屏幕适配 canvas scale screen match mode

锚点固定

接口使一组不相关的类具有相同的行为。

单例模式 内存只有一个实例，减少存开支，扩展难 只要play manager 可能引发内存泄露 堆内存无法释放，导致进程减慢甚至崩溃

.SDK导出aar包，作为插件导入Unity，然后Unity生成APK，这个方式比较好，也是我刚开始采用的，基于此，还写了个一键打包方法。

协程是什么 和线程有什么区别

dc是什么每次引擎准备数据并通知gup的过程叫一次dc

数组和arraylist

优点：数组在内存中是连续存储的、所以它的索引速度是非常快的、时间复杂度为O(1)、而且它的赋值/修改/获取元素也是非常简单的。

缺点：1、定义数组的时候需要指定数组的长度（过长会造成内存浪费、过短会导致程序异常System.IndexOutOfRangeException:"索引超出数组界限"）

2、插入和删除元素效率低、也比较麻烦。

在不清楚数组长度的时候、就很尴尬了。 所以C#提供了ArrayList了来处理这些问题...

二：ArrayList

使用大小会根据需要动态增加的数组。

优点：由于泛型List是强类型、编译器会验证类型安全。这样就避免了类型的不安全、以及数据强制转换导致装箱拆箱损耗性能。

right x upy forword z

16、线程和协程一样共享堆，而不共享栈，线程与协程最本质的区别是，线程在宏观上是同时进行的，而协程同一时刻只能执行一个协程，只不过协程可以通过某些条件暂时挂起不继续执行，等带达到条件要求之后在继续执行，从某种意义上来讲，协程可以看成是伪线程，本质上来讲，它们是包含关系，一个线程中可以包含多个协程。

依存关系 resource。unloadassets

回调函数

- 回调函数的作用通俗点讲就是完成了某个动作之后可以立马进行另一个动作，但那个动作你不确定具体会干什么，你可以根据不同的需求来进行不同的改变，这就是回调函数的作用

1、静态成员需要被static修饰，非静态成员不需要加static。

全局静态区只初始化一次

第二条的意思就是使用静态的东西可以减少堆上的内存

因为静态的东西不在堆上，再全局静态区

2、在一个非静态类中既可以出现静态成员，也可以出现非静态成员；而在一个静态类中只能出现静态成员；

3、在一个非静态方法中，既可以访问静态成员也可以访问非静态成员；而在一个静态方法中，只允许访问静态成员。

4、调用方法的区别：

实例方法需要使用对象调用，对象名.方法名；

静态方法使用类调用，类名.方法名。

5、静态类不允许创建对象。

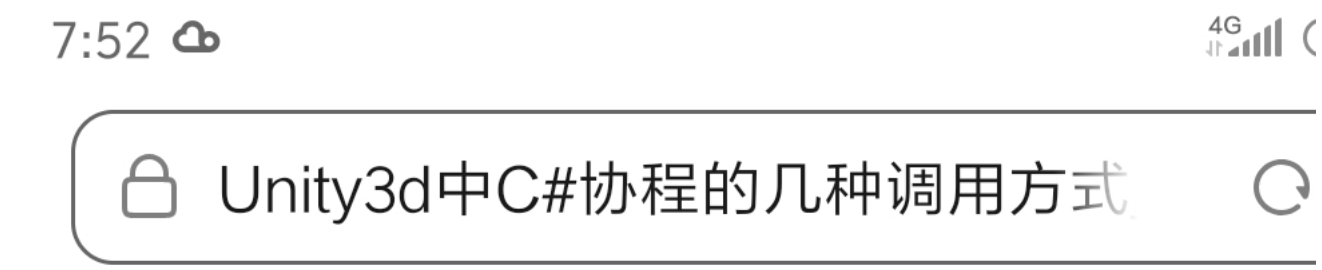
6、静态和非静态类的使用时机：

如果写的是工具类，可以考虑使用静态类，例如Console类；

静态的好处是资源共享；

静态类需要占用内存，所以静态类应越少越好。

静态方法和对象方法
静态方法static const在全局静态区只初始化一次 。 只分配一块内存
对象方法 new一次分配一次内存

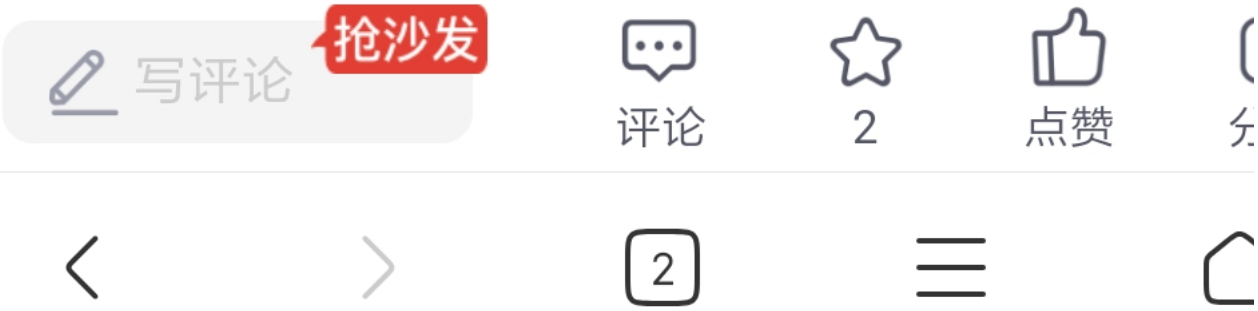



```
1. IEnumerator Start ()
2.         {
3.             return Test ();
4.         }
5.
6.     IEnumerator Test ()
7.     {
8.         Debug.Log ("开始等待：" + Time.time);
9.         yield return new WaitForSeconds (5);
10.        Debug.Log ("结束等待：" + Time.time);
11.    }
```

第三种方式

第二种方式

```
1. void Start ()
2.     {
3.         //StartCoroutine (Test ());
4.         StartCoroutine ("Test");
5.     }
6.     IEnumerator Test ()
```



协程使用IEnumerator修饰符，yield return返回

第一种方式：

```
1. .IEnumerator Start()
2.     {
3.         Debug.Log ("开始等待：" + Time.time);
```

```
4.        yield return new WaitForSeconds (5);
5.        Debug.Log ("结束等待：" + Time.time);
6.    }
```

第二种方式

```
1. IEnumerator Start ()
2.        {
3.            return Test ();
4.        }
5.
6.    IEnumerator Test ()
7.    {
8.        Debug.Log ("开始等待：" + Time.time);
9.        yield return new WaitForSeconds (5);
10.       Debug.Log ("结束等待：" + Time.time);
11.   }
```


评论
2
点赞