工作日记

1：C#中删除list中指定value

>在某一次面试的时候被问到这个问题。使用foreach或者for遍历一个List的时候，在过程中删除了List中的一个元素则会改变List内部的顺序，导致后面的遍历不正确。解决这个问题，比较中庸的方法是单独起一个List，把要删除的元素先保存起来，然后再去遍历这个新的List，针对每一个元素去调用源List的删除方法。还有一个比较取巧的方法对List采用从后往前的遍历方式。

2：世界BOSS界面切换到个人BOSS界面，偶现Bug

>这是一个变量没有被正确赋值而被使用的例子。

3：世界地图界面打开后左侧列表的slider不显示，把刷新slider放在界面打开的下一帧刻有解决，同类问题还有之前世界BOSS界面直接切换到野外BOSS或者个人BOSS时也会出现

>在Open方法里去实例化ScrollView和Grid，然后刷新，Slider并不会显示出来，只有延迟到下一帧去实例化并刷新才能解决

4：循环以一个prefab模板实例化多个的代码里，一定要先把新生成的GameObject.SetActive(true)，在unity编辑器里，一个uisprite的activeinhierarchy等于false时，snap按钮不起作用

5：世界地图和小地图在公用同一个MapManager时，出的问题。

>两个UI同时共享一个Manager，虽然在设计上有些省事的地方，并且两个UI还比较相似，但在Manager里稍不注意就会出错，导致问题，很多时候方法也要针对两个UI单独写，所以以后尽量一个UI只对应一个Manager。

6：世界地图在获取鼠标坐标并转化到世界3D世界坐标系中时，实现的方式很简单粗暴易懂。

>具体需求：点击地图界面，然后寻路到对应的世界坐标点。先是获取到鼠标点击的坐标，然后调用UICamera.mainCamera.ScreenToWorldPoint方法，该方法把一个屏幕坐标转换成世界坐标，得到世界坐标后把该坐标值赋值给一个UI上的组件，然后再通过UI组件拿到其局部坐标，这样就得到了鼠标点击的位置落在了UI地图上的某个点上，再通过反向推倒，能得出该UI地图上的某一个点对应的3D场景中的世界坐标。

7：世界地图开始做的第一天，主角的heroTag可以正常显示出来，总是在寻路开始的一瞬间，被删掉

>一个不正常的现象出现了，那么一定是一个bug引起的，没有无缘无故的爱与恨，刚开始可以被正常的显示，可马上就不见了。世界地图的界面prefab里有一个子物体用来存放路径点，Update方法里每一帧都会先删除所有的路径点再重新生成，而heroTag这个sprite不幸的也在这个子物体下，所以在Update执行的第一次就被删除了，后来把heroTag单独拿了出来。

8：一个window类（继承自monobehavor的类）挂载gameobject上，obj的active为false时，脚本执行的协程都会停掉

>这是一个关于unity协程的问题，协程是unity的MonoBehaviour里的方法，一个GameObject上挂载了多个继承自MonoBehaviour的脚本，其中某一个脚本的方法里执行了StopAllCoroutine方法，则会停掉该脚本里的所有协程，并不会影响其他脚本里的协程。但如果把GameObejct的active设置成false，则会打断所有脚本的协程，类似执行了yield break，GameObject的active重新被设置为true也不会再次唤醒协程。

9：飘字公共接口当时的疑问

>当时对概念理解不清楚，并不是什么疑问

10：在用foreach便利C#的字典时，每次都会当前值判空，是否需要？字典里的value是否可以为null，尝试一下

>字典里，List里的值都有可能为空，所以判空是很有必要的。

11：在代码里遇到数值类的一般要定义成常量，最好统一放在一个常量类里，字符串也要这样处理，可惜现在的项目里压根没有这种想法

>尽管整个项目里没有这种类似的机制，自己的代码里用到的常数都需要给单独用一个变量保存起来，最好是常量。数据有变化，只需要改一处就可以，并且后续如果变量通过表格读取方式获取，则更是方便，和代码逻辑没有关系。相当于把代码和代码里用到的数据隔离开了。

12：野外BOSS和个人BOSS左侧列表的刷新和世界地图左侧列表的刷新方式不一致，如果刷新scrollview滑动条会拉到最下方

>生成一个Grid列表后，在下一帧刷新该列表最终实现滑动条在最顶上。先刷新Grid.Reposition，再刷新ScrollView.ResetPosition。如果调用的是ScrollView.UpdatePosition则滑动条会莫名其妙的滑动到最下面。

13：学习连斩的滑动条效果，因为图是个半圆，不知道用的是什么组件

14：王伟说项目里的脚本meta文件不用提，资源meta文件要提，为什么？

>事实证明不管是什么类型的文件meta文件，都应该提交，脚本的meta可能在挂prefab的时候有用

15：想测试一下在lable和sprite的active为false时，给label赋值能否成功，给sprite换图能否成功，调用sprite的makepixelperfect能否成功

>UILabel在隐藏状态下可以赋值，但不能给ambigiousFont换值，因为即使在隐藏状态下换了值，最后显示出来也不是被换的那个值。

UISprite在隐藏状态下可以赋值，但不能执行makepixelperfect。

16：UIGrid的repositionNow是否能正常执行？测试一下

>repositionNow是一个字段，会不会立即执行得不到保证，可能是当帧也可能是下一帧刷新。

17：C#实现比较器，双重比较，尝试使用加权重的方式来简单处理

>已经实现，并且识记

18：有时间熟悉一下label上的各种属性代表的意思，还有sprite上的

>已经了解

19：策划表格里的百分比，有的是用0.05（代表5%），有的是5（代表5%），至于两种方式，后者更利于程序读取使用

>这是一个规范问题，没有好坏，但一定要在事前统一并保持一致

20：UIGrid组件，获取其子transform数量，是找的active=true的，如果UIGrid和table配合使用，UIGrid的gameObject为false时，UIGrid组件获取的子transform数组就是0

>这是在做成就界面时，获取某一类成就transform列表时报的错，开发中最好不用通过uigrid.transform.childcount等方式去遍历gird下的元素，最好还是保存并用key维护起来管理

21：在做世界地图的时候，有一处代码时根据grid的childcount是否为空来做判断，但调用的却是延迟一阵清空元素的函数，导致在判断的时候获取到的childcount不等于0（虽然前面已经调用了UIGrid的清空函数）

>提供的扩展方法之前的一个是下一帧清空，所以调用了清空，但在当前帧接下来的逻辑里拿到的count还是原始的，导致了错误

22：在做UI时，有明显层次结构的父节点上最好不要挂东西，并且位置是(0, 0, 0)最好。如果父节点上挂组件（spsrite和label）它要参与和背景图对照位置，一旦出了问题在他调整位置的时候，它的子物体也都会受牵连，最常见的是把背景挂在父节点上，这简直就是愚蠢，没脑子的做法

>最后一句有点极端了，只能说普遍情况下前面的方式是比较好的，但在做item的tips时就用到了最后一种做法拼UI。要调整tips的透明度，最好的方式就是调整整个整个父节点的，也就是整个大背景的。所以就把tips的大背景放到了tips的根节点上，并且这样做也方便调整位置

23：在往某个指定图集里添加sprite时，一定看清图集名字，因为unity默认给你选中你上次使用过的图集，一不小心就会打错图集

>打图集前看好，默认选中的是哪一个图集

24：在做世界地图时，各种缩略图上的标识在生成的时候给gameobject的name命名成了id，这样在点击某个标识的时候，可以通过click回调里的参数go，来取出其名字然后得知是点击了哪个标识，做寻路的逻辑。在任务系统里也是如此，在文位系统里右侧的列表里也是如此。这种设计下，如果gameobject的名字在某处被无意中修改，或者是被恶意修改，再或者在后续的逻辑里可能需要用到给gameobject的name属性的，都会导致问题。好的设计应该是单独维护起来成一个数据结构，以id作为key，在批量生成这些gameobject的时候把key和gameobejct具体信息一起存起来，注册事件，传入id，在具体的点击处理函数里根据id去数据结构里获取对应的gameobject的信息。高内聚，安全，易扩展

>后来已经把世界地图这块代码改成了高内聚的版本，以后做任务系统时可以借鉴

25：在领取奖励的时候要先检测背包的空余格子是否足够装得下即将获取的物品数量，而且在领取奖励的消息里还需要传入的背包的空余格子oid集合，这种判定应该在服务器做，如果背包格子不足可以发一个系统提示消息，也可以走邮件，都行，但现在的做法简直是最愚蠢的。

>目前的背包系统，实在有待改进，逻辑上问题不大，但功能逻辑组织，和对外接口上不够友好

26：在商城的购买框，以及通用的道具获取框，里的最大数量最小数量要有规定

>这个后来已经作了规范，是一个统一固定的值，由策划配置

27：背包里的每个格子里能存储的同一类型道具的数量上限要有规定

>这个后来已经作了规范，是一个统一固定的值，由策划配置

28：在显示道具数量，已有货币数量的地方，超出10万的要有数量折叠，这些都要预定义好规则，否则后期开发会各种不一致

>这些算是开发初期的规则性东西，最好也是做成一个公用组件或者方法，目前项目是这么做的

29：现在工程里的一些公用图片，本应该是同一份资源，可却在很多图集里出现，导致界面开发的找到一个就用哪个，出现不一致，这种公用的图片，就应该被打进同一个图集里，以后的修改都在这个图集里做修改。

>在后期的优化图集过程中，已经作了很大程度上的优化

30：现在项目开发的过程中，同时有好几个人在改公用图集，这样就造成了svn的冲突，现在的做法是谁改图集之前先确保其他人没改，这种做法其实可以改变一下，单独一个电脑去最这种事情，谁要去改公用的东西，就去同一台电脑上改，即使上个人的修改没提交，也不会因为后来人的修改而引起冲突，只会把上一个人的修改一并提交。

>公用一台电脑维护图集不现实，还是尽可能的快速修改，快速提交比较靠谱，如果需要长时间使用，则告知组里其他人

31：策划的表格字段应该有详细的文字描述，可惜没有，字段的名字有的策划用英文（英文还是自己想当然的居多，主动查询字典尽量靠谱的居少），有的策划甚至在同一张表里字段名有的英文有的拼音，有没有下划线什么时候需要下划线规则也看不出来，这些都是规范，但同样没有预定义。

>策划的表格命名也应该有一套规则，但现在并没有

32：消息协议应该统一由服务器定，消息协议的名字应该尽可能的让人看出来消息是一对儿一对儿的，即使不看注释，C2S和S2C，消息名后面的应该保持一样才能一眼看出是一对儿的。消息的注释，以及收发时间点不写，导致沟通成本。

33：消息协议里，数据配表里，代码里，参数里，等等，遇到时间的参数要注明是什么单位，是秒还是毫秒，还是什么格式。不管在哪里出现了，一定要注明它代表的单位，要不然又是隐患（异常，难查）

34：代码的函数参数里的id，要注明是哪个表的id，数据配置表里引用到了其他表的id，要注明是哪个表的，但我做到现在为止，也只看到过一个表是这么做的，字段处注明了是哪个表的id。

35：表格里的时间格式，要固定，用-还是用,

36：曾经遇到过一个Bug，邮件id用long来存储，从服务器接收到邮件列表后要在客户端排一个序，排序函数里接受两个Long型参数，返回值是int，函数里直接把两个long值相减返回，有可能导致long转int时溢出，所以导致异常。

37：数据表里有很多关联，比如在道具表里添加一个道具，同时要在道具来源表里添加一条记录，如果该道具是可以在商城里购买的，还要在商城表里添加一条记录，这种新添加的数据记录还算好，只是添加。如果是把某个基本数据表的记录做了修改，要修改所有引用该表字段的表，有没有那么一种方式，可以实现改表格中的一处，自动化的执行其余的工作，所有和本表的字段有关联的统统得到更新，这种工作流，要在项目一开始定好规则，并且由一个人单独去扩展该自动化工具。

38：svn的commit之前一定要先uppdate，可总是有人记不住，往往这些人还是那种起了冲突后无脑覆盖的。对于策划和美术，他们不一定有程序熟悉svn的机制，一定要最培训，要不然只会当猪队友

39：有空学习一下项目里的事件系统，带类型的方便携带参数信息

40：之前在做世界BOSS伤害排行榜时，排行榜里维护的字段总是在被初始化之前执行到，结果为空，因为字段初始化放在了start里，可能不再这一帧执行，start不能保证在实例化的同一帧被调用，而awake可以保证在实例化的同一帧被调用，所以改成awake里初始化解决了问题。

41：今天尝试了一个很不错的方法用来提供公开的接口，函数名开头是for于命名规则,manager里接收服务器下发消息的函数开头是On并且尽量和下发消息名字保持一致（如果对于某些名词服务器的下发消息中和客户端中的英文单词不一致，可以用客户端自己的英文单词风格），manager里接收到下发消息后，更新数据缓存，然后去调用界面对应的函数，window里的同功能函数一般和manager里的一致（比如升级成功，，，）。

42：网络游戏中的物品随机掉落解决方案

43：在itembuywindow里按钮的uibutton挂在一个obj上，按钮的背景放在obj的一个子obj上，这种方式有什么好处

44：输入框的onChange事件绑定，使用的是EventDelegate.Set，而大多数其他的事件绑定都是使用EventDelegate.Get

45：代码里的除法，一定要判断被除数是否等于0，不要假定或依赖被除数不为零

46：transform的Find和FindCihld有什么区别

47：civilianwindow的345行，传入的item,在函数里访问到的总是同一个，还是最后一个，因为这整个绑定是在一个循环里做的，和野外BOSS的地图名字事件绑定同一个问题，闭包用错了

48：在输入框的输入值做判断时，一般有加号，减号，最大值按钮，先做最大值判断，再做最小值判断，因为能够购买的最大数量值，可能是0

49：神龙赐福活动，客户端先写代码，最后和服务器对消息，发现好多东西都没有考虑到，其实都是一些逻辑状态和逻辑边界值的判定情况，考虑不全面

50：更好的计时器实现方案

50：游戏的框架层代码一定要多写log，便于调试，以及便于日后上线查日志，查崩溃点。

平时自己的模块也要这么要求去做，代码运行起来，谁的错误一眼就能看出来

51：在做一个系统前，表格应该是策划和程序一起制定的，并且程序要告诉策划，表里的哪些数据可以改，哪些不可以改，要不然策划瞎几把改，一改程序就出错，这其中也有程序的问题，代码不够灵活多变不够容错。

52：找时间看一下全局协程接口，为什么要放在全局里还是不太明白，还有文位的结算面板里的倒计时会越来越快，不知道是不是协程的坑

53：熟悉游戏里的快捷键系统

54：学习使用unity的性能检测工具

55：amuletwindow.cs类的第47行，关于sprite处于隐藏状态下，组件上获取不到uisprite脚本，导致设置出错

56：有些UI的元素在初始化的时候，或者隐藏或者显示，这些不要依赖于UI的prefab设置，应该在代码初始化的时候设置隐藏或显示，否则prefab一改动，代码里又没有做隐藏或者显示的初始化，就会莫名其妙的显示出来一个界面

57：mainWindow.cs的842行，和RejuvenationManager的42行，竟然是先调用的界面初始化然后是roleinfo，如果主界面需要roleinfo的数据做初始化，就会报错，改方式

58：回春术技能转CD，仅仅使用组件就解决了

59：BlessItemInfo继承了MonoBehaviour后，该类对象的生命周期就不由得我来控制，DragonBlessWindow的m\_dicBlessItemInfo里维护的BlessItemInfo会为空，我只是对GameObject设置了active，BlessItemInfo对象什么时机下被销毁的（C#好像有一个机制，一个类对象生命周期后（被销毁），之前该对象的所有引用都会被置为null，不会出现C++里的空指针现象）

60：使用协程控制时间间隔类的逻辑，这个时间间隔会越走越快，使用invoke就不会，为什么，使用全局协程呢？

61：做元宝和绑定元宝居中显示时，元宝和绑定元宝的位置竟然都是0点，ngui的界面坐标是怎么工作的？这种现象透露出ngui的什么原理？

62：一个继承自MonoBehaviour的类，其创建方式只能通过AddComponent的方式，并且其生存周期也必须和一个GameObject绑在一起。在最初设计一个类的时候，以为这个类可以不需要继承自MonoBehaviour，就直接维护了一个Transform，后面这个类需要使用unityengine里的类方法时，就需要重新继承自MonoBehaviour，这时就会出现矛盾，一个集成自MonoBehaviour的类内部还维护了一个Transform，这本身就是这个类在设计上的矛盾，而且还会出现一个很奇怪的现象，这个类实例的生存周期不受程序的逻辑的控制，反而转由unity去控制，导致明明刚刚实例化出来的一个脚本，转眼就为null了。

63：在ngui中，任何一个可以被渲染的东西（假定为item），都必须放在一个uipanel下，ngui会去从下往上找item的第一个uipanel，找到后拿到uipanel的depth和item自己的depth共同决定了这个item在全局的渲染队列里的一个depth，之前拼ui时，经常出现一个问题，命名我的depth比它高，可还是被遮住了。原因在与它的uipanel高。

64：技能cd总是在切场景的时候卡住，有的是重新转cd，原因在于cd数据存储在了一个切场景会被重新创建的类里，导致原来的cd数据都没保存下来。

65：SVN提交备注不为空，每次提交写清楚这次提交的内容和是什么，是改Bug，是实现新功能，还是改需求

66：各种各样的小界面弹出与隐藏逻辑，最后需要统一给规则时，却无从下手

67：做回春术小界面时，因为buff描述和额外效果描述，不确定有几个，并且不确定描述的文字有几行，但还需要做自适应时，就挺费劲，这个有什么好的方法去对文本自适应，感觉应该用到table

68：测试当UILabel处于active为false时，赋值是否能成功，UISprite同样测试

69：unity的PlayerPrefs是个什么东西，它能做什么，它被广泛使用吗，它有缺点吗

70：有空看一下项目里的背包系统，根据第一家公司时的背包积累，尝试做一个背包系统出来

71：一个继承自MonoBehaviour的类，如果要在“构造函数”中做些什么，是不是要放在Awake()中，而不是放在其真正的构造函数中？如果放在构造函数中是否可以呢？

72：在最界面打开关闭动画时，炼神和竞技场两个界面在一次打开关闭后，第二次打开竞技场里的模型就不显示了，也没有报错，原因是竞技场里的模型展示用的旧方式，每一个模型都维护了一个摄像机，而这个这相机的位置还随着界面位置变化，在执行动画的过程中，可能出现了问题。炼神第二次打开后摄像机的视野就变得很小，也是同样的原因。

73：unity里的摄像机是个什么概念？

74：整个游戏里的小界面，确认框，弹出框，他们有各种不同的UI规则，有的有遮罩，有的没遮罩等等。当需要在关闭界面时关闭这些小界面，或者在切场景时需要统一关闭所有打开着的面板，或者esc键后关闭当前面板等等这些需求时，如果这些小界面不是被统一在一个frefab下而是散落在各个窗口里，就会是噩梦，无法统一判断和管理，UI框架制定之处就应该把这个规范给定下来：所有的小面板都必须放在一个大的window下，统一管理，并且这个window必须保证整个游戏运行期间处于打开状态。小界面的打开方式为Show，并且小界面的打开和关闭不能影响到统一window的状态。

75：一种删除List里指定value的元素的方式，把这个List里的元素再单独放到每一个List里，相当于中间又套了一层List，类似第一个项目里lua的处理方式。

（还可以倒着遍历）

76：游戏里的声音管理方式？

77：unity项目被svn管理时，在add一个资源文件时，如果不提交其对应的meta文件会有什么问题，如果在另一个人的电脑上生成了meta文件又提交上去了，会出现什么隐患，unity项目中哪些文件的meta文件是必须要提的，哪些是不用的，meta文件里对于unity工程中的不同文件来说，其含义也不相同，那么他们分别代表什么含义呢？

78：通过看UIResourceManager，总结协程的使用方式，以及哪些情况下可以使用协程得到巧妙的解决

79：山海之路在执行关闭动画的过程中，本来应该被Panel遮罩的图片显示了出来，是因为Panel的clip受到了其GameObject的transform的scale影响，加入scale不一致导致Panel的clip失败

80：StopAllCoroutine方法是在MonoBehaviour里的，它停止的是该脚本里的所有协程

81：把GameObject的active设置成false，挂载在该对象上的所有脚本里的所有协程都会被停掉，类似于yield break，把GameObject的active设置成true后也没用

82：在消息处理逻辑里做UI相关的逻辑，这种做法非常不靠谱，不仅仅是耦合了，还没有做好对服务器回来的数据维护工作。最起码也要做到在处理UI逻辑时判断UI是否存在

83：OnEnable和OnDisable两个函数的调用时机，和GameObject的activeInHierarchy字段一致

84：网络游戏的服务器在接收到客户端请求时，版本号检测与版本更新策略

85：WWW scene = new WWW(scenePath);这句代码是否是开启了一个WWW下载线程

86：当一个UILabel脚本处于enable=false时，给其设置ambigiousFont没效果，如果有需要用设置alpha值来实现隐藏/显示

87：游戏开发过程中的GM指令，现在的项目里很多的GM指令都是没有和消息挂钩，对于服务器而言他们是只修改了服务器维护的数据的值，并没有把消息法出来，导致执行了GM指令后还要做一些额外的操作来让服务器的最新数据同步到客户端。提供GM指令的时候一定要明确对应到某一条消息上

88：最新在遍历list的时候也喜欢用foreach方式，这种方式和以前的遍历索引并通过索引取值的方式相比，有什么区别么，性能上的，以及安全性上的，总结出一个结论来

89：OnEnable的执行顺序先于Start方法，对于一个集成自MonoBehaviour的类，其UI组件变量的查找与初始化最好放在Awake方法里

90：一个集成自MonoBehaviour的类，在Update方法里不要去访问该类的UI组件，更不要访问其他类的UI组件，这里的访问是给UI组件赋值。如果在Update里需要动态的给UI上的一些组件赋值，最好是去调用函数，在函数内部判断UI组件是否为空。

91：按下[z]键，开启自动挂机。可在某一个副本内，UI上显示着自动挂机标识，而在程序里控制自动挂机的字段却并不是标识自动挂机状态。像这样的UI单独显示，状态变量单独控制的方式下，很容易出现状态紊乱，程序内部的逻辑和UI的表现对应不上。较好的方式：状态的控制是一段内聚独立的代码，对外提供的改变状态的接口一定是统一的，最好只提供一个，并且UI的表现一定是受维护状态的代码驱动的。

92：在之前的UI功能里，一个单独的小item被抽象出一个类单独维护时，item的初始化都被放在了item实例化的地方，而且还不是一个函数。后续的UI功能里，针对这种写法做了一些改进，把item的初始化放在了item内部的一个私有方法里，并且这个私有方法在构造函数里被调用，实例化item的时候只需要通过构造函数传递item需要的参数即可，item内部维护了所有的数据和状态，高内聚的做法。

93：山海印的红点问题，红点事件明明派发出去了，可就是没有调用红点的注册回调，最初以为是因为注册回调的方式问题而导致其被自动或其他方式删掉了，最后定位倒问题，红点事件的枚举重复了。

94：最初接手神翼系统，关于神翼的穿戴总是莫名其妙的不太一致，最后定位到神翼的穿戴是一个广播，消息里的参数是一个玩家唯一id，而客户端在处理消息的时候并没有去区分这个id是主玩家的还是其他玩家的，所以导致的错误，这类的问题最好的解决方法：先根据id去区分主玩家和其他玩家，然后用两个独立的函数去做逻辑，命名清晰，永远不会错。

95：找时间去了解一下游戏场景中的3D角色头顶的名字label怎么显示在NGUI系统下的，以及如何实现转换的。

96：添加野外BOSS主界面按钮tips界面时，我把tips的面板做在了主界面的prefab里，FieldBossTips是一个GameObject，在prefab里是隐藏状态，在主界面组件查找的时候，要通过AddComponent的方式加载FieldBossTipsPanel这个类，代码执行后，可FieldBossTipsPanel的Awake方法并没有紧接着执行，因为FieldBossTips这个GameObject在prefab里是隐藏状态，所以在这种情况下给GameObject AddComponent一些脚本，这些脚本的Awake等一系列方法都不会被执行，只有当GameObject在编辑器里被设置为可见时，才会执行Awake防范，以及一系列事件方法会正确有序的执行

97：之前回春术等级面板，回春术升级面板，活动倒计时列表面板，野外BOSS主界面按钮tips面板，还有一个主界面下方红点提示面板，这些面板的逻辑都不写在mainwindow这个类里，而是单独起一个类来维护逻辑。而这些子面板在MainWindow这个prefab里又不能全部都是默认显示状态，要不然主界面会很乱，所以MainWindow这个prefab上的绝大多数子面板都是默认隐藏的。假如维护子面板逻辑的类不是继承自MonoBehaviour，则直接new一个类，然后在其InitUI函数里传一个子面板的transform供其维护。如果该类需要一些MonoBehaviour里的方法，则需要通过AddComponent的方式实例化子面板类，而这个子面板的GameObject在prefab里又是默认隐藏的，这时候子面板类的Awake就不会立即执行，在这种情境下，就不能把一些逻辑放到Awake里去做，最好单独再起一个InitUI方法来初始化组件和变量赋值。

98：一个UISprite从很小被拉伸到很大，在sample模式下图片两边没有图素，而用sliced模式下就很完美的切边，UISprite的这几个属性代表什么意思，他们在内部对于一张图片的处理规则是什么。

99：世界BOSS，世界地图，商城，神龙赐福等系统界面里有很多小的item，将其封装成一个单独的类去维护状态与逻辑，这是一个很好的做法。而且这些item类可以选择继承MonoBehaviour类或者不继承。如果不继承则使用new的方式实例化item类，并将transform传给item类让其维护，假如你需要MonoBehaviour里的一些方法，则不得不继承自MonoBehaviour，并且实例化item类的方式也变成了AddComponent，不再需要InitUI方法，如果item的GameObject实例化出来后直接显示出来，则可以将Init逻辑放到Awake里，如果item的GameObject实例化后是被隐藏的，那其Awake方法不会在实例化后紧接着执行，初始化失败，只能再次强制调用Init方法初始化一些逻辑。

100：消息流，在roleinfo的处理消息里去掉界面数据初始化请求，改成roleinfo去接，

101：代码里的基础事件不全，触发节点各写各的

102：Boss大厅的红点改成带数字的这个过程中，修改了一些代码，把以前判断是否显示红点的地方改成了统计红点个数然后再根据个数判断是否显示红点，回头想，假如当初做的时候就是以统计数据为基准，然后根据数据决定逻辑，就不需要改什么了，只需要把这个统计好的个数返回出来，数据推动逻辑

103：需要延时处理的逻辑，一般都是以起协程的方式实现。在做物品tips的时候，就遇到了这种情况，如果在两个item之间快速的滑动，会出现闪烁，为第一个item起来的协程还没有执行完，鼠标移动到第二个item上，又要为第二个item起协程，这时候规范的做法是把第一个item起的协程停掉。不停掉的话，在这里会出现闪烁，在窗口执行缩放动画的时候回导致崩溃，以为在协程等待的时间里，发生的逻辑不确定，而这又很可能对协程造成致命的危害。

104：一个界面，左侧有一个列表，选中某一个元素右侧区域的显示跟着变，这类需求，右侧区域的刷新提供一个通用的接口，根据传进来的左侧元素id刷新。在界面初始化的时候也可以初始化左侧列表后，直接选中某一个左侧元素刷新右侧区域。

105：启动一个协程，参数传递尽量使用方法签名，不要用方法名字符串，要不然都不知道该函数的使用地方有多少。但要尽快确认，启动一个协程两种传递参数的方式有什么差异么？

106：关于界面和数据的关系，假如在界面打开的时候请求了数据，那么数据回来后一定要刷新界面。如果在界面打开的时候不是请求数据而是直接从manager里拿数据，则manager需要一些保证，比如在界面打开前manager里有当前的数据，并且数据更新的过程中及时刷新界面。关键点在于数据的请求时机，前者是在界面打开的时候，而后者是在登录的时候，登录的时候会更好一点，比如显示红点就需要在登录的时候拿到所有数据。

107：调研一个背包系统的查询数据，对于服务器来说有多大负担，大多数的公司都是怎么做的背包数据同步。同样还有战斗模块，重点是死亡逻辑，以及场景中的sprite状态管理控制。

108：道具入包的bug前前后后总结，

109：最好不要有延迟处理的逻辑，延迟处理意味着在等待期间会发生很多无法预期的事情，导致莫名其妙并且难查的问题。更好的方式是异步操作，然后给回调，这样看协程也可以做到，但要做逻辑上的包装才行。延迟逻辑一般在UI表现上可能会用的多一点。

110：ToLower方法，Replace方法会改变原有的变量吗？测试一下

111：C#的各种数据结构可以添加Null吗，每一次的判空是否有意义？测试一下

112：温习回春术升级代码，并总结这类动态维护一个队列的代码规范。

113：EquipmentManager本是一个单例，可却公开了构造方法，并且允许这个类在外部实例化，而类内也有一个实例化操作，这样在游戏运行的过程中就有两个EquipmentManager类实例。千里之堤毁于蚁穴，好好的上层逻辑，却毁在了两个EquipmentManager实例这个错误里，而这个错误的根源是写代码的人不清楚单例的使用方式，更不懂单例的设计雷区，一个类既然选择单例那么再公开构造方法的行为就是傻逼，写出这种代码的人就是傻逼。

114：在和服务器通讯的设计里，假如一个消息的结构体里有10个数据元素，而并不是每次都是10个元素，所以服务器会采用没有的就不发的原则来节省服务器的消息量。但这对于客户端来说并不友好，有的同事就以服务器的数据为准则来缓存本地的数据，假如第5号数据元素没有而服务器不会发回来，但客户端的其他代码里却需要第5号元素的信息，这时去访问本地维护的数据时就会出错。好的做法是客户端维护的数据结构不依赖于服务器发回来的消息结构，因为消息结构只是为了通信目的，而客户端的逻辑是客户端写的也是客户端维护的，更应该维护一套和客户端相匹配的数据结构。游戏编程更像是面向数据的编程，一个好的数据结构更方便组织客户端代码，更方便管理数据，更方便对外提供接口。同样还应该对服务器消息结构友好，便于把服务器回来的消息同步到本地数据结构，然后再去驱动客户端逻辑。

115:熟悉图片上的各种格式，代表的含义

116:ConvertString如果传入一个null会返回一个空字符串吗？

117：通过StartCoroutine起一个协程，类似于调用函数的执行顺序，先执行进入协程函数，从协程函数里出来后才会执行StartCoroutine后面的代码。

118：频繁访问WWW类的bytesDownloaded字段会导致unity卡死，因为www会在后台自动下载，那哪怕断点调试。

119：整理下cd管理器，并扩展成计时器，用框架和注册的方式去实现

120：由AMonoBehaviour起的协程只能由该类的StopCoroutine停止，由其他MonoBehaviour停止的话会报错

121：最近在测试登录流程，正常网速情况下很少出问题，但在低速网速情况下问题百出，各种漏洞，各种流程不正常。根源是代码流程不严谨，健壮的代码流程是清晰明确的，不管什么网络情况下，都是一致性的执行流程。假如是在弱网环境下可能因为网络问题卡在某一步，要写代码重试尽量恢复。

122：在网络一般的情况下，各种地方都有下载争网速，导致某些下载失败。下载的分支尽量少，不要同时下好多东西，多个下载连接要有一个优先级的概念，保证一次少下几个其余的在等待也不能因为抢网速导致失败。

123：foreach尽量少使用的原因是什么？

124：一键强化一键升级等界面操作，服务器会发下很多条同样的消息，单次强化/升级和一键强化/升级，应该是两条消息才行。玩家一次升多级后，升级消息也会一次一次的发回来。

125：如果一个UISprite没有切九宫格，但使用了sliced模式，会怎样？

126：当GameObject的active等于false时，其Start方法，Update方法等都不会执行，当然协程更是会全部停掉。当脚本的enable等于false时,Update的方法也不会执行，但协程并不受其影响

127：学习扩展类extension的各种扩展方法和用途

128：当一个接口返回一个带颜色的字符串时，颜色的格式应该都带[-]结尾，要不然会影响到使用该接口的代码

129：buff系统，应该是作为数值计算系统的一层或多层过滤器，而对客户端而言，显示出来的buff图标应该自带一个数据块，用于标识该buff的数值描述，不应该是读取客户端数据，除非是一些经常变化的。

130：每一个数组或者List在使用索引获取值时都需要做一个是否越界的判断，能否有一种方法：当传入的索引值不合法时，也不至于出现null访问，而是返回一个null。再或者是抛出一个异常或日志

131：开发阶段经常会出现一些bug，这些bug里一部分是常见的数据错误，一些是不符合规范（规范包括，基础代码需要调用者提供一定的保证，但调用者并不懂需要提供什么样的保证），这些bug在查起来虽然不是那么的难查，但也是需要时间的，比如我曾经给一个同事查过同一个问题三次，但这三次导致问题的原因都是不一样的，尤其是第三次花了近一个小时的时间，难倒是不难，只不过需要层层排查。可不可以提供这么一种机制，一些常见错误给出弹窗提示，非常明确的告知问题出在哪了，在发版本时去掉这个弹窗即可，这个机制算是日志系统的一部分

132：个人BOSS挑战次数和挑战券使用次数功能，当初做的时候维护了一个剩余次数结构，挑战次数和使用挑战券次数改变时，对应的计算剩余次数，但后续接入了VIP功能后，挑战总次数和挑战券使用总次数会根据VIP动态的变化，这时一旦VIP变化，则剩余次数这个结构就很难正确的维护，比如VIP等级提高次数增加，则剩余次数该加多少就不好算了（计算VIP功能变化引起的次数变更，然后对应的计算到剩余次数里，但这样会更麻烦）。好的做法是：挑战次数和挑战券使用次数单独维护起来，登录数据和发生变化时（使用一次或者跨天重置）做修改。再提供一个统一的计算总次数的接口，这个接口除了计算静态的次数，还会接入VIP以及以后的其他影响次数的功能，VIP以及其他系统动态的变化时额外添加的次数是很容易获取的，这样总次数就是一个很好算的值。这样在计算生育次数时，拿总次数和已经使用次数作差值就行。

133：现有的事件系统做成了模板，只支持一个类型，可以改成非模板的，并且固定一个参数，这个参数是代码里的通用参数集合数据类，定义5个int，5个float，5个string等，用的时候赋值就好。

134：当隐藏一个面板，如果面板上的某个脚本在执行协程就会出问题，一种推荐的方式是使用全局协程，不再单独在某个Mono脚本里使用。另一种方式是把隐藏改为设置其alpha值为0.001f。

135：主界面按钮的红点事件，目前是一个按钮有一个，这样在绑定按钮红点事件和按钮红点数量时都是类似的代码一个按钮一份。好的设计：主界面按钮红点使用一个事件，用不同的参数区别按钮，如果是带数量的红点可以再用额外的参数去区分，当然还可以把带数量的红点单独独立出来一个事件类型，这样的做法会更好。根据按钮表去控制一个按钮到底是纯粹红点还是带数量的红点。

136：代码里经常会去从List里取值，对于给定的索引不去判断是否越界是很危险的事情，为了避免出现null访问的错误，需要在每一次访问数组或者List时判断是否越界，这实在不够友好，能不能实现一个定制的数组和List，这样在接收一个索引时，越界判断放到了定制的类里，如果越界则返回Null，这样也不至于出现Null访问，更好的做法是当越界了直接抛出异常。

137：定制一个table组件，能够实现各个方向上的对齐，并且能够避免widget全算

138：自动挂机的UI显示和程序里维护的是否自动挂机状态字段，是由两个方法控制的，这就是一个非常不好的设计，UI应当紧密的和数据关联在一起，并且他们应该是原子的（数据变动了，UI一定要跟着联动，他们两个都变化才是一个完整的过程，一个变动了一个没变动这就不是原子的）

139：UIStarSlider这个类是神X系统的通用类，但这个类设计的并不好，它最初的本意是统一的进度条行走速度以及亮星特效。但最后却臃肿成了一整套功能的集合。在转盘和山海印系统里，我只是想用它来管理进度条进度，但却不得不设置好多参数才能调用这个方法，并且里面有很多代码是我并不需要的。最好的设计：一整套功能应该分成独立的若干个模块，有一个总控类来管理，这样各个子功能模块都可以得到最小粒度的复用。

140：游戏的主循环逻辑以及一些核心代码，应该独立出来，不受其他代码的影响。要做到：某一个子模块update里出了再大的问题，都不能影响到其他的update的，实现方法可以验证一下用try catch是否可行，另一个问题：一旦用了try catch问题会在catch里被捕捉而不是暴露出来，一方面要求代码健壮和log齐全，一方面要在catch里处理（消化）掉这个异常。主循环里的好多update，如果每个里面都加try catch则会很麻烦，不统一，可以定义一个接口，所有需要在主循环里做更新的模块都要实现这个接口，维护一个接口集合，并且try catch代码只写一份就行。