UNITY3D 编辑器插件编写教程

在阅读本教程之前，你需要对 Unity 的操作流程有一些基础的认识，并且最好了解内置的 GUI 系统如何使用。

如何让编辑器运行你的代码

Unity3D 可以通过事件触发来执行你的编辑器代码，但是我们需要一些编译器参数来告知编译器何时需要触发该段代码。

[MenuItem(XXX)]声明在一个函数上方，告知编译器给 Unity3D 编辑器添加一个菜单项，并且当点击该菜单项的时候调用该函数。触发函数里可以编写任何合法的代码，可以是一个资源批处理程序，也可以弹出一个编辑器窗口。代码里可以访问到当前选中的内容（通过 Selection 类），并据此来确定显示视图。与此类似，[ContextMenu("XXX")]可以向你的上下文菜单中添加一个菜单项。

当你编写了一些 Component 脚本，当它被附属到某个 GameObject 时，想在编辑视图即可在 Scene 视图观察到效果，那么你可以把[ExecuteInEditMode]写在类上方来通知编译器，该类的 OnGUI 和 Update 等函数在编辑模式也也会被调用。我们还可以使用 [AddComponentMenu("XXX/XXX")]来把该脚本关联到 Component 菜单中，点击相应菜单项即可为 GameObject 添加该 Component 脚本。

开始编写编辑器

为了避免不必要的包含，Unity3D 的运行时和编辑器类分辨存储在不同的 Assemblies 里（UnityEngine 和 UnityEditor）。当你准备开始编写编辑器之前，你需要 using UnityEditor 来导入编辑器的名称空间。

有些代码可能是运行时和编辑器都需要执行的，如果你想在其中加以区分，那么可以使用 #if UNITY_EDITOR ... #endif 宏来对编辑器代码做特殊处理。

在你开始真正编写代码之前，我认为你还需要知道所有放在命名为 Editor 目录下的脚本会在其它脚本之后进行编译，这方便了你去使用那些运行时的内容。而那些目录下的脚本是不能访问到 Editor 目录下的内容的。所以，你最好把你的编辑器脚本写在 Editor 目录下。

如何创建自定义编辑器窗口

创建你的窗口

如果你想自定义一个可编辑的面板，那么你需要编写一个继承自 EditorWIndow 的类。通常情况下，你还需要写一个[MenuItem]来告知编译器何时打开这个面板。这个事件的回调应该是一个静态方法，并且返回一个窗口的实例。

现在，当你点击对应的菜单项时，会弹出一个空白的窗口。并且你可以像 Unity3D 编辑器预制的窗口一样随意拖动和停靠。下面来看看我们如何来在窗口内实现我们想要的功能吧。

扩展你的窗口

和运行时的 GUI 一样，如果你需要在窗口中添加交互控件，那么必须重写 OnGUI 方法。具体的重写方式和运行时的 GUI 一样，你甚至可以使用任何扩展自原生 GUI 系统的插件（例如 iGUI 和 GUIX）来简化你的插件开发流程（仅经过初步测试，更深层次的可用性尚待验证）。同时 UnityEditor 名称空间下的 EditorGUILayout 在原生 GUI 之上提供了一些更方便的接口和控件，让你可以轻松的使用一些编辑器特有的 UI 控件。

除了 OnGUI 外，你可能还会需要如下一些回调来触发某些具体的逻辑（完整的列表请参考官方文档）：

- OnSelectionChange，但你点选物品时触发
- OnFocus /OnLostFocus，获得和失去焦点时触发

进一步扩展你的窗口

自定义控件

和运行时 GUI 的使用方式一样，如果你打算自定义自己的控件，那么最简单的方式就是实现一个静态方法（也可以不是静态的），并提供一些可选参数，在方法内部根据这些参数来完成对控件的布局（就像你在 OnGUI 中做的一样）。

如果你打算把自定义控件实现在窗口类内部，你可以使用 Partial 类来更好的管理你的代码。

绘制 2D 内容

绘制图片

可以使用 GUI.DrawTexture 来完成对图片资源的绘制。

绘制基础图元

GUI 本身并没有提供绘制基础图元的方法，但是可以通过一些方式来封装出这些方法。

- 绘制线段：通过一个像素的贴图资源配合 GUI.DrawTexture 和矩阵旋转来完成线段的绘制。
- 绘制矩形框：通过 GUI.Box 和样式设置来封装出对矩形框和矩形填充框。

资源选择器

EditorLayout.ObjectField 控件提供一个资源选择逻辑，生成时需要指定某种资源类型。然后你可以拖动该种资源到该控件或点击控件旁边的小圆圈进行列表进行选择。

如何存储编辑内容

你可能需要创建一个继承自 SerializedObject 的类来保存编辑的数据。继承自 SerializedObject 的对象能用于存储数据而不参与渲染，并可以最终打包到 AssetBundle。

针对当前的编辑选项等内容的存储，可能需要另外一个 SerializedObject 类（和具体的系统设计相关）。

向导式的编辑窗口

在很多情况下可能你都会需要一个有很多参数的编辑面板，然后在编辑结束后有一个按钮加以确认。这你不用自己来实现，UnityEditor 提供了 ScriptableWizard 来帮助你快捷的进行开发。

他是继承自 EditorWindow 的，所以他们的使用是很类似的。不过注意，当你点击确认按钮时，OnWizardCreate()会被调用。另外，ScriptableWizard.DisplayWizard 可以帮助你生成并显示出该窗口。

如何扩展 INSPECTOR 面板

当你在 Unity3D 中点选一个对象时，Inspector 面板会随即显示出此对象的属性。我们可以针对某个类型的对象扩展该面板，这在为 Unity3D 开发插件时是非常有用的。

定义 INSPECTOR 何时被触发

自定义的 Inspector 面板需要继承 Editor 类。由于功能相对具体，所以你无需定义代码何时被触发，对应代码会在你点击它所对应的物体时自动执行。

那么如何定义它所对应的类型呢？只需要在你的类定义之前通过编译器的命令[CustomEditor(typeof(XXX))]就可以完成这项工作了。

访问被编辑的对象

在 Inspector 视图中，我们经常需要访问正在被编辑的对象。Editor 类的成员变量 target 正是提供了这一关联。

尽管如此，需要注意 target 是一个 Object 类型的对象，具体使用时可能需要类型转换（可以使用 C#的泛型来避免重复的类型转换）。

实现你自己的 INSPECTOR 界面

扩展 Editor 与扩展 EditorWindow 唯一的不同在于你需要重写的是 OnInspectorGUI 而不是 OnGUI。另外，如果你想绘制默认的可编辑项，只需调用 DrawDefaultInspector 即可。

在 SCENE 界面定义编辑句柄

当选中一个物体的时候，可能我们希望在 Scene 视图里也能定义一些编辑或展现。这一工作可以通过 OnSceneGUI 和 Handle 类来完成。OnSceneGUI 用来处理来自 Scene 视图的事件，而 Handle 类用来在 Scene 视图实现一些 3D 的 GUI 控件（例如控制对象位置的 Position 控制器）。

具体的使用方式可以参考官方的参考文档。

一些常用的功能说明

- AssetDatabase.CreateAsset 可以帮住你从资源目录中创建一个资源实例。
- Selection.activeObject 返回当前选中的对象。
- EditorGUIUtility.PingObject 用来实现在 Project 窗口中点击某一项的操作。
- Editor.Repaint 用来重绘界面所有的控件。
- XXXImporter 用来设置某种资源的具体导入设置（例如在某些情况下你需要设置导入的贴图为可读的）。
- EditorUtility.UnloadUnusedAssets 用于释放没有使用的资源，避免你的插件产生内存泄漏。
- Event.Use 用来标记事件已经被处理结束了。
- EditorUtility.SetDirty 用来通知编辑器数据已被修改，这样在下次保存时新的数据将被存储。