**编写着色器**

Unity 中的着色器可以用三种不同的方式编写：

**表面着色器**

如果着色器需要受到光照和阴影的影响，则[表面着色器](file:///E:\\UnityDocumentation\\Manual\\SL-SurfaceShaders.html)是您的最佳选择。表面着色器以简洁的方式使编写复杂的着色器变得容易 - 这是与 Unity 光照管线进行交互的更高级别抽象。大多数表面着色器自动支持前向和延迟光照。您可以用几行 **Cg/HLSL** 编写表面着色器，随后就能在此基础上自动生成更多代码。

如果您的着色器与光照没有任何关联，请不要使用表面着色器。对于[后期处理效果](file:///E:\UnityDocumentation\Manual\PostProcessingOverview.html)或许多特殊效果着色器，表面着色器是次优选项，因为它们会无缘无故进行大量光照计算。

**顶点和片元着色器**

如果您的着色器不需要与光照交互，或者如果您需要无法由表面着色器处理的非常奇特的效果，则需要使用[顶点和片元着色器](file:///E:\\UnityDocumentation\\Manual\\SL-ShaderPrograms.html)。以这种方式编写的着色器程序是创建所需效果的最灵活方法（甚至表面着色器也会自动转换为一堆顶点和片元着色器），但这需要付出代价：必须编写更多的代码并且更难以让它与光照互动。这些着色器也是以 **Cg/HLSL** 编写。

**固定函数着色器**

固定函数着色器是传统的着色器语法，用于非常简单的效果。建议编写可编程的着色器，因为这样可以提供更大的灵活性。固定函数着色器完全用一种名为 **ShaderLab** 的语言编写，此语言类似于 Microsoft 的 .FX 文件或 NVIDIA 的 CgFX。在内部，所有固定函数着色器在导入着色器时转换为[顶点和片元着色器](file:///E:\\UnityDocumentation\\Manual\\SL-ShaderPrograms.html)。

**ShaderLab**

无论您选择哪种类型的着色器，实际的着色器代码始终包含在用于组织着色器结构的 ShaderLab 中。如下所示：

Shader "MyShader" {

Properties {

\_MyTexture ("My Texture", 2D) = "white" { }

// 此处还放置颜色或矢量等其他属性

}

SubShader {

// 此处放置

// - 表面着色器或

// - 顶点和片元着色器或

// - 固定函数着色器

}

SubShader {

// 此处放置上述 SubShader 更简单的 "fallback" 版本

// 此版本能够在较旧的显卡上运行

}

}

我们建议您首先阅读 [ShaderLab 参考](file:///E:\UnityDocumentation\Manual\SL-Shader.html)中的 ShaderLab 语法的一些基本概念，然后继续学习下面列出的教程。

这些教程包含了大量不同类型着色器的示例。Unity 的[后期处理效果](file:///E:\\UnityDocumentation\\Manual\\PostProcessingOverview.html)允许您使用着色器创建许多有趣的效果。

请继续阅读着色器的介绍，并查看[着色器参考](file:///E:\\UnityDocumentation\\Manual\\SL-Reference.html)！

* [教程：ShaderLab 和固定函数着色器](file:///E:\UnityDocumentation\Manual\ShaderTut1.html)
* [教程：顶点和片元着色器](file:///E:\UnityDocumentation\Manual\ShaderTut2.html)
* [表面着色器](file:///E:\UnityDocumentation\Manual\SL-SurfaceShaders.html)
* [编写顶点和片元着色器](file:///E:\UnityDocumentation\Manual\SL-ShaderPrograms.html)

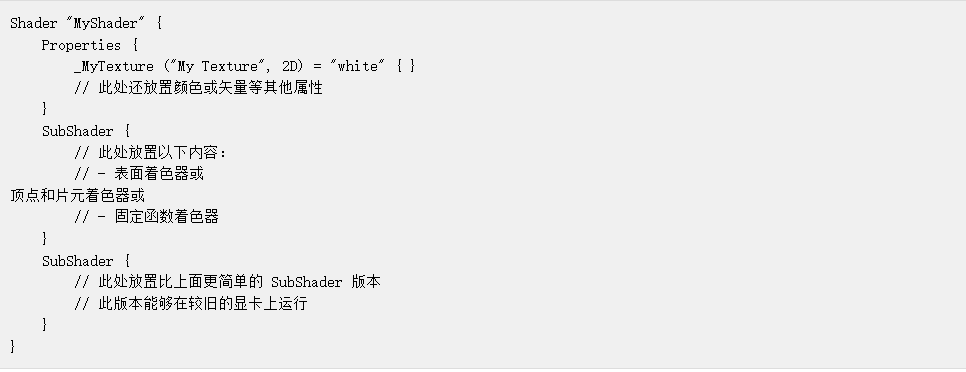
**着色器参考**

Unity 中的着色器可以用三种不同的方式编写：

* 作为[**表面着色器**](file:///E:\UnityDocumentation\Manual\SL-SurfaceShaders.html)，
* 作为**[顶点和片元着色器](file:///E:\\UnityDocumentation\\Manual\\SL-ShaderPrograms.html)**，或
* 作为**固定函数着色器**。

[着色器教程](file:///E:\UnityDocumentation\Manual\Shaders.html)可以指导您选择适合您需求的类型。

无论您选择哪种类型的着色器，着色器代码的实际内容始终以名为 ShaderLab 的语言进行包装（这种语言用于组织着色器结构）。如下所示：

我们建议您首先阅读以下各部分了解关于 [ShaderLab 语法](file:///E:\UnityDocumentation\Manual\SL-Shader.html)的一些基本概念，然后再继续阅读其他各部分了解表面着色器或顶点和片元着色器。由于固定函数着色器仅使用 ShaderLab 编写而成，因此您可以在 ShaderLab 参考本身中找到有关这些着色器的更多信息。

以下参考资料包含了不同类型着色器的大量示例。如果还需要特定表面着色器的更多示例，可从[资源部分 (Resources)](http://www.unity3d.com/support/resources/assets/built-in-shaders) 获取 Unity 内置着色器的源代码。Unity 的[后期处理效果](file:///E:\\UnityDocumentation\\Manual\\PostProcessingOverview.html)允许您使用着色器创建许多有趣的效果。

请继续阅读着色器参考资料并查看[着色器教程](file:///E:\\UnityDocumentation\\Manual\\Shaders.html)！

# ShaderLab 语法

Unity 中的所有[着色器](file:///E:\\UnityDocumentation\\Manual\\class-Shader.html)文件都是用名为“ShaderLab”的声明性语言编写的。在文件中，通过嵌套括号 语法声明着色器的各个描述，例如，材质检视面板中应显示的着色器属性； 要执行哪些类型的硬件回退；要使用哪些类型的混合模式等；而且实际“着色器代码” 编写在同一着色器文件中的 CGPROGRAM 代码片段中（请参阅[表面着色器](file:///E:\\UnityDocumentation\\Manual\\SL-SurfaceShaders.html)和[顶点和片元着色器](file:///E:\UnityDocumentation\Manual\SL-ShaderPrograms.html)）。

本页以及子页面将介绍嵌套括号“ShaderLab”语法。CGPROGRAM 代码片段是用常规 HLSL/Cg 着色语言编写的，请参阅[相关文档页面](file:///E:\\UnityDocumentation\\Manual\\SL-ShaderPrograms.html)。

**Shader** 是着色器文件的根命令。每个文件必须定义一个（且仅一个）Shader。Shader 指定当对象的材质使用此着色器时如何渲染此类对象。

## 语法

Shader "name" { [Properties] Subshaders [Fallback] [CustomEditor] }

定义着色器。此着色器将以指定的**名称**显示在材质检视面板中。着色器可以选择性地定义在材质检视面板中显示的**属性**列表。在此之后是子着色器的列表，以及可选的回退和/或自定义编辑器声明。

## 详细信息

### 属性

着色器可以具有[属性](file:///E:\UnityDocumentation\Manual\SL-Properties.html)列表。着色器中声明的所有属性都会显示在 Unity 中的[材质检视面板](file:///E:\\UnityDocumentation\\Manual\\class-Material.html)中。典型的属性包括对象颜色、纹理或者着色器要使用的任意值。

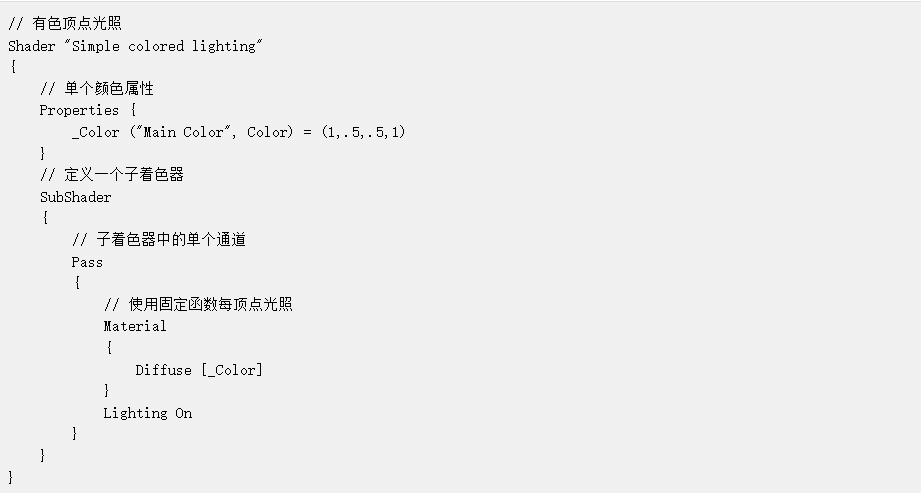
### 子着色器和回退

每个着色器均包含一组[子着色器](file:///E:\\UnityDocumentation\\Manual\\SL-SubShader.html)。必须至少包含一个子着色器。加载着色器时，Unity 将检查子着色器的列表，并选取最终用户的机器支持的第一个子着色器。如果不支持任何子着色器，Unity 将尝试使用[回退着色器](file:///E:\\UnityDocumentation\\Manual\\SL-Fallback.html)。

不同显卡具有不同功能。这给游戏开发者带来了一个永恒的问题；您希望游戏在最新款硬件上呈现出色的画面，但又不希望这些出色画面只能在数量仅占 3% 的最新款硬件上呈现。这时候就需要使用子着色器。可创建一个子着色器，使其包含您所能想象的所有奇幻图形特效，然后为型号更老的显卡添加更多子着色器。这些子着色器可能会以更慢的速度来实现您想要的效果，或者可能选择不实现某些细节。

着色器的“细节级别”(LOD) 和“着色器替换”是两种同样基于子着色器的技术，请参阅[着色器 LOD](file:///E:\\UnityDocumentation\\Manual\\SL-ShaderLOD.html) 和[着色器替换](file:///E:\\UnityDocumentation\\Manual\\SL-ShaderReplacement.html)以了解想象信息。

## 示例

下面是最简单的着色器之一：

此着色器定义了颜色属性 **\_Color\_\_（在材质检视面板中显示为 \_Main Color\_），默认值为** (1,0.5,0.5,1)\_\_。然后，定义了单个子着色器。该子着色器包含一个 [Pass](file:///E:\UnityDocumentation\Manual\SL-Pass.html)，用于启用固定函数顶点光照并为其设置基本材质。

请参阅[表面着色器示例](file:///E:\UnityDocumentation\Manual\SL-SurfaceShaderExamples.html)或 [顶点和片元着色器示例](file:///E:\UnityDocumentation\Manual\SL-VertexFragmentShaderExamples.html)中提供的更复杂示例。

# ShaderLab：Properties

着色器可以定义 Unity [材质检视面板](file:///E:\UnityDocumentation\Manual\Materials.html)中由美术师设置的参数列表。[着色器文件](file:///E:\UnityDocumentation\Manual\SL-Shader.html)中的 Properties 代码块将定义这些属性。

## 语法

#### Properties

Properties { Property [Property ...]}

定义属性代码块。在大括号内，多个属性定义如下。

#### 数字和滑动条

name ("display name", Range (min, max)) = number

name ("display name", Float) = number

name ("display name", Int) = number

这些全都定义一个具有默认值的数字（标量）属性。Range 格式使该属性显示为一个滑动条，范围在 min 到 max 之间。

#### 颜色和矢量

name ("display name", Color) = (number,number,number,number)

name ("display name", Vector) = (number,number,number,number)

使用给定 RGBA 分量的默认值定义颜色属性，或使用默认值定义 4D 矢量属性。颜色属性会显示拾色器，并根据颜色空间按需进行调整（请参阅[着色器程序中的属性](file:///E:\\UnityDocumentation\\Manual\\SL-PropertiesInPrograms.html)）。矢量属性显示为四个数字字段。

#### 纹理

name ("display name", 2D) = "defaulttexture" {}

name ("display name", Cube) = "defaulttexture" {}

name ("display name", 3D) = "defaulttexture" {}

分别定义 [2D 纹理](file:///E:\UnityDocumentation\Manual\class-TextureImporter.html)、[立方体贴图](file:///E:\UnityDocumentation\Manual\class-Cubemap.html)或 [3D（体积）](file:///E:\UnityDocumentation\Manual\class-Texture3D.html)属性。

## 详细信息

着色器中的每个属性均通过 **name** 引用（在 Unity 中，着色器属性名称通常以下划线开头）。属性在材质检视面板中将显示为 **display name**。每个属性都在等号后给出默认值：

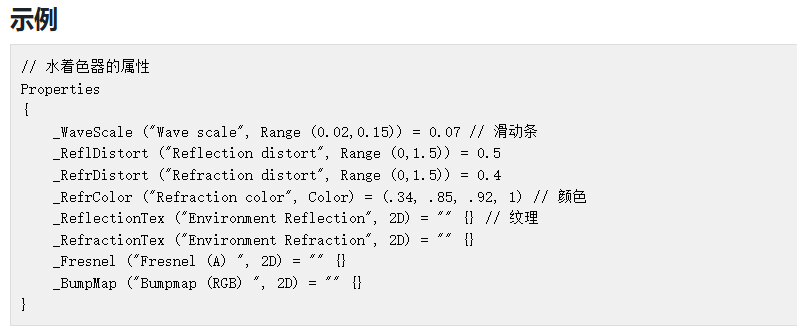
* 对于 Range 和 Float 属性，默认值仅仅是单个数字，例如“13.37”。
* 对于 Color 和 Vector 属性，默认值是括在圆括号中的四个数字，例如“(1,0.5,0.2,1)”。
* 对于 2D 纹理，默认值为空字符串或内置默认纹理之一：“white”（RGBA：1,1,1,1）、“black”（RGBA：0,0,0,0）、“gray”（RGBA：0.5,0.5,0.5,0.5）、“bump”（RGBA：0.5,0.5,1,0.5）或“red”（RGBA：1,0,0,0）。
* 对于非 2D 纹理（立方体、3D 或 2D 数组），默认值为空字符串。如果材质未指定立方体贴图/3D/数组纹理，则使用灰色（RGBA：0.5,0.5,0.5,0.5）。

稍后在着色器的固定函数部分中，可使用括在方括号中的属性名称来访问属性值：**[name]**。例如，可通过声明两个整数属性（例如“SrcBlend“和”DstBlend”）来使混合模式由材质属性驱动，然后让 [Blend 命令](file:///E:\UnityDocumentation\Manual\SL-Blend.html)使用它们：Blend [\_SrcBlend] [\_DstBlend]。

Properties 代码块中的着色器参数被序列化为[材质](file:///E:\\UnityDocumentation\\Manual\\Materials.html)数据。[着色器程序](file:///E:\UnityDocumentation\Manual\SL-ShaderPrograms.html)实际上可以有更多参数（如矩阵、矢量和浮点数），这些参数在运行时从代码中在材质上设置，但如果它们不是 Properties 代码块的一部分，则不会保存它们的值。这对于完全由脚本代码驱动的值最有用（使用 [Material.SetFloat](file:///E:\UnityDocumentation\ScriptReference\Material.SetFloat.html) 和类似函数）。

### 属性特性和绘制器

在属性前面，可指定可选的特性（用方括号括起）。这些是 Unity 可以识别的特性，或者它们可以指示您自己的 [MaterialPropertyDrawer 类](file:///E:\UnityDocumentation\ScriptReference\MaterialPropertyDrawer.html) 来控制它们在[材质检视面板](file:///E:\UnityDocumentation\Manual\class-Material.html)中的呈现方式。Unity 可以识别的特性包括：

* [HideInInspector] - 不在材质检视面板中显示属性值。
* [NoScaleOffset] - 对于具有此特性的纹理属性，材质检视面板不会显示纹理平铺/偏移字段。
* [Normal] - 表示纹理属性需要法线贴图。
* [HDR] - 表示纹理属性需要高动态范围 (HDR) 纹理。
* [Gamma] - 表示在 UI 中将浮点/矢量属性指定为 sRGB 值（就像颜色一样），并且可能需要根据使用的颜色空间进行转换。请参阅[着色器程序中的属性](file:///E:\\UnityDocumentation\\Manual\\SL-PropertiesInPrograms.html)。
* [PerRendererData] - 表示纹理属性将以 [MaterialPropertyBlock](file:///E:\UnityDocumentation\ScriptReference\MaterialPropertyBlock.html) 的形式来自每渲染器数据。材质检视面板会更改这些属性的纹理字段 UI。

# ShaderLab：SubShader

Unity 中的每个着色器都包含一个子着色器列表。当 Unity 必须显示网格时，它将找到要使用的着色器，并选择在用户的显卡上运行的第一个子着色器。

## 语法

Subshader { [Tags] [CommonState] Passdef [Passdef ...]}

将子着色器定义为可选标签、通用状态和通道定义列表。

## 详细信息

子着色器定义[渲染通道](file:///E:\UnityDocumentation\Manual\SL-Pass.html)的列表，并且可选择性地设置所有通道共同的任意状态。此外，还可以设置子着色器专用的[标签](file:///E:\UnityDocumentation\Manual\SL-SubShaderTags.html)。

当 Unity 选择要用于渲染的子着色器时，它会为每个定义的通道 (Pass) 渲染一次对象（并且数量可能由于光交互而增加）。由于对象的每次渲染成本都很高，因此应以尽可能少的通道数量定义着色器。当然，有时在某些图形硬件上，所需的效果不能在单个通道中完成；那么您别无选择，只能使用多个通道。

每个通道定义可以是[常规 Pass](file:///E:\UnityDocumentation\Manual\SL-Pass.html)、[Use Pass](file:///E:\\UnityDocumentation\\Manual\\SL-UsePass.html) 或 [Grab Pass](file:///E:\UnityDocumentation\Manual\SL-GrabPass.html)。

Pass 定义中允许的任何语句也可能出现在子 Subshader 代码块中。这将使所有通道都使用这一“共享”状态。

## 示例

// ...

SubShader {

Pass {

Lighting Off

SetTexture [\_MainTex] {}

}

}

// ...

以上子着色器定义了单个通道 (Pass)，用于关闭光照并且只显示一个名为 **\_MainTex** 的纹理网格。

# ShaderLab：Pass

Pass 代码块将使游戏对象的几何体被渲染一次。

## 语法

Pass { [Name and Tags] [RenderSetup] }

基本的 Pass 命令包含渲染状态设置命令的列表。

## 名称和标签

一个通道 (Pass) 可以定义一个[名称 (Name)](file:///E:\\UnityDocumentation\\Manual\\SL-Name.html) 和任意数量的[标签 (Tags)](file:///E:\\UnityDocumentation\\Manual\\SL-PassTags.html)。这些名称/值字符串用于将通道的意图传达给渲染引擎。

## 渲染状态设置

一个通道将设置图形硬件的各种状态，例如是否应开启 Alpha 混合或是否应使用深度测试。

命令如下所示：

### Cull

Cull Back | Front | Off

设置多边形剔除模式。

### ZTest

ZTest (Less | Greater | LEqual | GEqual | Equal | NotEqual | Always)

设置深度缓冲区测试模式。

### ZWrite

ZWrite On | Off

设置深度缓冲区写入模式。

### Offset

Offset OffsetFactor, OffsetUnits

设置 Z 缓冲区深度偏移。请参阅[剔除和深度页面](file:///E:\\UnityDocumentation\\Manual\\SL-CullAndDepth.html)的文档以了解更多详细信息。

请参阅[剔除和深度](file:///E:\\UnityDocumentation\\Manual\\SL-CullAndDepth.html)的文档以了解有关 Cull、ZTest、ZWrite 和 Offset 的更多详细信息。

### Blend

Blend sourceBlendMode destBlendMode

Blend sourceBlendMode destBlendMode, alphaSourceBlendMode alphaDestBlendMode

BlendOp colorOp

BlendOp colorOp, alphaOp

AlphaToMask On | Off

设置 Alpha 混合、Alpha 操作和 alpha-to-coverage 模式。请参阅有关[混合](file:///E:\\UnityDocumentation\\Manual\\SL-Blend.html)的文档以了解更多详细信息。

### ColorMask

ColorMask RGB | A | 0 | R、G、B、A 的任意组合

设置颜色通道写入遮罩。写入 ColorMask 0 可关闭对所有颜色通道的渲染。默认模式是 写入所有通道 (RGBA)，但是对于某些特殊效果，您可能希望不修改某些通道，或完全禁用颜色 写入。

使用多渲染目标 (MRT) 渲染时，可通过在末尾添加索引（0 到 7）来为每个渲染目标设置不同的颜色遮罩。例如，ColorMask RGB 3 将使渲染目标 #3 仅写入到 RGB 通道。

### 旧版固定函数着色器命令

一些命令用于编写旧版“固定函数样式”着色器。这是视为已弃用的功能，因为编写[表面着色器](file:///E:\\UnityDocumentation\\Manual\\SL-SurfaceShaders.html)或[着色器程序](file:///E:\UnityDocumentation\Manual\SL-ShaderPrograms.html) 可带来更大的灵活性。但是，对于非常简单的着色器，以固定函数样式编写着色器有时会更容易，因此这里提供了命令。请注意，如果不使用固定函数着色器，则会忽略以下所有命令。

#### 固定函数 Lighting 和 Material

Lighting On | Off

Material { Material Block }

SeparateSpecular On | Off

Color Color-value

ColorMaterial AmbientAndDiffuse | Emission

所有这些均控制固定函数每顶点光照：它们将其开启，设置材质颜色，开启镜面高光，提供默认颜色（如果顶点光照关闭），并控制网格顶点颜色如何影响光照。请参阅有关[材质](file:///E:\\UnityDocumentation\\Manual\\SL-Material.html)的文档以了解更多详细信息。

#### 固定函数 Fog

Fog { Fog Block }

设置固定函数 Fog 的参数。请参阅有关[雾化](file:///E:\\UnityDocumentation\\Manual\\SL-Fog.html)的文档以了解更多详细信息。

#### 固定函数 AlphaTest

AlphaTest (Less | Greater | LEqual | GEqual | Equal | NotEqual | Always) CutoffValue

开启固定函数 Alpha 测试。请参阅有关 [Alpha 测试](file:///E:\UnityDocumentation\Manual\SL-AlphaTest.html)的文档以了解更多详细信息。

#### 固定函数纹理组合器

设置渲染状态后，可使用 [SetTexture](file:///E:\UnityDocumentation\Manual\SL-SetTexture.html) 命令指定多个纹理及其组合模式：

SetTexture textureProperty { combine options }

## 详细信息

着色器通过几种方式与 Unity 的渲染管线进行交互；例如，通道可以使用 [Tags](file:///E:\UnityDocumentation\Manual\SL-PassTags.html) 命令来表明它只应该用于[延迟着色](file:///E:\\UnityDocumentation\\Manual\\RenderTech-DeferredShading.html)。某些通道也可以在同一个游戏对象上执行多次；例如，在[前向渲染](file:///E:\UnityDocumentation\Manual\RenderTech-ForwardRendering.html)中，根据影响游戏对象的 光源数量，将多次执行“ForwardAdd”通道类型。请参阅有关[渲染管线](file:///E:\UnityDocumentation\Manual\SL-RenderPipeline.html)的文档以了解更多详细信息。

## 另请参阅

有几个特殊的通道可用于重复使用通用功能或实现各种高端效果：

* [UsePass](file:///E:\UnityDocumentation\Manual\SL-UsePass.html) 包含来自另一个着色器的指定通道。
* [GrabPass](file:///E:\UnityDocumentation\Manual\SL-GrabPass.html) 将屏幕内容抓取到纹理中，以便在之后的通道中使用。