第 5 章

编辑器外观

本章将介绍影响编辑器外观的几个重要组成部分，包括 GUI 皮肤、GUI 样式、GUI 图标和 GUI 动画，本章将介绍如何将它们应用于实际的 GUI 设计中。在此之前，先了解 GUI 类中几个与编辑器外观相关的静态变量，见表 5-1。

表 5-1　GUI 类中的静态变量

变　　量	作　　用
backgroundColor	用于 GUI 渲染的所有背景元素的着色颜色
color	用于 GUI 渲染的着色颜色
contentColor	用于 GUI 渲染的所有文本的着色颜色
skin	用于 GUI 渲染的 GUISkin 皮肤

5.1　GUI 皮肤

GUISkin 是一种资产文件，选择 Assets→Create→GUI Skin 进行创建，加载该类型资产并通过 GUI.Skin 设置激活对应的皮肤，将 GUI.Skin 设置为 null 表示使用默认的皮肤，如图 5-1 所示。

在该类型资产中存储的是 GUI 各类型控件默认使用的样式和所有自定义的样式，以及相关的设置，见表 5-2。

表 5-2　GUISkin

变　　量	详　　解
box	GUI.Box 控件默认使用的样式
button	GUI.Button 控件默认使用的样式
toggle	GUI.Toggle 控件默认使用的样式
label	GUI.Label 控件默认使用的样式
textField	GUI.TextField 控件默认使用的样式
textArea	GUI.TextArea 控件默认使用的样式

续表

变　量	详　解
window	GUI 窗口控件默认使用的样式
horizontalSlider	GUI.HorizontalSlider 控件背景部分默认使用的样式
horizontalSliderThumb	GUI.HorizontalSlider 控件中用于拖动的滑块默认使用的样式
verticalSlider	GUI.VerticalSlider 控件背景部分默认使用的样式
verticalSliderThumb	GUI.VeriticalSlider 控件中用于拖动的滑块默认使用的样式
horizontalScrollbar	GUI.HorizontalScrollbar 控件背景部分默认使用的样式
horizontalScrollbarThumb	GUI.HorizontalScrollbar 控件中用于拖动的滑块默认使用的样式
horizontalScrollbarLeftButton	GUI.HorizontalScrollbar 控件中的向左按钮默认使用的样式
horizontalScrollbarRightButton	GUI.HorizontalScrollbar 控件中的向右按钮默认使用的样式
veriticalScrollbar	GUI.VerticalScrollbar 控件背景部分默认使用的样式
verticalScrollbarThumb	GUI.VerticalScrollbar 控件中用于拖动的滑块默认使用的样式
verticalScrollbarUpButton	GUI.VerticalScrollbar 控件中的向上按钮默认使用的样式
verticalScrollbarDownButton	GUI.VerticalScrollbar 控件中的向下按钮默认使用的样式
scrollView	GUI 滚动视图控件默认使用的样式
customStyles	所有自定义的样式集合
settings	相关设置

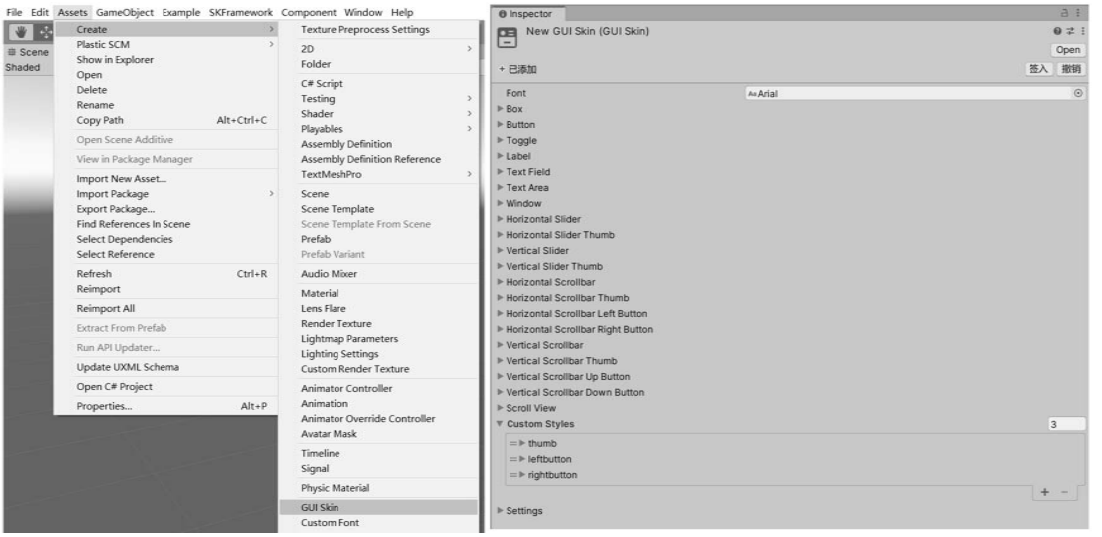

图 5-1　创建 GUISkin 资产

例如，在 Resources 文件夹内创建一个新的 GUISkin 资产，修改与 Label 风格相关的设置，提供一张 Normal 状态下的背景图片，将字体修改为粗体并居中对齐，如图 5-2 所示。

需要注意的是，用于编辑器的相关贴图，需要将其 Texture Type 修改为 Editor GUI and

Legacy GUI，如图 5-3 所示。

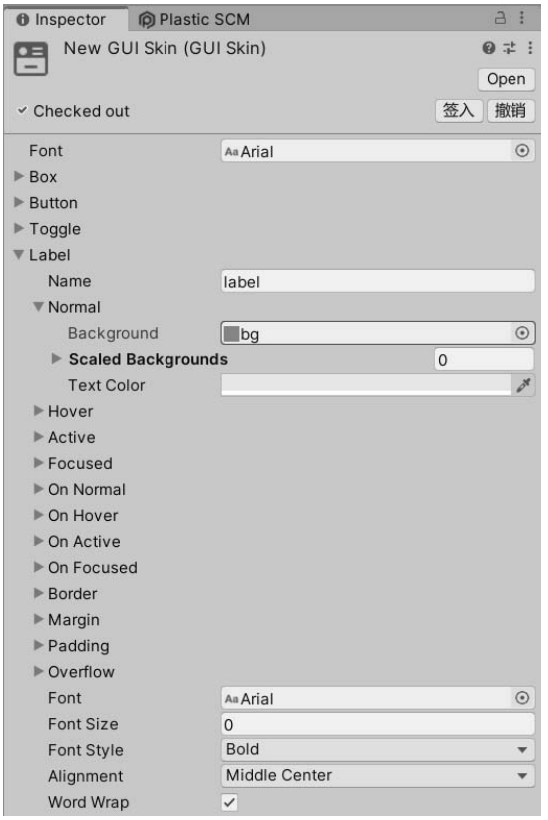

图 5-2　编辑 GUISkin 资产

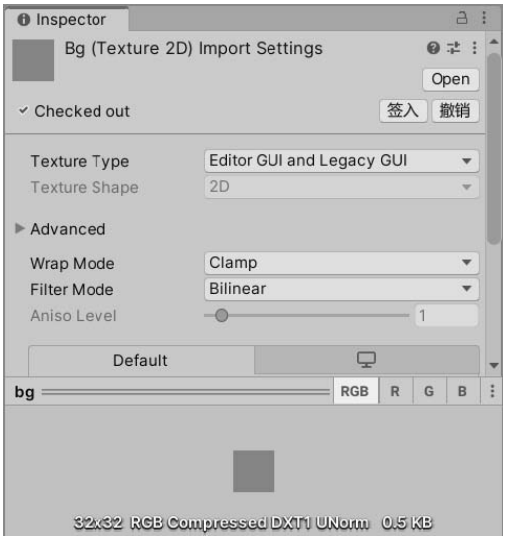

图 5-3　Editor GUI and Legacy GUI

因为该资产位于 Resources 特殊文件夹内，因此可以通过 Resources.Load()方法将该资产加载到内存中，然后获取并使用其中的 Label 风格，代码如下：

```
//第 5 章/GUISkinExampleEditor.cs

using UnityEngine;
using UnityEditor;

[CustomEditor(typeof(GUISkinExample))]
public class GUISkinExampleEditor : Editor
{
    private GUIStyle m_Style;

    public override void OnInspectorGUI()
    {
        base.OnInspectorGUI();
        OnGUIStyleExample();
```

```
    }
    private void OnGUIStyleExample()
    {
        if (m_Style == null)
        {
            GUISkin skin = Resources.Load<GUISkin>("New GUISkin");
            m_Style = skin.label;
        }
        GUILayout.Label("Hello World.", m_Style);
        GUILayout.Label("Today is a good day.", m_Style);
    }
}
```

效果如图 5-4 所示。

图 5-4　样式效果

5.2　GUI 样式

GUIStyle 是 UnityEngine 命名空间中的类，它用于设置编辑器中控件的样式。在绝大部分绘制控件的方法中提供了包含 GUIStyle 参数的重载方法，参数值可以传入 GUIStyle 类型的变量，也可以通过当前的 GUISkin 中的样式名称设定 GUIStyle，代码如下：

```
    GUILayout.Label("Hello World.", new GUIStyle() { alignment =
TextAnchor.MiddleLeft, fontStyle = FontStyle.Italic });
    GUILayout.Label("Hello World.", "BoldLabel");
```

在第 1 个 Label 文本中，创建了一个新的样式，将其对齐方式设置为居中靠左，将字体样式设置为斜体，除此之外，还可以设置字体大小等。GUISytle 中包含的变量见表 5-3。

表 5-3　GUIStyle 中的变量

变　量	详　解
alignment	用于设置文本的对齐方式，包含 UpperLeft、UpperCenter、UpperRight、MiddleLeft、MiddleCenter、MiddleRight、LowerLeft、LowerCenter、LowerRight 9 种类型
font	用于设置文本字体
fontSize	用于设置字体的大小
fontStyle	用于设置字体的样式，包含默认、加粗、斜体、加粗斜体共 4 种类型

续表

变　量	详　解
wordWrap	文本是否应该自动换行
richText	是否为富文本（文本内容是否含标记标签）
textClipping	如何处理要绘制的内容过长而无法放入给定区域的情况，包含 Clip、Overflow 两种类型，分别表示裁剪和溢出
imagePosition	GUIContent 中图像和文本的组合方式，包含 ImageLeft、ImageAbove、ImageOnly、TextOnly 共 4 种类型
contentOffset	要应用该样式的内容的像素偏移
fixedWidth	如果不为 0，则应用该样式的 GUI 元素都将使用该宽度
fixedHeight	如果不为 0，则应用该样式的 GUI 元素都将使用该高度
stretchWidth	是否可以水平拉伸该样式的 GUI 元素来改善布局效果
stretchHeight	是否可以垂直拉伸该样式的 GUI 元素来改善布局效果
normal	正常显示时的渲染设置
hover	鼠标悬浮时的渲染设置
active	按下控件时的渲染设置
focused	聚焦控件时的渲染设置
onNormal	控件处于启用状态时的渲染设置
onHover	控件处于启用状态并且鼠标悬浮在其上方时的渲染设置
onActive	控件处于启用状态并且按下控件时的渲染设置
onFocused	控件处于启用状态并且聚焦控件时的渲染设置
border	背景图片的边框
margin	使用该样式的 GUI 元素与其他 GUI 元素之间的边距
padding	从边缘到内容起始处的空间
overflow	要添加到背景图片的额外空间

在第 2 个 Label 文本中，使用了当前 GUISkin 中的样式名称 BoldLabel 来将其字体样式指定为粗体。通常情况下，在使用默认的 GUISkin 时，开发者并不知道其中包含哪些样式名称，因此需要创建一个工具，以此来查看其中包含的样式。

创建一个新的编辑器窗口，在该窗口中使用滑动列表来列出 GUISkin 中所有的自定义样式，并提供一个输入框，支持检索功能，代码如下：

```
//第 5 章/GUIStylePreviewer.cs

using UnityEngine;
using UnityEditor;

public class GUIStylePreviewer : EditorWindow
{
```

```
    [MenuItem("Example/GUIStyle Previewer")]
    public static void Open()
    {
        GetWindow<GUIStylePreviewer>().Show();
    }

    private Vector2 scroll;
    //检索的内容
    private string searchContent = string.Empty;

    private void OnGUI()
    {
        //搜索栏
        GUILayout.BeginHorizontal(EditorStyles.toolbar);
        GUILayout.Label("Search", GUILayout.Width(50f));
        searchContent = GUILayout.TextField(searchContent,
            EditorStyles.toolbarSearchField);
        GUILayout.EndHorizontal();

        //滚动视图
        scroll = EditorGUILayout.BeginScrollView(scroll);
        for (int i = 0; i < GUI.skin.customStyles.Length; i++)
        {
            var style = GUI.skin.customStyles[i];
            //是否符合检索内容
            if (style.name.ToLower().Contains(searchContent.ToLower()))
            {
                if (GUILayout.Button(style.name, style))
                {
                    //当按钮被单击时将样式的名称复制到粘贴板中
                    EditorGUIUtility.systemCopyBuffer = style.name;
                    Debug.Log(style.name);
                }
            }
        }
        EditorGUILayout.EndScrollView();
    }
}
```

结果如图 5-5 所示。

除了可以使用 GUISkin 中的样式外，EditorStyles 类中也有大量的样式可以使用。同样地，创建一个新的编辑器窗口，在里面列举出 EditorStyles 类中包含的样式，这些样式需要用反射的方式获取，代码如下：

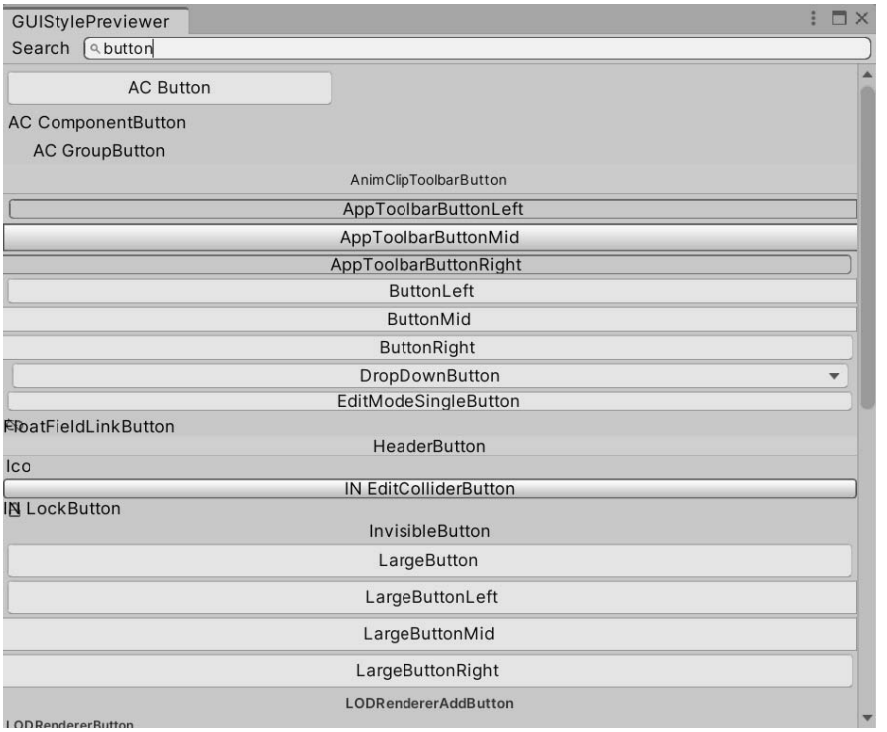

图 5-5　预览默认 GUISkin 中的样式

```
//第 5 章/EditorStylesPreviewer.cs

using System.Linq;
using System.Reflection;

using UnityEngine;
using UnityEditor;

public class EditorStylesPreviewer : EditorWindow
{
    [MenuItem("Example/EditorStyles Previewer")]
    public static void Open()
    {
        GetWindow<EditorStylesPreviewer>().Show();
    }

    private Vector2 scroll;
    private GUIStyle[] styles;

    private void OnEnable()
    {
        //以反射方式获取 EditorStylesPreviewer 中公开的样式
```

```
        PropertyInfo[] propertyInfos = typeof(EditorStyles)
            .GetProperties(BindingFlags.Public | BindingFlags.Static);
        propertyInfos = propertyInfos.Where(m
            => m.PropertyType == typeof(GUIStyle)).ToArray();
        styles = new GUIStyle[propertyInfos.Length];
        for (int i = 0; i < styles.Length; i++)
        {
            styles[i] = propertyInfos[i].GetValue(null, null) as GUIStyle;
        }
    }

    private void OnGUI()
    {
        //滚动视图
        scroll = EditorGUILayout.BeginScrollView(scroll);
        for (int i = 0; i < styles.Length; i++)
        {
            var style = styles[i];
            if (GUILayout.Button(style.name, style))
            {
                //当按钮被单击时将样式的名称复制到粘贴板中
                EditorGUIUtility.systemCopyBuffer = style.name;
                Debug.Log(style.name);
            }
        }
        EditorGUILayout.EndScrollView();
    }
}
```

结果如图 5-6 所示。

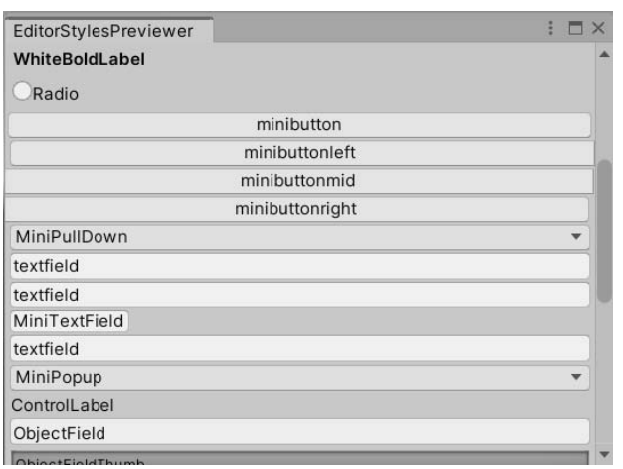

图 5-6　预览 EditorStylesPreviewer 中的公开样式

5.3　GUI 图标

查看 GUILayout 类中的 Label()方法可以发现，其中包含带有 Texture 类型或 GUI Content 类型参数的重载方法，因此 Label 方法不仅可以用于绘制文本内容，还可以用于绘制图片或者图标。例如，加载 Resources 文件夹中名为 Axe 的图片，将它通过 Label()方法绘制到编辑器上，代码如下：

```
//第 5 章/GUISkinExampleEditor.cs

private Texture axe;

private void OnEnable()
{
    axe = Resources.Load<Texture>("Axe");
}
public override void OnInspectorGUI()
{
    base.OnInspectorGUI();
    OnLabelImageExample();
}
private void OnLabelImageExample()
{
    GUILayout.Label(axe, GUILayout.Height(50f));
    GUILayout.Label(new GUIContent("这是一把斧子", axe),
        GUILayout.Height(50f));
}
```

结果如图 5-7 所示。

图 5-7　Label 绘制图片

Unity 中内置了大量的图标资产，如果想要使用这些图标，则需要用到 EditorGUIUtility 类中加载内置图标的方法 IconContent()，代码如下，参数 name 表示图标的名称，根据该名称加载 Assets/Editor Default Resources/Icons 中的图标；text 表示鼠标悬浮时的提示文本，需要以“|”字符开头以将其标记为提示文本。

```
public static GUIContent IconContent (string name, string text= null);
```

调用该方法可以加载 Assets/Editor Default Resources/Icons 中的内置图标资产，但是通常情况下，开发者并不知道里面有哪些图标及其对应的名称，它是不可见的，而且在不同的 Unity 版本中里面会有不同的内容，因此创建一个新的编辑器窗口，在编辑器窗口中列举出这些图标。

获取这些图标的名称可以先通过 Resources 中的 FindObjectsOfTypeAll()方法获取全部贴图资产，然后遍历这些贴图资产，根据其名称调用 IconContent()方法，如果返回的 GUIContent 中的 image 不为空，则说明资产在内置图标资产的文件夹中。为了防止每次打开编辑器窗口时重新加载，在第 1 次加载时将这些图标名称存储于一个文本文件中并写入缓存，代码如下：

```
//第 5 章/GUIIconPreviewer.cs

using System.IO;
using System.Text;
using System.Linq;
using System.Collections.Generic;

using UnityEngine;
using UnityEditor;

public sealed class GUIIconPreviewer : EditorWindow
{
    [MenuItem("Example/Built-In GUIIcon Previewer")]
    public static void Open()
    {
        GetWindow<GUIIconPreviewer>().Show();
    }

    private Vector2 scrollPosition;
    private string searchContent = "";
    private const float width = 50f;

    private List<string> iconNames;
    private string[] matchedIconNames;

    private void OnEnable()
    {
        //存储图标名称的文件路径
        string filePath = Path.GetFullPath(".").Replace("\\", "/")
            + "/Library/built-in gui icon names.txt";
        if (!File.Exists(filePath))
        {
            iconNames = new List<string>();
            StringBuilder sb = new StringBuilder();
```