**了解托管堆**

在 Unity 中，托管堆的扩展比收缩容易得多。此外，Unity 的垃圾收集策略往往会使内存怎么样，因此可能阻止大型堆的收缩？

什么是由项目脚本运行时（Mono 或 IL2CPP）的内存管理器自动管理的一段内存？

必须在托管堆上分配托管代码中创建的所有对象，严格来说，必须在托管堆上分配所有非null引用类型对象和所有什么类型对象？

什么定期运行，这时将扫描堆上的所有对象，将任何不再引用的对象标记为删除。然后会删除未引用的对象，从而释放内存？

至关重要的是，Unity 的垃圾收集（使用 Boehm GC 算法）是什么类型的？

什么意味着 GC 在执行每遍收集时必须扫描整个堆，因此随着堆的扩展，其性能会下降？

什么意味着不会为内存中的对象重新分配内存地址来消除对象之间的间隙？

新释放的空间仅可用于存储什么？

分配对象时，请注意对象在内存空间中的分配地址必须始终为什么？

什么导致这个核心问题：虽然堆中的可用空间总量可能很大，但是可能其中的部分或全部的可分配空间对象之间存在小的“间隙”。这种情况下，即使可用空间总量高于要分配的空间量，托管堆可能也找不到足够大的连续内存块来满足该分配需求。

如果分配了大型对象，却没有足够的连续可用空间来容纳该对象，Unity 内存管理器将执行哪两个操作？

Unity 在扩展托管堆后不会怎么样；它会乐观地保留扩展后的堆，即使该堆的大部分为空时也如此？

在大多数平台上，Unity 最终会怎么样？发生此行为的间隔时间是不确定的，因此不要指望靠这种方法释放内存。

许多Unity项目在每帧都有几十或几百 KB 的临时数据分配给什么？这种情况通常对项目的性能极为不利？

在Unity的CPU 性能分析器中，Overview 表有一个什么列？此列显示了在特定帧中的托管堆上分配的字节数

启用什么选项后，可以跟踪执行这些分配的方法？

Unity Profiler 不会跟踪在什么之外发生的分配？

务必使用什么来分析托管分配？

某些脚本方法在Editor中运行时会产生分配内存，但在构建项目后不会产生分配内存。什么是最常见的示例；此方法始终在Editor中执行时分配内存，而不会在已构建的项目中分配内存？

强烈建议所有开发人员在项目处于什么状态时最大限度减少托管堆内存分配。非交互操作（例如场景加载）期间的内存分配很少产生问题？

适用于Visual Studio的什么插件可以帮助找到代码中的内存分配？

使用Unity的深度性能分析什么模式可找到托管分配的具体原因？

有哪些基本的内存节省方法？

集合类开放了一个Clear方法，该方法会消除集合内的值，但不会怎么样？

实例：list不要在update中创建

C# 中的所有方法引用都是引用类型，因此在什么上进行分配？

其次，将匿名方法转换为闭包后，为了将闭包传递给接收闭包的方法，所需的内存量会怎么样？

匿名方法现在要求该方法能够访问方法作用域之外的变量状态，因此已成为什么？

必须以某种方式将 desiredDivisor 变量传递给闭包，以便闭包的实际代码可以使用该变量。为此，C# 将生成一个什么，该类可保存闭包所需的外部作用域变量。

通常，请尽可能在 C# 中避免使用什么？应在性能敏感的代码中尽可能减少匿名方法和方法引用，尤其是那些每帧都需要执行的代码中。

因此，如果打算在 IL2CPP 脚本后端上发布项目，有哪三个主要建议？

装箱是Unity项目中最常见的非预期临时内存分配来源之一。只要怎么样就会发生装箱？

这种情况最常发生在将什么类型的变量（例如 int 和 float）传递给对象类型的方法时？

实例：装箱的例子

如何识别装箱？

装箱的一个常见原因是使用什么类型作为字典的键？声明 enum 会创建一个新值类型，此类型在后台视为整数，但在编译时实施类型安全规则。

对于枚举键字典，每次方法调用都会导致什么？

实例：解决枚举键字典装箱的方法

在 Unity 的 Mono C# 编译器版本中，使用 foreach 循环会在每次循环终止时强制什么？

Unity5.5中的C#编译器升级版本显著提高了Unity生成 IL 的能力。特别值得注意的是，已从 foreach 循环中消除装箱操作。因此，节约了与 foreach 循环相关的内存开销。但是，由于方法调用开销，与基于数组的等效代码相比，怎么样？

返回数组的所有Unity API每次被访问时都会怎么样？在不必要的情况下访问数组值 Unity API 是极不适宜的。

实例：mesh.vertices和Input.touches