**晚自习面试题**

**第一天**

1问：C#常用的访问修饰符是哪几个？

答：public，private，protect，internal

2问：C#程序流程控制的3种基本结构？

答： 顺序，选择（分支），循环

3问：C# 数据类型有那些？

答：值类型和引用类型

4问：如何将double类型变量x转化为int类型的值并赋值给变量y？

答：y=（int）x float z = y

5问：C# 类型转换有几种？

答：隐式类型转换，显式类型转换（强制类型转换）。

6问：变量如何声明（定义）？

答：访问修饰符+变量类型+变量名

7问：C#中值类型和引用类型

答：值类型：byte,short,int,long,float double,bool,char,enum,struct

引用类型：string，class，object，delegate，interface 数组

8问：break和continue的区别？

答：break跳出整个循环，continue:跳出本次循环，继续执行下一次循环

9问：C#的循环语句有哪三种

答：while(循环条件){ } do{ }while(循环条件) for

10问：简述 private, protected、 public、 internal修饰符的访问权限

答：private：只能在本类中访问，protected：父类到子类，internal：在一个程序集中,public:全局访问

**第二天**

1，问：class和struct的区别

答：class和struct最本质的区别是class是引用类型，而struct是值类型

class（类）是面向对象编程的基本概念，所有的类都最终继承自System.object类，因此是一种引用类型。

struct（结构）是一种值类型，所有的结构都继承自System.ValueType类，struct实例在创建时分配在线程的堆栈（stack）上

1. 问：Sealed的作用是？

答：sealed的中文意思是密封，故名思义，就是由它修饰的类或方法将不能被继承或是重写。sealed关键字的作用： 在类声明中使用sealed可防止其它类继承此类；在方法声明中使用sealed修饰符可防止扩充类重写此方法。

3，问：public static const int A = 1;有什么错误

答：有错，static const只能使用一个因为const表示常数不能被static修饰

4，问：const（常量）和readonly（只读）的区别

答：const与readonly相同点：运行期间都只能读取不能更改

1. const定义时即初始化，运行期间无法再初始化；readonly除了在定义时可以初始化外，还能在运行期间的构造函数中初始化，实例只读变量只能在实例构造函数中初始化，静态只读变量只能在静态构造函数中初始化
2. const定义的变量只能通过类名称访问，而readonly会根据其是否定义为静态类型而分别通过对象与类名称访问
3. const只能定义值类型与字符串，若定义引用类型则初始化值必须为null，而readonly无此限制，可以定义引用类型时初始化为null，在对应的构造函数中再重新初始化

5，问：ref参数和out参数是什么？有什么区别？针对函数（方法）的参数的

答：

1. 使用 ref 参数时，传入的参数必须先被初始化。对 out 参数而言，必须在方法中对其完成初始化。
2. 使用 ref 和 out 时，在方法的参数和执行方法时，都要加 ref 或 out 关键字，以满足匹配。
3. out 适合用在需要 return 多个返回值的地方，而 ref 则用在需要被调用的方法，修改调用者的引用的时候。

6，什么时候选择用 ref ? 什么时候选择用 out ？

1. ref 的使用：使用 ref 进行参数的传递时，该参数在创建时，必须设置其初始值，所以ref 侧重于修改；
2. out 的使用：采用 out 参数传递时，该参数在创建时，可以不设置初始值，但是在方法中必须初始化，所以 out 侧重于输出；

7，问：全局和局部变量在内存中的区别

答：有区别。

全局变量保存在内存的全局存储区中，占用静态的存储单元；

局部变量保存在栈中，只有在所在函数被调用时才动态地为变量分配存储单元。

8，问：简述 private, protected、 public、 internal修饰符的访问权限

答：private：只能在本类中，protected：父类到子类，internal：在一个程序集中,public:全局访问

9，问：C#中值类型和引用类型

答：值类型：byte,short,int,long,float double,bool,char,enum,struct

引用类型：string，class，object，delegate，interface 数组

10，问：break和continue的区别？

答：break跳出整个循环，continue:跳出本次循环，继续执行下一次循环

**第三天**

1.问：面向对象七大设计原则？

答：1开闭原则；2单一职责；3迪米特法则原则；4合成/聚合原则；5依赖倒置；6里式替换；7接口隔离

2.问：面向对象三大特性？

答：封装、继承、多态。

3.重写Override与重载Overload的区别

答：重载（Overload）是重新加载的意思，指的是同一个类中方法名相同，但是参数列表不同的方法（返回值不能区别函数是否重载），重载没有关键字。通俗点说重载是对内存中同一个类中的函数或方法进行重新加载的意思。

重写（Override）指子类修改父类中某个函数的函数体，需要重写的父类中的函数必须是由virtual、abstract 或override标注的。

4.简述C#中的虚方法和抽象方法

答： 使用virtual关键字修饰的方法就是虚方法，虚方法是有方法体的，虚方法可以在派生类（子类）中被重写，也可以不被派生类（子类）中被重写使用abstract关键字修饰的方法就是抽象方法，抽象方法没有方法体，抽象方法必须在非抽象子类中被重写。

5.简述接口和接口继承

答：1.接口只定义不包含方法体的方法

　 2.接口可以包含方法,属性，事件和索引器

　　3.接口里的所有成员必须是公共的

　　4.接口不能直接实例化

　　5,接口不能包含任何字段

　　6.接口自身均可以从多个接口继承

　　7.类和结构可以从多个接口继承

8.接口类似于抽象类,但继承接口的类必须实现接口中所有定义成员对象

6.C#静态类构造方法特点

答：静态构造方法,是在构造方法的名字前使用static关键字修饰符的构造方法

特点如下: 只允许有一个无参的静态构造方法在一个类中存在.

　　　　　 静态的构造方法不会被继承

　　　　　　 在所有的静态成员被引用之前执行静态构造方法

　　　　　　 在所有的构造方法中最先被执行的是静态的构造方法.

7.什么是委托

答： C#中的委托都继承自Delegate类型,委托类型的声明与方法签名相似,

有一个返回值和任意数目类型的参数.

**第四天**

1，什么是协同程序？

答：在主线程运行的同时开启另一段逻辑处理，来协助当前程序的执行，协程很像多线程，但是不是多线程，Unity的协程是在每帧结束之后去检测yield的条件是否满足。

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，碰撞器根据物理引擎引发碰撞，产生碰撞的效果

触发器是碰撞器被物理引擎所忽略，没有碰撞效果，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，而且必须是运动的物体带有Rigidbody脚本才能检测到碰撞。

4，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目录和Library目录一起迁移

2.导出包，export Package

3.用unity自带的assets Server功能

5，Unity提供了几种光源，分别是什么

答：平行光：Directional Light

点光源：Point Light

聚光灯：Spot Light

区域光源：Area Light

6，CharacterController（角色控制器）和Rigidbody（刚体）的区别

答：Rigidbody具有完全真实物理的特性，包含了常用的物理特性,

而CharacterController可以说是受限的的Rigidbody，具有一定的物理效果但不是完全真实的

7，简述prefab的用处

答：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，便于团队的交流

**第五天**

1，问：物体自身旋转使用的函数？

答：Transform.Rotate(new Vector3(0,0,2))

2，问：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。

**Awake->OnEnable->Start->FixedUpdate->Update->LateUpdate->OnGUI->OnDisable->OnDestroy**

3，问：如何销毁一个UnityEngine.Object及其子类？

答：使用Destroy()方法;(含港澳台

4，问：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。缺点是增加了内存。

5，问：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。缺点是增加了内存。

6，问：U3D中用于记录节点空间几何（变化）信息的组件名称，及其父类名称

答：Transform 父类是 Component

7，问：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上，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。

8，问：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（一定要有碰撞器）发生碰撞时，它将停止发射。

9，问：如何让已经存在的GameObject在LoadLevel（切换场景）后不被卸载掉？

答：void Awake(){

DontDestroyOnLoad(transform.gameObject);

}

10，问：简述四元数的作用，欧拉角的优缺点，四元数优缺点？

答：**四元数**内部由四个数字（在Unity中称为x，y，z和w）组成，然而这些数字不表示角度或轴，并且通常不需要直接访问它们，四元数表示三维空间中的旋转。

**万向锁**：unity 3D欧拉角的旋转顺序（父子关系）是y-x-z。即旋转y轴x和z轴都变，旋转x轴只有z轴变化，旋转z轴其它轴不变。

**[欧拉角](https://so.csdn.net/so/search?q=%E6%AC%A7%E6%8B%89%E8%A7%92&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_45223588/article/details/_blank)**

优点：三个角度组成，直观，容易理解。

优点：可以进行从一个方向到另一个方向旋转大于180度的角度。

优点：由三个数字实现旋转效率相比要高。

弱点：万向锁问题。

**四元数**

优点：四元旋转不存在万向节锁问题。

优点：存储空间小，计算效率高。

弱点：单个四元数不能表示在任何方向上超过180度的旋转。

弱点：四元数的数字表示不直观。

11，问：什么是渲染管道？渲染原理是什么？

答： 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

12，问：移动相机动作在哪个函数里，为什么在这个函数里？

答：LateUpdate，是在所有的update结束后才调用，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。

**第六天**

1，Unity如何优化？内存（内存），渲染（GPU），性能（CPU）

1. 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(不是内存优化，是性能优化)；
2. 释放AssetBundle占用的资源；切换场景使用Resource.unload();（内存优化）
3. 降低模型的片面数，降低模型的骨骼数量，降低贴图的大小；（内存优化）；
4. 使用光照贴图（优化渲染），使用多层次细节(LOD)（性能优化），使用着色器(Shader)，使用预设(Prefab)。
5. 代码中少产生临时变量，少new,string拼接，使用Stringbluider（内存优化）
6. 静态批处理，动态批处理（性能优化）

2，动态加载资源的方式？他们之间的区别

答：1.Resources.Load(); 必须是Resources文件下的资源，

2.AssetBundle 可以任何文件下的资源，可以将资源外置Unity安装包之外

3，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)。

1. 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；
2. 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；
3. 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。
4. 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：支持多线程程序

1. 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（其他线程）thread,否则使用coroutine。
2. 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（其他线程）子线程：处理一些业务逻辑，处理网络数据，处理文件IO读写。
3. 协程（coroutine）：倒计时，延迟调用，辅助主线程处理组件
4. Unity3D的协程和C#线程之间的区别是什么？

答 多线程可以并发（一起运行），

协程：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。

6，当一个细小的高速的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），使用射线检测

7，什么叫动态合批（动态批处理）？跟静态合批（静态批处理）有什么区别？

答：**动态批处理：**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。

**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可以再移动了。

**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。

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

答：如果把摄像机的ClearFlags勾选为Depth Only,那么摄像机就会只渲染看得见的对象，把背景会完全透明，这种情况一般用在两个摄像机以上的场景中

9，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。Sprite需要2的整次幂，打包图片省资源

10，什么是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：

1. Dynamic Batching（动态批处理）

2. Static Batching（静态批处理）

3. 高级特性Shader降级为统一的低级特性的Shader。

11.Unity中用过哪些插件？具体功能

答 FXMaker，制作粒子特效；NGUI，制作2D界面；EasyTouch，摇杆；shaderForge，制作shader；Itween，制作动画；

**第七天**

特殊文件夹

1，Standard Assets

在这个文件夹中的脚本最先被编译。将脚本放在这个文件夹里，就可以用C#脚本来访问js脚本或其他语言的脚本。这个文件夹中的脚本会被导出到Assembly-CSharp-firstpass, Assembly-UnityScript-firstpass 或 Assembly-Boo-firstpass项目中，依语言而定。

2，Editor

以Editor命名的文件夹允许其中的脚本访问Unity Editor的API。如果脚本中使用了在UnityEditor命名空间中的类或方法，它必须被放在名为Editor的文件夹中。Editor文件夹中的脚本不会在build时被包含。在项目中可以有多个Editor文件夹。

注意：如果在普通的文件夹下，Editor文件夹可以处于目录的任何层级。如果在特殊文件夹下，那Editor文件夹必须是特殊文件夹的直接子目录。

3. Plugins

Plugins文件夹用来放native插件。它们会被自动包含进build中去。注意这个文件夹只能是Assets文件夹的直接子目录。

1）在Windows平台下，native 插件是.dll文件；

2）Mac OS X下，是.bundle文件；

3）Linux下，是.so文件。

3.1 Plugins/x86 放window32位的文件

Plugins/x86\_64 放window64位的文件，能兼容window32位的文件

3.2 Plugins/Android

在这个文件夹里放入Java.jar文件。用于java语言的plugins。.so文件也会被包含进来。

3.3 Plugins/IOS

在这个文件夹里放入.bundle文件。用于ObjectC语言的Plugins。

4. Resources

Resources文件夹允许你在脚本中通过文件路径和名称来访问资源。但还是推荐使用直接引用来访问资源。

一旦build(打包)游戏，Resources文件夹中的所有资源被打包进游戏存放资源的archive(包体中)中。当不再使用这些资源了，可以通过Destroy物体，再调用Resources.UnloadUnusedAssets来释放内存。

5. StreamingAssets

这里的文件会被拷贝到build文件夹中，不会修改（移动和网页版不同，他们会被嵌入到最终build文件中）。它们的路径会因平台而有差异，但都可以通过Application.streamingAssetsPath来访问。

6Third

这个文件夹放入第三方插件，例如DoTween、TimeLine

**第八天**

1. 对于UGUI中的Image和RawImage的区别

Image只能使用Sprite（2D and UI）类型的图片，Image有局限性，效率高，对于RawImage来说 他需要的类型是Texture 类型，没有局限性，RawImage的功能更强大 但相对的也更加消耗性能

2，对于UGUI的一些优化的方法

1. 动静划分(动静分离)

经常变化的东西，例如分数、操作按钮或者是对话框的文字等这些我们把它划分为动的UI。很不常用的按钮，任务的头像这些我们把它划分为静的UI，把动的和静的划分开，这样可以减少UIMesh动态更新，在某些比较复杂，常驻的界面可以这样优化。小的界面就没必要了，因为不必要的节点有可能会造成drawcall的增加的。

1. 合理增加UI层级的深度

在不必要的情况下，我们尽量减少UI层级的深度，在UGUI中Hierarchy面板，节点的的深度，表现的就是UI层级的深度，UI中有N层，N越大越靠前，会遮住后面的组件。当深度越深，不处在同一层级的UI就越多，drawcall就会越大。

1. mask使用的次数要少

每存在一个mask，把mask以内和以外的UI分割成两个“部分”，依次计算两个“部分”的drawcall，然后再相加。原因是mask以内和以外的UI不能通过unity3d一次渲染(batch)。所以在使用mask的时候要仔细思考，能不用就不用，实在要用可以考虑用带透明通道的图片代替mask的遮罩功能。

1. 注意不要图文交叉（图文混排）

所说的图文交叉就是text和Image的交叉,这里也不是单单的这两种，还有按钮那些都是,因为当2张图片和1个文字进行重叠时，处理不好会发生一些drawcall的多余，会有多余的内存的消耗。

1. image和text组件的Raycast Target属性

在组件不需要射线检测的时候，我们可以尽可能的把射线检测去掉，在运行的时候，就可以减少不必要的性能开销。

1. 打包图集，图片九宫格
2. 图片压缩

3，Unity Canvas UGUI 屏幕适配问题

1. 锚点适配，通过RectTransform中的Anchors和Pivot来进行控件和窗体的布局适配。
2. Anchors控制当前Panel（面板）相对于父窗体的布局位置，可以设置为居中或者左上角，当父窗体拉伸的时候当前Panel可以自动布局到正确位置。
3. Pivot是当前控件坐标系的锚点（支点）位置
4. 多分辨率适配（标准分辨率适配）

1），UI Scale Mode 设置为 Scale With Screen Size 时，可在下方设置一个默认尺寸,在设计 UI 时就是根据这个默认尺寸进行布局。针对不同屏幕的分辨率，会根据 Screen Match Mode 中的适配模式，对默认尺寸进行缩放来实现适配。

Screen Match Mode 提供三种适配方案。

可通过设置下方的 Match 值来选择是 1,适配宽度.2,适配高度，3,选择二者中间的某个值。

当 Match == 0 时，适配宽度。将宽度设置为屏幕宽度，并保持默认尺寸比例不变。如果此时高度超过屏幕高度，超出部分将会被裁切掉。

当 Match == 1 时，适配高度。将高度设置为屏幕高度，并保持默认尺寸比例不变。如果此时宽度超过屏幕宽度，超出部分将会被裁切掉。

Expand 保持缩放比例，不裁切,对默认尺寸进行缩放，并保证缩放后的宽高均小于或等于实际屏幕宽高的最大尺寸.

Shrink 保持缩放比例，裁切,对默认尺寸进行缩放，并保证缩放后的宽高均大于或等于实际屏幕宽高的最小尺寸

4, Canvas 的 Render Mode (渲染模式) 有三种

1. Screen Space - Overlay（屏幕/覆盖）

这种渲染模式表示 Canvas 下的所有的 UI 控件永远位于屏幕的前面 , 不管有没有相机 , UI元素永远在屏幕最前面

1. Screen Space - Camera（相机）

这种渲染模式 Canvas 和 摄像机之间有一定的距离 , 可以在摄像机和 Canvas 之间播放一些粒子特效

1. World Space（世界空间）

这种模式下 Canvas 就和普通的 3D 物体一样了 , 可以控制它的大小,旋转,缩放等 , 一般用来做血条

**第九天**

1.重载和重写的区别

1)所处位置不同，重载在同类中，重写在父子类中

2)定义方式不同，重载方法名相同参数列表不同，重写方法名和参数列表都相同

3)调用方式不同，重载使用相同对象以不同参数调用，重写用不同对象以相同参数调用

4)多态时机不同 重载时编译时多态 重写是运行时多态

2.面向对象的三大特点

封装、继承、多态

1、封装： 封装是将数据（属性）和行为（方法）相结合，通过行为约束（访问修饰符）代码修改数据的程度，增强数据的安全性，属性是C#封装实现的最好体现。

2、继承： 子类继承父类，继承父类的方法和属性、字段等，提高代码重用性，符合开闭原则。

3、多态性： 重载、重写、多个子类

3.请简述private，public，protected，internal的区别

public：对任何类和成员都公开，无限制访问

private：仅对该类公开

protected：对该类和其派生类公开

internal：只能在包含该类的程序集中访问该类

protected internal：protected + internal （可以在两个程序集中通过继承可以访问）

4.请简述 ArrayList 和 List<Int>的主要区别

ArrayList 不带泛型，数据类型是object类型， 需要装箱拆箱

List<T> 带泛型，泛型约束的类型， 不需要装箱拆箱

5.请描述 Interface（接口） 与抽象类之间的不同

1.接口可以多继承， 抽象类是单继承

2.接口的方法是没有方法体，抽象类只有抽象方法没有方法体，抽象类可以有普通方法

3.接口不能有字段，抽象类可以有字段

4.接口的所有成员都是public（公共的），抽象类可以是有任何访问修饰符

6.C# 函数 Func(string a, string b)用 Lambda 表达式怎么写?

(a,b) => {};（匿名函数或兰博大表达式）

7.数列 1,1,2,3,5,8,13...第 n 位数是多少?用 C#递归算法实现（斐波那契数列）

public int CountNumber(int num) {

if (num == 1 || num == 2) {

return 1;

} else {

return CountNumber(num -1) + CountNumber(num-2);

}

}

8.冒泡排序（手写代码）

public static void BubblingSort(int[] array) {

for (int i = 0; i < array.Length; i++){

for (int j = array.Length - 1; j > 0; j--){

if (array[j] < array[j]) {

int temp = array[j];

array[j] = array[j-1];

array[ j - 1] = temp;

}

}

}

}

9.C# string 类型比 stringBuilder 类型的优势是什么?

1、在处理字符串时：String是创建一个string对象，在创建对象时需要分配内存空间。StringBuilder是在原来的内存中修改，不需要分配内存空间。

2、从内存优化方面来说，频繁对字符串的操作使用StringBuilder更好。

3、从功能上来说String仍然比StringBuilder更强。

4、String主要用于公共API,通用性好，读取性能高，占用内存小。

5、StringBuilder主要用于拼接String,修改性能好。

6、String是不可变的

7、StringBuilder可变

不过现在的编译器已经把 String 的 + 操作优化成 StringBuilder 了， 所以一般用String 就可以了。

**第十天**

1，C#中有哪些常用的容器类（数据结构），各有什么特点。

答：List（列表）,HashTable（哈希表）,Dictionary（字典）,Stack（栈）,Queue（队列）

List：索引泛型容器，访问速度快，修改速度慢

HashTable/Dictionary：键值对，散列表格式，查询效率高，空间占用较大

Stack：后进先出

Queue: 先进先出

ArrayList：所有的元素类型都是object类型

HashTable：所有的键值对元素类型都是object类型

2，C#中委托和接口有什么区别？各用在什么场合？

答：接口（interface）是约束类应该具备的功能集合，约束了类应该具备的功能，接口约束类。

C#中的委托是约束方法集合的一个类型，委托约束方法

在以下情况中使用接口：

1.无法使用单继承的场合

2.完全抽象的场合

3.多人协作的场合

在以下情况中使用委托：多用于事件处理中

3，For，foreach，Enumerator.MoveNext的使用，与内存消耗情况

答：for 循环可以通过索引依次进行遍历，foreach和Enumerator.MoveNext通过迭代的方式进行遍历。

内存消耗上本质上并没有太大的区别。但是在Unity中的Update中，一般不推荐使用foreach 因为会遗留内存垃圾。

4，函数中多次使用string的+=处理，会产生大量内存垃圾（垃圾碎片），有什么好的方法可以解决？

答：通过StringBuilder那进行append，这样可以减少内存垃圾

5，Foreach循环迭代时，若把其中的某个元素删除，程序报错，怎么找到那个元素？以及具体怎么处理这种情况？(注：Try.....Catch捕捉异常，发送信息不可行)

答：foreach 不能进行元素的删除，因为迭代器会锁定迭代的集合，

解决方法：记录找到索引或者key值，迭代结束后再进行删除。

6，GameObject a=new GameObject() GameObject b=a 实例化出来了a，将a赋给b，现在将b=null删除，问a还存在吗？

答：存在，b删除只是将它在栈中的内存删除，而a对象本身是在堆中，所以a还存在

7，当需要频繁创建使用某个对象时，有什么好的程序设计方案来节省内存？

答：1，设计单例模式进行创建对象

2，使用对象池

**第十一天**

1.物体发生碰撞的必要条件

答：物体 A 必须带有（collider（碰撞器）加rigidbody（刚体））或者 CharacterController（角色控制器），另一个物体也必须至少带有 collider（碰撞器）

2.使用过哪些第三方插件？

答：一、界面制作 推荐：NGUI UGUI

二、2D 游戏制作 推荐：2D Toolkit //[ˈtu:lkɪt] 工具包，工具箱

三、可视化编程 推荐：PlayMaker

四、插值插件 推荐：iTween,DOTween

五、路径搜寻 推荐：Simple Path

六、美术及动画制作 推荐：RageSpline,Smooth Moves

七、画面增强 推荐：Bitmap2Material,Strumpy Shader Editor

八、摄像机管理 推荐：Security Camera

九、资源包 推荐：Nature Pack

十、造路插件 EasyRoads3D

3.Render（渲染） 的作用？描述 MeshRender（网格渲染） 和 SkinnedMeshRender（皮肤网格渲染） 的关系与不同？

答： Mesh 就是指模型的网格（同名组件是用于调整网格属性的），MeshFilter 一般是用于获得模型网格的组件，而 MeshRender 是用于把网格渲染出来的组件。

4.简述 SkinnedMesh 的实现原理

答： Skinned Mesh 中文一般称作骨骼蒙皮网格.这种网格中包含骨骼（ Bone ）和蒙皮 (Skinned Mesh) 两个部分，Skinned Mesh 中 Mesh 是一个整体网格，Skinned Mesh 技术的精华在于蒙皮，所谓的皮并不是模型的贴图。而是 Mesh 本身，蒙皮是指将 Mesh 中的顶点附着（绑定）在骨骼之上，而且每个顶点可以被多个骨骼所控制，这样在关节处的顶点由于同时受到父子骨骼的拉扯而改变位置就消除了裂缝。可以理解为具有蒙皮信息的 Mesh 或可当做皮肤用的 Mesh ，这个皮肤就是 Mesh 。而为了有皮肤功能， Mesh 还需要蒙皮信息，即 Skin 数据，没有 Skin 数据就是一个普通的静态 Mesh 了。 Skin 数据决定顶点如何绑定到骨骼上。顶点的 Skin 数据包括顶点受哪些骨骼影响以及这些骨骼影响该顶点时的权重 (weight) ，另外对于每块骨骼还需要骨骼偏移矩阵 (BoneOffsetMatrix) 用来将顶点从 Mesh 空间变换到骨骼空间。

5.MeshCollider 和其他 Collider 的一个主要不同点？

答： MeshCollider 是基于顶点的。建议还是用 boxcollider，boxcollider 本身是基于算法，没有面的概念。

6.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）

避免：(碰撞体变大，FixedUpdate, 代码限制) 射线

7.CharacterController 和 Rigidbody 的区别

答：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，CharacterController没有重力效果。

8.u3d 下如何安全的在不同工程迁移 asset 数据

答：方法 1，可以把 assets 目录和 Library 目录一起迁移，

方法 2，导出包

方法 3，用 unity 带的 assets server 功能

9.Unity3d 中的碰撞器和触发器的区别？

答：触发器只是碰撞器身上的一个属性，碰撞器是触发器的载体。

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。

a.如果不想让碰撞检测影响物体移动，但是又想检测到碰撞这时用到触发器（Trigger）

b.触发器用来检测一个物件是否经过空间中的某个区域

10.Unity3d 中的碰撞检测和触发检测的区别？

答：碰撞检测和触发检测的条件是两个物体都要有碰撞器，其中一个物体身上有刚体（运动的物体有刚体），触发检测将碰撞器上的IsTrigger=true，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。

a.如果不想让碰撞检测影响物体移动，但是又想检测到碰撞这时用到触发器（Trigger）

b.触发器用来检测一个物件是否经过空间中的某个区域

**第十二天**

1.Material 和 Physic Material 区别？

答：PhysicMaterial 物理材质：物理材质描述，如何处理物体碰撞（摩擦，弹性）。Material 材质（材质类）为了获得一个对象使用的材质，可以使用 Renderer.materia 属性

2.UGUI和NGUI的区别？为什么不使用NGUI？

答：1、uGUI的Canvas 有世界坐标和屏幕坐标，NGUI有2D和3D区别。（重点）

2、uGUI的Image可以使用material。（重点）

3、UGUI通过Mask来裁剪，而NGUI通过Panel的Clip

4、NGUI的渲染前后顺序是通过Widget的Depth，而UGUI渲染顺序根据Hierarchy的顺序，越下面渲染在顶层。（重点）

5、UGUI 不需要绑定Colliders，UI可以自动拦截事件。（重点）

6、UGUI的Anchor是相对父对象，没有提供高级选项，（重点）

7、UGUI没有Atlas一说，使用Sprite Packer。（重点）

8、UGUI的Navgation在Scene中能可视化。

9、UGUI的事件需要实现事件系统的接口，但写起来也算简单。

10、NGUI功能更丰富一些

之所以不用NGUI是因为UGUI是Unity官方推出的，慢慢会成为制作UI的主要工具，配套的插件也越来越多，但是具体使用NGUI还是UGUI还要看公司这边，因为这两个我都用过一段时间。

3.Unity3d中static batching（静态批处理）和dynamic batching（动态批处理） 各有什么用？

答：Dynamic Batching（动态批处理） 不需要任何操作，只要共享材质(即使是不同的Mesh模型也可以)，就会自动被合并。可以自由移动，旋转（重点）

**Dynamic Batching（动态批处理） 要求：**

(1) 模型文件共计顶点数不超过900。（重复使用同一个Mesh不计）（重点）

(2) 单个物体可以不超过300顶点，Shader可以有法线UV。 但如果Shader使用了 UV0 UV1两套UV,或者Tangent切线的话，单个物体只能不超过180点

(3)游戏对象使用相同模型和材质时，只有相同缩放(即xyz等比缩放,浮点尾数可以有细微差)的会被合并。（重点）

(4)场景烘焙：烘焙后同材质将不会被烘焙。lightmap 有隐藏的材质参数：offset/scale, 所以使用lightmap的物体不会被合并

(5)Shader不能使用多Pass:多Pass的Shader会破坏Dynamic Batching

**Static Batching（静态批处理）**

运行游戏后将一组游戏对象的多个模型会被动态合并为1个。这组游戏对象所有使用同一材质的在一个DrawCall来完成。这些游戏对象运行后无法移动、缩放旋转。但是Drawcall一定是最大化合并的，并且不受动态合并的诸多限制

注意：1，所以过多的batch会增加内存占用。例如场景里的树群就不适合Static Batch，而适合动态合并。（重点）

实现方法:

(1)MeshRenderer勾选Batching Static： 勾了即可（重点）

(2)代码中使用UnityEngine.StaticBatchingUtility实现(可以在任何平台调用):

1)将所有要合并的静态物体(不须勾Batching Static)放入统一一个root

2)StaticBatchingUtility.Combine(root); 之后就合并好了！

区别：勾选Batching Static：完全自动合并,在MeshFilter里显示的是 Combined Mesh(root:scene)。合并后不能移动

4.Unity3d中Awake和Start 谁先执行，update和fixedUpdate 有什么区别？

答： Awake先执行。Update是在每次渲染新的一帧的时候才会调用，FixedUpdate，是在固定的时间间隔执行，不受游戏帧率（fps）的影响 ，FixedUpdate的时间间隔可以在项目设置中更改，Edit->Project Setting->time 找到Fixed timestep。就可以修改了

**第十三天**

1，Unity3D中的协程（coroutine）和C#线程之间的区别是什么？

答：（1）多线程程序同时运行多个线程（并发） ，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。

（2）除主线程之外的线程无法访问 Unity3D 的对象、组件、方法。

（3）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。

（4）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，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。

（5）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

2，物体自旋转使用的函数叫什么？物体绕某点旋转使用函数叫什么？

答： transform.Rotate transform.RotateAround

3，Unity3从唤醒到销毁有一段生命周期，请列出系统自己调用的几个重要方法。

答：Reset –>Awake –>OnEnable –> Start –> FixedUpdate --> Update –> LateUpdate –>OnGUI –> OnDisable –> OnDestroy

4，Unity3D 如何在场景中动态加载的数据？

答： Resource.Load

AssetBundle.Load

5，u3d 中，几种施加力的方式，描述出来

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

6，物理更新一般在哪个系统函数里？

答：FixedUpdate，每固定帧（固定时间）绘制时执行一次，渲染效率低下的时候 Update 调用次数就会跟着下降，FixedUpdate 比较适用于物理引擎的计算

7，移动相机动作在哪个函数里，为什么在这个函数里。

答：LateUpdate,是在所有 update 结束后才调，LateUpdate适合摄像机的跟随

8，反向旋转动画的方法是什么？

答：反转动画，将动画的速度调到-1

9，获取、增加、删除组件的命令分别是什么

答： 获取：GetComponent

增加：AddComponent

删除：Destroy

10，localPosition 与 Position 的使用区别？

答：localPosition：自身位置，相对于父级的变换的位置。 Position：在世界坐标 transform的位置

11，编辑器类存放路径是什么？

答：工程目录下的 Assets/Editor 文件夹下。

**第十四天**

1，当游戏中需要频繁创建一个物体时，我们需要怎样做能够提高性能？

答： 1、使用预制物体对象 Prefab

2、使用对象池技术，不使用时关闭，使用时打开

2，Vector3.forward与Vector3(0,0,1)是一样的意思对吗？

答： 对

3，哪个函数在游戏进入新场景后会被马上调用？

答： MonoBehaviour.OnLevelWasLoaded

4，Unity3d中Resources目录一般用来放些什么，打包的时候会有什么影响？

答： Resource一般用来放置一些需要动态加载的资源，打包程序的时候会将Resource目录下的全部文件都加密压缩打进包内

5，Unity销毁GameObject的方式，简述Destroy和DestroyImmediate的区别？

答：Destroy销毁场景中的物体，但是内存中它还是存在的，只有当内存不够时，机制才会将它销毁并释放内存。

答：而DestroyImmediate会立即将对象销毁，并且将内存释放。

6，Unity中如何派发事件（消息）

答：在脚本里的Update函数里调用EventDispatcher.Instance().OnTick();就可以了

**第十五天**

1，LOD 是什么，优缺点是什么

答：LOD 技术即 Levels of Detail 的简称，意为多细节层次。LOD 技术指根据物体模型的顶点在显示环境中所处的位置和重要度，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。

优点：可根据距离动态地选择渲染不同细节的模型

缺点：加重美工的负担，要准备不同细节的同一模型，同样的会稍微增加游戏的容量。

2，MipMap 是什么？作用？

答：在三维计算机图形的贴图渲染中有一个常用的技术被称为 Mipmapping。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 或者 mipmap。

3，请简述一下对象池原理,什么情况下使用？

答：对象池的理念其实是非常简单的。我们将对象存储在一个池子（队列）中，当需要时在再次使用，而不是每次都实例化一个新的对象。池的最重要的特性，也就是对象池设计模式的本质是允许我们获取一个“新的”对象，而不管它真的是一个新的对象，还是循环使用的对象。对象池通常用于在那些需要经常创建又销毁的对象，比如我们场景中的小怪，经常需要创建和销毁，我们就可以在小怪销毁的时候，不真正的销毁，而是把它放入对象池中，等到下一次我们需要创建小怪的时候，再从对象池中取出来。

4，使用NGUI或UGUI开发滚动视图，当滚动列表达到几百条的时候，如何才能保证界面的滚动流畅运行？

答： 1. 每个滚动条目都是同一个预设体的实例

2. 做缓存，只要实例化视野范围内滚动条目，往上超出视野部分，自动填补到视野的下面

5，Unity资源加载的有几种方式，简述assetbundle

答： 1. 直接在脚本中public一个对象，然后在监视器面板进行赋值

2. 直接在程序中进行find查找

3. Resource.load

4. 把资源打成Assetbudle，然后用的时候load进来assetbundle，就是对资源的打包处理，同时这种资源格式便于从互联网上下载

**第十六天**

一，DrawCall

1，什么是DrawCall？

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

2，Static Batching 静态批处理

答：场景中有很多游戏对象，其中静态对象（Inspector勾选Static的）可以通过静态批处理来优化DrawCall。

3，Dynamic Batching 动态批处理

答：静态批处理需要给对象设置成静态，而动态批处理，则不需要，非静态的对象，系统自动做批处理。

1），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。

2），如果你的着色器使用顶点位置，法线和UV值三种属性，那么你只能批处理300顶点以下的物体；如果你的着色器需要使用顶点位置，法线，UV0，UV1和切向量，那你只能批处理180顶点以下的物体。

3），不要使用缩放。分别拥有缩放大小(1,1,1) 和(2,2,2)的两个物体将不会进行批处理。

4）、统一缩放的物体不会与非统一缩放的物体进行批处理。

5），使用缩放尺度(1,1,1) 和 (1,2,1)的两个物体将不会进行批处理，但是使用缩放尺度(1,2,1) 和(1,3,1)的两个物体将可以进行批处理。

6）、使用不同材质的实例化物体（instance）将会导致批处理失败。

7），拥有lightmap的物体含有额外（隐藏）的材质属性，比如：lightmap的偏移和缩放系数等。所以，拥有lightmap的物体将不会进行批处理（除非他们指向lightmap的同一部分）。

8），多通道的shader会妨碍批处理操作。比如，几乎unity中所有的着色器在前向渲染中都支持多个光源，并为它们有效地开辟多个通道。

9），预设体的实例会自动地使用相同的网格模型和材质。

二、物理组件

1，设置一个合适的Fixed Timestep。

并不是所有的游戏中物理计算都需要0.02秒执行一次。因此这个值，可以针对项目慢慢调试，设置出一个比较合适的值，这样即完成了物理计算，也可以减轻CPU的负担。

2，不要使用MeshCollider

原因很简单，因为MeshCollider太复杂了,MeshCollider是按照模型网格进行设置碰撞器的。

三，GC(Garbage Collection垃圾回收)

优化GC，就是减少对GC的调用。

1，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。

2，那么GC什么时候会触发呢？两种情况：堆的内存不足时，会自动调用GC。手动的调用GC,我们就不能频繁的触发GC。GC的优化说白了也就是代码的优化。

3，以下几点是需要注意的：

1），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。

2），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。

3），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。

4），使用“池”，以实现空间的重复利用。

5），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。

四，代码质量

1, 最好不要频繁使用GetComponent，尤其是在循环中。

2，善于使用OnBecameVisible()和OnBecameInvisible(),来控制物体的update()函数的执行以减少开销。

3，使用内建的数组，比如用Vector3.zero而不是new Vector(0, 0, 0);

4，对于方法的参数的优化：善于使用ref关键字。

5, 对于函数最好不要超过30行，函数越精简越好

**第十七天**

1,服务器端：

第一步：创建一个用于监听连接的Socket对像；

第二步：用指定的端口号和服务器的ip建立一个EndPoint对像；

第三步：用socket对像的Bind()方法绑定EndPoint；

第四步：用socket对像的Listen()方法开始监听；

第五步：接收到客户端的连接，用socket对像的Accept()方法创建一个新的用于和客户端进行通信的socket对像;

第六步：通信结束后一定记得关闭socket;

2,客户端：

第一步：建立一个Socket对像；

第二步：用指定的端口号和服务器的ip建立一个EndPoint对像；

第三步：用socket对像的Connect()方法以上面建立的EndPoint对像做为参数，向服务器发出连接请求；

第四步：如果连接成功，就用socket对像的Send()方法向服务器发送信息；

第五步：用socket对像的Receive()方法接受服务器发来的信息 ;

第六步：通信结束后一定记得关闭socket；

3,分包和粘包

分包：发送方发送字符串”helloworld”，接收方却接收到了两个字符串”hello”和”world”

粘包：发送方发送两个字符串”hello”+”world”，接收方却一次性接收到了”helloworld”

4,socket环境有以上问题,但是TCP传输数据能保证几点:

1.顺序不变,例如发送方发送hello,接收方也一定顺序接收到hello,这个是TCP协议承诺的,因此这点成为我们解决分包,黏包问题的关键.

2.分割的包中间不会插入其他数据

5,目前最常用的协议标准是

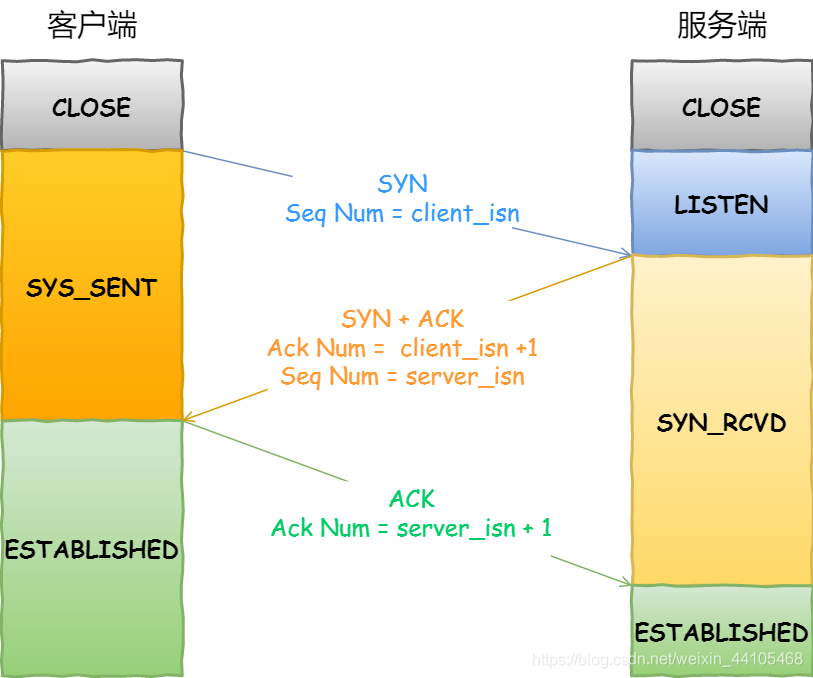
消息头部(包头：消息总长度)+协议号(当前消息是干什么的)+消息长度(消息正文长度)+消息正文

消息头部(包头：消息总长度)+协议号(当前消息是干什么的)+消息正文

1. tcp和udp的区别有：

1、udp是无连接的，tcp是面向连接的；2、udp是不可靠传输，tcp是可靠传输；3、udp是面向报文传输，tcp是面向字节流传输。

**7，TCP三次握手的过程**



第一次握手：

⼀开始，客户端和服务端都处于 CLOSED 状态。

服务端：主动监听某个端⼝，处于 LISTEN 状态。

客户端：随机初始化序号（ client\_isn ），将此序号置于 TCP ⾸部的「序号」字段中，同时把

SYN 标志位置为 1 ，表示 SYN 报⽂。接着把第⼀个 SYN 报⽂发送给服务端，表示向服务端发起连接，该报⽂不包含应⽤层数据，之后客户端处于 SYN-SENT 状态。

第二次握手：

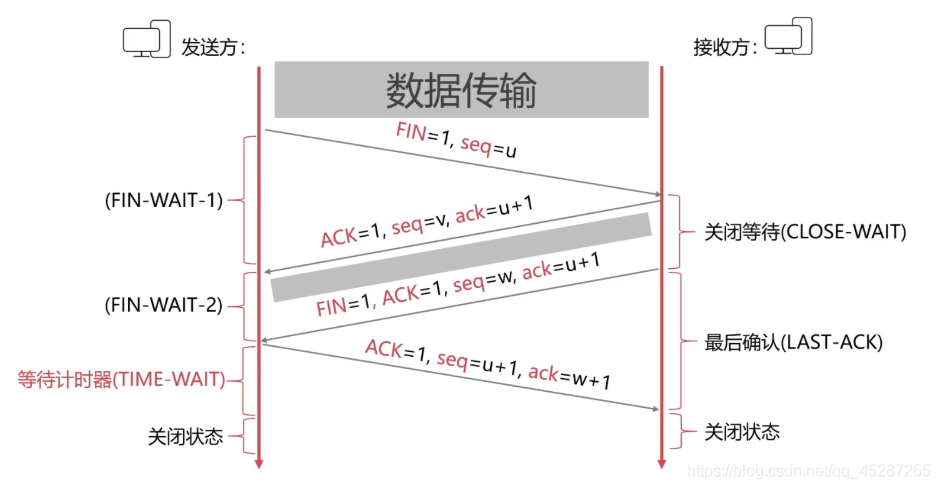
服务端：在收到客户端的 SYN 报⽂后，服务端也随机初始化⾃⼰的序号（ server\_isn ），将此序号填⼊TCP ⾸部的「序号」字段中，其次把 TCP ⾸部的「确认应答号」字段填⼊ client\_isn + 1 , 接着把SYN和 ACK 标志位置为 1 。最后把该报⽂发给客户端，该报⽂也不包含应⽤层数据，之后服务端处于SYN-RCVD 状态。

第三次握手：

客户端：在收到服务端报⽂后，还要向服务端回应最后⼀个应答报⽂，⾸先该应答报⽂ TCP ⾸部 ACK标志位置为 1 ，其次「确认应答号」字段填⼊ server\_isn + 1 ，最后把报⽂发送给服务端，这次报⽂可以携带客户到服务器的数据，之后客户端处于 ESTABLISHED 状态。

服务端：收到客户端的应答报⽂后，也进⼊ ESTABLISHED 状态。

**8，四次挥手的过程如下：**



第一次挥手：

客户端向服务器发送一个 FIN 数据包（FIN = 1，seq = u）主动断开连接，报文中会指定一个序列号。

告诉服务器：我要跟你断开连接了，不会再给你发数据了；

客户端此时还是可以接收数据的，如果一直没有收到被动连接方的确认包，则可以重新发送这个包。

此时客户端处于 FIN\_WAIT1 状态。

第二次挥手：

服务器收到 FIN 数据包之后，向客户端发送确认包（ACK = 1,ack = u + 1），把客户端的序列号值 + 1 作为 ACK 报文的序列号值，表明已经收到客户端的报文了

这是服务器在告诉客户端：我知道你要断开了，但是我还有数据没有发送完，等发送完了所有的数据就进行第三次挥手

此时服务端处于 CLOSE\_WAIT 状态，客户端处于 FIN\_WAIT2 状态

第三次挥手：

服务器向客户端发送FIN 数据包（FIN=1，seq = w），且指定一个序列号，以及确认包（ACK = 1， ack = u + 1），用来停止向客户端发送数据

这个动作是告诉客户端：我的数据也发送完了，不再给你发数据了

此时服务端处于LAST\_ACK状态，客户端处于TIME\_WAIT状态

第四次挥手：

客户端收到 FIN数据包 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 + 1 作为自己 ACK 报文的序列号值

此时客户端处于 TIME\_WAIT 状态。

需要过一了一定时间（2MSL）之后，客户端发送确认包（ACK = 1， ack = w + 1），此时客户端才会进入 CLOSED 状态，以确保发送方的ACK可以到达接收方，防止已失效连接请求报文段出现在此连接中。

至此，完成四次挥手。

**第十八天**

1.Array（数组）优缺点

答：优：存储在连续内存上；内容都是相同类型；可以通过下标访问。

缺：必须指定其长度，元素插入也不方便（过长浪费内存，多段会溢出）

2.ArrayList（动态数组）优缺点

答：优：可动态增加和删除 ，可存储不同类型；

缺：类型不安全，因为存储的是不同类型，都会当object来处理（插入值类型会发生装箱，索引取值会拆箱）

3.List<T>泛型

答：相当于 Array和ArrayList的结合（避免了拆装箱操作，同时也可以增加和删除，泛型T规定了其类型）

4.LinkedList<T>链表

答：适合于元素数量不固定，需要两端存取且经常增减节点

最大的特点是存储在内存空间不一定是连续的（意味着无法利用下标查找）插入和删除操作无需调整结构容量，靠各个对象的指针决定

5.Dictionary<K,T> 字典

答：Dictionary<TKey,TValue>是表示键值对的集合，Dictionary<TKey,TValue>类是作为一个哈希表来实现的，通过键来检索值的速度非常快（检索速度取决于为 TKey指定的类型的哈希算法的质量），接近于 O(1)，效率比List<T>高很多。

6.链表反转？

答：第一种是需要开一个新的链表，将原链表的元素从后到前的插入到新链表中（也就是原链表第一个元素被插入成新链表的最后一个元素）。

第二种是不需要开新的链表，而是逐步反转原链表中元素的指向，例如：原链表1->2->3->4->null  被逐步修改为

①2->1->null、3->4->null ②3->2->1->null、4->null ③4->3->2->1->null最后再将head指向4。

**第十九天** 面向对象七大原则

(1)单一职责原则

一个类或者一个方法尽可能的功能单一的

(2)开闭原则

软件实体应当对扩展开放，对修改关闭，开闭原则中原有“开”，是指对于组件功能的扩展是开放的，是允许对其进行功能扩展的；开闭原则中“闭”，是指对于代码的修改是封闭的，即不应该修改原有的代码。

(3)里氏替换原则

子类型必须能够替换掉它们的父类型,所有引用父类的地方必须能透明地使用其子类的对象。子类可以扩展父类的功能，但不能改变父类原有的功能.可以想想动态数组的使用

(4)迪米特法则

迪米特法则又叫最少知道原则,每个类都减少了不必要的依赖，因此的确可以降低耦合关系,只有使各个模块之间的耦合尽量的低，才能提高代码的复用率。

(5)依赖倒置原则

高层模块不应该依赖低层模块，二者都应该依赖其抽象；抽象不应该依赖细节；细节应该依赖抽象。中心思想是面向接口编程可以通过抽象类或者接口来进行函数和方法的调用,子类继承抽象类或者接口实现的是具体的细节.

(6)接口隔离原则

一个类对另一个类的依赖应该建立在最小的接口上

(7)合成/聚合原则

也有人叫做合成复用原则，及尽量使用合成/聚合，尽量不要使用类继承。换句话说，就是能用合成/聚合的地方，绝不用继承。

**第二十天 项目中UI系统和框架**

项目中UI系统:1登录和注册界面系统,2签到系统,3主界面系统,4活动系统,5仓库系统(背包系统),6商城系统,7金币和钻石系统,8任务系统,9装备系统,10升级系统(角色升级和武器升级),11抽卡系统,12排行榜系统,13好友系统,14物品兑换系统,15邮件和公告系统,16设置界面系统,17关卡系统,18战斗UI系统,19战斗背包系统,20结算系统,21LoadingUI界面系统.

项目中UI框架描述,三种角色:

1. **UIViewBase作为所有UI界面系统类的基类(父类)**
2. **具体的UI界面系统类**,UI界面系统类继承UIViewBase
3. **UI界面系统管理类**:本身是普通单例模式,管理UI界面

**框架描述:**

1,我会设计一个UIViewBase作为所有UI界面系统类的基类(父类),UIViewBase会继承MonoBehaviour,对应的UI界面系统类挂载到对应UI界面预设物,每个UI界面都会制作一个预设物,每个UI界面脚本都是一个Unity单例,我会在UIViewBase中设计Unity单例其他脚本继承UIViewBase自然他也是Unity单例模式了,UIViewBase中会设计一些方法和字段,例如:初始化方法,生命周期方法,Open()方法(设置从隐藏状态到显示状态),Hide()方法(设置从显示状态到隐藏状态)等等

2,我会再设计一个UI界面系统管理类,在管理类设计一个字典,字典的Key值是一个字符串(UI脚本名和UI界面预制物名字设置样),Value值是对应UI界面系统类的实例的应用,在UI界面系统管理类中会加载初始化的UI界面预制物并将对应UI界面系统类的实例添加到字典中.

3,在UI界面系统管理类设计一个方法来对UI界面进行切换.UI界面系统管理类本身是一个普通的单例模式.并不是一开始把所有UI界面初始化,而是我需要哪个UI界面系统我就初始化哪个UI界面系统,初始化加载完的UI界面系统会放到字典当中,方便下一次再次使用,

UI界面的切换是做显示和隐藏处理的,而不是加载和删除.

网络通信消息协议组成:消息协议号(代表当前消息是什么), 用户ID(谁发的消息),具体的消息内容。例如:

**签到系统:七天签到或者月签到**

我在打开签到UI界面之后,向服务器发送获取签到信息的请求,请求消息包括(消息协议号,用户ID),服务器给我一个响应信息,响应信息包括(消息协议号,用户ID,1到7的数字的数组,里面有哪些数字就认为哪天是被签过到的),当我点击签到按钮的时候向服务器发送签到请求信息,请求信息包括(消息协议号,用户ID,1到7的数字),服务器响应给我一个签到成功.

**背包系统**:我在打开背包UI界面之后,向服务器发送获取背包信息的请求,请求消息包括(消息协议号,用户ID),服务器给我一个响应信息,响应信息包括(消息协议号,用户ID,背包物品ID数组),我会根据背包物品ID数组和本地配表将我的背包物品显示出来,背包UI界面里面有”出售”和”使用”按钮,例如”使用”按钮,点击”使用”按钮时候向服务器发送使用请求信息,请求信息包括(消息协议号,用户ID,背包物品ID),服务器响应给我一个使用成功等等.

**第二十一天 AssetBundle包更新流程**

1. 将Unity中的资源进行分类,比如对资源做成预设物(UI系统界面,特效,场景物体等等),然后对预设物进行打AssetBundle包
2. 将资源打包成AssetBundle资源包并存放到Unity特殊文件夹StreamAssets下
3. 将所有AB文件的生成MD5码存放在StreamAssets/file.txt文件中，生成的版本文件存放在StreamAssets/version.txt
4. 将热更新资源包上传到资源服务器
5. 首次启动游戏，将StreamAssets文件夹下的所有文件复制到PersistentDataPath路径下，(当然也可以只解压file.txt和version.txt文件，若只解压这两个文件，之后加载资源时需要先判断PersistentDataPath中是否存在，若有则直接用，若无则再从StreamAssets文件夹下加载)
6. 上步完成资源解压后，获得到资源服务器上最新的版本号与本地version.txt中的版本号进行对比，若不一致则显示UI提示玩家进行更新.
7. 若玩家选择更新后，则对比从资源服务器下载的file.txt和本地file.txt的MD5码值，MD5码值不一致的文件就下载。下载到PersistentDataPath路径下,下载完成后，将新的file.txt和version.txt文件也下载到PersistentDataPath路径下,进行下一次资源热更新对比.
8. 若项目中使用资源需要从PersistentDataPath路径下加载资源多运行的项目中使用.