

Unity面试手册

[Unity面试手册：中级开发工程师面试题](#)

[Unity面试手册：2021最新Unity面试题汇总](#)

[Unity面试手册：Unity应届生面试题](#)

[Unity面试手册：Unity游戏公司真题](#)

[Unity面试手册：Unity官方认证考试模拟题](#)

[Unity面试手册：C#基础语法](#)

[Unity面试手册：C#面向对象](#)

[Unity面试手册：C#委托、集合、异常、泛型](#)

[Unity面试手册：C#多线程](#)

[Unity面试手册：设计模式](#)

[Unity面试手册：Lua面试题](#)

Unity面试手册： 中级开发工程师面试题

1.什么是渲染管道？

是指在显示器上为了显示出图像而经过的一系列必要操作。

渲染管道中的很多步骤，都要将几何物体从一个坐标系中变换到另一个坐标系中去。

主要步骤有：本地坐标->视图坐标->背面裁剪->光照->裁剪->投影->视图变换->光栅化。

2.如何进行内存优化？

1.压缩自带类库；

2.将暂时不用的以后还需要使用的物体隐藏起来而不是直接Destroy掉；

3.释放AssetBundle占用的资源；

4.降低模型的片面数，降低模型的骨骼数量，降低贴图的大小；

5.使用光照贴图，使用多层次细节(LOD)[全称 Levels of Detail ，进行物体不同细节层次之间的平滑过渡。根据物体所在的环境和所处位置的重要度，决定渲染。降低不重要和远的物体的面数和细节度。一般都是视距近的物体清楚，视距远的物体模糊。

3.动态加载资源的方式？

1.Resources.Load();

2.AssetBundle

4.什么是协同程序？

在主线程运行时同时开启另一段逻辑处理，来协助当前程序的执行。换句话说，开启协程就是开启一个线程。可以用来控制运动、序列以及对象的行为。

5.碰撞器和触发器的区别？

碰撞器有碰撞的效果，IsTrigger=false，可以调用OnCollisionEnter/Stay/Exit函数；

触发器没有碰撞效果，IsTrigger=true，可以调用OnTriggerEnter/Stay/Exit函数。

6.物体发生碰撞的必要条件？

两个物体都必须带有碰撞器(Collider)，其中一个物体还必须带有Rigidbody刚体。

7.在物体发生碰撞的整个过程中，有几个阶段，分别列出对应的函数？

三个阶段

1.OnCollisionEnter

2.OnCollisionStay

3.OnCollisionExit

8.Unity3d提供了一个用于保存和读取数据的类(PlayerPrefs)，请列出保存和读取整形数据的函数？

PlayerPrefs.SetInt() PlayerPrefs.GetInt()

9.解释一些Unity3d中的灯光有哪些？

有4种，点光源，区域光源，聚光灯，平行光。

10.Unity3d脚本从唤醒到销毁有着一套比较完善的生命周期，请举几个例子？

Awake——>Start——>Update——>FixedUpdate——>LateUpdate——>OnGUI——>Reset——>OnDisable——>OnDestroy

11.物流更新一般放在那个系统函数里？

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。

12.移动摄像机的动作放在那个系统函数中，为啥？

LateUpdate，在每帧执行完毕调用，它是在所有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。

13.当游戏中需要频繁创建一个物体时，我们需要怎么做？

使用预制物体对象Prefab，然后复制创建。

14.请简述Unity3d下如何安全的在不同工程间迁移asset数据，请列举出三种方法？

1.可以把assets目录和Library目录一起迁移

2.导出包

3.用Unity带的assets Server功能

15.请描述游戏动画有哪几种，以及其原理？

主要有关节动画、骨骼动画、单一网格模型动画(关键帧动画)。

关节动画：把角色分成若干独立部分，一个部分对应一个网格模型，部分的动画连接成一个整体的动画，角色比较灵活，Quake2中使用这种动画；

骨骼动画，广泛应用的动画方式，集成了以上两个方式的优点，骨骼按角色特点组成一定的层次结构，有关节相连，可做相对运动，皮肤作为单一网格蒙在骨骼之外，决定角色的外观；

单一网格模型动画由一个完整的网格模型构成，在动画序列的关键帧里记录各个顶点的原位置及其改变量，然后插值运算实现动画效果，角色动画较真实。

16.MipMap是什么，作用？

MipMapping：在三维计算机图形的贴图渲染中有常用的技术，为加快渲染进度和减少图像锯齿，贴图被处理成由一系列被预先计算和优化过的图片组成的文件，这样的贴图被称为MipMap。

17.localPosition和Position的区别？

全局变量和局部变量

Unity面试手册：2021最新Unity面试题汇总

1、什么是协同程序？

答：在主线程运行时同时开启另一段逻辑处理，来协助当前程序的执行。换句话说，开启协程就是开启一个可以与程序并行的逻辑。可以用来控制运动、序列以及对象的行为。

2、Unity3D中的碰撞器和触发器的区别？

答：碰撞器是触发器的载体，而触发器只是碰撞器身上的一个属性。

当Is Trigger=false时，碰撞器根据物理引擎引发碰撞，产生碰撞的效果，可以调用OnCollisionEnter/Stay/Exit函数；

当Is Trigger=true时，碰撞器被物理引擎所忽略，没有碰撞效果，可以调用OnTriggerEnter/Stay/Exit函数。

如果既要检测到物体的接触又不想让碰撞检测影响物体移动或要检测一个物件是否经过空间中的某个区域这时就可以用到触发器。

3、物体发生碰撞的必要条件？

答：两个物体都必须带有碰撞器Collider，其中一个物体还必须带有Rigidbody刚体。

4、七：简述四元数Quaternion的作用，四元数对欧拉角的优点？

答：四元数用于表示旋转

相对欧拉角的优点：能进行增量旋转、避免万向锁、给定方位的表达方式有两种，互为负（欧拉角有无数种表达方式）

5、如何安全的在不同工程间安全地迁移asset数据？三种方法

答：

1.将Assets和Library一起迁移

2.导出包package

3.用unity自带的assets Server功能

6、OnEnable、Awake、Start运行时的发生顺序？哪些可能在同一个对象周期中反复的发生？

答：Awake—>OnEnable—>Start，OnEnable在同一周期中可以反复地发生！

7、MeshRender中material和sharedmaterial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。

8、TCP/IP协议栈各个层次及分别的功能？

答：网络接口层：这是协议栈的最低层，对应OSI的物理层和数据链路层，主要完成数据帧的实际发送和接收。

网络层：处理分组在网络中的活动，例如路由选择和转发等，这一层主要包括IP协议、ARP、ICMP协议等。

传输层：主要功能是提供应用程序之间的通信，这一层主要是TCP/UDP协议。

应用层：用来处理特定的应用，针对不同的应用提供了不同的协议，例如进行文件传输时用到的FTP协议，发送email用到的SMTP等。

9、Unity提供了几种光源，分别是什么？

答：

四种。

平行光：Directional Light

点光源：Point Light

聚光灯：Spot Light

区域光源：Area Light

10、简述一下对象池，你觉得在FPS里哪些东西适合使用对象池？

对象池就存放需要被反复调用资源的一个空间，比如游戏中要常被大量复制的对象，子弹，敌人，以及任何重复出现的对象。

11、CharacterController和Rigidbody的区别？

Rigidbody具有完全真实物理的特性，而CharacterController可以说是受限的Rigidbody，具有一定的物理效果但不是完全真实的。

12、移动相机动作在哪个函数里，为什么在这个函数里？

LateUpdate，是在所有的Update结束后才调用，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，都是所有的Update操作完才进行摄像机的跟进，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。

13、简述prefab的用处

在游戏运行时实例化，prefab相当于一个模板，对你已经有的素材、脚本、参数做一个默认的配置，以便于以后的修改，同事prefab打包的内容简化了导出的操作，便于团队的交流。

14、GPU的工作原理？

简而言之，GPU的图形（处理）流水线完成如下的工作：（并不一定是按照如下顺序）。

顶点处理：这阶段GPU读取描述3D图形外观的顶点数据并根据顶点数据确定3D图形的形状及位置关系，建立起3D图形的骨架。在支持DX8和DX9规格的GPU中，这些工作由硬件实现的Vertex Shader

（定点着色器）完成。

光栅化计算：显示器实际显示的图像是由像素组成的，我们需要将上面生成的图形上的点和线通过一定的算法转换到相应的像素点。把一个矢量图形转换为一系列像素点的过程就称为光栅化。例如，一条数学表示的斜线段，最终被转化成阶梯状的连续像素点。

纹理贴图：顶点单元生成的多边形只构成了3D物体的轮廓，而纹理映射（texture mapping）工作完成对多边形表面的贴图，通俗的说，就是将多边形的表面贴上相应的图片，从而生成“真实”的图形。TMU（Texture mapping unit）即是用来完成此项工作。

像素处理：这阶段（在对每个像素进行光栅化处理期间）GPU完成对像素的计算和处理，从而确定每个像素的最终属性。在支持DX8和DX9规格的GPU中，这些工作由硬件实现的Pixel Shader（像素着色器）完成。

最终输出：由ROP（光栅化引擎）最终完成像素的输出，1帧渲染完毕后，被送到显存帧缓冲区。

总结：GPU的工作通俗的来说就是完成3D图形的生成，将图形映射到相应的像素点上，对每个像素进行计算确定最终颜色并完成输出。

15、什么是渲染管道？

答：是指在显示器上为了显示出图像而经过的一系列必要操作。渲染管道中的很多步骤，都要将几何物体从一个坐标系中变换到另一个坐标系中去。

主要步骤有：

本地坐标->视图坐标->背面裁剪->光照->裁剪->投影->视图变换->光栅化。

16、如何优化内存？

答：有很多方式，例如

- 1.压缩自带类库；
- 2.将暂时不用的以后还需要使用的物体隐藏起来而不是直接Destroy掉；
- 3.释放AssetBundle占用的资源；
- 4.降低模型的片面数，降低模型的骨骼数量，降低贴图的大小；
- 5.使用光照贴图，使用多层次细节(LOD)，使用着色器(Shader)，使用预设(Prefab)。

18、动态加载资源的方式？

1.Resources.Load();

2.AssetBundle

Unity5.1版本后可以选择使用Git: <https://github.com/applexiaohao/LOAssetFramework.git>

19、使用Unity3d实现2d游戏，有几种方式？

答：

1.使用本身的GUI、UGUI

2.把摄像机的Projection(投影)值调为Orthographic(正交投影)，不考虑z轴；

3.使用2d插件，如：2DToolkit、NGUI

20、在物体发生碰撞的整个过程中，有几个阶段，分别列出对应的函数 三个阶段

答：

OnCollisionEnter、

OnCollisionStay、

OnCollisionExit

21、Unity3d的物理引擎中，有几种施加力的方式，分别描述出来

答：

rigidbody.AddForce、

rigidbody.AddForceAtPosition

22、什么叫做链条关节？

答：Hinge Joint，可以模拟两个物体间用一根链条连接在一起的情况，能保持两个物体在一个固定距离内部相互移动而不产生作用力，但是达到固定距离后就会产生拉力。

23、物体自身旋转使用的函数？

答：Transform.Rotate()

24、Unity3d提供了一个用于保存和读取数据的类(PlayerPrefs)，请列出保存和读取整形数据的函数

答：

PlayerPrefs.SetInt()、
PlayerPrefs.GetInt()

25、Unity3d脚本从唤醒到销毁有着一套比较完整的生命周期，请列出系统自带的几个重要的方法。

答：Awake——>Start——>Update——>FixedUpdate——>LateUpdate——>OnGUI——>Reset——>OnDisable——>OnDestroy

26、物理更新一般放在哪个系统函数里？

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。

27、在场景中放置多个Camera并同时处于活动状态会发生什么？

答：游戏界面可以看到很多摄像机的混合。

28、如何销毁一个UnityEngine.Object及其子类？

答： 使用Destroy()方法;

29、请描述游戏动画有哪几种，以及其原理？

答：主要有关节动画、骨骼动画、单一网格模型动画(关键帧动画)。 关节动画：把角色分成若干独立部分，一个部分对应一个网格模型，部分的动画连接成一个整体的动画，角色比较灵活，Quake2中使用这种动画；

骨骼动画，广泛应用的动画方式，集成了以上两个方式的优点，骨骼按角色特点组成一定的层次结构，有关节相连，可做相对运动，皮肤作为单一网格蒙在骨骼之外，决定角色的外观；

单一网格模型动画由一个完整的网格模型构成，在动画序列的关键帧里记录各个顶点的原位置及其改变量，然后插值运算实现动画效果，角色动画较真实。

30、请描述为什么Unity3d中会发生在组件上出现数据丢失的情况

答： 一般是组件上绑定的物体对象被删除了

31、alpha blend工作原理？

答：Alpha Blend 实现透明效果，不过只能针对某块区域进行alpha操作，透明度可设。

32、写出光照计算中的diffuse的计算公式？

答：diffuse = $K_d \times \text{colorLight} \times \max(N \cdot L, 0)$ ； K_d 漫反射系数、colorLight 光的颜色、N 单位法线向量、L 由点指向光源的单位向量、其中N与L点乘，如果结果小于等于0，则漫反射为0。

33、LOD是什么，优缺点是什么？

答：LOD(Level of detail)多层次细节，是最常用的游戏优化技术。它按照模型的位置和重要程度决定物体渲染的资源分配，降低非重要物体的面数和细节度，从而获得高效率的渲染运算。

33、两种阴影判断的方法、工作原理？

本影和半影：

本影：景物表面上那些没有被光源直接照射的区域（全黑的轮廓分明的区域）。

半影：景物表面上那些被某些特定光源直接照射但并非被所有特定光源直接照射的区域（半明半暗区域）

工作原理：从光源处向物体的所有可见面投射光线，将这些面投影到场景中得到投影面，再将这些投影面与场景中的其他平面求交得出阴影多边形，保存这些阴影多边形信息，然后再按视点位置对场景进行相应处理得到所要求的视图（利用空间换时间，每次只需依据视点位置进行一次阴影计算即可，省去了一次消隐过程）

34、Vertex Shader是什么，怎么计算？

答：顶点着色器是一段执行在GPU上的程序，用来取代fixed pipeline中的transformation和lighting，Vertex Shader主要操作顶点。

Vertex Shader对输入顶点完成了从local space到homogeneous space（齐次空间）的变换过程，homogeneous space即projection space的下一个space。在这其间共有world transformation, view transformation和projection transformation及lighting几个过程。

35、MipMap是什么，作用？

答：MipMapping：在三维计算机图形的贴图渲染中有常用的技术，为加快渲染进度和减少图像锯齿，贴图被处理成由一系列被预先计算和优化过的图片组成的文件，这样的贴图被称为MipMap。

36、请描述Interface与抽象类之间的不同

答：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。

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。

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。

37、.Net与Mono的关系？

答：mono是.net的一个开源跨平台工具，就类似java虚拟机，java本身不是跨平台语言，但运行在虚拟机上就能够实现了跨平台。.net只能在windows下运行，mono可以实现跨平台编译运行，可以运行于Linux，Unix，Mac OS等。

38、简述Unity3D支持的作为脚本的语言的名称？

答：Unity的脚本语言基于Mono的.Net平台上运行，可以使用.NET库，这也为XML、数据库、正则表达式等问题提供了很好的解决方案。

Unity里的脚本都会经过编译，他们的运行速度也很快。这三种语言实际上的功能和运行速度是一样的，区别主要体现在语言特性上。

JavaScript、C#、Boo

39、Unity3D是否支持写成多线程程序？如果支持的话需要注意什么？

答：仅能从主线程中访问Unity3D的组件，对象和Unity3D系统调用

支持：如果同时你要处理很多事情或者与Unity的对象互动小可以用thread,否则使用coroutine。

注意：C#中有lock这个关键字,以确保只有一个线程可以在特定时间内访问特定的对象

40、Unity3D的协程和C#线程之间的区别是什么？

答：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。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。

41、U3D中用于记录节点空间几何信息的组件名称，及其父类名称

答：Transform 父类是 Component

42、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

43、矩阵相乘的意义及注意点？

答：用于表示线性变换：旋转、缩放、投影、平移、仿射

注意矩阵的蠕变：误差的积累

44、为何大家都在移动设备上寻求U3D原生GUI的替代方案

答：不美观，OnGUI很耗费时间，使用不方便

45、请简述如何在不同分辨率下保持UI的一致性

答：NGUI很好的解决了这一点，屏幕分辨率的自适应性，原理就是计算出屏幕的宽高比跟原来的预设的屏幕分辨率求出一个对比值，然后修改摄像机的size。

46、为什么dynamic font在unicode环境下优于static font

答：Unicode是国际组织制定的可以容纳世界上所有文字和符号的字符编码方案。

使用动态字体时，Unity将不会预先生成一个与所有字体的字符纹理。当需要支持亚洲语言或者较大的字体的时候，若使用正常纹理，则字体的纹理将非常大。

47、当一个细小的高速物体撞向另一个较大的物体时，会出现什么情况？如何避免？

答：穿透（碰撞检测失败）

48、请简述OnBecameVisible及OnBecameInvisible的发生时机，以及这一对回调函数的意义？

答：当物体是否可见切换之时。可以用于只需要在物体可见时才进行的计算。

49、什么叫动态合批？跟静态合批有什么区别？

答：如果动态物体共用着相同的材质，那么Unity会自动对这些物体进行批处理。动态批处理操作是自动完成的，并不需要你进行额外的操作。

区别：动态批处理一切都是自动的，不需要做任何操作，而且物体是可以移动的，但是限制很多。静态批处理：自由度很高，限制很少，缺点可能会占用更多的内存，而且经过静态批处理后的所有物体都不能再移动了。

50、简述StringBuilder和String的区别？

答：

String是字符串常量。

StringBuffer是字符串变量，线程安全。

StringBuilder是字符串变量，线程不安全。

String类型是个不可变的对象，当每次对String进行改变时都需要生成一个新的String对象，然后将指针指向一个新的对象，如果在一个循环里面，不断的改变一个对象，就要不断的生成新的对象，所以效率很低，建议在不断更改String对象的地方不要使用String类型。

StringBuilder对象在做字符串连接操作时是在原来的字符串上进行修改，改善了性能。这一点我们平时使用中也许都知道，连接操作频繁的时候，使用StringBuilder对象。

51、什么是LightMap?

答：LightMap:就是指在三维软件里实现打好光，然后渲染把场景各表面的光照输出到贴图，最后又通过引擎贴到场景上，这样就使物体有了光照的感觉。

42、Unity和cocos2d的区别

答：

Unity3D支持C#、javascript等，cocos2d-x 支持c++、Html5、Lua等。

cocos2d 开源 并且免费

Unity3D支持iOS、Android、Flash、Windows、Mac、Wii等平台的游戏开发，cocos2d-x支持iOS、Android、WP等。

43、Unity3D Shader分哪几种，有什么区别？

答：表面着色器的抽象层次比较高，它可以轻松地以简洁方式实现复杂着色。表面着色器可同时在前向渲染及延迟渲染模式下正常工作。

顶点片段着色器可以非常灵活地实现需要的效果，但是需要编写更多的代码，并且很难与Unity的渲染管线完美集成。

固定功能管线着色器可以作为前两种着色器的备用选择，当硬件无法运行那些酷炫Shader的时，还可以通过固定功能管线着色器来绘制出一些基本的内容。

44、协同程序的执行代码是什么？有何用处，有何缺点？

```
1 function Start() {
2     // - After 0 seconds, prints "Starting 0.0"
3     // - After 0 seconds, prints "Before WaitAndPrint Finishes 0.0"
4     // - After 2 seconds, prints "WaitAndPrint 2.0"
5     // 先打印"Starting 0.0"和"Before WaitAndPrint Finishes 0.0"两句,2秒后打
    印"WaitAndPrint 2.0"
6     print ("Starting " + Time.time );
7     // Start function WaitAndPrint as a coroutine. And continue execution
    while it is running
8     // this is the same as WaitAndPrint(2.0) as the compiler does it for
    you automatically
9     // 协同程序WaitAndPrint在Start函数内执行,可以视同于它与Start函数同步执行.
10    StartCoroutine(WaitAndPrint(2.0));
11    print ("Before WaitAndPrint Finishes " + Time.time );
12 }
```

```
1 function WaitAndPrint (waitTime : float) {
2     // suspend execution for waitTime seconds
3     // 暂停执行waitTime秒
4     yield WaitForSeconds (waitTime);
5     print ("WaitAndPrint "+ Time.time );
6 }
```

作用：一个协同程序在执行过程中,可以在任意位置使用yield语句。yield的返回值控制何时恢复协同程序向下执行。协同程序在对象自有帧执行过程中堪称优秀。协同程序在性能上没有更多的开销。

缺点：协同程序并非真线程，可能会发生堵塞。

45、Mock和Stub有何区别？

Mock与Stub的区别：Mock:关注行为验证。细粒度的测试，即代码的逻辑，多数情况下用于单元测试。Stub：关注状态验证。粗粒度的测试，在某个依赖系统不存在或者还没实现或者难以测试的情况下使用，例如访问文件系统，数据库连接，远程协议等。

46、射线检测碰撞物的原理是？

答：射线是3D世界中一个点向一个方向发射的一条无终点的线，在发射轨迹中与其他物体发生碰撞时，它将停止发射。

47、客户端与服务器交互方式有几种？

答：socket通常也称作"套接字",实现服务器和客户端之间的物理连接，并进行数据传输，主要有UDP和TCP两个协议。Socket处于网络协议的传输层。

http协议传输的主要有http协议 和基于http协议的Soap协议（web service）,常见的方式是 http 的 post 和get 请求，web 服务。

48、八十三：Unity中，照相机的Clipping Planes的作用是什么？调整Near、Fare两个值时，应该注意什么？

答：剪裁平面。从相机到开始渲染和停止渲染之间的距离。

49、如何在Unity3D中查看场景的面试，顶点数和Draw Call数？如何降低Draw Call数？

答：在Game视图右上角点击Stats。降低Draw Call 的技术是Draw Call Batching

50、请问alpha test在何时使用？能达到什么效果？

Alpha Test,中文就是透明度测试。简而言之就是V&F shader中最后fragment函数输出的该点颜色值（即上一讲frag的输出half4）的alpha值与固定值进行比较。Alpha Test语句通常于Pass{}中的起始位置。Alpha Test产生的效果也很极端，要么完全透明，即看不到，要么完全不透明。

51、UNITY3d在移动设备上的一些优化资源的方法

答：

1.使用assetbundle，实现资源分离和共享，将内存控制到200m之内，同时也可以实现资源的在线更新

2.顶点数对渲染无论是cpu还是gpu都是压力最大的贡献者，降低顶点数到8万以下，fps稳定到了30帧左右

3.只使用一盏动态光，不是用阴影，不使用光照探头

粒子系统是cpu上的大头

4.剪裁粒子系统

5.合并同时出现的粒子系统

6.自己实现轻量级的粒子系统，nimator也是一个效率奇差的地方

7.把不需要跟骨骼动画和动作过渡的地方全部使用animation，控制骨骼数量在30根以下

8.animator出视野不更新

9.删除无意义的animator

10.animator的初始化很耗时（粒子上能不能尽量不用animator）

11.除主角外都不要跟骨骼运动apply root motion

12.绝对禁止掉那些不带刚体带包围盒的物体（static collider）运动

NUGI的代码效率很差，基本上runtime的时候对cpu的贡献和render不相上下

13每帧递归的计算finalalpha改为只有初始化和变动时计算

14去掉法线计算

15不要每帧计算viewsize 和window size

16filldrawcall时构建顶点缓存使用array.copy

17.代码剪裁：使用strip level，使用.net2.0 subset

18.尽量减少smooth group

19.给美术定一个严格的经过科学验证的美术标准，并在U3D里面配以相应的检查工具

52、四元数有什么作用？

答：对旋转角度进行计算时用到四元数

53、将Camera组件的ClearFlags选项选成Depth only是什么意思？有何用处？

答：仅深度，该模式用于对象不被裁剪。

54、如何让已经存在的GameObject在LoadLevel后不被卸载掉？



Plain Text

复制代码

```
1 void Awake()  
2 {  
3     DontDestroyOnLoad(transform.gameObject);  
4 }
```

55、在编辑场景时将GameObject设置为Static有何作用？

答：设置游戏对象为Static将会剔除（或禁用）网格对象当这些部分被静态物体挡住而不可见时。因此，在你的场景中的所有不会动的物体都应该标记为Static。

56、有A和B两组物体，有什么办法能够保证A组物体永远比B组物体先渲染？

答：把A组物体的渲染对列大于B物体的渲染队列

57、将图片的TextureType选项分别选为Texture和Sprite有什么区别

答：Sprite作为UI精灵使用，Texture作用模型贴图使用。

58、问一个Terrain，分别贴3张，4张，5张地表贴图，渲染速度有什么区别？为什么？

答：没有区别，因为不管几张贴图只渲染一次。

59、什么是DrawCall? DrawCall高了又什么影响? 如何降低DrawCall?

答：Unity中，每次引擎准备数据并通知GPU的过程称为一次Draw Call。DrawCall越高对显卡的消耗就越大。降低DrawCall的方法：

Dynamic Batching

Static Batching

高级特性Shader降级为统一的低级特性的Shader。

60、实时点光源的优缺点是什么?

答：可以有cookies — 带有 alpha通道的立方图(Cubemap)纹理。点光源是最耗费资源的。

61、Unity的Shader中，Blend SrcAlpha OneMinusSrcAlpha这句话是什么意思?

答：作用就是Alpha混合。公式：最终颜色 = 源颜色 源透明值 + 目标颜色 (1 - 源透明值)

62、简述水面倒影的渲染原理?

答：原理就是对水面的贴图纹理进行扰动，以产生波光粼粼的效果。用shader可以通过GPU在像素级别作扰动，效果细腻，需要的顶点少，速度快

63、简述NGUI中Grid和Table的作用?

答：对Grid和Table下的子物体进行排序和定位

64、请简述NGUI中Panel和Anchor的作用

答：

只要提供一个half-pixel偏移量，它可以让一个控件的位置在Windows系统上精确的显示出来（只有这个Anchor的子控件会受到影响）

如果挂载到一个对象上，那么他可以将这个对象依附到屏幕的角落或者边缘

UIPanel用来收集和管理它下面所有widget的组件。通过widget的geometry创建实际的draw call。没有panel所有东西都不能够被渲染出来,你可以把UIPanel当做Renderer

Unity面试手册：Unity应届生面试题

第一部分 判断题

- 1 C#支持继承多个类，达到重用代码功能的效果。 (×)
- 2 修改Renderer的sharedMaterial，所有使用这个材质球的物体都会被改变，并且也改变储存在工程里的材质设置。 (√)
- 3 Unity中可以创建子线程，并在子线程中直接修改UI对象。 (×)
- 4 Unity不支持在协程中嵌套调用协程。 (×)
- 5 C#不同命名空间中可以存在相同类名。 (√)
- 6 Unity会自动为MonoBehaviour子类的public变量做序列化。 (√)
- 7 每个枚举成员均具有相关联的常数值，可以设置为负数常数。 (√)
- 8 只带有 get 访问器的属性称为只读属性，无法对只读属性赋值。 (√)
- 9 protected成员只能被本类内部访问，无法被子类直接访问。 (×)
- 10 父物体发生Transform变化的时候，子物体跟随一起变化，但是子物体发生变化的时候，父物体不动。 (√)

第二部分 填空题

- 1 Unity中 (Game) 视图可以设置分辨率，在该视图中呈现的就是摄像机渲染的画面。
- 2 gameObject.AddComponent()的时候，Test脚本的 (Awake) 函数会立即被调用。
- 3 任何游戏对象在创建的时候都会附带 (Transform) 组件,用于储存并操控物体的位置、旋转和缩放。

- 4 只在编辑器环境下运行的代码，可以使用（UNITY_EDITOR） 宏把代码包起来。
- 5 Unity中可用四元数Quaternion表示 （旋转） ， 不受万向锁影响， 可以进行插值运算。
- 6 Unity协程中可以使用（ yield return null ） 实现暂缓一帧， 在下一帧接着往下处理。
- 7 transform.forward表示物体的（z） 轴的方向。
- 8 C#中的委托类似于C/C++中的 （函数指针） ,委托类型的声明以 delegate 关键字开头。
- 9 Unity中的 （Plugins） 目录用于放置Native插件文件， Android平台的jar文件必须放置在 （Assets/Plugins/Android/libs） 目录中。
- 10 在移动平台， Resources目录中的资源通过 （Resources.Load） 接口来加载， 如果想实现资源增量更新， 则一般考虑把资源打包成 （AssetBundle） 资源类型。
- 11 定义对象间的一种一对多依赖关系， 使得每当一个对象状态发生改变时， 其相关依赖对象皆得到通知并被自动更新， 可以使用 （观察者 设计模式）
- 12 Unity中每个材质球必须绑定一个 （shader（脚本））， 它决定了该材质的渲染方式以及可配置属性。
- 13 Unity中 （StreamingAssets ） 文件夹是只读的， 里面的所有文件将会被原封不动地复制制到目标平台机器上的特定文件夹里， 不会被压缩。在Android或iOS平台， 通过 （WWW） 类来读取其中的文件。
- 14 当场景中有多个摄像机时， 可以设置摄像机的 （depth） 值， 调整相机的渲染顺序。
- 15 为了加快渲染速度和减少图像锯齿， 贴图被处理成由一系列被预先计算和优化过的图片组成的文件， 这样的贴图被称为 （MipMap） 。

第三部分 问答题

1、C#中的委托是什么？

```
delegate int MyDelegate(int value); //声明委托类型
```

C#所有的委托派生自 System.Delegate 类，委托是存有对某个方法的引用的一种引用类型变量，委托变量可以当作另一个方法的参数来进行传递，实现事件和回调方法。

有点类似C中的函数指针，但是又有所不同。在C中，函数指针不是类型安全的，它指向的是内存中的某一个位置，我们无法判断这个指针实际指向什么，对于参数和返回类型难以知晓。

而C#的委托则完全不同，它是类型安全的，我们可以清晰的知道委托定义的返回类型和参数类型。

委托和事件：

本质区别：从定义上说，委托被编译器编译成一个类，所以它可以像类一样在任何地方定义，而事件被编译成一个委托类型的私有字段和两个公有add 和 remove 方法（有点类似于属性的定义）不过这两个方法都有一个参数，这个参数就是委托，所以，它只能定义在一个类里面。

```
event MyDelegate myevent; //定义事件
```

委托相当于一系列函数的抽象类，这一系列函数要求拥有相同的参数和返回值；而事件（event）相当于委托的一个实例，事件是委托类型的成员，委托可以定义在类外面，而事件只能定义在类里面。

事件使用 发布-订阅（publisher-subscriber）模型。

发布者（publisher）是一个包含事件和委托定义的对象。事件和委托之间的联系也定义在这个对象中。发布者（publisher）类的对象调用这个事件，并通知其他的对象。

订阅器（subscriber）是一个接受事件并提供事件处理程序的对象。在发布者（publisher）类中的委托调用订阅器（subscriber）类中的方法（事件处理程序）。

为什么需要事件？

事件最常用的应用场景是图形用户界面（GUI），如一个按钮点击事件，菜单选择事件，文件传输完成事件等。简单的说，某件事发生了，你必须要作出响应。你不能预测事件发生的顺序。只能等事件发生，再作出相应的动作来处理。触发事件的类本身对怎样处理事件不感兴趣。按钮说：“我被点过了”，响应类作出合适的响应。

2、值类型与引用类型的区别？

- 1.值类型存储在栈（stack）中，引用类型数据存储在堆（heap）中，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。

数据结构的堆和栈：

堆和栈都是一种数据项按序排列的数据结构。

栈就像装数据的桶，具有后进先出性质；堆像一棵倒过来的树，堆是一种经过排序的树形数据结构，每个结点都有一个值。堆的存取是随意，这就如同我们在图书馆的书架上取书，虽然书的摆放是有顺序的，但是我们想取任意一本时不必像栈一样，先取出前面所有的书。

内存结构：

栈中分配局部变量空间；

堆区是向上增长的用于分配程序员申请的内存空间；

静态区是分配静态变量、全局变量空间的；

只读区是分配常量和程序代码空间的；

3、接口Interface与抽象类abstract class的区别？

接口和抽象类是支持抽象定义两种机制。

接口是完全抽象的，只能声明方法，而且只能声明public的方法，不能声明private及protected的方法，不能定义方法体，也不能声明实例变量。抽象类是可以有私有的方法或者私有的变量，如果一个类中有抽象方法，那么就是抽象类。

一个类可以实现多个接口，但一个类只能继承一个抽象类。

接口强调特定功能的实现，具有哪些功能，而抽象类强调所属关系。

尽管接口实现类及抽象类的子类都必须要实现相应的抽象方法，但实现的形式不同。接口中的每一个方法都是抽象方法，都只是声明的，没有方法体，实现类必须都要实现；而抽象类的子类可以有选择地实现，只实现其中的抽象方法，覆盖其中已实现了的方法。

4.如何弱化代码依赖关系？

在代码的控制流中，调用关系和依赖关系几乎是完全吻合的，如果缺乏良好的封装与接口提取，那么调用者必须掌握被调用者的代码实现。

而抽象良好的接口，能够使控制流对代码的依赖实现反转，比如面向同一个接口协议，被调用者需要在协议的约束下对提供的服务进行实现，它的代码依赖协议的制定，而调用者只用依据协议按需获取服务即可，在控制流上依赖接口，而不再需要在代码上依赖被调用者，此即是从接口到被调用者的控制流—代码依赖关系反转。

代码依赖关系弱化，意味着业务可以模块化、组件化，拆分的功能组团可以以“插件”的方式并行独立开发维护，这种隔离大大提升开发运维效率，同时独立部署的能力也更加符合软硬件发展的趋势。

5、Unity实现跨平台的原理？

Unity的跨平台技术是通过一个Mono虚拟机实现的。就是通过Mono将C#脚本代码编译成CIL，然后Mono运行时利用JIT或者AOT将CLI编译成目标平台的原生代码实现的。

不过这个虚拟机更新太慢，不能很好地适应众多的平台，所以后来推出了IL2CPP，把本来应该再mono的虚拟机上跑的中间代码转换成cpp代码，这样再把生成的cpp代码，利用c的跨平台特性，在各个平台上通过对各平台都有良好优化的native c编译器编译，以获得更高的效率和更好的兼容性。

IL是.NET框架中间语言（Intermediate Language）的缩写。使用.NET框架提供的编译器可以直接将源程序编译为.exe或.dll文件，但此时编译出来的程序代码并不是CPU能直接执行的机器代码，而是一种中间语言IL（Intermediate Language）。

使用中间语言的优点有两点，一是可以实现平台无关性，既与特定CPU无关；二是只要把.NET框架某种语言编译成IL代码，就实现.NET框架中语言之间的交互操作(这就是为什么unity3D里面可以c#和js混编)。

在Mac OS上，因为iOS的现有限制，面向iOS的C#代码会通过AOT编译技术直接编译为ARM汇编代码。而在Android上，应用程序会转换为IL，启动时再进行JIT编译。

讲讲JIT：

JIT：即时编译（Just In-Time compile），这是.NET运行可执行程序的基本方式，编译一个.NET程序时，编译器将源代码翻译成中间语言，它是一组可以有效地转换为本机代码且独立于CPU的指令。

当执行这些指令时，实时（JIT）编译器将它们转化为CPU特定的代码。部分加密软件通过挂钩JIT来进行IL加密，同时又保证程序正常运行。JIT也会将编译过的代码进行缓存，而不是每一次都进行编译。所以说它是静态编译和解释器的结合体。

AOT：静态编译，它在程序运行之前就编译好了。

6、四元数的作用？

四元数用于表示旋转。

其相对于欧拉角的优点：

- 1.避免万向锁。
- 2.只需要一个4维的四元数就可以执行绕任意过原点的向量的旋转，方便快捷，在某些实现下比旋转矩阵效率更高。
- 3.可以提供平滑插值。

7.什么是欧拉角？

用一句话说，欧拉角就是物体绕坐标系三个坐标轴(x,y,z轴)的旋转角度。

- 1，静态：即绕世界坐标系三个轴的旋转，由于物体旋转过程中坐标轴保持静止，所以称为静态。
- 2，动态：即绕物体坐标系三个轴的旋转，由于物体旋转过程中坐标轴随着物体做相同的转动，所以称为动态。
- 物体的任何一种旋转都可分解为分别绕三个轴的旋转，但分解方式不唯一。

unity 3D欧拉角的旋转顺序（父子关系）是y-x-z。

unity中最简单的万向锁就是先让X轴旋转90度，z轴旋转和y轴旋转效果是一样。

讲讲万向锁：

万向锁（英语：Gimbal lock）是在使用动态欧拉角表示三维物体的旋转时出现的问题。

万向节死锁的根本问题是欧拉角（EulerAngles）保存的信息不足以描述空间中的唯一转向。

8、Unity脚本生命周期与执行顺序？

名称	触发时机	用途
Awake	脚本实例被创建时调用	用于游戏对象的初始化，注意Awake的执行早于所有脚本的Start函数
OnEnable	当对象变为可用或激活状态时被调用	
Start	Update函数第一次运行之前调用	用于游戏对象的初始化
Update	每帧调用一次	用于更新游戏场景和状态
FixedUpdate	每个固定物理时间间隔调用一次	用于物理状态的更新
LateUpdate	每帧调用一次（在update之后调用）	用于更新游戏场景和状态，和相机有关的更新一般放在这里
OnGUI	渲染和处理OnGUI事件	
OnDisable	当前对象不可用或非激活状态时被调用	
OnDestroy	当前对象被销毁时调用	

9、讲讲你对Unity的协程的理解？

协程不是线程。协程的实现原理是迭代器，而迭代器的实现原理是状态机。

unity中协程执行过程中，通过 yield return XXX，将程序挂起，去执行接下来的内容。在遇到 yield return XXX语句之前，协程方法和一般的方法是相同的，也就是程序在执行到 yield return XXX语句之

后，接着才会执行的是 StartCoroutine()方法之后的程序，走的还是单线程模式，仅仅是将 yield return XXX语句之后的内容暂时挂起，等到特定的时间才执行。

那么挂起的程序什么时候才执行？协同程序主要是Update()方法之后，LateUpdate()方法之前调用的。

通过设置MonoBehaviour脚本的enabled对协程是没有影响的，但如果gameObject.SetActive(false)则已经启动的协程则完全停止了，即使在Inspector把gameObject激活还是没有继续执行。也就是说协程虽然是在MonoBehaviour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。

第四部分 场景题

1、现在打出的Android包启动闪退，应该怎么定位问题？

使用ADB真机调试，通过日志定位问题。

2、现在要开发一个点击屏幕开炮发射子弹的功能，说下你的做法？

首先把子弹进行抽象，把属性和行为方法提炼出来，比如具有速度、威力、碰撞大小等属性，具有飞行、碰撞和伤害等行为。

封装子弹的抽象类，可以不继承MonoBehaviour。

监听屏幕点击事件，触发开炮逻辑。子弹通过对象池管理，复用子弹，防止因为频繁创建销毁带来的性能问题。

另外，子弹的坐标更新，可以统一由一个弹道控制器的Update遍历每个子弹对象来计算，而不是每个子弹都挂一个MonoBehaviour去更新，因为MonoBehaviour的Update是通过反射被调用的，如果有1000颗子弹，就会调用1000次反射，这样性能上比较差。

Unity面试手册：Unity游戏公司真题

1.Unity中碰撞器(Collider)和触发器(Trigger)的区别?

碰撞器(Collider)有碰撞效果, IsTrigger=false, 可以调用OnCollisionEnter/Stay/Exit函数

触发器(Trigger)没有碰撞效果, isTrigger=true, 可以调用OnTriggerEnter/Stay/Exit函数

2.物体发生碰撞的必要条件?

必须带有collider碰撞器和rigidbody刚体属性或者人物控制器, 其实人物控制器就包含了前两者, 另外一个物体也要必须带有Collider, Collider分类: 网格碰撞器, 盒子碰撞器, 胶囊碰撞器, 球型碰撞器, 地形碰撞器!

3.CharacterController和Rigidbody的区别?

Rigidbody具有完全真实物理的特性, 而CharacterController可以说是受限的Rigidbody, 具有一定的物理效果但不是完全真实的。

4.物体发生碰撞时, 有几个阶段, 分别对应的函数?

三个阶段, OnCollisionEnter/Stay/Exit函数

5.物体发生碰撞时, 几种施加压力的方式, 描述出来?

rigidbody.AddForce/AddForceAtPosition,都是rigidbody的成员函数

6.Unity3d提供了几种光源, 分别是什么?

共4种, DirectionalLight、PointLight、SpotLight、AreaLight (用于烘焙)

7.物理更新一般在哪个系统函数里？

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，如果你的渲染帧效率低下的时候FixedUpdate调用次数就会跟着下降。FixedUpdate比较适合用于物理引擎的计算，因为是跟每帧的渲染有关。Update就比较适合做控制。

8.移动相机动作在哪个函数里，为什么在这个函数里。

LateUpdate，是在所有的update结束后才调用，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，都是所有的update操作完才进行摄像机的跟进，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。

9.简述一下Prefab的用处？

在游戏运行时实例化，prefab相当于一个模板，对你已经有的素材、脚本、参数做一个默认的配置，以便于以后的修改，同事prefab打包的内容简化了导出的操作，便于团队的交流。

10.简述一下对象池,你觉得在FPS游戏里哪些东西适合使用对象池？

对象池就存放需要被反复调用资源的一个空间，比如游戏中要常被大量复制的对象，子弹，敌人，以及任何重复出现的对象。

Unity面试手册： Unity官方认证考试模拟题

1. 以下哪一个选项不属于Unity引擎所支持的视频格式文件？

- A.后缀名为mov的文件
- B.后缀名为mpg的文件
- C.后缀名为avi的文件
- D.后缀名为swf的文件

2.HDR高动态光照渲染属于下列哪个选项的属性？

- A.Lightmapping视图
- B.Light Probe组件
- C.Occlusion Culling视图
- D.Camera组件

3. Unity引擎的中，以下对Mesh Renderer组件描述正确的是哪一项？

- A.Mesh Renderer组件决定了场景中游戏对象的位置，旋转和缩放。
- B.为场景中的某一游戏对象增添物理的特性，需要为该游戏对象添加Mesh Renderer组件。
- C.Mesh Renderer组件从Mesh Filter组件中获得网格信息，并根据物体的Transform组件所定义的位置进行渲染。
- D.Mesh Renderer是从网格资源中获取网格信息的组件。

4. 当一个物体在视野内被其它物体遮挡，不希望对该物体进行渲染，可以通过以下哪一个模块实现该功能？

- A. Occlusion Culling
- B. NavMesh
- C. Light Probes
- D. Culling Mask

5. 制作Skybox时，为了避免出现纹理拼接时产生的接缝，这些纹理的循环方式应该设置为以下哪一项？

- A. Repeat
- B. Clamp
- C. Trilinear
- D. Bilinear

6. 下列哪个菜单可以用于打开设置发布程序选项的面板

- A. General Settings → Layers → Collision
- B. Edit → Render Settings
- C. Edit → Project Settings → Player
- D. Edit → Preferences

7. Unity引擎使用的是左手坐标系还是右手坐标系？

- A. 左手坐标系
- B. 右手坐标系
- C. 可以通过Project Setting切换左右手坐标系
- D. 可以通过Reference切换左右手坐标系

8.对摄像机Rendering Path的设置中，以下哪一项可以使Spot Light对物体光照产生的阴影以实时的方式渲染？

- A. Vertex Lit
- B. Forward
- C. Deferred Lighting
- D. 以上都可以

9. 以下哪组摄像机中Normalized View Port Rect的数值设置可以使得摄像机显示的画面位于1280*720分辨率的屏幕画面的右上角。

- A. X=640, Y=-360, W=640, H=360
- B. X=640, Y=0, W=640, H=360
- C. X=0.5, Y=0, W=0.5, H=0.5
- D. X=0.5, Y=0.5, W=0.5, H=0.5

10. 以下哪个组件是任何Game Object必备的组件？

- A. Mesh Renderer
- B. Transform
- C. Game Object
- D. Main Camera

11. 在Unity编辑器中，停止对Game视图进行预览播放的快捷键操作是以下哪一项？

- A. CTRL/CMD + P
- B. CTRL/CMD + SHIFT + P

C. CTRL/CMD + Alt + S

D. CTRL/CMD+S

12. 在对2D纹理的设置中，什么用途的纹理通常可以不强制使用2次幂的宽高数值？

A. 用于制作天空盒的纹理

B. 用于UI元素的纹理

C. 用于三维模型贴图的纹理

D. 用作Cookie贴图的纹理

13. 在Unity引擎中， Collider所指的是什么？

A. Collider是Unity引擎中所支持的一种资源，可用作存储网格信息

B. Collider是Unity引擎中内置的一种组件，可用作对网格进行渲染

C. Collider是Unity引擎中所支持的一种资源，可用作游戏对象的坐标转换

D. Collider是Unity引擎中内置的一种组件，可用作游戏对象间的碰撞检测

14. 可以通过以下哪个视图来录制场景中Game Object的动画？

A. Mecanim

B. Animation

C. Animator

D.Navigation

15. 在Unity工程的一个场景中，需控制多个摄像机的渲染画面的前后层次，可以通过Camera组件中哪个选项来进行设置。

A. Field of view

B. Depth

C. Clear Flags

D. Rendering Path

16. 如果想让一个Game Object在受到Directional Light的照射下对场景中的其它Game Object产生阴影，应该开启该Game Object上哪种组件的哪种属性？

A. Camera组件上的“Receive Shadows”属性

B. Mesh Render组件上的“Cast Shadows”属性

C. Collider上的“is Trigger”属性

D. Camera组件上的“Culling Mask”属性

17. 如何在Unity引擎中为场景创建一个Shuriken粒子系统？

A. 依次点击菜单栏：Game Object -> Create Other -> Particle System

B.在Project窗口，依次点击Create -> Particle System，创建后拖入场景中

C.在Hierarchy视窗中空白处右键，在右键菜单中依次选择：Create -> Particle System

D.以上都可以

18. 下列选项中有关Animator的说法错误的是？

A.Animator是Unity引擎中内置的组件

- B.任何一个具有动画状态机功能的Game Object都需要一个Animator组件
- C.它主要用于角色行为的设置，包括State Machines、混合树Blend Trees以及通过脚本控制的事件
- D.Animator同Animation组件的用法是相同的

19. 在Unity引擎中，关于如何向工程中导入图片资源，以下做法错误的是？

- A. 将图片文件复制或剪切到项目文件夹下的Assets文件夹或Assets子文件夹下
- B. 通过Assets->Import New Asset....导入资源
- C. 选中所需图片，按住鼠标左键拖入Project视图中
- D.选中所需图片，按住鼠标左键拖入Scene视图中

20. Unity引擎中，可通过下列哪个步骤创建Animator Controller？

- A. Game Object->Animator Controller
- B. Component->Animator Controller
- C. 在Hierarchy视图中点击Create->Animator Controller
- D. 在Project视图中点击Create->Animator Controller

21. 下列选项中，关于Transform组件的Scale参数的描述正确？

- A.Transform组件的Scale参数不会影响ParticleSystem产生粒子的大小
- B.Transform组件的Scale参数不会影响GUITexture的大小
- C.添加Collider组件后的GameObject，其Collider组件的尺寸不受Transform组件的Scale参数影响

D.添加Rigidbody组件后的物体，大小将不再受Transform组件中Scale参数的影响

22. Mecanim 动画系统非常适合用于人形角色动画的制作。当导入一个FBX的人形模型后，如需要对该模型做人形动画骨骼重定向的话，则需要将动画类型设置为以下哪一项？

A. Humanoid

B. Generic

C. Legacy

D. Auto

23. 在Unity中的场景中创建Camera时，默认情况下除了带有Transform、Camera、GUI Layer、Flare Layer组件之外，还带有以下哪种组件？

A. Mouse Look

B. FPS Input Controller

C. Audio Listener

D. Character Motor

24. 如果将一个声音剪辑文件从Project视图拖动到Inspector视图或者Scene视图中的游戏对象上，则该游戏对象会自动添加以下哪种组件？

A. Audio Listener

B. Audio Clip

C.Audio Source

D.Audio Reverb Zone

25. 在Unity工程的Project视图中选中一张图片，此时在Inspector视图中Preview窗口中显示的图片的大小表示的是以下哪一项？

- A. 图片实际占用外存的大小
- B. 当工程运行时，该图片在场景中所占用显存的大小
- C. 对图片的格式，压缩后所占用外存的大小
- D. 以上均不正确

26. 下列哪个视图主要用于显示和编辑所选游戏对象或资源的相关属性？

- A. Scene
- B. Project
- C. Inspector
- D. Hierarchy

27. 在场景中需要使一个带有网格的Game Object接受来自Light Probes的环境光照，需要对下列选项中哪个组件的Use Light Probes属性进行设置？

- A. Mesh Filter
- B. Mesh Collider
- C. Mesh Renderer
- D. Material

28. 下列叙述中有关Prefab说法错误的是哪一项？

- A. Prefab是一种资源类型

- B. Prefab是一种可以反复使用的游戏对象
- C. Prefab可以多次在场景进行实例
- D. 当一个Prefab添加到场景中时，也就是创建了一个实例

29. 以下哪一类型的纹理能够在渲染时为三维模型的表面增加凹凸细节？

- A. Normal map
- B. Cube map
- C. Light map
- D. Specular map

30. 若要给Sphere游戏对象添加Physics Material，需要使用下列选项中的哪个组件的Material属性？

- A. Mesh Renderer
- B. Mesh Filter
- C. Sphere Collider
- D. Rigidbody

Unity面试手册：C#基础语法

1. 字符串中string str=null和string str=""和string str=string.Empty的区别?

string.Empty相当于"",Empty是一个静态只读的字段。 string str="" ,初始化对象, 并分配一个空字符串的内存空间 string str=null,初始化对象, 不会分配内存空间

2. byte b = 'a'; byte c = 1; byte d = 'ab'; byte e = '啊'; byte g = 256; 这些变量有些错误是错再哪里?

本题考查的是数据类型能承载数据的大小。

1byte =8bit, 1个汉字=2个byte, 1个英文=1个byte=8bit

所以bc是对的, deg是错的。'a'是char类型, a错误

java byte取值范围是-128~127, 而C#里一个byte是0~255

3.string和StringBuilder的区别,两者性能的比较

都是引用类型, 分配再堆上

StringBuilder默认容量是16, 可以允许扩充它所封装的字符串中字符的数量.每个StringBuffer对象都有一定的缓冲区容量, 当字符串大小没有超过容量时, 不会分配新的容量, 当字符串大小超过容量时, 会自动增加容量。

对于简单的字符串连接操作, 在性能上stringbuilder不一定总是优于strin因为stringbulider对象的创建也消耗大量的性能, 在字符串连接比较少见的情况下, 过度滥用stringbuilder会导致性能的浪费而非节约, 只有大量无法预知次数的字符串操作才考虑stringbuilder的使用。从最后分析可以看出如果是相对较少的字符串拼接根本看不出太大差别。

Stringbulider的使用, 最好制定合适的容量值, 否则优于默认值容量不足而频繁的进行内存分配操作, 是不妥的实现方法。

参考链接: <https://www.cnblogs.com/haofuqi/p/4826262.html>

4.什么是扩展方法?

一句话解释，扩展方法使你能够向现有类型“添加”方法，无需修改类型

条件：按扩展方法必须满足的条件，1.必须要静态类中的静态方法2.第一个参数的类型是要扩展的类型，并且需要添加this关键字以标识其为扩展方法

建议：通常，只在不得已的情况下才实现扩展方法，并谨慎的实现

使用：不能通过类名调用，直接使用类型来调用

5.byte a =255;a+=5;a的值是多少？

byte的取值范围是-2的8次方至2的8次方-1，-256至258，a+=1时，a的值是0，a+=5时，a的值是0，所以a+=5时，值是4

6.什么是装箱和拆箱？

装箱就是隐式地将一个值类型转换成引用类型，如：

▼ Plain Text 复制代码

```
1  int i=0;
2  System.Object obj=i;
```

拆箱就是将引用类型转换成值类型，如：

▼ Plain Text 复制代码

```
1  int i=0;
2  System.Object obj=i;
3  int j=(int)obj; (将obj拆箱)
```

7.值类型和引用类型的区别

值类型变量是直接包含值。将一个值类型变量赋给另一个值类型变量，是复制包含的值，默认值是0。

引用类型变量的赋值只复制对对象的引用，而不复制对象本身，默认值是null
值类型有整形、浮点型、bool、枚举。

引用类型有class、delegate、Object、string

值类型存储在栈中，引用类型存储在堆中

8.new关键字的作用？

运算符：创建对象实例

修饰符：在派生类定义一个重名的方法，隐藏掉基类方法

约束：泛型约束定义，约束可使用的泛型类型,如：

▼ Plain Text | 复制代码

```
1 public class ItemFactory<T> where T : IComparable, new()  
2 {  
3 }
```

9. int?和int有什么区别？

int? 为可空类型，默认值可以是null

int默认值是0

int?是通过int装箱为引用类型实现

10. C#中的委托是什么？

一句话解释就是：将方法当作参数传入另一个方法的参数。 .net中有很多常见的委托如：Func 、
Action 作用：提高方法的扩展性

11.用最有效的方法算出2乘以8等于几?

位运算是最快，使用的是位运算 逻辑左位移<<。方法是 $2 \ll 3$ 相当于0000 0000 0000 0010（2的16位int二进制）左移三位就是 0000 0000 0001 0000（16的二进制）

12.const和readonly有什么区别?

都可以标识一个常量。主要有以下区别：

- 1、初始化位置不同。const必须在声明的同时赋值；readonly即可以在声明处赋值，也可以在静态构造方法（必须是静态构造方法，普通构造方法不行）里赋值。
- 2、修饰对象不同。const即可以修饰类的字段，也可以修饰局部变量；readonly只能修饰类的字段
- 3、const是编译时常量，在编译时确定该值；readonly是运行时常量，在运行时确定该值。
- 4、const默认是静态的；而readonly如果设置成静态需要显示声明
- 5、修饰引用类型时不同，const只能修饰string或值为null的其他引用类型；readonly可以是任何类型。

13.现有一个整数number，请写一个方法判断这个整数是否是2的N次方

4 (100) 、5 (101) 、8 (1000) 、16 (10000)

取模运算：用 $\text{number} \% 2 == 0$ 可以判断，但是这个有点低级

位运算：（使用位运算逻辑并，两个位上的都为1才是1，其余都是0，判断是否等于0）

$4 \& 3$ 相当于 $100 \& 011$ ，结果是000等于0，所以4是2的n次方

$5 \& 4$ 相当于 $101 \& 100$ ，结果是100不等于0，所以5不是2的n次方

如果要问如果是2的N次方，这个N是多少？这该怎么算？

▼ Plain Text 复制代码

```
1 private static byte get(int n)
2     {
3         byte number = 1;
4         while (n/2!=1)
5         {
6             n = n / 2;
7             number += 1;
8         }
9         return number;
10    }
```

14.CTS、CLS、CLR分别作何解释

CTS：通用语言系统。CLS：通用语言规范。CLR：公共语言运行库。

CTS：Common Type System 通用类型系统。Int32、Int16→int、String→string、Boolean→bool。每种语言都定义了自己的类型，.Net通过CTS提供了公共的类型，然后翻译生成对应的.Net类型。

CLS：Common Language Specification 通用语言规范。不同语言语法的不同。每种语言都有自己的语法，.Net通过CLS提供了公共的语法，然后不同语言翻译生成对应的.Net语法。

CLR：Common Language Runtime 公共语言运行时，就是GC、JIT等这些。有不同的CLR，比如服务器CLR、Linux CLR（Mono）、Silverlight CLR(CoreCLR)。相当于一个发动机，负责执行IL。

15.在.net中，配件的意思是？

程序集。（中间语言，源数据，资源，装配清单）

16.分析下面代码，a、b的值是多少？

▼ Plain Text 复制代码

```
1 string strTmp = "a1某某某";
2 int a = System.Text.Encoding.Default.GetBytes(strTmp).Length;
3 int b = strTmp.Length;
```

分析：一个字母、数字占一个byte，一个中文占两个byte，所以a=8,b=5

17.Strings = new String("xyz");创建了几个String Object?

两个对象，一个是“xyz”，一个是指向“xyz”的引用对象s。

18.静态成员和非静态成员的区别

- 1.静态成员用static修饰符声明，在类被实例化时创建，通过类进行访问
- 2.不带static的变量是非静态变量，在对象被实例化时创建，通过对象进行访问，
- 3.静态方法里不能使用非静态成员，非静态方法可以使用静态成员
- 4.静态成员属于类，而不属于对象

19.c#可否对内存直接操作

C#在unsafe 模式下可以使用指针对内存进行操作，但在托管模式下不可以使用指针，C#NET默认不运行带指针的，需要设置下，选择项目右键->属性->选择生成->“允许不安全代码”打勾->保存

20.short s1 = 1; s1 = s1 + 1;有什么错? short s1 = 1; s1 += 1;有什么错?

s1+1不能显式转换成short类型，可以修改为s1 =(short)(s1 + 1) 。short s1 = 1; s1 += 1正确

21.什么是强类型，什么是弱类型？哪种更好些？为什么？

强类型是在编译的时候就确定类型的数据，在执行时类型不能更改，而弱类型在执行的时候才会确定类型。没有好不好，二者各有好处，强类型安全，因为它事先已经确定好了，而且效率高。

一般用于编译型编程语言，如c++,java,c#,pascal等,弱类型相比而言不安全，在运行的时候容易出现错误，但它灵活，多用于解释型编程语言，如javascript,vb,php等

22.using关键字的作用

1.引用命名空间，也可using 别名

2.释放资源，实现了IDisposable的类在using中创建，using结束后会自定调用该对象的Dispose方法，释放资源。

23.ref和out有什么区别

1.都是按引用类型进行传递

2.属性不是变量不能作为out、ref参数传递

3.ref参数必须初始化。out不需要初始化

4.作用，当方法有多个返回值时，out非常有用

24.a.Equals(b)和a==b一样吗？

不一样，a==b仅仅表示a和b值相等，a.Equals(b)表示a与b一致

25.下面这段代码求值

```
1  class Class1
2  {
3      internal static int count = 0;
4      static Class1()
5      {
6          count++;
7      }
8      public Class1()
9      {
10         count++;
11     }
12 }
13 Class1 o1 = new Class1();
14 Class1 o2 = new Class1();
```

o1.count的值是多少？

答案：3，静态 构造方法计算一次，两个实例化对象计算两次。

26.关于构造函数说法正确的是哪个？

- a)构造函数可以声明返回类型。
- b)构造函数不可以用private修饰
- c)构造函数必须与类名相同
- d)构造函数不能带参数

答案：c，构造函数必须与类名相同，可以传递多个参数，作用就是便于初始化对象成员，不能有任何返回类型

27.Math.Round(11.5)等於多少？ Math.Round(-11.5)等於多少？

Math.Round(11.5)=12

Math.Round(-11.5)=-12

28.&和&&的区别

相同点

&和&&都可作逻辑与的运算符，表示逻辑与（and），当运算符两边的表达式的结果都为true时，其结果才为true，否则，只要有一方为false，则结果为false。（ps：当要用到逻辑与的时候&是毫无意义，&本身就不是干这个的）

不同点

...

```
if(loginUser!=null&&string.IsNullOrEmpty(loginUser.UserName))
```

...

&&具有短路的功能，即如果第一个表达式为false，则不再计算第二个表达式，对于上面的表达式，当loginUser为null时，后面的表达式不会执行，所以不会出现NullPointerException如果将&&改为&，则会抛出NullPointerException异常。（ps：所以说当要用到逻辑与的时候&是毫无意义的）

& 是用作位运算的。

总结

&是位运算，返回结果是int类型 &&是逻辑运算，返回结果是bool类型

29. i和i有什么区别？

1.i++是先赋值，然后再自增；++i是先自增，后赋值。

2.i=0, i++=0, ++i=1; Console.WriteLine(i==i); 结果位true

30.as和is的区别

as在转换的同时判断兼容性，如果无法进行转换，返回位null（没有产生新的对象），as转换是否成功判断的依据是是否位null is只是做类型兼容性判断，并不执行真正的类型转换，返回true或false，对象为null也会返回false。

as比is效率更高，as只需要做一次类型兼容检查

Unity面试手册：C#面向对象

1.什么是构造函数？

概念：构造函数的方法名与类型相同、没有返回类型

作用：完成对类的对象初始化

创建一个类的新对象时，系统会自动调用该构造函数初始化新对象，如果没有写定义，那么系统会自动提供一个不带任何参数的public 构造函数

2.class和struct的区别？

相同点 都可以实现接口

不同点

1.class是引用类型，struct是值类型

2.class允许继承、被继承，struct不允许，只能继承接口

3.class可以初始化变量，struct不可以

4.class可以有无参的构造函数，struct不可以，必须是有参的构造函数，而且在有参的构造函数必须初始化所有成员

使用场景

1.Class比较适合大的和复杂的数据，表现抽象和多级别的对象层次时。

2.Struct适用于作为经常使用的一些数据组合成的新类型，表示诸如点、矩形等主要用来存储数据的轻量级对象时，偏简单值。

3.Struct有性能优势，Class有面向对象的扩展优势。

3.简述一下面向对象的三大特性？

封装、继承、多态。

封装：是通过把对象的属性的实现细节隐藏起来，仅对外提供公共的访问方法。

继承：是通过子类继承基类、继承抽象类、继承接口实现的。

多态：是通过重写基类的override 方法、重写虚方法实现的。好处是：方便维护、易扩展。缺点是：比面向过程性能低。

4.构造函数是否能被重写？

构造器Constructor不能被继承，因此不能重写，但可以被重载

5.抽象类和接口有什么区别？

相同点：都不能直接实例化 不同点：

1.抽象类用abstract修饰、接口用interface修饰

2.抽象类中的方法可以实现，也可以不实现，有抽象方法的类一定要用abstract修饰，接口中的方法不允许实现

3.抽象类只能单继承，接口支持多实现

4.抽象类有构造方法，接口不能有构造方法

5.接口只负责功能的定义，通过接口来规范类的，（有哪些功能），而抽象类即负责功能的定义有可以实现功能（实现了哪些功能）

6..类的执行顺序？

执行顺序：父类，子类，静态块，静态字段，非静态块，非静态字段，构造器，方法

7.接口是否可继承接口？抽象类是否可实现（implements）接口？抽象类是否可继承实现类（concrete class）？

接口可以继承接口，抽象类可以实现接口，抽象类可以继承实现类，但前提是实现类必须有明确的构造函数。

8.继承最大的好处？

对父类成员进行重用，增加代码的可读性、灵活性。

9.请说说引用和对象？

对象和引用时分不开的，对象生成一个地址，放在堆里面，引用则指向这个地址，放在栈里面

10.什么是匿名类，有什么好处？

不用定义、没有名字的类，使用一次便可丢弃。好处是简单、随意、临时的。

11.重写和重载的区别？

重写方法：关键字用override修饰，派生类重写基类的方法，方法命名、返回类型，参数必须相同

重载方法：方法名必须相同，参数列表必须不相同，返回类型可以不相同。

作用：重写主要是实现面向对象的多态性、重载主要是实现实例化不同的对象

12.C#中有没有静态构造函数，如果有是做什么用的？

特点：静态构造函数既没有访问修饰符，也没有参数。在创建第一个实例或引用任何静态成员之前，将自动调用静态构造函数来初始化类。无法直接调用静态构造函数。在程序中，用户无法控制何时执行静态构造函数。用途：当类使用日志文件时，将使用这种构造函数向日志文件中写入项。

13.怎样理解静态变量？静态成员和非静态成员的区别？

静态变量属于类，而不属于对象；并对所有对象所享；静态成员在加类的时候就被加载。

14.属性能在接口中声明吗？

可以，不能有访问修饰符，不能初始化赋值。

15.在项目中为什么使用接口？接口的好处是什么？什么是面向接口开发？

接口是一种约束，描述类的公共方法/公共属性，不能有任何的实现

好处是：结构清晰，类之间通信简单易懂，扩展性好，提高复用性。

面向interface编程，原意是指面向抽象协议编程，实现者在实现时要严格按协议来办。

16.什么时候用重载？什么时候用重写？

当一个类需要用不同的实现来做同一件事情，此时应该用重写，而重载是用不同的输入做同一件事情

17.静态方法可以访问非静态变量吗？如果不可以为什么？

静态方法和非静态变量不是同一生命周期，静态方法属于类，非静态变量属于具体的对象，静态方法和具体的对象没有任何关联

18.在.Net中所有可序列化的类都被标记为_？

[serializable]

19.C#中 property 与 attribute的区别，他们各有什么用处，这种机制的好处在哪里？

一个是属性，用于存取类的字段，一个是特性，用来标识类，方法等的附加性质

20.当使用new B()创建B的实例时，产生什么输出？

▼ Plain Text 复制代码

```
1      using System;
2      class A
3      {
4          public A()
5          {
6              PrintFields();
7          }
8          public virtual void PrintFields(){}
9      }
10     class B:A
11     {
12         int x=1;
13         int y;
14         public B()
15         {
16             y=-1;
17         }
18         public override void PrintFields()
19         {
20             Console.WriteLine("x={0},y={1}",x,y);
21         }
22     }
```

答：X=1,Y=0;

21.能用foreach遍历访问的对象需要实现 接口或声明方法的类型

答：IEnumerable 、 GetEnumerator。

Unity面试手册：C#委托、集合、异常、泛型

1. IList 接口与List的区别是什么？

IList 泛型接口是 ICollection 接口的子代，并且是所有非泛型列表的基接口。Ilist 实现有三种类别：只读、固定大小、可变大小。无法修改只读 Ilist。固定大小的 Ilist 不允许添加或移除元素，但允许修改现有元素。可变大小的 Ilist 允许添加、移除和修改元素。

IList 是个接口,定义了一些操作方法这些方法要你自己去实现，当你只想使用接口的方法时,这种方式比较好.他不获取实现这个接口的类的其他方法和字段，有效的节省空间.

List 是个类型 已经实现了IList 定义的那些方法。



Plain Text

复制代码

```
1 List List11 =new List ();
```

是想创建一个List，而且需要使用到List的功能，进行相关操作。

而



Plain Text

复制代码

```
1 IList IList11 =new List ();
```

只是想创建一个基于接口IList的对象的实例，只是这个接口是由List实现的。所以它只是希望使用到IList接口规定的功能而已。

2.泛型的主要约束和次要约束是什么？

当一个泛型参数没有任何约束时，它可以进行的操作和运算是非常有限的，因为不能对实参进行任何类型上的保证，这时候就需要用到泛型约束。泛型的约束分为：主要约束和次要约束，它们都使实参必须满足一定的规范，C#编译器在编译的过程中可以根据约束来检查所有泛型类型的实参并确保其满足约束

条件。

(1) 主要约束

一个泛型参数至多拥有一个主要约束，主要约束可以是一个引用类型、class或者struct。如果指定一个引用类型（class），那么实参必须是该类型或者该类型的派生类型。相反，struct则规定了实参必须是一个值类型。下面的代码展示了泛型参数主要约束：

```
1  public class ClassT1<T> where T : Exception
2  {
3      private T myException;
4      public ClassT1(T t)
5      {
6          myException = t;
7      }
8      public override string ToString()
9      {
10         // 主要约束保证了myException拥有source成员
11         return myException.Source;
12     }
13 }
14
15 public class ClassT2<T> where T : class
16 {
17     private T myT;
18     public void Clear()
19     {
20         // T是引用类型，可以置null
21         myT = null;
22     }
23 }
24
25 public class ClassT3<T> where T : struct
26 {
27     private T myT;
28     public override string ToString()
29     {
30         // T是值类型，不会发生NullReferenceException异常
31         return myT.ToString();
32     }
33 }
```

(2)次要约束

次要约束主要是指实参实现的接口的限定。对于一个泛型，可以有0到无限的次要约束，次要约束规定了实参必须实现所有的次要约束中规定的接口。次要约束与主要约束的语法基本一致，区别仅在于提供的不是一个引用类型而是一个或多个接口。例如我们为上面代码中的ClassT3增加一个次要约束：

▼ Plain Text 复制代码

```
1 public class ClassT3<T> where T : struct, IComparable
2 {
3     .....
4 }
```

3. 如何把一个array复制到arraylist里？

▼ Plain Text 复制代码

```
1 foreach( object arr in array)
2 {
3     arraylist.Add(arr);
4 }
```

4.List, Set, Map是否继承自Collection接口？

List, Set是， Map不是

5. Set里的元素是不能重复的，那么用什么方法来区分重复与否呢？是用==还是equals()？它们有何区别？

Set里的元素是不能重复的，那么用iterator()方法来区分重复与否。equals()是判读两个Set是否相等。

equals()和==方法决定引用值是否指向同一对象，equals()在类中被覆盖，为的是当两个分离的对象的 内容和类型相配的话，返回真值。

6.有50万个int类型的数字，现在需要判断一下里面是否存在重复的数字，请你简要说一下思路。

1.使用C#的List集合自带的去重方法，例如 Distinct(), GroupBy()等

2.利用 Dictionary 的Key值唯一的特性, HashSet 元素值唯一的特性 进行判断

7.数组有没有length()这个方法? String有没有length()这个方法?

数组没有length()这个方法, 有length的属性。String有length()这个方法。

8.一个整数List中取出最大数(找最大值)。不能用Max方法。

```
1      private static int GetMax(List<int> list)
2      {
3          int max = list[0];
4          for (int i = 0; i < list.Count; i++)
5          {
6              if (list[i]>max)
7              {
8                  max = list[i];
9              }
10         }
11         return max;
12     }
```

9. C#异常类有哪些信息?

C#中, 所有异常都继承自System.Exception类, Exception类定义了C#异常应该具有的信息和方法。值得注意的属性有:

```
1  public virtual string Message { get; }// 错误的信息, 文字描述
2  public virtual string StackTrace { get; }// 发生异常的调用堆栈信息
3  public System.Reflection.MethodBase TargetSite { get; }//引发这个错误的方法
4  public Exception InnerException { get; }// 子异常
```

10. 如何创建一个自定义异常?

根据类继承原则和异常处理原则，我们可以使用以下方式来自定义一个类：



Plain Text

复制代码

```
1 public class CustomException : Exception
2 {
3 }
```

11. 利用IEnumerable实现斐波那契数列生成？



Plain Text

复制代码

```
1 IEnumerable<int> GenerateFibonacci(int n)
2 {
3     if (n >= 1) yield return 1;
4
5     int a = 1, b = 0;
6     for (int i = 2; i <= n; ++i)
7     {
8         int t = b;
9         b = a;
10        a += t;
11
12        yield return a;
13    }
14 }
```

12.请利用 foreach 和 ref 为一个数组中的每个元素加 1

注意 foreach 不能用 var ，也不能直接用 int ，需要 ref int ，注意 arr 要转换为 Span 。

```
1  int[] arr = { 1, 2, 3, 4, 5};
2  Console.WriteLine(string.Join(",", arr)); // 1,2,3,4,5
3
4  foreach (ref int v in arr.AsSpan())
5  {
6      v++;
7  }
8
9  Console.WriteLine(string.Join(",", arr)); // 2,3,4,5,6
```

13.如何针对不同的异常进行捕捉?

```
1  public class Program
2  {
3      public static void Main(string[] args)
4      {
5          Program p = new Program();
6          p.RiskWork();
7
8          Console.ReadKey();
9      }
10
11     public void RiskWork()
12     {
13         try
14         {
15             // 一些可能会出现异常的代码
16         }
17         catch (NullReferenceException ex)
18         {
19             HandleExpectedException(ex);
20         }
21         catch (ArgumentException ex)
22         {
23             HandleExpectedException(ex);
24         }
25         catch (FileNotFoundException ex)
26         {
27             HandlerError(ex);
28         }
29         catch (Exception ex)
30         {
31             HandleCrash(ex);
32         }
33     }
34
35     // 这里处理预计可能会发生的，不属于错误范畴的异常
36     private void HandleExpectedException(Exception ex)
37     {
38         // 这里可以借助log4net写入日志
39         Console.WriteLine(ex.Message);
40     }
41
42     // 这里处理在系统出错时可能会发生的，比较严重的异常
43     private void HandlerError(Exception ex)
44     {
45         // 这里可以借助log4net写入日志
```



```

46         Console.WriteLine(ex.Message);
47         // 严重的异常需要抛到上层处理
48         throw ex;
49     }
50
51     // 这里处理可能会导致系统崩溃时的异常
52     private void HandleCrash(Exception ex)
53     {
54         // 这里可以借助log4net写入日志
55         Console.WriteLine(ex.Message);
56         // 关闭当前程序
57         System.Threading.Thread.CurrentThread.Abort();
58     }
59 }

```

14.如何避免类型转换时的异常？

其中有些是确定可以转换的（比如将一个子类类型转为父类类型），而有些则是尝试性的（比如将基类引用的对象转换成子类）。当执行常识性转换时，我们就应该做好捕捉异常的准备。

当一个不正确的类型转换发生时，会产生InvalidCastException异常，有时我们会用try-catch块做一些尝试性的类型转换，这样的代码没有任何错误，但是性能却相当糟糕，为什么呢？异常是一种耗费资源的机制，每当异常被抛出时，异常堆栈将会被建立，异常信息将被加载，而通常这些工作的成本相对较高，并且在尝试性类型转换时，这些信息都没有意义。

在.NET中提供了另外一种语法来进行尝试性的类型转换，那就是关键字 is 和 as 所做的工作。

(1) is 只负责检查类型的兼容性，并返回结果：true 和 false。→ 进行类型判断

▼ Plain Text 复制代码

```
1 public static void Main(string[] args)
2 {
3     object o = new object();
4     // 执行类型兼容性检查
5     if(o is ISample)
6     {
7         // 执行类型转换
8         ISample sample = (ISample)o;
9         sample.SampleShow();
10    }
11
12    Console.ReadKey();
13 }
```

(2) as 不仅负责检查兼容性还会进行类型转换，并返回结果，如果不兼容则返回 null 。→ 用于类型转型

▼ Plain Text 复制代码

```
1 public static void Main(string[] args)
2 {
3     object o = new object();
4     // 执行类型兼容性检查
5     ISample sample = o as ISample;
6     if(sample != null)
7     {
8         sample.SampleShow();
9     }
10    Console.ReadKey();
11 }
```

两者的共同之处都在于：不会抛出异常！综上所述，as 较 is 在执行效率上会好一些，在实际开发中应该量才而用，在只进行类型判断的应用场景时，应该多使用 is 而不是 as。

15.委托是什么？

委托是寻址的.NET版本。在C++中，函数指针只不过是一个指向内存位置的指针，它不是类型安全的。我们无法判断这个指针实际指向什么，像参数和返回类型等项更无从知晓了。

而.NET委托完全不同，委托是类型安全的类，它定义了返回类型和参数的类型。委托类不仅包含对方法的引用，也可以包含对多个方法的引用。

16.如何自定义委托？

声明一个委托类型，它的实例引用一个方法，该方法获取一个int参数，返回void。

▼

Plain Text | 复制代码

```
1 public delegate void Feedback(int num);
```

理解委托的一个要点是它们的安全性非常高。在定义委托时，必须给出它所表示的方法的签名和返回类型等全部细节。

理解委托的一种比较好的方式是把委托当作这样一件事情：它给方法的签名和返回类型指定名称。

其语法类似于方法的定义，需要在定义方法的前面加上delegate关键字。定义委托基本上就是定义一个新的类，所以可以在任何地方定义类的相同地方定义委托，也就是说，可以在另一个类的内部定义，也可以在任何类的外部定义，还可以在名称控件中把委托定义为定义为顶层对象。访问修饰符可以是public/private/protected等。

17 .NET默认的委托类型有哪几种？

1. Action < T >

泛型Action委托表示引用一个void返回类型的方法。这个委托类存在16种重载方法。

例如Action<in T1,In T2>调用没有参数的方法

2.Func< T >

Func调用带返回类型的方法。有16种重载方法。

例如Func委托类型可以调用带返回类型且无参数的方法，Func<in T,out TResult>委托类型调用带有4个参数和一个返回类型的方法。

18.什么是泛型委托？

Action就是泛型委托。

注意事项：

1.建议尽量使用这些委托类型，而不是在代码中定义更多的委托类型。这样可以减少系统中的类型数目，同时简化编码

2.如果需要使用ref或out关键字，以传引用的方式传递一个参数，就可能不得不定义自己的委托：

```
delegate void Test(ref int i)
```

3.如果委托要通过C#的params关键字获取可变数量的额参数，要为委托的任何按树指定默认值，或者要对委托的泛型类型参数进行约束，也必须定义自己的委托类型

```
delegate void EventHandler(Object sender, TEventArgs e)
where TEventArgs : EventArgs;
```

4.使用获取泛型实参和返回值的委托时，可利用逆变与协变。逆变：父类转换为子类；协变：子类转换为父类

19. 什么事匿名方法？

匿名方法是用作委托的参数的一段代码。

```
1    //匿名方法,例1
2    Func<int, int> anon = delegate(int i)
3    {
4        i = i+1;
5        return i;
6    };
7    //输出2
8    Console.WriteLine(anon(1));
9    //匿名方法,例2
10   Action<int> anon2 = delegate(int i)
11   {
12       i = i + 1;
13   };
14   //输出2
15   Console.WriteLine(anon(1));
```

20.什么是闭包?

通过Lambda表达式可以访问Lambda表达式块外部的变量，这成为闭包。

当引用外部变量时，需要注意，外部变量变化时，lambda表达式的结果也可能会随着外部变量变化而变化。

如下面的例子：

```
1    int y = 5;
2    Func<int, int> lambda = x => x + y;
3    Console.WriteLine(lambda(1));
4    y = 10;
5    Console.WriteLine(lambda(1));
```

21.什么是协变和逆变?

可变性是以一种类型安全的方式，将一个对象作为另一个对象来使用。其对应的术语则是不变性 (invariant) 。

可变性：

可变性是以一种类型安全的方式，将一个对象作为另一个对象来使用。例如对普通继承中的可变性：若某方法声明返回类型为Stream，在实现时可以返回一个MemoryStream。可变性有两种类型：协变和逆变。

协变性：

可以建立一个较为一般类型的变量，然后为其赋值，值是一个较为特殊类型的变量。例如：

▼ Plain Text | 复制代码

```
1  string str = "test";
2  // An object of a more derived type is assigned to an object of a less derived type.
3  object obj = str;
```

因为string肯定是一个object，所以这样的变化非常正常。

逆变性：在上面的例子中，我们无法将str和一个新的object对象画等号。如果强行要实现的话，只能这么干：

▼ Plain Text | 复制代码

```
1  string s = (string) new object();
```

但这样还是会在运行时出错。这也告诉我们，逆变性是很不正常的。

泛型的协变与逆变：

协变性和out关键字搭配使用，用于向调用者返回某项操作的值。例如下面的接口仅有一个方法，就是生产一个T类型的实例。那么我们可以传入一个特定类型。如我们可以将IFactory视为IFactory。这也适用于Food的所有子类型。（即将其视为一个更一般类型的实现）

▼ Plain Text 复制代码

```
1 interface IFactory<T>
2 {
3     T CreateInstance();
4 }
```

逆变性则相反，和in关键字搭配使用，指的是API将会消费值，而不是生产值。此时一般类型出现在参数中：

▼ Plain Text 复制代码

```
1 interface IPrint<T>
2 {
3     void Print(T value);
4 }
```

这意味着如果我们实现了IPrint< Code >，我们就可以将其当做IPrint< CsharpCode >使用。（即将其视为一个更具体类型的实现）

如果存在双向的传递，则什么也不会发生。这种类型是不变体(invariant)。

▼ Plain Text 复制代码

```
1 interface IStorage<T>
2 {
3     byte[] Serialize(T value);
4     T Deserialize(byte[] data);
5 }
```

这个接口是不变体。我们不能将它视为一个更具体或更一般类型的实现。

假设有如下继承关系People → Teacher, People → Student。

如果我们以协变的方式使用（假设你建立了一个IStorage< Teacher >的实例，并将其视为IStorage）则我们可能会在调用Serialize时产生异常，因为Serialize方法不支持协变（如果参数是People的其他子类，例如Student，则IStorage< Teacher >将无法序列化Student）。

如果我们以逆变的方式使用（假设你建立了一个IStorage的实例，并将其视为IStorage< Teacher >），则我们可能会在调用Deserialize时产生异常，因为Deserialize方法不支持逆变，它只能返回People不能返回Teacher。

22.什么是IEnumerable?

IEnumerable及IEnumerable的泛型版本IEnumerable是一个接口，它只含有一个方法GetEnumerator。Enumerable这个静态类型含有很多扩展方法，其扩展的目标是IEnumerable。

实现了这个接口的类可以使用Foreach关键字进行迭代（迭代的意思是对于一个集合，可以逐一取出元素并遍历之）。实现这个接口必须实现方法GetEnumerator。

23.IEnumerable的缺点有哪些?

IEnumerable功能有限，不能插入和删除。

访问IEnumerable只能通过迭代，不能使用索引器。迭代显然是非线程安全的，每次IEnumerable都会生成新的IEnumerator，从而形成多个互相不影响的迭代过程。

在迭代时，只能前进不能后退。新的迭代不会记得之前迭代后值的任何变化。

24. 泛型的优点有哪些?

代码的可重用性。无需从基类型继承，无需重写成员。

扩展性好。

类型安全性提高。泛型将类型安全的负担从你那里转移到编译器。没有必要编写代码来测试正确的数据类型，因为它会在编译时强制执行。降低了强制类型转换的必要性和运行时错误的可能性。

性能提高。泛型集合类型通常能更好地存储和操作值类型，因为无需对值类型进行装箱。

25.try {}里有一个return语句，那么紧跟在这个try后的finally {}里的code会不会被执行，什么时候被执行，在return前还是后?

会执行，在return前执行。

26.C# 中的异常类有哪些？

C# 异常是使用类来表示的，异常类主要是直接或间接地派生于 System.Exception 类。

System.ApplicationException 和 System.SystemException 类是派生于 System.Exception 类的异常类。

System.ApplicationException 类支持由应用程序生成的异常，所以我们自己定义的异常都应派生自该类。

System.SystemException 类是所有预定义的系统异常的基类。

System.IO.IOException 用于处理 I/O 错误(读写文件)。

System.IndexOutOfRangeException 用于处理当方法指向超出范围的数组索引时生成的错误。

System.ArrayTypeMismatchException 用于处理当数组类型不匹配时生成的错误。

System.NullReferenceException 用于处理当依从一个空对象时生成的错误。

System.DivideByZeroException 用于处理当除以零时生成的错误。例如：100/0就会报这个错误。

System.InvalidCastException 用于处理在类型转换期间生成的错误。

System.OutOfMemoryException 用于处理空闲内存不足生成的错误。

System.StackOverflowException 用于处理栈溢出生成的错误。

27.泛型有哪些常见的约束？

泛型约束 public void GetEntity() where T:class

where T :struct //约束T必须为值类型

where K : class //约束K必须为引用类型

where V : IComparable //约束V必须是实现了IComparable接口

where W : K //要求W必须是K类型，或者K类型的子类

where X :class ,new () // 或者写出 new class() ; X必须是引用类型，并且要有一个无参的构造函数（对于一个类型有多有约束，中间用逗号隔开）

28.Collection和Collections的区别？

Collection 是集合类的上级接口，Collections 是针对集合类的一个帮助类，它提供一系列静态方法来实现对各种集合的搜索，排序，线程安全化操作。

29.能用foreach 遍历访问的对象的要求？

需要实现IEnumerable接口或声明GetEnumerator方法的类型。

30.说出五个集合类？

List：泛型类；

Stack：堆栈，后进先出的访问各个元素

Dictionary<TKey, TValue>：字典类，key是区分大小写；value用于存储对应于key的值

HashSet：此集合类中不能有重复的子元素

SortedList<TKey, TValue>：排序列表，key是排好序的数组。

31.HashMap和Hashtable区别？

Collection是集合类的上级接口，Collections是针对集合类的一个帮助类，它提供一系列静态方法来实现对各种集合的搜索，排序，线程安全化操作。

Unity面试手册：C#多线程

1.根据线程安全的相关知识，分析以下代码，当调用test方法时i>10时是否会引起死锁?并简要说明理由。

```
1  public void test(int i)
2  {
3      lock(this)
4      {
5          if (i>10)
6          {
7              i--;
8              test(i);
9          }
10     }
11 }
```

不会发生死锁，（但有一点int是按值传递的，所以每次改变的都只是一个副本，因此不会出现死锁。但如果把int换做一个object，那么死锁会发生）

2.描述线程与进程的区别？

线程(Thread)与进程（Process）二者都定义了某种边界，不同的是进程定义的是应用程序与应用程序之间的边界，不同的进程之间不能共享代码和数据空间，而线程定义的是代码执行堆栈和执行上下文的边界。一个进程可以包括若干个线程，同时创建多个线程来完成某项任务，便是多线程。

而同一进程中的不同线程共享代码和数据空间。用一个比喻来说，如果一个家庭代表一个进程，在家庭内部，各个成员就是线程，家庭中的每个成员都有义务对家庭的财富进行积累，同时也有权利对家庭财富进行消费，当面对一个任务的时候，家庭也可以派出几个成员来协同完成，而家庭之外的人则没有办法直接消费不属于自己家庭的财产。

3.Windows单个进程所能访问的最大内存量是多少？它与系统的最大虚拟内存一样吗？这对于系统设计有什么影响？

这个需要针对硬件平台，公式为单个进程能访问的最大内存量=2的处理器位数次方/2，比如通常情况下，32位处理器下，单个进程所能访问的最大内存量为： $2^{32} / 2 = 2G$ 。单个进程能访问的最大内存量是最大虚拟内存的1/2，因为要分配给操作系统一半虚拟内存。

4.using() 语法有用吗？什么是IDisposable？

有用，实现了IDisposable的类在using中创建，using结束后会自定调用该对象的Dispose方法，释放资源。

5.前台线程和后台线程有什么区别？

通过将 Thread.IsBackground 属性设置为 true，就可以将线程指定为后台线程

前台线程：应用必须结束掉所有的前台线程才能结束程序，只要有一个前台线程没退出进程就不会自动退出，当然线程是依附在进程上的，所以你直接把进程KO掉了的话自然所有前台线程也会退出。

后台线程：进程可以不考虑后台直接自动退出，进程自动退出后所有的后台线程也会自动销毁。

6.什么是互斥？

当多个线程访问同一个全局变量，或者同一个资源(比如打印机)的时候，需要进行线程间的互斥操作来保证访问的安全性。

7.如何查看和设置线程池的上下限？

线程池的线程数是有限制的，通常情况下，我们无需修改默认的配置。但在一些场合，我们可能需要了解线程池的上下限和剩余的线程数。线程池作为一个缓冲池，有着其上下限。在通常情况下，当线程池中的线程数小于线程池设置的下限时，线程池会设法创建新的线程，而当线程池中的线程数大于线程池设置的上限时，线程池将销毁多余的线程。

PS：在.NET Framework 4.0中，每个CPU默认的工作者线程数量最大值为250个，最小值为2个。而IO线程的默认最大值为1000个，最小值为2个。

在.NET中，通过 ThreadPool 类型提供的5个静态方法可以获取和设置线程池的上限和下限，同时它还额外地提供了一个方法来让程序员获知当前可用的线程数量，下面是这五个方法的签名：

- ① static void GetMaxThreads(out int workerThreads, out int completionPortThreads)
- ② static void GetMinThreads(out int workerThreads, out int completionPortThreads)
- ③ static bool SetMaxThreads(int workerThreads, int completionPortThreads)
- ④ static bool SetMinThreads(int workerThreads, int completionPortThreads)
- ⑤ static void GetAvailableThreads(out int workerThreads, out int completionPortThreads)

8. Task状态机的实现和工作机制是什么？

CPS全称是Continuation Passing Style，在.NET中，它会自动编译为： 1. 将所有引用的局部变量做成闭包，放到一个隐藏的状态机的类中； 2. 将所有的await展开成一个状态号，有几个await就有几个状态号； 3. 每次执行完一个状态，都重复回调状态机的MoveNext方法，同时指定下一个状态号； 4. MoveNext方法还需处理线程和异常等问题。

9.await的作用和原理，并说明和GetResult()有什么区别？

从状态机的角度出发，await的本质是调用Task.GetAwaiter()的UnsafeOnCompleted(Action)回调，并指定下一个状态号。

从多线程的角度出发，如果await的Task需要新的线程上执行，该状态机的MoveNext()方法会立即返回，此时，主线程被释放出来了，然后在UnsafeOnCompleted回调的action指定的线程上下文中继续MoveNext()和下一个状态的代码。

而相比之下，GetResult()就是在当前线程上立即等待Task的完成，在Task完成前，当前线程不会释放。 注意：Task也可能不一定在新的线程上执行，此时用GetResult()或者await就只有会不会创建状态机的区别了。

10.Task和Thread有区别吗？

Task和Thread都能创建用多线程的方式执行代码，但它们有较大的区别。Task较新，发布于.NET 4.5，能结合新的async/await代码模型写代码，它不止能创建新线程，还能使用线程池（默认）、单线程等方式编程，在UI编程领域，Task还能自动返回UI线程上下文，还提供了许多便利API以管理多个Task。

11.多线程有什么用？

(1) 发挥多核CPU的优势

随着工业的进步，现在的笔记本、台式机乃至商用的应用服务器至少也都是双核的，4核、8核甚至16核的也都不少见，如果是单线程的程序，那么在双核CPU上就浪费了50%，在4核CPU上就浪费了75%。单核CPU上所谓的”多线程”那是假的多线程，同一时间处理器只会处理一段逻辑，只不过线程之间切换得比较快，看着像多个线程”同时”运行罢了。多核CPU上的多线程才是真正多线程，它能让你的多段逻辑同时工作，多线程，可以真正发挥出多核CPU的优势来，达到充分利用CPU的目的。

(2) 防止阻塞

从程序运行效率的角度来看，单核CPU不但不会发挥出多线程的优势，反而会因为单核CPU上运行多线程导致线程上下文的切换，而降低程序整体的效率。但是单核CPU我们还是要应用多线程，就是为了防止阻塞。试想，如果单核CPU使用单线程，那么只要这个线程阻塞了，比方说远程读取某个数据吧，对端迟迟未返回又没有设置超时时间，那么你的整个程序在数据返回回来之前就停止运行了。多线程可以防止这个问题，多条线程同时运行，哪怕一条线程的代码执行读取数据阻塞，也不会影响其它任务的执行。

(3) 便于建模

这是另外一个没有这么明显的优点了。假设有一个大的任务A，单线程编程，那么就要考虑很多，建立整个程序模型比较麻烦。但是如果把这个大的任务A分解成几个小任务，任务B、任务C、任务D，分别建立程序模型，并通过多线程分别运行这几个任务，那就简单很多了。

12. 两个线程交替打印0~100的奇偶数

这道题就是说有两个线程，一个名为偶数线程，一个名为奇数线程，偶数线程只打印偶数，奇数线程只打印奇数，两个线程按顺序交替打印。

```
1 public class ThreadExample
2 {
3     ///<summary>
4     ///两个线程交替打印0~100的奇偶数
5     ///</summary>
6     public static void PrintOddEvenNumber
7     {
8         var work = new TheadWorkTest;
9         var thread1 = new Thread(work.PrintOddNumber) { Name = "奇数线程" };
10        var thread2 = new Thread(work.PrintEvenNumber) { Name = "偶数线程" };
11        thread1.Start();
12        thread2.Start();
13    }
14 }
15
16 public class TheadWorkTest
17 {
18
19     private static readonly AutoResetEvent oddAre = new AutoResetEvent( false );
20
21     private static readonly AutoResetEvent evenAre = new AutoResetEvent( false );
22
23     public void PrintOddNumber
24     {
25         oddAre.WaitOne();
26         for( var i = 0; i < 100; i++ )
27         {
28             if( i % 2 != 1 ) continue;
29             Console.WriteLine($" {Thread.CurrentThread.Name}: {i} ");
30             evenAre.Set();
31             oddAre.WaitOne();
32         }
33     }
34
35     public void PrintEvenNumber
36     {
37         for( var i = 0; i < 100; i++ )
38         {
39             if( i % 2 != 0 ) continue;
40             Console.WriteLine($" {Thread.CurrentThread.Name}: {i} ");
41             oddAre.Set();
42             evenAre.WaitOne();
43         }
44     }
45 }
```


13.为什么GUI不支持跨线程调用?有什么解决方法?

因为GUI应用程序引入了一个特殊的线程处理模型，为了保证UI控件的线程安全，这个线程处理模型不允许

其他子线程跨线程访问UI元素。解决方法比较多的：

- 利用UI控件提供的方法，Winform是控件的Invoke方法，WPF中是控件的Dispatcher.Invoke方法；
- 使用BackgroundWorker；
- 使用GUI线程处理模型的同步上下文SynchronizationContext来提交UI更新操作

14.说说常用的锁，lock是一种什么样的锁?

常用的如SemaphoreSlim、ManualResetEventSlim、Monitor、ReadWriteLockSlim，lock是一个混合锁，其实质是Monitor

15.lock为什么要锁定一个参数（可否为值类型？）参数有什么要求？

lock的锁对象要求为一个引用类型。她可以锁定值类型，但值类型会被装箱，每次装箱后的对象都不一样，会导致锁定无效。

对于lock锁，锁定的这个对象参数才是关键，这个参数的同步索引块指针会指向一个真正的锁（同步块），这个锁（同步块）会被复用。

16.多线程和异步的区别和联系？

多线程是实现异步的主要方式之一，异步并不等同于多线程。实现异步的方式还有很多，比如利用硬件的特性、使用进程或纤程等。

在.NET中就有很多的异步编程支持，比如很多地方都有Begin、*End*的方法，就是一种异步编程支持，她内部有些是利用多线程，有些是利用硬件的特性来实现的异步编程。

17.线程池的有点和不足？

优点：减小线程创建和销毁的开销，可以复用线程；也从而减少了线程上下文切换的性能损失；在GC回收时，较少的线程更有利于GC的回收效率。缺点：线程池无法对一个线程有更多的精确的控制，如了解其运行状态等；不能设置线程的优先级；加入到线程池的任务（方法）不能有返回值；对于需要长期运行的任务就不适合线程池。

18.Mutex和lock有什么不同？一般用哪一种比较好？

Mutex是一个基于内核模式的互斥锁，支持锁的递归调用，而Lock是一个混合锁，一般建议使用Lock更好，因为lock的性能更好。

19.用双检锁实现一个单例模式Singleton。

▼ Plain Text 复制代码

```
1  public static class Singleton<T> where T : class,new()
2  {
3      private static T _Instance;
4      private static object _lockObj = new object();
5
6      /// <summary>
7      /// 获取单例对象的实例
8      /// </summary>
9      public static T GetInstance()
10     {
11         if (_Instance != null) return _Instance;
12         lock (_lockObj)
13         {
14             if (_Instance == null)
15             {
16                 var temp = Activator.CreateInstance<T>();
17                 System.Threading.Interlocked.Exchange(ref _Instance, temp);
18             }
19         }
20         return _Instance;
21     }
22 }
```

20.Thread 类有哪些常用的属性和方法？

属性：

CurrentContext：获取线程正在其中执行的当前上下文。

CurrentCulture：获取或设置当前线程的区域性。

CurrentPrincipal：获取或设置线程的当前负责人（对基于角色的安全性而言）。

CurrentThread：获取当前正在运行的线程。

CurrentUICulture：获取或设置资源管理器使用的当前区域性以便在运行时查找区域性特定的资源。

IsBackground：获取或设置一个值，该值指示某个线程是否为后台线程。

Priority：获取或设置一个值，该值指示线程的调度优先级。

ThreadState：获取一个值，该值包含当前线程的状态。

方法：

public void Abort()

在调用此方法的线程上引发 ThreadAbortException，以开始终止此线程的过程。调用此方法通常会终止线程。



Plain Text

复制代码

```
1 public static void ResetAbort()
```


取消为当前线程请求的 Abort。

public void Start()

开始一个线程。



Plain Text


 复制代码

```
1 public static void Sleep( int millisecondsTimeout )
```

让线程暂停一段时间。



Plain Text

 复制代码

```
1 public static bool Yield()
```

导致调用线程执行准备好在当前处理器上运行的另一个线程。由操作系统选择要执行的线程。

Unity面试手册：设计模式

1.说一下设计模式？你都知道哪些？

设计模式总共有 23 种，总体来说可以分为三大类：创建型模式（ Creational Patterns ）、结构型模式（ Structural Patterns ）和行为型模式（ Behavioral Patterns ）。

创建型模式：

工厂模式、抽象工厂模式、单例模式、建造者模式、原型模式 关注于对象的创建，同时隐藏创建逻辑

结构型模式：

适配器模式、过滤器模式、装饰模式、享元模式、代理模式、外观模式、组合模式、桥接模式 关注类和对象之间的组合

行为型模式： 责任链模式、命令模式、中介者模式、观察者模式、状态模式、策略模式、模板模式、空对象模式、备忘录模式、迭代器模式、解释器模式、访问者模式 关注对象之间的通信

2.什么是简单工厂模式？

简单工厂模式又叫静态工厂方法模式，就是建立一个工厂类，对实现了同一接口的一些类进行实例的创建。比如，一台咖啡机就可以理解为一个工厂模式，你只需要按下想喝的咖啡品类的按钮（摩卡或拿铁），它就会给你生产一杯相应的咖啡，你不需要管它内部的具体实现，只要告诉它你的需求即可。

优点：

工厂类含有必要的判断逻辑，可以决定在什么时候创建哪一个产品类的实例，客户端可以免除直接创建产品对象的责任，而仅仅“消费”产品；简单工厂模式通过这种做法实现了对责任的分割，它提供了专门的工厂类用于创建对象；

客户端无须知道所创建的具体产品类的类名，只需要知道具体产品类所对应的参数即可，对于一些复杂的类名，通过简单工厂模式可以减少使用者的记忆量；

通过引入配置文件，可以在不修改任何客户端代码的情况下更换和增加新的具体产品类，在一定程度上提高了系统的灵活性。

缺点：

不易拓展，一旦添加新的产品类型，就不得不修改工厂的创建逻辑；

产品类型较多时，工厂的创建逻辑可能过于复杂，一旦出错可能造成所有产品的创建失败，不利于系统的维护。

代码实现：

```
1  class Factory {
2      public static String createProduct(String product) {
3          String result = null;
4          switch (product) {
5              case "Mocca":
6                  result = "摩卡";
7                  break;
8              case "Latte":
9                  result = "拿铁";
10                 break;
11             default:
12                 result = "其他";
13                 break;
14         }
15         return result;
16     }
17 }
```

3.介绍一下原型模式(Prototype)?

在软件系统中，经常面临着“某些结构复杂的对象”的创建工作;由于需求的变化，这些对象经常面临着剧烈的变化,但是它们却拥有比较稳定一致的接口。

如何应对这种变化？如何向“客户程序（使用这些对象的程序）”隔离出“这些易变对象”，从而使得“依赖这些易变对象的客户程序”不随着需求改变而改变？

用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象。

适用性：

1. 当一个系统应该独立于它的产品创建，构成和表示时；
2. 当要实例化的类是在运行时刻指定时，例如，通过动态装载；
3. 为了避免创建一个与产品类层次平行的工厂类层次时；
4. 当一个类的实例只能有几个不同状态组合中的一种时。建立相应数目的原型并克隆它们可能比每次用合适的状态手工实例化该类更方便一些。

```
1 public abstract class NormalActor
2 {
3     public abstract NormalActor clone();
4 }
5 public class NormalActorA:NormalActor
6 {
7     public override NormalActor clone()
8     {
9         Console.WriteLine("NormalActorA is call");
10        return (NormalActor)this.MemberwiseClone();
11    }
12 }
13 public class NormalActorB :NormalActor
14 {
15     public override NormalActor clone()
16     {
17         Console.WriteLine("NormalActorB was called");
18         return (NormalActor)this.MemberwiseClone();
19     }
20 }
21 }
22
23
24 public class GameSystem
25 {
26     public void Run(NormalActor normalActor)
27     {
28         NormalActor normalActor1 = normalActor.clone();
29         NormalActor normalActor2 = normalActor.clone();
30         NormalActor normalActor3 = normalActor.clone();
31         NormalActor normalActor4 = normalActor.clone();
32         NormalActor normalActor5 = normalActor.clone();
33     }
34 }
35
36
37 class Program
38 {
39     static void Main(string[] args)
40     {
41         GameSystem gameSystem = new GameSystem();
42
43         gameSystem.Run(new NormalActorA());
44     }
45 }
```



```

46
47 如果又需要创建新的对象 (flyActor)，只需创建此抽象类，然后具体类进行克隆。
48 public abstract class FlyActor
49 {
50     public abstract FlyActor clone();
51 }
52 public class FlyActorB:FlyActor
53 {
54     /// <summary>
55     /// 浅拷贝，如果用深拷贝，可使用序列化
56     /// </summary>
57     /// <returns></returns>
58     public override FlyActor clone()
59     {
60         return (FlyActor)this.MemberwiseClone();
61     }
62 }
63 此时，调用的Main()函数只需如下：
64 class Program
65     {
66     static void Main(string[] args)
67     {
68         GameSystem gameSystem = new GameSystem();
69         gameSystem.Run(new NormalActorA(), new FlyActorB());
70     }
71 }

```

4.用双检锁实现一个单例模式Singleton。

```
1  public static class Singleton<T> where T : class,new()  
2  {  
3      private static T _Instance;  
4      private static object _lockObj = new object();  
5  
6      /// <summary>  
7      /// 获取单例对象的实例  
8      /// </summary>  
9      public static T GetInstance()  
10     {  
11         if (_Instance != null) return _Instance;  
12         lock (_lockObj)  
13         {  
14             if (_Instance == null)  
15             {  
16                 var temp = Activator.CreateInstance<T>();  
17                 System.Threading.Interlocked.Exchange(ref _Instance, t  
18     emp);  
19             }  
20             return _Instance;  
21         }  
22     }
```

5..NET框架有哪些适配器模式的应用？

(1)在.Net中复用com对象：

Com 对象不符合.net对象的接口

使用tlbimp.exe来创建一个Runtime Callable Wrapper(RCW)以使其符合.net对象的接口。

(2).NET数据访问类（Adapter变体）：

各种数据库并没有提供DataSet接口

使用DBDataAdapter可以将任何各数据库访问/存取适配到一个DataSet对象上。

(3)集合类中对现有对象的排序（Adapter变体）；

现有对象未实现IComparable接口

实现一个排序适配器（继承IComparer接口），然后在其Compare方法中对两个对象进行比较。

6.原型模式的优缺点？

原型模式的优点有：

原型模式向客户隐藏了创建新实例的复杂性

原型模式允许动态增加或较少产品类。

原型模式简化了实例的创建结构，工厂方法模式需要有一个与产品类等级结构相同的等级结构，而原型模式不需要这样。

产品类不需要事先确定产品的等级结构，因为原型模式适用于任何的等级结构

原型模式的缺点有：

每个类必须配备一个克隆方法

配备克隆方法需要对类的功能进行通盘考虑，这对于全新的类不是很难，但对于已有的类不一定很容易，特别当一个类引用不支持串行化的间接对象，或者引用含有循环结构的时候。

7.工厂模式的优点和缺点还有使用场景？

优点：

工厂方法去除了条件分支（解除了工厂类与分支的耦合），解决了简单工厂对修改开放的问题。

缺点：

工厂方法模式实现时，客户端需要决定实例化哪个工厂来实现对具体产品的构建，选择判断依然存在，也就是说，工厂方法模式将简单工厂的逻辑判断交给客户端去处理。

对简单工厂模式来说，增加功能是要修改工厂类的；但对工厂方法模式，修改的是客户端。

使用场景：

对于某个产品，调用者清楚地知道应该使用哪个具体工厂服务，实例化该具体工厂，生产出具体的产品来。

子类的数量不固定，随时可能有新的功能子类出现

8.简单工厂、工厂方法、抽象工厂的区别？

简单工厂是一个工厂只生产一类的产品,面对的是具体的类；

工厂方法是可以生产不同的产品,把公共的方法抽象出来,然后进行创建各种各样的产品；

抽象工厂把几种产品划出共同的东西,把相互依赖的对象抽象出来,只要实现这些接口就可以得到不同的产品.

9.什么是建造者模式？

建造者模式指的是将一个产品的内部表示与产品的构造过程分割开来，从而可以使一个建造过程生成具体不同的内部表示的产品对象。强调的是产品的构造过程。其实现要点有：

将产品的内部表示与产品的构造过程分割开来。问：如何把它们分割开呢？答：不要把产品的构造过程放在产品类中，而是由建造者类来负责构造过程，产品的内部表示放在产品类中，这样不就分割开了嘛。

10.介绍一下 代理模式？

在系统开发中，有些对象由于网络或其他障碍，以至于不能直接对其访问，此时可以通过一个代理对象来实现对目标对象的访问。如.NET中的调用Web服务等操作。

代理模式指的是给某一个对象提供一个代理，并由代理对象控制对原对象的访问。

11.外观模式、适配器模式和代理模式区别？

这三个模式的相同之处是，它们都是作为客户端与真实被使用的类或系统之间的一个中间层，起到让客户端间接调用真实类的作用，不同之处在于，所应用的场合和意图不同。

代理模式与外观模式主要区别在于，代理对象无法直接访问对象，只能由代理对象提供访问，而外观对象提供对各个子系统简化访问调用接口，而适配器模式则不需要虚构一个代理者，目的是复用原有的接口。外观模式是定义新的接口，而适配器则是复用原有的接口。

另外，它们应用设计的不同阶段，外观模式用于设计的前期，因为系统需要前期就需要依赖于外观，而适配器应用于设计完成之后，当发现设计完成的类无法协同工作时，可以采用适配器模式。然而很多情况下在设计初期就要考虑适配器模式的使用，如涉及到大量第三方应用接口的情况；代理模式是模式完成后，想以服务的方式提供给其他客户端进行调用，此时其他客户端可以使用代理模式来对模块进行访问。

总之，代理模式提供与真实类一致的接口，旨在用来代理类来访问真实的类，外观模式旨在简化接口，适配器模式旨在转换接口。

Unity面试手册：Lua面试题

1.实现替换字符串"abcdefgh"中的"abc"为"ddc"?



Plain Text

复制代码

```
1 string.gsub("abcdefgh","abc","ddc")
```

2.ipairs和pairs的区别?

1.ipairs遇到nil会停止，pairs会输出nil值然后继续下去

2.ipairs并不会输出table中存储的键值对,会跳过键值对，然后顺序输出table中的值。而pairs会输出table的键值对，先顺序输出值，再乱序(键的哈希值)输出键值对。

3.函数冒号与点的区别?

冒号的第一个参数默认为self，指向调用该函数的表。

4.print(string.find(“hello hello”, " hel"))的结果?

6 9

5.普通全局变量和static全局变量的区别?

生存周期相同，作用域不同，普通全局变量可以作用于所有文件，extern声明即可，而static全局变量只能作用于当前文件。

6.请写一个带有不定参数的lua函数,并输出所有的参数?

▼ Plain Text 复制代码

```
1 function test( ... )
2   local args = { ... }
3   for k,v in pairs(args) do
4     print(k,v)
5   end
6 end
```

7.如下一段程序，请在TODO处插入代码，使后面对table新建值时提示错误，并使其无效？

▼ Plain Text 复制代码

```
1 local table = setmetatable({}, {})
2 table.key = "iam key"
3 table.value = 123
4 print(table.key)
5 ---- TODO:在这里插入你的代码
```

答案：

▼ Plain Text 复制代码

```
1 local mt = getmetatable(table) -- 获得table的元表
2 function mt:__newindex(key,value) -- 添加__newindex元方法
3   table[key] = nil
4   print("cannot create new property" .. key)
5 end
```

8. 什么是热更新？

热更新也叫不停机更新，是在游戏服务器运行期间对游戏进行更新。实现不停机修正bug、修改游戏数据等操作。也可以这样讲：一辆车以时速150km跑着，突然爆胎了，然后司机告诉你，我不停车，你去把轮胎换了，小心点。

9.热更新的原理是什么？

第一种:

lua中的require会阻止多次加载相同的模块。所以当需要更新系统的时候，要卸载掉响应的模块。（把package.loaded里对应模块名下设置为nil，以保证下次require重新加载）并把全局表中的对应的模块表置 nil 。同时把数据记录在专用的全局表下，并用 local 去引用它。

初始化这些数据的时候，首先应该检查他们是否被初始化过了。这样来保证数据不被更新过程重置。

代码示例：

▼ Plain Text 复制代码

```
1 function reloadUp(module_name)
2     package.loaded[modulename] = nil
3     require(modulename)
4 end
```

这种做法简单粗暴，虽然能完成热更新，但是 问题很多，旧的引用的模块无法得到更新，这种程度的热更新显然不能满足现在的游戏开发需求。

第二种：

▼ Plain Text 复制代码

```
1 function reloadUp(module_name)
2     local old_module = _G[module_name]
3
4     package.loaded[module_name] = nil
5     require (module_name)
6
7     local new_module = _G[module_name]
8     for k, v in pairs(new_module) do
9         old_module[k] = v
10    end
11
12    package.loaded[module_name] = old_module
13 end
```

10. lua深拷贝和浅拷贝的区别？

使用 = 运算符进行浅拷贝

分2种情况

1.拷贝对象是string、number、bool基本类型。拷贝的过程就是复制黏贴！修改新拷贝出来的对象，不会影响原先对象的值，两者互不干涉。

2.拷贝对象是table表，拷贝出来的对象和原先对象时同一个对象，占用同一个对象，只是一个人两个名字，类似C#引用地址，指向同一个堆里的数据~，两者任意改变都会影响对方。

11.Lua如何实现深拷贝？

复制对象的基本类型，也复制源对象中的对象

常常需用对Table表进行深拷贝，赋值一个全新的一模一样的对象，但不是同一个表

Lua没有实现，封装一个函数，递归拷贝table中所有元素，以及设置metetable元表

如果key和value都不包含table属性，那么每次在泛型for内调用的Func就直接由if判断返回具体的key和value

如果有包含多重table属性，那么这段if判断就是用来解开下一层table的，最后层层递归返回。

核心逻辑：使用递归遍历表中的所有元素。

12.解释下lua中的元表和元方法？

元表metatable：允许该表table行为，行为关联元方法，类似一种“操作指南”，包含各种操作行为的解决方案！

元方法：当表执行某些操作失败的时候，操作指南里的元方法指导你的行为~~

13.Lua如何实现面向对象？

面向对象特性：封装、继承、多态

Lua中table是非常强大的数据结构，利用table再结合元表metatable，可以方便的在Lua中模拟类，并

实现面向对象编程中具有的特性：封装、继承、多态

14.说说lua中的闭包？

闭包=函数+引用环境

子函数可以使用父函数中的局部变量，这种行为可以理解为闭包！

1、闭包的数据隔离

不同实例上的两个不同闭包，闭包中的upvalue变量各自独立，从而实现数据隔离

2、闭包的数据共享

两个闭包共享一份变量upvalue，引用的是更外部函数的局部变量（即Upvalue），变量是同一个，引用也指向同一个地方，从而实现对共享数据进行访问和修改。

3、利用闭包实现简单的迭代器

迭代器只是一个生成器，他自己本身不带循环。我们还需要在循环里面去调用它才行。

1) while...do循环，每次调用迭代器都会产生一个新的闭包，闭包内部包括了upvalue(t,i,n)，闭包根据上一次的记录，返回下一个元素，实现迭代

2) for...in循环，只会产生一个闭包函数，后面每一次迭代都是使用该闭包函数。内部保存迭代函数、状态常量、控制变量。

15. 请深入说明Lua的数据结构和内存占用？

lua的数据结构的内存分配是动态的,常见的数据结构是string和table。

16.热更新方案有哪些？以及具体热更流程？

1、整包：存放在上SteamingAssets里

——策略：完整更新资源放在包里

——优点：首次更新少

——缺点：安装包下载时间长，首次安装久

2、分包

——策略：少部分资源放在包里，大部分更新资源存放在更新资源器中

——优点：安装包小，安装时间短，下载快

——缺点：首次更新下载解压缩包时间旧

3、适用性

——海外游戏大部分是使用分包策略，平台规定

——国内游戏大部分是使用整包策略

4、文件可读写路径

——Application.streamingAssetsPath 只读目录

——Application.persistentDataPath 可读写目录

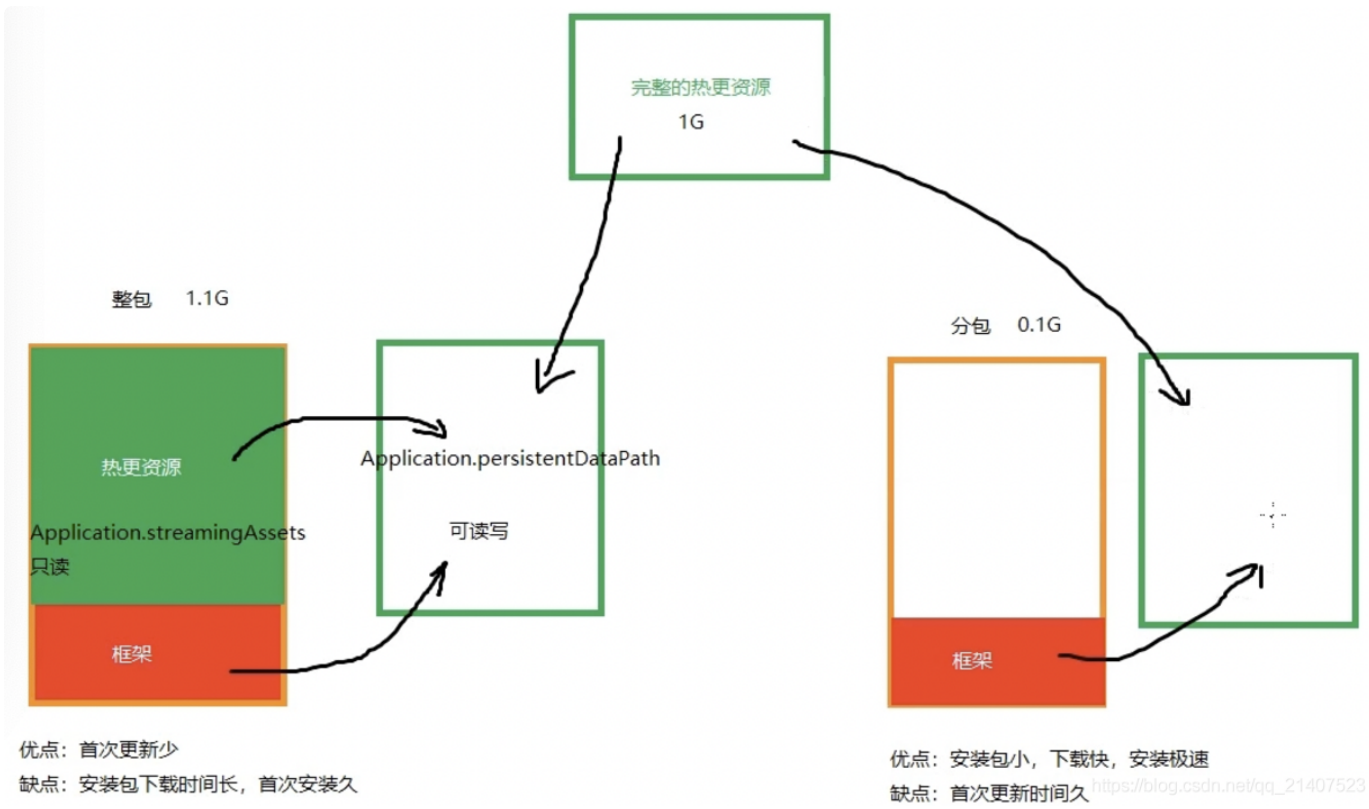
——资源服务器地址URL

5、【从资源服务器】下载单个文件或多个文件

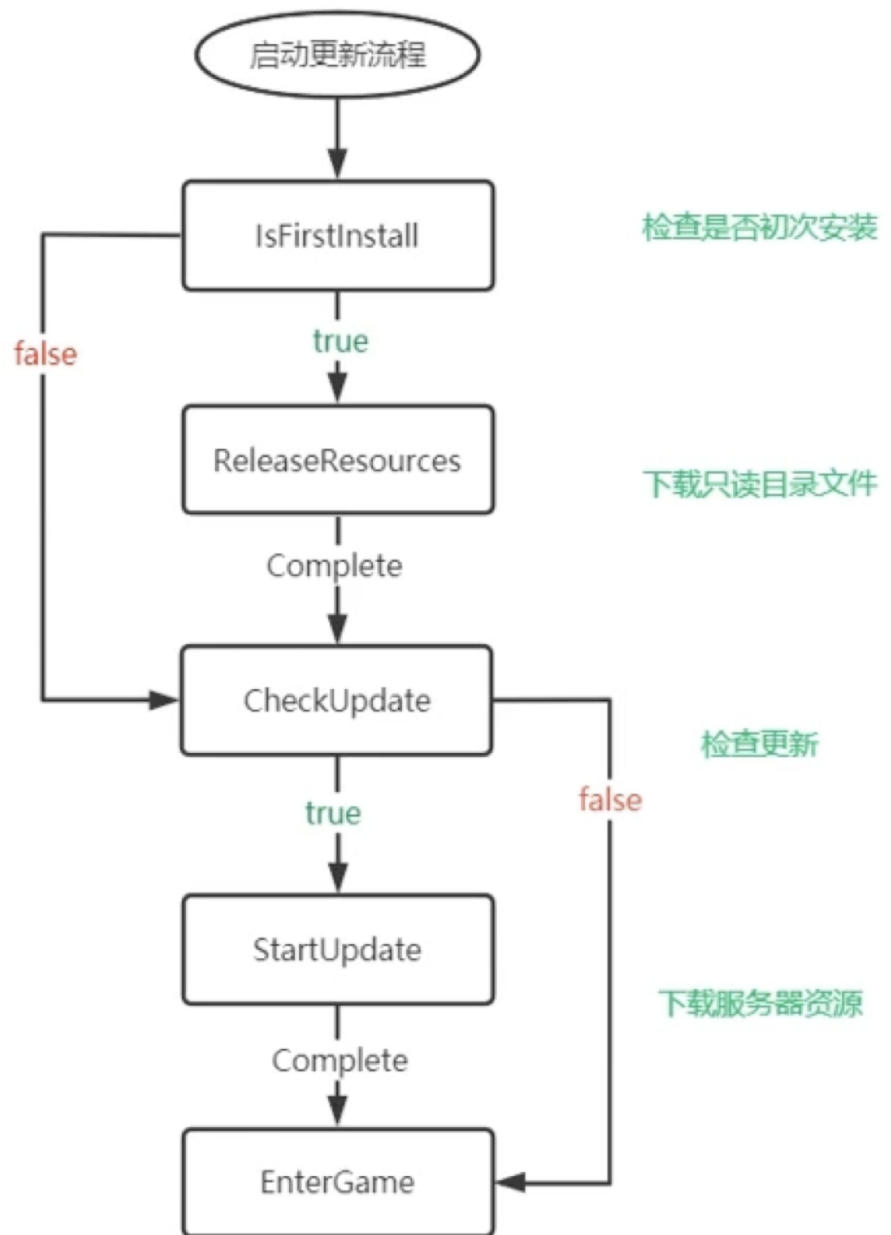
——NetWorking.UnityWebRequest获取URL，HTTP GET，连接资源服务器

——获取到downloadHandler的文件数据Data，完成后会回调方法，将文件Data作为参数传出

6、检查是否初次安装



1. 下载文件
2. 写入文件
3. 解析filelist



17. index和newindex元方法的区别？

访问不存在的数据，由__index提供最终结果

__index元方法可以是一个函数，Lua语言就会以【表】和【不存在键】为参数调用该函数

__index元方法也可以是一个表，Lua语言就访问这个元表

18. 请写一个多值返回的函数 ？



Plain Text

复制代码

```
1 function foo2 () return 'a','b' end
```

19.请写一个可变参数的函数？



Plain Text

复制代码

```
1 function g (a, b, ...) end
2
3 g(3, 4, 5, 8) a=3, b=4, arg={5, 8; n=2}
```

20. Lua如何实现C#构造函数new方法？

new传进的参数当做一个本地表，

元表本身Self作为元表，设置元方法，将其作为参数表o的元表，并返回出去

从而实现C# new构造函数。

调用new的时候，其实访问的是元表内容。



Plain Text

复制代码

```
1 function Class:(o) -- 传入self为Account
2     o = o or {}
3     self.__index = self --直接把表Class当做元表
4     setmetatable(o, self)
5     return o
6 end
7
8 local object = Class:new(o) --object变量可以访问元表
```

21.require, loadfile和dofile的区别？

在lua中我们引用其他模块的时候，可以使用三种方法：require, loadfile和dofile。那么它们有什么区别呢？

首先区分loadfile, loadfile只负责编译并不会执行模块代码，而require和dofile都会编译且执行

