

android
developers

Android开发教程&笔记



android

Android 虚拟机 Dalvik

Dalvik 冲击

随着 Google 的 AndroidSDK 的发布，关于它的 API 以及在移动电话领域所带来的预期影响这些方面的讨论不胜枚举。不过，其中的一个话题在 Java 社区是一石激起千层浪，这就是 Android 平台的基础——Dalvik 虚拟机。



Dalvik 和标准 Java 虚拟机(JVM)首要差别

Dalvik 基于寄存器，而 JVM 基于栈。基于寄存器的虚拟机对于更大的程序来说，在它们编译的时候，花费的时间更短。

Dalvik 和 Java 运行环境的区别

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 形势

Dalvik 的诞生也导致人们开始忧虑 Java 平台的第一次大规模的分道扬镳或许已经是进行时了——有人已经把 Dalvik 和微软的 JVM 以及 Sun 对微软的诉讼联系起来，等着看 Google 身上是否也会发生类似事情；另外一些人则指出，Google 并没有宣称 Dalvik 是一个 Java 实现，而微软却是这样做的。Sun 也对可能带来的阵营分裂表达了忧虑情绪，并提出和 Google 合作来保证 Dalvik 和 JVM 之间的兼容性——Google 对此的解释是，Dalvik 是对解决目前 JavaME 平台上分裂的一次尝试，也是为了提供一个拥有较少限制许可证的平台。甚至还有人怀疑这是否是 Sun 和 Google 两大阵营对 Java 之未来的一次大规模较量。

Android 中各种 JAVA 包的功能描述

在 Android 的应用程序开发中，通常使用的是 JAVA 语言，除了需要熟悉 JAVA 语言的基础知识之外，还需要了解 Android 提供的扩展的 JAVA 功能。

在一般的 JAVA 应用中，如果需用引用基础类库，通常需要使用如下的方式：

```
import javax.swing.*;
```

以上代码表示了引用 JAVA 的 GUI 组件 `Swing`, `javax.swing` 即 JAVA 中的一个包。

android 提供一些扩展的 JAVA 类库，类库分为若干个包，每个包中包含若干个类。



重要包的描述:

`android.app` : 提供高层的程序模型、提供基本的运行环境

`android.content` : 包含各种的对设备上的数据进行访问和发布的类

`android.database` : 通过内容提供者浏览和操作数据库

`android.graphics` : 底层的图形库，包含画布，颜色过滤，点，矩形，可以将他们直接绘制到屏幕上。

`android.location` : 定位和相关服务的类

`android.media` : 提供一些类管理多种音频、视频的媒体接口

`android.net` : 提供帮助网络访问的类，超过通常的 `java.net.*` 接口

`android.os` : 提供了系统服务、消息传输、IPC 机制

`android.opengl` : 提供 OpenGL 的工具

`android.provider` : 提供类访问 Android 的内容提供者

`android.telephony` : 提供与拨打电话相关的 API 交互

`android.view` : 提供基础的用户界面接口框架

`android.util` : 涉及工具性的方法，例如时间日期的操作

`android.webkit` : 默认浏览器操作接口

`android.widget` : 包含各种 UI 元素（大部分是可见的）在应用程序的屏幕中使用

Android 的相关文件类型

Java 文件——应用程序源文件

android 本身相当一部分都是用 java 编写而成(基本上架构图里头蓝色的部份都是用 Java 开发的), android 的应用必须使用 java 来开发。



Class 文件——Java 编译后的目标文件

不像 J2se, java 编译成 class 就可以直接运行, android 平台上 class 文件不能直接在 android 上运行。由于 Google 使用了自己的 Dalvik 来运行应用, 所以这里的 class 也肯定不能在 AndroidDalvik 的 java 环境中运行, android 的 class 文件实际上只是编译过程中的中间目标文件, 需要链接成 dex 文件后才能在 dalvik 上运行。

Dex 文件——Android 平台上的可执行文件

Android 虚拟机 Dalvik 支持的字节码文件格式 Google 在新发布的 Android 平台上使用了自己的 Dalvik 虚拟机来定义, 这种虚拟机执行的并非 Java 字节码, 而是另一种字节码: dex 格式的字节码。在编译 Java 代码之后, 通过 Android 平台上的工具可以将 Java 字节码转换成 Dex 字节码。虽然 Google 称 Dalvik 是为了移动设备定做的, 但是业界很多人认为这是为了规避向 sun 申请 Javalicense。这个 DalvikVM 针对手机程式/CPU 做过最佳化, 可以同时执行许多 VM 而不会占用太多 Resource。

Apk 文件——Android 上的安装文件

Apk 是 Android 安装包的扩展名, 一个 Android 安装包包含了与某个 Android 应用程序相关的所有文件。apk 文件将 AndroidManifest.xml 文件、应用程序代码(.dex 文件)、资源文件和其他文件打成一个压缩包。一个工程只能打进一个.apk 文件。

Android 的应用程序结构分析：HelloActivity

本例以一个简单的 HelloActivity 程序为例，简单介绍 Android 应用程序的源代码结构。事实上，Android 应用程序虽然不是很复杂，但是通常涉及了 JAVA 程序,XML 文件，Makefile 多方面的内容。HelloActivity 虽然简单，但是麻雀虽小，五脏俱全，是学习 Android 应用程序的最好示例。

第一部分：HelloActivity 的源代码

HelloActivity 工程的源代码在 Android 目录的 development/samples/HelloActivity/中，代码的结构如下所示：

```
development/samples/HelloActivity/  
|-- Android.mk  
|-- AndroidManifest.xml  
|-- res  
|   |-- layout  
|   |   `-- hello_activity.xml  
|   `-- values  
|       `-- strings.xml  
|-- src  
|   `-- com  
|       `-- example  
|           `-- android  
|               `-- helloactivity  
|                   `-- HelloActivity.java  
`-- tests  
    |-- Android.mk  
    |-- AndroidManifest.xml  
    `-- src  
        `-- com  
            `-- android  
                `-- helloactivity  
                    `-- HelloActivityTest.java
```

其中 tests 是一个独立的项目，可以暂时不考虑。其他部分看作一个 Android 的一应用程序的工程。这个工程主要的组成部分如下所示：

AndroidManifest.xml：工程的描述文件，在运行时有用处

Android.mk：整个工程的 Makefile


res: 放置资源文件的目录

src/com/example/android/helloactivity/HelloActivity.java: 这是 JAVA 类文件, 这个文件的路径表示在 Andorid 的 JAVA 包的结构中的位置, 这个包的使用方式为 com.example.android.helloactivity。

第二部分: 编译的中间结果

这个 HelloActivity 工程经过编译后将生成

out/target/common/obj/APPS/HelloActivity_intermediates/ 目录, 这个目录中的内容都是 HelloActivity 工程相关的, 更具体地说都与 development/samples/HelloActivity/ 中的 Android.mk 文件相关。



```
out/target/common/obj/APPS/HelloActivity_intermediates/
|-- classes.dex (字节码)
|-- classes.jar (JAR 文件)
|-- public_resources.xml (根据 resources 结构生成的 xml)
    |-- src
        |-- R.stamp
    |-- com
        |-- example
            |-- android
                |-- helloactivity
                    |-- R.java (resources 生成的文件)
```

classes.dex

是一个最重要的文件, 它是给 Android 的 JAVA 虚拟机 Dalvik 运行的字节码文件。

classes.jar

是一个 JAR 文件, JAR 的含义为 Java ARchive, 也就是 Java 归档, 是一种与平台无关的文件格式, 可将多个文件合成一个文件。解压缩之后的目录结构: (JAVA 标准编译得到的类)

```
classes
|-- META-INF
|   |-- MANIFEST.MF
|-- com
|   |-- example
|       |-- android
|           |-- helloactivity
|               |-- HelloActivity.class
|               |-- R$.attr.class
|               |-- R$.id.class
|               |-- R$.layout.class
|               |-- R$.string.class
|               |-- R.class
```

各个以 class 为扩展名的文件，事实上是 JAVA 程序经过编译后的各个类的字节码。

第三部分： 目标 **apk** 文件

目标 apk 文件是 Android 的 JAVA 虚拟机 Dalvik 安装和运行的文件，事实上这个 apk 文件将由编译的中间结果和原始文件生成。apk 文件的本质是一个 zip 包。这个 APK 包解压缩后的目录结构如下所示：

```
out/target/product/generic/obj/APPS/HelloActivity_intermediates/package.apk_FILES/
|-- AndroidManifest.xml
|-- META-INF
|   |-- CERT.RSA
|   |-- CERT.SF
|   |-- MANIFEST.MF
|-- classes.dex
|-- res
|   |-- layout
|       |-- hello_activity.xml
|-- resources.arsc
```

值得注意的是，这里的 xml 文件经过了处理，和原始的文件不太一样，不能按照文本文件的方式阅读。

第四部分： 源代码的各个文件

Android.mk 是整个工程的 “Makefile”，其内容如下所示：

- ◆ LOCAL_PATH:= \$(call my-dir)
- ◆ include \$(CLEAR_VARS)
- ◆ LOCAL_MODULE_TAGS := samples
- ◆ # Only compile source java files in this apk.
- ◆ LOCAL_SRC_FILES := \$(call all-java-files-under, src)
- ◆ LOCAL_PACKAGE_NAME := HelloActivity
- ◆ LOCAL_SDK_VERSION := current
- ◆ include \$(BUILD_PACKAGE)
- ◆ # Use the following include to make our test apk.
- ◆ include \$(call all-makefiles-under,\$(LOCAL_PATH))



这个文件在各个 Android 的工程中都是类似的，其中 LOCAL_PACKAGE_NAME 表示了这个包的名字。LOCAL_MODULE_TAGS 表示了模块的标，在这里使用的是 samples，正式的应用程序（packages 目录中的应用）中多使用 eng development。

AndroidManifest.xml 是这个 HelloActivity 工程的描述文件，其内容如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.helloactivity">
    <application android:label="Hello, Activity!">
        <activity android:name="HelloActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

其中 package 用于说明这个包的名称，android:label 中的内容是表示这个应用程序在界面上显示的标题，activity 中的 android:name 表示这个 Android 的活动的名称。

Android 编程基础

文件 `src/com/example/android/helloactivity/HelloActivity.java` 是程序主要文件，由 JAVA 语言写成

```
package com.example.android.helloactivity;
import android.app.Activity;
import android.os.Bundle;
public class HelloActivity extends Activity {
    public HelloActivity() {
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.hello_activity);
    }
}
```

`com.example.android.helloactivity` 表示的是这个包的名称，在文件的头部引入了两个包 `android.app.Activity` 是一个 Android 活动（Activity）包，每一个 Android 活动都需要继承 Activity 类。

包 `android.os.Bundle` 用于映射字符串的值。

`onCreate()` 是一个重载的函数，在这个函数中实现应用程序创建的所执行的过程。其中 `setContentView()` 设置当前的视图（View）。

设置的方法是使用一个文件，这个文件因此决定了视图中包含的内容。这里使用的是 `R.layout.hello_activity`，表示从 `res/layout/` 目录中使用 `hello_activity.xml` 文件。

`res/layout/hello_activity.xml` 文件的内容如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/text"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textSize="18sp"
    android:autoText="true"
    android:capitalize="sentences"
    android:text="@string/hello_activity_text_text" />
```

其中定义了一个可编辑的文本（EditText），下面的各项其实是它的各种属性，`android:text` 表示这个文本的内容，`string/hello_activity_text_text` 表示找到相应的文件，也就是 `res/value/string.xml` 文件中的 `hello_activity_text_text` 文本。

`res/value/string.xml` 的内容如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello_activity_text_text">He llo, World!</string>
</resources>
```

`hello_activity_text_text` 文本被 `res/layout/hello_activity.xml` 文件引用，正是应用程序运行时在屏幕显示的文本。

Android ADB 工具使用

adb(Android Debug Bridge)是 Android 提供的一个通用调试工具，借助这个工具，我们可以管理设备或手机模拟器的状态。

adb 功能操作：

- ◆ 快速更新设备或手机模拟器中的代码，如应用或 Android 系统升级
- ◆ 在设备上运行 shell 命令
- ◆ 管理设备或手机模拟器上预定端口
- ◆ 在设备或手机模拟器上复制、粘贴文件



adb 常用操作：

安装应用到模拟器

```
adb install app.apk
```

Android 没有提供一个卸载应用的命令，只能手动删除：

```
adb shell  
cd data/app  
rm app.apk
```

进入设备或模拟器的 Shell

```
adb shell
```

通过以上命令，可以进入设备或模拟器的 shell 环境中，在这个 Linux Shell 中，你可以执行各种 Linux 的命令，另外如果只想执行一条 shell 命令，可以采用以下方式：

```
adb shell [command]
```

如：

```
adb shell dmesg
```

会打印出内核的调试信息

发布端口

可以设置任意的端口号，做为主机向模拟器或设备的请求端口。如：

```
adb forward tcp:5555 tcp:8000
```


复制文件

可向一个设备或从一个设备中复制文件

- ◆ 复制一个文件或目录到设备或模拟器上：

```
adb push
```

如：

```
adb push test.txt /tmp/test.txt
```

- ◆ 从设备或模拟器上复制一个文件或目录

```
adb pull
```

如：

```
adb pull /android/lib/libwebcore.os
```



搜索/等待模拟器、设备实例

取得当前运行的模拟器、设备的实例列表及每个实例的状态 | 等待正在运行的设备

```
adb devices
```

```
adb wait-for-device
```

查看 Bug 报告

```
adb bugreport
```

记录无线通讯日志

无线通讯记录日志非常多，在运行时没必要记录，可以通过命令设置记录

```
adb shell
```

```
logcat -b radio
```

获取设备 ID 和序列号

```
adb get-product
```

```
adb get-serialno
```

访问数据库 SQLite3

```
adb shell
```

```
sqlite3
```

android
developers



android