7.Courtine 实现原理

1 枚举器的封装到协程

在之前我们对枚举器 IEnumerator 枚举器接口和 yield 关键字做出了解释说明，开始对协程（coroutine）解释说明。

其实协程的原理非常简单，我们来试着简单实现一下

通过 Unity 的 Update() 模拟协程。

修改上面代码，不使用 while 循环或者 foreach 关键字枚举元素。我们将在 Update() 实时刷新来枚举所有元素。

原理就是上面所说的每次 MoveNext() 方法后，改变当前得带位置为下一个元素位置。测试代码如下:

```
/************************************************************************
 * http://sikiedu.com liangxie
 ************************************************************************/

using System.Collections;
using UnityEngine;

namespace UniRxLesson
{
    public class CoroutineExample : MonoBehaviour
    {
        private Update2CoroutineTest mUpdate2CoroutineTest = new
Update2CoroutineTest();

        IEnumerator mE;
```

```
        public CoroutineExample()
        {
            mE = mUpdate2CoroutineTest.GetEnumerator();
        }

        private void Update()
        {
            if (mE.MoveNext())
            {
            }
        }


        class Update2CoroutineTest
        {
            public IEnumerator GetEnumerator()
            {
                Debug.Log("协程:" + 1);
                yield return 0;
                Debug.Log("协程:" + 2);
                yield return 0;
                Debug.Log("协程:" + "枚举器");
                yield return 0;
            }
        }
    }
}
```

输出结果为:

```
协程:1
协程:2
协程:枚举器
```

2 协程

现在我们用 StartCoroutine（）方法启动协程。

代码如下:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CoroutineTest : MonoBehaviour
{
    void Start()
    {
        CoroutineJsTest coroutineJsTest = new CoroutineJsTest();
        StartCoroutine(coroutineJsTest.GetEnumerator());
    }
}

public class CorotuineJsTest
{
    public IEnumerator GetEnumerator()
    {
        Debug.Log("协程:" + 1);
        yield return 0;
        Debug.Log("协程:" + 2);
        yield return 0;
        Debug.Log("协程:" + "枚举器");
        yield return 0;
    }
}
```

输出结果如下:

```
协程:1
协程:2
```

协程：枚举器

从上面可以看出，在 update 方法模拟协程和使用 Unity 自带 StartCoroutine() 方法启动协程效果差不多。看来自 Unity 实现的 StartCoroutine() 启动协程和我们 Update() 模拟是一样的。但是也不确定到底是不是通过类似 Update() 方法实现的。反编译 UnityEngine.dll 程序集也没有找到具体实现方法。。。。。但是唯一确定的一点就是 Unity 也是通过枚举一步步运行程序块的。

类似 Update() 模拟协程，每次遇到 yield return，就执行 yield 类型的 MoveNext() 方法，改变当前迭代位置为下一个元素位置。等待下一次 MoveNext() 方法调用。

StartCoroutine() 方法会不停的调用 MoveNext() 方法（这样就类似于 foreach）。直到枚举结束。

但是注意的是，yield return 后面跟的值除了 Unity 自带的累（如：new WaitForSeconds（0.2f)。继承自 YieldInstruction）

和协程语句块（返回值为 IEnumerator 的方法），其他值没有意义（yield return 0 和 yield return null） 其实都是一样的，只是遇到 yield return 就做相同处理，不会去处理后面跟的值了）。

Coroutine 的实现原理大致如此。

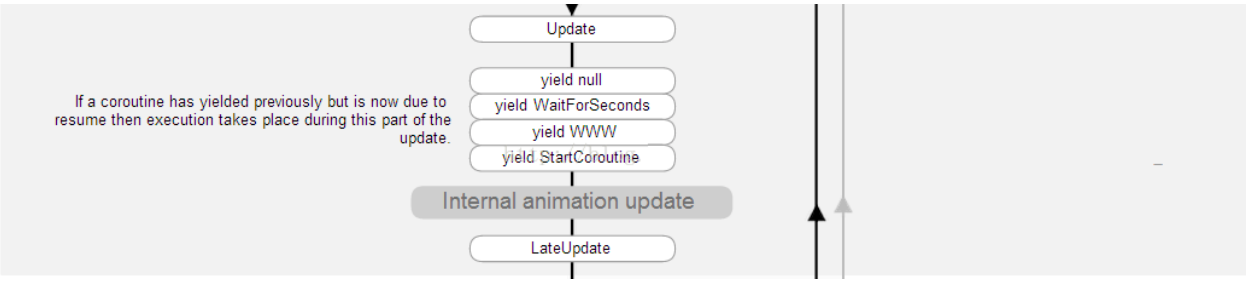
3.总结

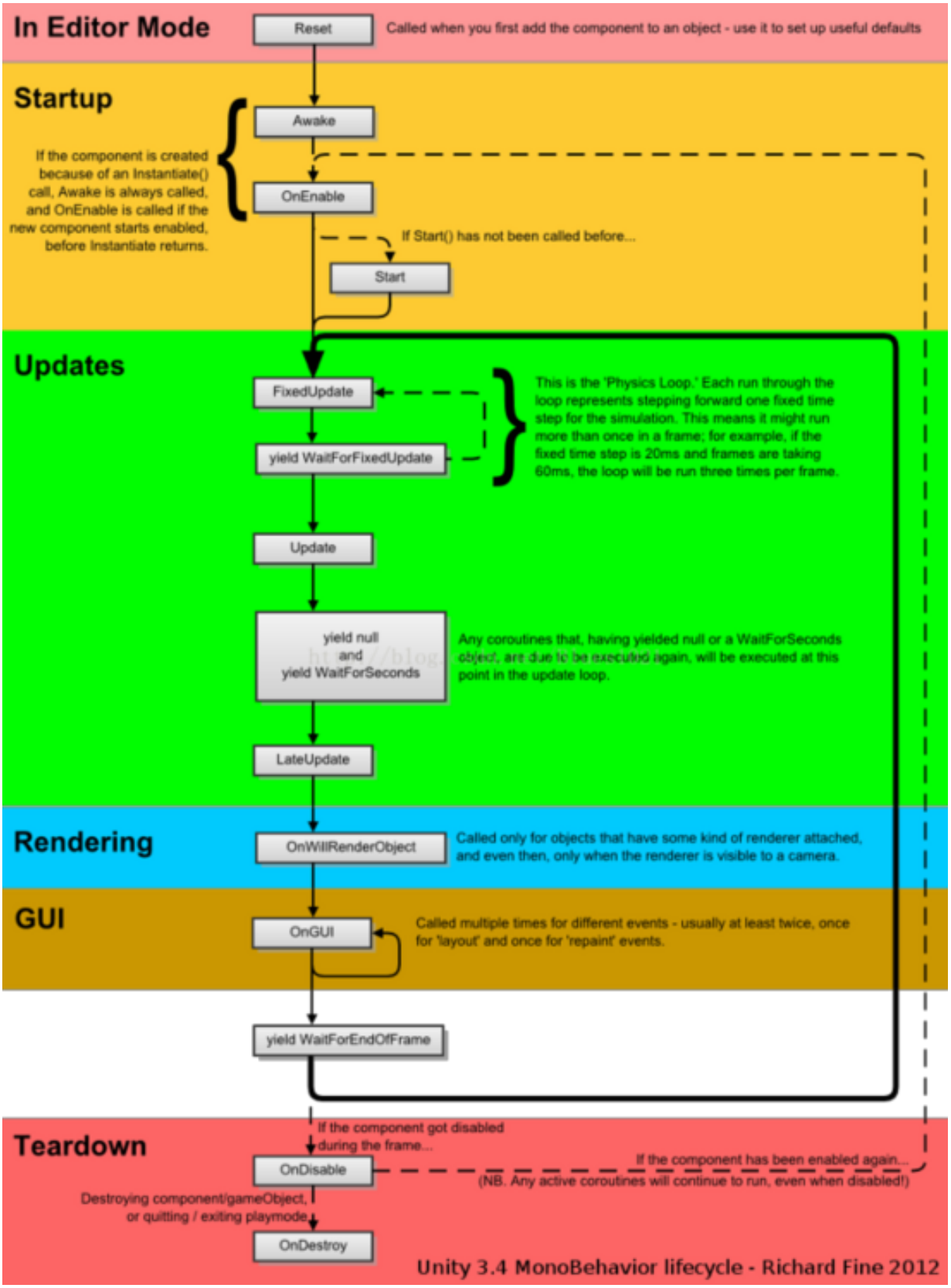
Coroutine 其实是一个 迭代器模式 + 定时器的一种实现。

与我们的 UniRx 的 Observable + Scheduler + LINQ 非常类似。

关于协程的原理我们理解到这里就够了。

赠送两张图:


Update
yield null
yield WaitForSeconds
yield WWW
yield StartCoroutine
If a coroutine has yielded previously but is now due to resume then execution takes place during this part of the update.
Internal animation update
LateUpdate


In Editor Mode
Reset
Called when you first add the component to an object - use it to set up useful defaults
Startup
If the component is created because of an Instantiate() call, Awake is always called, and OnEnable is called if the new component starts enabled, before Instantiate returns.
Awake
OnEnable
If Start() has not been called before...
Start
Updates
FixedUpdate
yield WaitForFixedUpdate
This is the 'Physics Loop.' Each run through the loop represents stepping forward one fixed time step for the simulation. This means it might run more than once in a frame; for example, if the fixed time step is 20ms and frames are taking 60ms, the loop will be run three times per frame.
Update
yield null
and
yield WaitForSeconds
Any coroutines that, having yielded null or a WaitForSeconds object, are due to be executed again, will be executed at this point in the update loop.
LateUpdate
Rendering
OnWillRenderObject
Called only for objects that have some kind of renderer attached, and even then, only when the renderer is visible to a camera.
GUI
OnGUI
Called multiple times for different events - usually at least twice, once for 'layout' and once for 'repaint' events.
yield WaitForEndOfFrame
Teardown
If the component got disabled during the frame...
OnDisable
If the component has been enabled again...
(NB. Any active coroutines will continue to run, even when disabled!)
Destroying component/gameObject, or quitting / exiting playmode
OnDestroy
Unity 3.4 MonoBehavior lifecycle - Richard Fine 2012

我们今天的内容就这些。