**Unity3D中的多线程及使用多线程**

Posted on 2013年04月10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 289 次

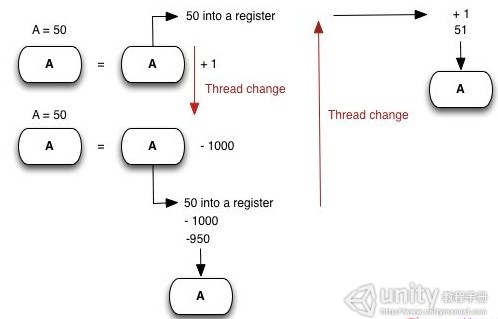
**Unity3D**中的多线程。线程是一个相当复杂的话题，但如果掌握了它，就可以从容的使用多个硬件处理器或处理很难划分管理数据块。

如在场景中用A\*算法进行大量的数据计算，变形网格中操作大量的顶点，持续的要运行上传数据到服务器，二维码识别等图像处理，如果同时你要处理很多事情或者与Unity的对象互动小可以用thread，否则使用coroutine。

线程是在程序中与其他线程同时运行的进行。在多处理器的计算机上可以做到多个线程的真正的同步，更多的线程取决于有多个处理核心。

[**Unity3D**](http://www.unitymanual.com/category/manual)编程时，总有个主线程执行你的代码，也可以创建额外的线程和主线程同时运行。而Unity中，你仅能从主线程中访问[**Unity3D**](http://www.unitymanual.com)的组件，对象和Unity3D系统调用。任何企图访问这些项目的第二个线程都将失败并引发错误，这是一个要重视的一个限制。

所以当你写代码时，你认为一个函数开始并达到它执行的点后返回，同样你做的东西又在另外一个函数中执行，但又没有发生相应的变化。操作系统决定你代码的执行，任何时候,你的代码只能暂时”休眠”掉，然后让另外的代码开始运行。

[](http://www.unitymanual.com/wp-content/uploads/2013/04/QQ截图20130410100916.jpg)

Unity3D中的多线程及使用多线程

在这个例子中，在第一个线程将A的值加载到CPU寄存器中准备+1后被中断，第二个线程来读取A的值，并减去1000，这时A应该是-950，现在第一个线程重新开始，它在寄存器中的50+1的结果存储于A，A变成了51，而-950已经丢掉了。

从根本上说，要在用多个线程在同时对变量或内存访问时，要采取很多预防措施来确保不会发生这样的事。所以Unity决定从另外线程访问这些变量或者内存是无效的，只是为了避免所有系统和框架对象出现问题。所以要确保一次只有一个线程来修改变量，这不意味着你不能用多线程工作，你可以用”排序”来解决这个问题。

C#中有lock这个关键字,以确保只有一个线程可以在特定时间内访问特定的对象，这里说对象是因为。法锁定一个类型值(value type)或原型(primitive)。

int a = 50;

object guard = new object();

void ThreadOneCode()  
{  
//一些代码在这

lock(guard)  
{  
a = a + 1;  
}

//其余一些代码在这

}

void ThreadTwoCode()  
{  
//一些代码在这

lock(guard)  
{  
a = a - 1000;  
}

//其余一些代码在这}

所有都锁定在guard内，保证同一个时间只有一个线程通过guard访问它，你可以使用任何合适的对象。现在你可能会有各种各样的问题，比如你要锁定的不止一件事，可能是互相嵌套的。那我们该怎么办呢?

我们这个类叫Loom，让你可以轻松在另一个线程运行代码,这里有两个要注意的功能:

RunAsync(Action)-在另一个线程上运行的一组代码.

QueueOnMainThread(Action,[可选]float time)-运行在主线程的语句(可选延迟).

用Loom.Current访问Loom-创建一个看不见的GameObject用来处理游戏主线程的互动，下面这个例子用Loom来更新一个网格所有的顶点乘的结果。

//缩放一个网格在第二个线程void ScaleMesh(Mesh mesh, float scale) { //Get the vertices of a mesh var vertices = mesh.vertices; //运行一个Action在新的线程Loom.RunAsync(()=>{ //遍历所有的顶点for(var i = 0; i < vertices.Length; i++) { //缩放顶点vertices[i] = vertices[i] \* scale; } //在主线程上运行一些代码//更新网格Loom.QueueOnMainThread(()=>{ //设置顶点mesh.vertices = vertices; //重新计算边界mesh.RecalculateBounds(); });

上面这个是个很好的例子，使用lambda函数在第二个线程上做一个没有参数，不需要返回任何内容的操作。closures都是在你自己的类和函数的参数和局部变量的访问。你可以用 ()=>{ … } 定义一个lambda函数来在新的线程上运行函数内所有的代码。

在主线程上我们需要将修改的网格顶点更新,所以我们使用QueueOnMainThread在接下来的时间更新周期运行处理(此帧或下一帧被称为接下来的更新周期)。 QueueOnMainThread也需要一个Action来将更新的顶点更新到原来的网格。

如果是UnityScript，你可以这样使用Loom：

//缩放一个网格在第二个线程function ScaleMesh(mesh : Mesh, scale : float) { //Get the vertices of a mesh var vertices = mesh.vertices; //运行一个Action在新的线程Loom.RunAsync(function() { //遍历所有的顶点for(var i = 0; i < vertices.Length; i++) { //缩放顶点vertices[i] = vertices[i] \* scale; } //在主线程上运行一些代码//更新网格Loom.QueueOnMainThread(function() { //设置顶点mesh.vertices = vertices; //重新计算边界mesh.RecalculateBounds(); });