博客 登录 | 注册

jasonwang18的博客

目录视图 摘要视图 RSS 订阅

【观点】人工智能会不会取代开发它的人？ CSDN日报20170410 ——《未经检视的人生不值得活》 【福利】微分享：大数据入门技术初探 博客搬家，有礼相送

Unity3d IEnumerator 协程的理解

标签： unity3d IEnumerator

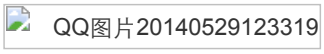
2017-02-17 18:35 228人阅读 评论(0) 收藏 举报

分类： Unity3D学习

目录(?) [+]

由于VR的关系，第一次接触到了Unity3D的项目，对C#Script一些语法不是很了解，特别是IEnumerator yield，在项目中大量被使用，下面谈谈对它们的理解，文章转自

作者：王选易，出处： http://www.cnblogs.com/neverdie/ 欢迎转载 ，也请保留这段声明。如果你喜欢这篇文章，请点【推荐】。谢谢！



为什么需要协程

在游戏中有许多过程（Process）需要花费多个逻辑帧去计算。

- 你会遇到“密集”的流程，比如说寻路，寻路计算量非常大，所以我们通常会把它分割到不同的逻辑帧去进行计算，以免影响游戏的帧率。
- 你会遇到“稀疏”的流程，比如说游戏中的触发器，这种触发器大多数时候什么也不做，但是一旦被调用会做非常重要的事情（比图说游戏中自动开启的门就是在门前放了一个Empty Object作为trigger，人到门前就会触发事件）。

不管什么时候，如果你想创建一个能够历经多个逻辑帧的流程，但是却不使用多线程，那你就需要把一个任务来分割成多个任务，然后在下一帧继续执行这个任务。

比如，A*算法是一个拥有主循环的算法，它拥有一个open list来记录它没有处理到的节点，那么我们为了不影响帧率，可以让A*算法在每个逻辑帧中只处理open list中一部分节点，来保证帧率不被影响（这种做法叫做time slicing）。

再比如，我们在处理网络传输问题时，经常需要处理异步传输，需要等文件下载完毕之后再执行其他任务，一般我们使用回调来解决这个问题，但是Unity使用

如下边的程序：

```
private IEnumerator Test()
{
    WWW www = new WWW(ASSEST_URL);
    yield return www;
    AssetBundle bundle = www.assetBundle;
}
```

协程是什么

个人资料

疾风细语

+ 加关注 发私信

访问： 2357次
积分： 357
等级： BLOG 2
排名： 千里之外

原创： 30篇
转载： 4篇
译文： 0篇
评论： 2条

文章搜索

文章分类

Android源码分析 (13)
Java进阶 (2)
Android 总结 (17)
Android ndk编译 (5)
Unity3D学习 (1)
Android高性能编码 (6)
Kotlin学习 (0)
Android 框架 (2)

文章存档

2017年04月 (1)
2017年03月 (16)
2017年02月 (18)
2016年12月 (1)

阅读排行

Unity3d IEnumerator 协程 (228)
Android Studio使用新的 (225)
Android 6.0修改系统权限 (163)

快速回复 我要收藏

关闭

Android 如何获取已连接 (150)
Android如何监听第三方 (133)
Android 6.0 修改系统权限 (122)
RESTful API的理解 (115)
Mac自定义Android ndk工 (108)
Android synchronized的 (97)
Android targetSdkVersic (96)

评论排行
Android Studio使用新的 (2)
Android pluginMVPM 可 (0)
使用自定义工具链toolch (0)
Mac自定义Android ndk工 (0)
Android Studio使用新的 (0)
Android Studio使用新的 (0)
Android targetSdkVersic (0)
源码分析Android bindSe (0)
Android 6.0 修改系统权限 (0)
Android 6.0修改系统权限 (0)

推荐文章
* 【《Real-Time Rendering 3rd》 提炼总结】(一) 全书知识点总览
* CSDN日报20170409 ——《扯蛋的密码规则》
* Shader2D: 一些2D效果的Shader实现
* 一个屌丝程序猿的人生 (六十一)
* 自定义控件三部曲视图篇（三）——瀑布流容器WaterFallLayout实现
* 面向服务的体系架构 (SOA) 一架构篇

最新评论
Android Studio使用新的Gradle构 疾风细语: ABI_PLATFORM = minSdkVersion这个好难发现
Android Studio使用新的Gradle构 疾风细语: API_PLATFORM = minSdkVersion这个好难发现

从程序结构的角度来讲，协程是一个有限状态机，这样说可能并不是很明白，说到协程（Coroutine），我们还要提到另一样东西，那就是子例程（Subroutine），子例程一般可以指函数，函数是没有 状态 的，等到它return之后，它的所有局部变量就消失了，但是在协程中我们可以在 一个函数里多次返回， 局部变量被当作状态保存在协程函数中，知道最后一次return，协程的状态才别清除。

简单来说，协程就是：你可以写一段顺序的代码，然后标明哪里需要暂停，然后在下一帧或者一段时间后，系统会继续执行这段代码。

协程怎么用?

一个简单的C#代码，如下：

```
IEnumerator LongComputation()
{
    while(someCondition)
    {
        /* 做一系列的工作 */

        // 在这里暂停然后在下一帧继续执行

        yield return null;
    }
}
```

快速回复
我要收藏

协程是怎么工作的

注意上边的代码示例，你会发现一个协程函数的返回值是IEnumerator，它是一个迭代器，你可以把它当成指向一个序列的某个节点的指针，它提供了两个重要的接口，分别是Current（返回当前指向的元素）和MoveNext()（将指针向前移动一个单位，如果移动成功，则返回true）。IEnumerator是一个interface，所以你不用担心的具体实现。

通常，如果你想实现一个接口，你可以写一个类，实现成员，等等。 迭代器块（iterator block） 是一个方便的方式实现IEnumerator没有任何麻烦-你只是遵循一些规则，并实现IEnumerator由编译器自动生成。

一个迭代器块具备如下特征：

1. 返回IEnumerator
2. 使用yield关键字

所以yield关键词是干啥的？它声明序列中的下一个值或者是一个无意义的值。如果使用yield x（x是指一个具体的对象或数值）的话，那么movenext返回为true并且current被赋值为x，如果使用yield break使得movenext()返回false。

那么我举例如下，这是一个迭代器块：

```
public void Consumer()
{
        foreach(int i in Integers())
        {
                Console.WriteLine(i.ToString());
        }
}

public IEnumerable<int> Integers()
{
        yield return 1;
        yield return 2;
        yield return 4;
        yield return 8;
        yield return 16;
        yield return 16777216;
}
```

关闭


注意上文在迭代的过程中，你会发现，在两个yield之间的代码只有执行完毕之后，才会执行下一个yield，在Unity中，我们正是利用了这一点，我们可以写出下面这样的代码作为一个迭代器块：

```
IEnumerator TellMeASecret(){
  PlayAnimation("LeanInConspiratorially");
  while(playingAnimation)
    yield return null;

  Say("I stole the cookie from the cookie jar!");
  while(speaking)
    yield return null;

  PlayAnimation("LeanOutRelieved");
  while(playingAnimation)
    yield return null;
}
```

然后我们可以使用下文这样的客户代码，来调用上文的程序，就可以实现延时的效果。

```
IEnumerator e = TellMeASecret();
```

```
while(e.MoveNext()) {
    // do whatever you like
}
```

快速回复
我要收藏

协程是如何实现延时的?

如你所见，yield return返回的值并不一定是有意义的，如null，但是我们更感兴趣的是，如何使用这个yield return的返回值来实现一些有趣的效果。

Unity声明了YieldInstruction来作为所有返回值的基类，并且提供了几种常用的继承类，如WaitForSeconds（暂停一段时间继续执行），WaitForEndOfFrame（暂停到下一帧继续执行）等等。更巧妙的是yield 也可以返回一个Coroutine真身，Coroutine A返回一个Coroutine B本身的时候，即等到B做完了再执行A。下面有详细说明：

```
Normal coroutine updates are run after the Update function returns. A coroutine is a function that can suspend its execution (yield) until the given YieldInstruction finishes. Different uses of Coroutines:

yield; The coroutine will continue after all Update functions have been called on the next frame.
yield WaitForSeconds(2); Continue after a specified time delay, after all Update functions have been called for the frame
yield WaitForFixedUpdate(); Continue after all FixedUpdate has been called on all scripts
yield WWW Continue after a WWW download has completed.
yield StartCoroutine(MyFunc); Chains the coroutine, and will wait for the MyFunc coroutine to complete first.
```

实现延时的关键代码是在StartCoroutine里面，以为笔者也没有见过Unity的源码，那么我只能猜想StartCoroutine这个函数的内部构造应该是这样的：

```
List<IEnumerator> unblockedCoroutines;
List<IEnumerator> shouldRunNextFrame;
List<IEnumerator> shouldRunAtEndOfFrame;
SortedList<float, IEnumerator> shouldRunAfterTimes;

foreach(IEnumerator coroutine in unblockedCoroutines){
        if(!coroutine.MoveNext())
                // This coroutine has finished
                continue;

        if(!coroutine.Current is YieldInstruction)
        {
```

关闭


```
                // This coroutine yielded null, or some other value we don't understand;
 run it next frame.
                shouldRunNextFrame.Add(coroutine);
                continue;
        }

        if(coroutine.Current is WaitForSeconds)
        {
                WaitForSeconds wait = (WaitForSeconds)coroutine.Current;
                shouldRunAfterTimes.Add(Time.time + wait.duration, coroutine);
        }
        else if(coroutine.Current is WaitForEndOfFrame)
        {
                shouldRunAtEndOfFrame.Add(coroutine);
        }
        else /* similar stuff for other YieldInstruction subtypes */}

unblockedCoroutines = shouldRunNextFrame;
```

当然了，我们还可以为YieldInstruction添加各种的子类，比如一个很容易想到的就是yield return new WaitForNotification(“GameOver”)来等待某个消息的触发，关于Unity的消息机制可以参考这篇文章： 【Unity3D技巧】在Unity中使用事件/委托机制（event/delegate）进行GameObject之间的通信 (二)：引入中间层NotificationCenter 。

快速回复 我要收藏

还有些更好玩的?

第一个有趣的地方是，yield return可以返回任意YieldInstruction，所以我们可以在这里加上一些条件判断：

```
YieldInstruction y;

if(something)
 y = null;else if(somethingElse)
 y = new WaitForEndOfFrame();else
 y = new WaitForSeconds(1.0f);

yield return y;
```

第二个，由于一个协程只是一个 迭代器块 而已，所以你也可以自己遍历它，这在一些场景下很有用，例如在对协程是否执行加上条件判断的时候：

```
IEnumerator DoSomething(){
  /* ... */}

IEnumerator DoSomethingUnlessInterrupted(){
  IEnumerator e = DoSomething();
  bool interrupted = false;
  while(!interrupted)
  {
    e.MoveNext();
    yield return e.Current;
    interrupted = HasBeenInterrupted();
  }}
```

第三个，由于协程可以yield协程，所以我们可以自己创建一

```
IEnumerator UntilTrueCoroutine(Func fn){
   while(!fn()) yield return null;}

Coroutine UntilTrue(Func fn){
  return StartCoroutine(UntilTrueCoroutine(fn));}

IEnumerator SomeTask(){
  /* ... */
```

关闭


```
yield return UntilTrue(() => _lives < 3);
/* ... */}
```

总结

这篇文章大部分是我从 这篇博客 里面翻译过来的，这是我见过最棒的一篇关于Coroutine的博客，所以我把它翻译过来与大家分享，希望你能喜欢。

顶 0 踩 0

上一篇 使用自定义工具链toolchain编译开源项目
下一篇 Android 开源网络框架（Android-Async-Http、Volley、OkHttp3、Retrofit2.0）对比

快速回复
我要收藏

参考知识库

Unity3D知识库
3520 关注 | 509 收录

C#知识库
4931 关注 | 827 收录

虚拟现实（VR）知识库
6877 关注 | 501 收录

算法与数据结构知识库
15413 关注 | 2320 收录

猜你在找

- 从此不求人:自主研发一套PHP前端开发框架
- PHP网站搭建入门
- Hadoop生态系统零基础入门
- HTML 5移动开发从入门到精通
- HTML 5视频教程系列之JavaScript学习篇
- Unity3DLua中使用协程coroutine和计时器timer
- Unity3d协程实现倒数计时
- Unity3D协程介绍 以及 使用
- Unity3D-关于协程的一些东西
- Unity3D开发小贴士二协程Coroutine

查看评论
暂无评论
您还没有登录,请[登录]或[注册]
* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立

核心技术类目
全部主题 Hadoop AWS 移动游戏 Java Android iOS Sw
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库
BI HTML5 Spring Apache .NET API HTML SDK IIS
Splashtop UML components Windows Mobile Rails QEMU
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspac

Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 | 江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved

快速回复
我要收藏

关闭