MICROOH 麦可网

Android-从程序员到架构师之路

出品人：Sundy
讲师：高焕堂（台湾）

http://www.microoh.com

F02_c

观摩：SurfaceView小框架的未来性设计(c)

By 高煥堂

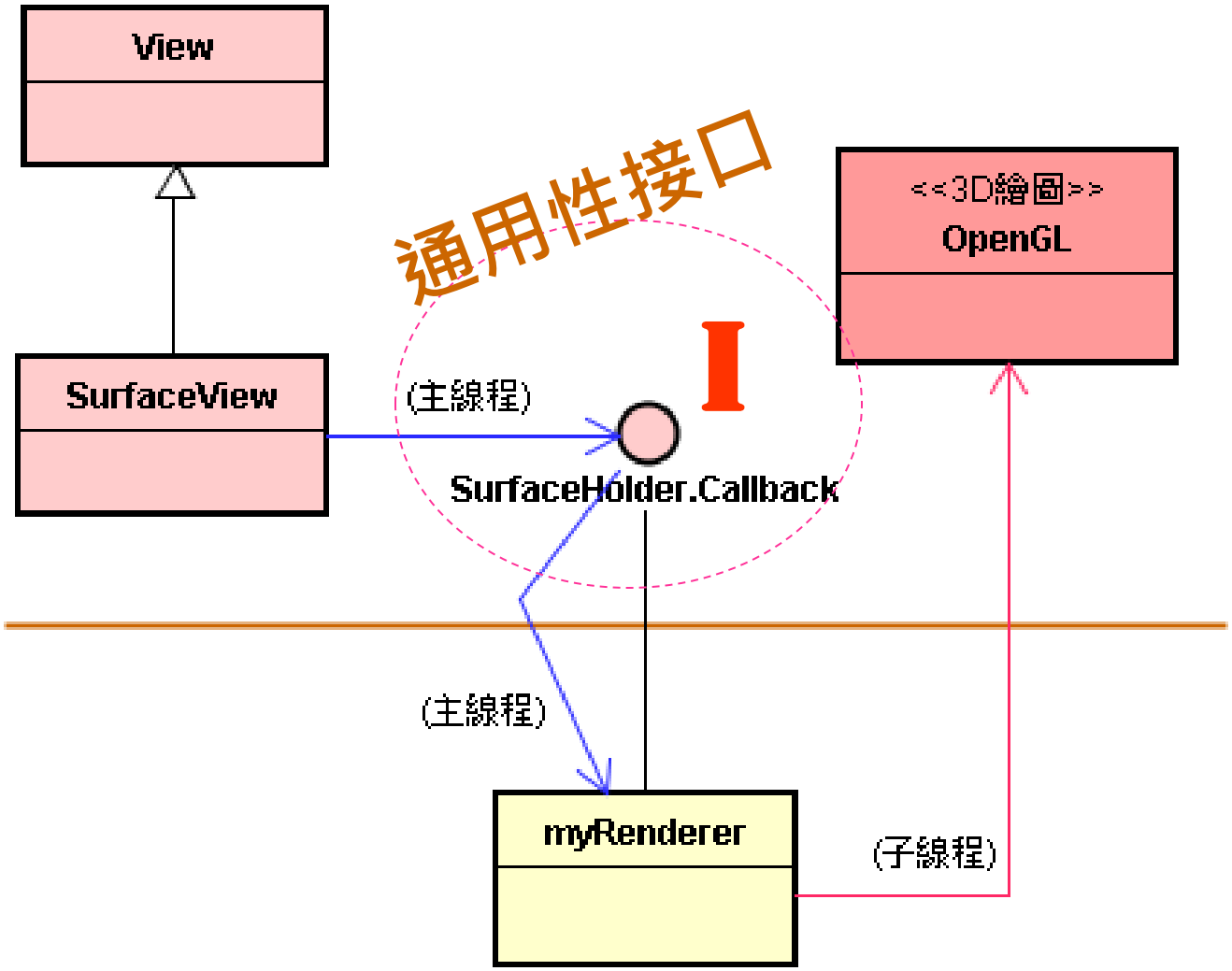
4、使用 OpenGL ES引擎

范例：画出一个旋转的立方体

前言

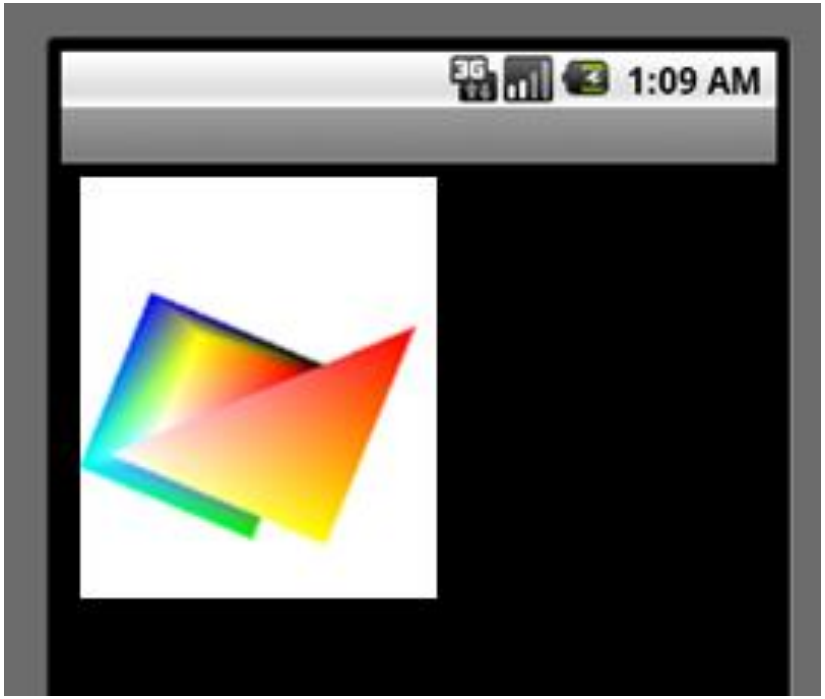
通用性(General) 接口

- SurfaceView小框架就是一个EIT造形，其提供了一个通用性(General)的<I>，就是：SurfaceHolder.Callback接口。


View
通用性接口
<<3D繪圖>>
OpenGL
SurfaceView
(主線程)
SurfaceHolder.Callback
(主線程)
myRenderer
(子線程)

- 架构师设计了这通用性接口，来与形形色色的配件(如Camera、OpenGLES、MediaPlayer等)做非常弹性的组合。
- 基于通用性接口和弹性组合，创造了未来性(能包容业主或用户的未来选择)。

- EX-1：SurfaceView + Callback 接口 + OpenGL ES；显示3D的动态绘图：



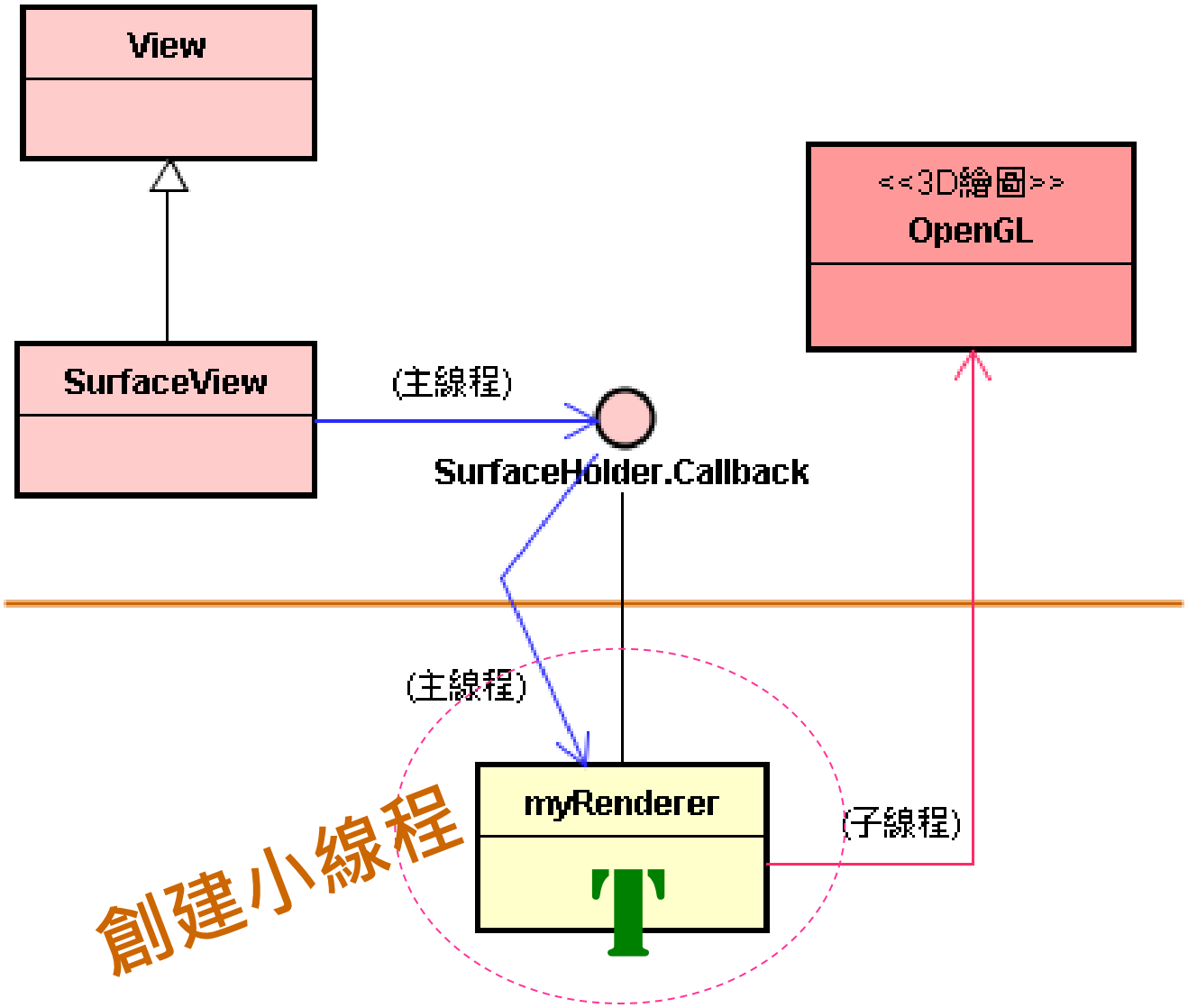
- EX-2：SurfaceView + Callback 接口 + MediaPlayer；播放MP4视频：

通用性接口不能顾及
特殊性的需求

- 例如，OpenGL的绘图是费时的工作，不适合使用UI线程(主线程)，于是产生了特殊需求：需要产生一个新线程(Thread)来担任绘图任务。此线程会去执行一个while循环，将不断地调整旋转角度，并呼叫MyCube对象，重新画出立方体的表面。

特殊性：
OpenGL需要小線程

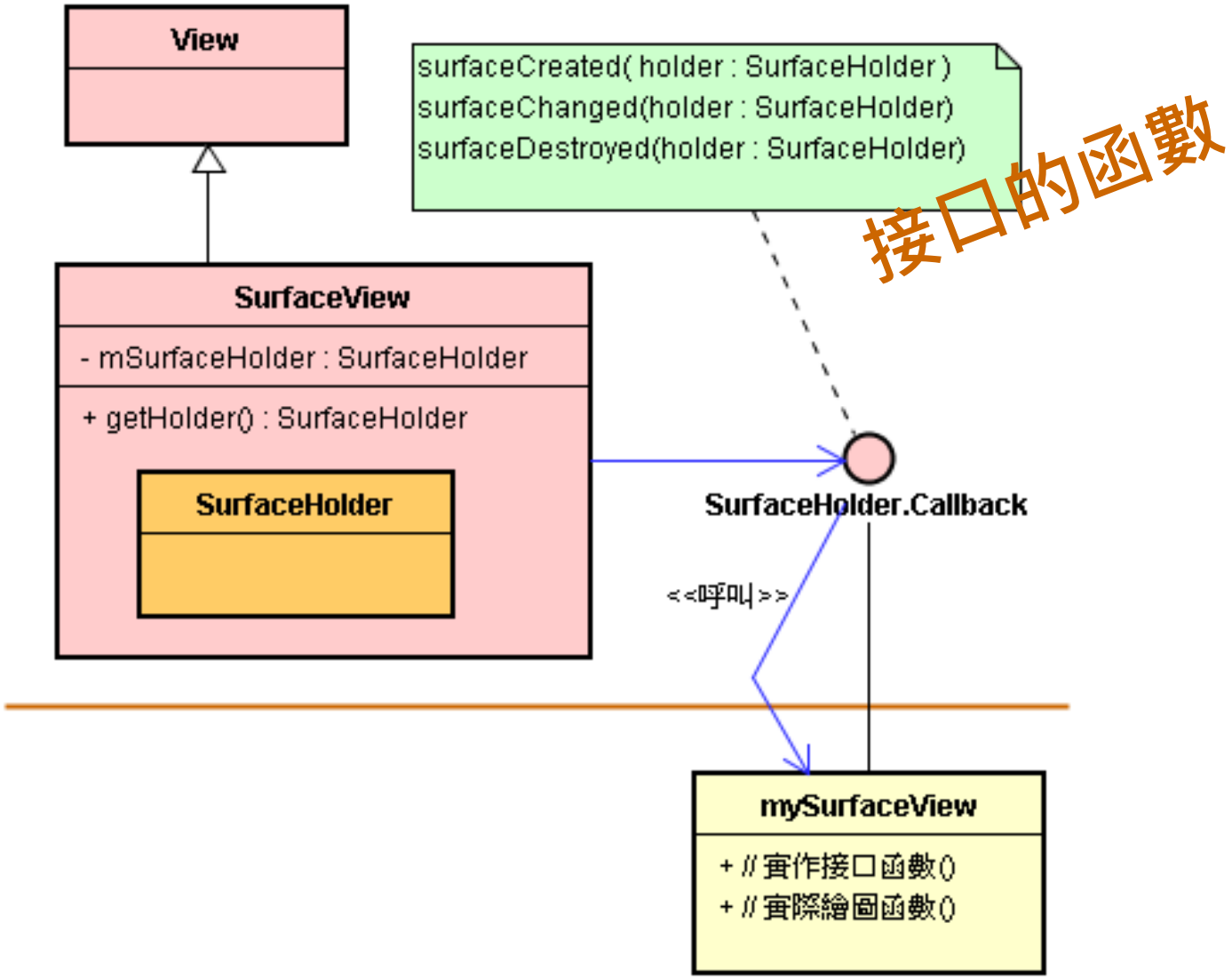
- 这些特殊性就表达在App子类(就是<T>)里；于是，<T>就来创建一个小线程去执行OpenGL引擎的绘图任务。


View
<<3D繪圖>>
OpenGL
SurfaceView
(主線程)
SurfaceHolder.Callback
(主線程)
myRenderer
(子線程)
創建小線程

通用性接口范例：SurfaceHolder.Callback

- 在绘图时，引擎需要在画布宣染，就由框架基类来提供画布。
- 于是，Callback接口让<E>反向调用<T>，在调用时就将画布传递给<T>，让业主选择的绘图引擎(如OpenGL ES)能在画布宣染了。

- 在绘图时，引擎需要在画布宣染，就由框架基类来提供画布。
- 于是，Callback接口让<E>反向调用<T>，在调用时就将画布传递给<T>，让业主选择的绘图引擎(如OpenGL ES)能在画布宣染了。


View
surfaceCreated(holder : SurfaceHolder)
surfaceChanged(holder : SurfaceHolder)
surfaceDestroyed(holder : SurfaceHolder)
接口的函數
SurfaceView
- mSurfaceHolder : SurfaceHolder
+ getHolder() : SurfaceHolder
SurfaceHolder
SurfaceHolder.Callback
<<呼叫>>
mySurfaceView
+ // 實作接口函數()
+ // 實際繪圖函數()

- 画布就藏在SurfaceHolder对象里，在画布诞生、改变和删除时，基类会调用接口，把SurfaceHolder对象(也就等于传画布)传给<T>。

范例代码：

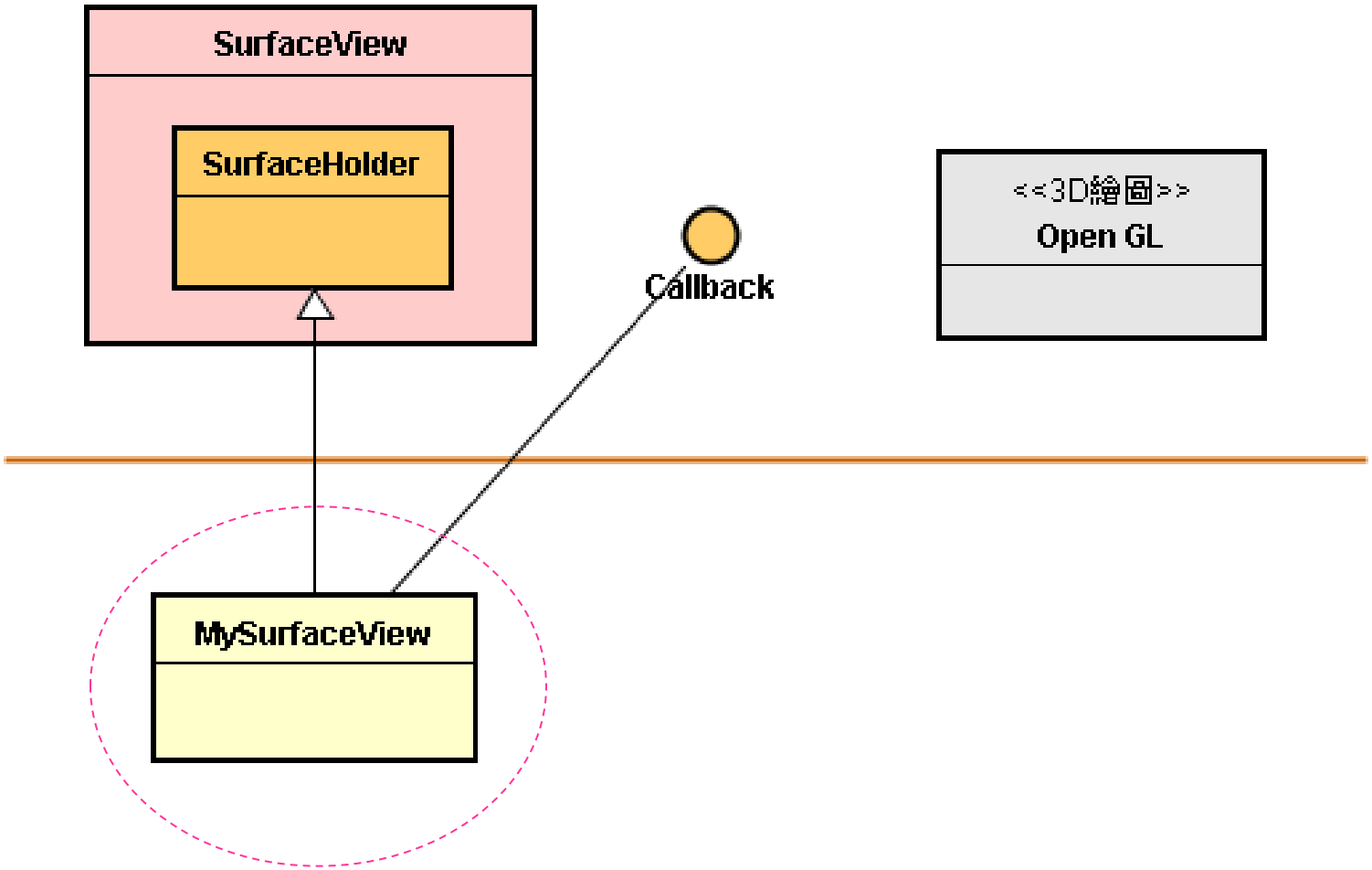
Step-1: 建立一个Android开发项目。

Step-2: 撰写Activity的子类：ac01。

```
//  ac01.java
// ........
public class ac01 extends Activity {
        private MySurfaceView sv;

        @Override protected void onCreate(Bundle icicle) {
              super.onCreate(icicle);
              setContentView(R.layout.main);
              sv = new MySurfaceView(this);
              setContentView(sv);
          }
}
```

- 这ac01对象里诞生一个MySurfaceView对象，由它来画出3D的旋转力方体。
- Step-3: 撰写 MySurfaceView类代码：

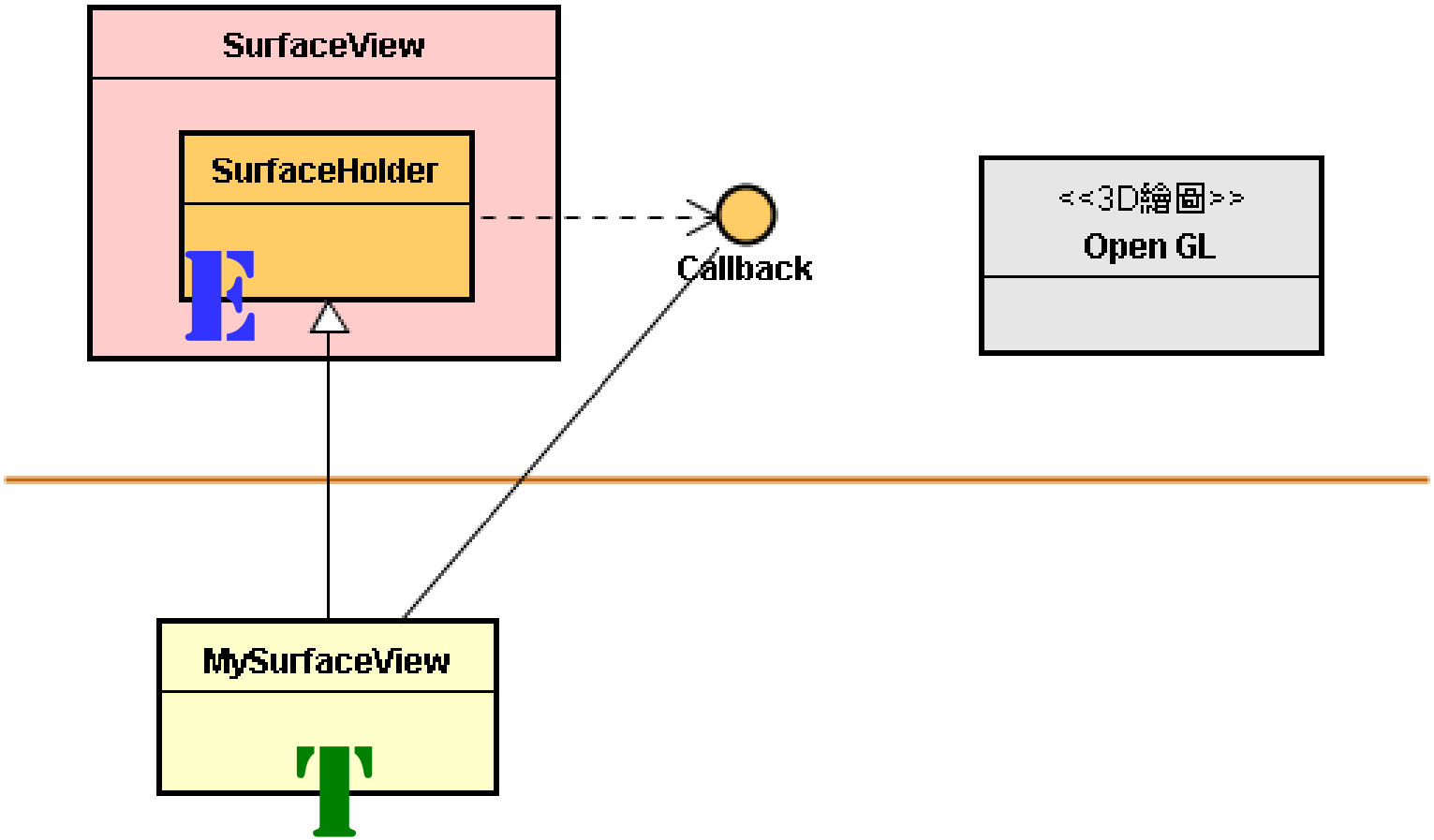
```
//  MySurfaceView.java
// ……..
class MySurfaceView extends SurfaceView
                        implements SurfaceHolder.Callback {
    private SurfaceHolder mHolder;
    private GLThread mGLThread;
    private MyCube mCube;
    private float mAngle;

    MySurfaceView(Context context)
        {   super(context);   init();   }
    public MySurfaceView(Context context, AttributeSet attrs) {
                super(context, attrs);   init();
    }
```


SurfaceView
SurfaceHolder
Callback
<<3D繪圖>>
Open GL
MySurfaceView

```
    private void init() {
        mHolder = getHolder();
        mHolder.addCallback(this);
        mHolder.setType(
                    SurfaceHolder.SURFACE_TYPE_GPU);
     }

public void surfaceCreated(SurfaceHolder holder) {
            mGLThread = new GLThread();
            mGLThread.start();    }
   public void surfaceDestroyed(SurfaceHolder holder) {
            mGLThread.requestExitAndWait();
            mGLThread = null;    }
   public void surfaceChanged(SurfaceHolder holder, int format,
                                        int w, int h) {
            mGLThread.onWindowResize(w, h);       }
```

将<T>装配到<E>里

```
mHolder = getHolder();
mHolder.addCallback(this);
```


SurfaceView
SurfaceHolder
Callback
<<3D繪圖>>
Open GL
MySurfaceView

```
// 诞生新线程来执行绘图任务
    class GLThread extends Thread {
        private boolean mDone;
        private boolean mSizeChanged = true;
        private int mWidth, mHeight;

        GLThread() {
                    super();
                    mDone = false;   mWidth = 0;   mHeight = 0;
                    // 决定欲绘出的(立方体)的面
                    byte indices[] = {
                            6, 0, 1,   5, 1, 0,
                            1, 5, 6,   0, 6, 5,
                            2, 3, 7,   6, 2, 7,
                        };
                    mCube = new MyCube(indices);
                }
```

```
@Override public void run() {
        EGL10 egl = (EGL10)EGLContext.getEGL();
        EGLDisplay dpy =
                    gl.eglGetDisplay(EGL10.EGL_DEFAULT_DISPLAY);
         // dpy 代表显示设备(screen)
         int[] version = new int[2];
         egl.eglInitialize(dpy, version);
         int[] configSpec = {
                                EGL10.EGL_RED_SIZE,      8,
                                EGL10.EGL_GREEN_SIZE,    8,
                                EGL10.EGL_BLUE_SIZE,     8,
                                EGL10.EGL_DEPTH_SIZE,   16,
                                EGL10.EGL_NONE
                    };
          EGLConfig[] configs = new EGLConfig[1];
          int[] num_config = new int[1];
          egl.eglChooseConfig(dpy, configSpec, configs, 1, num_config);
```

```
 // configs[]存有OpenGL ES 组态值(configuration)
EGLConfig config = configs[0];
EGLContext glc = egl.eglCreateContext(dpy, config,
                EGL10.EGL_NO_CONTEXT, null);
 // glc 代表OpenGL ES 的current context。

 //下面的surface则代表绘图面
 EGLSurface surface = null;   GL10 gl = null;
 while (!mDone) {
         int w, h;
         boolean changed;
```

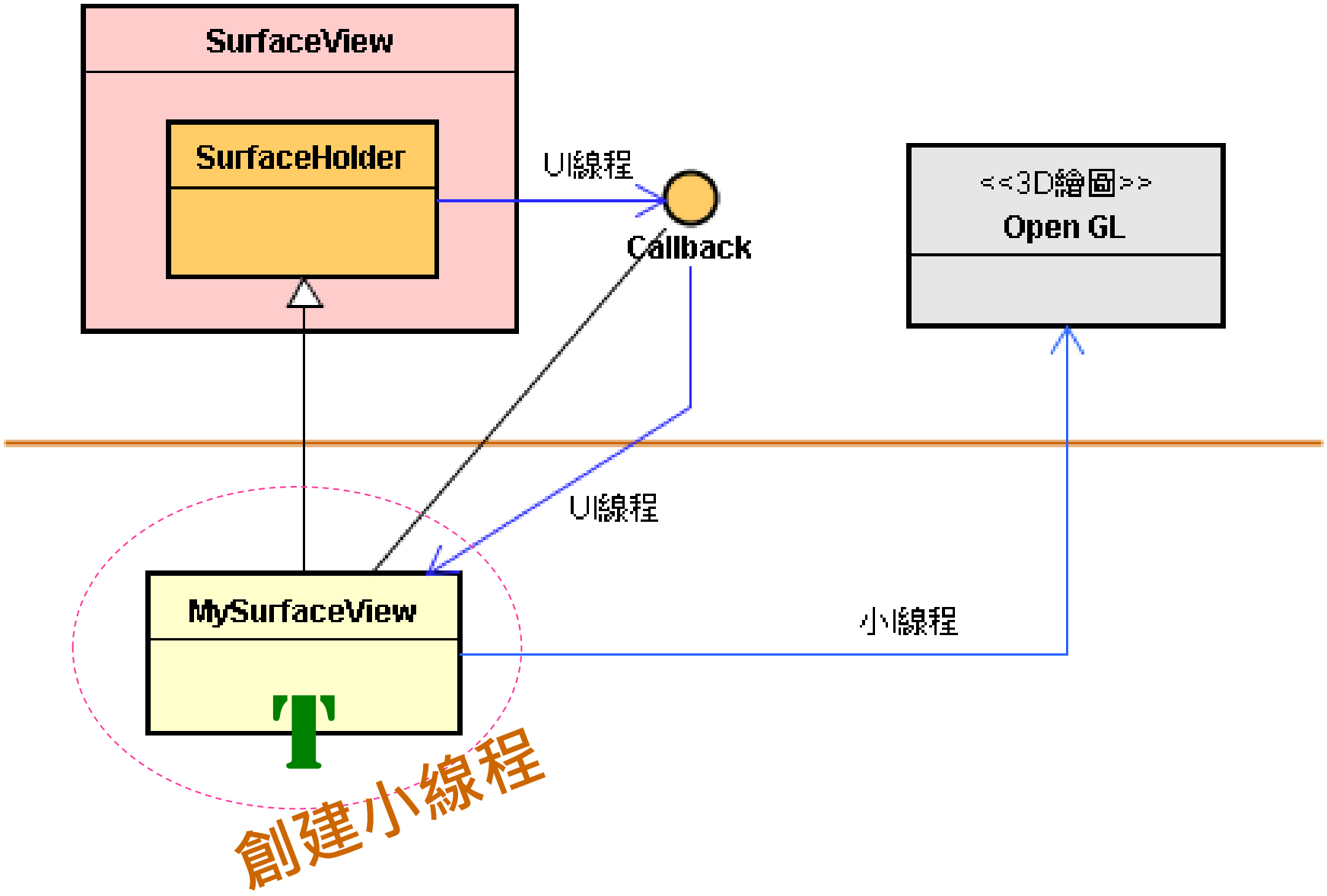
```
synchronized(this) {
            changed = mSizeChanged;
            w = mWidth;  h = mHeight;
            mSizeChanged = false;
        }
        // 当窗口大小改变了，立即重新诞生一个绘图面
        if (changed) {
          // 诞生新的绘图面
            surface = egl.eglCreateWindowSurface(dpy, config,
                        mHolder, null);
           // 将新绘图面加入到current context里
           egl.eglMakeCurrent(dpy, surface, surface, glc);
           //向current context取得它的绘图接口
           gl = (GL10)glc.getGL();
           gl.glDisable(GL10.GL_DITHER);
           //起始参数设定
          gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
                                        GL10.GL_FASTEST);
```

```
        gl.glClearColor(1,1,1,1);
        gl.glEnable(GL10.GL_CULL_FACE);
        gl.glShadeModel(GL10.GL_SMOOTH);
        gl.glEnable(GL10.GL_DEPTH_TEST);
        gl.glViewport(0, 0, w, h);

        //设定投影(projection)矩阵
        float ratio = (float)w / h;
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
    }
```

```
        //  实际绘图
       drawFrame(gl);
        //变换buffer
        egl.eglSwapBuffers(dpy, surface);
        if (egl.eglGetError() == EGL11.EGL_CONTEXT_LOST) {
                        Context c = getContext();
                        if (c instanceof Activity) {  ((Activity)c).finish();  }
           }}
        // 准备结束
        egl.eglMakeCurrent(dpy, EGL10.EGL_NO_SURFACE,
                        EGL10.EGL_NO_SURFACE,
                        EGL10.EGL_NO_CONTEXT);
        egl.eglDestroySurface(dpy, surface);
        egl.eglDestroyContext(dpy, glc);
        egl.eglTerminate(dpy);
    }
```

```
//--------------- 绘图3D对象 ---------------------
    private void drawFrame(GL10 gl) {
            gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
                    GL10.GL_DEPTH_BUFFER_BIT);
            // 设定(旋转)度
            gl.glMatrixMode(GL10.GL_MODELVIEW);
            gl.glLoadIdentity();
            gl.glTranslatef(0, 0, -3.0f);
            gl.glRotatef(mAngle, 0, 1, 0);
            gl.glRotatef(mAngle*0.25f,  1, 0, 0);
            gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
            gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
            mCube.draw(gl); // 绘出一个立方图
            mAngle += 1.2f;
     }
```

```
 public void onWindowResize(int w, int h) {
     synchronized(this) {
        mWidth = w;  mHeight = h;  mSizeChanged = true;
      }
 }
 public void requestExitAndWait() {
          // 避免deadlock
          mDone = true;
          try { join(); }
            catch (InterruptedException ex) { }
            }
}}
```


SurfaceView
SurfaceHolder
UI線程
Callback
<<3D繪圖>>
Open GL
UI線程
MySurfaceView
小線程
創建小線程

- 在此类别里定义了一个GLThread类别，执行时会产生一个新线程(Thread)来担任绘图任务。
- 在GLThread类别里，有个while循环，将不断地调整旋转角度，并呼叫MyCube对象，重新画出立方体的表面。
- **Step-4:** 撰写 MyCube类的代码如下：

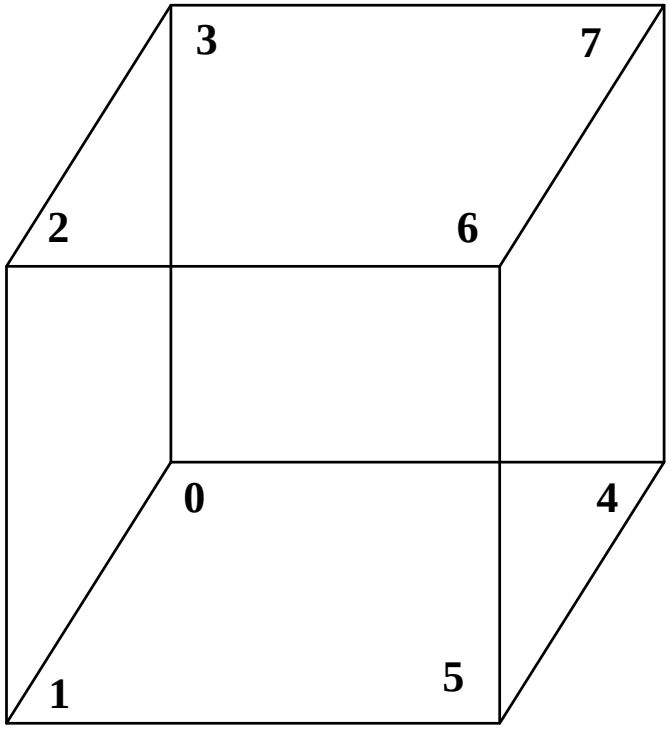
```
//  MyCube.java
// ..........

class MyCube{
    private int mTriangles;
    public MyCube(byte [] indices) {
        int one = 0x10000;
        int vertices[] = {
                -one, -one, -one,
                 one, -one, -one,
                 one,  one, -one,
                -one,  one, -one,
                -one, -one,  one,
                 one, -one,  one,
                 one,  one,  one,
                -one,  one,  one,
            };
```

```
        int colors[] = {
            0,    0,    0,  one,
            one,    0,    0,  one,
            one,    0,  one,  one,
            0,  one,    0,  one,
            0,    0,  one,  one,
            one,  one,    0,  one,
            one,  one,  one,  one,
            0,  one,  one,  one,
        };
    ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
    vbb.order(ByteOrder.nativeOrder());
    mVertexBuffer = vbb.asIntBuffer();
    mVertexBuffer.put(vertices);
    mVertexBuffer.position(0);
    ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length*4);
```

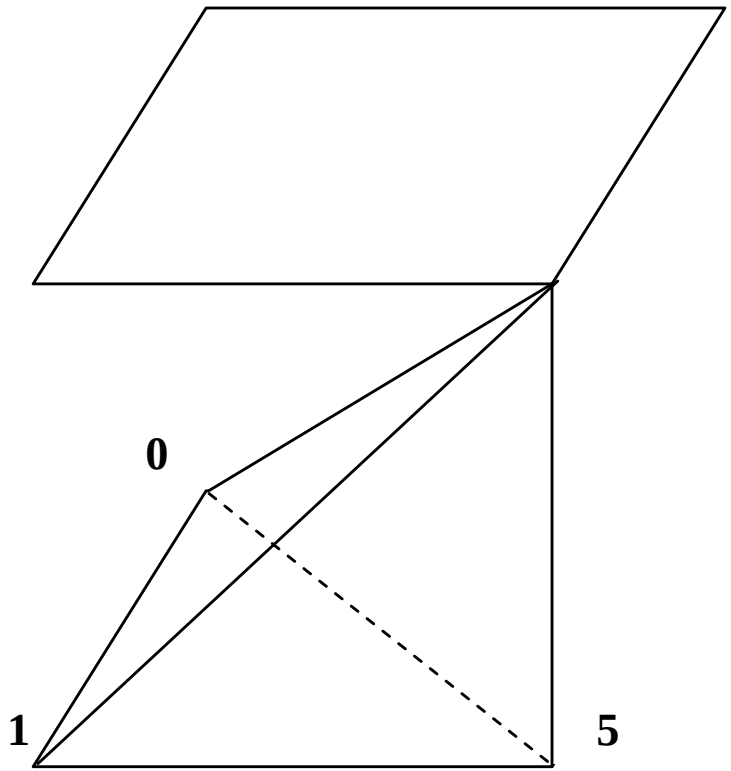
```
cbb.order(ByteOrder.nativeOrder());
        mColorBuffer = cbb.asIntBuffer();
        mColorBuffer.put(colors);           mColorBuffer.position(0);
        mIndexBuffer = ByteBuffer.allocateDirect(indices.length);
        mIndexBuffer.put(indices);          mIndexBuffer.position(0);
        mTriangles = indices.length;
    }
    public void draw(GL10 gl) {
        gl.glFrontFace(GL10.GL_CW);
        gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);
        gl.glColorPointer(4, GL10.GL_FIXED, 0, mColorBuffer);
        gl.glDrawElements(GL10.GL_TRIANGLES, mTriangles,
                GL10.GL_UNSIGNED_BYTE, mIndexBuffer);
    }
    private IntBuffer  mVertexBuffer, mColorBuffer;
    private ByteBuffer  mIndexBuffer;
}
```

说明：

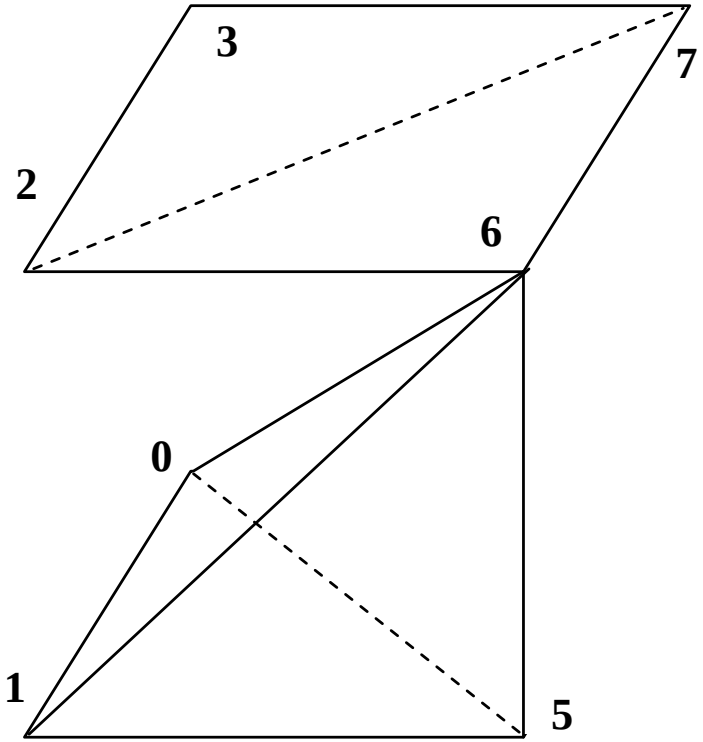
- 一个立方体，共有8个角，将其编号：



- 本范例并没有画出这个立方体的每一个表面，而是画出一个锥体，以及一个平面，如下图：



- 在OpenGL ES里，我们能以「三角决定一平面」的观念来叙述想要绘出那些表面。所以将其标上编号如下：



- 从图中，可看出共有6个「三角面」，于是在程序里可以定义如下：

```
byte indices[] = {
        6, 0, 1,   5, 1, 0,
        1, 5, 6,   0, 6, 5,
        2, 3, 7,   6, 2, 7,
    };
```

- 如此就能准确地画出上述的图形了。

结语：

- 创造未来性需要仰赖通用性接口，然而通用性接口未能顾及各方(如业主)的特殊决策，只能将这些特殊需求都表达于<T>里，增加了App开发<T>插件时的工作量。
- 所以，架构师还需要设计特殊性接口，来减轻App开发者的负担。

Thanks...

高煥堂