**1 . 什么时候会触发垃圾回收机制**

1. 在堆内存进行分配的操作,发现内存不够时候会触发垃圾回收机制回收利用限制的内存
2. GC会自动触发,,频率会和每个平台运行有关系
3. GC也可以强制执行

**2 . 垃圾回收时的操作**

1. **GC会检查堆内存上的每个储存变量**
2. **对每个变量会检查其引用是否处于激活状态**
3. **如果变量的引用不再处于激活状态,则会被标记为可回收**
4. **被标记的变量会被移除,其所占用的内存会被回收到堆内存上**

**GC操作是一个极其耗费的操作,堆内存上的变量或者引用越多则其运行的操作会更多,耗费时间越长**

**降低GC的影响的方法**

**有三种方法可以降低GC的影响**

1. **减少GC的运行次数**
2. **减少单次GC的运行时间**
3. **将GC的运行时间延迟,避免在关键时候触发,比如可以在场景加载的时候调用GC 似乎看起来简单,基于此,,我们可以采用三种策略:**
4. **对游戏进行重构,减少堆内存的分配和引用的分配. 更少的变量和引用会减少GC操作中的检测个数从而提高GC的运行效率**
5. **降低堆内存分配和回收的频率,尤其是在关键时刻,也就是说更少的时间触发GC操作,同时也降低堆内存的碎片化**
6. 我们可以试着测量GC和堆内存扩展的时间，使其按照可预测的顺序执行。当然这样操作的难度极大，但是这会大大降低GC的影响。

**减少内存垃圾的数量**

**缓存**

**不要在频繁调用的函数中反复进行堆内存分配**

**清除列表**

**使用对象池**

**造成不必要的堆内存分配的因素**

**字符串**

**Unity函数调用**

**装箱操作**

**协程**

**foreach 循环**

**函数引用**

**LINQ和常量表达式**

**定时执行GC操作**

**主动调用GC操作**

 　　如果我们知道堆内存在被分配后并没有被使用，我们希望可以主动地调用GC操作，或者在GC操作并不影响游戏体验的时候（例如场景切换的时候），我们可以主动的调用GC操作：

|  |  |
| --- | --- |
| 1 | System.GC.Collect() |

　　通过主动的调用，我们可以主动驱使GC操作来回收堆内存。

**https://www.cnblogs.com/zblade/p/6445578.html**