所有类型都从System.Object派生。

System.Object的Public方法：

1.Equals：如果两个对象具有相同的值就返回true（对于Object的Equals方法的默认实现：它实现的实际是同一性，而非相等性）

2.GetHashCode：返回对象的哈希码（如果某个类型的对象要在哈希表集合中作为键使用，类型应重写该方法）。

3.ToString：默认返回类型的完整名称（this.GetType().FullName）。但经常重写该方法来返回包含对象状态表示的String对象（例如Boolean和Int32等核心类型重写ToString方法来返回它们的值的字符串表示）

4.GetType：返回从Type派生的一个类型的实例，指出调用GetType的那个对象的类型。返回的Type对象可以和反射类型配合。获取与对象的类型有关的元数据信息。

PS：需要注意的是： 我们在重写Equals时，GetHashCode也要一起重写；

因为有一个不重写就会调用基类的方法，依然会根据地址判断；

eg: Equals(Enumerable,Enumerable),重写Equals,遍历内容判断是否相同

System.Object的Protect方法：

1.MemberwiseClone：这个非虚方法创建类型的新实例，并将新对象的实例字段设置与this对象的实例字段完全一致。返回对新实例的引用。（浅拷贝）

2.Finalize：在垃圾回收器判断对象应该作为垃圾被回收后，在对象的内存被实际回收之前，会调用这个虚方法。（需要在回收内存前执行清理工作的类型应重写这个方法）

什么是浅拷贝和深拷贝？

浅拷贝：仅仅把对象的引用进行拷贝，但是拷贝对象和源对象是引用同一份实体。此时，其中的一个的成员对象的改变都会影响到另外一个成员的对象

深拷贝：指的是拷贝一个对象时，不仅仅把对象的引用进行拷贝，还把该对象引用的值也一起拷贝。这样进行拷贝后的副本对象就和源对象互相独立，其中任何一个的成员对象改动都不会对两外一个成员对象造成影响

C#浅拷贝和深拷贝的实现：

C#实现浅拷贝：调用MemberwiseClone方法，创建一个新的对象，然后复制当前对象的非静态字段的新对象创建一个浅副本。

C# 深拷贝的有三种实现：1.反射2.序列化3.表达式树[(101条消息) c# 深拷贝浅拷贝\_海的那边-的博客-CSDN博客](https://blog.csdn.net/m0_58717895/article/details/123057561)

用new操作符创建一个对象时会发生什么：

1.计算类型及其所有基类中定义的所有实例字段需要的字节数。（堆上每个对象都需要一些额外的成员——开销成员，包括“类型对象指针”和“同步块索引”，CLR利用这些成员管理对象，额外成员的字节数要计入对象大小）

2.从托管堆（项目脚本运行时的内存管理器自动管理的一段内存。）中分配类型要求的字节数，从而分配对象的内存，分配的所有字节都设为0

3.初始化对象的“类型对象指针”和“同步块索引”成员

4.调用类型的实例构造器，传递在new调用中指定的实参。（并且会自动调用基类的构造器直到最终调用System.Object的构造器）

5.返回指向新建对象的引用（指针）给栈上。

CLR具有“类型安全”的特性，即CLR永远知道对象的类型是什么（即“Object o=new D（）”这行代码通过GetType方法返回的结果”D”而不是“Object”）。

类型向基类型转换隐式转换，从基类型向派生类转换时必须显式转换。在运行时，CLR检查转型操作，确定总是转换为对象的实际类型或任何它的基类型。

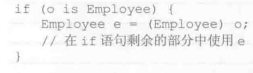
文本, 信件

描述已自动生成

这一段代码在编译时并不会报错，在运行时会报错。比较好的修改方式是吧PromoteEmployee的参数类型改为Employee而非Object，那么在编译时就会报错。

使用is和as操作符进行转型：

is操作符:is检查对象是否兼容于指定类型，然后进行转型，返回true或者false



上面这段代码使用is操作符时，CLR检查两次对象类型，第一次判断o是否兼容于Employee类型，在转型的时候进行第二次判断，核实o是否引用了一个Employee对象。

as操作符:as检查对象是否兼容与指定类型，返回同一个对象的非null引用或者null

文本

描述已自动生成

上面这段代码使用as操作符时，CLR只进行一次检查，只核实o是否兼容Employee类型

强制转换和as操作符速度比较，强制转换的速度慢于as操作符

关于程序集和命名空间，接第一章笔记：

C#编译器会识别命名空间，而CLR不会识别命名空间，因为CLR在访问一个类型时必须知道该类型的完整名称、以及该类型具体定义在哪个程序集中，因为需要通过元数据表加载对应的程序集然后才能使用对应的类型。命名空间和程序集不一定相关，同一个命名空间中的了些可能在不同程序集中实现，同一个程序集也可能包含不同命名空间中的类型。

对于命名空间以及类型名字都相同的不同类型，可以使用“外部类型“来解决这类问题。

**运行时的相互关系（重点！）**

仅存储在栈上：

Ps：字符串是引用类型，但能存储在栈上的原因。C#编译器和CLR（公共语言运行时）在优化方面做了一些工作。对于一些短小的字符串对象，编译器和CLR可能会选择将其存储在栈上，而不是堆上。这是因为栈上的内存分配和释放更快，而且栈上的对象不需要垃圾回收。这种优化称为"字符串驻留"（string interning），它可以提高性能并减少内存使用。需要注意的是，这种优化只适用于一些短小的字符串对象，并且是由编译器和CLR自动处理的。对于较长的字符串或者动态创建的字符串对象，它们仍然会被存储在堆上。因此，虽然字符串类型是引用类型，但具体的存储位置可能会有所不同，取决于编译器和CLR的优化策略。

1、

图示

描述已自动生成

2、

在运行函数的第一条语句前进行初始化（执行“序幕”代码），在线程栈上分配函数中变量“name”的内存

图示

描述已自动生成

3、

name变量作为函数M2的参数，因此name的地址被压入栈中，然后调用方法，在调用方法时一并把“返回地址”压入栈中

图示

描述已自动生成

4、

然后执行M2方法，执行序幕方法把length和tally先后分配内存压入栈中

图示

描述已自动生成

执行完M2方法后把相应的变量全部出栈，栈中只剩name变量的内存；执行完M1方法后name出栈，同时把name下方的途中没画出来的返回地址出栈。

存储在堆和栈上：

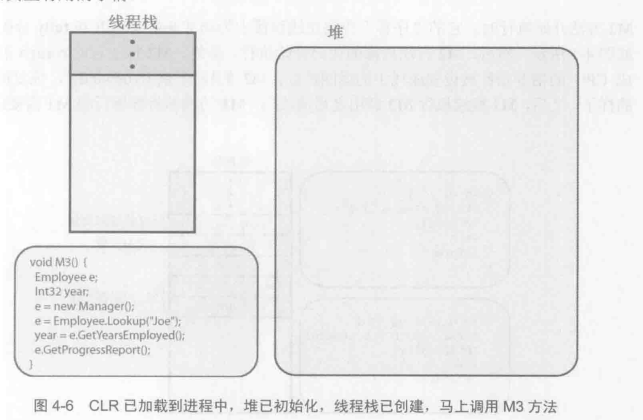
什么是类型对象指针和同步块索引？

类型对象指针：在.NET平台中，每个类型都有一个对应的类型对象（Type Object），它包含了关于该类型的元数据信息，例如类型的方法、字段、属性等。类型对象指针是指向类型对象的指针，它指向类型对象在内存中的位置。通过类型对象指针，运行时系统可以访问和操作类型对象的元数据信息。类型对象指针在.NET中被用于实现反射、类型检查和动态代码生成等功能。

同步块索引：在多线程环境下，为了确保对象的线程安全性，.NET平台使用同步块（Sync Block）来实现对象的互斥访问。每个对象都有一个同步块，用于控制对该对象的并发访问。同步块索引是一个整数值，它表示对象的同步块在同步块表中的索引位置。同步块表是一个全局的数据结构，用于存储所有对象的同步块信息。通过同步块索引，运行时系统可以快速定位和操作对象的同步块，实现线程同步和互斥访问。

类型对象：类型对象（Type Object）是在.NET平台中用于表示和描述一个类型的对象。每个在.NET中定义的类型都有一个对应的类型对象。

1、



2、

在堆上为类型对象分配空间，创建一些数据结构表示类型本身

图示

描述已自动生成

3、

在栈上执行序幕代码为局部变量和参数分配内存空间并进行初始化

图示

描述已自动生成

4、

在堆上创建Manager类型的实例，这个对象包含了类型对象指针和同步块索引以及Manager实例对象自己的所有示例数据字段并且容纳Manager的所有基类定义的实例字段。这个实例的类型对象指针指向实例对象对应的类型对象，

然后new操作符返回该实例的内存地址存储在栈上的e变量中。

图示

描述已自动生成

5、

调用静态方法时，CLR首先定位定义该静态方法的类型对象（如没有会加载对应的程序集并创建对应的类型对象），然后JIT编译器在类型对象的方法表中查找被调用方法对应的记录项，如果是第一次执行该方法则进行JIT编译，然后调用编译好的代码。（）

图示

描述已自动生成

6、

调用非虚实例方法时，JIT编译器会找到该实例(e)对应的类型的类型对象，如果该类型没有定义正在调用的方法，JIT编译器会回溯类的层次结构（直到Object，能回溯的原因是每个类型对象都有一个字段引用了它的基类型）并在沿途的每个类中查找该方法，然后对该方法JIT编译，调用JIT编译后的代码

图示

描述已自动生成

7、

调用虚方法时，JIT编译器要在方法中生成一些额外的代码，方法每次调用都会执行这些代码。这些代码首先检查发出调用的变量，并跟随地址来到发出调用的对象，然后代码检查被调用对象（实例对象）内部的类型对象指针成员，该成员指向对象的实际类型，然后代码在类型对象的方法表中查找引用了被调用方法的记录项对象进行JIT编译，最后调用JIT编译好的代码。

图示

描述已自动生成

Ps：在步骤2中对类型对象初始化即把类型对象的类型对象指针指向System.Type。 GetType方法是返回储存在指定实例化对象的类型对象指针成员中的地址，那么返回的是指向实例对象的类型对象的指针，因此可以判断任何实例对象的真实类型。