)，有了它，G1 手机就支持重力感应、方向判断等功能，在部分游戏或软件中可以自动识别屏幕的横屏、竖屏方向来改变屏幕显示布局。下面是 Android 中支持的几种传感器：

Sensor.TYPE_ACCELEROMETER：加速度传感器。

Sensor.TYPE_GYROSCOPE：陀螺仪传感器。

Sensor.TYPE_LIGHT：亮度传感器。

Sensor.TYPE_MAGNETIC_FIELD：地磁传感器。

Sensor.TYPE_ORIENTATION：方向传感器。

Sensor.TYPE_PRESSURE：压力传感器。

Sensor.TYPE_PROXIMITY：近程传感器。

Sensor.TYPE_TEMPERATURE：温度传感器。

下面我们通过一个例子来分析 Android 中传感器的使用(具体实现参见本书所附代码：第 9 章\ Examples_09_01)，这里分析的是方向传感器(TYPE_ORIENTATION)。

4.Android 中传感器的功能

要在 Android 中使用传感器，首先需要了解 SensorManager 和 SensorEventListener。顾名思义，SensorManager 就是所有传感器的一个综合管理类，包括了传感器的种类、采样率、精准度等。我们可以通过 getSystemService 方法来取得一个 SensorManager 对象。代码如下：

```
SensorManager mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
```

取得 SensorManager 对象之后，可以通过 getSensorList 方法来获得我们需要的传感器类型，保存到一个传感器列表中。通过如下代码可以得到一个方向传感器：

```
List sensors = mSensorManager.getSensorList(Sensor.TYPE_ORIENTATION);
```

要与传感器交互，应用程序必须注册以侦听与一个或多个传感器相关的活动。Android 中提供了 registerListener 来注册一个传感器，并提供了 unregisterListener 来卸载一个传感器。registerListener 方法包括 3 个参数：第 1 个参数，接收信号的 Listener 实例;第 2 个参数，想接收的传感器类型的列表(即上一步创建的 List 对象);第 3 个参数，接收频度。调用之后返回一个布尔值，true 表示成功，false 表示失败。当然，之后不再使用时，我们还需要卸载。代码如下：

```
//注册传感器
Boolean mRegisteredSensor = mSensorManager.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_FASTEST);
//卸载传感器
mSensorManager.unregisterListener(this);
```

其中，SensorEventListener 是使用传感器的核心部分，包括以下两个方法必须实现：

onSensorChanged (SensorEvent event) 方法在传感器值更改时调用。该方法只由受此应用程序监视的传感器调用。该方法的参数包括一个 SensorEvent 对象，该对象主要包括一组浮点数，表示传感器获得的方向、加速度等信息。例如，以下代码可以取得其值：

```
float x = event.values[SensorManager.DATA_X];
float y = event.values[SensorManager.DATA_Y];
float z = event.values[SensorManager.DATA_Z];
```

onAccuracyChanged (Sensor sensor,int accuracy) 方法在传感器的精准度发生改变时调用。其参数包括两个整数：一个表示传感器，另一个表示该传感器新的准确值。

- 具体实现如代码清单 1 所示。

代码清单 1 \Examples_09_01\src\com\yarin\android\Examples_09_01\Activity01.java

```
public class Activity01 extends Activity implements SensorEventListener
{
private boolean mRegisteredSensor;
//定义 SensorManager
private SensorManager mSensorManager;
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
mRegisteredSensor = false;
```

```
//取得 SensorManager 实例
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
}
protected void onResume()
{
super.onResume();
//接收 SensorManager 的一个列表(Listener)
//这里我们指定类型为 TYPE_ORIENTATION(方向传感器)
List sensors = mSensorManager.getSensorList
(Sensor.TYPE_ORIENTATION);
if (sensors.size() > 0)
{
Sensor sensor = sensors.get(0);
//注册 SensorManager
//this->接收 sensor 的实例
//接收传感器类型的列表
//接收的频率
mRegisteredSensor = mSensorManager.registerListener(this,
sensor, SensorManager.SENSOR_DELAY_FASTEST);
}
}
protected void onPause()
{
if (mRegisteredSensor)
{
//如果调用了 registerListener
//这里我们需要 unregisterListener 来卸载/取消注册
mSensorManager.unregisterListener(this);
```

```
mRegisteredSensor = false;
}
super.onPause();
}
//当精准度发生改变时
//sensor->传感器
//accuracy->精准度
public void onAccuracyChanged(Sensor sensor, int accuracy)
{
//处理精准度改变
}
// 当传感器在被改变时触发
public void onSensorChanged(SensorEvent event)
{
// 接收方向传感器的类型
if (event.sensor.getType() == Sensor.TYPE_ORIENTATION)
{
//这里我们可以得到数据，然后根据需要来处理
//由于模拟器上面无法测试效果，因此我们暂时不处理数据
float x = event.values[SensorManager.DATA_X];
float y = event.values[SensorManager.DATA_Y];
float z = event.values[SensorManager.DATA_Z];
}
}
}
```

上面的例子中演示了如何获得方向传感器的方向、加速度等信息，我们可以根据得到的数值与上一次得到的数值之间的关系来进行需要的操作。SensorManager 中还有很多常量和一些常用的方法，如下：

getDefaultSensor：得到默认的传感器对象。

getInclination：得到地磁传感器倾斜角的弧度值。

getOrientation：得到设备旋转的方向。

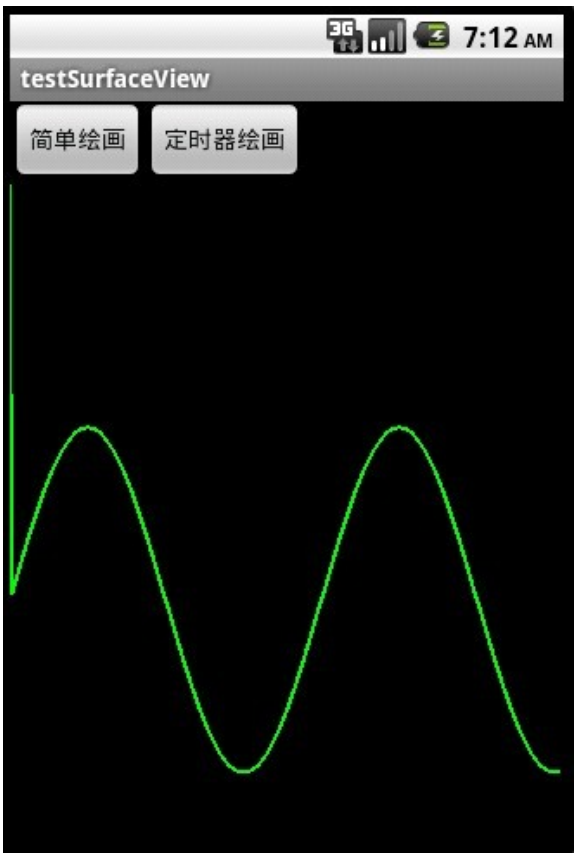
getSensorList：得到指定传感器的列表。

继续阅读文章

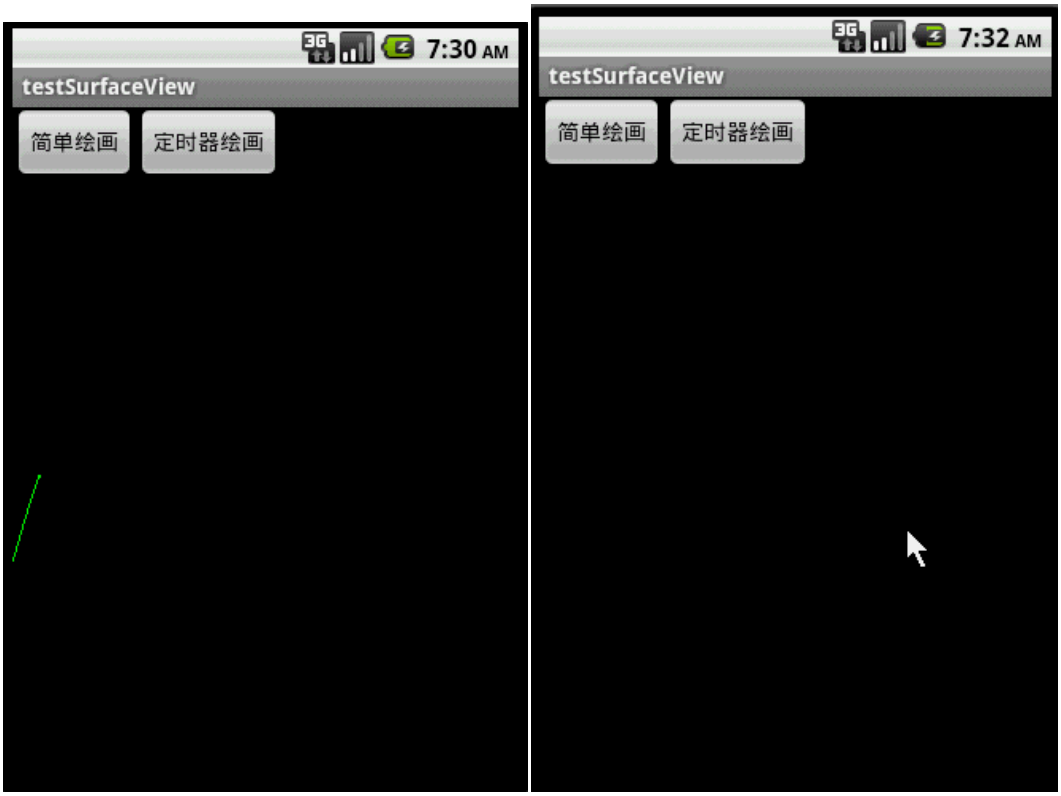
——Android 应用之 SurfaceView

- 文章介绍了 SurfaceView 的用法。网上介绍 SurfaceView 的用法有很多，写法也层出不同，例如继承 SurfaceView 类，或者继承 SurfaceHolder.Callback 类等，这个可以根据功能实际需要自己选择，我这里就直接在普通的用户界面调用 SurfaceHolder 的 lockCanvas 和 unlockCanvasAndPost。

  先来看看程序运行的截图：



截图 1 主要演示了直接把正弦波绘画在 SurfaceView 上



对比上面的左右两图，右图用.lockCanvas(null)，而左图用.lockCanvas(new Rect(oldX, 0, oldX + length, getWindowManager().getDefaultDisplay().getHeight()))，对比一下两个效果，由于左图是按指定 Rect 绘画，所以效率会比右图的全控件绘画高些，并且在清屏之后(canvas.drawColor(Color.BLACK))不会留有上次绘画的残留。

接下来贴出 main.xml 的源码:

```
view plaincopy to clipboardprint?
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:orientation="vertical">

    <LinearLayout android:id="@+id/LinearLayout01"
        android:layout_width="wrap_content" android:layout_height="wrap_content">
        <Button android:id="@+id/Button01" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="简单绘画"></Button>
        <Button android:id="@+id/Button02" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="定时器绘画"></Button>
    </LinearLayout>
```

```
    <SurfaceView android:id="@+id/SurfaceView01"
        android:layout_width="fill_parent" android:layout_height="fill_parent"></SurfaceView>
</LinearLayout>
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent" android:layout_height="fill_parent"
 android:orientation="vertical">

 <LinearLayout android:id="@+id/LinearLayout01"
  android:layout_width="wrap_content" android:layout_height="wrap_content">
  <Button android:id="@+id/Button01" android:layout_width="wrap_content"
   android:layout_height="wrap_content" android:text="简单绘画"></Button>
  <Button android:id="@+id/Button02" android:layout_width="wrap_content"
   android:layout_height="wrap_content" android:text="定时器绘画"></Button>
 </LinearLayout>
 <SurfaceView android:id="@+id/SurfaceView01"
  android:layout_width="fill_parent" android:layout_height="fill_parent"></SurfaceView>
</LinearLayout>
```

- 接下来贴出程序源码：

```
view plaincopy to clipboardprint?
package com.testSurfaceView;

import java.util.Timer;
import java.util.TimerTask;

import android.app.Activity;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Button;

public class testSurfaceView extends Activity {
```

```
/** Called when the activity is first created. */
Button btnSimpleDraw, btnTimerDraw;
SurfaceView sfv;
SurfaceHolder sfh;

private Timer mTimer;
private MyTimerTask mTimerTask;
int Y_axis[],//保存正弦波的 Y 轴上的点
centerY,//中心线
oldX,oldY,//上一个 XY 点
currentX;//当前绘制到的 X 轴上的点

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

    btnSimpleDraw = (Button) this.findViewById(R.id.Button01);
    btnTimerDraw = (Button) this.findViewById(R.id.Button02);
    btnSimpleDraw.setOnClickListener(new ClickEvent());
    btnTimerDraw.setOnClickListener(new ClickEvent());
    sfv = (SurfaceView) this.findViewById(R.id.SurfaceView01);
    sfh = sfv.getHolder();

    //动态绘制正弦波的定时器
    mTimer = new Timer();
    mTimerTask = new MyTimerTask();

    // 初始化 y 轴数据
    centerY = (getWindowManager().getDefaultDisplay().getHeight() - sfv
            .getTop()) / 2;
    Y_axis = new int[getWindowManager().getDefaultDisplay().getWidth()];
    for (int i = 1; i < Y_axis.length; i++) {// 计算正弦波
        Y_axis[i - 1] = centerY
                - (int) (100 * Math.sin(i * 2 * Math.PI / 180));
    }
}

class ClickEvent implements View.OnClickListener {
```

```
    @Override
    public void onClick(View v) {

        if (v == btnSimpleDraw) {
            SimpleDraw(Y_axis.length-1);//直接绘制正弦波

        } else if (v == btnTimerDraw) {
            oldY = centerY;
            mTimer.schedule(mTimerTask, 0, 5);//动态绘制正弦波
        }

    }

}

class MyTimerTask extends TimerTask {
    @Override
    public void run() {

        SimpleDraw(currentX);
        currentX++;//往前进
        if (currentX == Y_axis.length - 1) {//如果到了终点，则清屏重来
            ClearDraw();
            currentX = 0;
            oldY = centerY;
        }
    }

}

/*
 * 绘制指定区域
 */
void SimpleDraw(int length) {
    if (length == 0)
        oldX = 0;
    Canvas canvas = sfh.lockCanvas(new Rect(oldX, 0, oldX + length,
            getWindowManager().getDefaultDisplay().getHeight()));// 关键:获取画布
    Log.i("Canvas:",
            String.valueOf(oldX) + "," + String.valueOf(oldX + length));
```

```
    Paint mPaint = new Paint();
    mPaint.setColor(Color.GREEN);// 画笔为绿色
    mPaint.setStrokeWidth(2);// 设置画笔粗细

    int y;
    for (int i = oldX + 1; i < length; i++) {// 绘画正弦波
        y = Y_axis[i - 1];
        canvas.drawLine(oldX, oldY, i, y, mPaint);
        oldX = i;
        oldY = y;
    }
    sfh.unlockCanvasAndPost(canvas);// 解锁画布，提交画好的图像
}
```


——Android 应用之 SQLite 分页读取

- Android 包含了常用于嵌入式系统的 SQLite，免去了开发者自己移植安装的功夫。SQLite 支持多数 SQL92 标准，很多常用的 SQL 命令都能在 SQLite 上面使用，除此之外 Android 还提供了一系列自定义的方法去简化对 SQLite 数据库的操作。不过有跨平台需求的程序就建议使用标准的 SQL 语句，毕竟这样容易在多个平台之间移植。

  先贴出本文程序运行的结果：

本文主要讲解了 SQLite 的基本用法，如：创建数据库，使用 SQL 命令查询数据表、插入数据，关闭数据库，以及使用 GridView 实现了一个分页栏(关于 GridView 的用法)，用于把数据分页显示。

分页栏的 pagebuttons.xml 的源码如下：

view plaincopy to clipboardprint?

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content" android:paddingBottom="4dip"
    android:layout_width="fill_parent">
    <TextView android:layout_width="wrap_content"
        android:layout_below="@+id/ItemImage" android:layout_height="wrap_content"
        android:text="TextView01" android:layout_centerHorizontal="true"
        android:id="@+id/ItemText">
    </TextView>
</RelativeLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_height="wrap_content" android:paddingBottom="4dip"
 android:layout_width="fill_parent">
 <TextView android:layout_width="wrap_content"
  android:layout_below="@+id/ItemImage" android:layout_height="wrap_content"
  android:text="TextView01" android:layout_centerHorizontal="true"
  android:id="@+id/ItemText">
 </TextView>
</RelativeLayout>
```

main.xml 的源码如下：

view plaincopy to clipboardprint?

```
<?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:layout_height="wrap_content"
        android:layout_width="fill_parent" android:id="@+id/btnCreateDB"
        android:text="创建数据库"></Button>
    <Button android:layout_height="wrap_content"
        android:layout_width="fill_parent" android:text="插入一串实验数据" android:id="@+id/btnInsertRec"></Button>
    <Button android:layout_height="wrap_content" android:id="@+id/btnClose"
        android:text="关闭数据库" android:layout_width="fill_parent"></Button>
    <EditText android:text="@+id/EditText01" android:id="@+id/EditText01"
        android:layout_width="fill_parent" android:layout_height="256dip"></EditText>
    <GridView android:id="@+id/gridview" android:layout_width="fill_parent"
        android:layout_height="32dip" android:numColumns="auto_fit"
        android:columnWidth="40dip"></GridView>
</LinearLayout>
<?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:layout_height="wrap_content"
  android:layout_width="fill_parent" android:id="@+id/btnCreateDB"
  android:text="创建数据库"></Button>
 <Button android:layout_height="wrap_content"
```

```
 android:layout_width="fill_parent" android:text="插入一串实验数据" android:id="@+id/btnInsertRec"></Button>
<Button android:layout_height="wrap_content" android:id="@+id/btnClose"
 android:text="关闭数据库" android:layout_width="fill_parent"></Button>
<EditText android:text="@+id/EditText01" android:id="@+id/EditText01"
 android:layout_width="fill_parent" android:layout_height="256dip"></EditText>
<GridView android:id="@+id/gridview" android:layout_width="fill_parent"
 android:layout_height="32dip" android:numColumns="auto_fit"
 android:columnWidth="40dip"></GridView>
</LinearLayout>
```

本文程序源码如下：

view plaincopy to clipboardprint?

```
package com.testSQLite;

import java.util.ArrayList;
import java.util.HashMap;
import android.app.Activity;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.GridView;
import android.widget.SimpleAdapter;

public class testSQLite extends Activity {
    /** Called when the activity is first created. */
    Button btnCreateDB, btnInsert, btnClose;
    EditText edtSQL;//显示分页数据
    SQLiteDatabase db;
    int id;//添加记录时的 id 累加标记，必须全局
    static final int PageSize=10;//分页时，每页的数据总数
    private static final String TABLE_NAME = "stu";
    private static final String ID = "id";
```

```
private static final String NAME = "name";

SimpleAdapter saPageID;// 分页栏适配器
ArrayList<HashMap<String, String>> lstPageID;// 分页栏的数据源，与 PageSize 和数据总数相关

@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   btnCreateDB = (Button) this.findViewById(R.id.btnCreateDB);
   btnCreateDB.setOnClickListener(new ClickEvent());

   btnInsert = (Button) this.findViewById(R.id.btnInsertRec);
   btnInsert.setOnClickListener(new ClickEvent());

   btnClose = (Button) this.findViewById(R.id.btnClose);
   btnClose.setOnClickListener(new ClickEvent());

   edtSQL=(EditText)this.findViewById(R.id.EditText01);

   GridView gridview = (GridView) findViewById(R.id.gridview);//分页栏控件
   // 生成动态数组，并且转入数据
   lstPageID = new ArrayList<HashMap<String, String>>();

   // 生成适配器的 ImageItem <====> 动态数组的元素，两者一一对应
   saPageID = new SimpleAdapter(testSQLite.this, // 没什么解释
         lstPageID,// 数据来源
         R.layout.pagebuttons,//XML 实现
         new String[] { "ItemText" },
         new int[] { R.id.ItemText });

   // 添加并且显示
   gridview.setAdapter(saPageID);
   // 添加消息处理
   gridview.setOnItemClickListener(new OnItemClickListener(){

      @Override
      public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
            long arg3) {
         LoadPage(arg2);//根据所选分页读取对应的数据
```

```
        }
    });

}
```

继续阅读文章

——Android 提高之探秘蓝牙隐藏 API

- 本文探讨下蓝牙方面的隐藏 API。用过 Android 系统设置(Setting)的人都知道蓝牙搜索之后可以建立配对和解除配对，但是这两项功能的函数没有在 SDK 中给出，那么如何去使用这两项功能呢？本文利用 JAVA 的反射机制去调用这两项功能对应的函数：createBond 和 removeBond，具体的发掘和实现步骤如下：

  1.使用 Git 工具下载 platform/packages/apps/Settings.git，在 Setting 源码中查找关于建立配对和解除配对的 API，知道这两个 API 的宿主(BluetoothDevice)；

  2.使用反射机制对 BluetoothDevice 枚举其所有方法和常量，看看是否存在:

  view plaincopy to clipboardprint?

```
static public void printAllInform(Class clsShow) {
    try {
        // 取得所有方法
        Method[] hideMethod = clsShow.getMethods();
        int i = 0;
        for (; i < hideMethod.length; i++) {
            Log.e("method name", hideMethod[i].getName());
        }
        // 取得所有常量
        Field[] allFields = clsShow.getFields();
        for (i = 0; i < allFields.length; i++) {
            Log.e("Field name", allFields[i].getName());
        }
    } catch (SecurityException e) {
        // throw new RuntimeException(e.getMessage());
        e.printStackTrace();
    } catch (IllegalArgumentException e) {
        // throw new RuntimeException(e.getMessage());
        e.printStackTrace();
```

```
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
 static public void printAllInform(Class clsShow) {
  try {
   // 取得所有方法
   Method[] hideMethod = clsShow.getMethods();
   int i = 0;
   for (; i < hideMethod.length; i++) {
    Log.e("method name", hideMethod[i].getName());
   }
   // 取得所有常量
   Field[] allFields = clsShow.getFields();
   for (i = 0; i < allFields.length; i++) {
    Log.e("Field name", allFields[i].getName());
   }
  } catch (SecurityException e) {
   // throw new RuntimeException(e.getMessage());
   e.printStackTrace();
  } catch (IllegalArgumentException e) {
   // throw new RuntimeException(e.getMessage());
   e.printStackTrace();
  } catch (Exception e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
 }
```

结果如下：

```
11-29 09:19:12.012: method name(452): cancelBondProcess
11-29 09:19:12.020: method name(452): cancelPairingUserInput
11-29 09:19:12.020: method name(452): createBond
11-29 09:19:12.020: method name(452): createInsecureRfcommSocket
11-29 09:19:12.027: method name(452): createRfcommSocket
11-29 09:19:12.027: method name(452): createRfcommSocketToServiceRecord
11-29 09:19:12.027: method name(452): createScoSocket
11-29 09:19:12.027: method name(452): describeContents
11-29 09:19:12.035: method name(452): equals
```

11-29 09:19:12.035: method name(452): fetchUuidsWithSdp
11-29 09:19:12.035: method name(452): getAddress
11-29 09:19:12.035: method name(452): getBluetoothClass
11-29 09:19:12.043: method name(452): getBondState
11-29 09:19:12.043: method name(452): getName
11-29 09:19:12.043: method name(452): getServiceChannel
11-29 09:19:12.043: method name(452): getTrustState
11-29 09:19:12.043: method name(452): getUuids
11-29 09:19:12.043: method name(452): hashCode
11-29 09:19:12.043: method name(452): isBluetoothDock
11-29 09:19:12.043: method name(452): removeBond
11-29 09:19:12.043: method name(452): setPairingConfirmation
11-29 09:19:12.043: method name(452): setPasskey
11-29 09:19:12.043: method name(452): setPin
11-29 09:19:12.043: method name(452): setTrust
11-29 09:19:12.043: method name(452): toString
11-29 09:19:12.043: method name(452): writeToParcel
11-29 09:19:12.043: method name(452): convertPinToBytes
11-29 09:19:12.043: method name(452): getClass
11-29 09:19:12.043: method name(452): notify
11-29 09:19:12.043: method name(452): notifyAll
11-29 09:19:12.043: method name(452): wait
11-29 09:19:12.051: method name(452): wait
11-29 09:19:12.051: method name(452): wait

3.如果枚举发现 API 存在(SDK 却隐藏)，则自己实现调用方法：

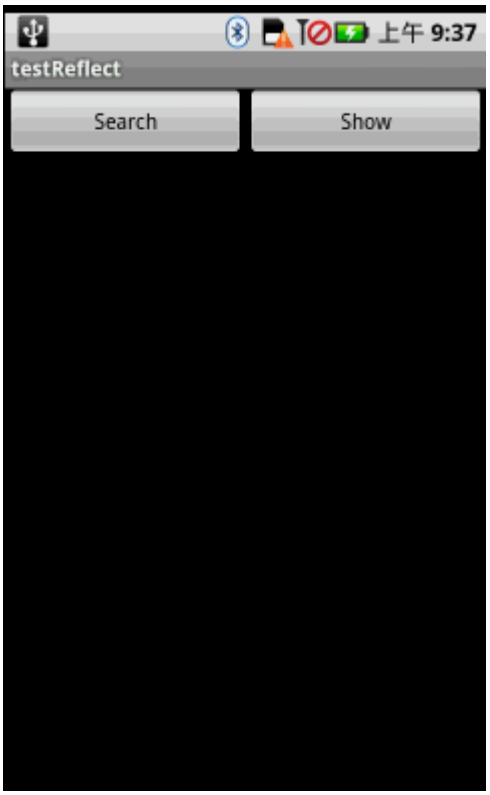
view plaincopy to clipboardprint?

```
/**
 * 与设备配对 参考源码：platform/packages/apps/Settings.git
 * \Settings\src\com\android\settings\bluetooth\CachedBluetoothDevice.java
 */
static public boolean createBond(Class btClass,BluetoothDevice btDevice) throws Exception {
    Method createBondMethod = btClass.getMethod("createBond");
    Boolean returnValue = (Boolean) createBondMethod.invoke(btDevice);
    return returnValue.booleanValue();
}

/**
 * 与设备解除配对 参考源码：platform/packages/apps/Settings.git
 * \Settings\src\com\android\settings\bluetooth\CachedBluetoothDevice.java
```

```
 */
static public boolean removeBond(Class btClass,BluetoothDevice btDevice) throws Exception {
    Method removeBondMethod = btClass.getMethod("removeBond");
    Boolean returnValue = (Boolean) removeBondMethod.invoke(btDevice);
    return returnValue.booleanValue();
}
 /**
  * 与设备配对 参考源码：platform/packages/apps/Settings.git
  * \Settings\src\com\android\settings\bluetooth\CachedBluetoothDevice.java
  */
 static public boolean createBond(Class btClass,BluetoothDevice btDevice) throws Exception {
  Method createBondMethod = btClass.getMethod("createBond");
  Boolean returnValue = (Boolean) createBondMethod.invoke(btDevice);
  return returnValue.booleanValue();
 }

 /**
  * 与设备解除配对 参考源码：platform/packages/apps/Settings.git
  * \Settings\src\com\android\settings\bluetooth\CachedBluetoothDevice.java
  */
 static public boolean removeBond(Class btClass,BluetoothDevice btDevice) throws Exception {
  Method removeBondMethod = btClass.getMethod("removeBond");
  Boolean returnValue = (Boolean) removeBondMethod.invoke(btDevice);
  return returnValue.booleanValue();
 }
```

PS:SDK 之所以不给出隐藏的 API 肯定有其原因，也许是出于安全性或者是后续版本兼容性的考虑，因此不能保证隐藏 API 能在所有 Android 平台上很好地运行。。。

本文程序运行效果如下：



main.xml 源码如下：

view plaincopy to clipboardprint?

```
<?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/LinearLayout01"
        android:layout_height="wrap_content" android:layout_width="fill_parent">
        <Button android:layout_height="wrap_content" android:id="@+id/btnSearch"
            android:text="Search" android:layout_width="160dip"></Button>
        <Button android:layout_height="wrap_content"
            android:layout_width="160dip" android:text="Show" android:id="@+id/btnShow"></Button>
    </LinearLayout>
    <LinearLayout android:id="@+id/LinearLayout02"
        android:layout_width="wrap_content" android:layout_height="wrap_content"></LinearLayout>
    <ListView android:id="@+id/ListView01" android:layout_width="fill_parent"
        android:layout_height="fill_parent">
    </ListView>
</LinearLayout>
<?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/LinearLayout01"
  android:layout_height="wrap_content" android:layout_width="fill_parent">
  <Button android:layout_height="wrap_content" android:id="@+id/btnSearch"
   android:text="Search" android:layout_width="160dip"></Button>
  <Button android:layout_height="wrap_content"
   android:layout_width="160dip" android:text="Show" android:id="@+id/btnShow"></Button>
 </LinearLayout>
 <LinearLayout android:id="@+id/LinearLayout02"
  android:layout_width="wrap_content" android:layout_height="wrap_content"></LinearLayout>
 <ListView android:id="@+id/ListView01" android:layout_width="fill_parent"
  android:layout_height="fill_parent">
 </ListView>
</LinearLayout>
```

工具类 ClsUtils.java 源码如下：

```
view plaincopy to clipboardprint?
package com.testReflect;

import java.lang.reflect.Field;
import java.lang.reflect.Method;

import android.bluetooth.BluetoothDevice;
import android.util.Log;

public class ClsUtils {
```


——Android 应用之蓝牙传感应用

- 如果传感器本身需要包含控制电路（例如采集血氧信号需要红外和红外线交替发射），那么传感器本身就需要带一片主控 IC，片内采集并输出数字信号了。Android 手机如何在不改硬件电路的前提下与这类数字传感器交互呢？可选的通信方式

就有 USB 和蓝牙，两种方式各有好处：USB 方式可以给传感器供电，蓝牙方式要自备电源；USB 接口标准不一，蓝牙普遍支持 SPP 协议。本文选择蓝牙方式做介绍，介绍 Android 的蓝牙 API 以及蓝牙客户端的用法。

在 Android 2.0，官方终于发布了蓝牙 API（2.0 以下系统的非官方的蓝牙 API 可以参考这里：http://code.google.com/p/android-bluetooth/）。Android 手机一般以客户端的角色主动连接 SPP 协议设备(接上蓝牙模块的数字传感器)，连接流程是：

1.使用 registerReceiver 注册 BroadcastReceiver 来获取蓝牙状态、搜索设备等消息；

2.使用 BlueAdatper 的搜索；

3.在 BroadcastReceiver 的 onReceive()里取得搜索所得的蓝牙设备信息(如名称，MAC，RSSI)；

4.通过设备的 MAC 地址来建立一个 BluetoothDevice 对象；

5.由 BluetoothDevice 衍生出 BluetoothSocket，准备 SOCKET 来读写设备；

6.通过 BluetoothSocket 的 createRfcommSocketToServiceRecord()方法来选择连接的协议/服务，这里用的是 SPP(UUID:00001101-0000-1000-8000-00805F9B34FB)；

7.Connect 之后(如果还没配对则系统自动提示)，使用 BluetoothSocket 的 getInputStream()和 getOutputStream()来读写蓝牙设备。

先来看看本文程序运行的效果图，所选的 SPP 协议设备是一款单导联心电采集表：

本文的代码较多，可以到这里下载：本文程序包含两个 Activity(testBlueTooth 和 WaveDiagram)，testBlueTooth 是搜索建立蓝牙连接。BluetoothAdapter、BluetoothDevice 和 BluetoothSocket 的使用很简单，除了前三者提供的功能外，还可以通过给系统发送消息来控制、获取蓝牙信息，例如：

注册 BroadcastReceiver：

view plaincopy to clipboardprint?

```
IntentFilter intent = new IntentFilter();
intent.addAction(BluetoothDevice.ACTION_FOUND);// 用 BroadcastReceiver 来取得搜索结果
intent.addAction(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
intent.addAction(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED);
intent.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
registerReceiver(searchDevices, intent);
  IntentFilter intent = new IntentFilter();
  intent.addAction(BluetoothDevice.ACTION_FOUND);// 用 BroadcastReceiver 来取得搜索结果
  intent.addAction(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
  intent.addAction(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED);
  intent.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
  registerReceiver(searchDevices, intent);
```

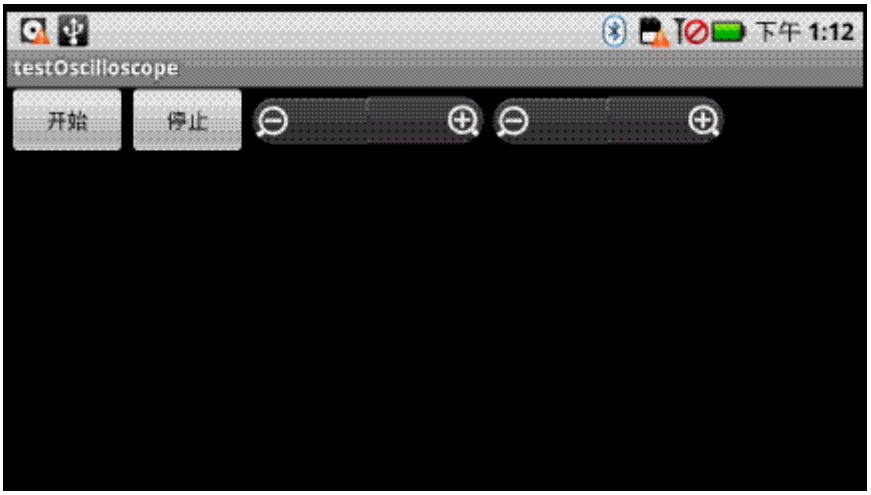
在 BroadcastReceiver 的 onReceive()枚举所有消息的内容：

继续阅读文章

——Android 提高应用篇之模拟信号示波器

- 本文结合 SurfaceView 实现一个 Android 版的手机模拟信号示波器(PS：以前也讲过 J2ME 版的手机示波器)。最近物联网炒得很火，作为手机软件开发者，如何在不修改手机硬件电路的前提下实现与第三方传感器结合呢？麦克风就是一个很好的 ADC 接口，通过麦克风与第三方传感器结合，再在软件里对模拟信号做相应的处理，就可以提供更丰富的传感化应用。

  先来看看本文程序运行的效果图(屏幕录像速度较慢，真机实际运行起来会更加流畅)：



本文程序使用 8000hz 的采样率，对 X 轴方向绘图的实时性要求较高，如果不降低 X 轴的分辨率，程序的实时性较差，因此程序对 X 轴数据缩小区间为 8 倍～16 倍。由于采用 16 位采样，因此 Y 轴数据的高度相对于手机屏幕来说也偏大，程序也对 Y 轴数据做缩小，区间为 1 倍～10 倍。在 SurfaceView 的 OnTouchListener 方法里加入了波形基线的位置调节，直接在 SurfaceView 控件上触摸即可控制整体波形偏上或偏下显示。

main.xml 源码如下：

view plaincopy to clipboardprint?

```
<?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/LinearLayout01"
        android:layout_height="wrap_content" android:layout_width="fill_parent"
        android:orientation="horizontal">
        <Button android:layout_height="wrap_content" android:id="@+id/btnStart"
            android:text="开始" android:layout_width="80dip"></Button>
        <Button android:layout_height="wrap_content" android:text="停止"
            android:id="@+id/btnExit" android:layout_width="80dip"></Button>
        <ZoomControls android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/zctlX"></ZoomControls>
        <ZoomControls android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/zctlY"></ZoomControls>
    </LinearLayout>
    <SurfaceView android:id="@+id/SurfaceView01"
        android:layout_height="fill_parent" android:layout_width="fill_parent"></SurfaceView>
</LinearLayout>
<?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/LinearLayout01"
  android:layout_height="wrap_content" android:layout_width="fill_parent"
  android:orientation="horizontal">
  <Button android:layout_height="wrap_content" android:id="@+id/btnStart"
   android:text="开始" android:layout_width="80dip"></Button>
  <Button android:layout_height="wrap_content" android:text="停止"
   android:id="@+id/btnExit" android:layout_width="80dip"></Button>
  <ZoomControls android:layout_width="wrap_content"
   android:layout_height="wrap_content" android:id="@+id/zctlX"></ZoomControls>
  <ZoomControls android:layout_width="wrap_content"
   android:layout_height="wrap_content" android:id="@+id/zctlY"></ZoomControls>
 </LinearLayout>
 <SurfaceView android:id="@+id/SurfaceView01"
  android:layout_height="fill_parent" android:layout_width="fill_parent"></SurfaceView>
</LinearLayout>
```

ClsOscilloscope.java 是实现示波器的类库，包含 AudioRecord 操作线程和 SurfaceView 绘图线程的实现，两个线程同步操作，代码如下：

```
view plaincopy to clipboardprint?
package com.testOscilloscope;
import java.util.ArrayList;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.media.AudioRecord;
import android.view.SurfaceView;
public class ClsOscilloscope {
    private ArrayList<short[]> inBuf = new ArrayList<short[]>();
    private boolean isRecording = false;//  线程控制标记
    /**
     * X 轴缩小的比例
     */
    public int rateX = 4;
    /**
     * Y 轴缩小的比例
```

```
 */
public int rateY = 4;
/**
 * Y 轴基线
 */
public int baseLine = 0;
/**
 * 初始化
 */
public void initOscilloscope(int rateX, int rateY, int baseLine) {
    this.rateX = rateX;
    this.rateY = rateY;
    this.baseLine = baseLine;
}
/**
 * 开始
 *
 * @param recBufSize
 *            AudioRecord 的 MinBufferSize
 */
public void Start(AudioRecord audioRecord, int recBufSize, SurfaceView sfv,
        Paint mPaint) {
    isRecording = true;
    new RecordThread(audioRecord, recBufSize).start();// 开始录制线程
    new DrawThread(sfv, mPaint).start();// 开始绘制线程
}
/**
 * 停止
 */
public void Stop() {
    isRecording = false;
    inBuf.clear();// 清除
}
/**
 * 负责从 MIC 保存数据到 inBuf
 *
 * @author GV
 *
 */
class RecordThread extends Thread {
```

```
    private int recBufSize;
    private AudioRecord audioRecord;
    public RecordThread(AudioRecord audioRecord, int recBufSize) {
        this.audioRecord = audioRecord;
        this.recBufSize = recBufSize;
    }
    public void run() {
        try {
            short[] buffer = new short[recBufSize];
            audioRecord.startRecording();// 开始录制
            while (isRecording) {
                // 从 MIC 保存数据到缓冲区
                int bufferReadResult = audioRecord.read(buffer, 0,
                        recBufSize);
                short[] tmpBuf = new short[bufferReadResult / rateX];
                for (int i = 0, ii = 0; i < tmpBuf.length; i++, ii = i
                        * rateX) {
                    tmpBuf[i] = buffer[ii];
                }
                synchronized (inBuf) {//
                    inBuf.add(tmpBuf);// 添加数据
                }
            }
            audioRecord.stop();
        } catch (Throwable t) {
        }
    }
};
/**
 * 负责绘制 inBuf 中的数据
 *
 * @author GV
 *
 */
class DrawThread extends Thread {
    private int oldX = 0;// 上次绘制的 X 坐标
    private int oldY = 0;// 上次绘制的 Y 坐标
    private SurfaceView sfv;// 画板
    private int X_index = 0;// 当前画图所在屏幕 X 轴的坐标
    private Paint mPaint;// 画笔
```

```
public DrawThread(SurfaceView sfv, Paint mPaint) {
    this.sfv = sfv;
    this.mPaint = mPaint;
}
public void run() {
    while (isRecording) {
        ArrayList<short[]> buf = new ArrayList<short[]>();
        synchronized (inBuf) {
            if (inBuf.size() == 0)
                continue;
            buf = (ArrayList<short[]>) inBuf.clone();// 保存
            inBuf.clear();// 清除
        }
        for (int i = 0; i < buf.size(); i++) {
            short[] tmpBuf = buf.get(i);
            SimpleDraw(X_index, tmpBuf, rateY, baseLine);// 把缓冲区数据画出来
            X_index = X_index + tmpBuf.length;
            if (X_index > sfv.getWidth()) {
                X_index = 0;
            }
        }
    }
}
/**
 * 绘制指定区域
 *
 * @param start
 *            X 轴开始的位置(全屏)
 * @param buffer
 *            缓冲区
 * @param rate
 *            Y 轴数据缩小的比例
 * @param baseLine
 *            Y 轴基线
 */
void SimpleDraw(int start, short[] buffer, int rate, int baseLine) {
    if (start == 0)
        oldX = 0;
    Canvas canvas = sfv.getHolder().lockCanvas(
            new Rect(start, 0, start + buffer.length, sfv.getHeight()));// 关键:获取画布
```

```
canvas.drawColor(Color.BLACK);// 清除背景
int y;
for (int i = 0; i < buffer.length; i++) {// 有多少画多少
    int x = i + start;
    y = buffer[i] / rate + baseLine;// 调节缩小比例，调节基准线
    canvas.drawLine(oldX, oldY, x, y, mPaint);
    oldX = x;
    oldY = y;
}
sfv.getHolder().unlockCanvasAndPost(canvas);// 解锁画布，提交画好的图像
```

继续阅读文章

——Android 应用之 SurfaceView 的双缓冲使用

- 这次介绍 SurfaceView 的双缓冲使用。双缓冲是为了防止动画闪烁而实现的一种多线程应用，基于 SurfaceView 的双缓冲实现很简单，开一条线程并在其中绘图即可。本文介绍基于 SurfaceView 的双缓冲实现，以及介绍类似的更高效的实现方法。

本文程序运行截图如下，左边是开单个线程读取并绘图，右边是开两个线程，一个专门读取图片，一个专门绘图：

对比一下，右边动画的帧速明显比左边的快，左右两者都没使用 Thread.sleep()。为什么要开两个线程一个读一个画，而不去开两个线程像左边那样都“边读边画”呢？因为 SurfaceView 每次绘图都会锁定 Canvas,也就是说同一片区域这次没画完下次就不能画，因此要提高双缓冲的效率，就得开一条线程专门画图，开另外一条线程做预处理的工作。

main.xml 的源码：

view plaincopy to clipboardprint?

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

    <LinearLayout android:id="@+id/LinearLayout01"
        android:layout_width="wrap_content" android:layout_height="wrap_content">
        <Button android:id="@+id/Button01" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="单个独立线程"></Button>
        <Button android:id="@+id/Button02" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="两个独立线程"></Button>
    </LinearLayout>
    <SurfaceView android:id="@+id/SurfaceView01"
        android:layout_width="fill_parent" android:layout_height="fill_parent"></SurfaceView>
</LinearLayout>
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent" android:layout_height="fill_parent"
 android:orientation="vertical">

 <LinearLayout android:id="@+id/LinearLayout01"
  android:layout_width="wrap_content" android:layout_height="wrap_content">
  <Button android:id="@+id/Button01" android:layout_width="wrap_content"
   android:layout_height="wrap_content" android:text="单个独立线程"></Button>
  <Button android:id="@+id/Button02" android:layout_width="wrap_content"
   android:layout_height="wrap_content" android:text="两个独立线程"></Button>
 </LinearLayout>
 <SurfaceView android:id="@+id/SurfaceView01"
  android:layout_width="fill_parent" android:layout_height="fill_parent"></SurfaceView>
</LinearLayout>
```

本文程序的源码：

```
 view plaincopy to clipboardprint?
package com.testSurfaceView;

import java.lang.reflect.Field;
import java.util.ArrayList;
import android.app.Activity;
import android.graphics.Bitmap;
```

```
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Button;

public class testSurfaceView extends Activity {
    /** Called when the activity is first created. */
    Button btnSingleThread, btnDoubleThread;
    SurfaceView sfv;
    SurfaceHolder sfh;
    ArrayList<Integer> imgList = new ArrayList<Integer>();
    int imgWidth, imgHeight;
    Bitmap bitmap;//独立线程读取，独立线程绘图

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

        btnSingleThread = (Button) this.findViewById(R.id.Button01);
        btnDoubleThread = (Button) this.findViewById(R.id.Button02);
        btnSingleThread.setOnClickListener(new ClickEvent());
        btnDoubleThread.setOnClickListener(new ClickEvent());
        sfv = (SurfaceView) this.findViewById(R.id.SurfaceView01);
        sfh = sfv.getHolder();
        sfh.addCallback(new MyCallBack());//  自动运行 surfaceCreated 以及 surfaceChanged
    }

    class ClickEvent implements View.OnClickListener {

        @Override
        public void onClick(View v) {

            if (v == btnSingleThread) {
```

```
            new Load_DrawImage(0, 0).start();//开一条线程读取并绘图
        } else if (v == btnDoubleThread) {
            new LoadImage().start();//开一条线程读取
            new DrawImage(imgWidth + 10, 0).start();//开一条线程绘图
        }

    }

}

class MyCallBack implements SurfaceHolder.Callback {

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
            int height) {
        Log.i("Surface:", "Change");

    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        Log.i("Surface:", "Create");
```

继续阅读文章

——Android 3D 游戏实现入门

- 此示例展示了一个立方体的具体实现过程，与之前的纯 Opengl es 实现相比，它采用了 JPCT-AE 来实现，因为个人认为这个框架很方便，于是从今天开始通过其网站上的 Wiki 来介绍 JPCT-AE 的实现。通过这个示例能让你快速了解 JPCT-AE 的帮助文档，也就是入门。

(1)什么是 JPCT：一种封装了 OPENGL es 的 3D 游戏引擎，有 j2se 与 android 两个版本。

(2)如何获得其 jar 包及帮助文档：http://download.csdn.net/user/Simdanfeg 处下载

第一个示例:同样的立方体，不同的实现

```
package com.threed.jpct.example;

import java.lang.reflect.Field;

import javax.microedition.khronos.egl.EGL10;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.egl.EGLDisplay;
import javax.microedition.khronos.opengles.GL10;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.view.MotionEvent;

import com.threed.jpct.Camera;
import com.threed.jpct.FrameBuffer;
import com.threed.jpct.Light;
import com.threed.jpct.Logger;
import com.threed.jpct.Object3D;
import com.threed.jpct.Primitives;
import com.threed.jpct.RGBColor;
import com.threed.jpct.SimpleVector;
import com.threed.jpct.Texture;
import com.threed.jpct.TextureManager;
import com.threed.jpct.World;
import com.threed.jpct.util.BitmapHelper;
import com.threed.jpct.util.MemoryHelper;

/**
 * 一个简单的例子。比起展示如何写一个正确的 android 应用它更着重于展示如何使用 JPCT-AE 这个 3D 游戏框架。
 * 它包含了 Activity 类去处理 pause 和 resume 等方法
 *
 * @author EgonOlsen
 *
```

```
 */
public class HelloWorld extends Activity {

    // HelloWorld 对象用来处理 Activity 的 onPause 和 onResume 方法
    private static HelloWorld master = null;

    // GLSurfaceView 对象
    private GLSurfaceView mGLView;

    // 类 MyRenderer 对象
    private MyRenderer renderer = null;

    // 当 JPCT 渲染背景时 FrameBuffer 类提供了一个缓冲,它的结果本质上是一个能显示或者修改甚至能进行更多后处理的图片。
    private FrameBuffer fb = null;

    // World 类是 JPCT 时最重要的一个类，它好像胶水一样把事物"粘"起来。它包含的对象和光线定义了 JPCT 的场景
    private World world = null;

    // 类似 java.awt.*中的 Color 类
    private RGBColor back = new RGBColor(50, 50, 100);

    private float touchTurn = 0;
    private float touchTurnUp = 0;

    private float xpos = -1;
    private float ypos = -1;

    // Object3D 类是一个三维对象,千万不要傻呼呼的认为它与 java.lang.Object 类似。
    // 一个 Object3D 对象作为一个实例被添加到在渲染的 World 对象中。Object3D 在 World
    // 中一次添加一个实例 ，他们可能被联系起作为孩子/父母来在他们中建立一个制度.
    // 人体模型当然也能应用在以上的规则中。他们常常不加到一个 World 实例中，而是
    // 绑定到其它对象中(人体模型或非人体模型)。有些方法 在这个类中需要一个实例
    // 添加到一个 World 实例中(用 World.addObject()方法可以实现)。
    private Object3D cube = null;

    // 每秒帧数
    private int fps = 0;

    // 光照类
    private Light sun = null;
```

```
    protected void onCreate(Bundle savedInstanceState) {
        // Logger 类中 jPCT 中一个普通的用于打印和存储消息，错误和警告的日志类。
        // 每一个 JPCT 生成的消息将被加入到这个类的队列中
        Logger.log("onCreate");
        // 如果本类对象不为 NULL,将从 Object 中所有属性装入该类
        if (master != null) {
            copy(master);
        }

        super.onCreate(savedInstanceState);

        // 实例化 GLSurfaceView
        mGLView = new GLSurfaceView(this);
        // 使用自己实现的 EGLConfigChooser,该实现必须在 setRenderer(renderer)之前
        // 如果没有 setEGLConfigChooser 方法被调用，则默认情况下，视图将选择一个与当前 android.view.Surface 兼容至少 16 位深度缓冲深度
EGLConfig。
        mGLView.setEGLConfigChooser(new GLSurfaceView.EGLConfigChooser() {
            public EGLConfig chooseConfig(EGL10 egl, EGLDisplay display) {
                // Ensure that we get a 16bit framebuffer. Otherwise, we""""""""""ll fall
                // back to Pixelflinger on some device (read: Samsung I7500)
                int[] attributes = new int[] { EGL10.EGL_DEPTH_SIZE, 16,
                        EGL10.EGL_NONE };
                EGLConfig[] configs = new EGLConfig[1];
                int[] result = new int[1];
                egl.eglChooseConfig(display, attributes, configs, 1, result);
                return configs[0];
            }
        });
        // 实例化 MyRenderer
        renderer = new MyRenderer();
        // 设置 View 的渲染器，同时启动线程调用渲染，以至启动渲染
        mGLView.setRenderer(renderer);
        // 设置一个明确的视图
        setContentView(mGLView);
    }

    // 重写 onPause()
    @Override
    protected void onPause() {
```

```
    super.onPause();
    mGLView.onPause();
}

// 重写 onResume()
@Override
protected void onResume() {
    super.onResume();
    mGLView.onResume();
}

// 重写 onStop()
@Override
protected void onStop() {
    super.onStop();
}



private void copy(Object src) {
    try {
        // 打印日志
        Logger.log("Copying data from master Activity!");
        // 返回一个数组，其中包含目前这个类的的所有字段的 Filed 对象
        Field[] fs = src.getClass().getDeclaredFields();
        // 遍历 fs 数组
        for (Field f : fs) {
            // 尝试设置无障碍标志的值。标志设置为 false 将使访问检查，设置为 true，将其禁用。
            f.setAccessible(true);
            // 将取到的值全部装入当前类中
            f.set(this, f.get(src));
        }
    } catch (Exception e) {
        // 抛出运行时异常
        throw new RuntimeException(e);
    }
}
```

```
public boolean onTouchEvent(MotionEvent me) {

    // 按键开始
    if (me.getAction() == MotionEvent.ACTION_DOWN) {
        // 保存按下的初始 x,y 位置于 xpos,ypos 中
        xpos = me.getX();
        ypos = me.getY();
        return true;
    }
    // 按键结束
    if (me.getAction() == MotionEvent.ACTION_UP) {
        // 设置 x,y 及旋转角度为初始值
        xpos = -1;
        ypos = -1;
        touchTurn = 0;
        touchTurnUp = 0;
        return true;
    }

    if (me.getAction() == MotionEvent.ACTION_MOVE) {
        // 计算 x,y 偏移位置及 x,y 轴上的旋转角度
        float xd = me.getX() - xpos;
        float yd = me.getY() - ypos;
        // Logger.log("me.getX() - xpos---------->>"
        // + (me.getX() - xpos));
        xpos = me.getX();
        ypos = me.getY();
        Logger.log("xpos---------->>" + xpos);
        // Logger.log("ypos---------->>" + ypos);
        // 以 x 轴为例，鼠标从左向右拉为正，从右向左拉为负
        touchTurn = xd / -100f;
        touchTurnUp = yd / -100f;
        Logger.log("touchTurn---------->>" + touchTurn);
        // Logger.log("touchTurnUp---------->>" + touchTurnUp);
        return true;
    }

    // 每 Move 一下休眠毫秒
    try {
        Thread.sleep(15);
```

```
    } catch (Exception e) {
        // No need for this...
    }

    return super.onTouchEvent(me);
}

// MyRenderer 类实现 GLSurfaceView.Renderer 接口
class MyRenderer implements GLSurfaceView.Renderer {
    // 当前系统的毫秒数
    private long time = System.currentTimeMillis();
    // 是否停止
    private boolean stop = false;

    // 停止
    public void stop() {
        stop = true;
    }

    // 当屏幕改变时
    public void onSurfaceChanged(GL10 gl, int w, int h) {
        // 如果 FrameBuffer 不为 NULL,释放 fb 所占资源
        if (fb != null) {
            fb.dispose();
        }
        // 创建一个宽度为 w,高为 h 的 FrameBuffer
        fb = new FrameBuffer(gl, w, h);
        Logger.log(master + "");
        // 如果 master 为空
        if (master == null) {

            // 实例化 World 对象
            world = new World();

            // 设置了环境光源强度。设置此值是负的整个场景会变暗，而为正将照亮了一切。
            world.setAmbientLight(20, 20, 20);

            // 在 World 中创建一个新的光源
            sun = new Light(world);
```

```
// 设置光照强度
sun.setIntensity(250, 250, 250);

// 创建一个纹理
// 构造方法 Texture(Bitmap image)
// static Bitmap rescale(Bitmap bitmap, int width, int height)
// static Bitmap convert(Drawable drawable)
Texture texture = new Texture(BitmapHelper.rescale(
        BitmapHelper.convert(getResources().getDrawable(
                R.drawable.glass)), 64, 64));

// TextureManager.getInstance()取得一个 Texturemanager 对象
// addTexture("texture",texture)添加一个纹理
TextureManager.getInstance().addTexture("texture", texture);

// Object3D 对象开始了:-)

// Primitives 提供了一些基本的三维物体，假如你为了测试而生成一些对象或为
// 其它目的使用这些类将很明智，因为它即快速又简单，不需要载入和编辑。
// 调用 public static Object3D getCube(float scale) scale:角度
// 返回一个立方体
cube = Primitives.getCube(10);

// 以纹理的方式给对象所有面"包装"上纹理
cube.calcTextureWrapSpherical();

// 给对象设置纹理
cube.setTexture("texture");

// 除非你想在事后再用 PolygonManager 修改,否则释放那些不再需要数据的内存
cube.strip();

// 初始化一些基本的对象是几乎所有进一步处理所需的过程。
// 如果对象是"准备渲染"(装载，纹理分配，安置，渲染模式设置，
// 动画和顶点控制器分配),那么 build()必须被调用，
cube.build();

// 将 Object3D 对象添加到 world 集合
```

- 此示例展示了一个立方体的具体实现过程，与之前的纯 Opengl es 实现相比，它采用了 JPCT-AE 来实现，因为个人认为这个框架很方便，于是从今天开始通过其网站上的 Wiki 来介绍 JPCT-AE 的实现。通过这个示例能让你快速了解 JPCT-AE 的帮助文档，也就是入门。

(1)什么是 JPCT：一种封装了 OPENGL es 的 3D 游戏引擎，有 j2se 与 android 两个版本。

(2)如何获得其 jar 包及帮助文档：http://download.csdn.net/user/Simdanfeg 处下载

第一个示例:同样的立方体，不同的实现

```
package com.threed.jpct.example;

import java.lang.reflect.Field;

import javax.microedition.khronos.egl.EGL10;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.egl.EGLDisplay;
import javax.microedition.khronos.opengles.GL10;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.view.MotionEvent;

import com.threed.jpct.Camera;
import com.threed.jpct.FrameBuffer;
import com.threed.jpct.Light;
import com.threed.jpct.Logger;
import com.threed.jpct.Object3D;
import com.threed.jpct.Primitives;
import com.threed.jpct.RGBColor;
import com.threed.jpct.SimpleVector;
import com.threed.jpct.Texture;
import com.threed.jpct.TextureManager;
import com.threed.jpct.World;
import com.threed.jpct.util.BitmapHelper;
import com.threed.jpct.util.MemoryHelper;
```

```
/**
 * 一个简单的例子。比起展示如何写一个正确的 android 应用它更着重于展示如何使用 JPCT-AE 这个 3D 游戏框架。
 * 它包含了 Activity 类去处理 pause 和 resume 等方法
 *
 * @author EgonOlsen
 *
 */
public class HelloWorld extends Activity {

    // HelloWorld 对象用来处理 Activity 的 onPause 和 onResume 方法
    private static HelloWorld master = null;

    // GLSurfaceView 对象
    private GLSurfaceView mGLView;

    // 类 MyRenderer 对象
    private MyRenderer renderer = null;

    // 当 JPCT 渲染背景时 FrameBuffer 类提供了一个缓冲,它的结果本质上是一个能显示或者修改甚至能进行更多后处理的图片。
    private FrameBuffer fb = null;

    // World 类是 JPCT 时最重要的一个类，它好像胶水一样把事物"粘"起来。它包含的对象和光线定义了 JPCT 的场景
    private World world = null;

    // 类似 java.awt.*中的 Color 类
    private RGBColor back = new RGBColor(50, 50, 100);

    private float touchTurn = 0;
    private float touchTurnUp = 0;

    private float xpos = -1;
    private float ypos = -1;

    // Object3D 类是一个三维对象,千万不要傻呼呼的认为它与 java.lang.Object 类似。
    // 一个 Object3D 对象作为一个实例被添加到在渲染的 World 对象中。Object3D 在 World
    // 中一次添加一个实例 ，他们可能被联系起作为孩子/父母来在他们中建立一个制度.
    // 人体模型当然也能应用在以上的规则中。他们常常不加到一个 World 实例中，而是
    // 绑定到其它对象中(人体模型或非人体模型)。有些方法 在这个类中需要一个实例
    // 添加到一个 World 实例中(用 World.addObject()方法可以实现)。
```

```
    private Object3D cube = null;

    // 每秒帧数
    private int fps = 0;

    // 光照类
    private Light sun = null;

    protected void onCreate(Bundle savedInstanceState) {
        // Logger 类中 jPCT 中一个普通的用于打印和存储消息，错误和警告的日志类。
        // 每一个 JPCT 生成的消息将被加入到这个类的队列中
        Logger.log("onCreate");
        // 如果本类对象不为 NULL,将从 Object 中所有属性装入该类
        if (master != null) {
            copy(master);
        }

        super.onCreate(savedInstanceState);

        // 实例化 GLSurfaceView
        mGLView = new GLSurfaceView(this);
        // 使用自己实现的 EGLConfigChooser,该实现必须在 setRenderer(renderer)之前
        // 如果没有 setEGLConfigChooser 方法被调用，则默认情况下，视图将选择一个与当前 android.view.Surface 兼容至少 16 位深度缓冲深度 EGLConfig。
        mGLView.setEGLConfigChooser(new GLSurfaceView.EGLConfigChooser() {
            public EGLConfig chooseConfig(EGL10 egl, EGLDisplay display) {
                // Ensure that we get a 16bit framebuffer. Otherwise, we"""""""""ll fall
                // back to Pixelflinger on some device (read: Samsung I7500)
                int[] attributes = new int[] { EGL10.EGL_DEPTH_SIZE, 16,
                        EGL10.EGL_NONE };
                EGLConfig[] configs = new EGLConfig[1];
                int[] result = new int[1];
                egl.eglChooseConfig(display, attributes, configs, 1, result);
                return configs[0];
            }
        });
        // 实例化 MyRenderer
        renderer = new MyRenderer();
        // 设置 View 的渲染器，同时启动线程调用渲染，以至启动渲染
        mGLView.setRenderer(renderer);
```

```
    // 设置一个明确的视图
    setContentView(mGLView);
}

// 重写 onPause()
@Override
protected void onPause() {
    super.onPause();
    mGLView.onPause();
}

// 重写 onResume()
@Override
protected void onResume() {
    super.onResume();
    mGLView.onResume();
}

// 重写 onStop()
@Override
protected void onStop() {
    super.onStop();
}



private void copy(Object src) {
    try {
        // 打印日志
        Logger.log("Copying data from master Activity!");
        // 返回一个数组，其中包含目前这个类的的所有字段的 Filed 对象
        Field[] fs = src.getClass().getDeclaredFields();
        // 遍历 fs 数组
        for (Field f : fs) {
            // 尝试设置无障碍标志的值。标志设置为 false 将使访问检查，设置为 true，将其禁用。
            f.setAccessible(true);
            // 将取到的值全部装入当前类中
            f.set(this, f.get(src));
        }
    } catch (Exception e) {
```

```
            // 抛出运行时异常
            throw new RuntimeException(e);
        }
    }
```

继续阅读文章

——Android os 设备谎言分辨率的解决方案

- 刚才一群里的兄弟问的一问题，稍微研究下，这里一起分享：新建的 Emulator -配置为：WAGA800 其分辨率是 800*480 的设备模拟器，当我们程序中在取得其 Height 和 Width 的时候发现，总是 320*533 ，明显是系统对我们撒了谎！如下图：

下面是官方文档原文：

http://androidappdocs.appspot.com/guide/practices/screens_support.html

那么为什么系统会对其分辨率进行撒谎呢？ 其作用是什么呢？

简单的来说，在 SDK1.6 （sdk version 4）以后，Android 增加了新功能“支持多屏”，所谓这项新功能也就是为了让我们的游戏、软件能在不同的分辨率，不同机型上一样流畅、玩美运行，其作用一来减轻我们的移植工作量，二来更好的体现 Android 越来越强劲的势头。

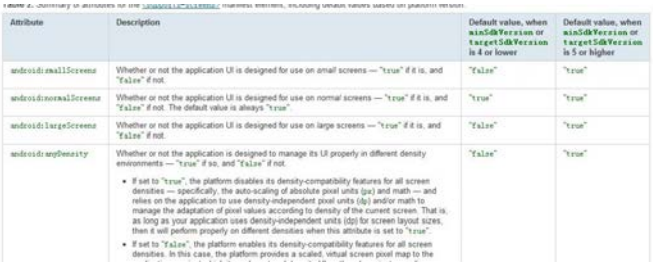
自适应效果如下图：（WVGA 高密度（左），中密度的 HVGA（中），低密度和 QVGA（右）

在不同的分辨率上想玩美的跑起来一款游戏和软件，有两种方式，一种是我们做游戏的时候都做成自适应屏幕的游戏方式，比如我们取坐标都根据屏幕的宽、高、图片的宽、高等等而不是写成死的位置坐标。 第二种那就是 Android os 在 1.6 以后的这种自适应技术；

但是有些时候显然 Android 提供的这种自适应有时候我们不需要，或者说不太适合我们的开发，（其实这也类似于现在的游戏引擎，很多人都在问我开发游戏用什么引擎，其实公司有自己的引擎。我自己写游戏不用引擎，因为没有一款游戏引擎适应所有的游戏类型开发，例如用 RPG 的引擎去做个益智连连看？是不是搞了点 - -。..当然现在市面上已经有不少的游戏开发引擎，但是使用别人的游戏引擎，对于开发来说，虽然提高了开发效率，缩短了开发周期，但是对于其扩展性不得不说很是头疼的一件事情，so~建议大家去吸收这些开源引擎的知识和技术，自己整理出一份属于自己的游戏引擎，毕竟自己的扩展起来就容易多了！而不能去一谓的去使用和强加灌输别人的思维方式到自己脑中）咳咳、回到主题上来，刚才说了，有时候我们并不想使用 Android os 提供的自适应，而是我们自己去写自适应，这样更加的灵活。

- 下面给讲解如何避开 Android os 的自适应的方法：

  先来看下官方的一段话：



Attribute	Description	Default value, when minSdkVersion or targetSdkVersion is 4 or lower	Default value, when minSdkVersion or targetSdkVersion is 5 or higher
android:smallScreens	Whether or not the application UI is designed for use on small screens — "true" if it is, and "false" if not.	"false"	"true"
android:normalScreens	Whether or not the application UI is designed for use on normal screens — "true" if it is, and "false" if not. The default value is always "true".	"true"	"true"
android:largeScreens	Whether or not the application UI is designed for use on large screens — "true" if it is, and "false" if not.	"false"	"true"
android:anyDensity	Whether or not the application is designed to manage its UI properly in different density environments — "true" if so, and "false" if not. • If set to "true", the platform disables its density-compatibility features for all screen densities — specifically, the auto-scaling of absolute pixel units (pt) and math — and relies on the application to use density-independent pixel units (dp) and/or math to manage the adaptation of pixel values according to density of the current screen. That is, as long as your application uses density-independent units (dp) for screen layout sizes, then it will perform properly on different densities when this attribute is set to "true". • If set to "false", the platform enables its density-compatibility features for all screen densities. In this case, the platform provides a scaled, virtual screen pixel map to the	"false"	"true"

这里是从官方文档中截取的一段，这里是在说，当 Android sdk 的版本是 4 或更低与版本为 5 或更高的之间的区别；

那么从这里可以得知 Android sdk 1.6（version 4）之前是不支持自适应的，那么解决的方法也就有了；

我们只需要在 AndroidMainFest 中，定义 《uses-sdk android:minSdkVersion=“4” /》 就 OK 了！

view plaincopy to clipboardprint?

······10·····20·····30·····40·····50·····60·····70·····80·····90·····100·····110·····120·····130·····140·····150

《？xml version=“1.0” encoding=“utf-8”？》

《manifest xmlns:android=“http://schemas.android.com/apk/res/android”

package=“com.desmo.testAd” android:versionCode=“1” android:versionName=“1.0”》

《application android:icon=“@drawable/icon” android:label=“@string/app_name”》

《activity android:name=“.Test” android:label=“@string/app_name”》

《intent-filter》

《action android:name=“android.intent.action.MAIN” /》

《category android:name=“android.intent.category.LAUNCHER” /》

《/intent-filter》

《/activity》

《/application》

《uses-sdk android:minSdkVersion=“4” /》

《/manifest》

然后我们看看修改后 xml 后的运行截图：

这样就正常啦， 这里呢我要给大家道个歉，大家也看到了，最近也没有更新文章，主要原因是一个回老家过年，今天刚回到公司第一天上班，第二点是由于出书的缘故，已经签下了《清华出版社》的合同，将大概在上半年完成一本关于 Android 游戏开发书籍，so~大家也体谅一下我，当然博客我肯定是还要更新的，不过速度不会跟以前一样快了。

这里还要说下，我写的这 21 篇文章，基本上对于学习游戏开发都是很实用很有用的，希望大家一定要细细的看，因为不少人问的都是写过的东西 - -;如果大家还有什么疑惑和问题可以来群里进行交流和互相学习。

OK 就写到这里，自己会尽快完成书籍，让大家早点看到。

继续阅读文章

——Android 游戏开发之详解 SQLite 存储

- 先介绍几个基本概念知识：

什么是 SQLite：

SQLite 是一款轻量级数据库，它的设计目的是嵌入式，而且它占用的资源非常少，在嵌入式设备中，只需要几百 KB！！！！！

SQLite 的特性：

轻量级

使用 SQLite 只需要带一个动态库，就可以享受它的全部功能，而且那个动态库的尺寸想当小。

独立性

SQLite 数据库的核心引擎不需要依赖第三方软件，也不需要所谓的“安装”。

隔离性

SQLite 数据库中所有的信息（比如表、视图、触发器等）都包含在一个文件夹内，方便管理和维护。

跨平台

SQLite 目前支持大部分操作系统，不至电脑操作系统更在众多的手机系统也是能够运行，比如：Android。

多语言接口

SQLite 数据库支持多语言编程接口。

安全性

SQLite 数据库通过数据库级上的独占性和共享锁来实现独立事务处理。这意味着多个进程可以在同一时间从同一数据库读取数据，但只能有一个可以写入数据。

优点：

1.能存储较多的数据。

2.能将数据库文件存放到 SD 卡中！

什么是 SQLiteDatabase？

一个 SQLiteDatabase 的实例代表了一个 SQLite 的数据库，通过 SQLiteDatabase 实例的一些方法，我们可以执行 SQL 语句，对数 据库进行增、删、查、改的操作。需要注意的是，数据库对于一个应用来说是私有的，并且在一个应用当中，数据库的名字也是惟一的。

什么是 SQLiteOpenHelper ？

根据这名字，我们可以看出这个类是一个辅助类。这个类主要生成一个数据库，并对数据库的版本进行管理。当在程

序当中调用这个类的 方法 getWritableDatabase（），或者 getReadableDatabase（）方法的时候，如果当时没有数据，那么 Android 系统就会自动生成一 个数 据库。SQLiteOpenHelper 是一个抽象类，我们通常需要继承它，并且实现里边的 3 个函数，

什么是 ContentValues 类?

ContentValues 类和 Hashmap/Hashtable 比较类似，它也是负责存储一些名值对，但是它存储的名值对当中的名是一个

String 类型，而值都是基本类型。

什么是 Cursor ？

Cursor 在 Android 当中是一个非常有用的接口，通过 Cursor 我们可以对从数据库查询出来的结果集进行随 机的读写访问。

OK，基本知识就介绍到这里，下面开始上代码：还是按照我的一贯风格，代码中该解释的地方都已经在代码中及时注释和讲解了！

顺便来张项目截图：

先给出 xml：

view plaincopy to clipboardprint?

······10·····20·····30·····40·····50·····60·····70·····80·····90·····100·····110·····120·····130·····140·····150

《? 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=“SQL 练习！（如果你使用的 SD 卡存储数据方式，为了保证正常操作，请你先点击创建一张表然后再操作）”

android:textSize=“20sp” android:textColor=“#ff0000” android:id=“@+id/tv_title” /》

《Button android:id=“@+id/sql_addOne” android:layout_width=“fill_parent”

android:layout_height=“wrap_content” android:text=“插入一条记录”》《/Button》

《Button android:id=“@+id/sql_check” android:layout_width=“fill_parent”

android:layout_height=“wrap_content” android:text=“查询数据库”》《/Button》

《Button android:id=“@+id/sql_edit” android:layout_width=“fill_parent”

android:layout_height=“wrap_content” android:text=“修改一条记录”》《/Button》

《Button android:id=“@+id/sql_deleteOne” android:layout_width=“fill_parent”

android:layout_height=“wrap_content” android:text=“删除一条记录”》《/Button》

《Button android:id=“@+id/sql_deleteTable” android:layout_width=“fill_parent”

android:layout_height=“wrap_content” android:text=“删除数据表单”》《/Button》

《Button android:id=“@+id/sql_newTable” android:layout_width=“fill_parent”

android:layout_height=“wrap_content” android:text=“新建数据表单”》《/Button》

《/LinearLayout》

- xml 中定义了我们需要练习用到的几个操作按钮，这里不多解释了，下面看 java 源码：先看我们继承的 SQLiteOpenHelper 类

view plaincopy to clipboardprint?

·····10·····20·····30·····40·····50·····60·····70·····80·····90·····100·····110·····120·····130·····140·····150

package com.himi;

import android.content.Context;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.util.Log;

/**

*

```
* @author Himi
* @解释  此类我们只需要传建一个构造函数  以及重写两个方法就 OK 啦、
*
*/
public class MySQLiteOpenHelper extends SQLiteOpenHelper {
public final static int VERSION = 1;//  版本号
public final static String TABLE_NAME = “himi”;//  表名
public final static String ID = “id”;//  后面 ContentProvider 使用
public final static String TEXT = “text”;
public static final String DATABASE_NAME = “Himi.db”;
public MySQLiteOpenHelper（Context context）  {
//  在 Android  中创建和打开一个数据库都可以使用 openOrCreateDatabase  方法来实现，
//  因为它会自动去检测是否存在这个数据库，如果存在则打开，不过不存在则创建一个数据库；
//  创建成功则返回一个  SQLiteDatabase 对象，否则抛出异常 FileNotFoundException。
//  下面是来创建一个名为“DATABASE_NAME”的数据库，并返回一个 SQLiteDatabase 对象
super（context，  DATABASE_NAME，  null，  VERSION）；
}
@Override
//  在数据库第一次生成的时候会调用这个方法，一般我们在这个方法里边生成数据库表;
public void onCreate（SQLiteDatabase db）  {
String str_sql = “CREATE TABLE ” + TABLE_NAME + “（” + ID
+ “ INTEGER PRIMARY KEY AUTOINCREMENT，” + TEXT + “ text  ）;”;
// CREATE TABLE  创建一张表  然后后面是我们的表名
//  然后表的列，第一个是 id  方便操作数据，int 类型
// PRIMARY KEY  是指主键  这是一个 int 型，用于唯一的标识一行;
// AUTOINCREMENT  表示数据库会为每条记录的 key 加一，确保记录的唯一性;
//  最后我加入一列文本  String 类型
// ----------注意：这里 str_sql 是 sql 语句，类似 dos 命令，要注意空格！
```

```
db.execSQL（str_sql）;
// execSQL（）方法是执行一句 sql 语句
// 虽然此句我们生成了一张数据库表和包含该表的 sql.himi 文件，
// 但是要注意 不是方法是创建，是传入的一句 str_sql 这句 sql 语句表示创建!!
}
@Override
public void onUpgrade（SQLiteDatabase db， int oldVersion， int newVersion） {
// 一般默认情况下，当我们插入 数据库就立即更新
// 当数据库需要升级的时候，Android 系统会主动的调用这个方法。
// 一般我们在这个方法里边删除数据表，并建立新的数据表，
// 当然是否还需要做其他的操作，完全取决于游戏需求。
Log.v（“Himi”， “onUpgrade”）;
}
}
```

我喜欢代码中立即附上解释，感觉这样代码比较让大家更容易理解和寻找，当然如果童鞋们不喜欢，可以告诉我，我改~嘿嘿~

- 下面看最重要的 MainActivity 中的代码：

view plaincopy to clipboardprint?

```
.....10.....20.....30.....40.....50.....60.....70.....80.....90.....100.....110.....120.....130.....140.....150
package com.himi;
import java.io.File;
import java.io.IOException;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
```

```
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
// ------------第三种保存方式--------《SQLite》---------
/**
* @author Himi
* @保存方式：SQLite 轻量级数据库、
* @优点： 可以将自己的数据存储到文件系统或者数据库当中， 也可以将自己的数据存
* 储到 SQLite 数据库当中，还可以存到 SD 卡中
* @注意 1：数据库对于一个游戏（一个应用）来说是私有的，并且在一个游戏当中，
* 数据库的名字也是唯一的。
* @注意 2 apk 中创建的数据库外部的进程是没有权限去读/写的，
* 我们需要把数据库文件创建到 sdcard 上可以解决类似问题。
* @注意 3 当你删除 id 靠前的数据或者全部删除数据的时候，SQLite 不会自动排序，
* 也就是说再添加数据的时候你不指定 id 那么 SQLite 默认还是在原有 id 最后添加一条新数据
* @注意 4 android 中 的 SQLite 语法区分大小写的!!!!! 这点要注意!
* String UPDATA_DATA = “UPDATE himi SET text=‘通过 SQL 语句来修改数据’ WHERE id=1”;
* 千万 不能可以写成
* String UPDATA_DATA = “updata himi set text=‘通过 SQL 语句来修改数据’ where id=1”;
*/
public class MainActivity extends Activity implements OnClickListener {
private Button btn_addOne， btn_deleteone， btn_check， btn_deleteTable，
btn_edit， btn_newTable;
private TextView tv;
private MySQLiteOpenHelper myOpenHelper;// 创建一个继承 SQLiteOpenHelper 类实例
```

```
private SQLiteDatabase mysql ;
//----------以下两个成员变量是针对在 SD 卡中存储数据库文件使用
// private File path = new File（“/sdcard/himi”）;// 创建目录
// private File f = new File（“/sdcard/himi/himi.db”）;// 创建文件
@Override
public void onCreate（Bundle savedInstanceState） {
super.onCreate（savedInstanceState）;
getWindow（）.setFlags（WindowManager.LayoutParams.FLAG_FULLSCREEN，
WindowManager.LayoutParams.FLAG_FULLSCREEN）;
this.requestWindowFeature（Window.FEATURE_NO_TITLE）;
setContentView（R.layout.main）;
tv = （TextView） findViewById（R.id.tv_title）;
btn_addOne = （Button） findViewById（R.id.sql_addOne）;
btn_check = （Button） findViewById（R.id.sql_check）;
btn_deleteone = （Button） findViewById（R.id.sql_deleteOne）;
btn_deleteTable = （Button） findViewById（R.id.sql_deleteTable）;
btn_newTable = （Button） findViewById（R.id.sql_newTable）;
btn_edit = （Button） findViewById（R.id.sql_edit）;
btn_edit.setOnClickListener（this）;
btn_addOne.setOnClickListener（this）;
btn_check.setOnClickListener（this）;
btn_deleteone.setOnClickListener（this）;
btn_deleteTable.setOnClickListener（this）;
btn_newTable.setOnClickListener（this）;
myOpenHelper = new MySQLiteOpenHelper（this）;// 实例一个数据库辅助器
//备注 1 ----如果你使用的是将数据库的文件创建在 SD 卡中，那么创建数据库 mysql 如下操作：
// if （！path.exists（）） {// 目录存在返回 false
// path.mkdirs（）;// 创建一个目录
```

```
// }
// if （！f.exists（）） {// 文件存在返回 false
// try {
// f.createNewFile（）;//创建文件
// } catch （IOException e） {
// // TODO Auto-generated catch block
// e.printStackTrace（）;
// }
// }
}
@Override
public void onClick（View v） {
try {
//备注 2----如果你使用的是将数据库的文件创建在 SD 卡中，那么创建数据库 mysql 如下操作：
// mysql = SQLiteDatabase.openOrCreateDatabase（f， null）;
//备注 3--- 如果想把数据库文件默认放在系统中，那么创建数据库 mysql 如下操作：
mysql = myOpenHelper.getWritableDatabase（）; // 实例数据库
if （v == btn_addOne） {// 添加数据
// ---------------------- 读写句柄来插入---------
// ContentValues 其实就是一个哈希表 HashMap， key 值是字段名称，
//Value 值是字段的值。然后 通过 ContentValues 的 put 方法就可以
//把数据放到 ContentValues 中，然后插入到表中去！
ContentValues cv = new ContentValues（）;
cv.put（MySQLiteOpenHelper.TEXT， “测试新的数据”）;
mysql.insert（MySQLiteOpenHelper.TABLE_NAME， null， cv）;
// inser（） 第一个参数 标识需要插入操作的表名
// 第二个参数 ：默认传 null 即可
// 第三个是插入的数据
```

```
// ---------------------- SQL 语句插入----------------
// String INSERT_DATA =
// “INSERT INTO himi （id，text） values （1， ‘通过 SQL 语句插入’）”;
// db.execSQL（INSERT_DATA）;
tv.setText（“添加数据成功！点击查看数据库查询”）;
} else if （v == btn_deleteone） {// 删除数据
// ---------------------- 读写句柄来删除
mysql.delete（“himi”， MySQLiteOpenHelper.ID + “=1”， null）;
// 第一个参数 需要操作的表名
// 第二个参数为 id+操作的下标 如果这里我们传入 null，表示全部删除
// 第三个参数默认传 null 即可
// ---------------------- SQL 语句来删除
// String DELETE_DATA = “DELETE FROM himi WHERE id=1”;
// db.execSQL（DELETE_DATA）;
tv.setText（“删除数据成功！点击查看数据库查询”）;
} else if （v == btn_check） {// 遍历数据
//备注 4------
```

继续阅读文章

——Android 提高之 SQLite 分页表格

- 本文实现并封装一个 SQL 分页表格控件，不仅支持分页还是以表格的形式展示数据。先来看看本文程序运行的动画：

这个 SQL 分页表格控件主要分为“表格区”和“分页栏”这两部分，这两部分都是基于 GridView 实现的。网上介绍 Android 上实现表格的 DEMO 一般都用 ListView。ListView 与 GridView 对比，ListView 最大的优势是格单元的大小可以自定义，可以某单元长某单元短，但是不能自适应数据表的结构；而 GridView 最大的优势就是自适应数据表的结构，但是格单元统一大小。。。对于数据表结构多变的情况，建议使用 GridView 实现表格。

本文实现的 SQL 分页表格控件有以下特点：

1.自适应数据表结构，但是格单元统一大小；

2.支持分页；

3.“表格区”有按键事件回调处理，“分页栏”有分页切换事件回调处理。

本文程序代码较多，可以到这里下载整个工程的源码：http://www.rayfile.com/files/72e78b68-f2e5-11df-8469-0015c55db73d/

items.xml 的代码如下，它是“表格区”和“分页栏”的格单元实现：

view plaincopy to clipboardprint?

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout01"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:background="#555555"
    android:layout_height="wrap_content">
    <TextView android:layout_below="@+id/ItemImage" android:text="TextView01"
       android:id="@+id/ItemText" android:bufferType="normal"
```

```
        android:singleLine="true" android:background="#000000"
        android:layout_width="fill_parent" android:gravity="center"
        android:layout_margin="1dip" android:layout_gravity="center"
        android:layout_height="wrap_content">
    </TextView>
</LinearLayout>
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout01"
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent" android:background="#555555"
 android:layout_height="wrap_content">
 <TextView android:layout_below="@+id/ItemImage" android:text="TextView01"
  android:id="@+id/ItemText" android:bufferType="normal"
  android:singleLine="true" android:background="#000000"
  android:layout_width="fill_parent" android:gravity="center"
  android:layout_margin="1dip" android:layout_gravity="center"
  android:layout_height="wrap_content">
 </TextView>
</LinearLayout>
```

main.xml 的代码如下：

view plaincopy to clipboardprint?

```
<?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" android:id="@+id/MainLinearLayout">

    <Button android:layout_height="wrap_content"
        android:layout_width="fill_parent" android:id="@+id/btnCreateDB"
        android:text="创建数据库"></Button>
    <Button android:layout_height="wrap_content"
        android:layout_width="fill_parent" android:text="插入一串实验数据" android:id="@+id/btnInsertRec"></Button>
    <Button android:layout_height="wrap_content" android:id="@+id/btnClose"
        android:text="关闭数据库" android:layout_width="fill_parent"></Button>
</LinearLayout>
<?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" android:id="@+id/MainLinearLayout">
```

```
 <Button android:layout_height="wrap_content"
  android:layout_width="fill_parent" android:id="@+id/btnCreateDB"
  android:text="创建数据库"></Button>
 <Button android:layout_height="wrap_content"
  android:layout_width="fill_parent" android:text="插入一串实验数据" android:id="@+id/btnInsertRec"></Button>
 <Button android:layout_height="wrap_content" android:id="@+id/btnClose"
  android:text="关闭数据库" android:layout_width="fill_parent"></Button>
</LinearLayout>
```

演示程序 testSQLite.java 的源码：

```
view plaincopy to clipboardprint?
package com.testSQLite;

import android.app.Activity;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.Toast;

public class testSQLite extends Activity {
    GVTable table;
    Button btnCreateDB, btnInsert, btnClose;
    SQLiteDatabase db;
    int id;//添加记录时的 id 累加标记，必须全局

    private static final String TABLE_NAME = "stu";
    private static final String ID = "id";
    private static final String NAME = "name";
    private static final String PHONE = "phone";
    private static final String ADDRESS = "address";
    private static final String AGE = "age";

    @Override
    public void onCreate(Bundle savedInstanceState) {
```

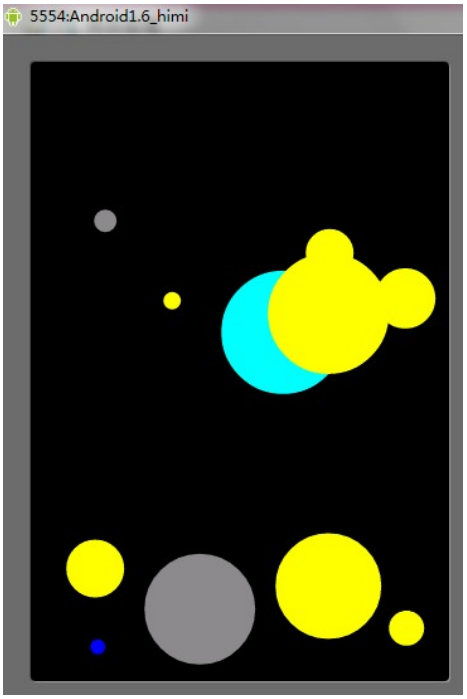
```
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
btnCreateDB = (Button) this.findViewById(R.id.btnCreateDB);
btnCreateDB.setOnClickListener(new ClickEvent());

btnInsert = (Button) this.findViewById(R.id.btnInsertRec);
btnInsert.setOnClickListener(new ClickEvent());

btnClose = (Button) this.findViewById(R.id.btnClose);
btnClose.setOnClickListener(new ClickEvent());

table=new GVTable(this);
table.gvSetTableRowCount(8);//设置每个分页的 ROW 总数
LinearLayout ly = (LinearLayout) findViewById(R.id.MainLinearLayout);

table.setTableOnClickListener(new GVTable.OnTableClickListener() {
   @Override
   public void onTableClickListener(int x,int y,Cursor c) {
      c.moveToPosition(y);
      String str=c.getString(x)+"  位置:("+String.valueOf(x)+","+String.valueOf(y)+")";
      Toast.makeText(testSQLite.this, str, 1000).show();
   }
```

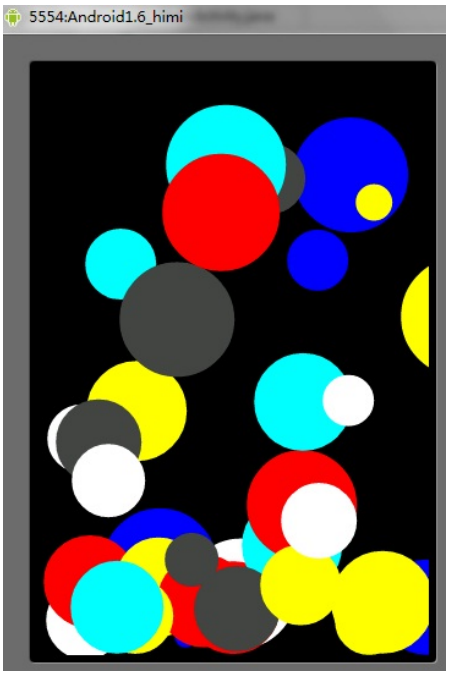
继续阅读文章

——Android 游戏开发之重力系统开发

- 在重力传感器中，虽然我也实现了一个圆形会根据手机反转的角度而拥有不同的速度，但是其内置加速度算法都是 Android os 封装好的，而今天我们要讲的重力系统就是去模拟这个加速度，从而让一个自由落体的圆形，感觉跟现实中的皮球一样有质有量！下落的时候速度加快，反弹起来以后速度慢慢减下来~

  OK，先上两张截图，然后简单介绍之后进行讲解：





Demo：简介：（咳咳、玩的有点 H，狂点按钮搞的满屏都是 - -）

当你点击模拟器任意按键的时候会随机在屏幕上生成一个随机大小、随即颜色、随即位置、不停闪烁的一个圆形，并且圆形都拥有重力，在做自由落体，当圆形触到屏幕底部的时候会反弹，并且反弹的高度一次比一次低！

这个实例中，为了好看，我没有让圆形最终慢到停下来，会一直在一个高度进行的反弹，下落；

还有一点：对于圆形当从一个高度自由落体的时候可能它在 X 坐标系上没有发生改变，当然这是在我们代码中，属于理想状态，因为现实生活中，一般 X/Y 坐标系都会有变动，在此 Demo 中，我主要把垂直下落并且反弹的功能做出来了，关于水平的加速度我没做，第一是因为和垂直的处理思路基本一致，第二点我没时间 - -。..

好了 不废话！先介绍一下我自定义的圆形类：

MyArc.java

view plaincopy to clipboardprint?

······10·····20·····30·····40·····50·····60·····70·····80·····90·····100·····110·····120·····130·····140·····150

```
package com.himi;

import java.util.Random;

import android.graphics.Canvas;

import android.graphics.Color;

import android.graphics.Paint;

import android.graphics.RectF;

/**

* @author Himi

* @自定义圆形类

*/

public class MyArc {

private int arc_x， arc_y， arc_r;//圆形的 X，Y 坐标和半径

private float speed_x = 1.2f， speed_y = 1.2f;//小球的 x、y 的速度

private float vertical_speed;//加速度

private float horizontal_speed;//水平加速度，大家自己试着添加吧

private final float ACC = 0.135f;//为了模拟加速度的偏移值

private final float RECESSION = 0.2f;//每次弹起的衰退系数

private boolean isDown = true;//是否处于下落 状态

private Random ran;//随即数库

/**
```

```
* @定义圆形的构造函数
* @param x 圆形 X 坐标
* @param y 圆形 Y 坐标
* @param r 圆形半径
*/
public MyArc（int x， int y， int r） {
ran = new Random（）;
this.arc_x = x;
this.arc_y = y;
this.arc_r = r;
}
public void drawMyArc（Canvas canvas， Paint paint） {//每个圆形都应该拥有一套绘画方法
paint.setColor（getRandomColor（））;//不断的获取随即颜色，对圆形进行填充（实现圆形闪烁效果）
canvas.drawArc（new RectF（arc_x + speed_x， arc_y + speed_y， arc_x + 2 *
arc_r + speed_x， arc_y + 2 * arc_r + speed_y）， 0， 360， true， paint）;
}
/**
* @return
* @返回一个随即颜色
*/
public int getRandomColor（） {
int ran_color = ran.nextInt（8）;
int temp_color = 0;
switch （ran_color） {
case 0：
temp_color = Color.WHITE;
break;
case 1：
```

```
temp_color = Color.BLUE;
break;
case 2：
temp_color = Color.CYAN;
break;
case 3：
temp_color = Color.DKGRAY;
break;
case 4：
temp_color = Color.RED;
break;
case 6：
temp_color = Color.GREEN;
case 7：
temp_color = Color.GRAY;
case 8：
temp_color = Color.YELLOW;
break;
}
return temp_color;
}
/**
* 圆形的逻辑
*/
public void logic（） {//每个圆形都应该拥有一套逻辑
if （isDown） {//圆形下落逻辑
/*--备注 1-*/speed_y += vertical_speed;//圆形的 Y 轴速度加上加速度
int count = （int） vertical_speed++;
```

```
//这里拿另外一个变量记下当前速度偏移量
//如果下面的 for （int i = 0; i 《 vertical_speed++; i++） {}这样就就死循环了 - -
for （int i = 0; i 《 count; i++） {//备注 1
/*--备注 2-*/ vertical_speed += ACC;
}
} else {//圆形反弹逻辑
speed_y -= vertical_speed;
int count = （int） vertical_speed--;
for （int i = 0; i 《 count; i++） {
vertical_speed -= ACC;
}
}
if （isCollision（）） {
isDown = ！ isDown;//当发生碰撞说明圆形的方向要改变一下了！
vertical_speed -= vertical_speed * RECESSION;//每次碰撞都会衰减反弹的加速度
}
}
/**
* 圆形与屏幕底部的碰撞
* @return
* @返回 true 发生碰撞
*/
public boolean isCollision（） {
return arc_y + 2 * arc_r + speed_y 》= MySurfaceViee.screenH;
}
}
```

代码比较简单主要讲解下几个备注：

备注 1：

估计有些同学看到这里有点小晕，我解释下，大家都知道自由落体的时候，速度是越来越快的，这是受到加速度的影响，所以这里我们对原有的圆形 y 速度基础上再加上加速度！

备注 2：

虽然加速度影响了圆形原有的速度，但是我们的加速度也不是恒定的，为了模拟真实球体的自由下落，这里我们不仅对加速度增加了偏移量 ACC，而且我们还要对其变化的规律进行模拟，让下次的加速度偏移量成倍增加！所以为什么要 for 循环的时候把加速度的值当成 for 循环的一个判定条件！

继续阅读文章

——android 电源管理

- Android 的电源管理也是很重要的一部分。比如在待机的时候关掉不用的设备，timeout 之后的屏幕和键盘背光的关闭，用户操作的时候该打开多少设备等等，这些都直接关系到产品的待机时间，以及用户体验。

framework 层主要有这两个文件：

frameworks\base\core\java\android\os\PowerManager.java

frameworks\base\services\java\com\android\server\PowerManagerService.java

其中 PowerManager.java 是提供给应用层调用的，最终的核心还是在 PowerManagerService.java。这个类的作用就是提供 PowerManager 的功能，以及整个电源管理状态机的运行。里面函数和类比较多，就从对外和对内分两块来说。

先说对外，PowerManagerService 如何来进行电源管理，那就要有外部事件的时候去通知它，这个主要是在 frameworks\base \services\java\com\android\server\WindowManagerService.java 里面。 WindowManagerService 会把用户的点击屏幕，按键等作为 user activity 事件来调用 userActivity 函数，PowerManagerService 就会在 userActivity 里面判断事件类型作出反映，是点亮屏幕提供操作，还是完全不理会，或者只亮一下就关掉。供 WindowManagerService 调用的方法还有 gotoSleep 和其他一些获取电源状态的函数比如 screenIsOn 等等。

在说对内，作为对外接口的 userActivity 方法主要是通过 setPowerState 来完成功能。把要设置的电源状态比如开关屏幕背光什么的作为参数调用 setPowerState，setPowerState 先判断下所要的状态能不能完成，比如要点亮屏幕的话但是现在屏幕被 lock 了那就不能亮了，否则就可以调用 Power.setScreenState（true）来透过 jni 跑到 driver 里面去点亮屏幕了。

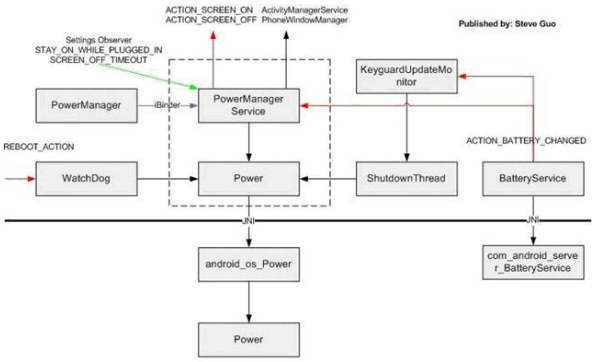
而电源的状态循环则主要是通过 Handler 来实现的。PowerManagerService 在 init 里面会启动一个 HandlerThread 一个后台消息循环来提供任务的延迟发送，就可以使用 Handler 来在定制推迟某一任务的执行时间，从而实现状态机的循环。比如 timeout，一段时间之后无操作要让屏幕变暗，然后关闭，反映在代码里如下：

userActivity 里面在调用 setPowerState 之后会用 setTimeoutLocked 来设置 timeout。然后在 setTimeoutLocked 里面会根据当前的状态来计算下一个状态以及时间，判断完再调用 mHandler.postAtTime（mTimeoutTask， when）来 post

一个 TimeoutTask。这样在 when 毫秒后就会执行 TimeoutTask。在 TimeoutTask 里面则根据设定的状态来调用 setPowerState 来改变电源状态，然后再设定新的状态，比如现在是把屏幕从亮改暗了，那就再用 setTimeoutLocked（now，SCREEN_OFF）来等下把屏幕完全关掉。如果这次已经是把屏幕关了，那这轮的 timeout 状态循环就算是结束了。

如果要定制的话，比如需求是在 timeout 屏幕关掉之后还要再关掉一些外围设备等等，那就在 TimeoutTask 里面把屏幕关掉之后再加上关闭其他设备的代码就好了。即使新的状态需求完全和原来的不一样，用 Handler 应该也不难。逻辑理清了把代码摆在合适的地方就好了。

- 总体上来说 Android 的电源管理还是比较简单的， 主要就是通过锁和定时器来切换系统的状态，使系统的功耗降至最低，整个系统的电源管理架构图如下： （注该图来自 Steve Guo）



接下来我们从 Java 应用层面， Android framework 层面， Linux 内核层面分别进行详细的讨论：

应用层的使用：

Android 提供了现成 android.os.PowerManager 类，该类用于控制设备的电源状态的切换。

该类对外有三个接口函数：

void goToSleep（long time）; //强制设备进入 Sleep 状态

Note：

尝试在应用层调用该函数，却不能成功，出现的错误好象是权限不够， 但在 Framework 下面的 Service 里调用是可以的。

newWakeLock（int flags， String tag）;//取得相应层次的锁

flags 参数说明：

PARTIAL_WAKE_LOCK： Screen off， keyboard light off

SCREEN_DIM_WAKE_LOCK： screen dim， keyboard light off

SCREEN_BRIGHT_WAKE_LOCK： screen bright， keyboard light off

FULL_WAKE_LOCK： screen bright， keyboard bright

ACQUIRE_CAUSES_WAKEUP： 一旦有请求锁时强制打开 Screen 和 keyboard light

ON_AFTER_RELEASE： 在释放锁时 reset activity timer

- Note：

如果申请了 partial wakelock，那么即使按 Power 键，系统也不会进 Sleep，如 Music 播放时

如果申请了其它的 wakelocks，按 Power 键，系统还是会进 Sleep

void userActivity（long when， boolean noChangeLights）;//User activity 事件发生，设备会被切换到 Full on 的状态，同时 Reset Screen off timer.

Sample code：

PowerManager pm = （PowerManager）getSystemService（Context.POWER_SERVICE）；

PowerManager.WakeLock wl = pm.newWakeLock （PowerManager.SCREEN_DIM_WAKE_LOCK， “My Tag”）；

wl.acquire（）；

……。

wl.release（）；

Note：

1. 在使用以上函数的应用程序中，必须在其 Manifest.xml 文件中加入下面的权限：

《uses-permission android:name=“android.permission.WAKE_LOCK” /》

《uses-permission android:name=“android.permission.DEVICE_POWER” /》

2. 所有的锁必须成对的使用，如果申请了而没有及时释放会造成系统故障。如申请了 partial wakelock，而没有及时释放，那系统就永远进不了 Sleep 模式。

Android Framework 层面：

其主要代码文件如下：

frameworks\base\core\java\android\os\PowerManager.java

frameworks\base\services\java\com\android\server\PowerManagerService.java

frameworks\base\core\java\android\os\Power.java

frameworks\base\core\jni\android_os_power.cpp

hardware\libhardware\power\power.c

其中 PowerManagerService.java 是核心， Power.java 提供底层的函数接口，与 JNI 层进行交互， JNI 层的代码主要在文件 android_os_Power.cpp 中，与 Linux kernel 交互是通过 Power.c 来实现的， Andriod 跟 Kernel 的交互主要是通过 sys 文件的方式来实现的，具体请参考 Kernel 层的介绍。

这一层的功能相对比较复杂，比如系统状态的切换，背光的调节及开关，Wake Lock 的申请和释放等等，但这一层跟硬件平台无关，而且由 Google 负责维护，问题相对会少一些，有兴趣的朋友可以自己查看相关的代码。

Kernel 层：

其主要代码在下列位置：

drivers/android/power.c

其对 Kernel 提供的接口函数有

EXPORT_SYMBOL（android_init_suspend_lock）; //初始化 Suspend lock，在使用前必须做初始化

EXPORT_SYMBOL（android_uninit_suspend_lock）; //释放 suspend lock 相关的资源

EXPORT_SYMBOL（android_lock_suspend）; //申请 lock，必须调用相应的 unlock 来释放它

EXPORT_SYMBOL（android_lock_suspend_auto_expire）;//申请 partial wakelock， 定时时间到后会自动释放

EXPORT_SYMBOL（android_unlock_suspend）; //释放 lock

EXPORT_SYMBOL（android_power_wakeup）; //唤醒系统到 on

EXPORT_SYMBOL（android_register_early_suspend）; //注册 early suspend 的驱动

EXPORT_SYMBOL（android_unregister_early_suspend）; //取消已经注册的 early suspend 的驱动

提供给 Android Framework 层的 proc 文件如下：

“/sys/android_power/acquire_partial_wake_lock” //申请 partial wake lock

“/sys/android_power/acquire_full_wake_lock” //申请 full wake lock

“/sys/android_power/release_wake_lock” //释放相应的 wake lock

“/sys/android_power/request_state” //请求改变系统状态，进 standby 和回到 wakeup 两种状态

“/sys/android_power/state” //指示当前系统的状态

继续阅读文章

——如何打造安全的 Android 产品？

- Android 开源多媒体操作系统已成为智能手机、上网本和其他众多新兴电子消费产品的重要选择。与此同时，这些设备的安全问题——通信、数字版权管理（DRM） 和金融交易保密变得越来越重要。功能与安全通常不能兼得，Android® 就是高端产品存在安全漏洞的一个很好例证。本文将介绍如何利用 ARM® TrustZoneTM 大幅提升 Android 产品的安全标准，满足市场日益增加的应用安全需求。

Android 的安全：历史回顾

早在 2008 年 9 月第一款 A ndroidTM 智能手机发布时，Google 就高调宣称：与其他智能手机操作系统相比，Android 安全架构采用了重大创新。Android 官网的原文是：“Android 安全架构的设计要点是，默认情况下，任何应用程序均无权执行会对其他应用程序、操作系统或者用户产生不利影响的操作。”

就在该款手机发布后几天，其 Web 浏览器软件中便发现了一个广为人知的严重漏洞。Google 工程师在没有认真了解内容和来源的情况下，整体采用了数百万条开源代码，其中包括网络浏览器。随后在 2008 年 11 月，黑客又发现了一个更严重的操作系统漏洞——可在手机上安装任意程序的方法，让 Google 也不禁哀叹：“为确保 Andr oid 的安全，我们确实下足了功夫，但毫无疑问，这的确是一个很大的程序缺陷，我们认为这是一个重大的安全问题，因为设备的根存取权限打破了我们的应用程序沙盒。”

如果说 Google、微软、苹果和 IBM 的智能手机技术为我们的社会进步做出了巨大贡献，恐怕一点也不为过。Android 无疑是出色的多媒体平台，各种眼花缭乱的功能和开源开发环境对技术创新和提高生产力有很大帮助。

然而，Android 和其他通用多媒体操作系统（最近 iOS 遭黑客攻击表明目前还没有哪个智能手机平台可以幸免）无法为用户、企业和重要基础设施供应商提供足够的安全保障，也使他们缺乏足够的信心，不敢轻易在智能手机应用安全关键型功能。更为严重的是，服务供应商可能已将智能手机列为高安全级别服务对象，并没有真正意识到潜在的风险。

策略与方法

既然你的技术无法抵御全球日益凶猛的专业黑客攻击，我们为何要将自己的数字身份和金钱托付给你？我们希望智能手机能够实现安全的个人信息保护、移动付费、汽车远程控制和企业重要信息处理。但现在缺少的是值得信任的安全平台。

TrustZone 简介

对于上述难题，倒是有一套技术解决方案，这就是现代移动微处理器中一项经常被忽略且被严重低估的技术：ARM TrustZone。TrustZone 可以实现专业的硬件系统虚拟化。

TrustZone 提供了两个区域：“正常”区域和“信任”或“安全”区域。使用 Tru st Zo n e 时，多媒体操作系统（即用户通常可看到的系统）在正常区域运行，而安全关键型软件则在信任区域运行（图 1）。安全区域管理员权限软件可以访问正常区域，但却不允许逆向访问。因此，正常区域实际上充当了虚拟机的角色，由信任区域中运行的系统管理程序进行控制，不过，与 Intel V T 等其他硬件虚拟化技术不同，如果不运行 Trust Zone，正常区域客户操作系统运行时不会额外占用资源。因此，Trust Zone 解决了在低功耗器件中采用系统虚拟化技术的性能障碍（或者说是最大障碍）。

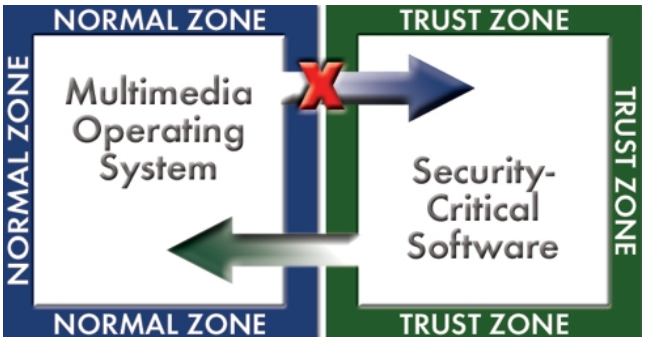


图 1：TrustZone 硬件虚拟化

TrustZone 现代 ARM 应用处理器内核（包括 A R M1176、CortexTM-A5、CortexTM-A8 和 Cortex-A9）中固有的一项功能。不过，值得注意的是，并非所有使用这些内核的 S o C 都完全兼容 Tr u st Zo n e。芯片供应商必须允许对储存器进行安全分区，并提供全面的 I/O 外设中断处理。此外，芯片供应商还必须为第三方可信任操作系统和应用程序开放安全区。目前飞思卡尔的 i.MX53（Cor texTM-A8）和德州仪器的 OMAP 4430（CortexTM-A9）移动芯片都能支持 TrustZone。

多年来，Green Hills Software 公司与这些芯片供应商及其他支持 TrustZone 处理器的主流供应商能力合作，提供功能全面并且获得安全认证的操作系统和软件开发工具（SDK），为移动设备开发和安装可信任开源应用程序提供支持。

可信任软件可能包含加密算法、网络安全协议（如 S S L/ T L S）和密钥材料、数字版权管理（DRM）软件、用于可信任路径身份输入的虚拟键盘、移动支付子系统、电子身份数据，以及服务提供商、手机制造商和/或移动 SoC 供应商可能认为值得为之提供安全保护的其他任何资料。

除了提高安全性能外，针对移动设备的某些功能需要金融和其他重要行业的认证授权，TrustZone 还能降低这些设备的开发成本，缩短产品上市时间。利用 TrustZone，银行（或认证机构）就可以将认证限制在信任区域，从而避免多媒体操作环境进行认证的复杂性（如果认证工作切实可行）。

经过认证的安全区域操作系统可以进一步降低成本和减少认证时间，原因有两个：第一，经过认证的操作系统已经得到信任，并且认证机构可以获取其所有设计和测试工件，这样就无需对安全区域操作环境进行认证，从而节省了时间和成本。第二，由于安全区域是一个完整的逻辑 A R M 内核，操作系统可以使用其内存管理单元（M M U）分区功能将安全区域进一步划分为元区域（图 2）。例如，银行可能要求对用于银行交易信息验证和加密的加密元区域进行认证，但是银行不会关心对多媒体 DRM 元区域的认证，尽管多媒体 DRM 元区域对整个设备来说至关重要，但由于它不用于银行交易，因此，安全操作系统只要保证其不产生干扰就行了。

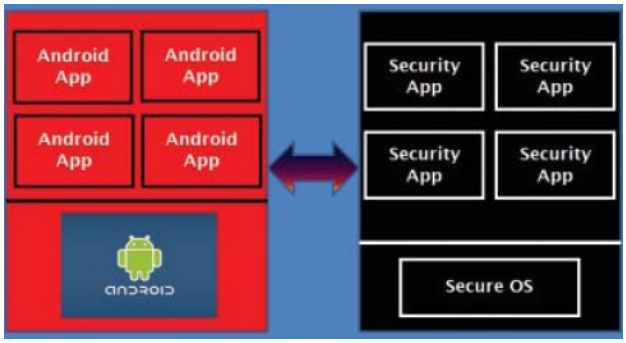


图 2：利用 TrustZone 中的元区域（meta-zone）大幅减少认证时间和成本

继续阅读文章

电子工程：

功率设计
可编程逻辑
IC 设计
MCU/控制技术
缓冲/存储技术
放大/调整/转换
封装/测试
嵌入式设计
数字信号处理
传感技术
RF/无线
光电/显示
网络/协议
EMC/EMI/ESD 设计

----------推荐热门电子书----------

----------推荐热门专题----------