**Unity3D教程：脚本初级知识（四）**

Posted on 2013年01月29日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 316 次

**用C#编写脚本**

除了语法，使用C#或者Boo编写脚本还有一些不同。最需要注意的是：

1.从MonoBehaviour继承

所有的行为脚本必须从MonoBehaviour继承（直接或间接）。在Javascript中这自动完成，但是必须在C#或Boo脚本中显示申明。如果你在Unity内部使用Asset -> Create -> C Sharp/Boo Script菜单创建脚本，创建模板已经包含了必需的定义。

public class NewBehaviourScript : MonoBehaviour {...} // C#

class NewBehaviourScript (MonoBehaviour): ... # Boo

2.使用Awake或Start函数来初始化

Javascript中放置在函数之外的代码，在C#或Boo中要放置在Awake或Start中。

Awake和Start的不同是Awake在场景被加载时候运行，而Start在第一次调用Update或FixedUpdate函数之前被调用，所有Awake函数在任何Start函数调用之前被调用。

3.类名必须与文件名相同

Javascript中，类名被隐式地设置为脚本的文件名（不包含文件扩展名）。在c#和Boo中必须手工做。

4.在C#中Coroutines有不同语法。

Coroutines必有一个IEnumerator返回类型，并且yield使用yield return… 而不是yield…

using System.Collections;

using UnityEngine;

public class NewBehaviourScript : MonoBehaviour {

// C# coroutine

IEnumerator SomeCoroutine ()

{

yield return 0;// 等一帧

yield return new WaitForSeconds (2);//等两秒

}  
}

5．不要使用命名空间

目前Unity还不支持将代码放置在一个命名空间中，这个需要将会出在未来的版本中。

6.只有序列化的成员变量会显示在检视面板中

私有和保护成员变量只在专家模式中显示，属性不被序列化或显示在检视面板中。

7．避免使用构造函数

不要在构造函数中初始化任何变量，使用Awake或Start实现这个目的。即使是在编辑模式中Unity也自动调用构造函数，这通常发生在一个脚本被编译之后，因为需要调用构造函数来取向一个脚本的默认值。构造函数不仅会在无法预料的时刻被调用，它也会为预设或未激活的游戏物体调用。

单件模式使用构造函数可能会导致严重的后果，带来类似随机null引用异常。

因此如果你想实现，如，一个单件模式，不要使用构造函数，而是使用Awake。其实上，没有理由一定要在继续自MononBehaviour类的构造函数中写任何代码。

**性能优化**

1．使用静态类型

在使用Javascript时最重要的优化是使用静态类型而不是动态类型，Unity使用一种叫做类型推理的技术来自自动转换Javascript为静态类型编码而不需要你做任何工作。

var foo=5；

在上面的例子里foo会自动被推断为一个整型值。因此，Unity可能使用大量的编译时间来优化。而不使用耗时的动态名称变量查找等。这就是为什么Unity比其他在JavaScript的实现平均快20倍的原因之一。

唯一的问题是，有时并非一切都可以做类型推断。Unity将会为这些变量重新使用动态类型。通过回到动态类型，编写JavaScript代码很简单。但是这也使得代码运行速度较慢。

让我们看一些例子：

function Start ()  
{  
var foo = GetComponent(MyScript);  
foo.DoSomething();  
}

这里foo将是动态类型，因此调用DoSomething函数将使用较长时间，因为foo的类型是未知的，它必须找出它是否支持DoSomething函数，如果支持，调用它。

function Start ()  
{  
var foo : MyScript = GetComponent(MyScript);  
foo.DoSomething();  
}

这里我们强制foo为指定类型，你将获得更好的性能。

2．使用#pragma strict

当然现在问题是，你通常没有意识到你在使用动态类型。#pragma strict解决了这个！简单的在脚本顶部添加#pragma strict。然后，unity将在脚本中禁用动态类型，强制使用静态类型，如果一个类型未知。Unity将报告编译错误。那么在这种情况下foo将在编译时产生一个错误：

#pragma strict

function Start ()  
{  
var foo = GetComponent(MyScript);  
foo.DoSomething();  
}

3.缓存组件查找

另一个优化是组件缓存。不幸的是该优化需要一点编码，并且不一定是值得的，但是如果你的脚本是真的用了很长时间了，你需要把最后一点表现出来，这是一个很好的优化。

当你访问一个组件通过GetComponent或访问变量，Unity会通过游戏对象找到正确的组件。这一次可以很容易地通过缓存保存在一个私有变量里引用该组件。

简单地把这个：

function Update ()  
{

transform.Translate(0, 0, 5);  
}

变成：

private var myTransform : Transform;

function Awake ()  
{  
myTransform = transform;  
}

function Update ()  
{  
myTransform.Translate(0, 0, 5);  
}

后者的代码将运行快得多，因为Unity没有找到变换在每一帧游戏组件中的对象。这同样适用于脚本组件，在你使用GetComponent代替变换或者其它的东西。

4.使用内置数组

内置数组的速度非常快，所以请使用它们。

而整列或者数组类更容易使用，因为你可以很容易地添加元素，他们几乎没有相同的速度。内置数组有一个固定的尺寸，但大多数时候你事先知道了最大的大小在可以只填写了以后。关于内置数组最好的事情是，他们直接嵌入在一个结构紧凑的缓冲区的数据类型没有任何额外的类型信息或其他开销。因此，遍历是非常容易的，作为一切缓存在内存中的线性关系。

private var positions : Vector3[];

function Awake ()  
{  
positions = new Vector3[100];  
for (var i=0;i<100;i++) positions = Vector3.zero; }

5.如果你不需要就不要调用函数

最简单的和所有优化最好的是少工作量的执行。例如，当一个敌人很远最完美的时间就是敌人入睡时可以接受。直到玩家靠近时什么都没有做。这是种缓慢的处理方式的情况：

function Update ()

{

if (Vector3.Distance(transform.position, target.position) > 100)// 早期进行如果玩家实在是太遥远。  
return;  
perform real work work...  
}

这不是一个好主意，因为Unity必须调用更新功能，而你正在执行工作的每一个帧。一个比较好的解决办法是禁用行为直到玩家靠近。有3种方法来做到这一点：

1.使用OnBecameVisible和OnBecameInvisible。这些回调都是绑到渲染系统的。只要任何相机可以看到物体，OnBecameVisible将被调用，当没有相机看到任何一个，OnBecameInvisible将被调用。这种方法在很多情况下非常有用，但通常在AI中并不是特别有用，因为只要你把相机离开他们敌人将不可用。

function OnBecameVisible () {  
enabled = \*\*\*e;  
}

function OnBecameInvisible ()  
{  
enabled = false;  
}

2.使用触发器。一个简单的球形触发器会工作的非常好。一旦你离开这个影响球你将得到OnTriggerEnter/Exit调用。

function OnTriggerEnter (c : Collider)  
{  
if (c.CompareTag("Player"))  
enabled = \*\*\*e;  
}

function OnTriggerExit (c : Collider)  
{  
if (c.CompareTag("Player"))  
enabled = false;  
}

3.使用协同程序。Update调用的问题是它们每帧中都发生。很可能会只需要每5秒检检查一次到玩家的距离。这应该会节省大量的处理周期。

**脚本编译（高级）**

Unity编译所有的脚本为.NET dll文件，.dll将在运行时编译执行。

这允许脚本以惊人的速度执行。这比传统的javascript快约20倍。比原始的C++代码慢大约50%。在保存的时候，Unity将花费一点时间来编译所有脚本，如果Unity还在编译。你可以在Unity主窗口的右下角看到一个小的旋转进度图标。

**脚本编译在4个步骤中执行：**

1.所有在"Standard Assets", "Pro Standard Assets" 或 "Plugins"的脚本被首先编译。

在这些文件夹之内的脚本不能直接访问这些文件夹之外脚本。不能直接引用或它的 变量，但是可以使用GameObject.SentMessage与它们通信。

2.所有在"Standard Assets/Editor", "Pro Standard Assets/Editor" 或 "Plugins/Editor"的脚本被首先编译。

如果你想使用UnityEditor命名空间你必须放置你的脚本在这些文件夹中，例如添加菜单项或自定义的向导，你都需要放置脚本到这些文件夹。这些脚本可以访问前一组中的脚本。

3.然后所有在"Editor"中的脚本被编译。

如果你想使用UnityEditor命名空间你必须放置你的脚本在这些文件夹中。例如添加菜单单项或自定义的向导，你都需要放置脚本到这些文件夹。这些脚本可以访问所有前面组中的脚本，然而它们不能访问后面组中的脚本。这可能会是一个问题，当编写编辑器代码编辑那些在后面组中的脚本时。有两个解决方法：1、移动其他脚本到"Plugins"文件夹 2、利用JavaScript的动态类型，在javascript中你不需要知道类的类型。在使用GetComponent时你可以使用字符串而不是类型。你也可以使用SendMessage，它使用一个字符串。

4.所有其他的脚本被最后编译

所有那些没有在上面文件夹中的脚本被最后编译。所有在这里编译的脚本可以访问第一个组中的所有脚本（"Standard Assets"，"ProStandard Assets" or "Plugins"）。这允许你让不同的脚本语言互操作。例如，如果你想创建一个JavaScript。它使用一个C#脚本;放置C#脚本到"Standard Assets"文件夹并且JavaScript放置在"Standard Assets"文件夹之外。现在JavaScript可以直接引用c#脚本。放置在第一个组中的脚本，将需要较长的编译时间，因为当他们被编译后，第三组需要被重新编译。因此如果你想减少编译时间，移动那些不常改变的到第一组。经常改变的到第四组。