

1G: 模拟制式手机: 就是大哥大, 手机类似无线电台

2G: GSM, CDMA等数字手机 (2G), 具有稳当的通话质量、支持各种各样的彩信业务GPRS和网WAP业务

3G: 移动通信技术 (3G): 第三代移动通信技术. 由于网速的提升, 能够处理图像、音乐、视频等.

4G: 该技术包括TD-LTE和FDD-LTE两种模式. 相比3G网速提升比较多. 基本能够满足无限服务的要求

2G模式:

1. GSM: 移动/联通
2. CDMA: 电信

3G模式:

1. WCDMA: 联通 欧美标准
2. TD-SCDMA: 移动 中国自己标准
3. CDMA2000: 电信

4G制式:

1. FDD-LTE: 电信/联通 (分频双工)
2. TD-LTE: 移动/电信/联通 (分时双工)

TD-LTE是我国自主研发的4G标准,是由TD-SCDMA (3G网络) 发展而来. LTE FDD是现在国际上主流的, 使用最广泛的4G网络, 现在全球有超过200个LTE的商用网络, 其中超过90%是FDD的, 从技术上说, TD-LTE采用的是时分双工, 而LTE FDD采用的是频分双工

通信系统的工作方式分为: 单工、半双工和全双工

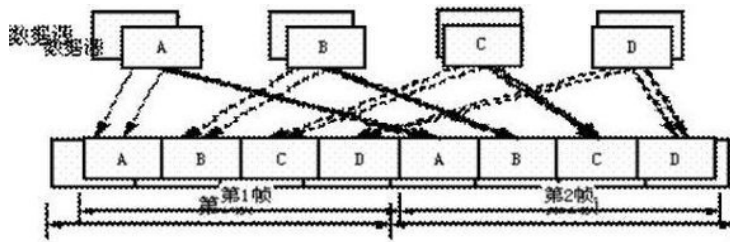
- 单工就是信息只能向一个方向传播。例如寻呼机和收音机, 只能接收信息, 不能发出信息。
- 半双工就是信息可以双向传播, 但是上传信息的时候只能上传, 下载的时候只能下载。例如对讲机, 你说话的时候听不见别人别人说, 听别人说的时候自己不能说。
- 全双工就是信息可以同时双向传播。例如手机, 可以边听边说。
- 其中全双工又分为: 时分双工TDD, 与频分双工FDD。

所谓的频分双工就是将信息上传和信息下载放在两个不同的频段, 称为上行频段和下行频段, 且这两个频段必须对称。为了不防止上下行频段之间的信息串频, 两个频段不能重叠, 而且中间必须隔开一段, 称为保护频。如下图: 上行和下行频段相互分开



所谓的时分双工就是将上传和下载放在同一个频段, 也就是上行频段和下行频段完全一样。那它是如何做到上下的信息不串频呢? 其实很简单, 顾名思义, 频分双工分的是频段, 那时分双工分的就是时间。将波传播的时间轴一分为二,

前半部分用于信息的上传, 后一部分用于信息的下载。其实这从理论上更像是同步的半双工, 但是由于上行和下行时间差距极短, 我们无法感觉到, 所以从效果上也是全双工。如下图: 时间帧的第一帧为上行, 第二帧而为下行, 上下行共用一个频段, 用时间的差将他们隔开。



接下来说一说移动TD- LTE和电信联通LTE FDD的优缺点，其实就是时分双工与频分双工的特点
TD- LTE由于采用上行和下行分时，所以上行和下行的时间差导致他信息传输的速度受到一定的限制。
他的理论下载峰值为100Mbps，相比FDD的150Mbps来说，慢了不少。当然TD- LTE的基站信号覆盖半径也比LTE FDD小一点

LTE FDD也并非都是优点。FDD制式必须要找对称分开的上行、下行频段，也就是说它必须浪费更多的频段资源。那中国联通的FDD网为例，工信部给他发的频段牌照为，上行1755~1765MHz和下行1850~1860MHz（2×10M），实际联通只得到了上下各10M的频段，但是加上当中的保护频段，一共占用了1755~1860MHz也就是105MHz的频段。这就导致了频段资源利用率低，是一种资源的浪费。目前，频率资源本身就紧张，能找到对称的上行和下行频段就尤其难。频率除了用在手机上网通讯上，像广播，军事，甚至无线路由器等民用领域也广泛使用。将来随着科技的进步，频率频段的稀缺会日益严重

TD-LTE和FDD-LTE 是4G的两种国际标准，各有利弊。TD-LTE占用频段少，节省资源，带宽长，适合区域热点覆盖；FDD速度更快，覆盖更广，但占用资源多。适合广域覆盖

Web App, Native APP,Hybird App介绍 (重要)

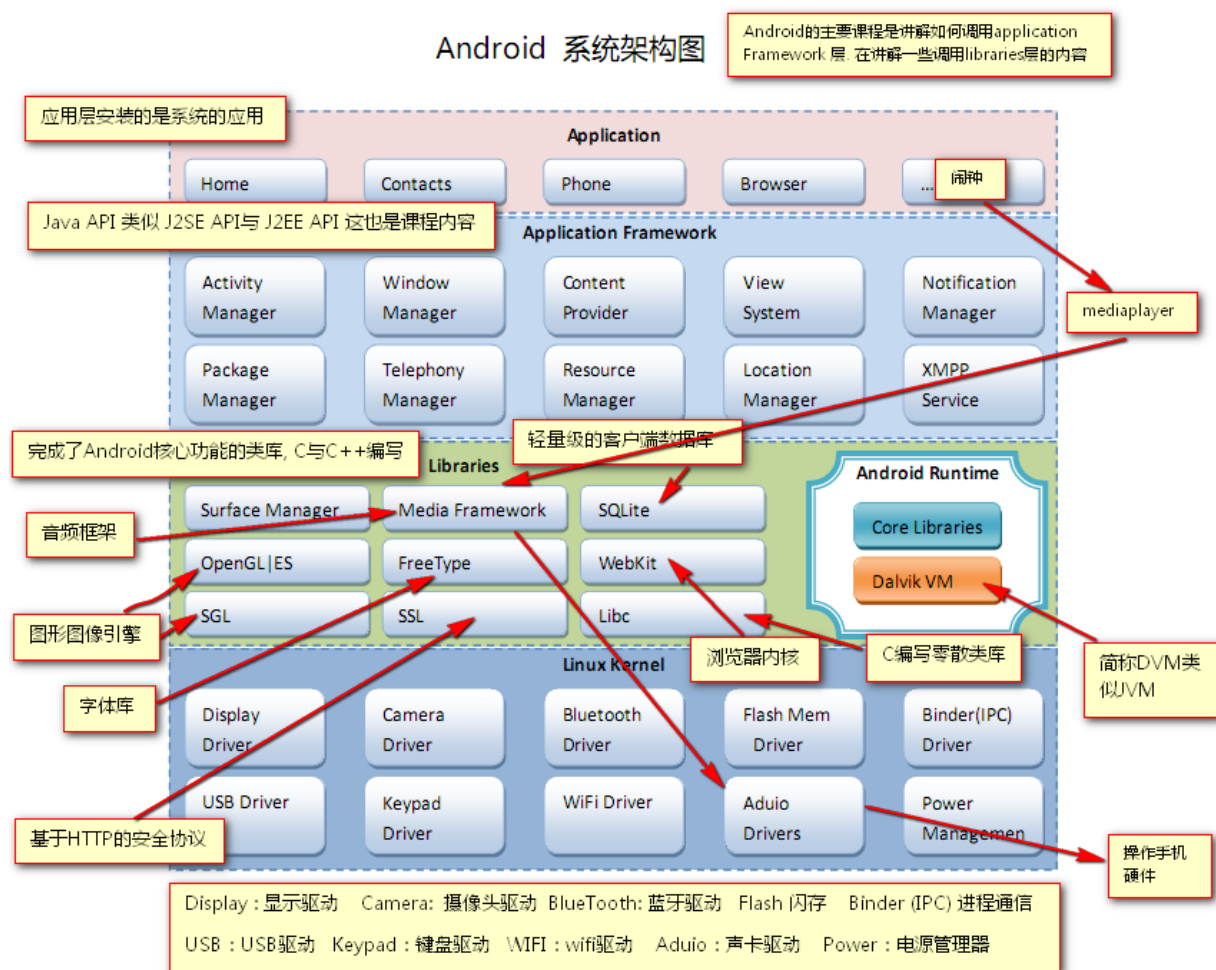
Web App : 用户无需下载，通过不同平台的浏览器访问来实现跨平台，同时可以通过浏览器支持充分使用HTML5特性，缺点是这些基于浏览器的应用无法调用系统API来实现一些高级功能，也不适合高性能要求的场合

Native APP : 就是所谓的原生应用.指的是用平台特定的开发语言所开发的应用.使用它们的优点是可以完全利用系统的API和平台特性，在性能上也是最好的

Hybird App : 则是为了弥补如上两者开发模式的缺陷的产物.分别继承双方的优势.首先它让为数众多的web开发人员可以几乎零成本的转型成移动应用开发者；其次，相同的代码只需针对不同平台进行编译就能实现在多平台的分发，大大提高了多平台开发的效率；而相较于web App，开发者可以通过包装好的接口，调用大部分常用的系统API。PhoneGap正是Hybird APP的代表开发框架

Android的体系结构

了解 Android 几大体系的构成, 知道学习的定位与范围, 然后通过一个闹钟的案例, 把相关的内容都串起来操作

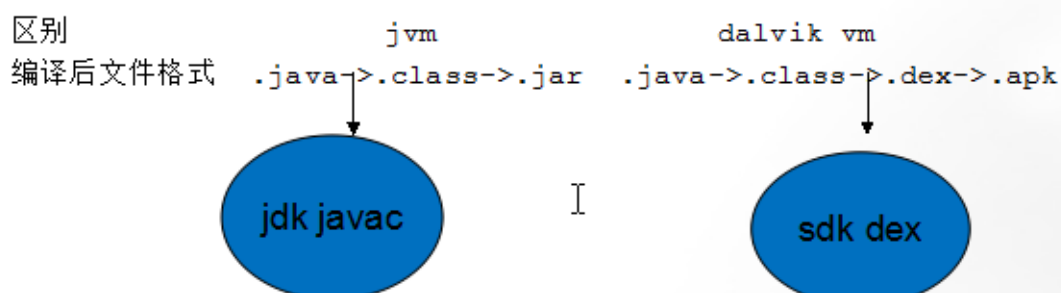


DVM和JVM的区别 : (可以搜索Sun与Google打官司的背景)

1. DVM: Dalvik Virtual Machine
2. JVM: Java Virtual Machine

Dalvik是Google公司自己设计用于Android平台的Java虚拟机。Dalvik虚拟机是Google等厂商合作开发的Android移动设备平台的核心组成部分之一。它可以支持已转换为 .dex (即Dalvik Executable) 格式的Java应用程序的运行.dex格式是专为Dalvik设计的一种压缩格式, 适合内存和处理器速度有限的系统。Dalvik 经过优化, 允许在有限的内存中同时运行多个虚拟机的实例, 并且每一个Dalvik 应用作为一个独立的Linux 进程执行。独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭

Dalvik 基于寄存器, 而 JVM 基于栈 (可以演示 .jar 与 apk 作为对比)



Android 硬件比IOS要好, 但是为什么运行起来没有IOS流畅

- IOS系统本身就是由Object C编写的. Iphone的应用也是采用Object C 编写. 因此称为目标程序
- Linux是由C于C++编写的, 但是要去运行Java编写的程序. 那么在Dalvik 应用每次运行程序的时候字节码都需要及时编译转化为机器码. 这样会拖慢应用的运行效率
- ART (Android Runtime): 应用在第一次安装的时候字节码就会先编译成机器码. 则成为真正的本地应用, 这样会导致安装体积增加但是启动速度和执行速度会显著上升

ART模式在Android4.0系统时作为可选项, 但是在Android5.0系统后作为标准配置了

1. 更多--->关于设备--->Android版本. 可以查看到Android的版本
2. 在Android 4.x的版本如何切换到ART格式 <http://www.orsoon.com/news/azjc/22195.html>



MyEclipse10.0 安装 ADT插件 (采用link的方式安装, 安装重启myeclipse之后会提示安装sdk, sdk不要乱更新, 否则又会导致adt的更新)

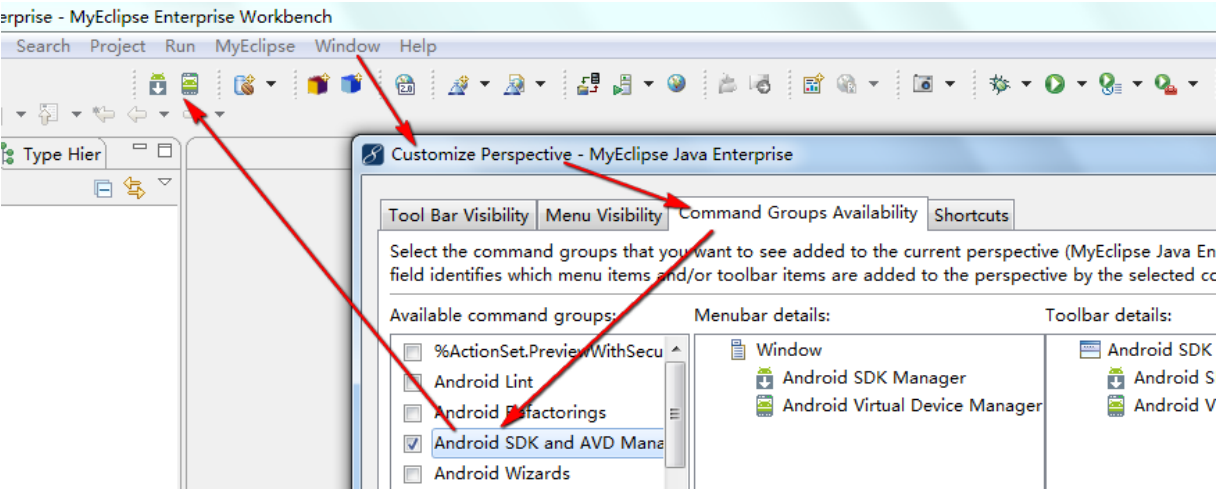
- Eclipse ADT 是 Eclipse 平台下用来开发 Android 应用程序的插件。
- 我们自己的程序要依靠java基本的类和API来编写, 所以要JDK。
- 要用到各种Android平台, 好比android2.3等, 便要有SDK。

ADT(Android Development Tools): 目前Android开发所用的开发工具是Eclipse, 在Eclipse 编译IDE环境中, 安装ADT, 为Android开发提供开发工具的升级或者变更, 简单理解为在Eclipse下 开发工具的升级下载工具。adt只是一个eclipse的插件, 里面可以设置sdk路径. 如果你不用Eclipse作为你的开发工具, 你就不需要下载ADT, 只下载SDK即可开发。SDK可以自己编译

SDK(Software Development Kit): 一般是一些被软件工程师用于为特定的软件包、软件框架、硬件平台、操作系统等建立应用软件的开发工具的集合。在Android中, 他为开发者提供了库文件以及其

他开发所用到的工具。简单理解为开发工具包集合，是整体开发中所用到的工具包

注意：如果配置完SDK后.如果在MyEclipse中没有显示SDK相关图标则参考如下步骤



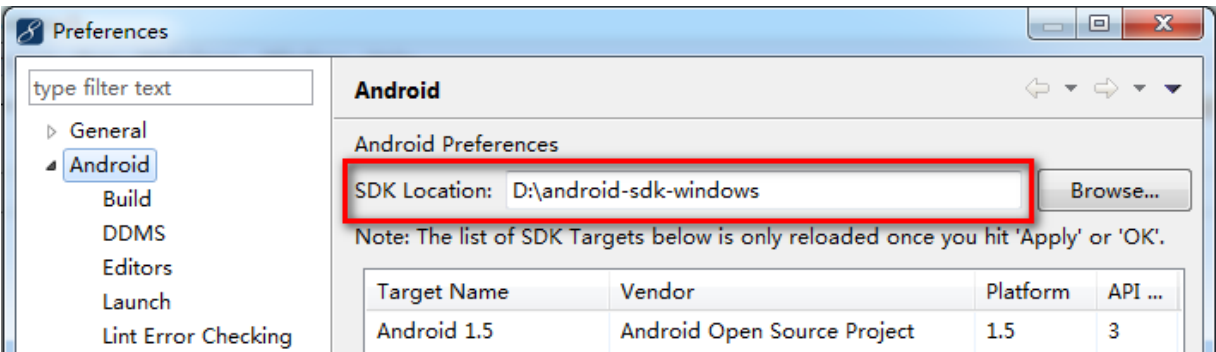
Android平台与SDK Tools版本、ADT版本的对应关系

Android平台	SDK Tools的版本	ADT版本
Android 2.2	R7	ADT-0.9.9
Android 2.3	R8	ADT-8.0.1
Android 3.0 预览版	R9	ADT-9.0.0
Android 3.0	R10	ADT-10.0.0
Android 3.1	R11	ADT-11.0.0
Android 3.2	R12	ADT-12.0.0
Android 4.0	R14	ADT-14.0.0

当下载完毕SDK之后,通过在MyEclipse安装ADT插件,如果插件安装成功会在MyEclipse工具中出现:



如果此时的SDK与MyEclipse没有设置关联.这可以通过如下菜单配置相关的SDK(注意ADT与SDK 版本要一一对应

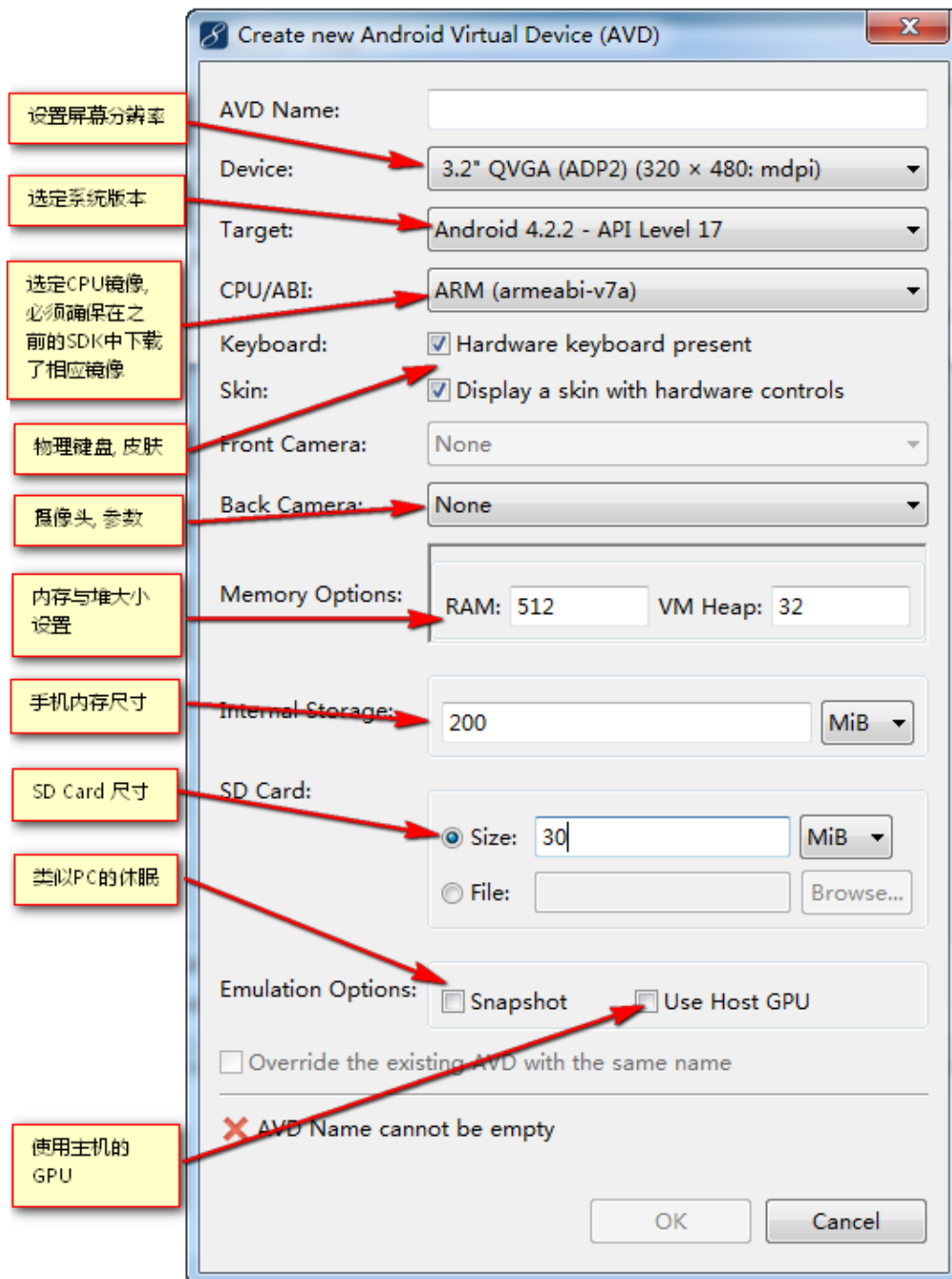


SDK下载完毕之后结构图如下，不一定所有的版本都需要下载,刚刚开始学习的时候下载相应的 Android 4的版本就好

Tools				
Android SDK Tools	22.0.1		Update available: rev. 24.0.2	Android开发需要的设备,例如模拟器
Android SDK Platform-tools	17		Update available: rev. 20	
Android SDK Build-tools	19.1		Not installed	Android开发的常用名称,例如ADB
Android SDK Build-tools	19.0.3		Not installed	
Android SDK Build-tools	19.0.2		Not installed	
Android SDK Build-tools	19.0.1		Not installed	
Android SDK Build-tools	19		Not installed	
Android SDK Build-tools	18.1.1		Not installed	
Android SDK Build-tools	18.1		Not installed	
Android SDK Build-tools	18.0.1		Not installed	
Android SDK Build-tools	17		Installed	
Android 4.4.2 (API 19)				
Android 4.3 (API 18)				
Android 4.2.2 (API 17)				
Documentation for Android SDK	17	2	Installed	Android API
SDK Platform	17	2	Installed	API的调用案例
Samples for SDK	17	1	Installed	系统镜像(手机内核)
ARM EABI v7a System Image	17	2	Installed	
MIPS System Image	17	1	Installed	扩展 API例如地图
Google APIs	17	3	Installed	
Sources for Android SDK	17	1	Installed	API的源码包
Android 4.1.2 (API 16)				
Android 4.0.3 (API 15)				
Android 4.0 (API 14)				
Android 3.2 (API 13)				
Android 3.1 (API 12)				
Android 3.0 (API 11)				
Android 2.3.3 (API 10)				
Android 2.2 (API 8)				
Android 2.1 (API 7)				
Android 1.6 (API 4)				
Android 1.5 (API 3)				
Extras				
Android Support Library	13		Installed	Intel 系统镜像的加速器, 用来启动模拟器加速
Google AdMob Ads SDK	11		Installed	
Google Analytics Sdk	2		Installed	
Google Play APK Expansion Library	3		Installed	
Google Play Billing Library	4		Installed	
Google Market Licensing	2		Installed	
Google USB Driver	7		Installed	
Google Webdriver	2		Installed	
Intel x86 Emulator Accelerator (HAXM)	3		Installed	

当SDK环境搭建完毕之后,则可以创建AVD模拟器了(Android Virtual Device)

作业: 创建两个模拟器,测试相互之间是否能通话



Java 内存区域划分: <http://www.cnblogs.com/dolphin0520/p/3613043.html>

启动加载的流程: <http://blog.csdn.net/hmzdbql/article/details/8097172>

知识补充: Java内部类几种实现方式

1. 成员内部类
2. 静态类部类
3. 匿名类部类
4. 局部类部类

成员内部类: 把内部类看成为外部类的普通成员, 在创建内部类时要先创建外部类

```
1. public class OutClass {  
2.  
3.     private int num = 1;
```

```
4.
5.     public class InnerClass {
6.
7.         private int num = 2;
8.
9.         public void showData() {
10.             int num = 3;
11.             System.out.println("showData.num:" + num);
12.             System.out.println("InnerClass.num:" + this.num);
13.             System.out.println("OutClass.num:" + OutClass.this.num);
14.         }
15.     }
16.
17.     public static void main(String[] args) {
18.         // 在创建内部类时要先创建外部类
19.         OutClass.InnerClass innerClass=new OutClass().new InnerClass();
20.         innerClass.showData();
21.     }
22.
23. }
```

静态内部类：把内部类看成为外部类的普通成员。在创建内部类时要先创建外部类

```
1.     public class OutClass {
2.
3.         private static int num1 = 1;
4.         private int num2 = 2;
5.
6.         public static class InnerClass {
7.
8.             private static int num1 = 11;
9.             private int num2 = 22;
10.
11.             // 静态类部类的静态方法
12.             public static void showData() {
13.                 int num1 = 111;
14.                 System.out.println("showData.num:" + num1);
15.                 System.out.println("InnerClass.num:" + InnerClass.num1);
16.                 System.out.println("OutClass.num:" + OutClass.num1);
17.             }
18.
19.             // 静态类部类的普通方法
20.             public void showData2() {
21.                 int num2 = 222;
22.                 System.out.println("showData.num:" + num2);
23.                 System.out.println("InnerClass.num:" + this.num2);
24.                 // 思考为啥不能调用OutClass.this.num2
25.                 // System.out.println("OutClass.num:" + OutClass.this.num2);
26.             }
27.         }
28.
29.         public static void main(String[] args) {
30.             OutClass.InnerClass.showData();
31.             OutClass.InnerClass inner=new OutClass.InnerClass();
32.             inner.showData2();
33.         }
34.     }
```



```

33.     }
34.     }

```

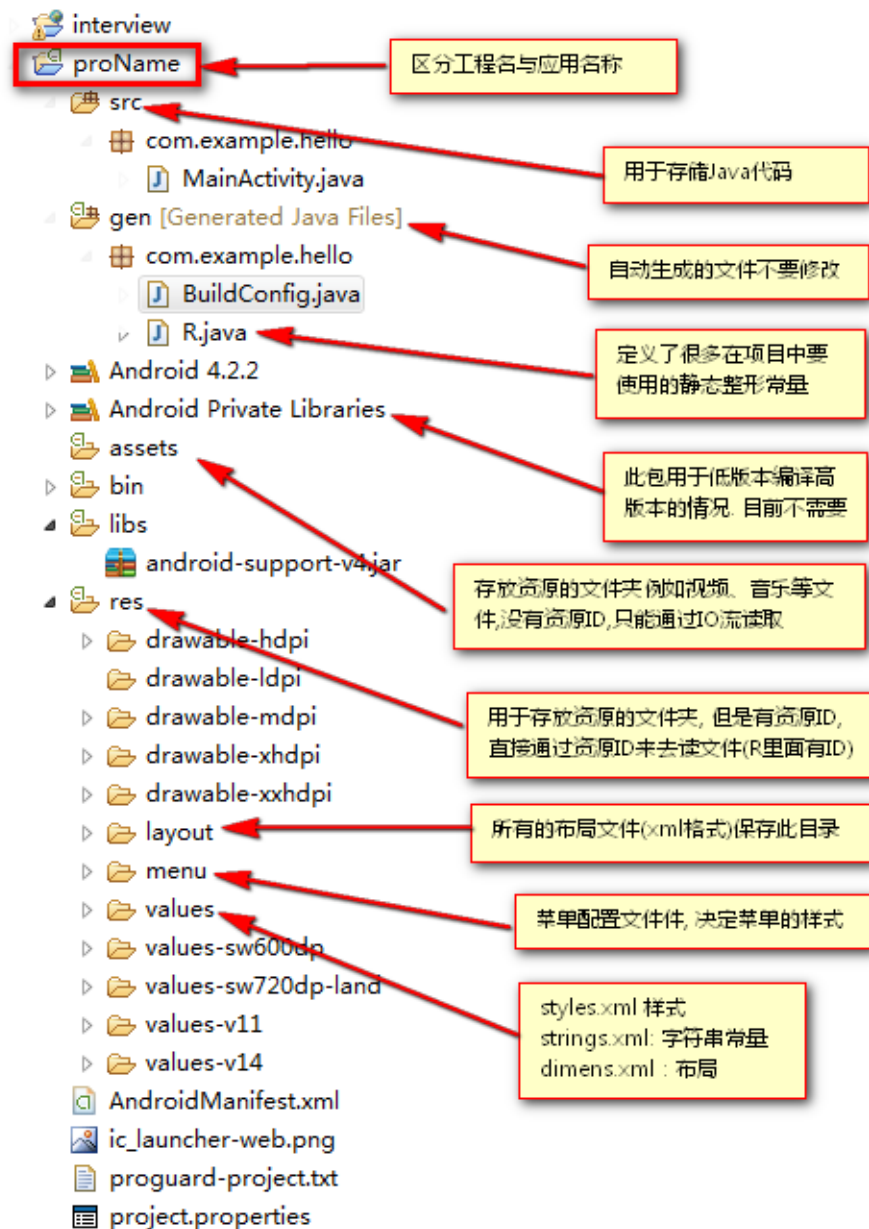
匿名内部类:

```

1.  public static void main(String[] args) {
2.      String[] names = new File("C:/test").list(new FilenameFilter() {
3.          @Override
4.          public boolean accept(File dir, String name) {
5.              return name.endsWith(".jar");
6.          }
7.      });
8.      for (String temp : names) {
9.          System.out.println(temp);
10.     }
11. }

```

作业: 创建一个Android Application 然后部署到AVD中 (搞清楚应用名称,与项目名称的区别) 例如自己的4.0.3已经下载全



AndroidManifest.xml 清单文件分析

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <!-- 1: package: 它是在系统中的唯一名称.如果修改了此名称,相同的程序可以部署多
    次.默认与代码的包名相同,但是没有必然联系
3.      2: versionCode: 版本号, 低版本的微信可以覆盖低版本的微信,如果package一
    直则通过versionCode确定高低版本
4.      3: versionName: 带外的版本名称. 是给人看的,系统看versionCode
5.  -->
6.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
7.      package="com.example.hello"
8.      android:versionCode="1"
9.      android:versionName="1.0" >
10.  <!--
11.      最低支持的版本: 10 默认就是android 2.3
12.      目标的SDK: 新版本已经删除了这两个属性
13.  -->
14.  <uses-sdk
15.      android:minSdkVersion="10"
16.      android:targetSdkVersion="17" />
17.  <!--
18.      1: icon: 指定应用的图标 在XML中访问R中的资源
19.      2: label: 指定的是应用程序管理列表的名称
20.      3: activity ==> android:label
21.  -->
22.  <application
23.      android:allowBackup="true"
24.      android:icon="@drawable/ic_launcher"
25.      android:label="@string/app_name"
26.      android:theme="@style/AppTheme">
27.  <!-- 如果这里没有设置,则 使用上面appliation标签中的名称-->
28.  <activity
29.      android:name="com.example.hello.MainActivity"
30.      android:label="@string/app_name" >
31.  <!-- 如果activity里面有intent-filter则说明当前的activity是入口的
    activity -->
32.      <intent-filter>
33.          <action android:name="android.intent.action.MAIN" />
34.          <category android:name="android.intent.category.LAUNCHER"
35.      />
36.      </intent-filter>
37.      </activity>
38.  </application>
39.  </manifest>

```

可以直接切换到此视图, 查看链接手机的状态 DDMS: Dalvik debug monitor service 虚拟机调试监控服务

ADB: android debug bridge 建立eclipse与android设备之间链接, 配置环境变量: D:\android-sdk-windows\platform-tools

- adb start-server : 启动adb进程
- adb kill-server : 关闭adb进程
- adb install E:\aaa.apk 安装手机应用

- adb uninstall 应用包名
- adb devices: 列出与开发环境链接的Android设备列表

Android四大组件(Activity、service、broadcast receiver、content provider)之Activity介绍

1. 一个Activity通常就是一个单独的屏幕
2. Activity之间通过Intent进行通信
3. Android应用中每一个Activity都必须要要在AndroidManifest.xml配置文件中声明

第一个案例：电话拨号器(通过模拟Android打电话的功能可以看出,打电话需要调用系统两个apk，一个是拨号的apk另一个是call phone的apk)

1. 练习组件的布局
2. 练习事件注册 + 系统意图的使用 + 申请权限

设置拨号页面的布局信息：

```
1.    <TextView
2.        android:layout_width="wrap_content"
3.        android:layout_height="wrap_content"
4.        android:text="@string/phone_txt" />
5.
6.    <EditText
7.        android:layout_width="match_parent"
8.        android:layout_height="wrap_content"
9.        android:inputType="text" android:id="@+id/edit_call" />
10.    <!-- @+id: 向 R内部类 ID中新增加一个变量 -->
11.    <Button
12.        android:onClick="getClickBtn"
13.        android:layout_width="match_parent"
14.        android:layout_height="wrap_content"
15.        android:text="@string/phone_call" />
```

MainActivity中响应事件处理程序

```
1.    public class MainActivity extends Activity {
2.        public void getClickBtn(View v){
3.            EditText et=(EditText)findViewById(R.id.edit_call);
4.            String phone = et.getText().toString();
5.            System.out.println("输入的电话号码为:" + phone);
6.            // 创建意图对象
7.            Intent intent =new Intent();
8.            // 把动作封装到意图对象中
9.            intent.setAction(Intent.ACTION_CALL);
10.           // 设置打电话给谁 tel: 为固定写法
11.           intent.setData(Uri.parse("tel:" + phone));
12.           // 启动意图对象,实现功能或者界面跳转
13.           startActivity(intent);
14.       }
15.   }
```

Intent对Android的核心和灵魂,是各组件之间的桥梁。四大组件分别为Activity、Service、


BroadcastReceiver、ContentProvider。

而这四种组件是独立的，它们之间可以互相调用，协调工作，最终组成一个真正的Android应用
显示网页的意图：

```
1. Uri uri = Uri.parse("http://www.baidu.com");
2. // 此处默认需要http:前缀,如果没有则需要添加
3. Intent it = new Intent(Intent.ACTION_VIEW, uri);
4. startActivity(it);
```

如何在真实手机上调试：

1. 首先到摩托罗拉官网下载xt701驱动安装在计算机上(一些没有驱动的山寨机和安装万能驱动的手机可能无效)
2. 用数据线把手机连接在计算机上(在手机上进行设置:设置-->应用程序-->开发-->USB调试)
3. 在eclipse中打开: window---->show view---->other---->devices 在该窗口中你将会看到你的手机设备信息
4. 确定你的应用程序的运行环境和你的手机系统版本保持一致(如果不一致,修改AndroidManifest.xml文件中的<uses-sdk android:minSdkVersion="7" /> 版本号)
5. 在项目上单击右键: run as--->android application

Name			
 samsung-sch_i959-4d00d959342470fb	Online	5.0.1	
cn.jxy.phone	30558	8600	

屏幕大小：指屏幕的物理尺寸，一般用屏幕对角线长度表示，单位英寸。(用尺量屏幕对角线长度，然后换算成英寸(2.54厘米 = 1 英寸))

- 如 5 英寸屏幕. 1英寸 = 2.54 厘米 $5 * 2.54 = 12.7 \text{ cm}$
- 屏幕大不一定代表清晰度就高。比如说，一个 5 英寸屏幕，分辨率为 800*600，而一个 4.5 英寸屏幕，分辨率为 1280 * 800，这代表了前者屏幕更大，而后者屏幕上的图像更清晰

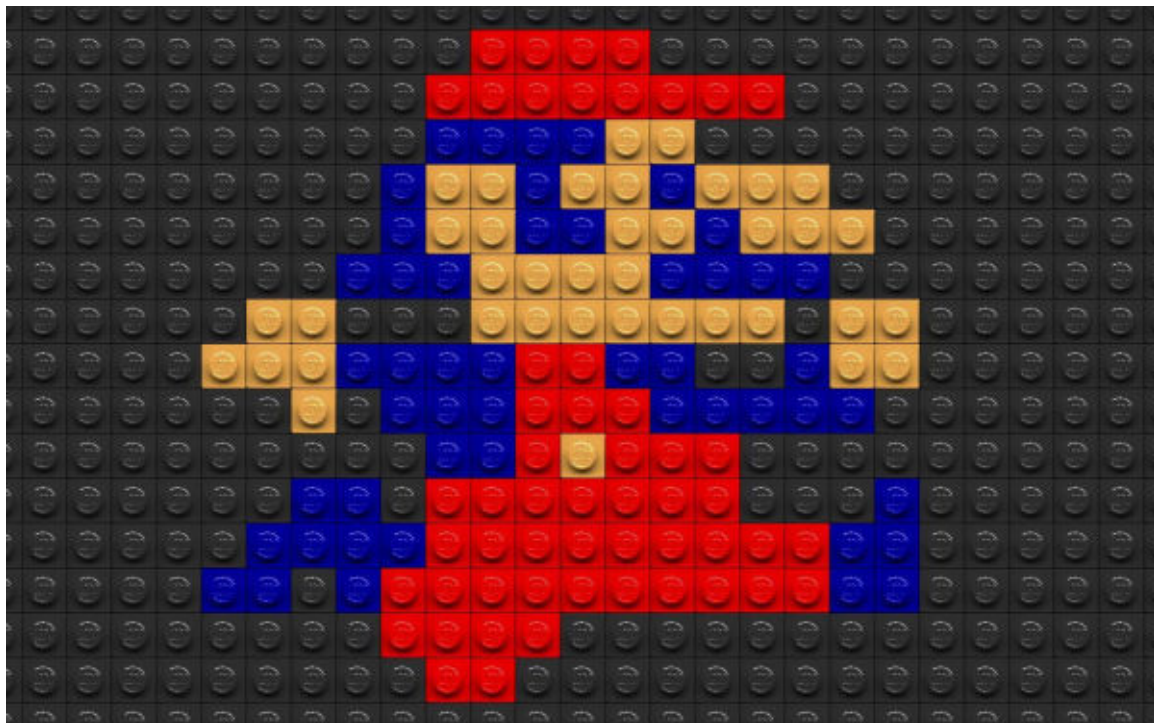


常见机型的几种固定分辨率

- **VGA**: Video Graphics Array，即：显示绘图矩阵，相当于640×480 像素
- **QVGA**: Quarter VGA，即：VGA的四分之一，分辨率为320×240，一般用于小屏手机 像三星盖世Mini S5570就是使用这分辨率

- **HVGA**: Half-size VGA, 即: VGA的一半, 分辨率为480×320, 像三星盖世Ace S5830就是使用这分辨率
- **WQVGA**: Wide Quarter VGA, 即: 扩大的QVGA, 分辨率比QVGA高, 比VGA低, 一般是: 400×240, 480×272
- **WVGA**: Wide Video Graphics Array, 即: 扩大的VGA, 分辨率为800×480像素, 像三星i9000就是使用这分辨率
- **FWVGA**: Full Wide VGA, 数码产品屏幕材质的一种, VGA的另一种形式, 比WVGA分辨率高, 别名: Full Wide VGA, 其分辨率为854×480像素(16:9)

px: 即像素, **1px**代表屏幕上一个物理的像素点, 在**Android**开发中, **px**单位不被建议使用, 因为它是一个固定的单位, 不太适合**Android**开发中的不同手机的图像适配机制, 像素是构成位图的基本要素。



dpi(density-independent pixel): 缩放比例无关的像素, 计算公式如下: (非常重要)

屏幕多少英寸指的是对角线的长度。像素密度是指(以1920×1080, 5英寸为例), 1920和1080的平方和开根号(就是直角三角形斜边长的算法), 开出来等于2202.9, 除以5英寸就得到ppi441左右

例: "HTC One (32GB/单卡/国际版)

4.7英寸屏幕, 分辨率1920x1080 求解像素密度?

解: $\sqrt{1920^2 + 1080^2} = 2202.9071$

$2202.9/5 = 468.7021$ (ppi) ≈ 469 ppi

不同的手机/平板可能具有不同的像素密度, 例如同为4寸手机, 有480x320分辨率的也有800x480分辨率的, 前者的像素密度就比较低。**Android**系统定义了四种像素密度:

1. 低(120 ldpi) low dpi 系数 0.75 就可以得到相应的px数
2. 中(160 mdpi) medium dpi 系数为 1 就可以得到相应的px数

3. 高(240 hdpi) high dpi 系数为 1.5 就可以得到相应的px数
4. 超高(320 xhdpi) extra high dpi 系数为 2 就可以得到相应的px数

系数乘以dp长度就是像素数。手机上的图片最终还是要以像素显示的。例如界面上有一个长度为“80dp”的图片，那么它在240dpi的手机上实际显示为 $80 \times 1.5 = 120\text{px}$ ，在320dpi的手机上实际显示为 $80 \times 2 = 160\text{px}$ 。如果你拿这两部手机放在一起对比，会发现这个图片的物理尺寸“差不多”，这就是使用dp作为单位的，通过模拟器的计算来体会dp与px之间的换算效果

	Nexus 4 (4.7", 768 × 1280: xhdpi)
	Nexus 10 (10.1", 2560 × 1600: xhdpi)
	Nexus 7 (7.3", 800 × 1280: tvdpi)
	Galaxy Nexus (4.7", 720 × 1280: xhdpi)
	Nexus S (4.0", 480 × 800: hdpi)
	Nexus One (3.7", 480 × 800: hdpi)
	10.1" WXGA (Tablet) (1280 × 800: mdpi)
	7.0" WSVGA (Tablet) (1024 × 600: mdpi)
	5.4" FWVGA (480 × 854: mdpi)
	5.1" WVGA (480 × 800: mdpi)
	4.7" WXGA (1280 × 720: xhdpi)
	4.65" 720p (720 × 1280: xhdpi)
✓	4.0" WVGA (480 × 800: hdpi)
	3.7" FWVGA slider (480 × 854: hdpi)
	3.7" WVGA (480 × 800: hdpi)
	3.4" WQVGA (240 × 432: ldpi)
	3.3" WQVGA (240 × 400: ldpi)
	3.2" QVGA (ADP2) (320 × 480: mdpi)
	3.2" HVGA slider (ADP1) (320 × 480: mdpi)
	2.7" QVGA slider (240 × 320: ldpi)
	2.7" QVGA (240 × 320: ldpi)

sp: scaled pixels(放大像素). 主要用于字体显示 best for textsize, 此参数值也会随着