

Android简介

课程目标

- 简要介绍Android平台、Android SDK
 - 特性
 - 架构
 - 底层实现

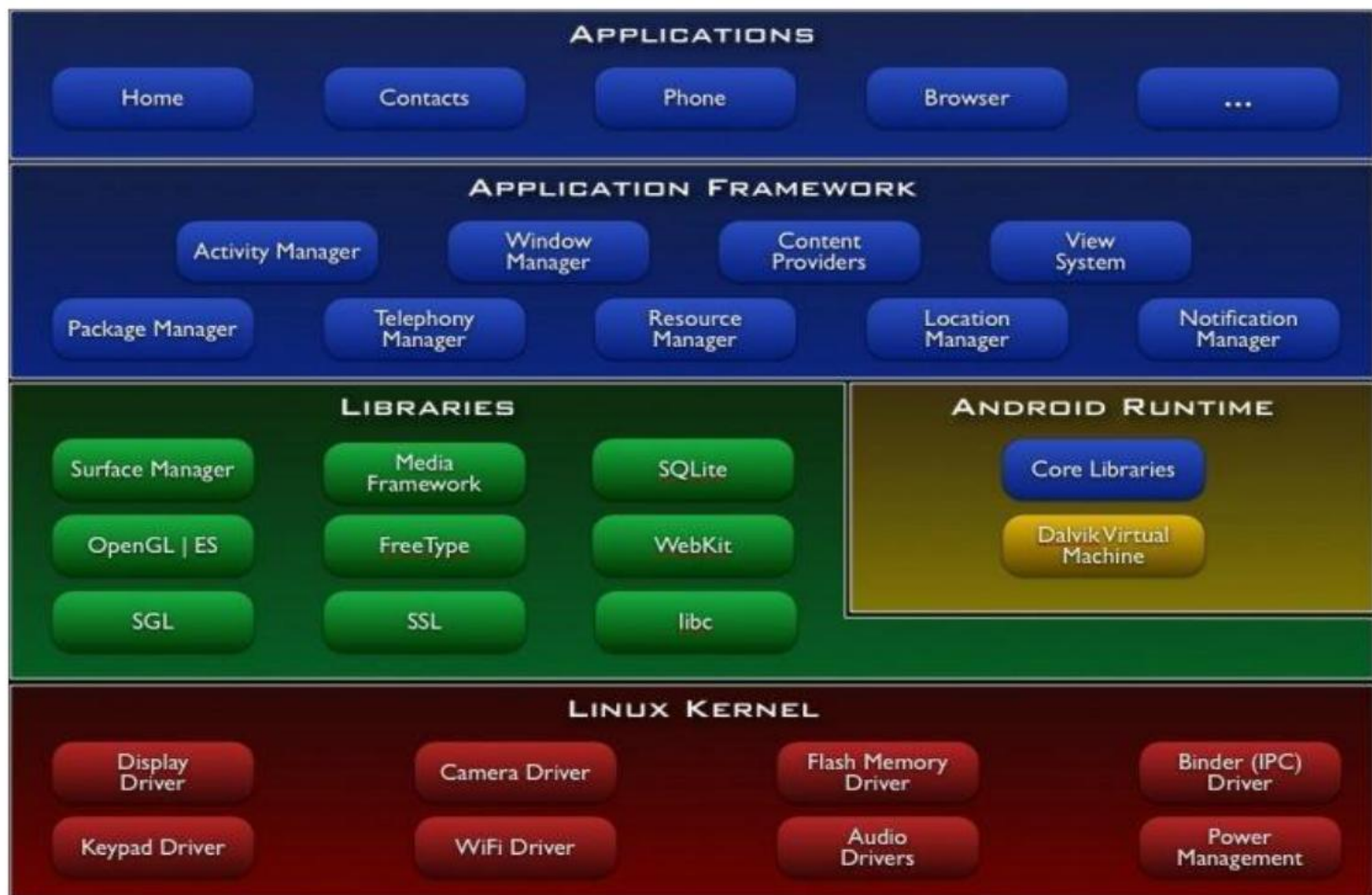
什么是Android ?

- **Android 是一个专门针对移动设备的软件集**
一个操作系统，中间件和一些重要的应用程序
- **Android SDK 提供了在Android平台上使用Java语言进行Android应用开发必须的工具和API接口。**

Android特性

- 应用程序框架 支持组件的重用与替换
- Dalvik 虚拟机 专为移动设备优化
- 集成的浏览器 基于开源的[WebKit](#) 引擎
- 优化的图形库 包括定制的2D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、视频和静态图像格式（如 MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF）
- GSM 电话技术（依赖于硬件）
- 蓝牙Bluetooth, EDGE, 3G, 和 WiFi（依赖于硬件）
- 照相机，GPS，指南针，和加速度计（accelerometer）（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

Android架构



应用程序

- **Android会同一系列核心应用程序包一起发布，该应用程序包包括。所有的应用程序都是使用JAVA语言编写的。**
 - **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**

应用程序框架

- 开发人员也可以完全访问核心应用程序所使用的API框架。该应用程序的架构设计简化了组件的重用；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）。同样，该应用程序重用机制也使用户可以方便的替换程序组件。
- 隐藏在每个应用后面的是一系列的服务和系统, 其中包括；
- 丰富而又可扩展的视图（[Views](#)），可以用来构建应用程序，它包括列表（lists），网格（grids），文本框（text boxes），按钮（buttons），甚至可嵌入的web浏览器。
- 内容提供器（[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。

应用程序框架

- 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，和布局文件 (layout files)。
- 通知管理器 ([Notification Manager](#)) 使得应用程序可以在状态栏中显示自定义的提示信息。
- 活动管理器 ([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。

Android平台程序库

- Android 包含一些C/C++库，这些库能被Android系统中不同的组件使用。它们通过 Android 应用程序框架为开发者提供服务。以下是一些核心库：
- 系统 C 库 - 一个从 BSD 继承来的标准 C 系统函数库（libc），它是专门为基于 embedded linux 的设备定制的。
- 媒体库 - 基于 PacketVideo OpenCORE；该库支持多种常用的音频、视频格式回放和录制，同时支持静态图像文件。编码格式包括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了2D和3D图层的无缝融合。
- LibWebCore - 一个最新的web浏览器引擎用，支持Android浏览器和一个可嵌入的web视图。
- SGL - 底层的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）和矢量（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被设计成一个设备可以同时高效地运行多个虚拟系统。Dalvik虚拟机执行（.dex）的Dalvik可执行文件，该格式文件针对小内存使用做了优化。同时虚拟机是基于寄存器的，所有的类都经由JAVA编译器编译，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内核的一些功能，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栈之间的抽象层。

Android SDK核心包

- **android.util**
包含有许多底层的工具类，例如专用容器类，XML工具等等。
- **android.os**
提供基本的操作系统服务、消息传递和进程间通讯。
- **android.graphics**
核心图形绘制包。
- **android.text, android.text.method, android.text.style, and android.text.util**
提供丰富的文本处理工具、富文本支持以及输入法等等。
- **android.database**
包含数据库相关的底层API。
- **android.content**
提供大量的访问存储在设备上的数据的服务：安装在设备上的应用及相关资源，用于持续的动态数据的内容提供器（content provider）。
- **android.view**
核心用户界面框架。
- **android.widget**
提供构建view包的标准用户界面元素（列表、按钮、布局管理器等等）。
- **android.app**
提供高层应用模型，通过活动实现。

Android SDK附加包

- [android.provider](#)

包括了平台内的各个内容提供器 (content providers) 的定义。

- [android.telephony](#)

提供与设备的电话协议栈交互的API。

- [android.webkit](#)

包含各种用于web内容的API。

- ...

课程内容回顾

- 简要介绍Android平台、Android SDK
 - 特性
 - 架构
 - 底层实现

课程目标

- **安装Android SDK**
- **设置开发环境**
 - **JDK** **6.0u14**
 - **Eclipse** **3.5(Galileo)**
 - **Android SDK** **1.5**
 - **ADT** **0.9.1**

系统和软件需求

- 用Android sdk的代码和工具开发Android应用程序,你需要适合的开发电脑以及开发环境,如下描述:
- 所需操作系统:
- Windows XP 或 Vista
- Mac OS X 10.4.8 或更高版本 (仅支持x86)
- Linux (Linux Ubuntu Dapper Drake 版本已测试)
- 所需开发环境:
- Eclipse IDE
 - [Eclipse](#) 3.3 (Europa), 3.4 (Ganymede) 3.5(Galileo)
 - Eclipse [JDT](#) 插件 (大多数的 Eclipse IDE 包包含)
 - [WST](#) (可选,但Android编辑器的功能需要,它被包含在[most Eclipse IDE packages](#) 中)
 - [JDK 5 或 JDK 6](#) (只有JRE是不够的)
 - [Android Development Tools 插件](#) (可选)
 - 不兼容GNU Java编译器(gcj)
- 其他开发环境或 IDE
 - [JDK 5 或 JDK 6](#) (仅有JRE不够)
 - [Apache Ant](#) 1.6.5 或更高版本 (Linux 和 Mac环境下), 1.7 或更高版本 (Windows环境下)
 - 不兼容GNU Java编译器(gcj)

安装SDK

- 下载完SDK后，把.zip文件解压到你电脑上合适位置. 默认情况下, SDK文件被解压到 `android_sdk_<platform>_<release>_<build>` 文件夹. 这个文件夹包含 `tools/`, `samples/`等.
- 请注意系统里SDK解压后的文件夹的名字和位置 — 当你安装Android插件和使用SDK工具时，你将需要引用这个文件夹。
- 你可以添加 SDK tools 的文件夹路径到你的环境变量中. 如上所述, `tools/`文件夹位于SDK文件夹中。
- Linux环境下,修改`~/.bash_profile`或者`~/.bashrc`文件. 找到环境变量设置的地方，加入`tools/`的绝对路径。如果找不到该设置，你需要新添加一行：
- `export PATH=${PATH}:<your_sdk_dir>/tools`
- Mac环境下,在你的home文件夹里面查找`.bash_profile`，然后和linux一样处理。如果之前没有`.bash_profile`文件，你可以创建一个新的。
- Windows环境下, 右击我的电脑, 并选择属性. 在标签页高级, 点击环境变量, 当对话框出现, 在系统变量栏目里双击路径 (Path) .并添加`tools/` 文件夹的完整路径.
- 添加 `tools` 到你的环境变量里，这样你可以运行Android Debug Bridge (adb)和其他`tools`下命令，而不需要输入完整路径名。需要说明的是，如果你升级你的SDK，需要将你的相应环境变量更新到新的位置。

Eclipse插件 (ADT) 安装

- 使用 Eclipse IDE作为开发Android应用的环境，你可以安装支持Android工程和工具的通用插件Android Development Tools (ADT). ADT插件包含强大的扩张，使得创建，运行和调试Android 更快速，更简单。
- 如果你不使用 Eclipse IDE，就不必下载和安装ADT插件
- 下载和安装ADT插件，请按照你们各自Eclipse版本步骤安装。
- Eclipse 3.3 (Europa)Eclipse 3.4 (Ganymede) 开始Eclipse，然后选择 Help > Software Updates > Find and Install....
- 对话框出现后，选择 Search for new features to install 点击 Next.
- 点击New Remote Site.
- 在对话框中，输入远程站点的名字（如Android Plugin），输入站点如下：<https://dl-ssl.google.com/android/eclipse/> 点击OK.
- 你可以看到新的站点添加到搜索列表中(并检查)，点击Finish.
- 在下面的搜索结果对话框，选择复选框 Android Plugin > Developer Tools. 它将会检查特性: "Android Developer Tools", 和 "Android Editors". Android 编辑器的特性是可选的，但我们推荐安装它，如果你选择安装，需要前面提到的WST插件。点击 Next.
- 阅读许可协议，然后选择接受许可协议，点击 Next.

- 点击 Finish.
- ADT插件没有签名，你可以点击 “Install All” 来安装所有东西。
- 重启 Eclipse.
- 启动 Eclipse, 选择 Help > Software Updates....
- 在出现的对话框里，点击标签页 Available Software.
- 点击 Add Site...
- 输入下面的地址: <https://dl-ssl.google.com/android/eclipse/> 点击 OK.
- 返回可用软件的视图，你会看到这个插件. 选择下一步到 *Developer Tools* 并点击 Install...
- 在接下来的安装窗口，选中 "Android Developer Tools" 和 "Android Editors". Android编辑器特性是可选的，但是我们推荐安装它，如果你选择安装，需要前面提到的WST插件。点击 Finish.
- 重启 Eclipse.
- 重启之后, update your Eclipse preferences 指向SDK文件夹:
- 选择 Window > Preferences... 来打开属性面板。(Mac OS X: Eclipse > Preferences)
- 从左边面板选择 Android。
- 在主界面上定位SDK 点击 Browse... 然后定位 SDK文件夹。
- 点击 Apply, 然后点击 OK。

ADT安装疑难解答

- [下载ADT压缩文件](#) (不解压).
- 按照默认安装的第一部和第二步(如上).
- 在 Eclipse 3.3, 点击New Archive Site....
In Eclipse 3.4, click Add Site..., 然后点击Archive...
- 浏览和选择已经下载的压缩文件。
- 从第五步开始完成上面剩下的流程。
- 更新你的插件，你必须按照这些步骤代替默认的更行说明。

课程内容回顾

- **安装Android SDK**
- **设置开发环境**
 - **JDK**
 - **Eclipse**
 - **Android SDK**
 - **ADT**

在eclipse上开发Android应用程序

- 在用eclipse IDE开发android应用程序之前，你首先要创建一个Android工程，并且建立一个启动配置，在此之后你才可以开始编写，运行，以及调试你的应用程序。

创建一个Android工程

- ADT提供了一个新的工程向导，你可以快速的创建一个新的工程或者在现有代码上创建工程。创建工程的步骤如下：
- 选择File > New > Project 选择 Android > Android Project, 然后按下 Next
- 选择项目内容:
 - 选择 Create new project in workspace，为编码创建一个全新的工程。输入工程名称（project name），基础软件包的名称（the base package name），以及Activity 类的名称。以创建stub .java文件等文件和程序名字。
 - 选择Create project from existing source，为已有代码创建一个工程。如果你想编译运行SDK中提供的示例程序，可以使用这个选项。示例程序的存放在SDK的samples/目录下。浏览包含已有代码的目录，点击ok，如果目录中包含有可用的android manifest 文件，ADT将为你填写合适的软件包，activity，和应用程序名称。
- 按下Finish.
- ADT插件会根据你的工程类型创建合适的文件和文件夹，如下:
- src/ 包含stub .java Activity文件的文件夹.
- res/ 资源文件夹.
- AndroidManifest.xml 工程清单.

创建一个启动项

- 在eclipse上运行调试应用程序之前，你必须为它创建一个启动项。启动项指定哪个工程将被启动，哪个activity开始工作，以及使用哪些模拟器选项等。
- 按照以下步骤为Eclipse版本的应用程序创建合适的启动项：
- 打开启动项管理工具。
 - 在 Eclipse 3.3 (Europa)的版本中，酌情选择 Run > Open Run Dialog... or Run > Open Debug Dialog... 。
 - 在Eclipse 3.4 (Ganymede)版本中，酌情选择 Run > Run Configurations... or Run > Debug Configurations... 。
- 在左边的工程类型列表选择Android Application选择，双击（或者点击右键选择new），创建一个新的启动项。
- 输入启动项名称。
- 在 Android标签中，浏览要开始的工程和Activity 。
- 在Target标签中，设置想要显示的屏幕及网络属性，以及其他任何[模拟器启动选项](#)。
- 你可以在Common标签中设置更多的选项。
- 按下Apply保存启动配置，或者按下Run或Debug（ ）。

运行和调试应用程序

- 可以按照以下的说明运行和调试应用程序了。
- 从eclipse主菜单，根据情况选择Run>Run 或者 Run>Debug，开始运行或者调试活动启动项。
- 注意，这里活动启动项是在运行配置管理中最最近一次选中的那个。它不一定就是在Eclipse Navigation 面板中选择的程序（如果有的话）
- 设置和修改活动启动项，可以使用启动项管理工具。如何获得启动项管理工具可以参考[创建一个启动项](#)
- 运行或调试应用程序将触发以下动作：
- 启动模拟器，如果他还没有开始运行。
- 编译工程，如果在上次编译的基础上修改过代码，将重新编译。在模拟器上安装应用程序。
- Run选项，开始运行程序。
- Debug 在"Wait for debugger "模式下启动程序,然后打开调试窗口并将Eclipse Java调试器和程序关联。

应用程序签名（一）

- **Android系统要求所有的程序经过数字签名才能安装，如果没有可用的数字签名，系统将不许安装运行此程序。不管是模拟器还是真实设备，只要是android系统，这都适用。鉴于此原因，在设备或者是模拟器上运行调试程序之前，你必须为你的应用程序设置数字签名。**
- **理解android程序签名的重要几点：**
 - 所有的程序都必须签名，没有被签名的程序，系统将不能安装。
 - 你可使用自签署证书签署你的应用程序，必须是无凭证授权是的。
 - 系统仅仅会在安装的时候测试签名证书的有效期，如果应用程序的签名是在安装之后才到期，那么应用程序仍然可以正常启用。
 - 你可以使用标准工具-Keytool and Jarsigner-生成密钥，来签名应用程序的.apk文件。

应用程序签名（二）

- **Android SDK 工具可以帮助你在调试时给应用程序签名。ADT插件和Ant编译工具都提供了两种签名模式-debug模式和release模式**
 - **debug模式下，编译工具使用JDK中的通用程序Keytool通过已知方法和密码创建秘锁和密钥。每次编译的时候，工具使用debug密钥签名应用程序的.apk文件。因为密码是已知的，工具不需要在每次编译的时候提示你输入密锁和密钥。**
 - **测试证书期限，自创建时间起有一年的期限**
 - **当你的应用程序已经准备release了，你可以在release 模式下编译。release模式下，工具编译时不会将.apk文件签名。你需要用Keytool生成密钥和密锁，再用JDK中的Jarsigner工具给.apk文件签名。**
 - **Market受理应用一般要求30年期限**

使用ApiDemo示例应用程序

- **Android SDK包含了一套示例程序，他们验证了许多功能以及API的用法。ApiDemos软件包被提前安装在模拟器中，所以你可以启动模拟器，在主画面的应用程序抽屉里打开它。**
- **你也可以在<SDK>/samples/ApiDemos中找到源码，可用看看它，学习Demo的实现方法。**
- **如果你愿意，你还可以将ApiDemo的示例程序作为一个工程加载进来，修改并在模拟器上运行。然而，在这之前你首先要卸载之前已经安装的ApiDemos。如果你没有移除之前安装的版本而直接在开发环境中运行或修改ApiDemos,将会有安装错误。**

调试

- Android有相当广泛的一套工具帮助你调试你的应用程序：
- [DDMS](#) - 一个生动的程序，它支持端口转换（因此你可以在IDE中给你的代码下端点），支持抓取模拟器屏幕，线程和堆栈信息，以及许多其他功能。你还可以运行logcat重新获得Log信息。点击此连接查看更多信息。
- [logcat](#) - 转储系统信息，这些信息包括，模拟器抛出错误时堆栈的运行过程以及日志信息。运行logcat, 点击此连接。
- [Android Log](#) - 输出模拟器上log文件信息日志类。如果你在DDMS上运行了logcat，你可以实时阅读这些信息。在你的代码中添加logging方法的调用。使用 log类，你可以根据你想获得信息的重要程度不同调用Log.v(verbose), Log.d()(debug), Log.i()(information), Log.w()(warning) 或者 Log.e(error). 来分派log信息
Log.i("MyActivity", "MyClass.getView() — Requesting item number " + position) 你可以用logcat阅读这些信息。
- [Traceview](#) - Android可以将函数的调用情况以及调用时间保存到一个log文件中，你可以用图形阅读器Traceview查看详细内容。更多信息查看这个连接下的主题
- [Eclipse plugin](#) - Eclipse插件整合了相当数量的工具（ADB，DDMS，logcat output, 以及其它功能），点击此连接查看更多信息。
- Debug and Test Device Settings - Android揭示了很多有用的设定，例如CPU使用率和 帧速率

设备上的调试和测试设置

- **Android**允许你设置多个设定以便你测试和调试程序。获得模拟器的开发设置，可以选择Dev Tools>Development Settings。按照以下选项将打开开发设置页（或其中之一）：
- **Debug app** 选择要被调试的程序，你不需要设置这个来关联调试器，但是这个变量有两个作用：
 - 防止Android在调试的断点处长时间停留时抛出错误。
 - 允许你选择Wait for Debugger选项来暂停程序启动，直到调试器被关联上（如下介绍）
- **Wait for debugger** 阻塞程序加载直到关联上调试器。这样你可以在onCreate()中设置端点，这在调试Activity的启动进程时很重要。当你改变这个选项，任何当前运行的程序实例将被杀死。为选中此框，你必须如上面介绍的选择一个调试程序。这和在代码中添加[waitForDebugger\(\)](#)是一样的。
- **Immediately destroy activities** 告诉系统只要activity停止了就销毁它。（犹如Android必须回收内存）。这个在测试 [onSaveInstanceState\(Bundle\)](#) / [onCreate\(android.os.Bundle\)](#)代码路径时非常有用，否则将难以生效。选择这个选项可能带来很多问题，因为他们没有保存程序的状态。
- **Show screen updates** 选中这个选项时，屏幕上任何被重绘的矩形区域会闪现粉红色。这对于发现屏幕不必要的绘图很有用。
- **Show CPU usage** 在屏幕顶端显示一个CPU进度，显示CPU的使用情况。上面红色栏显示总的CPU使用率，下方绿色栏显示目前画面的CPU使用时间。注意：一旦打开此功能就不能关掉，除非重新启动模拟器。？？？
- **Show background** 没有activity屏幕显示时显示背景面板，这个通常在调试的时候才会发生。

模拟器重起后这些设置仍被记忆。

顶端调试技巧

- 快速堆栈转储
 - 从模拟器上获得堆转储，你可以登录adb shell，用"ps"命令找到你想要的进程，然后用"kill-3"，堆栈使用轨迹将显示在log文件中。
- 在模拟器屏幕上显示有用信息
 - 设备可以显示一些有用信息，例如CPU使用率，以及高亮显示重绘区域。可以在开发设定窗口打开和关闭这些功能。[Setting debug and test configurations on the emulator](#)中有详细介绍。你可以通过Dalvik Debug Monitor Service工具获得转储状态信息。请参考adb中介绍的[dumpsys and dumpstate](#)
- 获得模拟器中应用程序状态信息(dumpsys)
 - 你可以通过Dalvik Debug Monitor Service工具获得dumpsys信息。参考adb中介绍的[dumpsys and dumpstate](#)。
- 获得无线连接信息
 - 你可以通过Dalvik Debug Monitor Service工具获得无线连接信息。在Device菜单中选择"Dump radio state"

- 记录跟踪数据
 - 你可以在activity中通过调用 `android.os.Debug.startMethodTracing()` 来记录函数的调用以及其它跟踪数据。详细的参考[Running the Traceview Debugging Program](#)。
- 记录无线数据
 - 默认情况下系统不记录无线数据（数据很多）。然而，你可以用下面的命令记录无线数据：`adb shell logcat -b radio` 运行adb Android 有adb工具，他提供了许多功能，包括移动和同步文件到模拟器上，改变端口，在模拟器上运行 UNIX shell。详见[Using adb](#)。
- 获得模拟器屏幕截图
 - Dalvik Debug Monitor Server (DDMS)可以抓取模拟器屏幕截图。
- 使用调试帮助类
 - Android为方便使用提供了调试帮助类，例如[util.Log](#) 和 [Debug](#)

编译安装Android应用程序

- Android要求专门的编译工具可以正确的编译资源文件和应用程序的其他部分，因此，你必须为你的应用程序建立一个专门的编译环境。
- 专门Android编译器编译步骤包括，编译XML和其他资源文件并创建合适的输出格式。编译好的Android应用程序是一个.apk压缩文件，它含有[.dex](#)文件，资源文件，原data文件，以及其他文件。你可以通过scratch,或者源文件构造一个合适的Android工程。
- Android目前不支持的在本地代码上开发第三方应用程序。
- 比较推荐的Andriod应用程序开发方法是[use Eclipse with the Android plugin](#)，它支持编译，运行，调试Android应用程序。

移出一个Android应用程序

- 移出一个安装在模拟器上的应用程序，你需要执行adb删除.apk文件。.apk文件是在安装的时候发送到模拟器上的。使用adb shell进入设备的shell，切换到data/app目录下，用rm命令删除apk文件：
`rm your_app.apk`
- 设备应用程序管理器

课程内容回顾

- 介绍在android上开发调试应用程序
 - 创建
 - 编译
 - 运行
 - 调试

课程目标

- **Android应用中的四种组件**
 - **活动**
 - **广播接收器**
 - **服务**
 - **内容提供者**
- **XML配置文件**
AndroidManifest.xml

Android应用程序构成

- 一般情况Android应用程序是由以下四种组件构造而成的：
 - 活动
 - 广播接收器
 - 服务
 - 内容提供者
- 并不是每个Andorid应用程序都必须构建这4个组件，有些可能由这些组件的组合而成。
- XML配置文件，
 - 它用于定义应用程序中需要的组件、组件的功能及必要条件等。这个文件是必须的。

活动

- 活动是最基本的Android应用程序组件，应用程序中，一个活动通常就是一个单独的屏幕。每一个活动都被实现为一个独立的类，并且从[活动](#)基类中继承而来，活动类将会显示由[视图](#)控件组成的用户接口，并对事件做出响应。大多数的应用是由多屏幕显示组成。例如，一个文本信息的应用也许有一个显示发送消息的联系人列表屏幕，第二个屏幕用来写文本消息和选择收件人，再来一个屏幕查看消息历史或者消息设置操作等。这里每一个这样的屏幕就是一个活动，很容易实现从一个屏幕到一个新的屏幕并且完成新的活动。在某些情况下当前的屏幕也许需要向上一个屏幕动提供返回值--比如让用户从手机中挑选一张照片返回通讯录做为电话拨入者的头像。
- 当打开一个新的屏幕时，之前一个屏幕会被置为暂停状态并且压入历史堆栈中。用户可以通过回退操回到以前打开过的屏幕。我们可以选择性的移除一些没有必要保留的屏幕，因为Android会把每个从桌面打开的程序保留在堆栈中。

活动 (Intent 和 Intent Filters)

- 调用Android专有类 [Intent](#) 进行构屏幕之间的切换。Intent是描述应用想要做什么。Intent数据结构两最重要的部分是动作和动作对应的数据。典型的动作类型有：MAIN（活动的门户）、VIEW、PICK、EDIT等。而动作对应的数据则以URI的形式进行表示。例如：要查看某一个人的联系方式，你需要创建一个动作类型为VIEW的intent，以及一个表示这个人的URI。
- 与之有关系的一个类叫[IntentFilter](#)。当intent被要求做某事的时候，intent filter用于描述一个活动（或者BroadcastReceiver,看下面）能够操作哪些intent。一个活动如果要显示一个人的联系方式时，需要声明一个IntentFilter，这个IntentFilter要知道怎么去处理VIEW动作和表示一个人的URI。IntentFilter需要在AndroidManifest.xml中定义。
- 通过解析各种intent，从一个屏幕切换到另一个屏幕是很简单的。当向前导航时，活动将会调用[startActivity\(myIntent\)](#)方法。然后，系统会在所有安装的应用程序定义的IntentFilter中查找，找到最匹配myIntent的Intent对应的活动。新的活动接收到myIntent的通知后，开始运行。当start活动方法被调用将触发解析myIntent的动作，这个机制提供了两个关键好处：
 - 活动能够重复利用从其它组件中以Intent的形式产生的一个请求
 - 活动可以在任何时候被一个具有相同IntentFilter的新的活动取代

广播接收器

- 可以使用[BroadcastReceiver](#)来让你的应用对一个外部的事件做出响应。比如：当电话呼入时，数据网络可用时，或者到了晚上时。
BroadcastReceivers不能显示UI，它只能通过 [NotificationManager](#) 来通知用户这些有趣的事情发生了。BroadcastReceivers既可以在AndroidManifest.xml中注册，也可以在代码中使用[Context.registerReceiver\(\)](#)进行注册。但这些有趣的事情发生时，你的应用不必对请求调用BroadcastReceivers，系统会在需要的时候启动你的应用，并在必要情况下触发BroadcastReceivers。各种应用还可以通过使用[Context.sendBroadcast\(\)](#) 将它们自己的intent broadcasts广播给其它应用程序。

服务

- 一个**服务**是具有一段较长生命周期且没有用户界面的程序。
- 比较好的一个例子就是一个正在从播放列表中播放歌曲的媒体播放器。在一个媒体播放器的应用中，应该会有多个活动，让使用者可以选择歌曲并播放歌曲。然而，音乐重放这个功能并没有对应的活动，因为使用者当然会认为在导航到其它屏幕时音乐应该还在播放的。在这个例子中，媒体播放器这个活动会使用[Context.startService\(\)](#) 来启动一个服务，从而可以在后台保持音乐的播放。同时，系统也将保持这个服务一直执行，直到这个service运行结束。(你可以通过阅读[Life Cycle of an Android Application](#) 获取更多关于服务的介绍)。另外，我们还可以通过使用[Context.bindService\(\)](#) 方法，连接到一个服务上（如果这个服务还没有运行将启动它）。当连接到一个服务之后，我们还可以通过服务提供的接口与它进行通讯。拿媒体播放器这个例子来说，我们还可以进行暂停、重播等操作。

内容提供器

- 应用程序能够将它们的数据保存到文件、SQLite数据库中，甚至是任何有效的设备中。当你想将你的应用数据与其它的应用共享时，内容提供其将会很有用。一个内容提供器类实现了一组标准的方法，从而能够让其它的应用保存或读取此内容提供器处理的各种数据类型。更详细的内容提供器资料，可以参考附带文档中的 [Accessing Content Providers](#)。

课程内容回顾

- **Android应用中的四种组件**
 - **活动**
 - **广播接收器**
 - **服务**
 - **内容提供者**
- **XML配置文件**
AndroidManifest.xml

课程目标

- **利用Android框架和诸多工具建立自己的手机应用**
- **搭建工程、组织应用逻辑以及UI，乃至接下来的编译及运行可执行程序等等**
- **对Android应用开发中的一些极为重要的概念也会有更加深刻的理解**

记事本应用程序范例

- 从一个预先配置好的工程文件开始，该教程通过一个简单记事本应用程序完整的开发过程，并辅以贯穿始终的详尽例子，指导你如何搭建工程、组织应用逻辑以及UI，乃至接下来的编译及运行可执行程序等等。

练习前准备

- 熟悉了一些基本的Android应用概念和术语
- 集成开发环境是预装Android插件的Eclipse
- 训练所需的准备工作：
 1. 下载 [project exercises archive \(.zip\)](#)
 2. 解压至你本地的某一目录下
 3. 打开 NotepadCodeLab 文件夹
- 训练所需的准备工作：

练习

- [练习 1](#)

- 一个简单记事本应用程序的开发过程起始于此。构建一个可以添加新便笺，但不能编辑的简单便笺列表。演示ListActivity 的基本方面以及如何创建和控制菜单项，以及如何用一个SQLite数据库存储便笺内容。

- [练习 2](#)

- 为记事本应用程序添加第二个Activity。演示如何构造一个新Activity至Android 操作系统，在不同Activity之间传输数据，使用更多高级的屏幕布局，并演示如何通过调用startActivityForResult()这个 API来激活另一窗口并返回一个结果。

- [练习 3](#)

- 为应用程序添加对Event生命周期的控制，以得到应用程序在整个生命周期中的状态。

- [额外学习](#)

- 演示如何使用Eclipse的调试器及通过调试器查看Event产生后的整个生命周期。此节非必读，但仍极力推荐一读。

课程内容回顾

- 利用Android框架和诸多工具建立自己的手机应用
- 搭建工程、组织应用逻辑以及UI，乃至接下来的编译及运行可执行程序等等
- 对Android应用开发中的一些极为重要的概念也会有更加深刻的理解

课程目标

Android Activity Class :

- 类继承体系
- Activity生命周期及相应接口
- Activity启动及返回
- 持久数据保存
- 权限许可
- 进程生命周期

Activity概述

- activity是单独的，用于处理用户操作。几乎所有的activity都要和用户打交道，所以activity类创建了一个窗口，开发人员可以通过setContentView(View)接口把UI放到activity创建的窗口上，当activity指向全屏窗口时，也可以用其他方式实现：作为漂浮窗口（通过windowIsFloating的主题集合），或者嵌入到其他的activity（使用ActivityGroup）。大部分的Activity子类都需要实现以下两个接口：
 - onCreate(Bundle)接口是初始化activity的地方。在这儿通常可以调用setContentView(int)设置在资源文件中定义的UI, 使用findViewById(int) 可以获得UI中定义的窗口。
 - onPause()接口是使用者准备离开activity的地方，在这儿，任何的修改都应该被提交(通常用于ContentProvider保存数据)。
- 为了能够使用Context.startActivity()，所有的activity类都必须在AndroidManifest.xml文件中定义有相关的“activity”项。
- activity类是Android 应用生命周期的重要部分。

Activity类继承体系

- activity类处于android.app包中，继承体系如下：

`java.lang.Object`

`android.content.Context`

`android.content.ContextWrapper`

`android.view.ContextThemeWrapper`

`android.app.Activity`

- 直接继承子类：

`ActivityGroup`, `AliasActivity`, `ExpandableListActivity`, `ListActivity`

- 非直接继承子类：

`LauncherActivity`, `PreferenceActivity`, `TabActivity`

Activity生命周期

- 在系统中的Activity被一个Activity栈所管理。当一个新的Activity启动时，将被放置到栈顶，成为运行中的Activity，前一个Activity保留在栈中，不再放到前台，直到新的Activity退出为止。

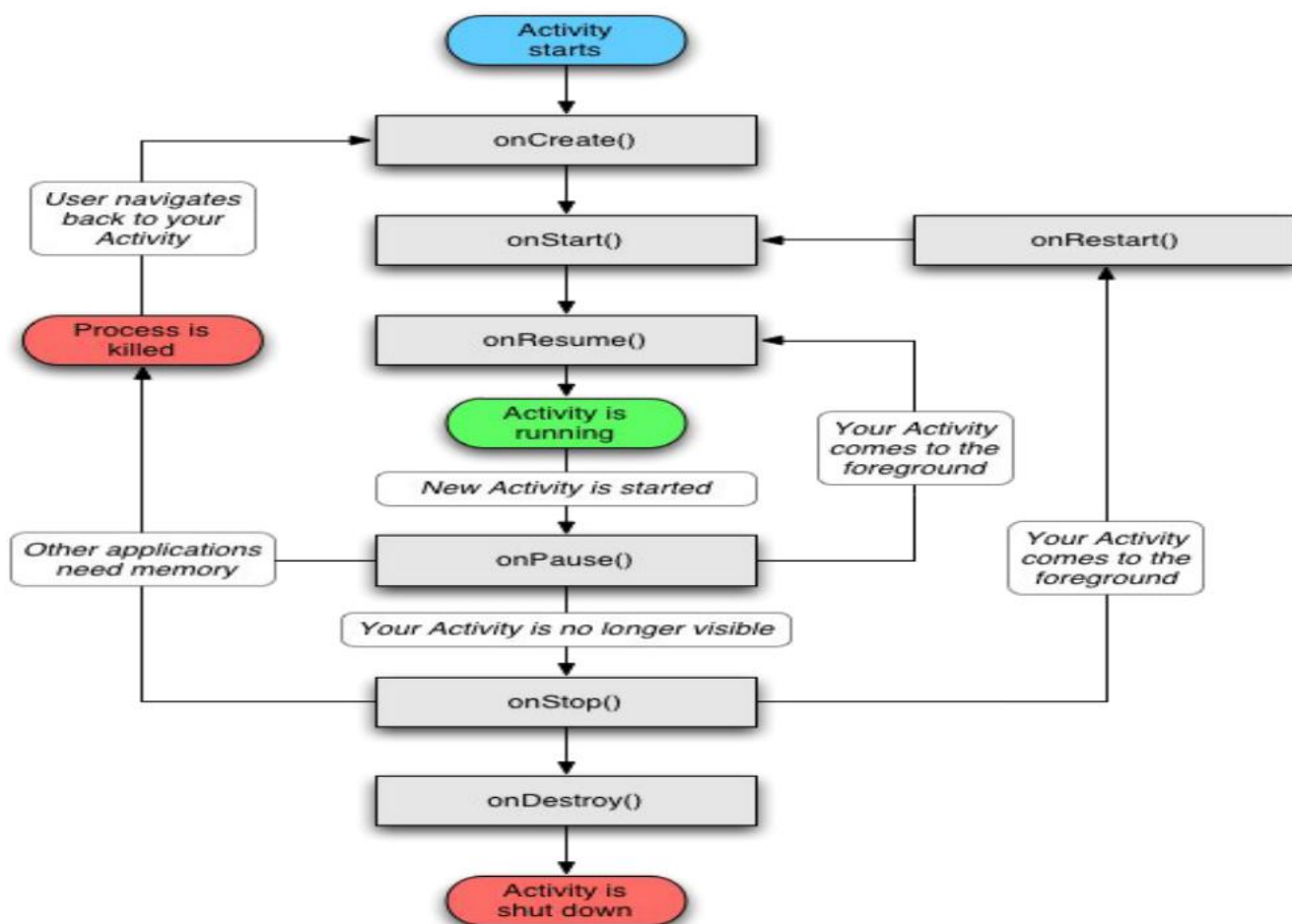
Activity的四种状态

- 1. 在屏幕的前台（Activity栈顶），叫做活动状态或者运行状态（active or running）
- 2. 如果一个Activity失去焦点，但是依然可见（一个新的非全屏的Activity 或者一个透明的Activity 被放置在栈顶），叫做暂停状态（Paused）。一个暂停状态的Activity依然保持活力（保持所有的状态，成员信息，和窗口管理器保持连接），但是在系统内存极端低下的时候将被杀掉。
- 3. 如果一个Activity被另外的Activity完全覆盖掉，叫做停止状态（Stopped）。它依然保持所有状态和成员信息，但是它不再可见，所以它的窗口被隐藏，当系统内存需要被用在其他地方的时候，Stopped的Activity将被杀掉。
- 4. 如果一个Activity是Paused或者Stopped状态，系统可以将该Activity从内存中删除，Android系统采用两种方式进行删除，要么要求该Activity结束，要么直接杀掉它的进程。当该Activity再次显示给用户时，它必须重新开始和重置前面的状态。

Activity状态转换

- **Activity的重要状态转换**，矩形框表明Activity在状态转换之间的回调接口，开发人员可以重载实现以便执行相关代码，带有颜色的椭圆形表明Activity所处的状态。

生命周期图示



三个关键循环

- **Activity有三个关键的循环：**
 1. 整个的生命周期，从onCreate(Bundle)开始到onDestroy()结束。Activity在onCreate()设置所有的“全局”状态，在onDestroy()释放所有的资源。例如：某个Activity有一个在后台运行的线程，用于从网络下载数据，则该Activity可以在onCreate()中创建线程,在onDestroy()中停止线程。
 2. 可见的生命周期，从onStart()开始到onStop()结束。在这段时间，可以看到Activity在屏幕上，尽管有可能不在前台，不能和用户交互。在这两个接口之间，需要保持显示给用户的UI数据和资源等，例如：可以在onStart中注册一个IntentReceiver来监听数据变化导致UI的变动，当不再需要显示时候，可以在onStop()中注销它。onStart()，onStop()都可以被多次调用，因为Activity随时可以在可见和隐藏之间转换。
 3. 前台的生命周期，从onResume()开始到onPause()结束。在这段时间里，该Activity处于所有Activity的最前面，和用户进行交互。Activity可以经常性地在resumed和paused状态之间切换，例如：当设备准备休眠时，当一个Activity处理结果被分发时，当一个新的Intent被分发时。所以在这些接口方法中的代码应该属于非常轻量级的。

生命周期在接口中的定义

- Activity的整个生命周期都定义在下面的接口方法中，所有方法都可以被重载。所有的Activity都需要实现 onCreate(Bundle)去初始化设置，大部分Activity需要实现 onPause()去提交更改过的数据，当前大部分的Activity也需要实现onFreeze()接口，以便恢复在onCreate(Bundle)里面设置的状态。

```
public class Activity extends ApplicationContext {  
    protected void onCreate(Bundle savedInstanceState);  
    protected void onStart();  
    protected void onRestart();  
    protected void onResume();  
    protected void onPause();  
    protected void onStop();  
    protected void onDestroy();  
}
```

Activity的启动及返回

- **Activity的创建启动**
 - **startActivity(Intent)**
- **Activity运行结果返回**
 - **startActivityForResult(Intent, int)**
 - **setResult(int)**
 - **onActivityResult(int, int, Intent)**

Activity的启动及返回（续一）

- `startActivity(Intent)`接口用于启动一个新的activity，新的activity将被放置到activity栈顶。使用一个单一的参数：描述activity的执行动作的Intent。
- 有时候我们希望在activity结束的时候获得activity的反馈结果，例如：在一个activity中，让用户在一个联系人列表中选择某一个人，在该页面结束的时候，能够返回选中的人员信息。要做到这一点，需要调用`startSubActivity(Intent, int)`接口进行启动，第二个参数为调用者的标识。结果将在`onActivityResult(int, int, String, Bundle)`中返回。
- 当一个activity退出时，可以调用`setResult(int)`返回数据给上一级的activity。它必须支持结果编码，可以是标准的`RESULT_CANCELED`, `RESULT_OK`, 或者从`RESULT_FIRST_USER`开始的自定义编码。此外，还可以返回一个字符串（通常是URL的某一项数据），也可以返回一个包含任意内容的包。所有的返回信息都会反馈到上一级的`Activity.onActivityResult()`接口，并带有上一级activity提供的原始标识号。
- 如果一个子activity失败了（如崩溃了），父activity将会接收到一个`RESULT_CANCELED`的结果编码。

(代码片段)

```
• public class MyActivity extends Activity {  
•     static final int PICK_CONTACT_REQUEST = 0;  
•     protected boolean onKeyDown(int keyCode, KeyEvent event) {  
•         if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {  
•             // When the user center presses, let them pick a contact.  
•             startActivityForResult(  
•                 new Intent(Intent.ACTION_PICK,  
•                     new Uri("content://contacts")),  
•                 PICK_CONTACT_REQUEST);  
•             return true;  
•         }  
•         return false;  
•     }  
•     protected void onActivityResult(int requestCode, int resultCode,  
•         Intent data) {  
•         if (requestCode == PICK_CONTACT_REQUEST) {  
•             if (resultCode == RESULT_OK) {  
•                 // A contact was picked. Here we will just display it  
•                 // to the user.  
•                 startActivity(new Intent(Intent.ACTION_VIEW, data));  
•             }  
•         }  
•     }  
• }
```

保存持久状态

- 一般来说，activity有两种持久状态需要处理：共享的文档类数据（典型的是使用了content provider存储在SQLite中），内在的状态（如用户的嗜好）。
- 针对内容提供的数据，google建议采用“编辑到位”的用户模型，也就是说，用户的编辑动作是立即有效的，无须进行额外的确认步骤，支持这个模式只需要简单遵守下面的两个规则：
 1. 当创建一个新的文档时，相关的数据库条目和文件都是立即创建的，例如：用户选择写一个新的电子邮件，一旦开始写，则新的电子邮件条目创建出来了。所以如果进入其他的activity，则这封电子邮件将会出现在草稿箱中。
 2. 当一个activity的onPause()接口被调用时，它应该提交用户的修改到内容提供者或者文件中。这样确保其他将要运行的activity能够看到这些变化。在整个activity的生命周期中，有很多时候都需要频繁提交数据，例如：在启动一个新的 activity之前，在结束自己的activity之前，在输入字段进行切换的时候等等。

保存持久状态（续一）

- 这个模型是为了防止用户在activity之间浏览时的数据丢失，允许系统在activity停止后的任何时间里安全地杀掉activity（因为系统资源要用在其他地方）。注意：这样意味着在activity里按了“返回”键并不等于“取消”，它意味着activity的当前数据被保存下来，离开了activity。在一个activity中要取消编辑动作必须有另外一套机制来实现，比如一个清晰的“恢复”或者“撤销”动作。
- 可以在content package里面查看更多内容提供者信息，有一些重要的方面，在activity调用和activity之间的数据传递的不同。
- activity也提供了一些API用于管理内在的状态，可以用在如浏览器中用户设置的缺省主页等。
- activity的持久状态由getPreferences(int)来进行管理，允许获取或者修改一套“名字/值”的对。为了在多个应用程序组件（activities, receivers, services, providers）之间共享持久状态，可以使用Context.getSharedPreferences()接口获取一个共享对象。（注意：不可能跨越应用程序包来共享数据，但是可以使用内容提供者来做）

保存持久状态（例子）

```
• public class CalendarActivity extends Activity {  
•     ...  
•     static final int DAY_VIEW_MODE = 0;  
•     static final int WEEK_VIEW_MODE = 1;  
•     private SharedPreferences mPrefs;  
•     private int mCurViewMode;  
  
•     protected void onCreate(Bundle savedInstanceState) {  
•         super.onCreate(savedInstanceState);  
•         SharedPreferences mPrefs = getSharedPreferences();  
•         mCurViewMode = mPrefs.getInt("view_mode" DAY_VIEW_MODE);  
•     }  
  
•     protected void onPause() {  
•         super.onPause();  
•         SharedPreferences.Editor ed = mPrefs.edit();  
•         ed.putInt("view_mode", mCurViewMode);  
•         ed.commit();  
•     }  
• }
```

权限许可

- 当activity在manifest的<activity> 标签中进行了声明，就有能力进行启动。其他的应用程序为了能够启动这个activity，需要在他们自己的manifest资源文件中的<uses-permission>元素中进行声明。
- 更多的权限和安全问题，可以参见Security Model 文档。

进程的生命周期

- **Android系统试图确保一个应用进程尽可能长久，但是在内存减少的时候最终要删除旧的进程。像Activity Lifecycle描述的一样，决定哪一个进程被删除是依据于和用户打交道的状态。一般来说，基于运行的activity的进程有四种状态，下面列出了重要的顺序，Android系统在杀掉重要进程之前先杀掉次要的进程。**

进程的生命周期（续一）

- 1. 前台的activity（activity在屏幕的最前面，正和用户进行交互）是最重要的，如果这个activity使用的内存超过了设备提供的内存，作为最后手段才杀掉这个activity的进程。通常在这点上表明设备已经达到了memory paging state（？），因为需要确保用户界面的交互。
- 2. 可见的activity（对用户可见，但不是在前台，如位于前台弹出对话框下面的界面）是很重要的，一般不会杀掉这个进程，除非为了确保前台activity的运行不得已而杀之。
- 3. 后台的activity（对用户不可见，已经暂停）不再重要，系统可以安全杀掉这个进程，回收内存给其他前台或者可见的进程。当用户按下“返回”键浏览这个activity（使它重新在屏幕上可见），它的onCreate(Bundle)接口将被调用，在暂停的时候由于已经调用过onSaveInstanceState(Bundle)接口重置了状态，因此这个activity能够重新以相同的设置启动。
- 4. 空进程是没有activity组件和其他应用程序组件（Service or IntentReceiver）的进程，在系统内存变低的时候很快就被删除了。出于这个原因，任何在activity之外的后台操作必须在IntentReceiver 或者Service的环境下执行，这样才能确保进程能够一直运行。

进程的生命周期（续二）

- 有时候activity希望能够执行一些长时间的操作，自立存在于activity的生命周期之外。例如：相机应用程序允许用户上传一幅图片到网站，上传将花费很长的时间，应用程序应该允许用户离开正在执行的程序。为了完成这些，activity应该启动一个Service来执行上传任务，在上传区间，系统能够给进程一个良好的优先级别（比那些不可见的进程更高的优先级别），而不用考虑源activity是否处于暂停，停止或者结束状态。

课程内容回顾

Android Activity Class :

- 类继承体系
- Activity生命周期及相应接口
- Activity启动及返回
- 持久数据保存
- 权限许可
- 进程生命周期

课程目标

- **Intent机制**
- **Intent用途**
- **Intent基本使用方法**

概述

- Intent这个英语单词的本意是“目的、意向”等
- Android中提供了Intent机制来协助应用间的交互与通讯
- Intent不仅可用于应用程序之间，也可用于应用程序内部的Activity/Service之间的交互。
- 相对函数调用来说，Intent是更为抽象的概念
- 利用Intent所实现的软件复用的粒度是Activity/Service，比函数复用更高一些，另外耦合也更为松散。

概述

- 一个Intent类似于一个Event
- Intent类绑定一次操作，它负责携带这次操作所需要的数据以及操作的类型等
- Android中与Intent相关的还有Action/Category及Intent Filter等

概述

- Intent的两个最重要的成员操作类型（Action）和数据（Data）
 - Intent的Action的取值主要是一些定义好了的常量，例如PICK_ACTION，VIEW_ACTION，EDIT_ACTION之类的
 - Data则是一个ContentURI类型的变量
- Intent要定位事件的目的地，无外乎需要以下几个信息：
 - 1.种类（category），比如我们常见的 LAUNCHER_CATEGORY 就是表示这是一类应用程序。
 - 2.类型（type），表示数据的类型，这是隐性Intent定位目标的重要依据。
 - 3.组件（component），常用的是setClass，不过也可以用setComponent来设置intent跳转的前后两个类实例。
 - 4.附加数据（extras），在ContentURI之外还可以附加一些信息，它是Bundle类型的对象。

Intent用途

- 加载活动

startActivity

- 广播发送（访问广播接收器）

broadcastIntent

- 访问服务

startService(Intent)

bindService(Intent, ServiceConnection, int)

Intent基本用法

- **显式的Intent**
 - 即在构造Intent对象时就指定接收者，这种方式与普通的函数调用类似，只是复用的粒度有所差别
- **隐式的Intent**
 - 即Intent的发送者在构造Intent对象时，并不知道也不关心接收者是谁，这种方式与函数调用差别比较大，有利于降低发送者和接收者之间的耦合
- **（用于广播的Intent）**

显式Intent(Explicit Intent)

- 同一个应用程序中的Activity切换
 - 通常一个应用程序中需要多个UI屏幕，也就需要多个Activity类，并且在这些Activity之间进行切换，这种切换就是通过Intent机制来实现的。
 - 在同一个应用程序中切换Activity时，我们通常都知道要启动的Activity具体是哪一个，因此常用显式的Intent来实现。

显式Intent使用例子

- 一个非常简单的应用程序SimpleIntentTest，它包括两个UI屏幕也就是两个Activity——SimpleIntentTest类和TestActivity类，SimpleIntentTest类有一个按钮用来启动TestActivity

例子代码-SimpleIntentTest

- `package com.tope.samples.intent.simple;`
- `import android.app.Activity;`
- `import android.content.Intent;`
- `import android.os.Bundle;`
- `import android.view.View;`
- `import android.widget.Button;`
- `public class SimpleIntentTest extends Activity implements`
`View.OnClickListener{`
- `/** Called when the activity is first created. */`
- `public void onCreate(Bundle savedInstanceState) {`
- `super.onCreate(savedInstanceState);`
- `setContentView(R.layout.main);`
- `Button startBtn = (Button)findViewById(R.id.start_activity);`
- `startBtn.setOnClickListener(this);`
- `}`

```
• public void onClick(View v) {  
•     switch (v.getId()) {  
•         case R.id.start_activity:  
•             Intent intent = new Intent(this, TestActivity.class);  
•             startActivity(intent);  
•             break;  
•         default:  
•             break;  
•     }  
• }  
• }
```


例子代码-TestActivity

- **package com.tope.samples.intent.simple;**
- **import android.app.Activity;**
- **import android.os.Bundle;**
- **public class TestActivity extends Activity {**
- **/** Called when the activity is first created. */**
- **@Override**
- **public void onCreate(Bundle savedInstanceState) {**
- **super.onCreate(savedInstanceState);**
- **setContentView(R.layout.test_activity);**
- **}**
- **}**

隐式的Intent

- 松耦合的实现方法，适用于那些较大的系统或者多个不同的应用之间的调用
- 可以与不同来源的多个应用之间方便地互操作

隐式Intent例子

- **HelloThreeProvider** , 实现ContentProvider接口
- **Activity HelloThree**创建发送隐式Intent
- **Android**平台负责根据Intent的Data信息中的authorities , 找到ContentProvider , 然后getType , 用type和intent中的Action两个信息 , 再找到可以处理这个intent的消费者

例子代码-HelloThreeProvider

```
• public class HelloThreeProvider extends ContentProvider {  
•     public boolean onCreate() {  
•         return true;  
•     }  
•     public int delete(ContentURI url, String where, String[] whereArgs) {  
•         return 0;  
•     }  
•     public ContentURI insert(ContentURI url, ContentValues initialValues){  
•         return url;  
•     }  
•     public Cursor query(ContentURI url, String[] projection, String selection,  
•         String[] selectionArgs, String groupBy, String having, String sort) {  
•         return null;  
•     }  
•     public int update(ContentURI url, ContentValues values, String where, String[] whereArgs)  
•     {  
•         return 0;  
•     }  
•  
•     public String getType(ContentURI url) {  
•         return "vnd.sharetop.hello.three/vnd.hello.three";  
•     }  
• }
```


例子代码-AndroidManifest.xml

- `<application android:icon="@drawable/icon">`
- `<provider class="HelloThreeProvider"`
- `android:authorities="cn.sharetop.android.hello" />`
- `<activity class="HelloThree" android:label="@string/app_name">`
- `<intent-filter>`
- `<action android:value="android.intent.action.MAIN" />`
- `<category android:value="android.intent.category.LAUNCHER"`
- `/>`
- `</intent-filter>`
- `</activity>`
- `<activity class="HelloThreeB" android:label="bbb">`
- `<intent-filter>`
- `<action android:value="android.intent.action.VIEW" />`
- `<category android:value="android.intent.category.DEFAULT" />`
- `<type android:value="vnd.sharetop.hello.three/vnd.hello.three" />`
- `</intent-filter>`
- `</activity>`
- `</application>`

例子代码-调用代码

- **Intent intent = new Intent();**
- **intent.setData(new
ContentURI("content://cn.sharetop.android.hello/one"));**
- **intent.setAction(intent.VIEW_ACTION);**
- **startActivity(intent);**

通过Intent进行服务调用

- 显示网页
- 地图相关
 - 显示地图
 - 路径规划
- 打电话
- 传送SMS/MMS
- 传送 Email
- 播放多媒体
- Market 相关
- Uninstall 应用程序

服务调用-显示网页

- 1. `Uri uri = Uri.parse("http://google.com");`
- 2. `Intent it = new Intent(Intent.ACTION_VIEW, uri);`
- 3. `startActivity(it);`

服务调用-显示地图

- 1. `Uri uri = Uri.parse("geo:38.899533,-77.036476");`
- 2. `Intent it = new Intent(Intent.ACTION_VIEW, uri);`
- 3. `startActivity(it);`

服务调用-路径规划

- **1. Uri uri =**
Uri.parse("http://maps.google.com/maps?f=d&saddr=startLat%20startLng&daddr=endLat%20endLng&hl=en");
- **2. Intent it = new Intent(Intent.ACTION_VIEW, uri);**
- **3. startActivity(it);**
- **4. //where startLat, startLng, endLat, endLng are a long with 6 decimals like: 50.123456**

服务调用-电话

- 1. //叫出拨号程序
- 2. `Uri uri = Uri.parse("tel:0800000123");`
- 3. `Intent it = new Intent(Intent.ACTION_DIAL, uri);`
- 4. `startActivity(it);`
- 1. //直接打电话出去
- 2. `Uri uri = Uri.parse("tel:0800000123");`
- 3. `Intent it = new Intent(Intent.ACTION_CALL, uri);`
- 4. `startActivity(it);`
- 5. //用這個，要在 AndroidManifest.xml 中，加上
- 6. `//<uses-permission id="android.permission.CALL_PHONE" />`

服务调用-SMS/MMS

- 1. //调用短信程序
- 2. Intent it = new Intent(Intent.ACTION_VIEW, uri);
- 3. it.putExtra("sms_body", "The SMS text");
- 4. it.setType("vnd.android-dir/mms-sms");
- 5. startActivity(it);

- 1. //传送消息
- 2. Uri uri = Uri.parse("smsto://08000000123");
- 3. Intent it = new Intent(Intent.ACTION_SENDTO, uri);
- 4. it.putExtra("sms_body", "The SMS text");
- 5. startActivity(it);

- 1. //传送 MMS
- 2. Uri uri = Uri.parse("content://media/external/images/media/23");
- 3. Intent it = new Intent(Intent.ACTION_SEND);
- 4. it.putExtra("sms_body", "some text");
- 5. it.putExtra(Intent.EXTRA_STREAM, uri);
- 6. it.setType("image/png");
- 7. startActivity(it);

服务调用-Email

- 1. Uri uri = Uri.parse("mailto:xxx@abc.com");
- 2. Intent it = new Intent(Intent.ACTION_SENDTO, uri);
- 3. startActivity(it);

- 1. Intent it = new Intent(Intent.ACTION_SEND);
- 2. it.putExtra(Intent.EXTRA_EMAIL, "me@abc.com");
- 3. it.putExtra(Intent.EXTRA_TEXT, "The email body text");
- 4. it.setType("text/plain");
- 5. startActivity(Intent.createChooser(it, "Choose Email Client"));

- 1. Intent it=new Intent(Intent.ACTION_SEND);
 - 2. String[] tos={"me@abc.com"};
 - 3. String[] ccs={"you@abc.com"};
 - 4. it.putExtra(Intent.EXTRA_EMAIL, tos);
 - 5. it.putExtra(Intent.EXTRA_CC, ccs);
 - 6. it.putExtra(Intent.EXTRA_TEXT, "The email body text");
 - 7. it.putExtra(Intent.EXTRA_SUBJECT, "The email subject text");
 - 8. it.setType("message/rfc822");
 - 9. startActivity(Intent.createChooser(it, "Choose Email Client"));
-
- 1. //传送附件
 - 2. Intent it = new Intent(Intent.ACTION_SEND);
 - 3. it.putExtra(Intent.EXTRA_SUBJECT, "The email subject text");
 - 4. it.putExtra(Intent.EXTRA_STREAM, "file:///sdcard/mysong.mp3");
 - 5. sendIntent.setType("audio/mp3");
 - 6. startActivity(Intent.createChooser(it, "Choose Email Client"));

服务调用-多媒体

- `Uri uri = Uri.parse("file:///sdcard/song.mp3");`
- `Intent it = new Intent(Intent.ACTION_VIEW, uri);`
- `it.setType("audio/mp3");`
- `startActivity(it);`

- `Uri uri =`
 `Uri.withAppendedPath(MediaStore.Audio.Media.INTERNAL_`
 `CONTENT_URI, "1");`
- `Intent it = new Intent(Intent.ACTION_VIEW, uri);`
- `startActivity(it);`

服务调用-Market相关

- 1.//寻找某个应用
 - 2.Uri uri = Uri.parse("market://search?q=pname:pkg_name");
 - 3.Intent it = new Intent(Intent.ACTION_VIEW, uri);
 - 4.startActivity(it);
 - 5.//where pkg_name is the full package path for an application
-
- 1.//显示某个应用的相关信息
 - 2.Uri uri = Uri.parse("market://details?id=app_id");
 - 3.Intent it = new Intent(Intent.ACTION_VIEW, uri);
 - 4.startActivity(it);
 - 5.//where app_id is the application ID, find the ID
 - 6.//by clicking on your application on Market home
 - 7.//page, and notice the ID from the address bar

服务调用-Uninstall应用

- 1. `Uri uri = Uri.fromParts("package", strPackageName, null);`
- 2. `Intent it = new Intent(Intent.ACTION_DELETE, uri);`
- 3. `startActivity(it);`

课程内容回顾

- **Intent机制**
- **Intent用途**
- **Intent基本使用方法**

课程目标

- **AndroidManifest.xml**

概述

- **AndroidManifest.xml**是每个android程序中必须的文件。它位于application的根目录，描述了package中的全局数据，包括了package中暴露的组件（activities, services, 等等），他们各自的实现类，各种能被处理的数据和启动位置。

概述

- 此文件一个重要的地方就是它所包含的intent-filters。这些filters描述了activity启动的位置和时间。每当一个 activity（或者操作系统）要执行一个操作，例如：打开网页或联系簿时，它创建出一个intent的对象。它能承载一些信息描述了你想做什么，你想处理什么数据，数据的类型，和一些其他信息。Android比较了intent对象中和每个application所暴露的intent-filter中的信息，来找到最合适的activity来处理调用者所指定的数据和操作。

概述

- 除了能声明你程序中的Activities, Content Providers, Services, 和 Intent Receivers,你还能指定permissions和instrumentation (安全控制和测试) 在AndroidManifest.xml文件中。

简单的例子

- `<?xml version="1.0" encoding="utf-8"?>`
- `<manifest`
`xmlns:android="http://schemas.android.com/apk/res/android"`
- `package="com.my_domain.app.helloactivity">`
- `<application android:label="@string/app_name">`
`<activity class=".HelloActivity">`
- `<intent-filter>`
- `<action android:value="android.intent.action.MAIN"/>`
- `<category`
`android:value="android.intent.category.LAUNCHER"/>`
- `</intent-filter>`
- `</activity>`
- `</application>`
- `</manifest>`

例子

- 几乎所有的AndroidManifest.xml（以及许多其他Android的xml的文件）在第一个元素中包含了命名空间的声明
`xmlns:android= "http://schemas.android.com/apk/res/android"`。这样使得Android中各种标准属性能在文件中使用，提供了大部分元素中的数据。
- 大部分manifests包含了单个<application>的元素，它定义了所有的application级别组件和属性，并能在package中使用。
- 任何被用户看作顶层应用程序，并能被程序启动器所用的package,需要包含至少一个Activity组件来支持MAIN操作和LAUNCHER种类，如上述代码中所见。

课程目标

- **用户界面**
 - 视图层次
 - 布局
 - 界面组件
 - 用户界面事件
 - 菜单
- **高级话题**
 - Adapter
 - 风格与主题

概述

- **关键类**

- 1. View

- 2. ViewGroup

- 3. Widget classes

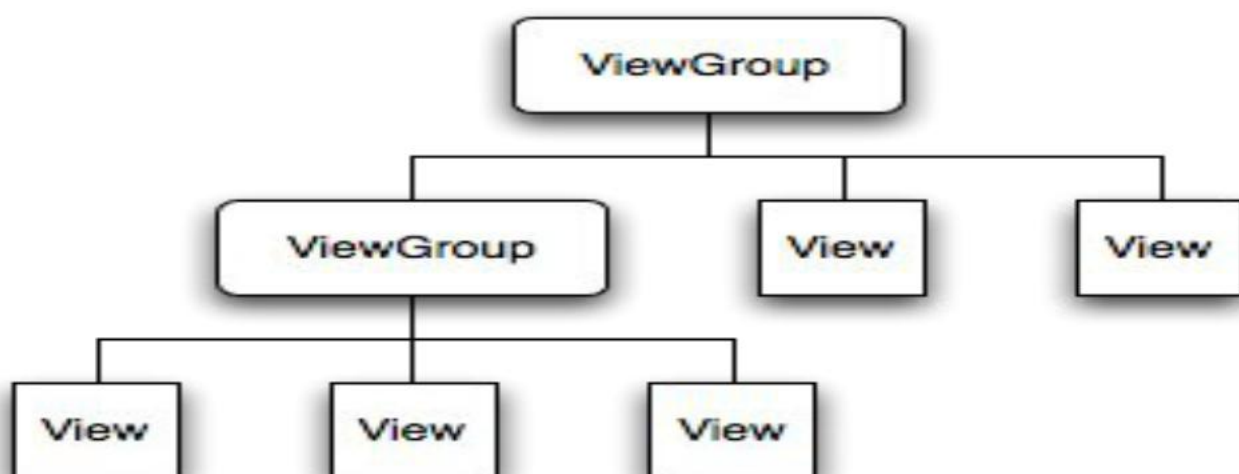
- 在一个Android应用中，用户界面是由View 和ViewGroup对象构建的。View与ViewGroup都有很多种类，而它们都是View类的子类。
- View对象是Android平台中用户界面体现的基础单位。View类是它称为“widgets（工具）”的子类的基础，它们提供了诸如文本输入框和按钮之类的UI对象的完整实现。ViewGroup类同样为其被称为“Layouts（布局）”的子类奠定了基础，它们提供了象流式布局、表格布局以及相对布局之类的布局架构。
- View对象是一个数据体，它的属性存储了用于屏幕上一块矩形区域的布局参数及内容。并负责这块它所辖的这个矩形区域之中所有测量、布局、焦点转换、滚动以及按键/触摸手势的处理。作为一个用户界面对象，View同时也担任着用户交互关键点以及交互事件接受者的角色。

概述

- 视图(View)类代表了一种基本的用户界面组成模块。一个视图占据了屏幕上的一个矩形区域，并响应绘制图形和事件处理。视图类是窗体类(Widget)的基类，而窗体类用来生成可交互的用户图形接口(interactive GUI)。
- 视图类的使用窗口中所有的视图构成一个树形结构。要想增加视图，既可以用直接添加代码的方法，也可以在一个或者多个XML文件中声明新视图构成的树。在视图类的子类中，有的可以用来控制，有的具有显示文字、图片或者其他内容的功能。
- 当视图树被创建后，以下这若干种通用操作将可以被使用：
- 1.设置属性(properties):比如，可以设置TextView类的一个实例的文本内容。不同的子类可以用来设置的属性与方法不同。注意：只有编译时能够检测到的属性才可以在XML布局管理(layout)文件中设置。
- 2.设置输入焦点(focus):为了响应用户输入，整个框架将处理移动的焦点。如果想把焦点强制指向某一个特定的视图，必须调用requestFocus()方法。
- 3.设置监听器(listener)：在视图中，允许设置监听器来捕获用户感兴趣的某些事件。比如说，在所有的视图中，无论视图是获得焦点还是失去焦点，都可以通过设置监听器来捕获。可以通过调用setOnFocusChangeListener(View.OnFocusChangeListener)来注册一个监听器。在其他视图子类中，提供了一些更加特殊的监听器。比如，一个按键(Button)可以触发按键被按下的事件。
- 4.设置是否可视(visibility):可以通过调用setVisibility(int)来显示或者隐藏视图

视图层次

- 在Android平台上，你可以用下图所示的View和ViewGroup层次图来定义一个Activity的UI。这个层次树可随你所愿的简单或者复杂化，你能使用Android预定义的一套工具和布局来创建它，或者使用你自己定义的Views来创建。



视图层次（续）

- 为了把一个视图层次树展现到屏幕上，你的Activity必须调用 `setContentView()` 方法，并传给它一个根节点对象的引用。Android系统将接受此引用，并用来进行界面的废止、测量并绘制这棵树。层次的根结点会要求它的子节点进行自我绘制——进而，每个视图组节点也负责调用它的子视图进行自我绘制。子节点将向父节点申请绘制的位置以及大小，而其父类享有子节点绘制的位置及大小的最终决定权。Android依次（自层次树顶层开始）解析你布局中的元素，实例化View并将它们添加到它们的父节点中。因为这个过程是依次进行的，所以如果出现了元素重叠的情况，最后一个绘制的元素将位于所有重叠元素之上显现。

布局

- 定义并展现你的视图层次的最常用的方法是使用XML布局文件。如同HTML一样，XML为布局提供了一种可读的结构。XML中的每个元素都是View或ViewGroup对象（抑或它们的子类）。View对象是树的叶节点，而ViewGroup对象是树的分支（参阅视图层次图）。
- XML元素的名称与它体现的Java类相对应。所以一个<TextView>元素将在你的UI中生成一个TextView，而<LinearLayout>则创建一个LinearLayout视图组。当你载入一个布局资源时，Android系统会根据你布局中的元素初始化这些运行时对象。

布局（例子）

- 举例来说，一个包含文本视图和一个按钮的简单垂直布局如下：
- `<?xml version="1.0" encoding="utf-8"?>`
- `<LinearLayout`
`xmlns:android="http://schemas.android.com/apk/res/android"`
- `android:layout_width="fill_parent"`
- `android:layout_height="fill_parent"`
- `android:orientation="vertical" >`
- `<TextView android:id="@+id/text"`
- `android:layout_width="wrap_content"`
- `android:layout_height="wrap_content"`
- `android:text="Hello, I am a TextView" />`
- `<Button android:id="@+id/button"`
- `android:layout_width="wrap_content"`
- `android:layout_height="wrap_content"`
- `android:text="Hello, I am a Button" />`
- `</LinearLayout>`
- 请注意：LinearLayout 元素包含了TextView 和Button 对象。你可以在其中另外安置一个LinearLayout（或其它类型的视图组），以延展这个视图层次，构建更复杂的布局。

布局（续）

- 也可以用Java代码来绘制View和ViewGroup对象，并用 `addView(View)` 方法动态的插入新的View和ViewGroup对象。
- 您有相当多的方法来对视图进行布局。使用大量不同种类的视图组，您可以有近乎无穷的方式来构建子视图和视图组。Android提供了一些预定义的视图组，其中包括 `LinearLayout`，`RelativeLayout`，`AbsoluteLayout`，`TableLayout`，`GridLayout` 以及其它的一些。每个都为定义子视图和布局结构提供了一套独特的布局参数。

界面组件

- 组件是为用户交互界面提供服务的视图对象。Android提供了一套完整的组件实现，包括按钮、复选框、文本输入框等，以助于你快速的构建UI。Android还提供了一些更高级的工具，比如日期选择、时钟以及缩放控制。但您并没有被局限于Android平台提供的这些组件上。如果您想创建一些您自己的定制动作元素，您可以这么做，只要定义自己的视图对象或者扩展或合并现有的组件就行。
- 可以在android.widget包中找到Android提供的工具列表

用户界面事件

- 欲获得用户界面事件通知，你需要做以下两件事情之一：

定义一个事件侦听器并将其注册至视图。通常情况下，这是你侦听事件的主要方式。View类包含了一大堆命名类似 `On<什么什么>Listener` 的接口，每个都带有一个叫做 `On<什么什么>()` 的回调方法。比如：`View.OnClickListener`（用以处理视图中的点击），`View.OnTouchListener`（用以处理视图中的触屏事件），以及 `View.OnKeyListener`（用以处理视图中的设备按键事件）。所以，如果你希望你的视图在它被“点击”（比如选择了一个按钮）的时候获得通知，你就要实现 `OnClickListener`，定义它的 `onClick()` 回调方法（在其中进行相应处理），并将它用 `setOnClickListener()` 方法注册到视图上。

为视图覆写一个现有的回调方法。这种方法主要用于你自己实现了一个View类，并想侦听其上发生的特定事件。比如说当屏幕被触摸（`onTouchEvent()`），当轨迹球发生了移动（`onTrackballEvent()`）或者是设备上的按键被按下（`onKeyDown()`）。这种方式允许你为自己定制的视图中发生的每个事件定义默认的行为，并决定是否需要将事件传递给其它的子视图。再说一次，这些是View类相关的回调方法，所以你只能在你构建自定义组件时定义它们。

菜单

- 应用程序菜单是应用程序用户界面中另外一个重要的组成部分。菜单为展现应用程序功能和设置提供了一个可靠的界面。按下设备上的MENU键会调出最普通的应用程序菜单。然而，你也可以加入当用户长按一个项目时调出的上下文菜单。
- 菜单也是用视图层次进行构架的，但你不必自己定义这个架构。你只要为你的Activity定义onCreateOptionsMenu() 和 onCreateContextMenu() 回调方法，并声明你想要包含在菜单中的项目就行了。Android将为你菜单自动创建视图层次，并在其中绘入你的菜单项。
- 菜单会自行处理它们的事件，所以你不必为你菜单中的项目注册事件侦听器。当你菜单中的一项被选定时，框架将自动调用 onOptionsItemSelected() 或 onContextItemSelected() 方法。
- 如同应用程序布局一样。你也可以在一个XML文件中定义你菜单中的项目。

高级话题

- 一旦你对创建用户界面的基础了如指掌，你就可以尝试着用一些高级功能来创建更加复杂的应用程序界面。

Adapter

- 有时候你会想要用一些无法硬编码的信息来填充视图组。你想将源于外部的数据绑定到你的视图中。为达到这个目的，你可以使用AdapterView作为你的视图组，并用Adapter传来的数据初始化每个子视图并填入其中。
- AdapterView对象是一个用给定的Adapter对象为基础构建它的子视图的ViewGroup实现。而Adapter在你的数据源（可能是一个外部字符串数组）和显示这些数据的AdapterView之间扮演着一个信使的角色。针对特定的任务有着很多不同的Adapter类实现，比如 CursorAdapter依据Cursor读出一个数据库的数据，而一个ArrayAdapter则从任一个数组进行读取。

风格与主题

- 或许你对标准工具的外表不是那么满意。为了解决这个问题，你可以创建你自己的风格和主题。

风格是一套包含一个或多个格式化属性的整体，你可以把它们加诸于你布局中的单个元素之上。比如，你可以定义一个包含特定文本字体大小和颜色的风格，并将它单独施用于特定的视图元素。

主题也是一套包含一个或多个格式化属性的整体，但却应用于一个应用程序中的所有 Activity，或单独一个 Activity。比如说，你可以定义一个包含了特定窗口边框颜色和版面背景、以及一套字体大小和菜单颜色的主题。这个主题可以施用于特定的 Activity 抑或整个应用程序。

- 风格与主题隶属于资源。Android 提供了一些默认的风格和主题供你使用，你也可以定制你自己的风格和主题资源。

课程内容回顾

- 用户界面
 - 视图层次
 - 布局
 - 界面组件
 - 用户界面事件
 - 菜单
- 高级话题
 - Adapter
 - 风格与主题

课程目标

- **ContentProvider**
- **SQLite**

概述

- **Android应用程序可以使用文件或SqlLite数据库来存储数据。**
- **Content Provider提供了一种多应用间数据共享的方式，比如：联系人信息可以被多个应用程序访问。**
- **Content Provider是个实现了一组用于提供其他应用程序存取数据的标准方法的类。**

ContentProvider中可执行操作

- 查询数据
- 修改数据
- 添加数据
- 删除数据

标准的Content Provider

- **Android提供了一些已经在系统中实现的标准Content Provider , 比如联系人信息 , 图片库等等 , 你可以用这些Content Provider来访问设备上存储的联系人信息 , 图片等等。**

查询记录

- 在Content Provider中使用的查询字符串有别于标准的SQL查询。很多诸如 select, add, delete, modify等操作我们都使用一种特殊的URI来进行
- URI由3个部分组成，
 - “content://”，代表数据的路径，和一个可选的标识数据的ID。以下是一些示例URI:
 - content://media/internal/images 这个URI将返回设备上存储的所有图片
 - content://contacts/people/ 这个URI将返回设备上的所有联系人信息
 - content://contacts/people/45 这个URI返回单个结果（联系人信息中ID为45的联系人记录）

查询记录-2

- 尽管这种查询字符串格式很常见，但是它看起来还是有点令人迷惑。为此，Android提供一系列的帮助类（在android.provider包下），里面包含了很多以类变量形式给出的查询字符串，这种方式更容易让我们理解一点，参见下例：
 - `MediaStore.Images.Media.INTERNAL_CONTENT_URI`
 - `Contacts.People.CONTENT_URI`
- 因此，如上面`content://contacts/people/45`这个URI就可以写成如下形式：
- `Uri person = ContentUris.withAppendedId(People.CONTENT_URI, 45);`
- 然后执行数据查询：
- `Cursor cur = managedQuery(person, null, null, null);`
- 这个查询返回一个包含所有数据字段的游标，我们可以通过迭代这个游标来获取所有的数据：

查询记录-示例

- `/*示范了一个如何依次读取联系人信息表中的指定数据列name和number*/`
- `public class ContentProviderDemo extends Activity {`
- `@Override`
- `public void onCreate(Bundle savedInstanceState) {`
- `super.onCreate(savedInstanceState);`
- `setContentView(R.layout.main);`
- `displayRecords();`
- `}`
- `private void displayRecords() {`
- `//该数组中包含了所有要返回的字段`
- `String columns[] = new String[] { People.NAME, People.NUMBER };`
- `Uri mContacts = People.CONTENT_URI;`
- `Cursor cur = managedQuery(`
- `mContacts,`
- `columns, // 要返回的数据字段`
- `null,      // WHERE子句`
- `null,      // WHERE 子句的参数`
- `null      // Order-by子句`
- `);`

```
•   if (cur.moveToFirst()) {  
•       String name = null;  
•       String phoneNo = null;  
•       do {  
•           // 获取字段的值  
•           name = cur.getString(cur.getColumnIndex(People.NAME));  
•           phoneNo = cur.getString(cur.getColumnIndex(People.NUMBER));  
•           Toast.makeText(this, name + " " + phoneNo,  
•               Toast.LENGTH_LONG).show();  
•       } while (cur.moveToNext());  
•   }  
• }  
• }
```


修改记录

- 使用ContentResolver.update()方法来修改数据，我们来写一个修改数据的方法：
 - `private void updateRecord(int recNo, String name) {`
 - `Uri uri =`
`ContentUris.withAppendedId(People.CONTENT_URI,`
`recNo);`
 - `ContentValues values = new ContentValues();`
 - `values.put(People.NAME, name);`
 - `getContentResolver().update(uri, values, null, null);`
 - `}`
- 调用上面的方法来更新指定记录：
 - `updateRecord(10, " XYZ" );` //更改第10条记录的name字段值为 "XYZ"

添加记录

- 要增加记录，我们可以调用`ContentResolver.insert()`方法，该方法接受一个要增加的记录的目标URI，以及一个包含了新记录值的Map对象，调用后的返回值是新记录的URI，包含记录号。

添加记录-示例

- 基于联系人信息簿这个标准的Content Provider，现在我们继续来创建一个insertRecord() 方法以对联系人信息簿中进行数据的添加：

```
• private void insertRecords(String name, String phoneNo) {  
•     ContentValues values = new ContentValues();  
•     values.put(People.NAME, name);  
•     Uri uri = getContentResolver().insert(People.CONTENT_URI, values);  
•     Log.d(" ANDROID" , uri.toString());  
•     Uri numberUri = Uri.withAppendedPath(uri,  
People.Phones.CONTENT_DIRECTORY);  
•     values.clear();  
•     values.put(Contacts.Phones.TYPE, People.Phones.TYPE_MOBILE);  
•     values.put(People.NUMBER, phoneNo);  
•     getContentResolver().insert(numberUri, values);  
• }
```

删除记录

- Content Provider中的`getContextResolver.delete()`方法可以用来删除记录，下面的记录用来删除设备上所有的联系人信息：
- ```
private void deleteRecords() {
```
- ```
    Uri uri = People.CONTENT_URI;
```
- ```
 getContentResolver().delete(uri, null, null);
```
- ```
}
```
- 你也可以指定WHERE条件语句来删除特定的记录：
- ```
getContentResolver().delete(uri, "NAME=" + " 'XYZ XYZ' ", null);
```
- 这将会删除name为 'XYZ XYZ' 的记录。

# 定制Content Provider

- 定制需遵循步骤：
  - 1. 创建一个继承了ContentProvider父类的类
  - 2. 定义一个名为CONTENT\_URI，并且是public static final的Uri类型的类变量，你必须为其指定一个唯一的字符串值，最好的方案是以类的全名称，如：  
`public static final Uri CONTENT_URI = Uri.parse( "content://com.google.android.MyContentProvider" );`
  - 3. 创建你的数据存储系统。大多数Content Provider使用Android文件系统或SQLite数据库来保持数据，但是你也可以以任何你想要的方式来存储。
  - 4. 定义你要返回给客户端的数据列名。如果你正在使用Android数据库，则数据列的使用方式就和你以往所熟悉的其他数据库一样。但是，你必须为其定义一个叫\_id的列，它用来表示每条记录的唯一性。
  - 5. 如果你要存储字节型数据，比如位图文件等，那保存该数据的数据列其实是一个表示实际保存文件的URI字符串，客户端通过它来读取对应的文件数据，处理这种数据类型的Content Provider需要实现一个名为\_data的字段，\_data字段列出了该文件在Android文件系统上的精确路径。这个字段不仅是供客户端使用，而且也可以供ContentResolver使用。客户端可以调用ContentResolver.openOutputStream()方法来处理该 URI指向的文件资源，如果是ContentResolver本身的话，由于其持有的权限比客户端要高，所以它能直接访问该数据文件。
  - 6. 声明public static String型的变量，用于指定要从游标处返回的数据列。

# 定制Content Provider-示例

- 代码将创建一个Content Provider，它仅仅是存储用户名称并显示所有的用户名称（使用 SQLite数据库存储这些数据）



## 定制Content Provider-示例

- 类中定义了Content Provider的CONTENT\_URI , 以及数据列
  - package com.wissen.testApp;
  - public class MyUsers {
  - public static final String AUTHORITY =  
      “com.wissen.MyContentProvider” ;
  - // BaseColumn类中已经包含了 \_id字段
  - public static final class User implements BaseColumns
  - {
  - public static final Uri CONTENT\_URI =  
      Uri.parse(“ content://com.wissen.MyContentProvider” );
  - // 表数据列
  - public static final String USER\_NAME =  
      “USER\_NAME” ;
  - }
  - }

# 定制ContentProvider的XML配置

- Content Provider的入口需要在AndroidManifest.xml中配置:
  - `<provider android:name=" MyContentProvider"  
android:authorities=" com.wissen.MyContentProvider"  
/>`

# 定制ContentProvider的使用

- 先向数据库中添加一条用户数据，然后显示数据库中所有的用户数据
  - package com.wissen.testApp;
  - public class MyContentDemo extends Activity {
  - @Override
  - protected void onCreate(Bundle savedInstanceState) {
  - super.onCreate(savedInstanceState);
  - insertRecord(" MyUser" );
  - displayRecords();
  - }
  - private void insertRecord(String userName) {
  - ContentValues values = new ContentValues();
  - values.put(MyUsers.User.USER\_NAME, userName);
  - getContentResolver().insert(MyUsers.User.CONTENT\_URI,
  - values);
  - }
  -

```
- private void displayRecords() {
- String columns[] = new String[] { MyUsers.User._ID,
MyUsers.User.USER_NAME };
- Uri myUri = MyUsers.User.CONTENT_URI;
- Cursor cur = managedQuery(myUri, columns,null, null,
null);
- if (cur.moveToFirst()) {
- String id = null;
- String userName = null;
- do {
- id =
cur.getString(cur.getColumnIndex(MyUsers.User._ID));
- userName =
cur.getString(cur.getColumnIndex(MyUsers.User.USER_NAME));
- Toast.makeText(this, id + " " + userName,
Toast.LENGTH_LONG).show();
- } while (cur.moveToNext());
- }
- }
- }
```

# SQLite-打开数据库

- **Context类的openDatabase可以打开一个已经存在的数据库，如果数据库不存在，将会抛出FileNotFoundException异常。可以通过Context类的createDatabase函数建立一个新的数据库。通过调用SQLiteDatabase的execSQL方法，执行一条SQL语句建立一个新的数据表。**



# SQLite-打开数据库

```
• public DBHelper(Context ctx) {
• try {
• //打开已经存在的数据库
• db = ctx.openDatabase(DATABASE_NAME, null);
• } catch (FileNotFoundException e) {
• try {
• //建立新的数据库
• db = ctx.createDatabase(DATABASE_NAME,
DATABASE_VERSION, 0, null);
• //建立数据表
• db.execSQL(DATABASE_CREATE);
• } catch (FileNotFoundException e1) {
• db = null;
• }
• }
• }
• }
```

# SQLite-获取表中的数据

- 建立一个游标类Cursor 通过SQLiteDatabase 的query方法查询一个表格。有了Cursor就可以遍历所有的记录了。

# SQLite-获取表中的数据

```
• public List<row> fetchAllRows() { ArrayList<row> ret = new
 ArrayList<row>();
• try {
• Cursor c =
• db.query(DATABASE_TABLE, new String[] {
• "rowid", "title", "body"}, null, null, null, null, null);
• int numRows = c.count();
• c.first();
• for (int i = 0; i < numRows; ++i) {
• Row row = new Row();
• row.rowId = c.getLong(0);
• row.title = c.getString(1);
• row.body = c.getString(2);
• ret.add(row);
• c.next();
• }
• } catch (SQLException e) {
• Log.e("booga", e.toString());
• }
• return ret;
• }
```

# SQLite-添加新记录

- 构造一个ContentValues类，通过调用put方法，可以设置一条记录的属性。通过调用SQLiteDatabase的insert方法添加一条新的记录。

# SQLite-添加新记录

- **public void createRow(String title, String body) {**
- **ContentValues initialValues = new ContentValues();**
- **initialValues.put("title", title);**
- **initialValues.put("body", body);**
- **db.insert(DATABASE\_TABLE, null, initialValues);**
- **}**



# SQLite-删除记录

- 直接调用SQLiteDatabase的delete方法，第二个参数是一个SQL条件表达式。

# SQLite-删除记录

- **public void deleteRow(String str)**
- **{**
- **db.delete(DATABASE\_TABLE, "title=\'" + str+ "\'", null);**
- **}**

# SQLite-关闭数据库

- **db.close()**
  - **SQLiteDatabase db = null;**
  - ...
  - **if (db != null) {**  
    **db.close();**  
    **Db = null;**  
    **}**

# 生命周期

- 广播接收器只有一个回调方法：
  - `void onReceive(Context curContext, Intent broadcastMsg)`
- 当广播消息抵达接收器时，Android调用它的`onReceive()` 方法并将包含消息的Intent对象传递给它。广播接收器仅在它执行这个方法时处于活跃状态。当`onReceive()`返回后，它即为失活状态。

## 生命周期（续）

- 拥有一个活跃状态的广播接收器的进程被保护起来而不会被杀死。但仅拥有失活状态组件的进程则会在其它进程需要它所占有的内存的时候随时被杀掉。
- 这种方式引出了一个问题：如果响应一个广播信息需要很长的一段时间，我们一般会将其纳入一个衍生的线程中去完成，而不是在主线程内完成它，从而保证用户交互过程的流畅。如果onReceive()衍生了一个线程并且返回，则包涵新线程在内的整个进程都会被判为失活状态（除非进程内的其它应用程序组件仍处于活跃状态），于是它就有可能被杀掉。这个问题的解决方法是令onReceive()启动一个新服务，并用其完成任务，于是系统就会知道进程中仍然在处理着工作。



# 课程目标

- 进程与生命周期

# 概述

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

# 进程重要性列表

- 前台进程
- 可视进程
- 服务进程
- 背景进程
- 空进程

# 前台进程

- 前台进程是用户操作所必须的。当满足如下任一条件时，进程被认为是处于前台的：
  - 它运行着正在与用户交互的activity ( Activity对象的 `onResume()` 方法已被调用 )。
  - 一个正在与用户交互的activity使用着它提供的一个服务。
  - 它包含着一个正在执行生命周期回调方法 ( `onCreate()`、`onStart()`或`onDestroy()` ) 的Service对象。
  - 它包含着一个正在执行 `onReceive()` 方法的BroadcastReceiver对象。
- 任一时间下，仅有少数进程会处于前台，仅当内存实在无法供给它们维持同时运行时才会被杀死。一般来说，在这种情况下，设备已然处于使用虚拟内存的状态，必须要杀死一些前台进程以用户界面保持响应。

# 可视进程

- 可视进程没有前台组件，但仍可被用户在屏幕上所见。当满足如下任一条件时进程被认为是可视的
  - 它包含着一个不在前台，但仍然为用户可见的activity（它的onPause()方法被调用）。这种情况可能出现在以下情况：比如说，前台activity是一个对话框，而之前的activity位于其下并可以看到。
  - 它包含了一个绑定至一个可视的activity的服务。
- 可视进程依然被视为是很重要的，非到不杀死它们便无法维持前台进程运行时才会被杀死。



# 服务进程

- **服务进程是由 `startService()` 方法启动的服务，它不会变成上述两类。尽管服务进程不会直接为用户所见，但它们一般都在做着用户所关心的事情（比如在后台播放mp3或者从网上下载东西）。所以系统会尽量维持它们的运行，除非系统内存不足以维持前台进程和可视进程的运行需要。**

## 背景进程

- 背景进程包含目前不为用户所见的activity（Activity对象的onStop()方法已被调用）。这些进程与用户体验没有直接的联系，可以在任意时间被杀死以回收内存供前台进程、可视进程以及服务进程使用。一般来说，会有很多背景进程运行，所以它们一般存放于一个LRU（最后使用）列表中以确保最后被用户使用的activity最后被杀死。如果一个activity正确的实现了生命周期方法，并捕获了正确的状态，则杀死它的进程对用户体验不会有任何不良影响。

# 空进程

- **空进程不包含任何活动应用程序组件。这种进程存在的唯一原因是做为缓存以改善组件再次于其中运行时的启动时间。系统经常会杀死这种进程以保持进程缓存和系统内核缓存之间的平衡。**

# 进程重要性估量原则

- Android会依据进程中当前活跃组件的重要程度来尽可能高的估量一个进程的级别。比如说，如果一个进程中同时有一个服务和一个可视的activity，则进程会被判定为可视进程，而不是服务进程。
- 此外，一个进程的级别可能会由于其它进程依赖于它而升高。一个为其它进程提供服务的进程级别永远高于使用它服务的进程。比如说，如果A进程中的内容提供者进程为进程B中的客户端提供服务，或进程A中的服务为进程B中的组件所绑定，则A进程最低也会被视为与进程B拥有同样的重要性。
- 因为运行着一个服务的进程重要级别总高于一个背景activity。所以一个activity以启动一个服务的方式启动一个长时间运行过程比简单的衍生一个线程来进行处理要好。尤其是当处理过程比activity本身存在时间要长的情况之下。我们以背景音乐播放和上传一个相机拍摄的图片至网站上为例。使用服务则不论activity发生何事，都至少可以保证操作拥有“服务进程”的权限。如广播接收器生命周期中所提到的，这也正是广播接收器使用服务，而不是使用线程来处理耗时任务的原因。

# 课程内容回顾

- 进程与生命周期



# 课程目标

- 服务调用
- 服务生命周期及回调方法

# 服务的使用方式

- 服务以两种方式使用：

它可以启动并运行，直至有人停止了它或它自己停止。在这种方式下，它以调用 `Context.startService()` 启动，而以调用 `Context.stopService()` 结束。它可以调用 `Service.stopSelf()` 或 `Service.stopSelfResult()` 来自己停止。不论调用了多少次 `startService()` 方法，你只需要调用一次 `stopService()` 来停止服务。

它可以通过自己定义并暴露出来的接口进行程序操作。客户端建立一个到服务对象的连接，并通过那个连接来调用服务。连接以调用 `Context.bindService()` 方法建立，以调用 `Context.unbindService()` 关闭。多个客户端可以绑定至同一个服务。如果服务此时还没有加载，`bindService()` 会先加载它。

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

# 生命周期

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

# 生命周期循环

- 监控服务的两个嵌套的生命周期循环：
  - 服务的完整生命周期始于调用onCreate()而终于onDestroy()方法返回。如同activity一样，服务在onCreate()里面进行它自己的初始化，而在onDestroy()里面释放所有资源。比如说，一个音乐回放服务可以在onCreate()中创建播放音乐的线程，而在onDestroy()中停止这个线程。
  - 服务的活跃生命周期始于调用onStart()。这个方法用于处理传递给startService()的Intent对象。音乐服务会打开Intent来探明将要播放哪首音乐，并开始播放。
- 服务停止时没有相应的回调方法——不存在onStop()方法。

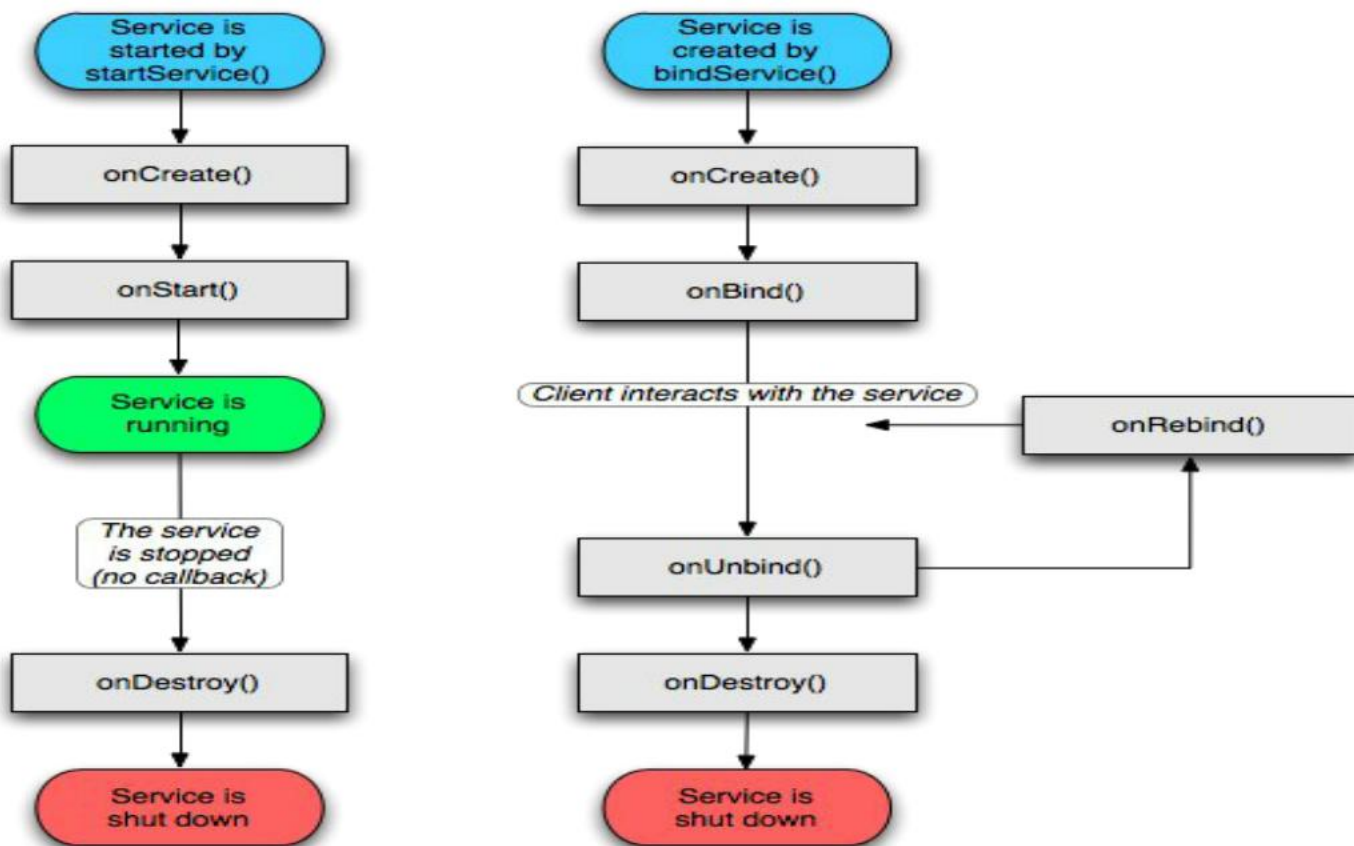


# 生命周期方法

- **onCreate()和onDestroy()方法在所有服务中都会被调用，不论它们是由Context.startService()还是由Context.bindService()所启动的。而onStart()仅会被startService()所启用的服务调用。**
- **如果一个服务允许别的进程绑定，则它还会有以下额外的回调方法以供实现：**
  - **IBinder onBind(Intent intent)**
  - **boolean onUnbind(Intent intent)**
  - **void onRebind(Intent intent)**
- **传递给bindService的Intent的对象也会传递给onBind()回调方法，而传递给unbindService()的Intent对象同样传递给onUnbind()。如果服务允许绑定，onBind()将返回一个供客户端与服务进行交互的通讯渠道。如果有新的客户端连接至服务，则onUnbind()方法可以要求调用onRebind()。**



# 服务生命周期及回调方法



# 服务的回调方法

- 图中对由startService 和bindService方法启动的服务做了区分，但要记住，不论一个服务是怎么启动的，它都可能允许客户端的连接，所以任何服务都可以接受onBind()和onUnbind()调用。

# 课程内容回顾

- **服务调用**
- **服务生命周期及回调方法**