C _CLI解析之基于堆栈的对象与跟踪引用 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/264/2021_2022_C___CLI_E8_A7_A3_c97_264480.htm 在托管堆上分配对象实例，似乎是使用托管扩展C 、C#、J#、VB.NET程序员的唯一方法，而使用本地C 的程序员，不但可以在堆上分配内存，甚至更惯于使用基于堆栈的对象实例。 现在回顾一下以前定义的Point引用类，再来看一下以下变量定义： Point p1, p2(3,4). 从本地C 的角度来说，p1与p2应为基于堆栈的引用类Point实例，哪怕是从一般性的角度来看，它们也是。P1由默认的构造函数初始化，而p2由接受x与y坐标的构造函数初始化。从实现上来看，Point是自包含类型的（也就是说，它不包含任何指针或句柄），然而，作为一个引用类的实例，它仍处于CLI运行时的掌控之下，且在必要时，会被垃圾回收--正因为此，所以不能定义一个引用类的静态或全局实例。 同时，也不能将sizeof应用于指明是引用类实例的表达式，因为sizeof是在编译时进行计算的，而Point对象的大小要直到运行时才能确定；但是，可将sizeof应用于句柄，因为它的大小在编译时就已经确定了。 另外，还不能定义一个基于堆栈的CLI数组实例。 跟踪引用 本地C 可通过amp. n2 = n1. 引用必须在定义时进行初始化，且在整个生命期中，它们都锁定于引用同一对象，也就是说，它的值不会改变。引用一个引用类的实例与引用一个本地类基本一致，只不过语法不同而已。 在程序执行期间，引用类的实例会在内存中"移Point% p3 = p2. 跟踪引用的内存存储方式必须为自动（atuomatic），另外，尽管本地对象不会在内存中"移Point^ hp = gcnew Point(2,5).Point% p4 =

*hp.Point% p5 = *gcnew Point(2,5). 在此，hp是一个Point的句柄，而p4是此句柄的别名。虽然句柄不是一个指针，但也能使用一元 * 操作符来对句柄解引用。（在C /CLI标准制定期间，是否就引入一元 ^ 操作符来取代 * 还进行过一场讨论，反方观点是，在编写模板时，* 对句柄或指针进行解引用有非常高的价值。）当然，即使hp有了一个新值，p4在此仍是同一Point的别名。另外要说明一点，当对象有一个句柄或跟踪引用时，就不能被垃圾回收器回收了。 再来看p5，对gcnew返回的句柄进行了解引用，虽然差不多每个引用类类型的句柄，都能被解引用，但有两种类型的句柄却不能被解引用，这两种类型是：System::String与array。取句柄操作符 如果想把p1的值写到标准输出，代码似乎应该像下面这样：Console::WriteLine("p1 is {0}", p1). 然而，这却不能通过编译，因为WriteLine没有一个可接受Point的重载版本。前面也提过，任何值类型的表达式（如int、long、double）会由一个"装箱"的过程，自动转换为Object^。虽然p1看上去比较像一个值类型的实例，但它实际上却不是，它是一个引用类的实例，所以代码需要这样修改： Console::WriteLine("p1 is {0}", %p1). 通过使用一元 % 操作符，我们创建了对象p1的一个句柄，因为每个引用类最终都是从System::Object继承的，而WriteLine也有一个其第二个参数可接受Object^的重载版本，所以，%p1的Point^就转换为Object^，并显示出p1相应的值。要留意的是，此处没有装箱，但这个操作符不能应用到本地类的实例上。 GC-Lvalues 在C 标准中定义及使用了lvalue术语，而C /CLI标准则添加了gc-lvalue术语，其指"一个引用CLI堆中对象、或包含此对象的数值成员的表达式"。如果

有一个指向gc-lvalue的句柄，可对其使用一元 * 操作符来产生一个gc-lvalue；而跟踪引用也是一个gc-lvalue，当%h中h是一个句柄时，它也可以产生一个gc-lvalue。（因为有从lvalue至gc-lvalue的标准转换，所以一个跟踪引用可绑定至任意的gc-lvalue或lvalue。）