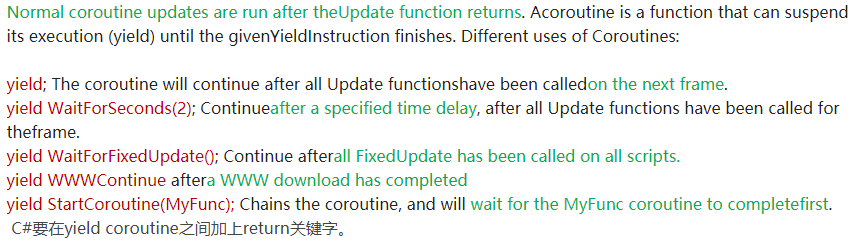
<http://blog.csdn.net/zzmkljd/article/details/51493905>

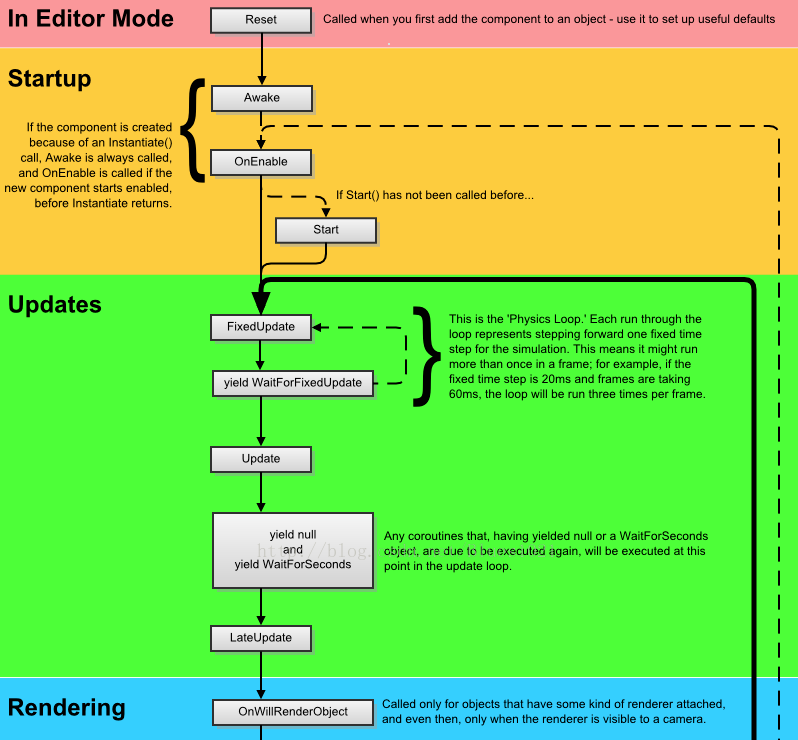
<http://blog.csdn.net/jasonwang18/article/details/55519165>（深入理解）

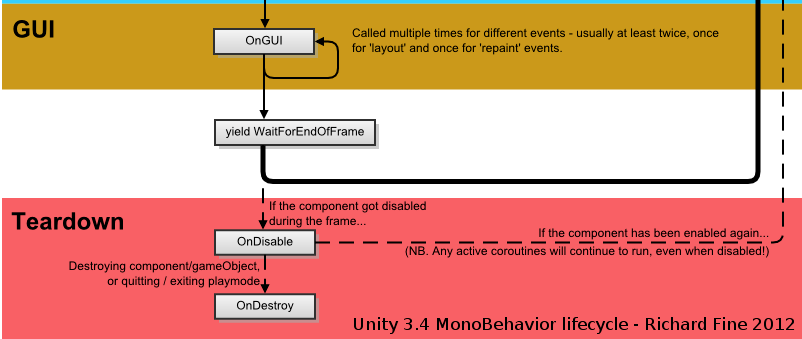
<http://blog.csdn.net/tkokof1/article/details/11842673> （解析unity的协程）

<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()

{

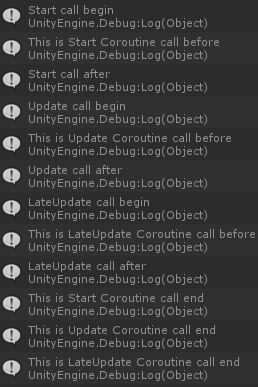
Debug.Log("This is LateUpdate Coroutine call before");

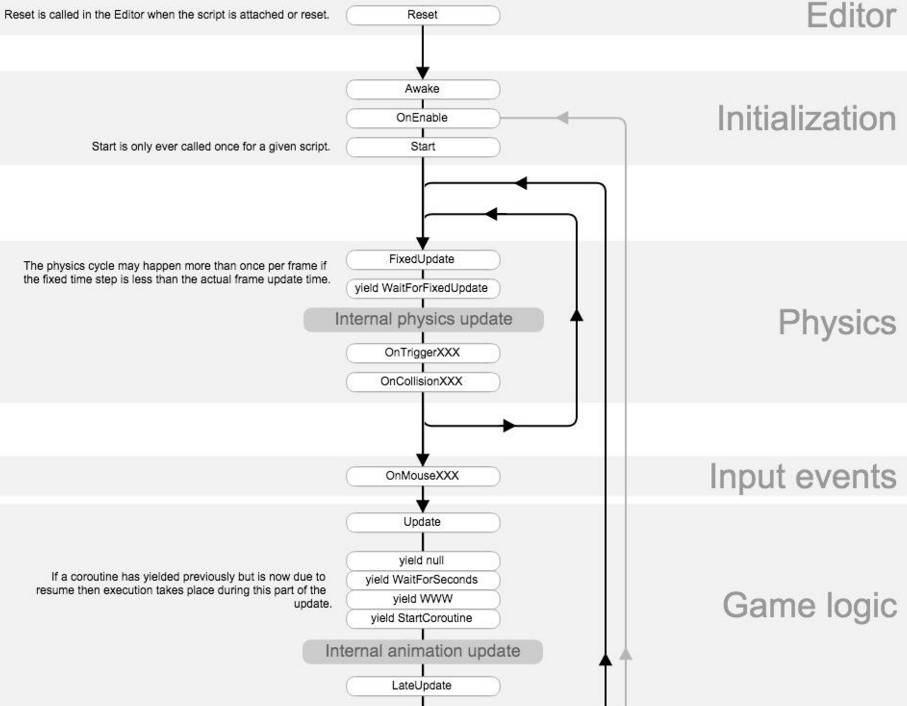
yield return new WaitForSeconds(1.0f);

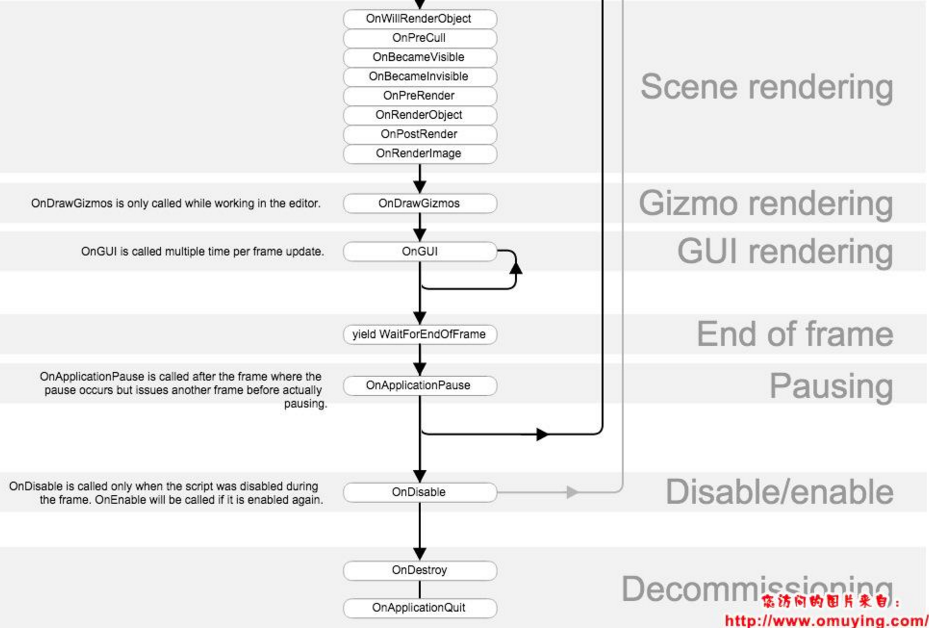
Debug.Log("This is LateUpdate Coroutine call end");

}

}







看上图协程位置

**FixedUpdate之后：**

Yield WaitForFixedUpdate

**Update之后：**

Yield null

Yield WaitForSeconds

Yield WWW

Yield StartCotoutine

**当前帧结束之后：**

Yeild WaitForEndOfFrame

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());

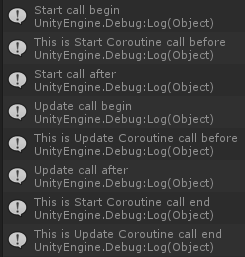
Debug.Log("Update call after");

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());

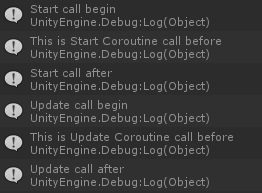
Debug.Log("Update call after");

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。

下面看一个例子：

void Start()

{

Debug.Log("Start framecount = " + Time.frameCount);

StartCoroutine(LoadResources());

}

void Update()

{

Debug.Log("Update framecount = " + Time.frameCount);

}

void LateUpdate()

{

Debug.Log("LateUpdate framecount = " + Time.frameCount);

}

string filePath = "file://";

IEnumerator LoadResources()

{

Debug.Log("11 framecount = " + Time.frameCount);

yield return null;

Debug.Log("22 framecount = " + Time.frameCount);

yield return 10;

Debug.Log("33 framecount = " + Time.frameCount);

string path = "www.baidu.com";

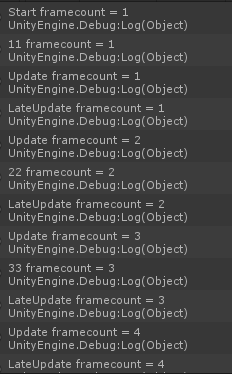
WWW www = new WWW(path);

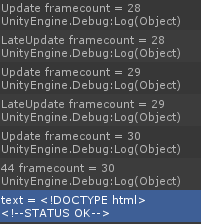
yield return www;

Debug.Log("44 framecount = " + Time.frameCount);

Debug.Log("text = " + www.text);

}





我们发现，在刚开始StartCoroutine时直接进入到了协程的第一个yield return处，然后再执行Update之后再执行协程，对于WWW会在下载完成之后执行后面的代码

查看4.3.1源码：

首先调用StartCoroutine，实际上是调用MonoBehaviour.cpp的StartCoroutine，然后调用InvokeMethodOrCoroutineChecked，接下来调用HandleCoroutineReturnValue，

实际上在StartCoroutine就是创建了一个协程（Coroutine）

Coroutine\* MonoBehaviour::HandleCoroutineReturnValue (ScriptingMethodPtr method, ScriptingObjectPtr returnValue)

{

if (!DoesMethodHaveIEnumeratorReturnType(method))

return NULL;

return **CreateCoroutine**(returnValue, method);

}

Coroutine\* MonoBehaviour::**CreateCoroutine**(ScriptingObjectPtr userCoroutine, ScriptingMethodPtr method)

{

ScriptingMethodPtr moveNext = scripting\_object\_get\_virtual\_method(userCoroutine, MONO\_COMMON.IEnumerator\_MoveNext, GetScriptingMethodRegistry());

#if !UNITY\_FLASH

ScriptingMethodPtr current = scripting\_object\_get\_virtual\_method(userCoroutine, MONO\_COMMON.IEnumerator\_Current, GetScriptingMethodRegistry());

#else

//todo: make flash use generic path. set a bogus value for flash here right now so it passes current != NULL check, flash path will never use this value for now.

ScriptingMethodPtr current = (ScriptingMethodPtr)1;

#endif

if (current == SCRIPTING\_NULL || moveNext == SCRIPTING\_NULL)

{

std::string message = (method != SCRIPTING\_NULL) ? Format ("Coroutine '%s' couldn't be started!", scripting\_method\_get\_name(method)) : "Coroutine couldn't be started!";

LogStringObject (message, this);

return NULL;

}

Coroutine\* coroutine = new Coroutine ();

coroutine->m\_CoroutineEnumeratorGCHandle = scripting\_gchandle\_new (userCoroutine);

coroutine->m\_CoroutineEnumerator = userCoroutine;

coroutine->m\_CoroutineMethod = method;

coroutine->SetMoveNextMethod(moveNext);

coroutine->SetCurrentMethod(current);

coroutine->m\_Behaviour = this;

coroutine->m\_ContinueWhenFinished = NULL;

coroutine->m\_WaitingFor = NULL;

coroutine->m\_AsyncOperation = NULL;

coroutine->m\_RefCount = 1;

coroutine->m\_IsReferencedByMono = 0;

#if DEBUG\_COROUTINE

printf\_console ("Allocate coroutine %d\n", coroutine);

AssertIf(GetDelayedCallManager().HasDelayedCall(coroutine->m\_Behaviour, Coroutine::ContinueCoroutine, CompareCoroutine, coroutine));

#endif

#if DEBUG\_COROUTINE\_LEAK

printf\_console ("Active coroutines %d\n", gCoroutineCounter);

gCoroutineCounter++;

#endif

**m\_ActiveCoroutines.push\_back (\*coroutine);**

AssertIf(&m\_ActiveCoroutines.back() != coroutine);

**m\_ActiveCoroutines.back ().Run (); // 这里会执行一次**

AssertIf(coroutine->m\_RefCount == 0);

if (coroutine->m\_RefCount <= 1)

{

Coroutine::CleanupCoroutine(coroutine);

return NULL;

}

Coroutine::CleanupCoroutine(coroutine);

return coroutine;

}

**Coroutine.cpp的Run方法：**

void Coroutine::Run ()

{

Assert(m\_RefCount != 0);

Assert(m\_Behaviour != NULL);

#if DEBUG\_COROUTINE

AssertIf(GetDelayedCallManager().HasDelayedCall(m\_Behaviour, Coroutine::ContinueCoroutine, CompareCoroutineMethodName, this));

if (m\_Behaviour == NULL)

{

printf\_console ("Coroutine fuckup %d refcount: %d behaviour%d\n", this, m\_RefCount, m\_Behaviour);

}

#endif

// - Call MoveNext (This processes the function until the next yield!)

// - Call Current (This returns condition when to continue the coroutine next.)

// -> Queue it based on the continue condition

// Temporarily increase refcount so the object will not get destroyed during the m\_MoveNext call

m\_RefCount++;

ScriptingExceptionPtr exception = NULL;

**bool keepLooping = InvokeMoveNext(&exception);** **// 调用一次moveNext方法**

AssertIf(m\_RefCount <= 0 || m\_RefCount > 10000000);

bool coroutineWasDestroyedDuringMoveNext = m\_RefCount == 1;

// Decrease temporary refcount so the object will not get destroyed during the m\_MoveNext call

CleanupCoroutine(this);

// The coroutine has been destroyed in the mean time, probably due to a call to StopAllCoroutines, stop executing further

if (coroutineWasDestroyedDuringMoveNext)

{

Assert(m\_ContinueWhenFinished == NULL);

return;

}

if (exception != NULL)

return;

// Are we done with this coroutine?

if (!keepLooping) **// 判断这个协程是否已经执行完毕**

{

// If there is a coroutine waiting for this one to finish Run it!

if (m\_ContinueWhenFinished) **// 这个协程是否其他协程的子协程，如果是，就执行其他协程**

{

AssertIf (this != m\_ContinueWhenFinished->m\_WaitingFor);

Coroutine\* continueWhenFinished = m\_ContinueWhenFinished;

m\_ContinueWhenFinished->m\_WaitingFor = NULL;

m\_ContinueWhenFinished = NULL;

// The coroutine might have been stopped inside of the last coroutine invokation

if (continueWhenFinished->m\_Behaviour)

continueWhenFinished->Run ();

CleanupCoroutine (continueWhenFinished);

}

return;

}

if (m\_Behaviour == NULL)

return;

**ProcessCoroutineCurrent(); // 处理当前协程**

}

void Coroutine::**ProcessCoroutineCurrent**()

{

ScriptingExceptionPtr exception = NULL;

#if !UNITY\_FLASH

ScriptingInvocation invocation(m\_Current);

invocation.object = m\_CoroutineEnumerator;

invocation.objectInstanceIDContextForException = m\_Behaviour->GetInstanceID();

invocation.classContextForProfiler = m\_Behaviour->GetClass();

ScriptingObjectPtr monoWait = invocation.Invoke(&exception);

#else

ScriptingObjectPtr monoWait = Ext\_Flash\_getProperty(m\_CoroutineEnumerator,"IEnumerator\_Current");

#endif

AssertIf(m\_RefCount <= 0 || m\_RefCount > 10000000);

if (exception != NULL)

return;

if (monoWait == SCRIPTING\_NULL)

{

m\_RefCount++;

CallDelayed (ContinueCoroutine, m\_Behaviour, 0.0F, this, 0.0F, CleanupCoroutine, DelayedCallManager::kRunDynamicFrameRate | DelayedCallManager::kWaitForNextFrame);

return;

}

**HandleIEnumerableCurrentReturnValue**(monoWait); **// 真正处理协程的地方**

}

void Coroutine::**HandleIEnumerableCurrentReturnValue**(ScriptingObjectPtr monoWait)

{

AsyncOperation\* async = NULL;

ScriptingClassPtr waitClass = scripting\_object\_get\_class (monoWait, GetScriptingTypeRegistry());

const CommonScriptingClasses& classes = GetMonoManager ().GetCommonClasses ();

// Continue the coroutine in 'wait' seconds

if (scripting\_class\_is\_subclass\_of (waitClass, classes.waitForSeconds)) **// WaitForSeconds**

{

m\_RefCount++;

float wait;

MarshallManagedStructIntoNative(monoWait,&wait);

**CallDelayed** (ContinueCoroutine, m\_Behaviour, wait, this, 0.0F, CleanupCoroutine, DelayedCallManager::kRunDynamicFrameRate | DelayedCallManager::kWaitForNextFrame); **// 设置时间以及mask，同时注意ContinueCoroutine方法，它是Coroutine的一个静态方法，会执行Couroutine.Run方法，也就是进行下一个MoveNext方法，将Current方法到集合中，实际上是一个multiset容器**

return;

}

// Continue the coroutine on the next fixed update

if (scripting\_class\_is\_subclass\_of (waitClass, classes.waitForFixedUpdate))

{

m\_RefCount++;

CallDelayed (ContinueCoroutine, m\_Behaviour, 0.0F, this, 0.0F, CleanupCoroutine, DelayedCallManager::kRunFixedFrameRate);

return;

}

// Continue the coroutine at the end of frame

if (scripting\_class\_is\_subclass\_of (waitClass, classes.waitForEndOfFrame))

{

m\_RefCount++;

CallDelayed (ContinueCoroutine, m\_Behaviour, 0.0F, this, 0.0F, CleanupCoroutine, DelayedCallManager::kEndOfFrame);

return;

}

// Continue after another coroutine is finished

if (scripting\_class\_is\_subclass\_of (waitClass, classes.coroutine))

{

Coroutine\* waitForCoroutine;

MarshallManagedStructIntoNative(monoWait,&waitForCoroutine);

if (waitForCoroutine->m\_DoneRunning)

{

// continue executing.

ContinueCoroutine(m\_Behaviour, this);

return;

}

if (waitForCoroutine->m\_ContinueWhenFinished != NULL)

{

LogStringObject ("Another coroutine is already waiting for this coroutine!\nCurrently only one coroutine can wait for another coroutine!", m\_Behaviour);

return;

}

m\_RefCount++;

waitForCoroutine->m\_ContinueWhenFinished = this;

m\_WaitingFor = waitForCoroutine;

return;

}

#if ENABLE\_WWW

// Continue after fetching an www object is done

if (classes.www && scripting\_class\_is\_subclass\_of (waitClass, classes.www ))

{

WWW\* wwwptr;

MarshallManagedStructIntoNative(monoWait,&wwwptr);

if(wwwptr != NULL)

{

m\_RefCount++;

**wwwptr->CallWhenDone (ContinueCoroutine, m\_Behaviour, this, CleanupCoroutine); // www表现有所不同，但本质上也是调用了CallDelayed**

}

return;

}

#endif

// Continue after fetching an www object is done

if ((scripting\_class\_is\_subclass\_of (waitClass, classes.asyncOperation)) && (async = ScriptingObjectWithIntPtrField<AsyncOperation> (monoWait).GetPtr()) != NULL)

{

m\_RefCount++;

if (async->IsDone())

{

CallDelayed (ContinueCoroutine, m\_Behaviour, 0.0F, this, 0.0F, CleanupCoroutine, DelayedCallManager::kRunDynamicFrameRate | DelayedCallManager::kRunDynamicFrameRate | DelayedCallManager::kWaitForNextFrame);

return;

}

// Use AysncOperation ContinueCoroutine - default path

if (async->HasCoroutineCallback ())

{

////@TODO: Throw exception?

ErrorString("This asynchronous operation is already being yielded from another coroutine. An asynchronous operation can only be yielded once.");

CallDelayed (ContinueCoroutine, m\_Behaviour, 0.0F, this, 0.0F, CleanupCoroutine, DelayedCallManager::kRunDynamicFrameRate | DelayedCallManager::kRunDynamicFrameRate | DelayedCallManager::kWaitForNextFrame);

return;

}

async->SetCoroutineCallback(ContinueCoroutine, m\_Behaviour, this, CleanupCoroutine);

m\_AsyncOperation = async;

m\_AsyncOperation->Retain();

return;

}

// Continue the coroutine on the next dynamic frame update

m\_RefCount++;

CallDelayed (ContinueCoroutine, m\_Behaviour, 0.0F, this, 0.0F, CleanupCoroutine, DelayedCallManager::kRunDynamicFrameRate | DelayedCallManager::kRunDynamicFrameRate | DelayedCallManager::kWaitForNextFrame);

//Ext\_MarshalMap\_Release\_ScriptingObject(monoWait);//RH TODO : RELEASE THE MONOWAIT OBJECTS SOMEWHERE

}

**下面来看看CallDelayed方法和wwwptr->CallWhenDone方法：**

**1 CallDelayed方法**（实际将IEnumerator中的Current封装成了一个CallBack，并放到一个multiset容器中，这里的fun为Coroutine::ContinueCoroutine，Coroutine::ContinueCoroutine本质上是继续执行了一次MoveNext方法，userdata为Coroutine本身）

void CallDelayed (DelayedCall \*func, PPtr<Object> o, float time, void\* userData, float repeatRate, CleanupUserData\* cleanup, int mode)

{

DelayedCallManager::Callback callback;

if (time == 0.0F)

time = -1.0F;

callback.time = time + GetCurTime ();

callback.frame = -1;

if (mode & DelayedCallManager::kWaitForNextFrame)

callback.frame = GetTimeManager().GetFrameCount() + 1;

callback.repeatRate = repeatRate;

callback.repeat = repeatRate != 0.0F;

AssertIf (callback.repeat && repeatRate < 0.00001F && (mode & DelayedCallManager::kWaitForNextFrame) == 0);

callback.userData = userData;

callback.call = func;

callback.cleanup = cleanup;

callback.object = o;

callback.mode = mode;

callback.timeStamp = GetDelayedCallManager ().m\_TimeStamp;

GetDelayedCallManager ().m\_CallObjects.insert (callback);

}

**2 CallWhenDone方法**

void WWW::CallWhenDone(DelayedCall\* func, Object\* o, void\* userData, CleanupUserData\* cleanup) {

WWWDelayCall\* delayCall = new WWWDelayCall(this, func, o, userData, cleanup); **// 这里将Coroutine::ContinueCoroutine方法赋给了m\_fun，将WWW本身赋给了m\_wait\_for**

CallDelayed(WWWDelayCall::Callback, o, 0.0F, (void\*)delayCall, -1.0F, WWWDelayCall::Cleanup, DelayedCallManager::kRunDynamicFrameRate | DelayedCallManager::kWaitForNextFrame);

}

可以看到CallWhenDone实际上也是在multiset中放了一个CallBack，只不过其CallBack并不是ContinueCoroutine，而是WWWDelayCall的callback，同时userdata为delayCall

看看WWWDelayCall的callback方法：

void WWWDelayCall::Callback(Object\* o, void\* userData) {

WWWDelayCall\* delayCall = (WWWDelayCall\*) userData;

AssertIf(delayCall->m\_o != o);

**// 这里会每帧都调用，会判断WWW（另外一个线程）的IsDone方法时候执行完毕，如果执行完毕，则实行m\_func方法（Couroutine::ContinueCoroutine方法，本质上就是继续调用MoveNext方法）**

if(delayCall->m\_wait\_for == NULL || delayCall->m\_wait\_for->IsDone()) {

delayCall->m\_func(o, delayCall->m\_userData);

GetDelayedCallManager ().CancelCallDelayed(o,WWWDelayCall::Callback, WWWDelayCall::MatchForCancel, userData);

}

}

**现在可以看到每次Run就执行当前的IEnumerator的MoveNext方法，然后将IEnumerator的Current放到DelayedCallManager的multiset中，下面来看看主循环（Player.playerloop方法）**

bool PlayerLoop (bool batchMode, bool performRendering, IHookEvent\* pHookEvt)

{

#if !UNITY\_FLASH

ReentrancyChecker checker( &s\_InsidePlayerLoop );

if (!checker.IsOK())

{

ErrorString ("PlayerLoop called recursively!");

return true;

}

#endif

#if SUPPORT\_LOG\_ORDER\_TRACE

if (RunningReproduction())

{

LogString(Format("Frame %d", GetTimeManager().GetFrameCount()));

}

#endif

#if ENABLE\_CLUSTER\_SYNC

if(GetIClusterRenderer())

GetIClusterRenderer()->SynchronizeCluster();

#endif

#if ENABLE\_PLAYERCONNECTION && !UNITY\_EDITOR

PlayerConnection::Get().Poll();

#endif

#if ENABLE\_PROFILER && GAMERELEASE

UnityProfiler::RecordPreviousFrame(kProfilerGame);

bool profilerEnabled = UnityProfiler::StartNewFrame(kProfilerGame);

if (profilerEnabled)

{

GfxDevice& device = GetGfxDevice();

device.BeginFrameStats();

}

#endif // ENABLE\_PROFILER && GAMERELEASE

GPU\_TIMESTAMP(); // Initial GPU time

#if ENABLE\_SUBSTANCE

{

SubstanceSystem\* system = GetSubstanceSystemPtr ();

if (system != NULL)

{

PROFILER\_AUTO(gSubstanceUpdate, NULL)

system->Update();

}

}

#endif

;;FrameStats::Timestamp ts = 0;

#if SUPPORT\_REPRODUCE\_LOG

static bool didSetWebplayerSize = false;

if (!didSetWebplayerSize)

{

didSetWebplayerSize = true;

WriteWebplayerSize(GetScreenManager().GetWidth(), GetScreenManager().GetHeight());

}

#endif // SUPPORT\_REPRODUCE\_LOG

if (!batchMode)

ProcessMouseInWindow();

// In the editor, clear intermediate renderers before loop. So that in paused state or when resizing windows,

// we can still draw the previous ones.

#if UNITY\_EDITOR

GetScene().ClearIntermediateRenderers();

#endif

ClearLines();

GfxDevice& device = GetGfxDevice();

GfxDevice::PresentMode presentMode = device.GetPresentMode();

bool playmode = IsWorldPlaying();

#if !UNITY\_EDITOR

if (!batchMode)

PresentBeforeUpdate(presentMode);

#endif

// Reset frame stats after present (case 496221)

if (presentMode == GfxDevice::kPresentBeforeUpdate)

device.ResetFrameStats();

#if SUPPORT\_REPRODUCE\_LOG

RepeatReproductionScreenshot();

ReadWriteReproductionTime();

#endif

#if ENABLE\_CLUSTER\_SYNC

if(!GetIClusterRenderer() || GetIClusterRenderer()->IsMasterOfCluster())

#endif

{

PlayerUpdateTime();

}

#if ENABLE\_WWW

{

PROFILER\_AUTO(gUpdateWebStream, NULL);

UnityWebStream::UpdateAllUnityWebStreams();

}

#endif

////@TODO: CLeanup code where input is processed after level loading

/// All input should be processed prior to level loading

// In reproduce mode make sure to handle input postprocessor prior to loading a level.

// This fixes potential issues where scripts use input in awake from load

#if SUPPORT\_REPRODUCE\_LOG

if (RunningReproduction())

{

if (playmode)

GetInputManager ().ProcessInput ();

ReadWriteReproductionInput();

}

#endif

#if SUPPORT\_REPRODUCE\_LOG

if (RunningReproduction())

GetPreloadManager().WaitForAllAsyncOperationsToComplete();

#endif

{

PROFILER\_AUTO(gUpdatePreloading, NULL)

GetPreloadManager().UpdatePreloading ();

}

if (performRendering)

GetScene().NotifyInvisible();

{

PROFILER\_AUTO(gPlayerCleanupCachedData, NULL)

GetRenderBufferManager().GarbageCollect();

TextMeshGenerator2::GarbageCollect();

}

////@TODO: CLeanup code where input is processed after level loading

/// All input should be processed prior to level loading

#if SUPPORT\_REPRODUCE\_LOG

if (!RunningReproduction())

#endif

{

if (playmode)

GetInputManager ().ProcessInput ();

}

ProcessRemoteInput ();

;;ts = GetTimestamp();

**GetDelayedCallManager ().Update (DelayedCallManager::kRunStartupFrame); // 执行kRunStarupFrame类型的CallBack**

;;gFrameStats.coroutineDt += GetTimestamp() - ts;

int stepCounter = 0;

// Make sure collider positions update when time scale is zero, so raycasts will still work as expected.

if (CompareApproximately (GetTimeManager().GetTimeScale(), 0.0f))

{

CALL\_UPDATE\_MODULAR(PhysicsRefreshWhenPaused)

}

#if UNITY\_XENON

xenon::Kinect::Update(xenon::Kinect::kUT\_BeforeUserScripts);

#endif

#if UNITY\_IPHONE

iphone::DeliverPlatformEvents();

#endif

// Fixed framerate loop (fixed behaviours, dynamics, delayed calling)

while (GetTimeManager ().StepFixedTime ())

{

// Placed here so we ensure it is also called in edit-mode (fix for case 379024: pressing stop did not properly clear fixedStepLines)

ClearLines ();

if (playmode)

{

if (stepCounter == 0)

{

CALL\_UPDATE\_MODULAR(PhysicsResetInterpolatedTransformPosition)

CALL\_UPDATE\_MODULAR(Physics2DResetInterpolatedTransformPosition)

}

#if ENABLE\_AUDIO

IAudio\* audioModule = GetIAudio();

if (audioModule)

audioModule->FixedUpdate ();

#endif

// Script.FixedUpdate

;;ts = GetTimestamp();

**GetFixedBehaviourManager ().Update ();// FixedUpdate方法**

;;gFrameStats.fixedBehaviourManagerDt += (GetTimestamp() - ts);

// Animation (Root motion)

;;ts = GetTimestamp();

CALL\_UPDATE\_MODULAR(AnimatorFixedUpdateFKMove)

CALL\_UPDATE\_MODULAR(LegacyAnimationUpdate)

;;gFrameStats.animationUpdateDt += (GetTimestamp() - ts);

// Physics

;;ts = GetTimestamp();

CALL\_UPDATE\_MODULAR(PhysicsFixedUpdate)

CALL\_UPDATE\_MODULAR(Physics2DFixedUpdate)

;;gFrameStats.fixedPhysicsManagerDt += GetTimestamp() - ts;

// Animation IK and write bones

CALL\_UPDATE\_MODULAR(AnimatorFixedUpdateRetargetIKWrite)

// Script Coroutines

;;ts = GetTimestamp();

**GetDelayedCallManager ().Update (DelayedCallManager::kRunFixedFrameRate); // 执行kRunFixedFrameRate类型的CallBack**

;;gFrameStats.coroutineDt += GetTimestamp() - ts;

}

stepCounter++;

}

gFrameStats.fixedUpdateCount = stepCounter;

// Dynamics, animation, behaviours

if (playmode)

{

CALL\_UPDATE\_MODULAR(PhysicsUpdate)

CALL\_UPDATE\_MODULAR(Physics2DUpdate)

}

bool oldTextFocus = GetInputManager().GetTextFieldInput();

#if ENABLE\_UNITYGUI

GetGUIManager().SendQueuedEvents ();

#endif

#if ENABLE\_NEW\_EVENT\_SYSTEM

#if ENABLE\_UNITYGUI

if (playmode && !GetGUIManager().GetMouseUsed())

#else

if (playmode)

#endif

GetInputManager().SendInputEvents();

#endif

#if ENABLE\_RETAINEDGUI

GetGUITracker().SendInputEvents(GetGUIManager().GetQueuedEvents());

#endif

#if ENABLE\_SCRIPTING && SUPPORT\_MOUSE

#if ENABLE\_NEW\_EVENT\_SYSTEM

// Deprecated in Unity 4.1. Mouse and touch events will now be sent via C++. Look inside InputManager.cpp.

if (playmode && !IS\_CONTENT\_NEWER\_OR\_SAME(kUnityVersion4\_1\_a1))

#else

if (playmode)

#endif

{

if (GetBuildSettings().usesOnMouseEvents)

{

PROFILER\_AUTO(gOnMouseHandlers, NULL)

int mouseUsed = GetGUIManager().GetMouseUsed () ? 1 : 0;

int skipRTCameras = IS\_CONTENT\_NEWER\_OR\_SAME(kUnityVersion4\_0\_a1);

ScriptingInvocation invocation(MONO\_COMMON.doSendMouseEvents);

invocation.AddInt(mouseUsed);

invocation.AddInt(skipRTCameras);

invocation.Invoke();

}

}

#endif

CALL\_UPDATE\_MODULAR(NavMeshUpdate)

;;ts = GetTimestamp();

**GetBehaviourManager ().Update (); // 执行Update方法**

;;gFrameStats.dynamicBehaviourManagerDt += GetTimestamp() - ts;

;;ts = GetTimestamp();

**GetDelayedCallManager ().Update (DelayedCallManager::kRunDynamicFrameRate); // 执行kRunDynamicFrameRate类型的CallBack**

;;gFrameStats.coroutineDt += GetTimestamp() - ts;

// Dynamic Step Animation Update

if (playmode)

{

;;ts = GetTimestamp();

CALL\_UPDATE\_MODULAR(AnimatorUpdateFKMove);

CALL\_UPDATE\_MODULAR(LegacyAnimationUpdate);

CALL\_UPDATE\_MODULAR(AnimatorUpdateRetargetIKWrite);

;;gFrameStats.animationUpdateDt += (GetTimestamp() - ts);

}

#if ENABLE\_NETWORK

if (playmode)

{

CALL\_MANAGER\_IF\_EXISTS(ManagerContext::kNetworkManager, NetworkUpdate());

CALL\_MANAGER\_IF\_EXISTS(ManagerContext::kMasterServerInterface, NetworkUpdate());

}

#endif

ParticleSystem::BeginUpdateAll ();

;;ts = GetTimestamp();

**GetLateBehaviourManager ().Update (); // 执行LaterUpdate方法**

;;gFrameStats.dynamicBehaviourManagerDt += GetTimestamp() - ts;

;;ts = GetTimestamp();

**GetDelayedCallManager ().Update (DelayedCallManager::kRunDynamicFrameRate); 执行kRunDynamicFrameRate类型的CallBack**

;;gFrameStats.coroutineDt += GetTimestamp() - ts;

#if ENABLE\_AUDIO

IAudio\* audioModule = GetIAudio();

if (audioModule)

audioModule->Update();

#endif

ParticleEmitter::UpdateAllParticleSystems();

#if ENABLE\_PHYSICS

if (playmode && performRendering && device.IsValidState())

{

CALL\_UPDATE\_MODULAR(PhysicsSkinnedClothUpdate);

}

#endif

// We need to sync particle systems here to make sure they update their renderers properly

ParticleSystem::EndUpdateAll ();

RenderManager::UpdateAllRenderers();

if(pHookEvt)

pHookEvt->Execute();

;;ts = GetTimestamp();

if (performRendering && device.IsValidState()) // skinning touches graphics resources; only do that when device is not lost

{

SkinnedMeshRenderer::UpdateAllSkinnedMeshes(SkinnedMeshRenderer::kUpdateNonCloth);

}

;;gFrameStats.skinMeshUpdateDt += GetTimestamp() - ts;

#if UNITY\_IPHONE || UNITY\_ANDROID || UNITY\_BB10 || UNITY\_TIZEN

WebScripting::Get().ProcessSendMessages();

#endif

GetUpdateManager ().Update ();

if(pHookEvt)

pHookEvt->Execute();

#if ENABLE\_MOVIES || ENABLE\_WEBCAM

if (performRendering && device.IsValidState()) // movies touch graphics resources; only do that when device is not lost

{

IAudio\* audioModule = GetIAudio();

if (audioModule)

audioModule->UpdateVideoTextures ();

}

#endif

#if UNITY\_WII && !WIIWARE

void ProcessDVDErrors();

ProcessDVDErrors();

#endif

// In d3d ref mode only render when taking screenshots

#if SUPPORT\_REPRODUCE\_LOG && GFX\_SUPPORTS\_D3D9 && ENABLE\_FORCE\_GFX\_RENDERER

if (::g\_ForceD3D9RefDevice && RunningReproduction() && !GetInputManager().GetKey(SDLK\_F5))

performRendering = false;

#endif

#if !UNITY\_EDITOR

if (performRendering)

{

;;ts = GetTimestamp();

// Perform rendering

if (!batchMode)

PlayerRender(false);

// We want to send 'end-of-frame' events even if running in batch mode

**PlayerSendFrameComplete(); // 执行kEndOfFrame类型的CallBack**

PlayerTakeScreenShots();

if (!batchMode)

{

device.EndFrame();

PresentAfterDraw(presentMode);

}

;;gFrameStats.renderDt += GetTimestamp() - ts;

} // End rendering

// In the player, clear intermediate renderers after loop.

GetScene().ClearIntermediateRenderers();

#endif // !UNITY\_EDITOR

#if SUPPORT\_REPRODUCE\_LOG

ReadWriteReproductionEnd();

#endif

GetScreenManager().SetRequestedResolution();

// Clear the input string and the keydown events at the end of the Loop.

// This makes sure all input string is cleared

GetInputManager ().InputEndFrame ();

// Let Fonts uncache characters if they are not used in the next frame.

Font::FrameComplete();

// We entered Text Field input this frame, Game mode input is disable.

// Reset axes, so they don't stick.

if(GetInputManager().GetTextFieldInput() && !oldTextFocus)

GetInputManager ().ResetInputAxes ();

#if WEBPLUG

if (!LoadQueuedWebPlayerData())

return false;

#endif

#if THREADED\_LOADING\_DEBUG

static double endFrameTime = 0.0F;

float delta = GetTimeSinceStartup() - endFrameTime;

endFrameTime = GetTimeSinceStartup();

printf\_console("FPS %f\n", 1.0F / delta);

#endif

#if ENABLE\_PROFILER && GAMERELEASE

if (profilerEnabled)

{

device.EndFrameStats();

device.SynchronizeStats();

}

#endif

#if ENABLE\_MEMORY\_MANAGER

GetMemoryManager().FrameMaintenance();

#endif

#if ENABLE\_GAMECENTER

GameCenter::ExecuteGameCenterCallbacks();

#endif

// Reset frame stats after present and UnityProfiler EndFrame() (case 494732)

if (presentMode == GfxDevice::kPresentAfterDraw)

device.ResetFrameStats();

return true;

}

void **PlayerSendFrameComplete**()

{

GetDelayedCallManager ().Update (DelayedCallManager::kEndOfFrame);

}

现在看看DelayedCallManager的Update方法：

// Call all delayed functions when the time has come.

void DelayedCallManager::Update (int modeMask)

{

// PROFILER\_AUTO(gDelayedCallProfile, NULL)

// For robustness we are using a iterator that is stored in the manager and when Remove is called

// it makes sure that the iterator is set to the next element so that we never end up with a stale ptr

float time = GetCurTime();

int frame = GetTimeManager().GetFrameCount();

Container::iterator i = m\_CallObjects.begin ();

m\_TimeStamp++;

while (i != m\_CallObjects.end () && i->time <= time) **// 这里通过i->time与time的比较就可以执行WaitForSeconds**

{

m\_NextIterator = i; m\_NextIterator++;

Callback &cb = const\_cast<Callback&> (\*i);

//- Make sure the mask matches.

//- We never execute delayed calls that are added during the DelayedCallManager::Update function

if ((cb.mode & modeMask) && cb.timeStamp != m\_TimeStamp && cb.frame <= frame) **// 掩码相同，并通过cb.frame与frame的比较就可以将其他在再下一帧执行**

{

// avoid loading stuff from persistent manager in the middle of async loading

Object \*o = Object::IDToPointer (cb.object.GetInstanceID ());

if (o)

{

void\* userData = cb.userData;

DelayedCall\* callback = cb.call;

// Cleanup and Removal is a bit hard

// Problems are

// - CancelCall might be called from inside the callback so the best way is to remove the callback structure before.

// - of course we still need the user data to be deallocated calling the callback not before

if (!cb.repeat)

{

// Remove callback structure from set

CleanupUserData\* cleanup = cb.cleanup;

RemoveNoCleanup (cb, i);

//call callback

**callback (o, userData); // 这里执行Coroutine::ContinueCoroutine方法，但是对于WWW而言是执行WWWDelayCall的callback方法**

//afterwards cleanup userdata

if (cleanup && userData)

cleanup (userData);

}

else

{

// Advance time and reinsert (We dont call the cleanup function because we are repeating the call.

// It can only be canceleld by CancelDelayCall)

cb.time += cb.repeatRate;

if (cb.mode & DelayedCallManager::kWaitForNextFrame)

cb.frame = GetTimeManager().GetFrameCount() + 1;

m\_CallObjects.insert (cb);

RemoveNoCleanup (cb, i);

// call callback

callback (o, userData);

}

}

else

Remove (cb, i);

}

i = m\_NextIterator;

}

}

总结：

所谓的yield return仅仅只是创建了一个IEnumerator，这里面有MoveNext方法和Current对象，至于如果所延时就看什么时候调用MoveNext方法

1， C#层调用**StartCoroutine**方法，将IEnumerator对象（或者是用于创建IEnumerator对象的方法名字符串）传入C++层，创建出一个对应的Coroutine对象，之后调用这个Coroutine对象的Run()方法，也就是会第一次会执行到第一个yield return处

2， 在**Coroutine.Run()**中，首先通过mono的反射功能，找到IEnumerator上的MoveNext()、get\_Current()两个方法，然后调用一次**MoveNext()**；如果MoveNext()返回false，表示Coroutine执行结束，；如果返回true，表示Coroutine执行到了一句yield return处；这时就需要调用**get\_Current()**取出yield return返回的对象（monoWait），再根据monoWait的具体类型（null、WaitForSeconds、WaitForFixedUpdate等），将Coroutine对象保存到**DelayedCallManager的callback列表**中；至此，Coroutine在当前帧的执行即结束；

3， 之后游戏运行过程中，游戏主循环的**PlayerLoop()**方法会在每帧的不同时间点以不同的modeMask调用**DelayedCallManager.Update**方法（注：PlayerLoop中的具体过程以及调用Coroutine时间点，可以参考：<http://docs.unity3d.com/uploads/Main/monobehaviour_flowchart.svg>）；Update方法中会遍历callback列表中的Coroutine对象；如果某个Coroutine对象的monoWait的执行条件满足，则将其从callback列表中取出，执行这个Coroutine对象的Run()方法，回到2的执行流程中。

注意上面的流程中Coroutine和WWW有点特别：

对于Coroutine实际上相当于一个链表结构，当某个Coroutine执行完毕后，然后调用等待Coroutine执行完毕后的Coroutine的Run方法，实际上也就是执行m\_WhenCallFinished

对于WWW，其他的协程对象都是执行Coroutine::ContinueCoroutine，但WWW执行的是WWWDelayedCall.callback方法，在这里面实际上也封装了Coroutine::ContinueCoroutine方法，它是一个新线程，每一帧都会调用WWWDelayedCall.callback方法，但只有在WWW的IsDone执行完毕后才会调用Coroutine::ContinueCoroutine方法，也就是调用MoveNext方法