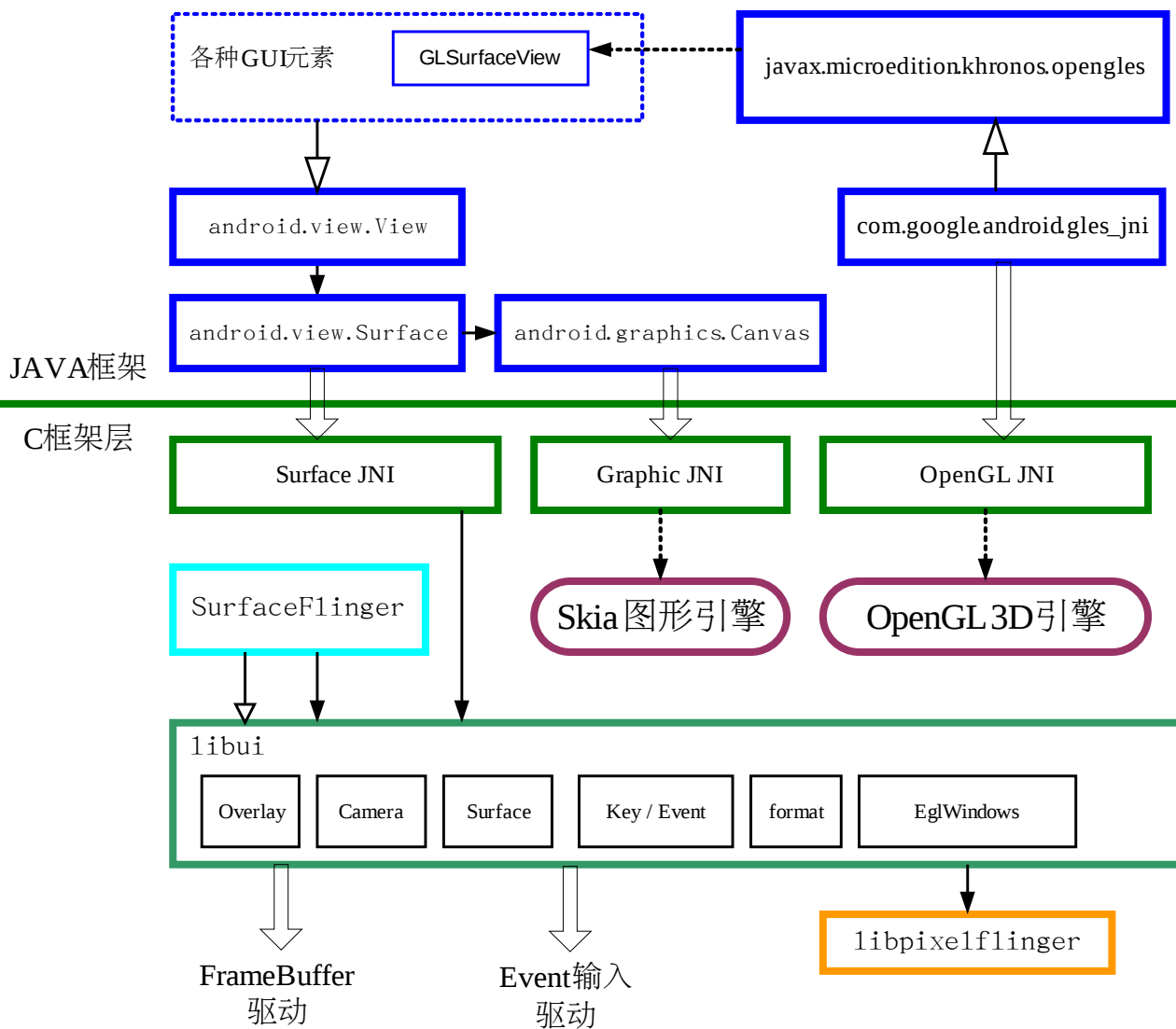


# Android 的 GUI 系统

# Android 的 GUI 系统

- 第一部分 Android GUI 系统综述
- 第二部分 pixelflinger 和 libui 库
- 第三部分 Surface 系统
- 第五部分 Skia 系统
- 第六部分 OpenGL 系统架构

# 第一部分 Android GUI 系统综述



# 第一部分 Android GUI 系统综述

Android 的 GUI 系统由 C 语言的框架和 JAVA 语言的框架组成。

GUI 系统的 C 语言部分包括：

- ❑ PixelFlinger
- ❑ libui (框架库)
- ❑ SurfaceFlinger (Surface 的管理)
- ❑ Skia 图形图像引擎
- ❑ OpenGL 3D 引擎
- ❑ 各种 JNI (向 JAVA 提供接口)

# 第一部分 Android GUI 系统综述

GUI 系统 JAVA 语言的核心包括:

- ❑ `android.graphics` (对应 **Skia** 底层库)
- ❑ `android.view.Surface` (构建显示界面)
- ❑ `android.view.View` 及其继承者  
(用于构建 **UI** 元素)
- ❑ **OpenGL** 的功能类

`javax.microedition.khronos.opengles`  
(由 `com.google.android.gles_jni`  
实现)

## 第二部分 pixelflinger 和 libui 库

2.1 pixelflinger

2.2 libui

## 2.1 pixelflinger

**libpixelflinger.so** 是一个下层的工具性的类，这个类对外的主要内容是 **GGLContext** 结构，以及初始化和卸载的函数。

[system/core/include/pixelflinger/](#)  
[system/core/libpixelflinger/](#)

```
ssize_t gglInit(GGLContext** context);  
ssize_t gglUninit(GGLContext* context);
```

**libpixelflinger.so** 这个库对其他的库没有依赖，也并不提供实际的功能，类似一个用于管理工具的库。

## 2.2 libui

**libui** 是一个框架性质的集成库，它不仅是显示的中枢，也是整个 **GUI** 系统的中枢。

**UI lib** ( **libui** → **libpixelflinger** )，这个的相关内容在以下的路径中：

[frameworks/base/include/ui/](#)  
[framework/base/libs/ui/](#)

**libui** 包含了颜色格式，用于实际显示的 **Egl** 窗口，按键及事件处理，**Surface** 界面，**Overlay**，**Camera** 等几个方面的接口。



## 2.2 libui

### **format 部分:**

这个部分本身定义颜色空间的枚举和数据结构，它需要充用 `pixelflinger` 中的一些关于数据格式定义。

### **EglWindows 部分:**

包含了 EGL 头文件构建的 `egl_native_window_t`，它依赖 OpenGL 的结构，并给 `libEGL` 使用的。`EGLDisplaySurface` 操作了硬件的 `framebuffer` 的驱动。这也是整个系统显示的基础。

### **Key/Event 部分:**

这是 Android 系统输入的基础，其中定义按键的映射，通过操作 `event` 事件设备来实现获取系统的输入的。

## 2.2 libui

### **Surface :**

Surface 相关的头文件和实现为 SurfaceFlinger 定义接口和框架。

### **Overlay :**

定义视频输出的接口。

### **Camera :**

定义摄像头的框架和接口。

## 2.3 Android 的显示输出系统

Android 使用标准的 **framebuffer** 作为驱动程序，Android 的本地框架中提供了系统和 **framebuffer** 驱动程序之间的适配层（硬件抽象层）。

输出部分的硬件抽象（**donut** 之前）：

**EGLDisplaySurface.cpp**

调用标准的 **FrameBuffer** 驱动。

## 2.3 Android 的显示输出系统

**Gralloc Module** 是 **Eclair** 版本之后显示部分的抽象层，它是系统和 **Framebuffer** 设备的接口，以硬件模块的形式存在。

头文件路径：

[hardware/libhardware/include/hardware/gralloc.h](#)

**Gralloc** 模块实现：

[hardware/libhardware/modules/gralloc/](#)

**Gralloc** 被 **libui** 使用。

## 2.3 Android 的显示输出系统

**Gralloc.h** 中包含了 **Gralloc** 模块所需要的接口。

```
typedef struct gralloc_module_t {
    struct hw_module_t common;
    int (*registerBuffer)(struct gralloc_module_t const* module,
        buffer_handle_t handle);
    int (*unregisterBuffer)(struct gralloc_module_t const* module,
        buffer_handle_t handle);
    int (*lock)(struct gralloc_module_t const* module,
        buffer_handle_t handle, int usage,
        int l, int t, int w, int h,
        void** vaddr);
    int (*unlock)(struct gralloc_module_t const* module,
        buffer_handle_t handle);
    int (*perform)(struct gralloc_module_t const* module,
        int operation, ... );
    void* reserved_proc[7];
} gralloc_module_t;
```

## 2.3 Android 的显示输出系统

```
enum {
    GRALLOC_USAGE_SW_READ_NEVER    = 0x00000000,
    GRALLOC_USAGE_SW_READ_RARELY   = 0x00000002,
    GRALLOC_USAGE_SW_READ_OFTEN    = 0x00000003,
    GRALLOC_USAGE_SW_READ_MASK     = 0x0000000F,

    GRALLOC_USAGE_SW_WRITE_NEVER    = 0x00000000,
    GRALLOC_USAGE_SW_WRITE_RARELY   = 0x00000020,
    GRALLOC_USAGE_SW_WRITE_OFTEN    = 0x00000030,
    GRALLOC_USAGE_SW_WRITE_MASK     = 0x000000F0,

    /* buffer will be used as an OpenGL ES texture */
    GRALLOC_USAGE_HW_TEXTURE        = 0x00000100,
    /* buffer will be used as an OpenGL ES render target */
    GRALLOC_USAGE_HW_RENDER         = 0x00000200,
    /* buffer will be used by the 2D hardware blitter */
    GRALLOC_USAGE_HW_2D              = 0x00000C00,
    /* buffer will be used with the framebuffer device */
    GRALLOC_USAGE_HW_FB              = 0x00001000,
    /* mask for the software usage bit-mask */
    GRALLOC_USAGE_HW_MASK           = 0x00001F00,
};
```

## 2.3 Android 的显示输出系统

Gralloc 模块是显示模块的实现。其中，`framebuffer.cpp` 用于基于 `framebuffer` 的显示实现，`gralloc` 是基于 `pmem` 的实现。

```
int gralloc_device_open(const hw_module_t* module, const char* name,
                        hw_device_t** device)
{
    int status = -EINVAL;
    if (!strcmp(name, GRALLOC_HARDWARE_GPU0)) {
        gralloc_context_t *dev;
        dev = (gralloc_context_t*)malloc(sizeof(*dev));
        memset(dev, 0, sizeof(*dev));
        dev->device.common.tag = HARDWARE_DEVICE_TAG;
        dev->device.common.version = 0;
        dev->device.common.module = const_cast<hw_module_t*>(module);
        dev->device.common.close = gralloc_close;
        dev->device.alloc = gralloc_alloc;
        dev->device.free = gralloc_free;
        *device = &dev->device.common;
        status = 0;
    } else {
        status = fb_device_open(module, name, device);
    }
    return status;
}
```

## 2.3 Android 的显示输出系统

libui 对 gralloc 模块实现了调用，在 [ui/FramebufferNativeWindow.cpp](#) 中打开了 gralloc 设备。

```
FramebufferNativeWindow::FramebufferNativeWindow()
: BASE(), fbDev(0), grDev(0), mUpdateOnDemand(false)
{
    hw_module_t const* module;
    if (hw_get_module(GRALLOC_HARDWARE_MODULE_ID, &module) == 0) {
        int stride;
        int err;
        err = framebuffer_open(module, &fbDev);
        LOGE_IF(err, "couldn't open framebuffer HAL (%s)", strerror(-err));

        err = gralloc_open(module, &grDev);
        LOGE_IF(err, "couldn't open gralloc HAL (%s)", strerror(-err));
        // .....
    }
}
```



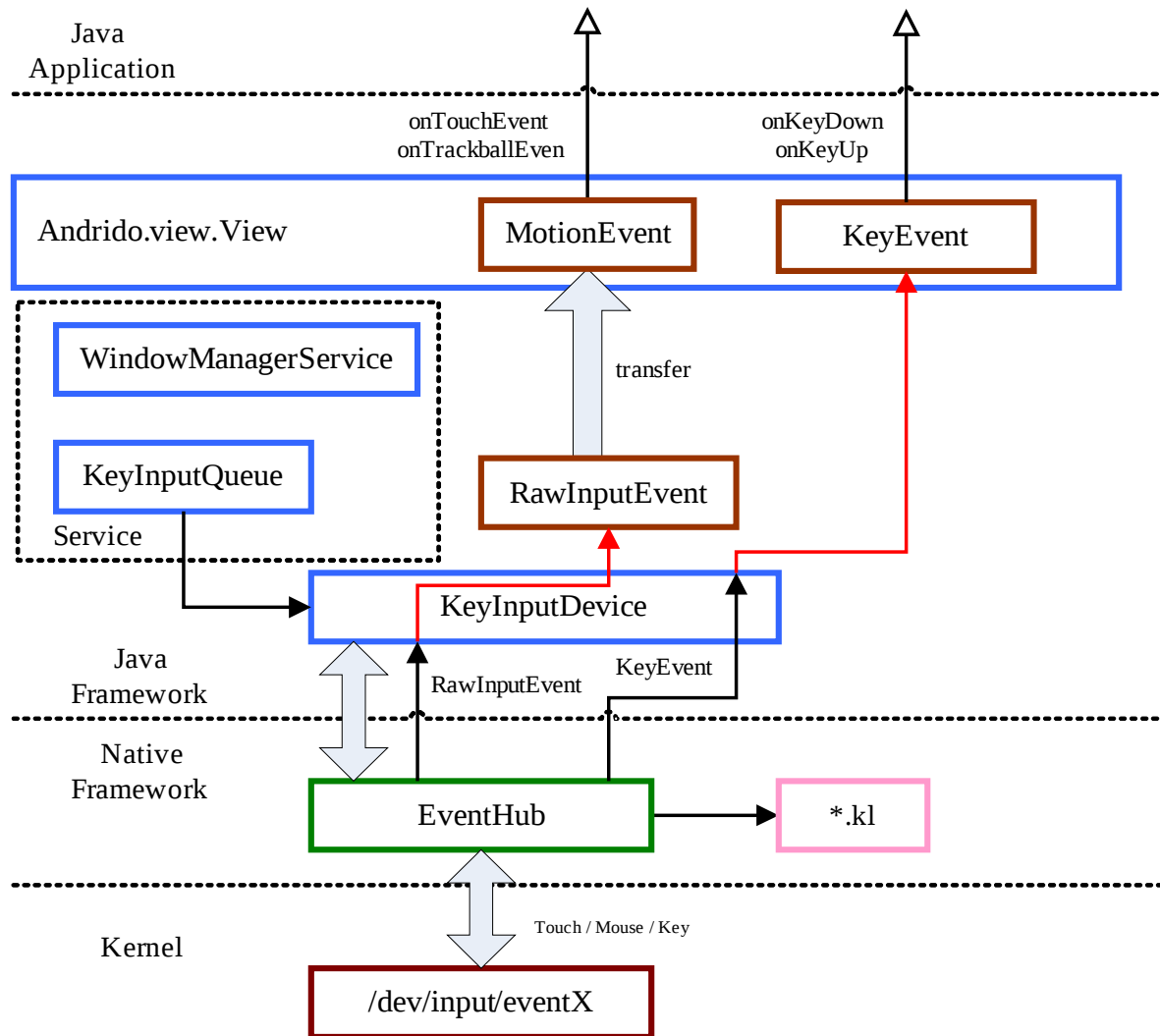
## 2.4 Android 的用户输入系统

Android 输入系统由 **Event** 驱动程序，**libui** 中的 **EventHub** 和 **JAVA** 框架中的几个类组成。

**Event** 的功能被集成在 **android.view** 包的 **View** 类中，在应用程序层调用主要通过 **View** 类及其继承者。

**KeyEvent** 和 **MotionEvent** 的处理方法略有不同，**KeyEvent** 通过转化按键码得到，**MotionEvent** 通过转化 **RawEvent** 得到。

## 2.4 Android 的用户输入系统



## 2.4 Android 的用户输入系统

输入部分的硬件抽象：

**EventHub.cpp**

调用标准的 **Event** 设备驱动。

KeyCodeLabel.h <=> android/view/KeyEvent.java

qwerty.kl 键盘布局文件

## 2.4 Android 的用户输入系统

**Multi-Touch** 是 **Eclair** 版本的新特性。

多点触摸的特性，需要从硬件到软件系统的支持。作为 **Android** 的 **GUI** 系统，最终的就是将消息从下传递到上面。

输入设备中增加了一个新的类型：  
**TOUCHSCREEN\_MT**（**EventHub.h** 定义），  
**EventHub** 获得信息只有交由 **JAVA** 层处理。

## 2.4 Android 的用户输入系统

`android.view.RawInputEvent` 增加多点数据表示, `InputDevice` 作处理, 并保持对非多点触摸的兼容性, `KeyInputQueue` (`android/server/KeyInputQueue`) 进行多点数据的转化。

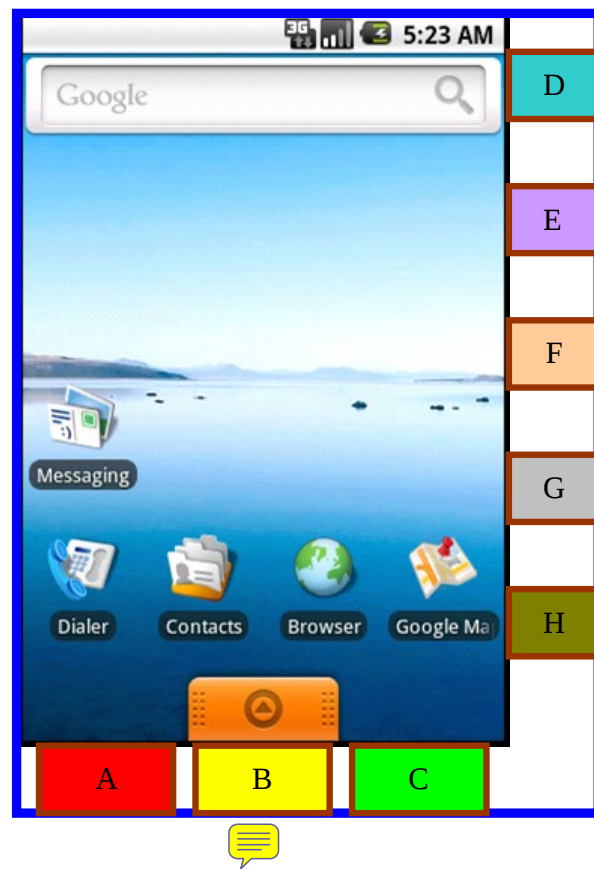
`MotionEvent` 增加了对多点的支持, 最多支持同时有 3 个点触摸。这是上层程序中用到的接口。

注意: 没有手势方面的解析, 需要应用程序根据多点信息自己实现解释。

## 2.4 Android 的用户输入系统

**Virtual Key** 是 Eclair 版本的新特性。

**Virtual Key** 的功能是利用触摸屏，模拟按键发生的事件，这样就可以利用触摸屏的边缘，实现一些可以自定义的按键效果。



## 2.4 Android 的用户输入系统

从应用程序的角度，触摸屏设备发送的是 **RawInputEvent**（原始输入事件），而按键发送的是 **KeyEvent**（按键事件）。虚拟按键的作用是在某种情况下，将 **RawInputEvent** 转换成 **KeyEvent**。

按键文件：

[/sys/board\\_properties/virtualkeys.{devicename}](#)



代码路径：

[services/java/com/android/server/InputDevice.java](#)

通过 **readVirtualKeys**，进行消息的转化，将 **RawInputEvent** 转换成 **KeyEvent**。

## 2.4 Android 的用户输入系统

`virtualkeys.{devicename}` 是虚拟按键的适配文件，需要放置在目标系统的以下目录中：  
**`/sys/board_properties/`**

文件的格式如下所示：

**`0x1: 扫描码 :X:Y:W:H:0x1: .....`**



## 第三部分 Surface 系统

3.1 Surface 系统的结构

3.2 SurfaceFlinger 本地代码

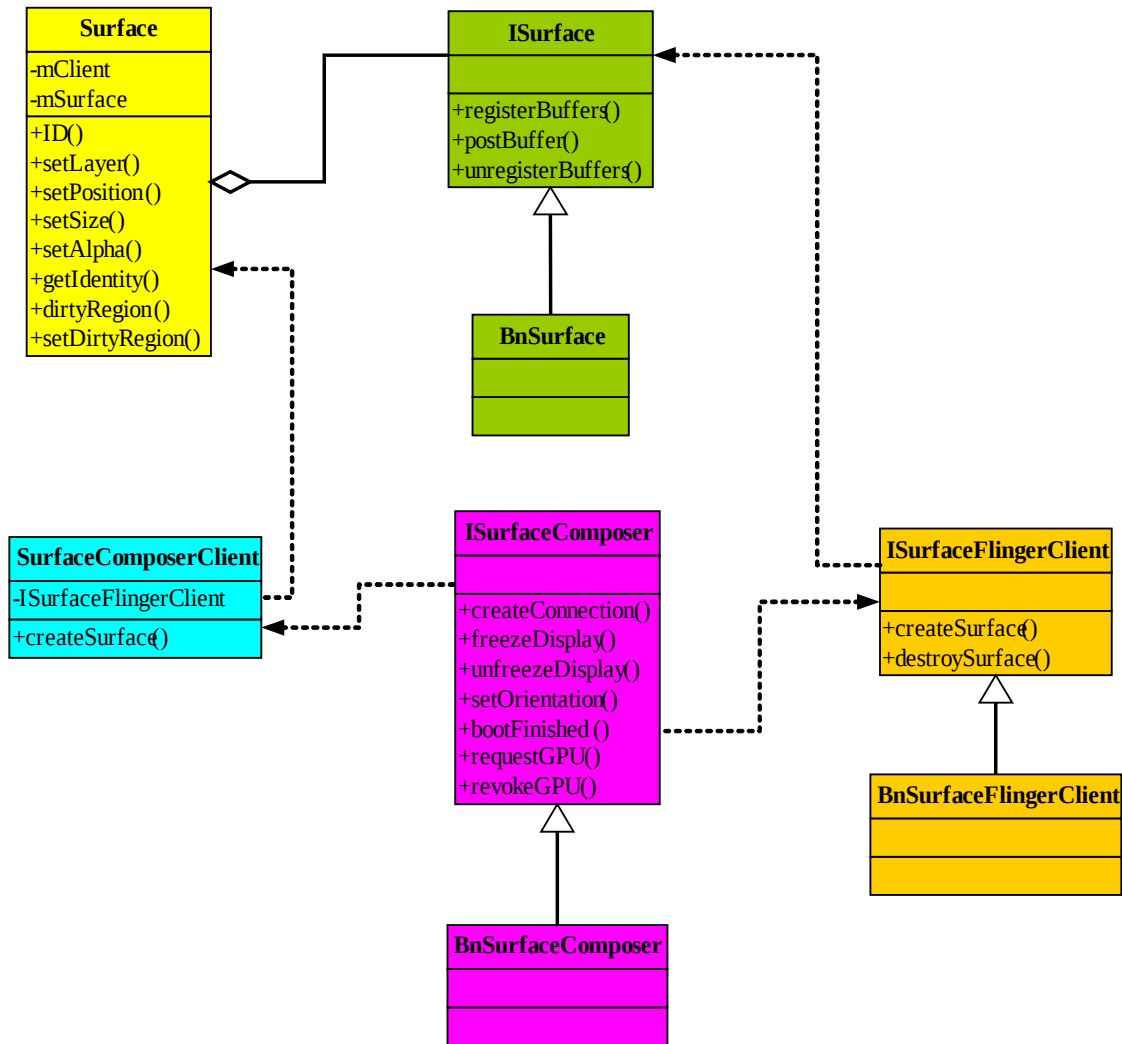
3.3 Surface 的 Java 和 Jni 代码

## 3.1 Surface 系统结构

Surface 系统的结构:

- ❑ `libui.so` 提供与 Surface 接口。
- ❑ `libsurfacefinger.so` 提供实现。
- ❑ Java 框架层次主要调用 Surface 向 UI 提供接口。
- ❑ Native（本地调用）部分主要使用 `ISurface`。

## 3.1 Surface 系统的结构



# 3.1 Surface 系统的结构

## Surface 系统的头文件

(路径为: [frameworks/base/include/ui/](#)) :

ISurface.h

ISurfaceComposer.h

ISurfaceFlingerClient.h

Surface.h

SurfaceComposerClient.h

## Surface 系统的源代码文件

(路径为: [frameworks/base/libs/surfaceflinger/](#)) :

ISurfaceFlingerClient.cpp

SurfaceComposerClient.cpp

IsurfaceComposer.cpp

Surface.cpp

Isurface.cpp

SurfaceFlinger 类继承了 ISurfaceComposer , 是一个核心的实现。

## 3.1 Surface 系统的结构

```
class ISurfaceComposer : public IInterface
{
public:
    enum { // (keep in sync with Surface.java)
        eHidden          = 0x00000004,
        eGPU              = 0x00000008,
        eHardware         = 0x00000010,
        eDestroyBackbuffer = 0x00000020,
        eSecure           = 0x00000080,
        eNonPremultiplied  = 0x00000100,
        ePushBuffers       = 0x00000200,
        eFXSurfaceNormal   = 0x00000000,
        eFXSurfaceBlur     = 0x00010000,
        eFXSurfaceDim      = 0x00020000,
        eFXSurfaceMask     = 0x000F0000,    };
    enum {
        ePositionChanged    = 0x00000001,
        /* ... */    };
    enum {
        eLayerHidden        = 0x01,
        /* ... */    };
    enum {
        eOrientationDefault  = 0,
        /* ... */    };
}
```

## 3.2 SurfaceFlinger 本地代码

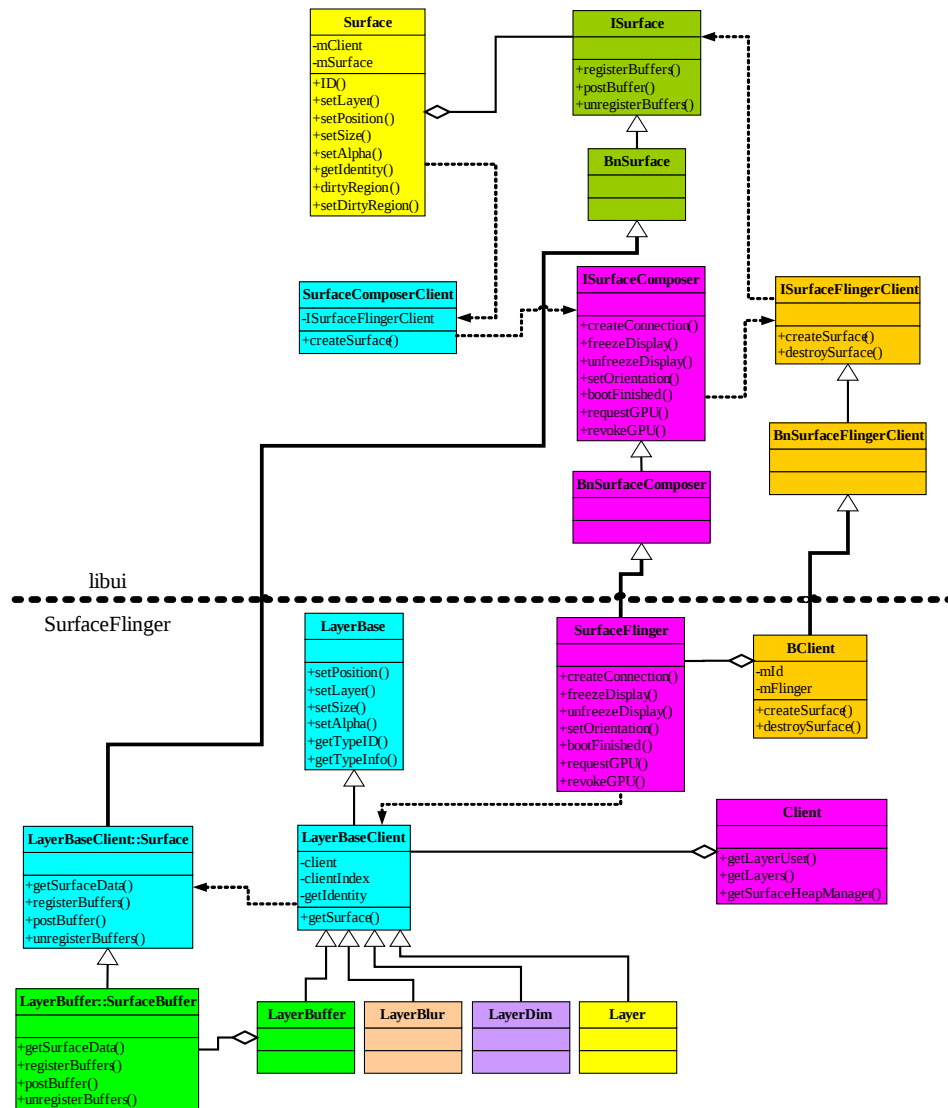
代码的路径：

[frameworks/base/libs/surfaceflinger/](#)

```
class LayerBaseClient : public LayerBase
    class Surface : public BnSurface
```


```
class Layer : public LayerBaseClient
class LayerBuffer : public LayerBaseClient
    class SurfaceBuffer : public LayerBaseClient::Surface
class LayerDim : public LayerBaseClient
class LayerBlur : public LayerBaseClient
```

## 3.2 SurfaceFlinger 本地代码



## 3.2 SurfaceFlinger 本地代码

### For Create a Surface:

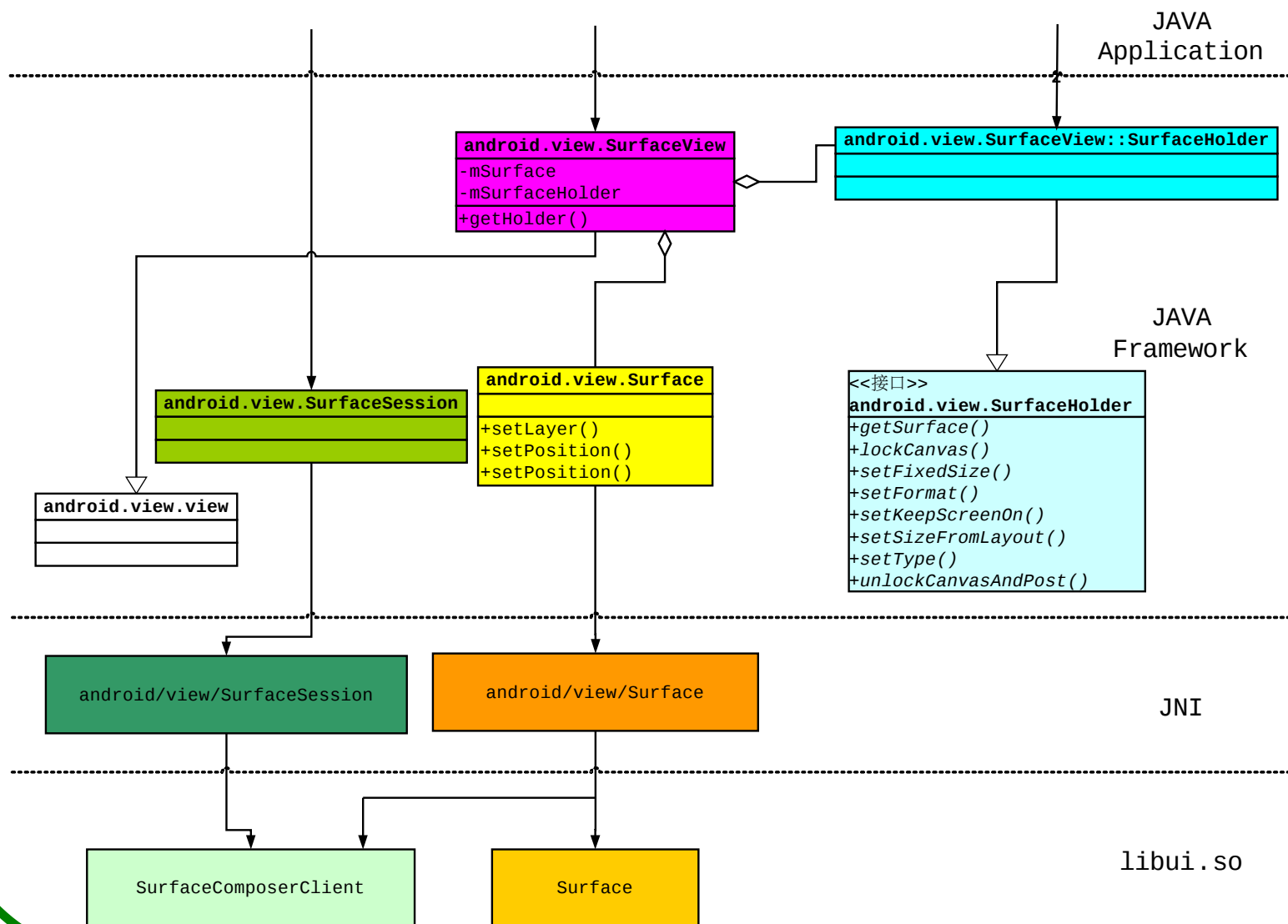
```
SurfaceComposerClient::createSurface  
(ISurface → Suface,  
frameworks/base/libs/ui/SurfaceComposerClient.cpp)  
→ (IsurfaceFlingerClient:: createSurface)  
→ Bclient::createSurface  
(framework/base/libs/surfaceflinger/SurfaceFlinger.cpp)  
→ SurfaceFlinger::createSurface  
(framework/base/libs/surfaceflinger/SurfaceFlinger.cpp)  
LayerBaseClient* layer = 0;  
sp<LayerBaseClient::Surface> surfaceHandle;  
    eFXSurfaceNormal → Layer or LayerBuffer  
    eFXSurfaceBlur → LayerBlur   
    eFXSurfaceDim → LayerDim  
surfaceHandle = layer->getSurface();
```

### For setSize:

```
Surface::setSize()  
→ SurfaceComposerClient::setSize()
```



# 3.3 Surface 的 Java 和 Jni 代码



## 3.3 Surface 的 Java 和 Jni 代码

**JNI Code path:**

[frameworks/base/core/jni/android\\_view\\_Surface.cpp](#)

```
#include <ui/SurfaceComposerClient.h>
const char* const kSurfaceSessionClassName
    = "android/view/SurfaceSession";
const char* const kSurfaceClassName
    = "android/view/Surface";
static void nativeClassInit(JNIEnv* env, jclass clazz);
```

## 3.3 Surface 的 Java 和 Jni 代码

**JNI 代码的路径：**

[frameworks/base/core/jni/android\\_view\\_Surface.cpp](#)

```
#include <ui/SurfaceComposerClient.h>
const char* const kSurfaceSessionClassName
    = "android/view/SurfaceSession";
const char* const kSurfaceClassName
    = "android/view/Surface";
static void nativeClassInit(JNIEnv* env, jclass clazz);
```

## 3.3 Surface 的 Java 和 Jni 代码

Java 代码的路径：

[frameworks/base/core/java/android/view/](#)

3 个主要的类：

Surface (Surface.java)

SurfaceView (SurfaceView.java)

SurfaceSession (SurfaceSession.java)

1 个接口：

SurfaceHolder (SurfaceSession.java)

`android.view.Surface` 表示了一个可以绘制图形的界面，它是实际调用底层的 `Surface` 接口来完成来控制的硬件载体。

## 3.3 Surface 的 Java 和 Jni 代码

关于 `android.view.View` 类:

**View** 类呈现了最基本的 UI 构造块。一个视图占据屏幕上的一个方形区域，并且负责绘制和时间处理。

**View** 是 **widgets** 的基类，常用来创建交互式的用户图形界面 (GUI)。

**View** 中包含了一个 **Surface**，并处理 `onDraw(Canvas)` 事件。

## 第四部分 Skia 及 android 的图形系统

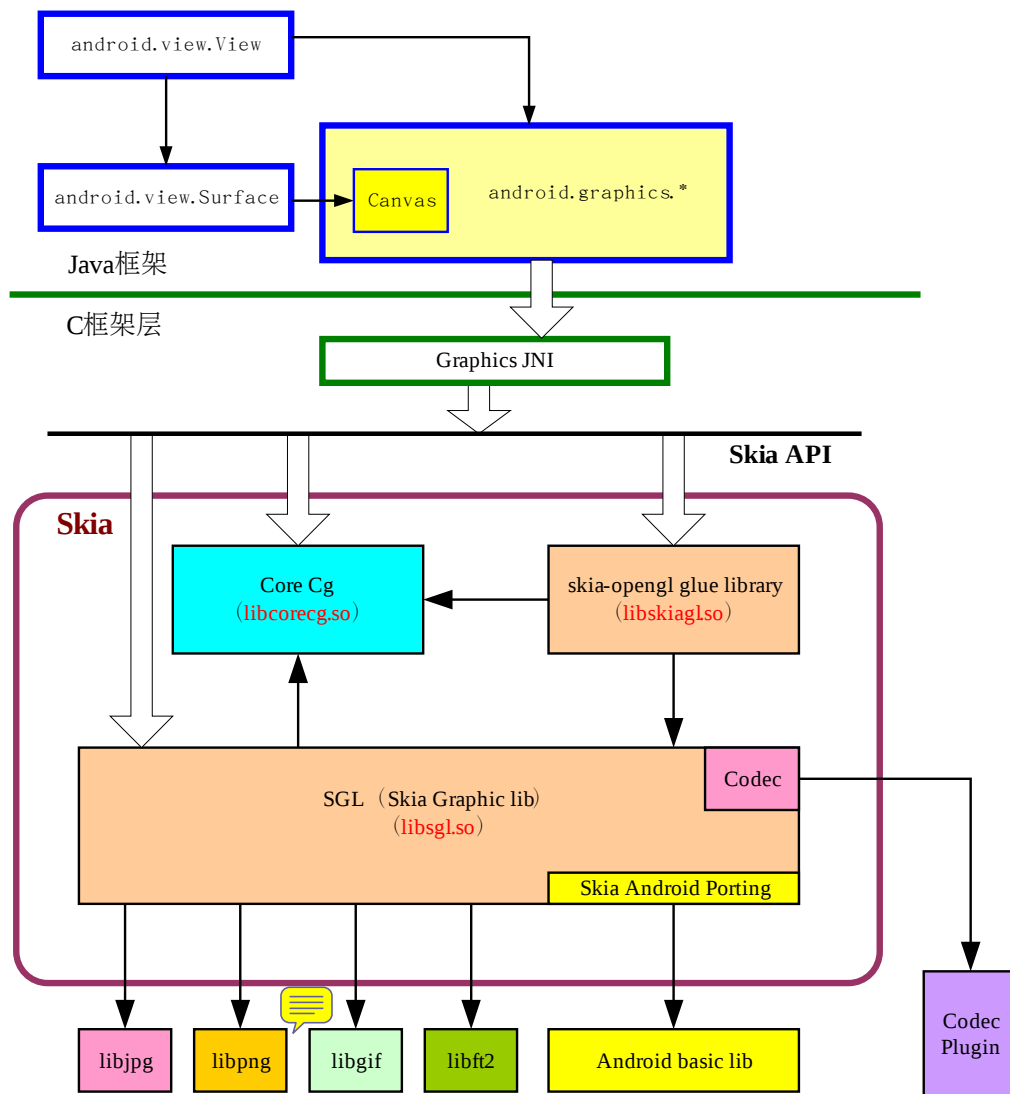


4.1 Skia 底层库

4.2 Android 图形系统的 JNI 接口

4.3 Android 的图形类

# 第四部分 Skia 及 android 的图形系统



## 4.1 Skia 底层库

Skia 是 Google 一个底层的图形，图像，动画，SVG，文本等多方面的图形库，它是 Android 中图形系统的引擎。

Skia 代码的路径：

[external/skia/](#)

Skia 主要包含三个库：

- ❑ Core Cg : **libcorecg.so**
- ❑ GL ( Skia Graphic Lib ) : **liblibsgl.so**
- ❑ skia-opengl glue library : **libskiagl.so**



## 4.1 Skia 底层库

Skia 主要包含三个库的代码路径:

▣ 核心图形库: **libcorecg.so**

[src/core/](#)

▣ Skia 图形库: **liblibsgl.so**

[src/effects/](#)

[src/images/](#)

[src/ports/](#)

[src/core/](#)

[src/utils/](#)

▣ skia-opengl glue library : **libskiagl.so**

[src/gl/](#)

## 4.2 Android 图形系统的 JNI 接口

Android 的图形系统和 Skia 底层库的联系比较紧密。Android 的图形系统的 JNI 提供了从 Skia 底层库到 JAVA 上层的支持。

Android 的图形系统的 JNI 代码的路径:

[frameworks/base/core/jni/android/graphic/](#)

Android 的图形系统对 JAVA 层提供了绘制基本图形的功能，是 GUI 系统的基

## 4.2 Android 图形系统的 JNI 接口

Canvas.cpp 是 JNI 中核心的接口，为 JAVA 上层的 `android.graphics.Canvas` 类提供了支持。

```
static SkCanvas* initRaster(JNIEnv* env, jobject, SkBitmap* bitmap) {  
    return bitmap ? new SkCanvas(*bitmap) : new SkCanvas;  
}  
static SkCanvas* initGL(JNIEnv* env, jobject) {  
    return new SkGLCanvas;  
}
```

## 4.3 Android 的图形类

Android 的图形类的包是 `android.graphics` , 它通过调用图形系统的 JNI 提供了对 JAVA 框架中图形系统的支持。

Android 的图形系统的代码的路径:

[frameworks/graphics/java/android/graphics/](#)

## 4.3 Android 的图形类

Canvas.java 实现图形系统中最为重要的一个类：**android.graphics.Canvas**。

**Canvas** 类处理 “draw” 的调用，当绘制（draw）内容的时候需要 4 个基本的组件：一个保持像素的 **A Bitmap**，一个处理绘制调用的 **Canvas**（写入 bitmap），绘制的内容（例如 **Rect**，**Path**，**text**，**Bitmap**）和一个 **paint**（用来描述颜色和样式）。

# 第五部分 OpenGL 系统

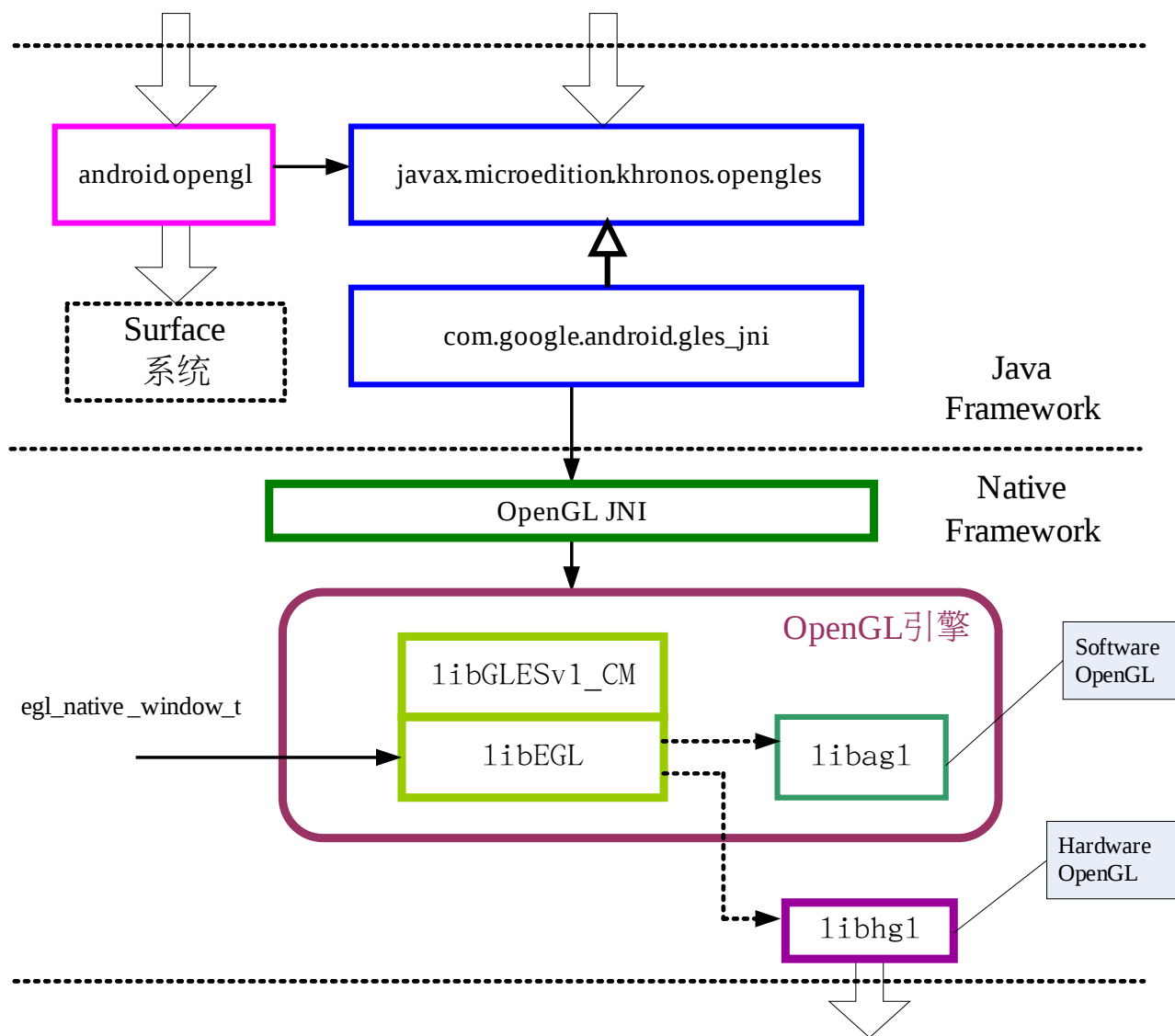


5.1 OpenGL 系统结构

5.2 OpenGL 的 Native 代码

5.3 OpenGL 的 JAVA 和 JNI 代码

## 5.1 OpenGL 系统结构



## 5.2 OpenGL 的 Native 代码

OpenGL 的本地代码：

[frameworks/base/opengl/libagl/](#)  
[frameworks/base/opengl/libs/](#)

OpenGL 的本地头文件：

[frameworks/base/opengl/include/EGL/](#)  
[frameworks/base/opengl/include/GLES/](#)

OpenGL 的库：

- ❑ libEGL.so （EGL 库）
- ❑ libGLSv1\_CM.so （OpenGL ES 库的封装）
- ❑ libagl.so （OpenGL 的软件实现库）



## 5.2 OpenGL 的 Native 代码

测试代码的路径：

[frameworks/base/opengl/tests](#)

使用 **Surface** 的代码：\_

[frameworks/base/libs/surfaceflinger/](#)  
[DisplayHardware/DisplayHardware.cpp](#)

另外的一个本地的接口：

[frameworks/base/opengl/include/EGL/egl natives.h](#)

```
struct egl_native_window_t* android_createDisplaySurface();
```

其实现：

[frameworks/base/libs/ui/EGLDisplaySurface.cpp](#)

```
egl_native_window_t* android_createDisplaySurface();
```

## 5.2 OpenGL 的 Native 代码

Android Eclair 中版本之后 `libagl` 的库名称为 `libGLES_android.so`，放置到目标系统的目录 [system/lib/egl](#)。

新增 OpenGL ES2 的支持：

[frameworks/base/opengl/libs/ELGS\\_CM](#)

库的名称为 `libGLESv2.so`，它与 `libGLESv1_CM.so` 是并列关系。

## 5.2 OpenGL 的 Native 代码

Android Eclair 中版本的 `libegl.so` 将从 [`system/lib/egl`](#) 目录中加载 OpenGL 的实现库，加载的方式由其中的 `egl.cfg` 文件来决定，OpenGL 的软件实现 `libGLES_android.so`，通常作为加载的一个库。另外还可以选择加载硬件实现的 OpenGL 库。

`egl.cfg` 文件的实例如下：

```
0 0 android
0 1 XXX
```

## 5.2 OpenGL 的 Native 代码

```
#include <EGL/egl.h>
#include <GLES/gl.h>
#include <GLES/glext.h>
int main(int argc, char** argv)
{
    EGLint s_configAttribs[] = {
        EGL_RED_SIZE, 5, EGL_GREEN_SIZE, 6, EGL_BLUE_SIZE, 5, EGL_NONE
    };
    EGLint numConfigs = -1; EGLint majorVersion; EGLint minorVersion;
    EGLConfig config; EGLContext context;
    EGLSurface surface; EGLint w, h;
    EGLDisplay dpy;
    dpy = eglGetDisplay(EGL_DEFAULT_DISPLAY);
    eglInitialize(dpy, &majorVersion, &minorVersion);
    eglChooseConfig(dpy, s_configAttribs, &config, 1, &numConfigs);
    surface = eglCreateWindowSurface(dpy, config,
        android_createDisplaySurface(), NULL);
    context = eglCreateContext(dpy, config, NULL, NULL);
    eglMakeCurrent(dpy, surface, surface, context);
    eglQuerySurface(dpy, surface, EGL_WIDTH, &w);
    eglQuerySurface(dpy, surface, EGL_HEIGHT, &h);
    GLint dim = w < h ? w : h;
```

## 5.2 OpenGL 的 Native 代码

```
glBindTexture(GL_TEXTURE_2D, 0);
glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexEnvx(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
glEnable(GL_TEXTURE_2D);
glColor4f(1,1,1,1);
glDisable(GL_DITHER);
glShadeModel(GL_FLAT);
/* ..... */
return 0;
}
```

## 4.3 OpenGL 的 JAVA 和 JNI 代码

Android 的 OpenGL 的实现方法是使用一个类来继承 OpenGL JAVA 的标准类，通过对这个类的实现，实现 OpenGL 的功能，在 JAVA 层只需要使用标准类。

javax.microedition.khronos.opengles ( JAVA 标准类 ) :

[opengl/java/javax/microedition/khronos/egl/](#)

(GL10.java GL11.java)

[opengl/java/javax/microedition/khronos/opengles/](#)

(EGL10.java)

com.google.android.gles\_jni (Android 的 GLES 的实现 )

[opengl/java/com/google/android/gles\\_jni/](#)

com.google.android.gles\_jni JNI:

[core/jni/com google android gles\\_jni GLImpl.cpp](#)

[core/jni/com google android gles\\_jni EGLImpl.cpp](#)

## 5.3 OpenGL 的 JAVA 和 JNI 代码

### **JAVA** 代码:

javax.microedition.khronos.opengles ( JAVA 标准类) :  
[opengl/java/javax/microedition/khronos/egl/](#)  
(GL10.java GL11.java)  
[opengl/java/javax/microedition/khronos/opengles/](#)  
(EGL10.java)

com.google.android.gles\_jni (Android 的 GLES 的实现类 )  
[opengl/java/com/google/android/gles\\_jni/](#)

### **JNI** 代码:

com.google.android.gles\_jni JNI:  
[core/jni/com google android gles\\_jni GLImpl.cpp](#)  
[core/jni/com google android gles\\_jni EGLImpl.cpp](#)

## 5.3 OpenGL 的 JAVA 和 JNI 代码

Android 的 `android.opengl` 是一个 OpenGL 相关的包，它主要提供了为 OpenGL 的输出界面。

`android.opengl` 的代码路径：  
[opengl/java/android/opengl/](#)

核心的文件：  
[opengl/java/android/opengl/GLSurfaceView.java](#)

```
public class GLSurfaceView extends SurfaceView  
    implements SurfaceHolder.Callback
```



## 5.3 OpenGL 的 JAVA 和 JNI 代码

在 Android 的 JAVA 应用中使用 OpenGL 通常需要 `javax.microedition.khronos.opengles` 类和 `android.opengl` 包的结合使用。

```
public class XXXActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mGLSurfaceView = new XXXGLSurfaceView(this);
        setContentView(mGLSurfaceView);
        mGLSurfaceView.requestFocus();
        mGLSurfaceView.setFocusableInTouchMode(true);
    }
    private GLSurfaceView mGLSurfaceView;
}
class XXXGLSurfaceView extends GLSurfaceView {
    private class CubeRenderer implements GLSurfaceView.Renderer { /* ... */ }
    /* ... */
}
```



谢谢！