Android 学习手记

ZNing in Neuedu, Qingdao

2016-01-18

目 录

Day 001: Android 环境配置与初步接触

一、Android 环境配置

这里，我们学习一下在 Windows 环境下进行 Android 环境配置的相关内容。本教程所使用示例机器为 Windows XP Professional SP3，其他 Windows 版本配置大同小异。

Step 1: 卸载旧/已安装版本 JDK，并删除原有环境变量（非必需）

首先卸载已安装的 JDK，可通过“控制面板-添加或删除程序”进行卸载，或者通过第三方软件卸载。此处不再赘述。

卸载完成后，右击“我的电脑”，选择“属性”，打开“系统属性”，选择“高级”-“环境变量”。在打开的对话框中删除 JAVA_HOME 及 Path 中与刚卸载版本 JDK 相关的配置路径字符串。后连续单击确定即可。

删除前

删除后

注意：如果安装后无法运行 eclipse，可能需要重新添加 JAVA_HOME 变量及 Path、CLASSPATH 变量 Java 环境相关部分。

Step 2: 根据操作系统情况，安装需要使用的版本 JDK

Windows XP 是 32 位系统，因此这里我们选择 32 位 JDK 进行安装。这里我们选用的 JDK 版本是 Java 1.7。

JDK（Java 1.7）安装包运行

JDK（Java 1.7）安装中

JDK（Java 1.7）安装完毕

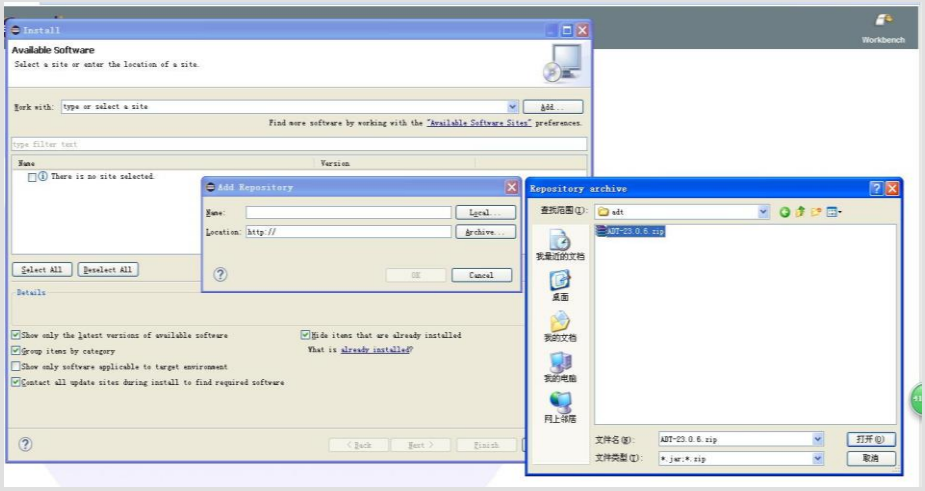
注意：安装时请务必不要出现中文路径，建议按默认路径安装。安装 JDK 期间需要安装 JRE（Java Runtime Environment，Java 运行环境），安装环境时请务必与 JDK 同目录下。

JRE（Java Runtime Environment，Java 运行环境）安装

Step 3: 配置 eclipse，为 eclipse 添加 ADT 插件：

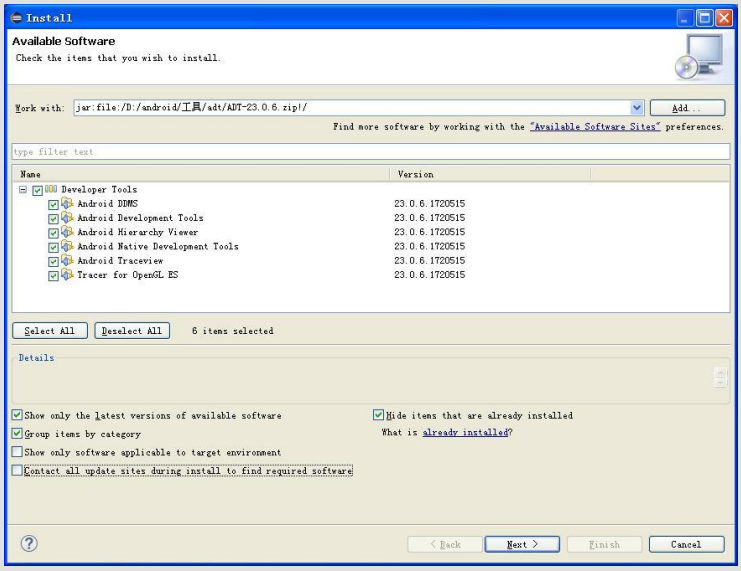
这里我们有多种方法可以进行插件的安装，例如在线安装和离线安装，这里我们演示离线安装方法：

在 eclipse 工具之中选择 Help-Install New Software，在弹出的对话框中，选择“work with”一栏最右边的“Add…”。在弹出的对话框中单击“Archive…”，选择你所下载的 ADT 插件压缩包，单击 OK。

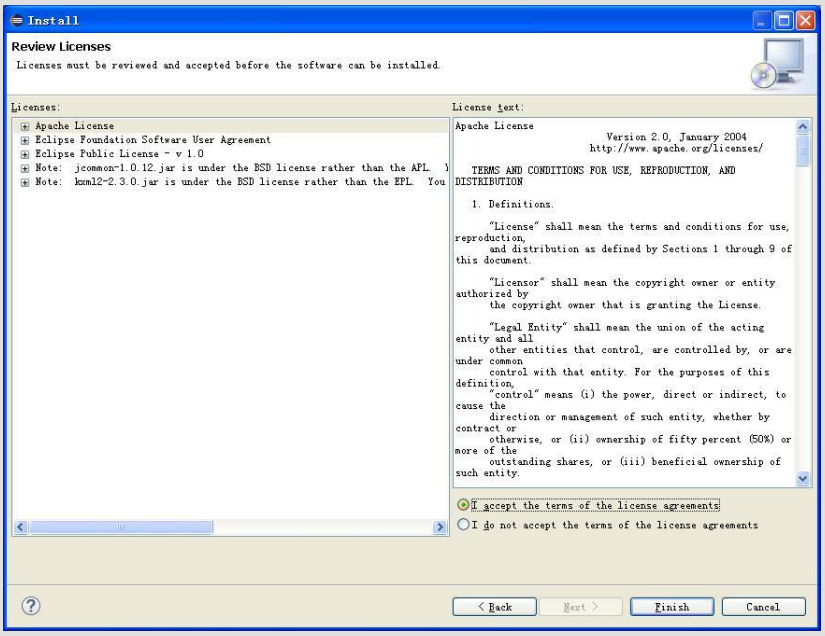


Install New Software 安装设置

返回最初的对话框后，取消选中“Contact all update sites during install to find required software”，单击 next，下一步选择需要安装的插件内容，默认全选，再下一步同意协议并单击 finish 完成后，完成插件添加。



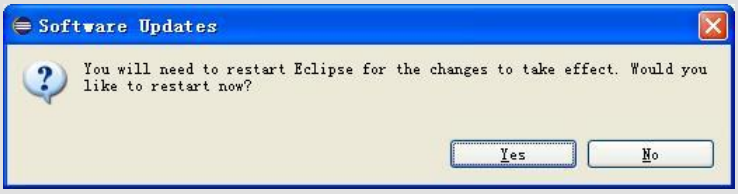
返回界面，需取消选中“Contact all update sites during install to find required software”



同意协议，单击 finish 完成

添加完成后需要重启 eclipse，按照提示操作即可。

正在加载插件中

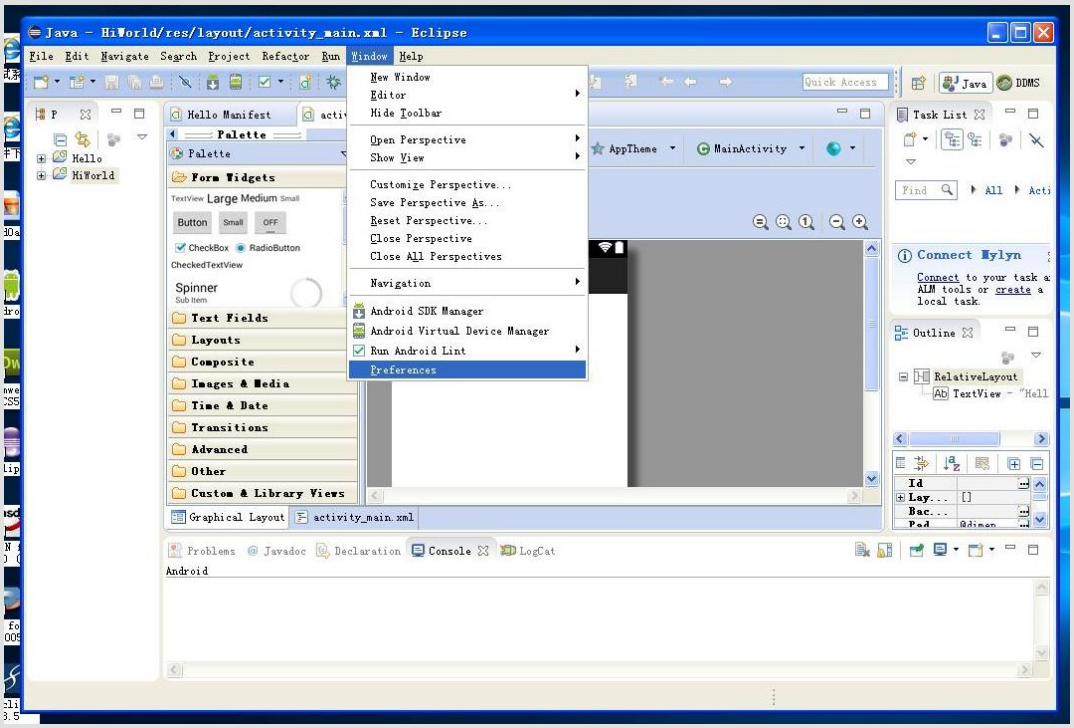


加载插件完成，软件提示重启 eclipse，单击 Yes 自动重启软件

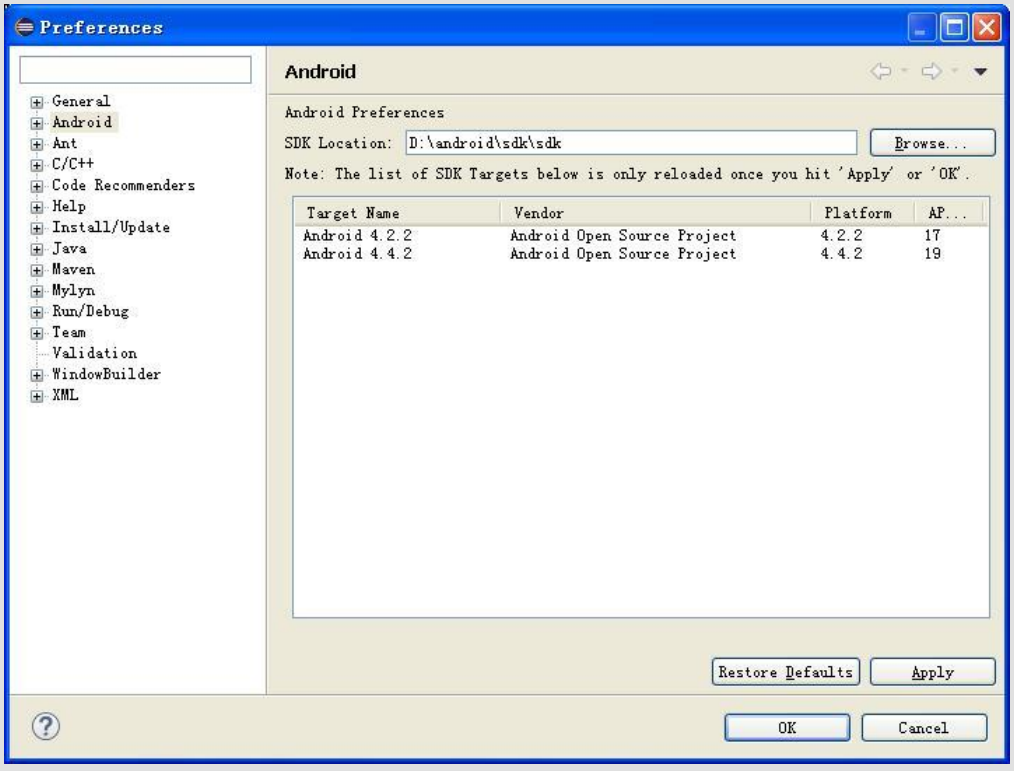
Step 4: 下载或离线安装 SDK 并更新，配置 SDK

解压 SDK 文件包放置在合适目录，等待使用即可。

随后在 eclipse 设置 Android SDK 环境，在菜单栏选择“Window-Perferences”，在打开的对话框中选择左侧“Android”之后再右侧“SDK Location”一栏选择“Browse…”，在打开的对话框选择 SDK 所在目录。后单击下面的“Apply”按钮后，单击“OK”即可设置完成。



设置 Android SDK 环境，在菜单栏选择“Window-Perferences”



选择 SDK 所在目录后单击下面的“Apply”、“OK”即可设置完成

Tips: SDK 文件夹相关解释与内容

add-one 目录下的是 Google 提供地图开发的库函数，支持基于 Google Map 的地图开发

add-ons：这里面保存着附加库，比如 Google Maps，当然你如果安装了 Ophone SDK，这里也会有一些类库在里面。

docs 目录下的是 Android SDK 的帮助文档，通过根目录下的 documentation.html 文件启动

platforms 目录中是不同版本的 Android SDK 的库函数、外观样式、程序示例和辅助工具等

tools 目录下的是通用的 Android 开发和调试工具

usb_driver 目录下保了用于 amd64 和 x86 平台的 USB 驱动程序

RELEASE_NOTES.html 是 Android SDK 的发布说明

market_licensing：作为 Android Market 版权保护组件，一般发布付费应用到电子市场可以用它来反盗版

Step 5: 使用 Intel 加速驱动开启计算机的 CPU 加速

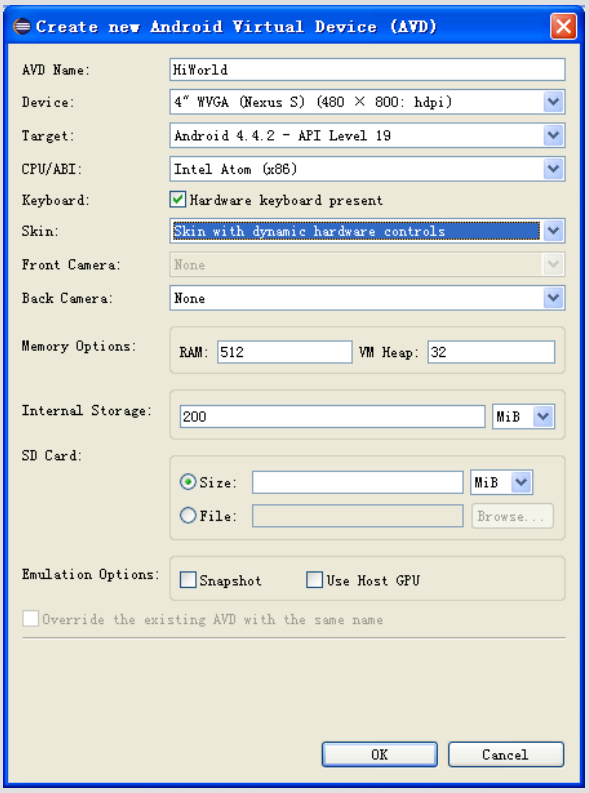
这里，为了更快的使用 Android SDK 模拟器模拟的安卓设备系统，我们在这里安装 Intel (R) Graphics Media Accelerator Driver 进行硬件加速。这里软件下载后默认安装即可，此处不再赘述。

注意：如果非 Intel 的 CPU，我们可以使用第三方的模拟器，例如 BlueStacks 等。（BlueStacks 安装需要.NET Framework 4.0 基本环境支持，请先预装环境。）

Step 6: 模拟器的创建

启动“Android Virtual Device (AVD) Manager”，单击“Create”按钮，根据本机硬件情况进行相关设置即可。这里我们机子内设置的虚拟设备参数如下图，如有需要可按照下述设置进行虚拟设备的创建。

注意：这里如果尚未安装上一步所提到的“Intel (R) Graphics Media Accelerator Driver” 或者其他进行硬件加速的相关软件的话，在“CPU/ABI”中不要选择“Intel Atom (x86)”。



创建虚拟设备的设置

至此，Android 环境配置已基本结束。

二、Android 初步接触

Step 1: 创建 Android 工程

可以创建新 Android 工程，也可利用已有代码创建 Android 工程。缺省为创建新 Android 工程；打开 eclipse，选择“File-New-Project”，在打开的对话框中选择“Android Application Project”。

可以使用默认位置存储，也可取消复选框，选择其他位置保存，缺省为使用默认位置 D:/Android/workplace/。使用默认位置存储,这里保存位置在：D:/Android/workplace/HelloAndroid

Tips: Android 工程文件夹内相关解释与内容

src：保存的是所有.java 文件

gen：保存的是编译器自动编译生成的文件

R.java(Resource，源)：保存的是我们页面中需要的图片，尺寸，颜色，控件的编号。我们开发的时候通过 R 来获取相应的资源。

R.dimen.XXX：用于获取 dimens.xml 文件中的尺寸

R.string.XXX：用于获取 strings.xml 文件的数据，保存页面固定显示的文本

R.id.XXX：对应的是页面中显示控件的 id

R.drawable.XXX：用户查找 res 下所有以 drawable 开始的目录中的图片

res：用于保存 app 中显示的所有资源数据

drawable-XXX：用于保存图片通常会在 res 下创建 drawable 目录，存放所有图片

layout：布局，所有页面显示的控件

menu：菜单

values：所有显示的数据

strings.xml：显示的文本

dimens.xml：显示的尺寸

colors.xml：色彩(是一个 16 进制的数据) e.g. #rrggbb #aarrggbb

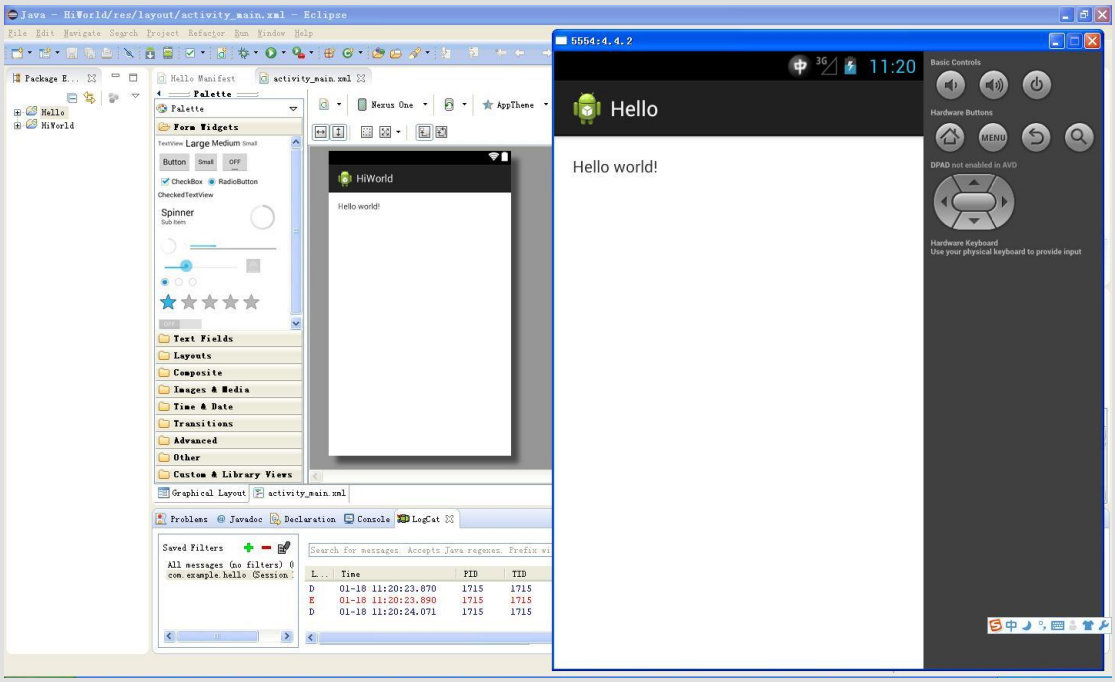
styles.xml：存放样式(用于美化页面)

AndroidManifest.xml：android 的清单文件（*）

注意：Android 中所有图片等资源命名必须符合 java 的命名规范必须以英文字母开始，可以包含数字字母下划线

Step 2: 运行并展示 Hello World 程序

这里，因为建立好的程序就是一个 Hello World 程序。建立好工程后，我们运行即可。首先在 AVD 中启动模拟器，后在 eclipse 进行编译，编译成功则在模拟器内显示已建立的 Android Hello World 程序（如图）：



已建立的 Android Hello World 工程

Day 002: Android 的 Activity、Intent、资源使用（一）

一、Java 面向对象的特性

面向对象主要有四大特性：封装、抽象、继承和多态。

封装：在面向对象语言中，封装特性是由类来体现的，我们将现实生活中的一类实体定义成类，其中包括属性和行为（在 Java 中就是方法），就好像人类，可以具有 name,sex,age 等属性，同时也具有 eat(),sleep()等行为，我们在行为中实现一定的功能，也可操作属性，这是面向对象的封装特性。

抽象：抽象就是将一类实体的共同特性抽象出来，封装在一个抽象类中，所以抽象在面向对象语言是由抽象类来体现的。比如鸟就是一个抽象实体，因为抽象实体并不是一个真正的对象，它的属性还不能完全描述一个对象，所以在语言中体现为抽象类不能实例化。

抽象是忽略一个主题中与当前目标无关的方面，把现实世界中的概念转换为对象,可以是表示数据的 VO 对象或抽象类，也可以是表示一类行为的接口。有利于从众多变化的状态中抽离出不变的东西。有利于使用继承来构造类的体系，实现多态，使用模板和工厂模式方法来实现对

业务过程的抽象。抽象是概念建模和业务流程建模很重要的工具，是面向对象体系的基石，而不是一堆杂乱、混乱、重复、散沙般的、关系错综复杂的类。

继承：继承就像是我们现实生活中的父子关系，儿子可以遗传父亲的一些特性，在面向对象语言中，就是一个类可以继承另一个类的一些特性，从而可以代码重用，其实继承体现的是 is-a 关系，父类同子类在本质上还是一类实体。

继承是一种连结类的层次模型，并且允许和鼓励类的重用，提供了一种明确表示共性的方法。 继承意味着能声明一个类型，而且能继承父类中的所有操作和数据，同事还有机会声明自己的新方法以改进、具体化、代替或者扩展父类中的方法。使用父类为应用领域建立层次模型，提供代码重用并允许定制。

如果需要进行防止继承，我们可以利用 final 声明类或方法，意味不能被修改。当一个类被声明为 final 时，只有方法自动被设为 final,而字段不会。final 有安全、高效等优点。而对于类型转换，Java 要求必须在继承体系内进行，并且转换之前应先使用 instanceof 进行检验。super 关键词不是一个对对象的引用，而是指向编译器调用超类方法或调用超类的构造器的超类的专用关键字。

多态：多态就是通过传递给父类对象引用不同的子类对象从而表现出不同的行为。一个对象变量可以指向多种实际类型的现象。

二、什么是 Activity

在认识 Activity 之前，我们需要认识 Android 系统中的四大重要组件。在 Android 系统中，组件是可以调用的基本功能模块，Android 应用程序就是由组件组成的。

Android 系统有四个重要的组件，分别是 Activity、Service、BroadcaseReceiver 和 ContentProvider。

Activity：

Activity 是 Android 程序的呈现层，显示可视化的用户界面，并接收与用户交互所产生的界面事件。Android 应用程序可以包含一个或多个 Activity，一般

在程序启动后会呈现一个 Activity，用于提示用户程序已经正常启动。在界面上的表现形式：全屏窗体，非全屏悬浮窗体，对话框。

Service：

Service 用于没有用户界面，但需要长时间在后台运行的应用

ContentProvider：

ContentProvider 是 Android 系统提供的一种标准的共享数据的机制，应用程序可以通过 ContentProvider 访问其他应用程序的私有数据。私有数据可以是存储在文件系统中的文件，也可以是 SQLite 中的数据库。Android 系统内部也提供一些内置的 ContentProvider，能够为应用程序提供重要的数据信息。

BroadcaseReceiver：

BroadcaseReceiver 是用来接受并响应广播消息的组件，不包含任何用户界面。可以通过启动 Activity 或者 Notification 通知用户接收到重要信息。Notification 能够通过多种方法提示用户，包括闪动背景灯、震动设备、发出声音或在状态栏上放置一个持久的图标。

所有 Android 组件都具有自己的生命周期，是从组件建立到组件销毁的整个过程，在生命周期中，组件会在可见、不可见、活动、非活动等状态中不断变化。

三、Activity 的生命周期

Activity 生命周期指 Activity 从启动到销毁的过程。其中，Activity 表现为四种状态，分别是活动状态、暂停状态、停止状态和非活动状态：

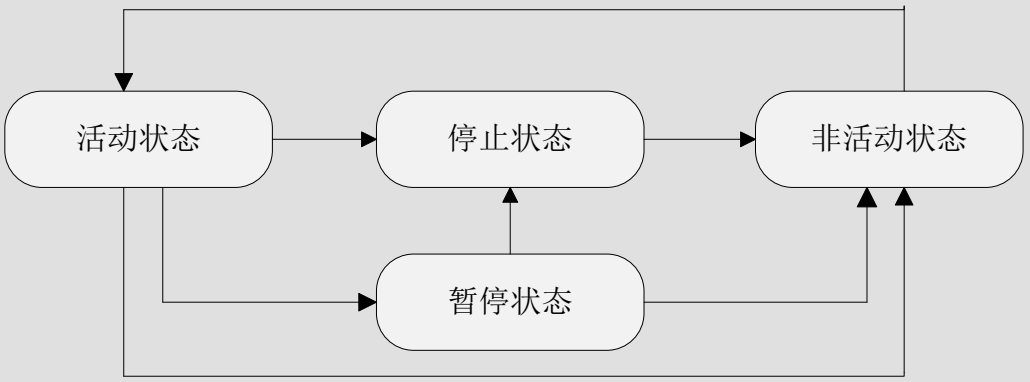
活动状态：Activity 在用户界面中处于最上层，完全能被用户看到，能够与用户进行交互。

暂停状态：Activity 在界面上被部分遮挡，该 Activity 不再处于用户界面的最上层，且不能够与用户进行交互。

停止状态：Activity 在界面上完全不能被用户看到，也就是说这个 Activity 被其他 Activity 全部遮挡。

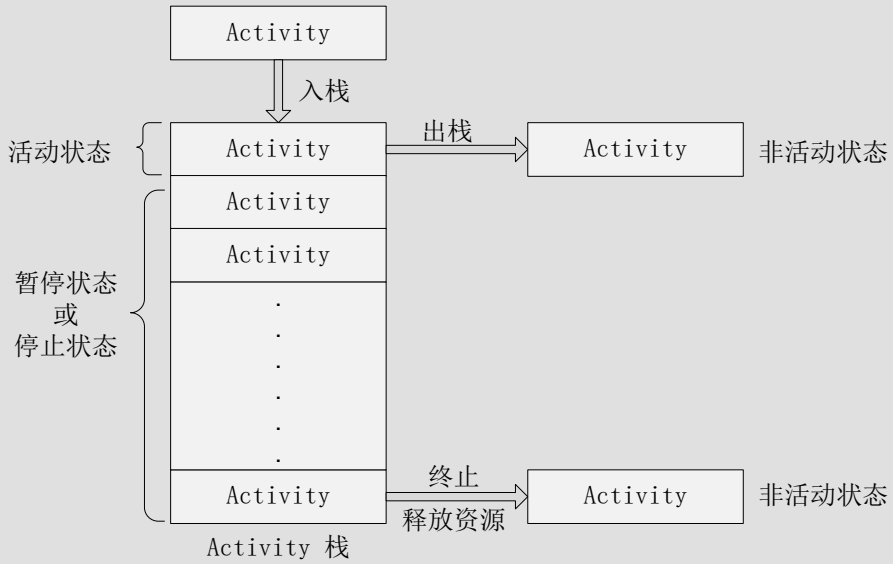
非活动状态：不在以上三种状态中的 Activity 则处于非活动状态。

需要说明的是，Activity 生命周期的几种状态之间可以互相转换。



Activity 的四种状态的变换关系图

此外，Activity 在 Android 中的调用形式是以 Activity 栈来进行活动情况判断的。即 Activity 的调用遵循“后进先出”的规则。



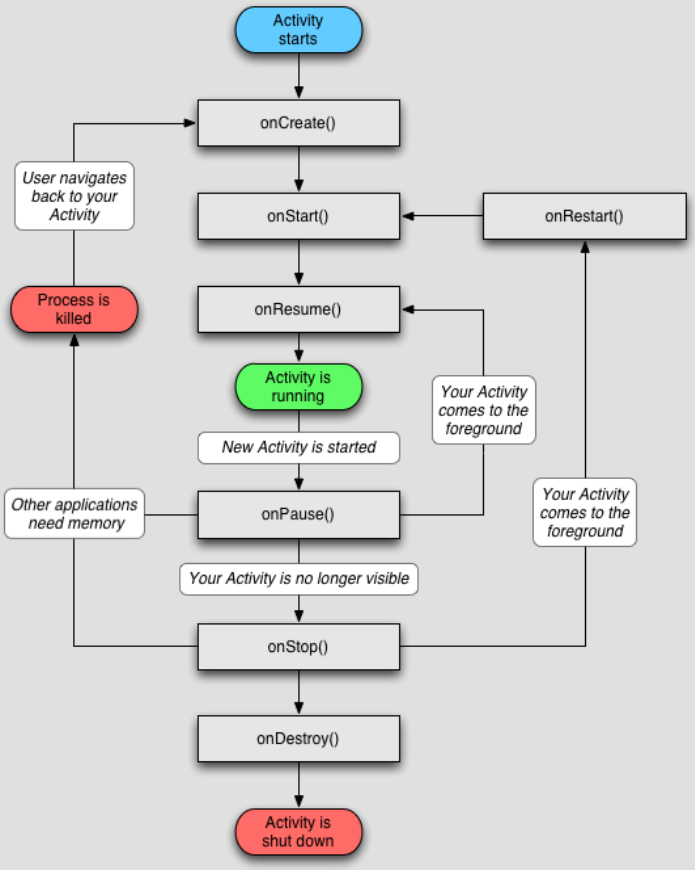
Activity 栈示意图

Activity 的构造方法是负责创建 Activity 的一个实例，一般省略，这里概不赘述。而 Activity 中对于用户操作产生的事件都有明确的方法，这里称为事件的回调函数。回调函数在 Activity 方法内主要是以下形式的存在：

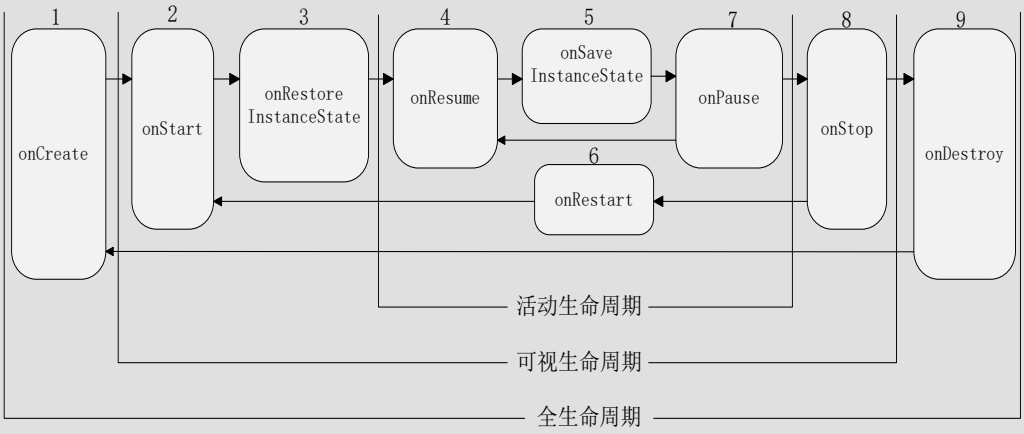
```
1    public class MyActivity extends Activity {
2        //负责把布局文件加载到 Java 中，必需重写（覆盖），生命周期中仅调用一
     次
3        protected void onCreate(Bundle savedInstanceState);
4        //在 Activity 呈献给用户的时候调用，生命周期中仅调用一次
5        protected void onStart();
6        //重新开始 Activity，生命周期中仅调用一次
```

```
7         protected void onRestart();
8         //Activity 整个呈现给用户的时候调用的方法，表示 Activity 已向用户
      展示
9         protected void onResume();
10        //暂停 Activity
11        protected void onPause();
12        //停止 Activity，生命周期中仅调用一次
13        protected void onStop();
14        //销毁 Activity，生命周期中仅调用一次
15        protected void onDestroy();
16    }
```

其中，MyActivity 即为你所定义的 Activity 名称。这里我们可以看到，一个 Activity 的基本的事件回调函数有 onCreate()、onStart()、onRestart()、onResume()、onPause()、onStop()、onDestroy()。这些事件的回调函数在一个 Android 应用中的流程图如下：



事件的回调函数在 Android 应用中流程图



事件的回调函数在 Android 应用 Activity 的生命周期划分

例如，现有 A、B 两个 Activity。由 A 调用 B，其中 B 以对话框的方式出现，则事件的回调函数调用流程是：onCreate->onStart->onResume->B 出现->调用 A 的 onPause->B 关闭->调用 A 的 onResume。

若 A 调用 B，其中 B 以完全覆盖 A 的方式出现，则事件的回调函数调用流程是：onCreate->onStart->onResume->B 出现->调用 A 的 onPause->调用 A 的 onStop-> B 关闭->调用 A 的 onRestart->调用 A 的 onStart->调用 A 的 onResume。

再举一个例子，若在 A 出现的时候，需要读取文本文件中的内容显示给用户，读取文本文件的代码应该在 onCreate、onStart、onResume。特别注意的是，onResume 推荐适用于经常修改的文本文件的读取代码的放置，因为这样，Activity 成为焦点的时候每一次都会执行，如果放置不经常修改的文件，会造成资源浪费。对于不经常修改的文件建议放在 onCreate 或前两者。

注意：在 Android 开发中，如果使用 System.out.println 之类的输出方法的话，不会在 Console 控制台输出，所有内容都在 LogCat 下输出。

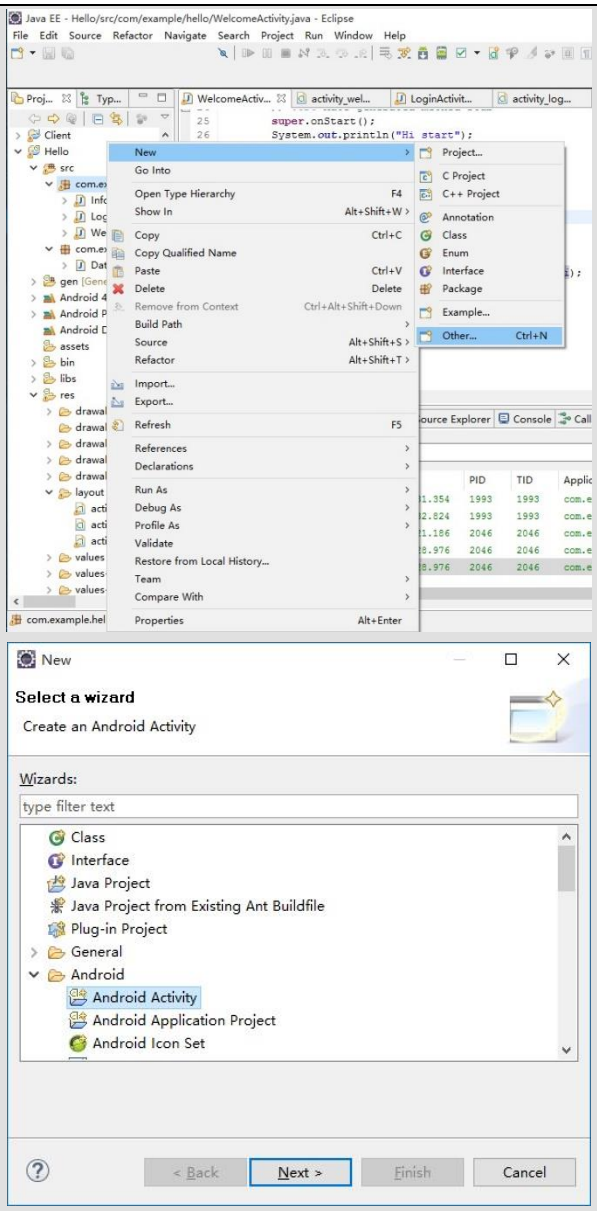
四、Intent 在不同 Activity 之间实现跳转

1、创建 Activity 并添加事件

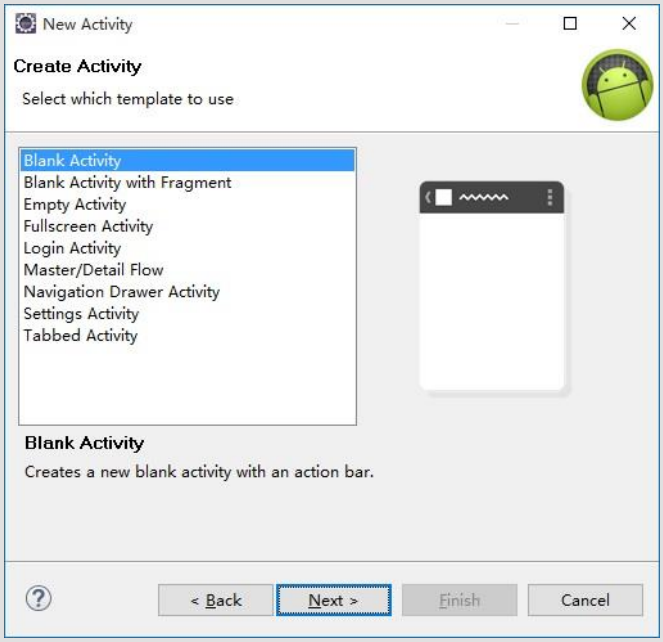
首先，我们需要学习一下如何创建 Activity。这里有两种方法可以创建 Activity。手动创建和 New 自动创建。

手动创建主要分为“创建布局文件”、“创建类继承 Activity”、“重写 onCreate()及 SetContextView(R.layout.XXX)方法”、“在 AndroidManifest 文件中配置 Activity”等四步。由于比较繁琐这里不再赘述。

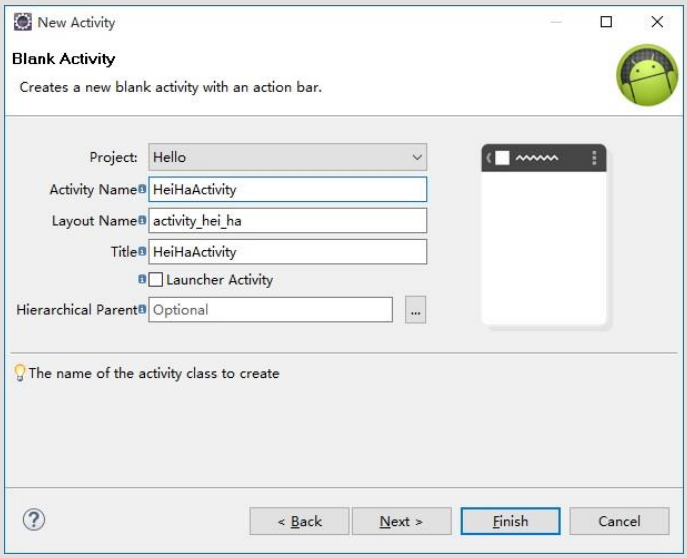
New 自动创建是通过 Android SDK 开发已给的模板进行创建即可。在需要创建 Activity 的包右击，选择“New-Other-Android-Android Activity”后单击“Next >”进行下一步操作，一般我们默认设置的是 Empty Activity 进行自我排版。单击“Next >”会进行 Activity 的名称设置。再单击“Next >”将展示需要修改的文件的具体情况，如没有问题可直接单击“Finish”结束即可。



New 自动创建选择“New-Other-Android-Android Activity”



设置 Empty Activity 进行自我排版



进行 Activity 的名称设置

在布局文件中，给 Button 控件添加点击事件添加如下代码，即可更改显示效果：android:theme="@android:style/Theme.Dialog"

Tips: 在改布局文件对应的 Java 中创建方法，格式固定：

doXXX：表示用于执行的方法

toXXX：表示用于跳转的方法

```
public void 方法名(View view){
}
```

以上方法是给控件添加事件的一种方式。

2、Intent 在不同 Activity 之间实现跳转（待补充）

Intent (意图) ，Intent 提供了一种通用的消息系统，它允许在你的应用程序与其它的应用程序间传递 Intent 来执行动作和产生事件，Intent 是对一次将要进行的操作的抽象描述。我们使用 Intent 启动一个 Activity、Service、Boradcast 等，并进行数据的传递。Intent 可以划分成显式意图和隐式意图：

显式意图：调用 Intent.setComponent()或 Intent.setClass()方法指定了组件名或类对象的 Intent 为显式意图，显式意图明确指定了 Intent 应该传递给哪个组件。即调用应用本身内部的相关操作。

隐式意图：Android 系统会根据隐式意图中设置的动作(action)、类别(category)、数据（URI 和数据类型）找到最合适的组件来处理这个意图。即调用系统的相关操作。

Intent Filter（过滤器）其实就是用来匹配隐式 Intent 的，如果 Intent Filter 定义的动作、类别、数据（URI 和数据类型）与 Intent 匹配，就会使用 Intent Filter 所在的组件来处理该 Intent。想要接收使用 startActivity()方法传递的隐式意图的活动必须在它们的意图过滤器中包含"android.intent.category.DEFAULT"。

由于显示意图即在应用内跳转 Activity，即

```
1  Intent intent = new Intent(本Activity.this,跳转目标的.class);
```

在 Activity 跳转的时候，可以通过 finish();方法关闭上一个 Activity。

3、打开新的 Activity，传递参数（待补充）

Android 中操作控件三部曲：

1、声明该控件类型的对象

2、通过 FindViewById 给对象赋值

3、通过对象中的方法，操作该控件

setText(“ddd”); 用于给控件设置显示的文本

getText().toString(); 获取控件显示的文本

Day 003: Android 的 Activity、Intent、资源使用（二）与 Android UI 组件

一、Intent 在不同 Activity 之间实现跳转（续）

4、返回数据

首先，我们通过 startActivityForResult(Intent intent, int requestCode)启动，这里 requestCode 是用于该返回的结果是哪个请求，区分发送请求的对象。紧接着，使用 startActivityForResult(Intent intent, int requestCode)方法打开新的 Activity，新 Activity 关闭前需要向前面的 Activity 返回数据需要使用系统提供的 setResult(int resultCode, Intent data)方法实现，通过 setResult(resultCode, intent);返回结果，其中 resultCode 表示返回码，用于判断是否正确返回。这个可以判断是通过应用按钮返回还是系统硬件返回。而 intent 中封装的是返回的数据。

注意：当前 Activity 必须 finish();

最后，显示返回的数据。通过重写 onActivityResult(int requestCode, int resultCode, Intent data)方法，处理用户返回的数据。

Tips: 在 Java 中编程使用常量代替固定的值可使用，例如：

public static final int CITY_REQUEST_CODE = 666;

且常量通常必须写在接口中。

Tips：把不同功能的代码，放在不同的 package 中。例如：util 一般都是工具包、view(activity)一般是视图包存放界面。

二、资源创建与使用

所有应用程序都有代码指令和数据构成。Andriod 工程中，资源文件和 Java 类文件时分开存储的。大多数常见的资源类型被存储于 XML 中。

Tips：Android 资源目录的层级结构：所有资源存放在 res 目录中

/res/drawable-*/ 图形资源文件

/res/layout/ 用户界面资源

/res/values/ 简单数据，字符串字符值等。

/res/values/strings.xml 字符串

/res/values/colors.xml 颜色

/res/values/dimens.xml 尺寸

/res/drawable/ 图像 .png .jpg 或 xml 定义的图形

/res/anim/xxxxx.xml 补间动画 <set> <apha><scale>…

/res/drawabe/xxx.xml 逐帧动画 <animation-list> <item>

/res/menu/helpmenu.xml 菜单

/res/xml/data.xml xml 文件

/res/raw 原始文件 xx.mp3 xx.mp4 xx.txt

/res/layout 布局

/res/values 样式和主题 <style>

在布局文件中需要通过@string/XXX、@color/XXX、@color/XXX、@dimen/XXX 调用，在 Java 类中需要通过 getResource().getString(R.string.XXX)、getResource().getColor(R.color.XXX)、getResource().getDimension (R.dimen.XXX)调用，通过@android:drawable/图片名可以访问使用 Android 系统自带的图片资源。

使用 string 字符串资源

```
1  <resources>
2  <string name="app_name">HelloAndroid</string>
3  </resources>
```

在应用程序访问字符串资源

```
1   String app_name=getResources().getString(R.string.app_name);
```

使用 string 字符串数组资源:

```
1  <resources>
2      <string-array name="flavors">
3          <item>java</item>
4          <item>android</item>
5          </string-array>
6  </resources>
```

在应用程序访问字符串资源

```
1   String flavors[] =
   getResources().getStringArray(R.array.flavors);
```

使用颜色

Android 可以存储 RGB 颜色值,RGB 颜色值以#开头，还可以给出 alpha 值以控制透明度.

#RGB (#F00 12 位颜色　红色)

#ARGB (#F00 12 位颜色　红色 alpha 为 50%红色)

#RRGGBB (#FF00FF 24 位颜色　羊红色)

#AARRGGBB (#80FF00FF 12 位颜色　红色 alpha 为 50%羊红色)

使用尺寸

Android 支持下列所有单位：

px（像素）：屏幕上的点。in（英寸）：长度单位。

mm（毫米）：长度单位。pt（点）：1/72 英寸。

dp（与密度无关的像素）：一种基于屏幕密度的抽象单位。在每英寸 160 点的显示器上，1dp=1px。

dip：与 dp 相同。

sp（与刻度无关的像素）：与 dp 类似，但是可以根据用户的字体大小 首选项进行绽放。

为了使用界面能够在现在和将来的显示器类型上正常显示，一般建议始终使用 sp 作为文字大小的单位，将 dip/dp 作为其它元素的单位 。另外也可以考虑使用矢量图，而不是位图。

Tips: px 和 dp 的区别

如果屏幕密度为 160，这时 dp、sp 和 px 是一样的。1dp=1sp=1px，但当使用 px 单位的时候，如果屏幕大小不变（假设还是 3.2 英寸），而屏幕密度变成了 320。那么假如原来控件的宽度设成 160px，这时候就会发现，该控件在 320 密度的屏幕下短了一半。但如果设置成 160dp 或 160sp 的话。系统会自动将 width 属性值设置成 320px。

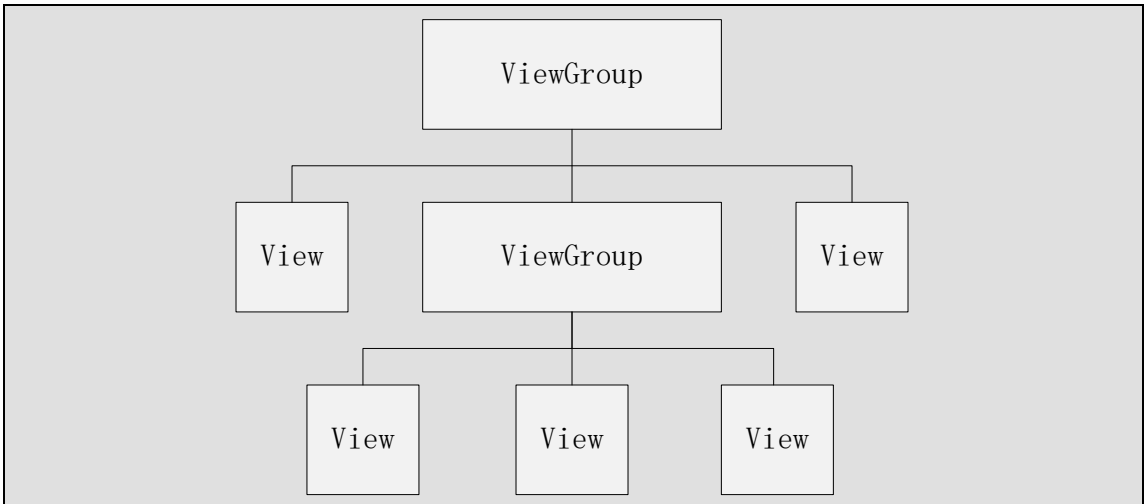
```
1  <dimen name="fourtpt">10pt</dimen>
2  float myDimen = getResources().getDimension(R.dimen.fourtpt);
```

注意：所有空间尺寸建议采用 dp 为单位，所有 XML 固定值都必须写在相应的文件中。

三、了解各种用户界面的控件的使用方法

UI 控件主要由系统控件与自定义控件。其中系统控件介绍如下：

Android 用户界面框架中的界面元素以一种树型结构组织在一起，称为视图树。Android 系统会依据视图树的结构从上至下绘制每一个界面元素。每个元素负责对自身的绘制，如果元素包含子元素，该元素会通知其下所有子元素进行绘制。



Android 用户界面框架（Android UI Framework）采用视图树（View Tree）模型

视图树由 View 和 ViewGroup 构成。View 是界面的最基本的可视单元，存储了屏幕上特定矩形区域内所显示内容的数据结构，并能够实现所占据区域的界面绘制、焦点变化、用户输入和界面事件处理等功能。View 也是一个重要的基类，所有在界面上的可见元素都是 View 的子类。ViewGroup 是一种能够承载含多个 View 的显示单元。ViewGroup 功能：一个是承载界面布局，另一个是承载具有原子特性的重构模块。

单线程用户界面。在单线程用户界面中，控制器从队列中获取事件和视图在屏幕上绘制用户界面，使用的都是同一个线程。其特点是处理函数具有顺序性，能够降低应用程序的复杂程度，同时也能减低开发的难度。但如果事件处理函数过于复杂，可能会导致用户界面失去响应。因此，在 Java 进行开发应用的时候，多线程是很好的帮手。

常见的系统控件：TextView、EditView、Button、Menu、RadioGroup、RadioButton、CheckBox、ProgressBar、ListView、TabWidget、SeekBar、ScrollView、GirdView、ImageSwitcher 等控件。

多知一点：关于 Java/Android 多线程的一些知识

刚才我们提到，Android UI 控件是单线程用户界面，那么单线程作为程序最基本的执行方式会出现由于事件处理函数过于复杂，从而导致用户界面失去响应的情况。因此，这里我们需要引入多线程来进行问题的解决。

关于线程的概念：线程是进程内部的一个执行单元。系统创建好进程后，实际上就启动执行了该进程的主线程，主线程以函数地址形式将程序的启动点提供给操作系统。主线程终止了，进程也就随之终止。多线程可以实现并行处理，避免了某项任务长时间占用 CPU 时间。

在 Java/Android 开发中，多线程的使用比例非常多。而线程的撰写和使用也跟 MFC 大同小异但更加方便。多线程使用主要有两个关键的问题需要解决：

1）线程的创建与调用

2）线程与线程之间的通讯

这里，前者 Java 已经有相关的封装类为我们做好了这一点，而对于后者，Android 通过消息通信机制让线程与线程之间的通信成为可能。以下是创建线程的代码示例（一个倒计时的小应用），相关包引用未列出：

```
1       //创建一个线程的对象，并指定任务
2       //匿名类
3       Thread t=new Thread(new Runnable(){
4           @Override
5           public void run() {
6               for(int i=10;i>0;i--){
7                   //不允许在子线程中访问主线程的内容
8                   //我们需要通过消息机制实现线程之间的通信
9                   //1、创建的消息的句柄 handler，实现消息的发送与处理
10                  //2、在子线程中通过 handler（java 内的类）。sendMessage
   向主线程中发消息。
11                  //textMsg.setText(String.valueOf(i));
12                  System.out.println("==="+i);
13                  /*************************************/
14                  //创建消息并发送
15                  Message msg = new Message();
16                  msg.what = 1;//设置消息的类型，将来用于区分各种消息
17                  msg.obj=i;//设置传递的数据
18                  handler.sendMessage(msg);
19                  /*************************************/
20                  try {
21                      Thread.sleep(1000);
```

```
22                } catch (InterruptedException e) {
23                    // TODO Auto-generated catch block
24                    e.printStackTrace();
25                }
26            }
27            Message msg = new Message();
28            msg.what = 2;//设置消息的类型，将来用于区分各种消息
29            handler.sendMessage(msg);
30        }
31    });
32    //开启线程
33    t.start();
```

上述代码中，Thread.sleep(1000);部分（语句功能：控制 CPU 执行时间为 1 秒一次）由于是线程控制，会抛出异常，因此在 Eclipse 提示下转变为如下代码：（try-catch 语句）

```
1  try {
2     Thread.sleep(1000);
3  } catch (InterruptedException e) {
4     // TODO Auto-generated catch block
5     e.printStackTrace();
6  }
```

这里，我们可以总结出 Java 中创建并调用线程的代码框架（即一般方法）：

```
1      Thread t=new Thread(new Runnable(){
2          @Override
3          public void run() {
4              // TODO Auto-generated method stub
5              //在此添加执行的代码即可
6          }
7      });
8      //开启线程
9      t.start();
```

线程创建好后，正常情况下，线程之间是无法进行操作的，也不允许在子线程中访问主线程的内容。这里我们就需要创建相关方法进行线程消息通信。这里，Android 系统中的 android.os.Handler 包中给我们带来了一个方便实用的方法，我们称其为句柄（或语柄）。相关调用示例如下：

线程内的消息创建与发送：（详见本附录第一页第一块代码）

```
1                //创建消息并发送
2                Message msg = new Message();
```

```
3                 msg.what = 1;//设置消息的类型，将来用于区分各种消息
4                 msg.obj=i;//设置传递的数据
5                 handler.sendMessage(msg);
```

这里，通过创建 Message 对象，并设置 msg.what 消息类型和 msg.obj 传递的参数，来对 Message 对象进行创建（即创建消息）。之后，在 Activity 继承类之中，我们需要创建 handler 类型的对象，并覆盖处理消息的方法，用于重写 handleMessage 方法，处理发送过来的消息。具体示例代码如下：（相关包引用未列出，主要引用 android.os.Handler 包内容）

```
1  //创建 handler 类型的对象，并覆盖处理消息的方法
2  private Handler handler = new Handler(){
3      //重写 handleMessage 方法用于处理发送过来的消息
4      public void handleMessage(android.os.Message msg){
5          switch (msg.what) {
6          case 1:
7              textMsg.setText(msg.obj.toString());
8              break;
9          case 2:
10             textMsg.setText("Time running out");
11             break;
12         default:
13             break;
14         }
15     };
16 };
```

这里，我们可以总结出 Android 中创建 Handler 对象并进行线程间的消息通信的代码框架（即一般方法）：

```
1  private Handler handler = new Handler(){
2      public void handleMessage(android.os.Message msg){
10             // TODO Auto-generated method stub
11             //在此添加执行的代码即可
3          }
4      };
5  };
```

多线程在 Android 系统中用途广泛，小型应用一般在倒计时、时钟显示、广告轮换或灯箱等显示方面运用频繁。是 Android UI 控件设计之中的非常令人心悦的方法。不过由于其涉及代码跨度大，操作较复杂，需要勤加练习才可熟练掌握。