Android 系统下基于 NDK 方式的图形开发①

王有禄, 李代平

(广东工业大学 计算机学院, 广州 510006)

摘 要: Google 提供了 Android 系统下基于 C/C++开发的 NDK 开发工具, 通过使用这个工具可以嵌入 C/C++代码到 Android 应用程序中, 即可以使用“Java+C/C++”的编程方式进行程序开发. 基于这种开发模式, 开发 Android 应用程序中的图形模块, 通过在 Native C/C++中调用两个图形引擎 Skia GL 和 OpenGL ES 的链接库, 实现在应用程序中不使用 Android SDK 的图形模块 API 函数, 同样能完成 2D 和 3D 的图形开发.

关键词: Android; NDK; 图形开发; Skia GL; OpenGL ES

Graphics Development of Android System Based on NDK Mode

WANG You-Lu, Li Dai-Ping

(Guangdong University of Technology, Faculty of Computer, GuangZhou 510006, China)

Abstract: Google provided NDK development tools to produce C/C++ development base on Android system, through use of this tool, C/C++ code can be embedded into the Android application, then use “Java+C/C++” programming way to develop application. Based on this model to produce Android graphics development, through use C/C++ to call link library of two graphics engine Skia GL and OpenGL ES, realize 2D/3D graphics development in application without to call graphics API of Android SDK.

Key words: Android; NDK; graphics development; Skia GL; OpenGL ES

1 引言

众所周知, Android 的 SDK 是基于 Java 实现的, 这就意味着基于 Android?SDK 进行开发的第三方应用程序都必须使用 Java 语言. 但是并不等于在程序中只能使用 Java 语言, 因为程序中同样可以调用非 Android SDK 所实现的功能. 在最早 Android SDK 发布的时候, Google 就宣称 Android 虚拟机 Dalvik 支持 JNI 编程方式, 也就是第三方在开发自己的应用时完全可以通过 JNI 调用自己的 C/C++动态库, 即在 Android 平台上, “Java+C/C++”的编程方式是一直都可以实现的. 但是对于如何使用 JNI 完成了自己的 C/C++ 动态链接库开发, 动态链接库如何和应用程序一起打包成 apk 并发布, Google 对这种技术并没有做相关的说明, 也没有提供相关的开发组件进行支持[1]. 因此普通开发者使用这种开发模式并不知道如何去创建自己的 C/C++动态链接库, 通过什么方式将动态链接库捆绑进应用程序中去. 但是, 这一状况随着 Google 发布 NDK 开发工具而随之改变.

2 Android NDK

Google 的 Android NDK, 全称是 Android Native Developer Kit, 是一系列的 Android 原生态开发工具, NDK 集成了交叉编译器, 允许开发人员在 Android 应用程序中嵌入 C/C++编写的代码, 帮助开发者快速开发 C/C++的动态链接库(.so 文件), 并提供了相应的 mk 文件隔离 CPU 、平台、 ABI 等差异, 开发人员只需要简单修改 mk 文件, 就可以创建出.so 文件, 并能自动将.so 文件 和 Java 应用一起打包成 apk[2]. 这些工具的使用对开发者使用 C/C++做 Android 开发的帮助是巨大的. NDK 可以将本地 C/C++组件代码, 嵌

① 收稿时间:2012-04-18;收到修改稿时间:2012-05-28

入到应用程序中使用, 能够 Android 的应用程序直接使用原生态开发语言 C/C++来开发, 增加了软件开发的灵活性.

NDK 的发布, 意味着官方正式的支持在 Android 应用程序开发中嵌入 C/C++编写的代码, 使“Java+C/C++”的开发方式成为一种好的选择. 使用 NDK 做 Android 的应用程序开发有以下几个优势: ①可以将要求高性能的应用逻辑使用 C/C++ 开发, 从而提高应用程序的执行效率; ②代码的保护, 由于 apk 的 Java 层代码很容易被反编译, 而 C/C++库反汇难度较大, 我们可以将需要保密的应用逻辑使用 C C/C++开发; ③在 NDK 中调用第三方 C/C++库, 因为大部分的开源库都是用 C/C++代码编写的, 可以重用现有的大量 C/C++代码, 可以实现基于 Android SDK 没有实现的功能; ④对于开发跨手机系统的应用软件, 可以用 C/C++开发跨系统软件基于 Android 系统部分的 API 函数; ⑤便于移植, 用 C/C++写的库可以方便在其他的嵌入式平台上再次使用, 代码的重用性较高.

3 Android的图形系统

Android 的图形系统在 Android SDK 中主要有两个模块, 一个基于 Skia Graphics Library 实现的包 Android.view、Android.graphics、Android.widget, 另一个是 OpenGL ES 所在的包 Android.OpenGL, 其中前者主要用于应用程序中的 2D 画图, 也会调用部分 OpenGL 的内容来实现简单的 3D 效果, 后者主要用于程序中得 3D 画图[3]. 其图形系统如图 1 所示.

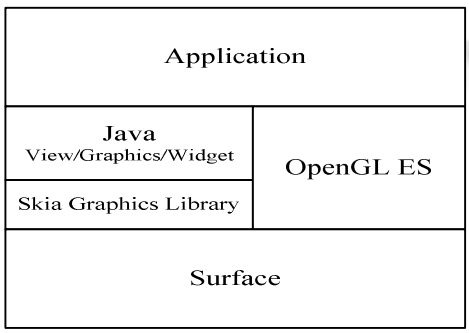

图 1 Android 的图形系统

Skia 是一个向量显示引擎, 能在低端装置如手机上呈现高品质的 2D 图形. Google 于 2005 年 11 月收购 Skia 公司后, 在 2007 年将经过改良的 Skia 2D 向量图形处理函数库 Skia Graphics Library 作为 Android 平台的图形引擎[4]. 而且 Skia 作为 Android 核心图形引擎, Android 中有很多绘图类(Android.graphics 包下的类)都是对 Skia C++类通过 JNI 的方式进行了调用及封装. 其画图方式为在程序中创建一个 View, 在其 onDraw 方法中能获得一个 Canvas 对象, 或通过创建一个 SurfaceView, 根据其 getHolder 函数获得 SurfaceHolder, 再根据 SurfaceHolder 的 lockCanvas 获得一个 Canvas 对象, 然后就可以使用所得到的 Canvas 调用 SDK 中的 API 函数进行画图了.

OpenGL ES (OpenGL for Embedded Systems) 是 OpenGL 三维图形 API 的子集, 针对手机、PDA 和游戏主机等嵌入式设备而设计. Android 系统中提供的关于 OpenGL 的相关 API 在 Android.OpenGL 可以找到. Android 系统中从 Android SDK 1.0 开始就开始包含了 OpenGL ES 1.0 和 1.1 API, 而从 Android SDK 2.2 开始, 才支持 OpenGL ES 2.0 API 规范, 目前大部分的 Android 设备都支持 OpenGL ES 2.0 且把把它作为开发 OpenGL 的重要选择[5]. 基于 OpenGL ES 运行机制的不同, Android 为其提供了一个专用在 3D 上画图的 GLSurfaceView, 其中实现了其他 View 所不具备的操作, 然后在 GLSurfaceView 里面获得 OpenGL 的句柄进行画图.

Android 程序中创建一个 view 类或其子类, 或者在 Activity 中调用 setContentView 后, 都会从系统中获取一块 Surface 内存, 作图都是针对提供给应用程序的 Surface 内存填充数据. 因此基于 Android SDK 的画图, 就是调用系统中的 2D 或 3D API 画图函数, 将画下来的图保存在这个内存中, 然后这个内存里面的内容会被系统渲染后变为屏幕上的像素信息.

4 基于NDK方式的图形开发

上面所讲 Android 的图形系统都是通过 Java 调用 SDK 中的 API 进行作图, 接下来分析如何通过 NDK 方式使用原生的 C/C++进行图形开发. 通过这种方式进行图形开发, 可以使用 C/C++调用 Skia GL 和 OpenGL ES 的 C++链接库, 将所有需要实现的图形模块都在 C/C++端实现并封装好, 提供接口给 Java 端. 将写好的 C/C++代码使用 NDK 提供的编译工具编译成动态链接库(.so 文件), 然后该链接库将会打包进应用程序最终的 APK 文件. Java 端通过 JNI 的方式调用所生成的动态链接库中的函数, 其系统

结构如图 2 所示.

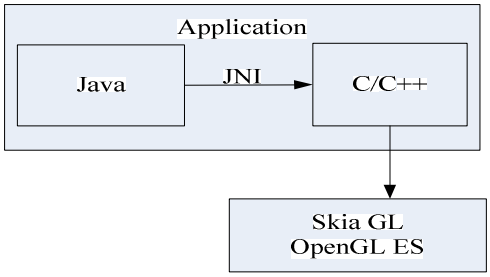

图 2 NDK 方式下图形开发的系统结构图

4.1 NDK 方式下使用 Skia GL 的图形开发

在 NDK 方式下使用 Skia GL 进行图形开发, 需要事先提供 Skia GL 的头文件和链接库, 但是在 NDK 开发工具中并没有包含 Skia 的头文件和链接库. 因此要调用这些文件, 还需要下载 Android 源代码并进行编译, 然后将源代码中的 Skia 头文件以及编译好的 Skia 库文件在 Android.mk 文件中配置好. 不过 Android 系统中已经有这些库文件, 所以部署到手机上运行是没有问题的. 下面介绍通过这个方式实现一个简单得画图功能:

首先创建一个 Android 工程, 在这个工程里新建一个继承于 view 的类, 在 Activity 中创建该类的一个对象, 并将其设为 onCreate()中 setContentView 函数的参数, 替换之前默认的参数, 即将这个 View 设为该 Activity 的主界面. 之后在这个类里面定义一个动态链接库和一个本地函数, 且这个本地函数有个 Canvas 类型的参数:

```
static{ System.loadLibrary("SkiaDemo");}
public native void drawDemo(Canvas canvas);
```

重写 view 类的 onDraw 方法, 在这个方法里调用上面所定义的本地方法 drawDemo,并且把自身获得 Canvas 对象作为参数传给本地函数:

```
protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);
    drawDemo(canvas);
}
```

然后完成在 C/C++中相对应的本地函数, 在函数的开始, 调用 GraphicsJNI::getNativeCanvas 方法处理 Java 端传递过来的 Canvas 对象, 获取 SkCanvas 对象指针, 有了 SkCanvas, 就可以在上面进行绘制自己想要图形了. 在这里做了一个简单得绘图, 绘制了一个红色的正方形:

```
SkCanvas* canv = GraphicsJNI::getNativeCanvas (env, canvas);
    if (!canv) {    return;    }
    SkPaint paint;
    paint.setColor(SK_ColorRED);
    SkRect  r;
    r.set(25, 25, 145, 145);
    canv ->drawRect(r, paint);
```

需要注意的是, 这里使用了 GraphicsJNI 、SkCanvas、SkPaint、SkRect 等几个类, 需要将其相应的头文件引入. 其中 SkCanvas 是 Skia 的绘图上下文, 由它控制直接往哪里绘图(即缓冲屏幕的像素), 维持矩阵和裁剪区域的堆栈, 但是不维持其他的绘图属性(例如颜色、画笔尺寸). 颜色、画笔尺寸这些属性在每次的绘画调用中通过 SkPaint 明确指定, 正方形的具体绘制颜色和风格由 paint 指定, 而不是 canvas. 图 3 是程序在模拟器中运行结果.

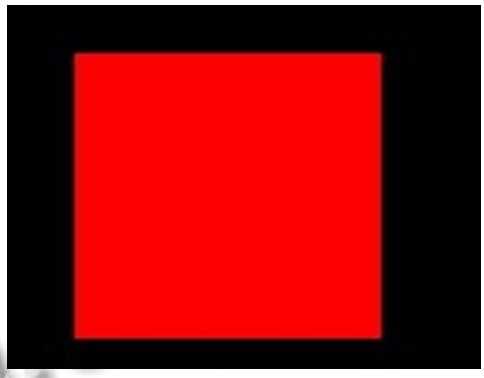

图 3 Skia 开发 Native C/C++开发运行结果

4.2 NDK 方式下使用 OpenGL ES 的图形开发

在 NDK 方式下我们使用 OpenGL ES 进行图形开发, 和上面所讲的使用 Skia GL 开发是类似的, 有所不同的是 OpenGL ES 的相关头文件及库函数在 NDK 开发工具已经包含, 并不需要从源代码里引用. 在 NDK 中, 同时包含了 OpenGL ES 1.0 和 OpenGL ES 2.0 的头文件及库函数(分别为 libGLESv1_CM.so 、libGLESv2.so), 可以根据需要选择使用哪个版本. 在开发过程中, OpenGL 帧渲染以及整个图形状态的维护、修改用 C/C++实现, 提供 JNI 接口给 Java 层, C/C++层被动地按照 Java 层的指示进行渲染. 下面通过画一个旋转的等边三角形示例来说明该方式的实现.

在图 4 中给出了实现这个旋转等边三角形程序的体系结构, 根据程序各部分实现的功能将其划分为四个层次, Activity 层、主 UI 线程层、渲染线程层及 C/C++实现层, 下面逐一分析各层的功能及其实现的方式.

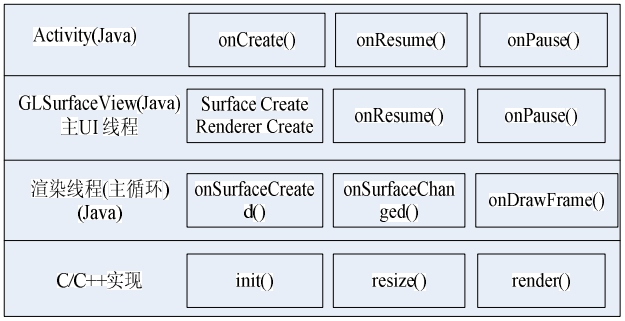

图 4 旋转三角形程序的体系结构

Activity 是程序的入口, 主要功能是构造 GLSurfaceView、Renderer 以及程序生命周期(暂停、恢复)的控制. 其中 onCreate()中的实现为:

```
GLSurfaceView = new GLSurfaceView(this);
GLSurfaceView.setRenderer((Renderer) new TriangleRenderer());
setContentView(GLSurfaceView);
```

这里使用了系统中的 GLSurfaceView 以及程序自身实现的渲染器 TriangleRenderer, 并且把该渲染器注册给 GLSurfaceView. 然都在 onResume()与 onPause()函数中分别调用 GLSurfaceView 的 onResume()与 onPause()函数, 这样程序的生命周期同时也控制了 GLSurfaceView 的生命周期.

GLSurfaceView 其主要功能是提供 OpenGL 的绘图窗口、控制渲染线程的生命周期以及 UI 事件捕获、分发到渲染线程. 在 GLSurfaceView 注册渲染器时会启动渲染线程, 之后主 UI 线程和渲染线程会发生一系列的交互, 由于在这里使用系统中的 GLSurfaceView, 并不需要再做相关的实现.

渲染线程由 GLSurfaceView 控制(创建、销毁、暂停、恢复等), 主要任务是 OpenGL 初始化和销毁、主循环的帧渲染以及事件转发. 在这里要实现一个渲染器类 TriangleRenderer, 这个类中必须要实现三个方法:onSurfaceCreated(), onSurfaceChanged()及 onDrawFrame(), 主循环的帧渲染中会调用这个三个方法. 这三个方法在这里并没有做相关的实现, 只是分别调用了三个本地函数的接口, 具体的实现由本地 C/C++函数来实现.

C/C++实现层定义了基本的 OpenGL 操作: 初始化图形场景(init)、设置好屏幕分辨率(resize)和渲染一帧(render). 在这里初始化图形场景主要是调用函数 glEnableClientState, 确保客户端可以进行 OpenGL 操作, 屏幕分辨率设置则是将 Java 端传过来的屏幕分辨率参数, 设置 OpenGL 操作的投影矩阵, 而 render 函数的主要实现为:

```
angle += 3;
if (angle > 360) {
angle -= 360;
}
glRotatef(angle, 0.0f, 0.0f, 1.0f);
glColor4f(0.63671875f, 0.76953125f, 0.22265625f, 0.0f);
glVertexPointer(3, GL_FLOAT, 0, triangleCoords);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

其中 angle 是一个浮点型全局静态变量, 指的是图形当前的偏转角度, 每一帧的当前角度比上一帧增加三度, 图形围绕向量(0.0f, 0.0f, 1.0f)旋转到当前角度. Triangle Coords 是一个浮点数组, 存储着每个顶点的坐标值, glVertexPointer 指定好顶点后, 由 glDrawArrays 画出来.

编译及运行程序后, 在模拟器中结果如图 5 所示:

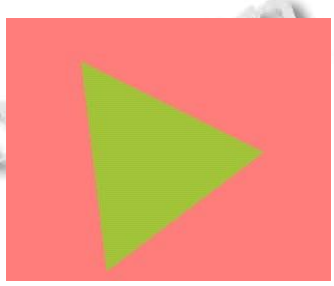

图 5 旋转的等边三角形程序运行结果

5 结语

通过使用 NDK 开发工具在 Android 应用程序中成功调用图形引擎 Skia 和 OpenGL ES 的库函数, 实现使用 C/C++进行应用程序的图形开发. 这就让程序的图形开发不局限于使用 Android SDK, 增加了图形开发的灵活性, 同时这也说明在 Android 系统的应用程序特别是大型游戏开发中, 可以复用其他平台基于 Skia 和 OpenGL ES 图形引擎且使用 C/C++开发好的图形模

(下转第 124 页)

返回一个布尔值, true 表示工作流启动成功, false 表示不成功.

2.1.2 Web Service 处理逻辑

下面以伪代码的方式表述 Web Service 处理逻辑:

```
IF (根据传入参数判断, 如果流程实例不存在)
  {
创建流程实例;
创建任务;
拼接报表连接;
初始化物理表;
}ELSE{                    //流程实例存在
  IF(上报状态=0)   //表示该预算处于在编辑状态
    { 将该记录状态置为 10; }
ELSE{ return FALSE; } //表示预算已经在审批, 不可重新上报
  }
获得下级节点任务;
获得下级节点参与者账户;
关闭当前任务;
创建下一级节点任务;
return TRUE; //表示上报成功
```

2.2 向 BPS 状态表写入审批状态

前述流程设计中, 已经说明了在审批流程的各个节点需要记录的状态. 这些状态需要实时反馈给 BPS 系统中. 预算审批工作流系统通过调用 BPS 中的重置状态表的 RFC 函数来实现.

首先需要在 BPS 系统中自定义一个可供远程调用的 RFC 函数. 该函数根据工作流系统传递的相关状态参数来重写 BPS 的状态表.

工作流需要传递给 BPS 系统的参数有:

1) 公司代码;

2) 部门代码(上级预算审批流程可不传, 为空);

3) 会计年度;

4) 预算期间;

5) 预算类型代码(上级预算审批流程可不传, 为空);

6) 状态值.

BPS 系统返回参数 0, 表示状态写入成功.

其次, 工作流系统在每个节点结束时, 根据当前节点的审批动作确定状态值, 并且调用 RFC 函数.

2.1 以及 2.2 阐述的两个关键技术实现, 是预算工作流系统和 SAP BPS 系统相交互的实现案例. 它的正确有效应用是本系统成功的关键.

3 结语

中国北车全面预算审批系统的实践表明, 上文提出的二级审批流程设计方案以及审批工作流系统与 SAP BPS 系统的交互实现技术具有良好的应用效果, 以及使用效率. 并且具有与 SAP BPS 系统良好的交互性及高效性. 为集团企业预算审批工作提供了有力的技术支撑.

参考文献

1 郑皓,董绍华.人工神经网络销售量预测法在 SAP BPS 系统中的研究与实现.物流技术,2006,12:48–51.

2 胡稳安,及俊川,焦文彬.B/S 模式的预算管理系统的设计与实现.计算机系统应用,2010,19(10):158–162.

3 闪四清.ERP 系统原理和实施.第 2 版.北京:清华大学出版社,2008.

4 张长胜.企业全面预算管理.北京:北京大学出版社,2007.

5 许可,杨路明.工作流技术在企业的应用研究.电脑与信息技术,2006,14(5):22–25.

(上接第 59 页)

块, 或者将 Android 系统使用 C/C++开发好的图形模块复用到其他平台的应用程序中去, 这对于开发不同平台下同一功能的软件有着非常积极的意义.

参考文献

1 金智义,张戟.基于 Android 平台的串口通信实现.电脑知识与技术,2011,5(7):2983–2986.

2 Android Developers. http://developer.android.com/sdk/ndk/overview.html

3 叶炳发,孟小华.Android 图形系统的分析与移植.电信科学,2010.2.

4 淺談 Google Skia 图形处理引擎.[2009-03-21]. http://blog.linux.org.tw/~jserv/archives/002095.html.

5 Android Developers. http://developer.Android.com/guide/topics/graphics/OpenGL.html.