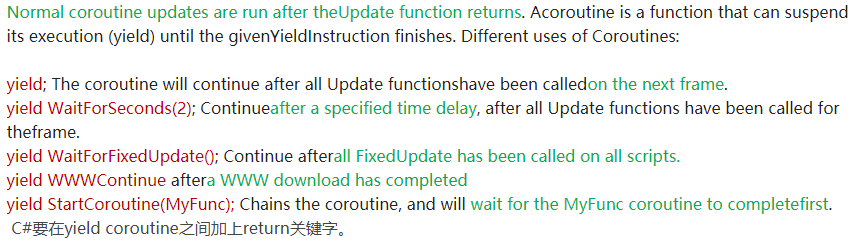
<http://blog.csdn.net/zzmkljd/article/details/51493905>

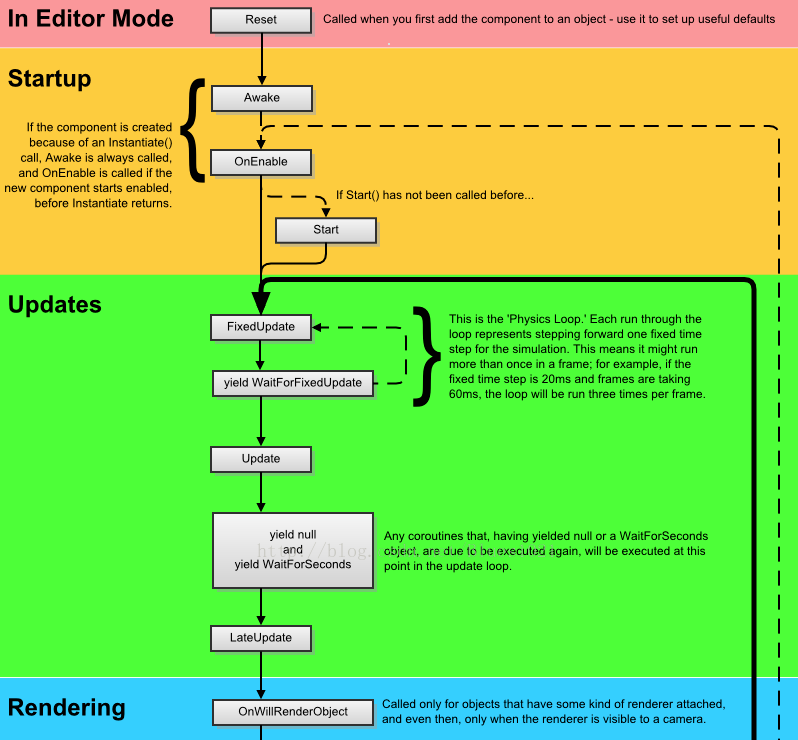
<http://blog.csdn.net/jasonwang18/article/details/55519165>（深入理解）

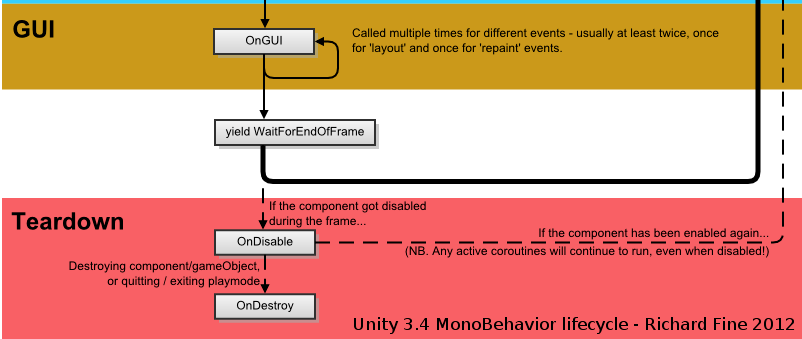
<http://blog.csdn.net/tkokof1/article/details/11842673> （解析unity的协程）

<http://dsqiu.iteye.com/blog/2049743>

<http://altdevblog.com/2011/07/07/unity3d-coroutines-in-detail/>







**1.使用的地方和不能使用的地方：**

必须在MonoBehaviour或继承于MonoBehaviour的类中调用 yield coroutine。yield不可以在Update或者FixedUpdate里使用。

**2.开启协程：**

StartCoroutine(string methodName)和StartCoroutine(IEnumeratorroutine)都可以开启一个协程，

区别：

使用字符串作为参数时，开启协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.删除协程：**

1).在Unity3D中，使用StopCoroutine(stringmethodName)来终止该MonoBehaviour指定方法名的一个协同程序，使用StopAllCoroutines()来终止所有该MonoBehaviour可以终止的协同程序。

包括StartCoroutine(IEnumerator routine)的。

2).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。

协程不是线程，也不是异步执行的，协程和MonoBehaviour的Update函数一样也是在Main Thread中执行的，使用协程你不用考虑同步或锁的问题

public class TestCoroutine : MonoBehaviour {

private bool isStartCall = false;

private bool isUpdateCall = false;

private bool isLateUpdateCall = false;

// Use this for initialization

void Start () {

if (!isStartCall)

{

Debug.Log("Start call begin");

StartCoroutine(StartCoutine());

Debug.Log("Start call after");

isStartCall = true;

}

}

// Update is called once per frame

void Update () {

if (!isUpdateCall)

{

Debug.Log("Update call begin");

StartCoroutine(UpdateCoutine());

Debug.Log("Update call after");

isUpdateCall = true;

}

}

void LateUpdate()

{

if (!isLateUpdateCall)

{

Debug.Log("LateUpdate call begin");

StartCoroutine(LateUpdateCoutine());

Debug.Log("LateUpdate call after");

isLateUpdateCall = true;

}

}

IEnumerator StartCoutine()

{

Debug.Log("This is Start Coroutine call before");

yield return new WaitForSeconds(1.0f);

Debug.Log("This is Start Coroutine call end");

}

IEnumerator UpdateCoutine()

{

Debug.Log("This is Update Coroutine call before");

yield return new WaitForSeconds(1.0f);

Debug.Log("This is Update Coroutine call end");

}

IEnumerator LateUpdateCoutine()

{

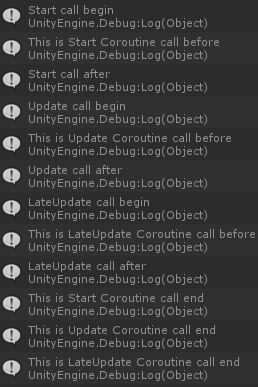
Debug.Log("This is LateUpdate Coroutine call before");

yield return new WaitForSeconds(1.0f);

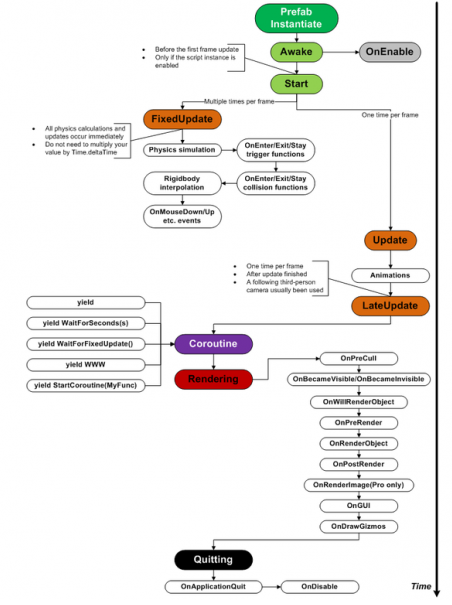
Debug.Log("This is LateUpdate Coroutine call end");

}

}



**实际上协程的执行时在LateUpdate之后的**



如果我们将Update修改为如下：也就是添加了一个this.enable=false

void Update () {

if (!isUpdateCall)

{

Debug.Log("Update call begin");

StartCoroutine(UpdateCoutine());

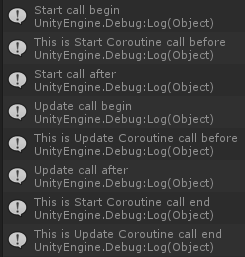
Debug.Log("Update call after");

isUpdateCall = true;

**this.enabled = false;**

}

}



我们会发现，我们开启的协程仍然会执行，当然LateUpdate不会执行，也就不会调用LateUpdate的协程了。

如果我们将Update修改如下：实际上也就是添加gameObject.SetActive(false);

void Update () {

if (!isUpdateCall)

{

Debug.Log("Update call begin");

StartCoroutine(UpdateCoutine());

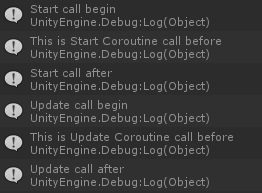
Debug.Log("Update call after");

isUpdateCall = true;

**gameObject.SetActive(false);**

}

}



我们会发现启动的协程不执行了

**总结：**

通过设置monoBehaviour脚本的enable对协程是没有影响的，但如果gameObejct.setActive(false)则已经启动的协程完全停止了，即使在Inspector把gameObject 激活还是没有继续执行。也就说协程虽然是在MonoBehvaviour启动的（StartCoroutine）但是协程函数的地位完全是跟MonoBehaviour是一个层次的，不受MonoBehaviour的状态影响，但跟MonoBehaviour脚本一样受gameObject 控制。也应该是和MonoBehaviour脚本一样每帧“轮询” yield 的条件是否满足。