**Unity面试题**

**####\*\*- C#代理和事件的区别？\*\***

答：委托是一个类，它定义了方法的类型，使得可以将方法当作另一个方法的参数来进行传递，这种将方法动态地赋给参数的做法，可以避免在程序中大量使用If-Else(Switch)语句，同时使得程序具有更好的可扩展性。

**####\*\*- 什么是Draw Call？减少Draw Call的方法？\*\***

答：Unity每次在准备数据并通知GPU渲染的过程称为一次Draw Call。

其实就是对底层图形程序（比如：OpenGL ES)接口的调用，以在屏幕上画出东西。

一般情况下，渲染一次拥有一个网格并携带一种材质的物体便会使用一次Draw Call。

对于渲染场景中的这些物体，在每一次Draw Call中除了在通知GPU的渲染上比较耗时之外，切换材质与shader也是非常耗时的操作。

Draw Call的次数是决定性能比较重要的指标。

1. 使用Draw Call Batching，也就是描绘调用批处理。Unity在运行时可以将一些物体进行合并，从而用一个描绘调用来渲染他们。

2. 通过把纹理打包成图集来尽量减少材质的使用。

3. 尽量少的使用反光啦，阴影啦之类的，因为那会使物体多次渲染。

**####\*\*- 资源动态加载的方式和优缺点？\*\***

答：

1. 通过Resources模块，调用它的load函数：可以直接load并返回某个类型的Object，前提是要把这个资源放在Resource命名的文件夹下，Unity不关有没有场景引用，都会将其全部打入到安装包中。

2. 通过bundle的形式：即将资源打成 asset bundle 放在服务器或本地磁盘，然后使用WWW模块get 下来，然后从这个bundle中load某个object。

3. 通过AssetDatabase.loadasset ：这种方式只在editor范围内有效，游戏运行时没有这个函数，它通常是在开发中调试用的

Resources的方式需要把所有资源全部打入安装包，这对游戏的分包发布（微端）和版本升级（patch）是不利的，所以unity推荐的方式是不用它，都用bundle的方式替代，把资源达成几个小的bundle，用哪个就load哪个，这样还能分包发布和patch，但是在开发过程中，不可能没更新一个资源就打一次bundle，所以editor环境下可以使用AssetDatabase来模拟，这通常需要我们封装一个dynamic resource的loader模块，在不同的环境下做不同实现。

**####\*\*- Asset Serialization 中mixed、force binary、force text 区别？\*\***

**####\*\*- 添加组件和删除组件的方法？\*\***

答：获取： GetComponent

增加： AddComponent

删除： Destroy

\*\*\*Note that there is no RemoveComponent(), to remove a component, use Object.Destroy.\*\*\*

**####\*\*- 协程是什么？列举常用的场景？\*\***

**####\*\*- 协程和C#多线程的区别？\*\***

答：

http://blog.csdn.net/kongbu0622/article/details/8775037

1. 多线程程序同时运行多个线程 ，而在任一指定时刻只有一个协程在运行，并且这个正在运行的协同程序只在必要时才被挂起。除主线程之外的线程无法访问 Unity3D 的对象、组件、方法。 Unity3d 没有多线程的概念，不过 unity 也给我们提供了StartCoroutine （协同程序）和 LoadLevelAsync （异步加载关卡）后台加载场景的方法。

2. StartCoroutine 为什么叫协同程序呢，所谓协同，就是当你在 StartCoroutine 的函数体里处理一段代码时，利用 yield 语句等待执行结果，这期间不影响主程序的继续执行，可以协同工作。而 LoadLevelAsync 则允许你在后台加载新资源和场景，所以再利用协同，你就可以前台用 loading 条或动画提示玩家游戏未卡死，同时后台协同处理加载的事宜。

**####\*\*- 光照举例？\*\***

答：

平行光： Directional Light

聚光灯： Spot Light

点光源： Point Light

区域光源： Area Light （只用于烘培）

**####\*\*- Cam设置为Depth Only有什么用？\*\***

答：为相机使用Depth Only作为清除标志，会使相机仅仅根据相机的深度信息来输出画面，比如相机深度为N的清除标志设为Depth only，那么它的输出会直接覆盖在所有深度小于N的相机输出画面上，而不管在真实的3D环境中，各种对象的z值如何。

**####\*\*- Unity如何与ios和安卓交互？\*\***

**####\*\*- Mesh中material和sharedMaterail的区别？\*\***

答：修改 sharedMaterial 将改变所有物体使用这个材质的外观，并且也改变储存在工程里的材质设置。

不推荐修改由 sharedMaterial 返回的材质。如果你想修改渲染器的材质，使用 material替代。

**####\*\*- Alpha Test和Alpha Blend区别？\*\***

**####\*\*- FixedUpdate和Update 区别？\*\***

答：FixedUpdate ，每固定帧绘制时执行一次，和 update 不同的是 FixedUpdate 是渲染帧执行，如果你的渲染效率低下的时候 FixedUpdate 调用次数就会跟着下降。FixedUpdate 比较适用于物理引擎的计算，因为是跟每帧渲染有关。 Update 就比较适合做控制。

**####\*\*- cam设置写在哪个回调函数里？\*\***

答：LateUpdate, ，是在所有 update 结束后才调，比较适合用于命令脚本的执行。官网上例子是摄像机的跟随，都是在所有 update 操作完才跟进摄像机，不然就有可能出现摄像机已经推进了，但是视角里还未有角色的空帧出现。

**####\*\*- LightMap是什么？\*\***

答：简单地说, 就是把物体光照的明暗信息保存到纹理上, 实时绘制时不再进行光照计算, 而是采用预先生成的光照纹理(lightmap)来表示明暗效果。

\*\*好处:\*\*

- 由于省去了光照计算, 可以提高绘制速度；

- 对于一些过度复杂的光照(如光线追踪, 辐射度, AO等算法), 实时计算不太现实. 如果预先计算好保存到纹理上, 这样无疑可以大大提高模型的光影效果；

- 保存下来的lightmap还可以进行二次处理, 如做一下模糊, 让阴影边缘更加柔和

\*\*当然, 缺点也是有的:\*\*

- 模型额外多了一层纹理, 这样相当于增加了资源的管理成本(异步装载, 版本控制, 文件体积等). 当然, 也可以选择把明暗信息写回原纹理, 但这样限制比较多, 如纹理坐标范围, 物体实例个数...

- 模型需要隔外一层可以展开到一张纹理平面的UV(范围只能是[0,1], 不能重合). 如果原模型本身就是这样, 可以结省掉. 但对于大多数模型来说, 可能会采用WRAP/MIRROR寻址, 这只能再做一层, 再说不能强制每个模型只用一张纹理吧? 所以, lightmap的UV需要美术多做一层, 程序展开算法这里不提及....

- 静态的光影效果与对动态的光影没法很好的结合. 如果光照方向改变了的话, 静态光影效果是无法进行变换的. 而且对于静态的阴影, 没法直接影响到动态的模型. 这一点, 反而影响了真实度

**####\*\*- 如何减少GC回调次数？\*\***

答：首先我们要明确所谓的GC是Mono运行时的机制，而非Unity3D游戏引擎的机制，所以GC也主要是针对Mono的对象来说的，而它管理的也是Mono的托管堆。 搞清楚这一点，你也就明白了GC不是用来处理引擎的assets（纹理啦，音效啦等等）的内存释放的，因为U3D引擎也有自己的内存堆而不是和Mono一起使用所谓的托管堆。

其次我们要搞清楚什么东西会被分配到托管堆上？不错咯，就是引用类型咯。比如类的实例，字符串，数组等等。而作为int，float，包括结构体struct其实都是值类型，它们会被分配在堆栈上而非堆上。所以我们关注的对象无外乎就是类实例，字符串，数组这些了。

\*\*那么GC什么时候会触发呢？两种情况：\*\*

首先当然是我们的堆的内存不足时，会自动调用GC。

其次呢，作为编程人员，我们自己也可以手动的调用GC。

所以为了达到优化CPU的目的，我们就不能频繁的触发GC。而上文也说了GC处理的是托管堆，而不是Unity3D引擎的那些资源，所以GC的优化说白了也就是代码的优化。那么匹夫觉得有以下几点是需要注意的：

- 字符串连接的处理。因为将两个字符串连接的过程，其实是生成一个新的字符串的过程。而之前的旧的字符串自然而然就成为了垃圾。而作为引用类型的字符串，其空间是在堆上分配的，被弃置的旧的字符串的空间会被GC当做垃圾回收。

- 尽量不要使用foreach，而是使用for。foreach其实会涉及到迭代器的使用，而据传说每一次循环所产生的迭代器会带来24 Bytes的垃圾。那么循环10次就是240Bytes。

- 不要直接访问gameobject的tag属性。比如if (go.tag == “human”)最好换成if (go.CompareTag (“human”))。因为访问物体的tag属性会在堆上额外的分配空间。如果在循环中这么处理，留下的垃圾就可想而知了。

- 使用“池”，以实现空间的重复利用。

- 最好不用LINQ的命令，因为它们会分配临时的空间，同样也是GC收集的目标。而且我很讨厌LINQ的一点就是它有可能在某些情况下无法很好的进行AOT编译。比如“OrderBy”会生成内部的泛型类“OrderedEnumerable”。这在AOT编译时是无法进行的，因为它只是在OrderBy的方法中才使用。所以如果你使用了OrderBy，那么在IOS平台上也许会报错。

**####\*\*- timescale = 0.5时 执行速度是变慢还是变快？\*\***

答：变慢

**####\*\*- 简单说明渲染管线？\*\***

**####\*\*- 物体围绕特定轴旋转的API？\*\***

答：`transform.RotateAround(Vector3 point, Vector3 axis, float angle);`

**####\*\*- 请简述值类型与引用类型的区别？\*\***

答：区别：

1. 值类型存储在内存栈中，引用类型数据存储在内存堆中，而内存单元中存放的是堆中存放的地址。

2. 值类型存取快，引用类型存取慢。

3. 值类型表示实际数据，引用类型表示指向存储在内存堆中的数据的指针和引用。

4. 栈的内存是自动释放的，堆内存是 .NET 中会由 GC 来自动释放。 5. 值类型继承自 System.ValueType, 引用类型继承自 System.Object 。

可参考 http://www.cnblogs.com/JimmyZhang/archive/2008/01/31/1059383.html

**####\*\*- 请描述 Interface 与抽象类之间的不同？\*\***

答：

1. 抽象类表示该类中可能已经有一些方法的具体定义，但接口就是公公只能定义各个方法的界面 ，不能具体的实现代码在成员方法中。

2. 类是子类用来继承的，当父类已经有实际功能的方法时该方法在子类中可以不必实现，直接引用父类的方法，子类也可以重写该父类的方法。

3. 实现接口的时候必须要实现接口中所有的方法，不能遗漏任何一个。

**####\*\*- 请简述 private ， public ， protected ， internal 的区别？\*\***

答：

1. public ：对任何类和成员都公开，无限制访问

2. private ：仅对该类公开

3. protected：对该类和其派生类公开

4. internal ：只能在包含该类的程序集中访问该类

**####\*\*- Unity是左手还是右手坐标系？\*\***

答：左手坐标系

**####\*\*- 向量的点乘、叉乘以及归一化的意义？\*\***

答：

1. 点乘描述了两个向量的相似程度，结果越大两向量越相似，还可表示投影

2. 叉乘得到的向量垂直于原来的两个向量

3. 标准化向量：用在只关系方向，不关心大小的时候

**####\*\*- Struct 和 Class 的区别？\*\***

答：

- 类是引用类型，struct是值类型

- 在托管堆上创建类的实例，在栈上创建struct实例

- 类实例的赋值，赋的是引用地址，struct实例的赋值，赋的是值

- 类作为参数类型传递，传递的是引用地址，struct作为参数类型传递，传递的是值

- 类没有默认无参构造函数，struct有默认无参构造函数

- 类支持继承，struct不支持继承

- 类偏向于"面向对象",用于复杂、大型数据，struct偏向于"简单值"，比如小于16字节，结构简单

- 类的成员很容易赋初值，很难给struct类型成员赋初值

- 类的实例只能通过new SomeClass()来创建，struct类型的实例既可以通过new SomeStruct()来创建，也可以通过SomeStruct myStruct;来创建

**####\*\*- 静态批处理与动态批处理的区别？\*\***

答：

#####Static Batching 静态批处理：

只要这些物体不移动，并且拥有相同的材质，静态批处理就允许引擎对任意大小的几何物体进行批处理操作来降低描绘调用。静态批处理的好处很多，其中之一就是与下面要说的动态批处理相比，约束要少很多。所以一般推荐的是draw ﻿call的静态批处理来减少draw call的次数。那么接下来，我们就继续聊聊draw call的动态批处理。

#####Dynamic Batching 动态批处理

首先要明确一点，Unity3D的draw call动态批处理机制是引擎自动进行的，无需像静态批处理那样手动设置static。如果动态物体共享相同的材质，则引擎会自动对draw call优化，也就是使用批处理。

- 批处理动态物体需要在每个顶点上进行一定的开销，所以动态批处理仅支持小于900顶点的网格物体。

- 使用不同材质的实例化物体（instance）将会导致批处理失败。

- 统一缩放的物体不会与非统一缩放的物体进行批处理。

- 预设体的实例会自动地使用相同的网格模型和材质。