**C++的类与C#的类的区别**

## 微软公司给C#(读为C-Sharp)赋予C++某些面向对象的本质，比如模板，但改变了类的创建方法。本文，我将对比C++和C#的类，并着重说明微软在C#中类创建和使用方面的改变。

## 一、简介

面向对象(OO)编程在应用设计中已经发展二十来年了。程序不再是一系列函数的堆彻(象一些范例那样的程序)，而是对象的集合，每个对象都有其独特的属性和方法来与其它对象打交道。  
  
"C"语言系列是面向对象设计发展的最好例子。C++为开发者提供了优秀的面向对象编程工具，程序员可以显式地创建构造函数，拷贝构造函数，重载操作符，使用模板等等。  
  
象C++这样复杂语言的主要问题是程序员要花上好几个月来掌握面向对象设计本质。新程序员必须学会掌握模板，函数重载，当然还要会创建和使用功能良好的类。  
  
微软公司给C#(读为C-Sharp)赋予C++某些面向对象的本质，比如模板，但改变了类的创建方法。本文，我将对比C++和C#的类，并着重说明微软在C#中类创建和使用方面的改变。  
  
本文对象是熟练的C++程序员，通过一些细节来解释C#面向对象的本质。

## 二、C#的类有了哪些改变？ 如你所知，C#是部分基于C++，部分基于Java语法的语言。C#中还有一些细节上的改变，使得它可以用于现代设计。 当你开始用C#建类时就会立即看到这点。让我们通过一个简单的例子开始了解在C++和C#中是如何建类并进行实例化的。 C++版本： #include <iostream> class MyClass { 　public: void doSomething() 　{ 　　std::cout << "This is some text"; 　} }; void main() { 　MyClass mc; 　mc.doSomething(); } C# 版本： using System; class MyClass {     public void doSomething()     {         Console.WriteLine("This is some text");     } } class EntryPoint {     public static void Main()     {         MyClass mc = new MyClass();         mc.doSomething();     } } 上面的代码中有几个不同之处。 首先，C++用#include包含语句来指明包含文件iostream.h的物理路径。C#则告诉编译器程序将在System命名空间下操作，所有的命名空间和类都属于System命名空间。C#通过命名空间的名字来决定程序的作用范围(本例中只有System一个命名空间)，而不用指明物理路径的包含文件方法。 其次，C#的主程序用Main(注意M是大写)。 第三，C++的类声明结束后要在最后的大括号后面用分号结尾。C#则可用可不用，往往都是省略。 第四，你能看到在C#中必须显式地声明方法和成员的作用域。若不加声明，缺省为私有(只有类成员可以访问)，这点与C++一样。C#中有5种作用域： 　公有(public)：其他类成员也可以访问 　私有(private)：只有类成员才能访问 　保护(protected)：类成员和继承类成员可以访问 　内部(internal)：只有汇编里的成员才能访问(C#的汇编是代码和资源数据的结合，以asmx作文件后缀) 　内部保护(protected internal)：类成员和继承类成员可以访问 最后，与Java一样，C#的方法也可以声明为静态(static)的。静态变量的使用在C#和C++是一样的。在C#里，可以这样创建并调用类的静态方法： using System; class MyClass { 　public static void doSomething() 　{ 　　Console.WriteLine("This is some text"); 　} }; class EntryPoint { 　public static void Main() 　{ 　　MyClass.doSomething(); 　} } 注意，这里直接使用类声明而没有创建类的实例。这是为C#语言增加的非常便利的使用方法，可以节省我们的时间和内存。就是说，不要创建类实例，可以直接调用类的方法。

## 三、用类修饰语限制对类的访问 以前只能对类成员和类方法设定限制，但不能对类实体作限制。C#可以通过声明类修饰语来对类的实例实行限制，如上节提到的作用域。 C++不能对整个类作限制。看一下C++的类声明： class Car { 　public: 　　Car(); 　　Car(Car &c); 　　virtual ~Car(); 　private: 　　int numCars; 　　Car\* previous; 　　Car\* next; }; 这里有二种访问类型：公有(public)和私有(private)。继承或将类Car实例化后，程序只能继承这些代码，不能作其它变动，如果要作其它变动就不能将其作为基类。 C#对此了改变。可以附加访问修饰语来限制类成员和方法以及类实例的访问权。C#设定8个访问权限： 　公有(public)：可以被所有其它的类访问。没有其它限制修饰语，它的公有性质就一直是缺省的。 　私有(private)：只有类成员才能访问。 　保护(protected)：类成员和继承类成员可以访问。 　内部(internal)：只有汇编里的成员才能访问(C#的汇编是代码和资源数据的结合，以asmx作文件后缀)。 　内部保护(protected internal)：类成员和继承类成员可以访问。 　密封(sealed)：所有继承类都不能访问。无论直接或间接地将它作为基类，C#编译器都会跳错。 　抽象(abstract)：与C++的虚(virtual)类或虚方法相似，抽象类不能直接实例化，抽象函数含有函数名。但在作为基类或继承类时可以使用。 　新建(new)：用new创建嵌套类，可以隐藏继承方式，告诉编译器创建一个类的新版本。 　 举例来说，如果要创建一个类，并且这个类不能被作为基类或继承，那么就要创建一个密封类： sealed class Car { 　public void paintCar() 　{ 　　// Code to paint car goes here 　} } 这时如果要创建一个继承类RedCar： internal class RedCar : Car { 　// Won't work. } C#编译器就会跳错： error CS0509: 'RedCar' : cannot inherit from sealed class 'Car'  (不能从密封类'Car'继承)。

## 四、C++和C#中的虚函数 C++和C#都支持虚函数。在下面的C++例子里，有二个类，分别称为Base(基类)和Derived(继承类)： #include <iostream> using namespace std; class Base { 　public: 　　void doWork() 　　{ 　　　cout << "Base class working"; 　　} 　protected: 　　virtual void doWork1() = 0; }; class Derived : public Base { 　public: 　　void doWork2() 　　{ 　　　cout << "Derived class working"; 　　} 　　void doWork1() 　　{ 　　　cout << "Dervied pure virtual function working"; 　　} }; void main() { 　Derived d; 　d.doWork1(); } 基类里有二个函数，一个是doWork，另一个是纯虚函数doWork1。doWork1只能被基类的继承类使用。在继承类(公有地继承于基类)里有一个新函数doWork2，和继承于基类纯虚函数的超越函数doWork1。 在C#里实现同样的功能要更容易些。看下面的C#代码： using System; abstract class Base { 　public void doWork() 　{ 　　Console.WriteLine("Base class working"); 　} 　public abstract void doWork1(); } class Derived : Base { 　public void doWork2() 　{ 　　Console.WriteLine("Derived class working"); 　} 　public override void doWork1() 　{ 　　Console.WriteLine("Dervied pure virtual function working"); 　} }; class EntryPoint { 　public static void Main() 　{ 　　Derived d = new Derived(); 　　d.doWork1(); 　} } C#将基类定义为抽象类，将doWork1定义为抽象方法。这样就可以获得与上述C++纯虚函数同样的功能。Base类只能作为基类或被含有doWork1超越函数的继承类使用。 继承类在超越抽象函数doWork1时，在函数前面加上超越前缀(override)。C#编译器在发现继承类里的override关键字后，就检查基类的同名函数。只要不是直接显式地调用基类函数，编译器总是使用继承类中的方法。 为了让继承类直接操作基类成员和方法，C# 为基类命名了一个别名base。用这个别名，继承类可以直接引用基类成员和方法。示例如下： using System; class first { 　public void writeIt() 　{ 　　Console.WriteLine("Writing from base class"); 　} } class second : first { 　public second() 　{ 　　base.writeIt(); 　} } class EntryPoint { 　public static void Main() 　{ 　　second s = new second(); 　} } 在上述例子中，有二个类。一个是基类(first)，另一个是继承类(second)。当创建second类的实例时，它的构造函数自动调用基类的writeIt方法，用控制台指令Console.WriteLine打印屏幕。由此引出C++和C#中的多态性。

## 五、C++和C#中的多态性 实体的多态性使其具有多种表现形式。在C++和C#中处理多态性是很相像的。看一个简单例子： C++ 版本： #include <iostream> #include <string> using namespace std; class Person { 　public: 　　Person() 　　{ 　　　classType = "Person"; 　　} 　friend void ShowType(Person& p); 　private: 　　string classType; }; class Manager : public Person { 　public: 　　Manager() 　　{ 　　　classType = "Manager"; 　　} 　friend void ShowType(Person& p); 　private: 　　string classType; }; void ShowType(Person& p) { 　cout << p.classType << endl; } void main() { 　Person p; 　Manager m; 　ShowType(p); 　ShowType(m); } C# 版本： using System; class Person { 　public Person() 　{ 　　classType = "Person"; 　} 　public string classType; } class Manager : Person { 　public Manager() 　{ 　　classType = "Manager"; 　} 　public new string classType; } class EntryPoint { 　public static void ShowType(ref Person p) 　{ 　　Console.WriteLine(p.classType); 　} 　public static void Main() 　{ 　　Person p = new Person(); 　　Person m = new Manager(); 　　ShowType(ref p); 　　ShowType(ref m); 　} } 在上面的例子里，有一个基类Person，一个继承于基类的Manager类。在EntryPoint类里，创建了一个静态函数ShowType，其表达为： public static void ShowType(ref Person p) 注意参数里的ref关键字。ref告诉C#编译器向ShowType函数传递一个参数的引用(reference)。在C#中，如果不用ref关键字，函数的参数传递缺省为值(value)传递，将拷贝一个参数值传递到函数中去。 在C++中的参数引用传递表达为： void ShowType(Person& p) C++用"&"符号表示参数引用使得程序员新手感到困惑，尤其是那些从VB转过来的人。 在C#的主函数(entry point)里，创建了二个新的Person对象，p和m： Person p = new Person(); Person m = new Manager(); 值得一提是，关键字new在C#和C++中用法是不一样的。在C#里，new只创建一个对象实例。这个对象依然创建在管理堆里，但不返回指向对象的内存地址的指针。在上面的C#例子中，创建了二个Person类对象。第二个对象，m，却是Manager类的对象。它使用Manager的构造函数而不是用Person的构造函数。 将Person类对象和Manager类对象引用到ShowType函数，记住，Manager是Person的继承类，但C#的多态性将其表达为一个Person类： ShowType(ref p); ShowType(ref m); 当作用到ShowType函数时，它只处理Person对象。C#告诉它说m是Person继承类的对象，它就将m按Person类处理了。所以用p和m作参数调用ShowType函数后得到的输出为： Person Person [译者注：这样解释多态性有点离谱。这段 C#代码的真正含义在于诠释静态函数的作用，而不是什么多态性。上面一段C++代码则可以看成多态性用法的解释。]

## 六.c++的类对象是在栈上面创建的,是值类型,c#的类对象在托管堆里面创建的,是引用类型.c++的new操作符创建的是一个指向类对象的指针,c#new操作符创建的是一个引用类型的类对象.

## 七、结论 如果你是一个C++程序员，现在想转向电子商务编程或干脆就是想换一种更现代的语言，那么就是C#了。有这么三种原因： 如果会使用C#，你就能创建任何应用：Windows应用，控制台应用，Web应用和Web服务等等。 所有的.NET平台使用的语言都编译成为中间语言(IL)，并能按照系统环境进行优化。 非常非常容易将C++转换成C#。

转载于:https://www.cnblogs.com/anshangcun/archive/2007/10/16/926330.html