<http://blog.csdn.net/jason_520/article/details/54969906>

<http://blog.csdn.net/ys5773477/article/details/53615142>

<http://www.cgjoy.com/forum.php?mod=viewthread&tid=178822>

<http://blog.csdn.net/sgnyyy/article/details/44560023>

<http://blog.csdn.net/ecidevilin/article/details/52461525>

在unity中，Renderer组件有两个属性：material和sharedMaterial，它们都可以用来获取Renderer的材质属性，但它们之间又有很大的区别：

从效率上来说，最好用sharedMaterial，它是共享材质，无论如何操作材质的属性（如更换颜色或者更换shader），内存中只会占用一份。但是如果要是用material的话，每次更换属性的时候unity就会自动new一份新的material作用于它。它直到Application.LoadLevel()或者Resources.UnloadUnusedAssets()的时候才会释放内存，所以material就有可能造成内存泄漏

经过测试：

cube.GetComponent<Renderer>().material.color = Color.blue;

cube.GetComponent<Renderer>().sharedMaterial.color = Color.red;

Debug.Log("1 material.color = " + cube.GetComponent<Renderer>().material.color);

Debug.Log("2 sharedMaterial.color = " + cube.GetComponent<Renderer>().sharedMaterial.color);



猜测(伪代码)：

Get {

if (material == null) return 创建一个material，owner是自己

if (material.owner != 自己) return 创建一个material，owner是自己

return material

}

Set {

Material = 新的材质

}

初始化material为null，sharedMaterial始终为materials[0]，c++底层根本就没有sharedMaterial这个变量

也就是说只要set了material，然后get肯定会重新创建一个，因为新的材质的owner肯定不是自己

在初始化get的时候，material==null，也会创建一个material，访问sharedMaterial此时返回第一个，也就是刚才新创建的material

如果我们默认的material数量为0，那么sharedMaterial也是null









它会返回第一个material，如果是多个的话会返回编辑器里的第一个，也就是索引号为0的材质，因为在编辑器材质可以变换位置，因此返回的material也会变化（已经测试）

重点来了，在实际使用的时候，我们可以在开始的时候使用material产生一个新的材质球作用该renderer，然后之后的操作都使用sharedMaterial，这样效率更高，而且不会影响其他使用同一个材质球的renderer

例子：



两个物体Cube0与Cube1，创建一个材质M0，选择Shader为Unilit/Color，设置为红色，将材质M0赋值给两个物体，然后创建一个组件TestMaterial.cs，并将该组件赋值给Cube0

**（1）测试material：**

Void Start()

{

thisRenderer = GetComponent<Renderer>();

thisRenderer.material.color = Color.red;

}



**（2）测试sharedMaterial：**

void Start()

{

thisRenderer = GetComponent<Renderer>();

thisRenderer.sharedMaterial.color = Color.red;

}



在上面的例子中，如果是material则只会影响当前物体，如果是sharedMaterial则会影响所有使用该材质

**注意：sharedMaterial是会改变本地材质的**

**比如默认材质颜色为白色，如果我们调用了thisRenderer.sharedMaterial.color = Color.red;就相当于将该材质颜色变为红色了，以后再使用的时候都会是红色**

如果material与sharedMaterial同时作用：

void Start()

{

thisRenderer = GetComponent<Renderer>();

thisRenderer.sharedMaterial.color = Color.red;

thisRenderer.material.color = Color.blue;

}



void Start()

{

thisRenderer = GetComponent<Renderer>();

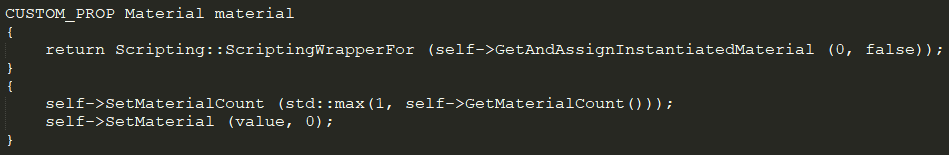
thisRenderer.material.color = Color.blue;

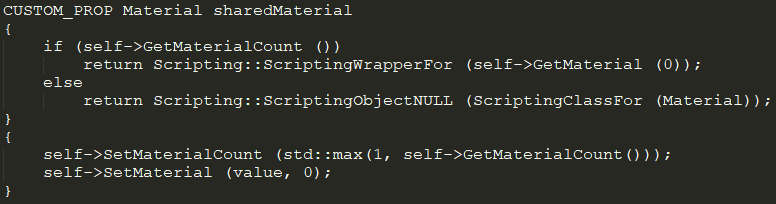
thisRenderer.sharedMaterial.color = Color.red;

}

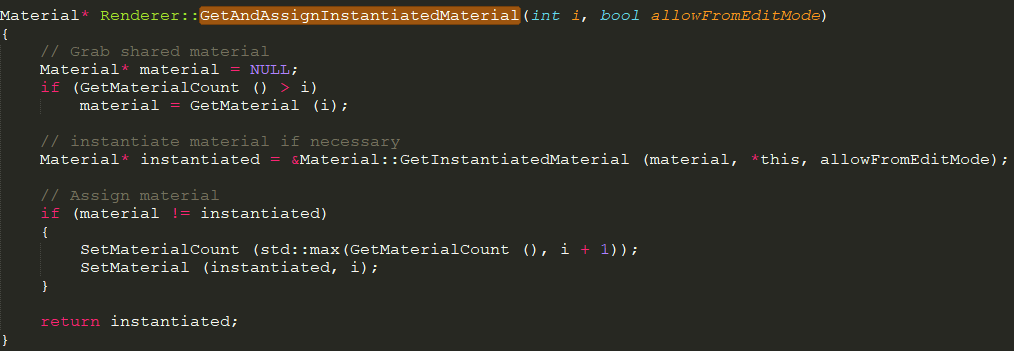


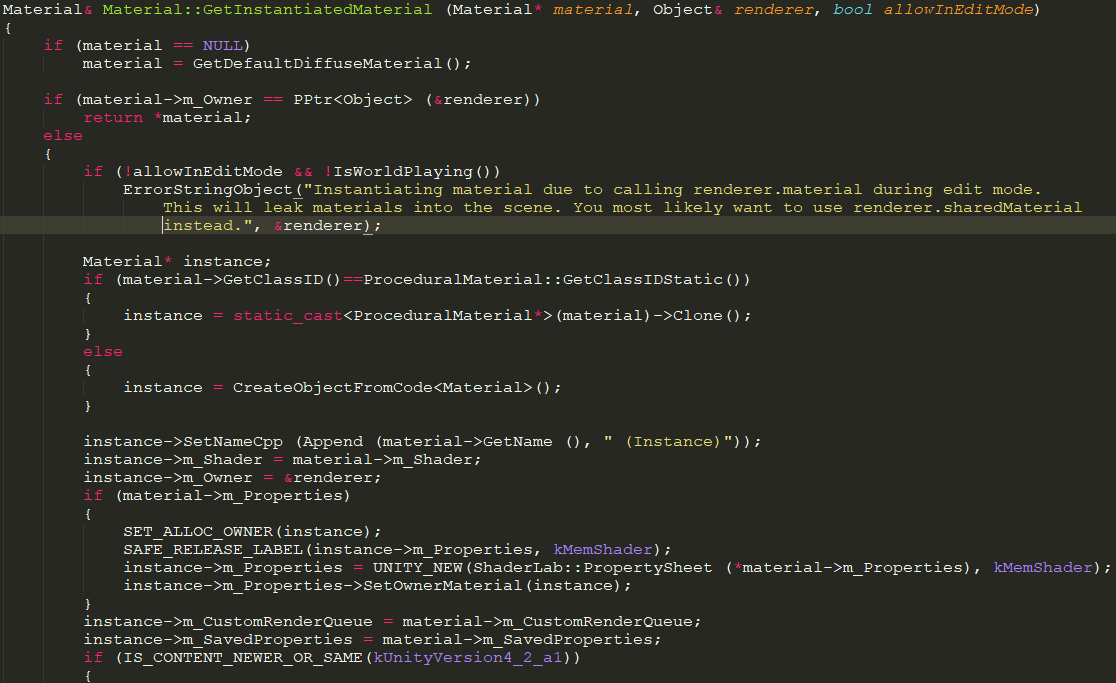
源码：

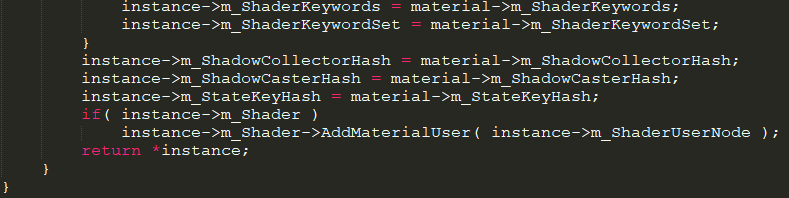




在Renderer中：







针对<https://blog.csdn.net/jason_520/article/details/54969906>的现象

首先两个物体A与B都添加了同一个材质

第一种情况（先按F1，在按F2，最后按F3）

先按F1，A物体的材质有一个，但是发现当前才是并不是属于自己的，然后创建了一个材质，放入第一个位置中，相当于覆盖了之前的共享材质，然后按下F2与F3，虽然都是操作的sharedMaterial，但是sharedMaterial都是指向的自己新创建的材质

先按F2，A物体的材质有一个，此时就直接修改了当前的这个材质，但是这个材质实际上B物体也使用了，因此B物体也被修改了，然后按F1，此时A物体仍然有一个之前被修改的材质，但是法线这个不是属于自己的，然后又创建了一个新的材质覆盖了之前已经被修改的共享材质，然后按F3，此时操作的是自己创建那个材质，它的材质列表中已经没有共享材质了

实际上也就说明了：

如果材质为空，那么共享材质就是空

如果有属于自己的材质，那么共享材质就是自己材质的第一个

如果没有属于自己的材质，但是材质列表不为空，那么共享材质就是返回的当前不属于自己材质的第一个