Unity之异步加载

当我们执行场景跳转的时候，经常会出现游戏卡顿的情况，为了过度掉这段卡顿的情况，可以在游戏中再次创建一个新的场景，专门用来加载将要跳转的下一个场景。

那么通常情况下，这个新创建的场景是用来加载下一个即将跳转的场景的，一般需要使用异步加载。

在我们学习异步加载之前需要先学习下，Unity中的协同程序。

1. 协同程序(协程)

(1)什么是协同程序?

协同程序，即在主程序运行时同时开启另一段逻辑处理，来协同当前程序的执行。换句话说，开启协同程序就是开启一个”线程"(伪线程)。

(2)协同程序的开启

在Unity3D中，使用MonoBehaviour.StartCoroutine方法即可开启一个协同程序，也就是说该方法必须在MonoBehaviour或继承于MonoBehaviour的类中调用。

在Unity3D中，使用StartCoroutine(string methodName)和StartCoroutine(IEnumerator routine)都可以开启一个线程。区别在于使用字符串作为参数时，开启线程时最多只能传递一个参数，并且性能消耗会更大一点，而使用IEnumerator作为参数则没有这个限制。

(3)协同程序的终止

在Unity3D中，使用StopCoroutine(string methodName)来终止一个协同程序，使用StopAllCoroutines()来终止所有可以终止的协同程序，但这两个方法都只能终止该MonoBehaviour中的协同程序。

还有一种方法可以终止协同程序，即将协同程序所在gameobject的active属性设置为false，当再次设置active为ture时，协同程序并不会再开启；如是将协同程序所在脚本的enabled设置为false则不会生效。这是因

为协同程序被开启后作为一个线程在运行，而MonoBehaviour也是一个线程，他们成为互不干扰的模块，除非代码中调用，他们共同作用于同一个对象，只有当对象不可见才能同时终止这两个线程。然而，为了管理我们额外开启的线程，Unity3D将协同程序的调用放在了MonoBehaviour中，这样我们在编程时就可以方便的调用指定脚本中的协同程序，而不是无法去管理，特别是对于只根据方法名来判断线程的方式在多人开发中很容易出错，这样的设计保证了对象、脚本的条理化管理，并防止了重名。

2. 异步加载场景

下面我们学习异步加载游戏场景，异步，顾名思义就是不影响当前游戏场景的前提下加载新场景。通常异步加载的方式分为两种：第一种是异步加载新游戏场景，当新场景加载完成后进入新场景并且销毁之前的场景。第二种:同样异步加载新场景，新场景加载完毕后，保留旧场景的游戏对象并且进入新场景。

(1)第一种异步加载的方法是:

```
Application.LoadLevelAsync("yourScene");
```

(2)第二种异步加载的方法是:

```
Application.LoadLevelAdditiveAsync (“yourScene");
```

这两种方法加载的方式完全一样。它们的返回值是一个AsyncOperation对象，通过该对象能够获取游戏加载的进度。异步加载其实重要还是应用于游戏Loading界面，毕竟Loading如果采用同步的机制会影响用户体验，简单明了的说法就是当前场景为A，加载场景为B，最终要跳到的场景为C，那么对应关系就是A->B->C。用B进行过度达到一个良好的用户体验。

同时为了游戏的可维护性，一般跳转的场景名称用一个类进行封装，用于记录场景的变量设为该类的静态成员。例如

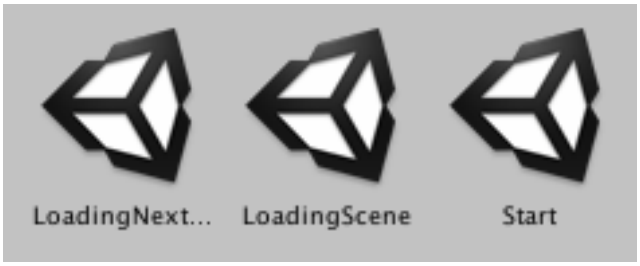
```
using UnityEngine;
using System.Collections;

public class Globe
{
```

```
        //在这里记录当前切换场景的名称
    public static string loadName;
}
```

我们来看看协同是如何在场景加载的时候应用的。

2.1 协同在异步加载中的应用

首先，需要创建一个Unity工程，然后新建几个不同的场景，每个场景最好不一样，用以标记不同的场景，其中一个场景为过度场景，用以加载跳转之后的场景。这三个场景都需要添加到Build Settings中。如下所示。



其中LoadingScene为开始场景，过度场景为LoadingNextScene，目标场景为Start，在LoadingScene中添加一个按钮，当点击按钮后直接进入LoadingNextScene场景，在LoadingScene中有一个进度条，如下所示。

上图的进度条用以记录游戏的加载进度。其中的逻辑代码如下所示

```
using UnityEngine;
using System.Collections;
public class LoadingNextScene : MonoBehaviour {
	private AsyncOperation asy;
	public UIProgressBar progressB;
	// Use this for initialization
	void Start () {
		if(progressB)
			progressB.value = 0;
		StartCoroutine(loadScene());
	}
	// Update is called once per frame
	void Update () {
		Debug.Log(asy.progress);
	//asy.progress在等于0.9时会跳转，因此需要在0.9的时候进度条走满
		if(asy.progress< 0.9f)
			progressB.value = asy.progress;
		else
			progressB.value = 1;
	}
	IEnumerator loadScene(){
		asy = Application.LoadLevelAsync("Start");
		yield return asy;//这里必须用”yield return”返回一个值
	}
}
```

当asy.progress为0.9的时候，开始跳转到Start场景中。

在这里需要说明一下，也可以直接跳转到场景中，然后异步的去激活场景的对象，这样也能让游戏不会出现卡顿的现象，可以自己尝试。例如如下代码。

(1)

```
using UnityEngine;
using System.Collections;

public class Test : MonoBehaviour {

    //这里是需要加载激活的游戏对象
    public GameObject  ☑ Objects;

    //当前加载的进度
    int load_index =0;
    void Start ()
    {
        //开启一个异步任务，加载模型。
         StartCoroutine(loadObject());
    }

    IEnumerator loadObject()
    {
         //便利所有游戏对象
         foreach(GameObject obj in Objects)
         {
              //激活游戏对象
              obj.active = true;
              //记录当前加载的对象
              load_index ++;

              //这里可以理解为通知主线程刷新UI
              yield return 0;
         }
         //全部便利完毕返回
         yield return 0;
    }

    void OnGUI ()
    {
        //显示加载的进度
         GUILayout.Box("当前加载的对象ID是:  " + load_index);
```

```
    }
}
```

也可以结合我们前面上的用预设体动态创建对象。代码如下

```
using UnityEngine;
using System.Collections;

public class Main : MonoBehaviour
 {

    public int count;
    //在编辑器中预设一个游戏对象
    public  GameObject prefabs;

    void Start ()
    {
        StartCoroutine(loaditem());
    }

    void OnGUI()
    {
        GUILayout.Box("游戏对象已经加载到 :  " + count);
    }

    IEnumerator loaditem()
    {
        //开始加载游戏对象
        for(int i =0; i< 1000; i++)
        {

            Instantiate(prefab);
            count = i;
            //可以理解为刷新UI，显示新加载的游戏对象
            yield return 0;
        }
        //结束
        yield return 0;
    }
}
```