1. **What is Android？ Why Android？**

Android是谷歌和OHA有可能彻底改变全球手机市场的软件平台。它有能力向许多其他（非手机）的嵌入式应用市场进军。

Android 拥有完整的一套移动设备的软件组件：操作系统，中间件，核心移动应用。计算设备日益一体化（趋向于移动互联网），手机不只是“手机”，手机市场竞争激烈

1. **Hello Android**

* 创建android 项目，填写项目名称，应用名称，包名，和最小支持的android SDK版本(MiniSDK)
* 写java代码（你懂的）setContentView(R.layout.main)….
* 运行，选择Android Application， 打开虚拟机

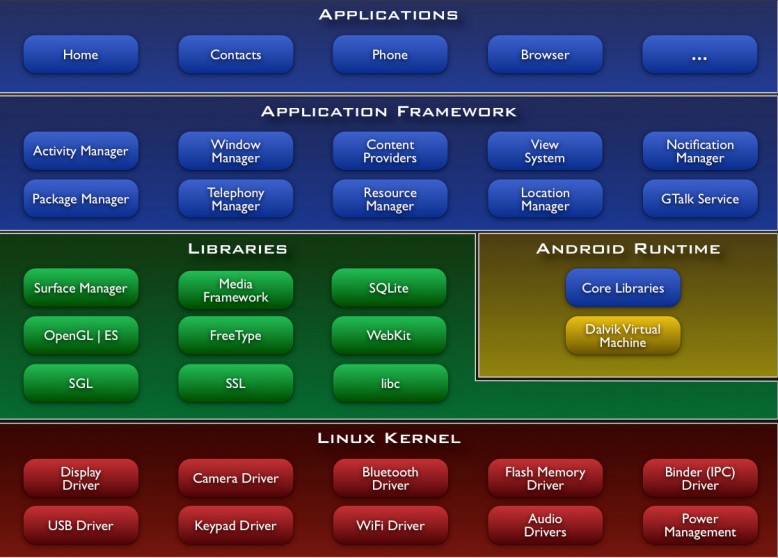
1. **Android 架构**

Application

Application Framework

Libraries & Android Runtime

Linux Kernel



* 1. Linux Kernel
* 基于Linux内核但不是Linux
* 没有图形界面，没有glibc支持
* 提供核心系统服务，如进程，内存，电源管理， 网络， 设备模型，安全性等
* 不包含所有的Linux内核的基础设施
* 打了补丁来支持 Android
  1. Library
* Bionic Libc：定制了libc 的实现，以适合嵌入式应用
* Function Libraries：webkit 浏览器，media框架， SQLite
* Native Servers（本地化服务）：Surface、Audio Flinger（提供把各种“图形，声音”组合在一起的能力
* Hardware Abstraction Libraries（硬件抽象库）：定义Android所需硬件驱动的实现接口，分离Android平台逻辑与硬件接口
  1. Android Runtime

语言：java

虚拟机：Dalvik ，指令集0：Dalvik Executable

Java标准库：编译java代码为Dalvik Executable （dex 格式）

* 1. Application Framework

组成：Activity Manager（管理应用生命周期），Content Provider（应用间共享数据）Resource Manager， Notification Manager(在状态栏显示应用通知)，Views System（构建UI）

* 1. Application

Java 编写

1. **DVM vs JVM**

* Google定制虚拟机的实现：增强可移植性和运行时的一致性；能运行优化过的格式（.dex）和Dalvik 二进制代码；Java的.class/.jar文件在编译时被转换成.dex
* 专门为嵌入式环境设计：支持在每个设备上运行多个虚拟机进程；优化的二进制代码编译器；有效的使用运行时内存。
* 拥有自己的核心库：提供了一个简单而有力的开发平台。
* 在自己的环境中跑程序
* 不受Sun的许可（License）限制
* 谷歌或者潜在的开源社区可以对其进行改进

1. **为什么使用Linux内核：**

* Linux内核是一个久经考验的核心平台
* 可靠性对于手机来说比系统表现更重要
* Linux提供硬件抽象层，使得在下层硬件有改动时上层也不需要改动
* 可以像其他Linux平台一样编写辅助程序

1. **Android 4大组件(概要介绍)：**

* Activity：一个Activity 代表一个用户可见的程序界面。一个程序可以有一个或者多个Activity，每个Activity被画在一个默认的窗口中，通过setContentView（）方法，将可见的内容赋给当前的Activity，并显示在窗口中。
* Service：Service没有可见的界面，是在一段不确定的时间内在后台运行，来完成应用程序的需求或是提供给其他应用程序的功能。与Activity等其他组件相同，Service也是运行在程序的主线程中的。
* Broadcast Receiver：用于接收广播信息并向其反馈。没有界面，可能会启动一个Activity来处理它们接收到的广播信息，或者是使用Notification Manager 来告知用户。
* Content Provider：通过一个特殊的程序数据集合与其他程序共享数据。通过继承ContentProvider类实现一系列方法来允许其他程序存储或者读取属于它的数据。不过，应用程序无法直接调用这些方法，必须使用ContentResolver对象来调用。

1. **进程与线程**

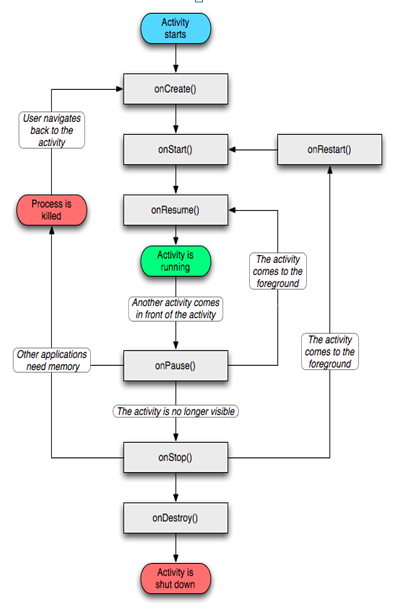
Linux内核开启一个只有一个线程的进程，来运行一个应用程序。这个进程由manifest 文件控制。所有的组件都在这个主线程中被实例化，并且系统对这些组件的调用请求都是从这个主线程中发出的。

Android会关闭一个进程，当内存不足并且被其他更需要被及时响应的应用程序请求时。当关闭进程时，Android系统会考虑这些线程对于用户的重要性。

虽然一开始的时候，系统只是分配的一个只有一个线程的进程来启动一个应用程序，但是我们可以在这个进程中生成其他的线程来执行一些后台的工作。

1. **Activity：**

生命周期：

****

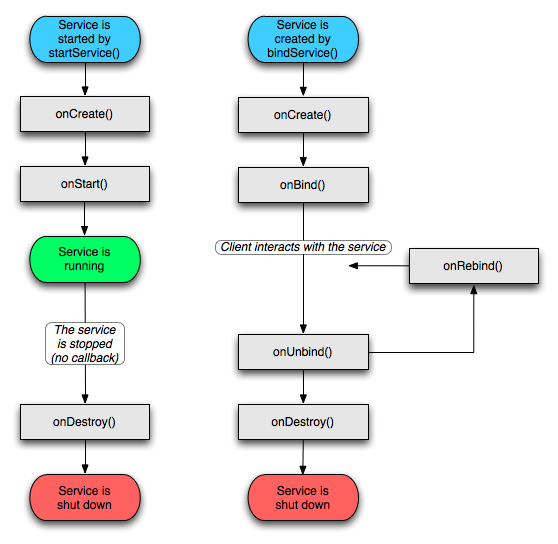
Activity 是一个可以通过提供屏幕界面来与用户交互的应用程序组件。

启动Activity：调用startActivity(intent)方法。传递的参数Intent定义了要启动的具体的Activity，并包含的需要执行的行为，和传递给新的Activity的数据。调用startActivityForResult(intent，requestCode) 方法，可以在新启动的Activity运行结束并返回后，获得该Activity的返回数据。通过requestCode标示该请求。通过使用onActivityResult()方法来处理接收的到返回数据。

保存Activity状态：实现onSaveInstanceState()，系统在Activity可能被销毁之前，调用该方法，并将Activity的数据保存在一个Bundle对象中。这个Bundle保存了（name，value）的数据对。如果系统销毁了一个Activity，而用户又通过返回操作，请求回到这个Activity中，那么系统将会重新调用onCreate()方法，并将保存的数据重新载入到Activity中。如果没有实现onSaveInstanceState()方法，系统将调用默认的onSaveInstateState()方法，通知界面上的所有View，将它们认为需要保存的数据保存起来。

1. **Service**

生命周期：



一个Service可以被启动，运行，直到有人停止它或是它自己停止。

客户端建立一个与Service的连接，并且通过连接来调用该服务。

1. **Broadcast Receiver**

生命周期：当有一个广播信息发送到接收者时，Android 调用onReceive()方法，并传递给它一个包含了这个广播消息的Intent对象。Broadcast Receiver 只有在执行这个方法时才是活跃的，onReceiver()方法返回后，就是闲置的。

通过Context.registerReceiver()来动态的注册一个Broadcast Receiver，或者在Manifest文件中加入<receiver>标签

广播分类：一般广播（Context.sendBroadcast()）--完全异步；有顺序的广播（Context.sendOrderedBroadcast()）--每个接受者按顺序执行处理，可以将一个处理广播信息的结果传递给另一个接受者（receiver），也可以直接抛弃结果来终止整个广播。

1. **进程和生命周期：**

重要性最低的那个进程将被第一个淘汰，然后依次类推。

共有5个等级的进程（重要性由高到低）：前台进程，有可视化界面的进程，服务进程，后台进程，空进程。

1. **UI**

树形（层级）结构：



包括View和ViewGroup， View是具体的界面组件：如Button，TextView等，ViewGroup是Layout，view的容器。

Android绘制界面：首先会找到层级结构中的根节点，从根节点开始，按照顺序分别调用每个View的draw()方法在可用的区域内画出自己。（先画Layout，然后画其中的View）。

画Layout：两个过程：计算和布局过程。计算过程将Layout中的所有子元素的大小都进行计算，并记录下来，布局过程是根据计算过程的数据将这个Layout放置在屏幕的合适位置。

setOnContentView（R.layout.main）将XML布局文件设为Activity的界面

1. **各个Layout**

FrameLayout：单纯的将一个View放在自己的左上角，如果有其他的View，将覆盖之前的View。

LinearLayout：线性布局，有水平和垂直两种。

TableLayout：表格布局，TableRow代表一行。

RelativeLayout：相对布局，根据View对应于屏幕的位置和与其他View的位置关系来布局

AbsoluteLayout：绝对布局，根据View与屏幕上边距和左边距来定位View。

1. **构建定制的界面组件**

可以创建自己的View界面组件，也可以组合多个View成一个界面组件，也可以修改系统的组件。并且还可以捕获其他时间，并定义自己的处理方式。

继承View父类，重写父类方法，如onDraw(), onMeasure(), onKeyDown()等。onDraw()方法实现了绘制该界面组件。onMeasure()方法定义了界面组件的大小。

创建ViewGroup和View类似。

1. **Android数据存储方式：**

* Shared Preferences：以（key, value）对存储私有的简单的数据。

在不同的Activity中共享

* Internal Storage：在设备内存中存储私有数据。

例如：Android中的文件，并且这个文件是该程序私有，程序卸载后，文件消失。

* External Storage：在一个共享的外部存储器中存储共有数据，如SD卡。

外存中的数据可以被用户自己删除。在外存中写入的cache文件，在程序被卸载后，将被删除。其他文件不会。

* SQLite-Database：在私有的数据库文件中存放结构化的数据。openDatabase(), SQLiteOpenHelper对象，getWritableDatabase(), getReadableDatabase()，获得写或者读的权限，query()进行简单数据读写。放在SD卡中的数据库文件是可以被其他程序读取的，只要知道数据库文件的位置。
* Network Connection：在你的网络服务器中存储数据。包含java.net.\*和android.net.\*

1. **Content Provider**

存储和检索数据，并是数据可以在多个程序中使用。这是程序间共享数据的唯一方式。

对于某些特殊的Content Provider，在请求获得它的数据时，需要特殊的权限声明。

你可以创建自己的Content Provider来共享自己的数据。

对Content Provider的数据操作，绝大多数是通过Content Resolver对象来实现。

在Android系统中，每个Content Provider只有一个实例，可以通过多个Content Resolver来读取它。

存储的数据结构是类似于数据库的表。通过\_ID标示。

每个Content Provider都有一个URI来标示自己的数据集。

Uri格式举例：android.provider.Contacts.Phone.CONTENT\_URI；

通过ContentResolver.query()方法获得数据集。Content Resolver 还提供对数据的增删改查操作。



1. **Notification**

一个Notification是在状态栏上显示的一个简短的消息。

用户点击了某个Notification后，就会执行一个与它相关联的Activity。

Notification Manager 通知用户在后台发送的事件（即产生Notification）

通过Notification Manager的Notification()、notify()方法来产生Notification。

通过setLastestEventInfo()来更新Notification状态。

通过cancel()或者cancelAll()来取消Notification。

1. **Location Service**

过程：

* 获得系统服务Location Manager：getSystemService（LOCATION\_SERVICE）
* 在Manifest 文件中定义权限。
* 获得service provider：getAllProvider()或者getBestProvider()
* 继承LocationListener接口
* 通过service provider和 LocationListener，调用requestLocationUpdates()来获得位置数据。

Geocoder：

简单说来，就是根据某个key寻找地理位置的坐标，这个key可以是地址，这个key还可以是ip地址。通过位置获得经纬度，也可以反向定位输入经纬度以后转换成地理位置。

MapView：地图界面（谷歌提供的）有Road View和Aerial View

1. **Sensor**

过程：

Android 的传感器是由外部的服务控制，并且只向它们的的服务发送事件。

一个应用程序向SensorManager注册一个回调函数来接受传感器事件。

每个传感器都有一个相关的XXXListener，程序的回调函数必须实现这个Listener。

1. **Android Test**

http://tech.sina.com.cn/s/2009-12-04/16071161345.shtml

1. **移动开发过程：**

* 为你的移动项目选择一种合适的软件开发方法
* 理解目标手机设备将怎样实现你应用程序的功能
* 持续进行全面、精确地可行性分析
* 使用试验性的手机设备降低风险
* 通过配置管理跟踪手机功能的变化
* 在内存受限制的系统上设计稳定且响应性强的应用程序
* 设计能够适用于不同手机设备且具有不同用户体验的用户界面
* 在目标设备上对应用程序进行全面、彻底的测试
* 积极引入第三方需求，以为应用程序的销售带来便利
* 部署和维护移动应用程序

1. **Android网络与通信**

三种接口：标准java接口（java.net.\*）,Apache接口（org.apache）,Android网络接口（Android.net.\*）

* Http通信：

HttpURLConnection接口（标准java接口）：

URL url = new URL(“<http://www.google.com>”);

HttpURLConnection urlConn = (HttpURLConnection)url.openConnection();

HttpClient接口（Apache接口）：

HttpClient httpclient = new DefaultHttpClient();

HttpResponse httpResponse = httpclient.execute(httpRequest);

* Socket通信



* WebView浏览网页：

WebSettings：设置WebView属性；WebViewClient：辅助处理各种事件；WebChromeClient：辅助WebView处理javascript。

* Wifi：

ScanResult：描述检测出的接入点；WifiConfiguration：wifi网络的配置，WifiInfo：无线连接的描述；WifiManager：管理wifi连接的API。

* 蓝牙：

权限：在manifest文件中添加蓝牙权限

过程：请求开启蓝牙，打开蓝牙，请求能被搜索（服务器），搜索蓝牙设备（客户端），通过socket建立服务器和客户端连接，数据传输，关闭蓝牙

* NFC（近距离无线通讯技术）

NFC技术可以让你在一个NFC Tag和一个Android设备或者两个Android设备间共享小型有效负载数据。

1. **什么是线程池，为什么使用线程池**

一般一个简单线程池至少包含下列组成部分。

* 线程池管理器（ThreadPoolManager）:用于创建并管理线程池
* 工作线程（WorkThread）: 线程池中线程
* 任务接口（Task）:每个任务必须实现的接口，以供工作线程调度任务的执行。
* 任务队列:用于存放没有处理的任务。提供一种缓冲机制。

Why？：

* 减少了创建和销毁线程的次数，每个工作线程都可以被重复利用，可执行多个任务
* 可以根据系统的承受能力，调整线程池中工作线线程的数目，防止因为因为消耗过多的内存，而把服务器累趴下(每个线程需要大约1MB内存，线程开的越多，消耗的内存也就越大，最后死机)

1. **进程间通讯**

* Bound Service：

一个BoundService 相当于C/S模式下的服务器，他可以接受其他组件(如Activity)的连接，然后可以发请求，接受回复，还可以进行IPC（interprocess communication）。

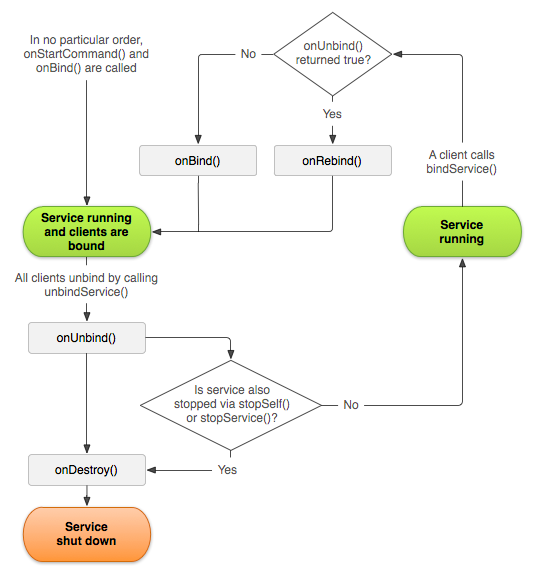
一个BoundService为客户端服务，不会无定期的运行在后台。

一个BoundService是Service的子类，它必须实现onBind()方法，返回一个IBinder对象给客户端。

客户端继承ServiceConnection接口，使用bindService()来绑定一个服务。通过onServiceConnected()获得服务端的IBinder对象，通过IBinder对象与服务交互。

多客户端情况，但是系统调用Service的onBind()方法并生成IBinder对象只在第一个客户端请求绑定时，其他客户端绑定时，直接返回已经生成的IBinder对象。

当所有的客户端解除与服务的绑定后，系统销毁service。



* IPC：

通过使用Message：

服务继承Handler接口用于响应每个客户端的请求；

Handler创建一个Messenger（发信人）对象；

Messenger对象创建IBinder对象，可以返回给客户端调用；

客户端用这个IBinder对象来实例化Messenger对象，用来向Service发送Message对象；

Service在Handler中接受Message，并用handleMessage()处理。

* IBinder

transact()方法，发送请求到IBinder，并且在Binder对象中接受到请求，是同步的过程，等待Binder.onTransact()方法返回。

使用的数据格式是：Parcel，定义了操作类型，数据等。

* AIDL：

AIDL是一种能够产生使得一个以Android为系统的设备上的进程间能够通过IPC通信的代码的接口定义语言。

基本使用步骤

创建aidl文件（定义接口）

编译AIDL文件（Eclipse会自动完成）

在Java文件中, 实现AIDL中定义的接口（假设为服务端）

实现向外（假设称为客户端）揭露所定义的接口以及实现机制使得客户可获得该接口

运行过程：

当AIDLServer运行后，启动了由AIDL（IMyService.aidl）文件定义的服务

运行AIDLClient之后，通过bindService绑定服务，获得可服务对象(myService)，及AIDLServer中类MyService中onBind函数返回的对象实例，该实例是aidl文件产生的Stub的类的一个实例

获得myService对象实例后，即可调用aidl文件中定义的方法来获得或者传递数据

* Android Binder机制

1. **Intent**

当有请求从Content Resolver发给Content Provider时，Content Provider才是有效的。Service， Activity 和Broadcast Receiver 通过一种叫Intent的异步消息激活（启动） 。

对于Activity 和Service来说，它们对请求的行为进行命名，并且标示需要的数据的URI。

* 启动Activity的方式：startActivity(); 或者startAcvitityForResult(); 启动的Activity可以通过使用getIntent()来获得启动该Activity的Intent。在Android中，Activity通过onNewIntent()方法来传递随后产生的Intents。
* 启动Service的方式：通过传递Intent参数给Context.startService()方法来启动Service。首先调用Service的onStart()方法，随后传递给该Service启动它的Intent。同样的，可以通过Context.bindService()来创建一个从调用端到启动的服务端的连接。而该服务将会在调用onBind()方法时收到Intent对象。
* 启动Broadcast的方式：调用Context.sendBroadcast(), Context.sendOrderedBroadcast( ) 或者Context.sendStickyBroadcast() 来发送广播。Broadcast Receiver通过onReceiver()方法获得广播。
* Activity调用finish()方法来结束自己。一个Activity能通过调用finishActivity()来结束由它startActivityForResult()产生的Activity。
* 调用Service的stopSelf()方法来结束自己。或者也可以调用Context.stopService();

Intent中的数据包括：

* Action：例如ACTION\_VIEW, ACTION\_EDIT, ACTION\_MAIN等
* Data

其他：category，type，component，extras

Start Activities and Getting Results :

