轻松使用C 深入研究.NET委托与事件设计 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/274/2021_2022__E8_BD_BB_E6_9D_BE_E4_BD_BF_E7_c67_274979.htm 简介 类型安全机制的实现原来采用的是C风格的回调(callback)函数，而.NET Framework引入了委托和事件来替代原来的方式；它们被广泛地使用。我们在这里尝试使用标准C 来实现与之类似的功能，这样我们不但可以对这些概念有一个更好的认识，而且同时还能够体验C 的一些有趣的技术。 C#中的委托与事件关键字 首先我们来看一个简单的C#程序(下面的代码略有删节)。执行程序的输出结果如下显示： SimpleDelegateFunction called from Ob1, string=Event fired! Event fired!(Ob1): 3:49:46 PM on Friday, May 10, 2002 Event fired!(Ob1): 1056318417 SimpleDelegateFunction called from Ob2, string=Event fired! Event fired!(Ob2): 3:49:46 PM on Friday, May 10, 2002 Event fired!(Ob2): 1056318417 所有这些都源于这样一行代码：dae.FirePrintString("Event fired!"). 在利用C 来实现这些功能时，我模仿了C#的语法并完全按照功能的要求进行开发。 namespace DelegatesAndEvents { class DelegatesAndEvents { public delegate void PrintString(string s). public event PrintString MyPrintString. public void FirePrintString(string s) { if (MyPrintString != null)MyPrintString(s). } } class TestDelegatesAndEvents { [STAThread] static void Main(string[] args) { DelegatesAndEvents dae =new DelegatesAndEvents(). MyDelegates d = new MyDelegates(). d.Name = "Ob1". dae.MyPrintString =new

DelegatesAndEvents.PrintString(d.SimpleDelegateFunction). // ... more code similar to the // above few lines ... dae.FirePrintString("Event fired!"). } } class MyDelegates { // ... "Name" property omitted... public void SimpleDelegateFunction(string s) { Console.WriteLine("SimpleDelegateFunction called from {0}, string={1}", m_name, s). } // ... more methods ... } } C 中的类型安全函数指针 对于“ 老式方法 ”的批判之一便是它们不是类型安全的[1]。下面的代码证明了这个观点： typedef size_t (*FUNC)(const char*). void printSize(const char* str) { FUNC f = strlen. (void) printf("%s is %ld chars\n", str, f(str)). } void crashAndBurn(const char* str) { FUNC f = reinterpret_cast ＜FUNC＞(strcat). f(str). } 代码在[2]中可以找到。当然，在你使用reinterpret_cast的时候，你可能会遇到麻烦。如果你将强制转换(cast)去掉，C 编译器将报错，而相对来说更为安全的static_cast也不能够完成转换。这个例子也有点像比较苹果和橙子，因为在C#中万事万物皆对象，而reinterpret_cast就相当于一种解决方式。下面的这个C 程序示例将会采取使用成员函数指针的方法来避免使用reinterpret_cast： struct Object { }. struct Str : public Object { size_t Len(const char* str) { return strlen(str). } char* Cat(char* s1, const char* s2) { return strcat(s1, s2). } }. typedef size_t (Object::*FUNC)(const char*). void printSize(const char* s) { Str str. FUNC f = static_cast＜FUNC ＞(amp.Str::Cat). (str.*f)(s). } static_cast运算符将转化Str::Len函数指针，因为Str是由Object派生来的，但是Str::Cat是类型安全的，它不能被转换，因为函数签名是不匹配的。 成员函数

指针的工作机制与常规的函数指针是非常相似的；唯一不同(除了更为复杂的语法外)的是你需要一个用来调用成员函数的类的实例。当然，我们也可以使用->*运算符来用指向类实例的指针完成对成员函数的调用。 Str* pStr = new Str(). FUNC f = static_cast＜FUNC＞(&amp.Str::Len). (void) printf("%s is %ld chars\n", s, (str-＞*f)(s)). 0delete pStr. 只要所有的类是从基类Object派生来的(C#中就是这样)，你就可以使用C 来创建类型安全的成员函数指针。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com