**1.协同程序实现原理，多线程和协程是一个概念吗？**

https://blog.csdn.net/LIQIANGEASTSUN/article/details/140809545

**2.Xml,Json,Protobuffer等序列化的区别。如何选型**

https://blog.csdn.net/LIQIANGEASTSUN/article/details/140831767

**3.解释ECS的概念和思想，如何应用，又用在何处**

<https://github.com/sschmid/Entitas>

https://blog.csdn.net/qq\_42461824/article/details/113141736

**4.如何为 Unity 的 C# 项目添加 dll 引用或安装 NuGet 包**

<https://blog.walterlv.com/post/unity-starter-reference-dlls-and-add-nuget-package-for-unity-csharp-projects.html>

**5.使用 Visual Studio 开发 Unity C# 脚本，说说根目录的那些 sln 和 csproj 文件**

https://blog.walterlv.com/post/unity-starter-the-sln-and-csproj-files.html

**6.堆空间、栈空间 的概念，区别，分别是存储什么数据的？**

托管堆、非托管堆IDispose

堆栈在内存中的存放顺序，说明堆栈的区别

博客：<https://blog.csdn.net/m0_49768044/article/details/131391832>

<https://zhuanlan.zhihu.com/p/42585757>

<https://blog.csdn.net/jkkk_/article/details/127856675>

https://blog.51cto.com/u\_16099174/10676149

Unity 内存管理https://docs.unity3d.com/cn/2022.3/Manual/performance-managed-memory.html

<https://zhuanlan.zhihu.com/p/367435572>

<https://blog.51cto.com/u_16099224/10948676>

<https://www.jianshu.com/p/e427a38e91c3>

https://maimai.cn/article/detail?fid=1735999578&efid=W1Z36XSSpVI2HBh32t6gXw

**7.C# 中值类型与引用类型分别有哪些，有什么不同，装箱拆箱是什么**

**8.虚拟内存和物理内存的区别是什么**

<https://maimai.cn/article/detail?fid=1735999578&efid=W1Z36XSSpVI2HBh32t6gXw>

**9.C# List 是数组实现的还是链表实现的**

数组

**10.C# 中委托是链表还是数组实现的**

链表

**11.C#中操作两个字符串相加时，会有几次内存操作？**

**C# string 相加和 StringBuilder.Append 的主要区别是什么？**

直接写`string str = "a" + "b";`会有几次内存申请操作

直接写`string str = "a" + "b";`会有几次内存申请操作

上面这种写法只会有一次内存操作，其实C#编译器比想象中要聪明得多，C#编译器直接编译为

str = “ab”;

**12.如果将C#的代码打包成DLL，DLL中的内容是编译之后的二进制文件吗，还是其他内容？**

是编译后的二进制文件

DLL中的内容是编译之后的二进制文件。**‌**

在C#中，‌当你使用csc /target:library /out:myDll.DLL myDll.cs这样的命令编译代码时，‌生成的是一个动态链接库（‌DLL）‌文件。‌这个DLL文件包含了编译后的二进制代码和数据，‌可以被其他程序调用和使用。‌DLL文件是一种可执行代码的共享库，‌它包含了可以被其他程序调用的函数和数据。‌通过将代码编译成DLL，‌可以实现代码的重用和模块化，‌提高软件的开发效率和可维护性

**13.Unity中的C#编译过程是什么样的？**

Unity中的C#脚本编译过程可以分为几个主要步骤：

1. \*\*编写代码\*\*：

开发者使用C#语言编写脚本，并保存在Unity项目的`Assets`文件夹中。这些脚本通常以`.cs`为文件扩展名。

2. \*\*脚本加载\*\*：

当Unity编辑器启动或脚本被添加到项目中时，Unity会加载这些脚本文件。

3. \*\*预处理\*\*：

在编译之前，Unity会进行一些预处理步骤，例如处理`#define`指令和条件编译符号。

4. \*\*编译\*\*：

Unity使用内置的Roslyn编译器（从2017.1版本开始）将C#脚本编译成.NET的中间语言（IL - Intermediate Language）代码。这个过程是自动的，通常在编辑器内部静默进行。

5. \*\*汇编\*\*：

编译后的IL代码随后被汇编成可执行的机器代码。Unity使用Mono作为其脚本运行时，因此汇编过程会生成与Mono虚拟机兼容的代码。

6. \*\*代码优化\*\*：

编译过程中，Unity还会进行代码优化，以提高运行时性能。

7. \*\*生成Assembly\*\*：

编译和优化后的代码被组织成Assembly（程序集）。在Unity中，每个Assembly是一个包含已编译代码的`.dll`文件。

8. \*\*链接\*\*：

生成的Assembly随后被链接到Unity的应用程序中。Unity会管理这些Assembly的依赖关系，并确保它们在运行时可以被正确加载。

9. \*\*运行时加载\*\*：

当Unity应用程序运行时，Mono虚拟机会加载这些Assembly，并执行其中的代码。

10. \*\*热重载\*\*（可选）：

Unity支持热重载，这意味着在不重启应用程序的情况下，开发者可以更改和重新编译脚本，而Unity会尝试重新加载这些更改过的脚本，无需中断应用程序的运行。

11. \*\*调试\*\*：

开发者可以使用Unity编辑器内置的调试器或Visual Studio等外部工具来调试C#脚本。

值得注意的是，Unity的编译过程是高度集成和自动化的，开发者通常不需要手动干预。Unity提供了一个优化的环境，使得C#脚本的编写、编译和调试变得简单和高效。此外，Unity的构建系统还支持将项目导出到不同平台，每个平台可能有自己的编译和构建要求。

**14.解释型语言和编译型语言的区别是什么？**

计算机不能直接理解高级语言，只能直接理解机器语言，所以必须要把高级语言翻译成机器语言，计算机才能执行高级语言编写的程序

一个是编译，一个是解释。两种方式只是翻译的时间不同。编译型语言写的程序执行之前，需要一个专门的编译过程，把程序编译成为机器语言的文件，比如exe文件，以后要运行的话就不用重新翻译了，直接使用编译的结果就行了（exe文件），因为翻译只做了一次，运行时不需要翻译，所以编译型语言的程序执行效率高，但也不能一概而论，部分解释型语言的解释器通过在运行时动态优化代码，甚至能够使解释型语言的性能超过编译型语言。

解释则不同，解释性语言的程序不需要编译，省了道工序，解释性语言在运行程序的时候才翻译，比如解释性basic语言，专门有一个解释器能够直接执行basic程序，每个语句都是执行的时候才翻译。这样解释性语言每执行一次就要翻译一次，效率比较低。解释是一句一句的翻译。

解释型语言：程序不需要编译，程序在运行时才翻译成机器语言，每执 行一次都要翻译一次。因此效率比较低。比如Basic语言，专门有一个解释器能够直接执行Basic程 序，每个语句都是执行的时候才翻译。(在运行程序的时候才翻译，专门有一个解释器去进行翻译，每个语句都是执行的时候才翻译。效率比较低，依赖解释器，跨 平台性好

**15.Unity上滚动列表ScrollView中的元素如果包含粒子特效，怎么处理元素滚动到被遮挡裁切部分的特效没有被正确裁切问题？**

推荐使用 UIParticle 插件

**16.Unity手机游戏开发时，通常如何处理不同手机的适配问题？**

**UGUI Canvas 如何适配，Canvas属性 UIScaleModel、ReferenceResolution、ScreenMatchModel、ReferencePixels Pre Unit 的作用**

在参考分辨率为1920x1080的情况下，如果在一个2000x1080分辨率的屏幕上，Canvas会怎么适配？

1920x1080的图片在2000x1080的屏幕上会怎么显示？

<https://zhuanlan.zhihu.com/p/644845905>

**17.如果要保证高度不变，应该怎么计算？**

**18.Unity游戏接入SDK的流程一般是什么？**

**19.Unity的Animation和Animator的区别是什么？IK是什么？**

Animation：使用关键帧动画，需要为每个对象的每个属性 位置、旋转、缩放等创建动画轨道，这种动画适用于简单的、预定好的动画序列

不适合复杂逻辑：Animation 动画通常不适用于需要复杂逻辑和交互的场景，因为他们不支持状态机和编程控制。

适合简单的动画

Animator：使用状态机系统控制，可以创建复杂的动画逻辑，定义不同的状态，状态之间的过度融合，可以创建复杂的动画和控制逻辑

可编程性：Animator 动画系统提供了更多的变成控制，可以在脚本中实现动画逻辑的交互和控制，可以通过代码控制动画状态，参数、过度等，可以相应动画事件

多层混合，可以使用多个层级Layer，来混合显示，如身体动画，可以同时显示动作A 的上半身，动画B 的下半身 等多层混合

适用于复杂的角色控制和互动

**20.Unity的UGUI界面的构建过程是什么？在制作UGUI界面Prefab时，有什么技巧或者方式来降低性能开销？UGUI中什么操作会导致重绘？UGUI的渲染流程是什么？**

<https://www.zhihu.com/question/559205719/answer/3228051762>

**21.一个相机中如何做分层渲染？底层原理是什么?**

**22.Socket TCP、Udp的连接的流程是怎样的，他们的区别？**

**23.项目中的打包工具做了什么事情？如何做自动化打包？**

**24.3D转2D碰撞检测的实现过程，使用什么算法进行检测？**

（1）可以使用 Colider2D、Rigidbody2D

通过 OnCollisionEnter2D 、OnTriggerEnter2D()方法触发

（2）使用AABB、OBB 相交检测算法

（3）将 两个物体 投影到 一维 坐标系，判断两个投影是否相交，如下两个红线是否相交

将相交检测转换到一维坐标系的相交检测

**25.讲讲你对URP的了解**

**26.代码安全具体做了什么内容？**

**27.LineRenderer的底层实现原理是什么？**

**28.讲一下二维碰撞检测的算法实现，四叉树。**

AABB（无旋转矩形）相交检测

OBB（有旋转矩形）相交检测

分离轴定理

slab 的碰撞检测算法

四叉树：一般以矩形方式分割划分地图区域，根节点表示整个地区区域的大小，然后每个节点切分成四个子节点，每个节点表示一个矩形区域，每个节点大小为父节点 长和宽的一半，依次类推，直到一个节点的长、宽到达一定值，然后停止。

将物体添加到四叉树：

令根节点为当前节点

A如果当前节点有子节点，判断物体包含在哪个子节点，令这个节点作为当前节点，继续执行A，否则执行B

B当前节点没有子节点，将物体添加到当天节点返回

判断一个物体与哪个物体相交？

H令根节点为当前节点

I如果物体不在当前节点范围内，则判定，没有与之相交的物体

J如果物体在当前节点

K如果当前节点有子节点，判断物体在哪个子节点内，令子节点为当前节点，继续K，否则到L

L如果当前节点没有子节点，则获取当前节点内的所有物体，遍历判断与物体是否相交

上面忽略了一个情况，就是物体是有体积的，一个物体可能同时与多个子节点相交，其实都要处理的，上面只是简略说一些大概逻辑，如下图，蓝色圆其实与四个节点都相交

|  |  |
| --- | --- |
|  |  |
|  |  |

**29.讲一下UGUI与NGUI有哪些不同的地方**

**30.Unity 脚本生命周期？请说明 MonoBehaviour 中**

Awake、Start、OnEnable 函数区别

Update、LateUpdate、FixedUpdate 函数区别

<https://docs.unity.cn/cn/2022.3/Manual/ExecutionOrder.html>

**31.Unity 提供了哪几种光源，主要区别是什么**

**32.Unity 怎么使用 Layer 和 Tag**

<https://blog.csdn.net/xiaoyaoACi/article/details/134947768>

**33.MeshRender 中 material 和 sharedmaterial 区别**

<https://blog.csdn.net/lcfengokok/article/details/138334714>

https://blog.csdn.net/weixin\_46292030/article/details/136434384

**34.前向(forward rendering) 和 延迟渲染（Deferred Rendering）的主要区别和优劣是什么？**

什么时候使用现象渲染、什么时候使用延迟渲染

**35.多人游戏的同步方案一般都哪几种，请简述其原理**

帧同步、状态同步

**36.常用的 AI 系统有哪些？**

FSM有限状态机，

Hfsm层次有限状态机，、

GOAP目标行为规划

BehaviorTree行为树，

**37.Android x86、arm64-v8a、armeabi-v7a、iOS arm64的区别？**

**38.C#闭包**

https://blog.csdn.net/m0\_62389241/article/details/132340740

在C#中，闭包（Closure）是一个可以捕获并封装其外部环境中的变量的函数，这些变量会和该函数一同存在，即使外部环境已经结束。

闭包通常由两部分组成：

函数

引用环境

下面是一个简单的闭包示例：

public Action CreateClosure(int externalVariable)

{

return () =>

{

Console.WriteLine(externalVariable);

};

}

*// 使用方法*

var closure = CreateClosure(10);

closure(); *// 输出: 10*

在这个例子中，CreateClosure 方法创建了一个函数，该函数捕获了其外部环境中的变量 externalVariable。即使 CreateClosure 方法结束并返回后，externalVariable 依然存在于闭包中。

当你调用 closure() 时，它会打印出 externalVariable 的值，即使这个变量在创建闭包时已经离开了其作用域。这就是闭包的行为。

**39.Unity C# Job System**

[file:///H:/Software/Unity/2022.3.26f1/Editor/Data/Documentation/en/Manual/JobSystem.html](file://H:\\Software\\Unity\\2022.3.26f1\\Editor\\Data\\Documentation\\en\\Manual\\JobSystem.html)

**40.Unity 阴影实现方案都有哪些**