

Component 扩展 实现特别功能的 Native 控件。例如：RichTextview，RefreshListview 等。
Weex 的自定义组件其实相当于将一个 native 的组件类的对象封装起来。

参考文献：<http://weex.apache.org/cn/guide/extend-android.html>
<https://github.com/weex-team/article/issues/67>
<http://ju.outofmemory.cn/entry/339229>
<http://tech.dianwoda.com/2017/09/21/weexce-bian-lan-de-shi-xian/>
*<https://www.jianshu.com/p/53f69bfc50>
<https://github.com/liuzhao2007/WeexList>

定义组件的步骤：

- 1.自定义组件必须继承自 WXComponent 或者 WXContainer ；
- 2.weex SDK 可以识别 @wxcomponentprop (name = value(value 是 attr 或者 dsl style))；
- 3.方法必须是 public 的；
- 4.组件类不能是一个内部类；
- 5.自定义组件不能被 ProGuard 之类的工具混淆；
- 6.组件方法在 UI 线程被调用，因此不要在里面进行耗时的操作；
- 7.Weex 的参数类型可以是 int, double, float, String, Map, List 和实现了 WXObject 接口的自定义类；

示例如下：

```
public class RichText extends WXComponent<TextView> {  
    //所有组件都需要相同的此语句块  
    public RichText(WXSDKInstance instance, WXDomObject dom,  
        WXVContainer parent) {  
        super(instance, dom, parent);  
    }  
    // 通过 initComponentsHostView 创建 Component 需要承载的 view  
    @Override  
    protected TextView initComponentsHostView(@NonNull Context context) {  
        TextView textView = new TextView(context);  
        textView.setTextSize(20);  
        textView.setTextColor(Color.BLACK);  
        return textView;  
    }  
    //@WXComponent 用来定义组件属性，若组件仅用于显示则可不要  
    @WXComponentProp(name = "tel")  
    public void setTel(String telNumber) {  
        getHostView().setText("tel: " + telNumber);  
    }  
}
```

*WXComponent(组件基类)负责承载 native view，可通过泛型指定承载的 view 类型

*通过 `initComponentHostView` 创建 `Component` 需要承载的 `view` 所有的 `Component` 必须重写 `initComponentHostView` 方法，返回需要承载的 `view` 的最外层容器。

注册组件：

```
WXSDKEngine.registerComponent("richText", RichText.class);
```

*需要 `import` 比如 `import com.weex.app.extend.OpenGLRenderer;`

前面的 `com.*****` 其实是 `package` 的名称在每个文件头都应该有，自己新建的 `.java` 文件应当添加比如 `package com.weex.app.extend;`

Js 调用：

```
<template>
  <div>
    <richText tel="12305" style="width:200;height:100">12305</richText>
  </div>
</template>
```

*直接在 `template` 中调用即可

定义 `GLSurfaceView` 组件

建议：应当先在 `Android studio` 中尝试把组件在 `native` 中开发运行出来，在将其代码封装成 `GLSurface` 组件，可以极大减少出错的概率。

1. 首先参考 `native` 代码

MainActivity.java(界面文件，仅仅是引用组件和给其传参)

```
package com.example.vampire.myapplication;

import android.opengl.GLSurfaceView;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;

/**
 * 构造 OpenGL ES View
 * 类似 平常创建项目显示的 Hello world 一样
 *
 * 编译运行后 屏幕是一个绿色界面
 */
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

```

        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
        GLSurfaceView glSurfaceView = new GLSurfaceView(this);//以下三句为实例化
GLSurfaceView 和 OpenGLRenderer, 并返回
        glSurfaceView.setRenderer(new OpenGLRenderer());
        setContentView(glSurfaceView);
    }
}

```

OpenGLRenderer.java（组件的样式 动作在此处）

此处构造一个自定义的 **OpenGLRenderer** 通过 **GLSurfaceView.Renderer** 注册到 **GLSurfaceView**

这是渲染器(Render)的公共接口，它的任务就是调用 OpenGL 的 API 来作帧的渲染。
GLSurfaceView 的实现类通常会创建一个 **Render** 的实现类，然后用 **setRenderer(GLSurfaceView.Renderer)**方法把渲染器注册到 **GLSurfaceView**。

***必须处在与 MainActivity.java 同一个文件夹且同一个包中(因为是直接在 MainActivity 代码中引用的没有 import)**

```

package com.example.vampire.myapplication;

/**
 * Created by Vampire on 2018/4/18.
 */

import android.opengl.GLSurfaceView;
import android.opengl.GLU;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

/**
 * Created by Administrator on 2016/9/26.
 * <p>
 * 定义一个统一图形绘制的接口
 */

public class OpenGLRenderer implements GLSurfaceView.Renderer {
    /**
     * 主要用来设置一些绘制时不常变化的参数，例如：背景色，是否打开 Z-buffer(去除隐藏面)等
     *
     * @param gl
     * @param config
     */
}

```

```

    */
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    //设置背景的颜色
    gl.glClearColor(0f, 1f, 0f, 0.5f);
    //使光滑的材质,默认不需要。
    gl.glShadeModel(GL10. GL_SMOOTH);
    //深度缓冲设置。
    gl.glClearDepthf(1.0f);
    //启用深度测试。
    gl.glEnable(GL10. GL_DEPTH_TEST);
    //深度测试类型
    gl.glDepthFunc(GL10. GL_LEQUAL);
    //最好的的角度计算。
    gl.glHint(GL10. GL_PERSPECTIVE_CORRECTION_HINT, GL10. GL_NICEST);
}

/**
 * 如果设备支持屏幕的横向和纵向切换,
 * 这个方法将发生在横向<=>纵向互换时,
 * 此时可以重新设置绘制的纵横比率。
 *
 * @param gl
 * @param width
 * @param height
 */
@Override
public void onSurfaceChanged(GL10 gl, int width, int height) {

    //将当前视图端口设置为新的大小。
    gl.glViewport(0, 0, width, height);
    //选择投影矩阵
    gl.glMatrixMode(GL10. GL_PROJECTION);
    //重置投影矩阵
    gl.glLoadIdentity();

    /**
     * 建立一个透视投影矩阵
     *
     * gl : GL10 接口
     * fovy : 指定领域的视角,在 Y 轴方向。指定方面定量 determin 领域在 x 方向上的
看法。
     *
     * 高宽比的比例是 x(宽度)y(高度)。
     * zNear : 指定观众的距离不远的剪裁平面的(总是正数)。
     * zFar : 指定了与观众的距离遥远的剪裁平面的(总是正数)。

```

```

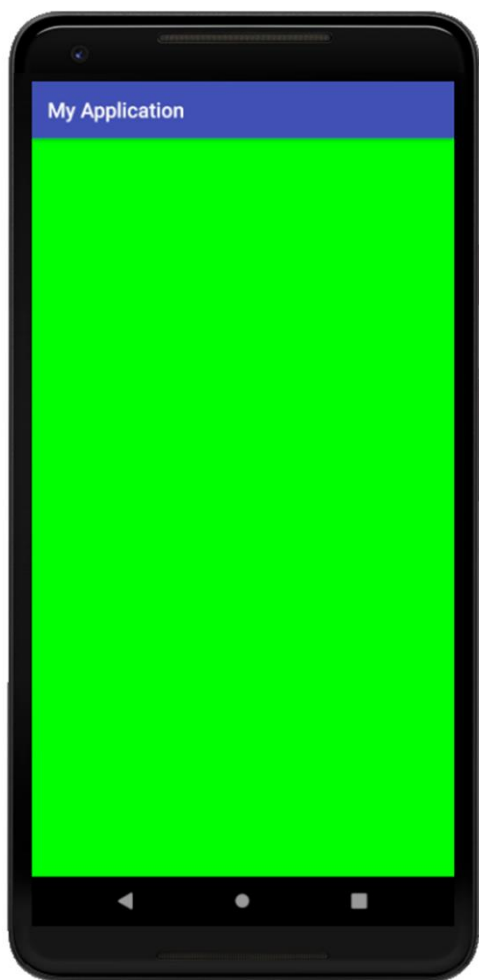
        */
        //计算窗口的长宽比
        GLU.gluPerspective(gl, 45.0f, (float) width / (float) height, 0.1f,
100.0f);
        //选择 modelview 矩阵
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        //重置投影矩阵
        gl.glLoadIdentity();

    }

    /**
     * 定义实际的绘图操作
     *
     * @param gl
     */
    @Override
    public void onDrawFrame(GL10 gl) {
        //清除屏幕和深度缓冲。
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    }
}








```

如此可显示出画布



2. 将 native 组件的对象封装成 weex 组件

在项目 `platforms\android\app\src\main\java\com\weex\app\extend` 中新建两个 java 文件其中一个为 native 组件实例化的 java 文件 直接放 native 源代码（当然包名要改 而且要 import 到 `WXApplication.java` 中），另一个为自定义组件封装 native 的 Java 文件 即手册上示例的代码

gls > platforms > android > app > src > main > java > com > weex > app > extend				
名称	修改日期	类型	大小	
 BlurTool.java	2018/4/18 12:38	Java 源文件	10 KB	
 BlurTransformation.java	2018/4/18 12:38	Java 源文件	2 KB	
 glsurface.java	2018/4/20 16:56	Java 源文件	2 KB	
 ImageAdapter.java	2018/4/18 12:38	Java 源文件	4 KB	
 OpenGLRenderer.java	2018/4/23 14:55	Java 源文件	4 KB	
 Triangle.java	2018/4/23 14:26	Java 源文件	2 KB	
 WXEventModule.java	2018/4/18 12:38	Java 源文件	1 KB	

OpenGLRenderer.java

同 native 相同，略。

自定义组件 glsurface 类 glsurface.java

```
package com.weex.app.extend;

import android.content.Context;
import android.support.annotation.NonNull;

import android.opengl.GLSurfaceView;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;

import com.taobao.weex.WXSDKInstance;
import com.taobao.weex.dom.WXDomObject;
import com.taobao.weex.ui.component.WXComponent;
import com.taobao.weex.ui.component.WXComponentProp;
import com.taobao.weex.ui.component.WXVContainer;

public class glsurface extends WXComponent<GLSurfaceView> {

    public glsurface(WXSDKInstance instance, WXDomObject dom, WXVContainer parent) {
        super(instance, dom, parent);
    }

    @Override
    protected GLSurfaceView initComponentsHostView(@NonNull Context context) {
        GLSurfaceView view = new GLSurfaceView(context);
        view.setRenderer(new OpenGLRenderer());
        return view;
    }

    // @WXComponentProp(name = "tel")
    // public void setTelLink(String tel){
    //     SpannableString spannable=new SpannableString(tel);
    //     spannable.setSpan(new URLSpan("tel:"+tel),0,tel.length(),
    Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
    //     ((GLSurfaceView) getHostView()).setText(spannable);
    // }
}
```

上面代码解析：由于只要将组件显示出来，故只需通过 `initComponentHostView` 创建 `Component` 需要承载的 `view`，而无需使用 `@WXComponentProp` 来设置属性（若需要传参或其他操作则需要）

3. 在 `WXApplication` 中注册组件（*一定要 `import`），并在 `vue` 文件中加入组件。

由于是封装了 `native` 原生组件，故只能在 `Android` 平台运行，使用 `npm start` 运行 `web` 版本一般而言无法显示出来。

在 `WXApplication` 中加入

```
import com.weex.app.extend.glsurface;
import com.weex.app.extend.OpenGLRenderer;

WXSDKEngine.registerComponent("glsurface", glsurface.class);
```

在要显示页面的 `vue` 文件中添加

```
<template>
  <div>
    <glsurface class="ggl"></glsurface>
  </div>
</template>

<style>
.ggl {
  width:800px;
  height:1600px;
}
</style>
```