理解C#多态性之前首先理解一下什么叫多态。同一操作作用于不同的对象，可以有不同的解释，产生不同的执行结果，这就是多态性。C#多态性通过派生类覆写基类中的虚函数型方法来实现。

C#多态性分为两种，一种是编译时的多态性，一种是运行时的多态性。

◆编译时的多态性：编译时的多态性是通过重载来实现的。对于非虚的成员来说，系统在编译时，根据传递的参数、返回的类型等信息决定实现何种操作。

◆运行时的多态性：运行时的多态性就是指直到系统运行时，才根据实际情况决定实现何种操作。C#中运行时的多态性是通过覆写虚成员实现。

**C++代码：**

 这就是众所周知的的多态。现代面向对象语言对这个概念的定义是一致的。其技术基础在于继承机制和虚函数。例如，我们可以定义一个抽象基类Vehicle和两个派生于Vehicle的具体类Car和Airplane：   
    // dynamic\_poly.h   
    #include   
    // 公共抽象基类Vehicle   
    class Vehicle   
    {   
    public:   
     virtual void run() const = 0;   
    };   
    // 派生于Vehicle的具体类Car   
    class Car: public Vehicle   
    {   
    public:   
     virtual void run() const   
     {   
     std::cout << "run a car\n";   
     }   
    };   
    // 派生于Vehicle的具体类Airplane   
    class Airplane: public Vehicle   
    {   
    public:   
     virtual void run() const   
     {   
     std::cout << "run a airplane\n";   
     }   
    };   
客户程序可以通过指向基类Vehicle的指针（或引用）来操纵具体对象。通过指向基类对象的指针（或引用）来调用一个虚函数，会导致对被指向的具体对象之相应成员的调用：

    // dynamic\_poly\_1.cpp   
    #include   
    #include   
    #include "dynamic\_poly.h"   
    // 通过指针run任何vehicle   
    void run\_vehicle(const Vehicle\* vehicle)   
    {   
     vehicle->run(); // 根据vehicle的具体类型调用对应的run()   
    }   
    int main()   
    {   
     Car car;   
     Airplane airplane;   
     run\_vehicle(&car); // 调用Car::run()   
     run\_vehicle(&airplane); // 调用Airplane::run()   
    }

此例中，关键的多态接口元素为虚函数run()。由于run\_vehicle()的参数为指向基类Vehicle的指针，因而无法在编译期决定使用哪一个版本的run()。在运行期，为了分派函数调用，虚函数被调用的那个对象的完整动态类型将被访问。这样一来，对一个Car对象调用run\_vehicle()，实际上将调用Car::run()，而对于Airplane对象而言将调用Airplane::run()。

或许动态多态最吸引人之处在于处理异质对象集合的能力：   
    // dynamic\_poly\_2.cpp   
    #include   
    #include   
    #include "dynamic\_poly.h"   
    // run异质vehicles集合   
    void run\_vehicles(const std::vector< Vehicle\* >& vehicles)   
    {   
     for (unsigned int i = 0; i < vehicles.size(); ++i)   
     {   
     vehicles[i]->run(); // 根据具体vehicle的类型调用对应的run()   
     }   
    }   
    int main()   
    {   
     Car car;   
     Airplane airplane;   
     std::vector< Vehicle\* > v; // 异质vehicles集合   
     v.push\_back(&car);   
     v.push\_back(&airplane);   
     run\_vehicles(v); // run不同类型的vehicles   
    }

在run\_vehicles()中，vehicles[i]->run()依据正被迭代的元素的类型而调用不同的成员函数。这从一个侧面体现了面向对象编程风格的优雅。

**C#代码**

using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Windows.Forms;

namespace WindowsFormsApplication1  
{  
    public class Vehicle  
    {  
        public virtual string Run()  
        {  
            return "aaa";  
        }  
    }  
    public class Car : Vehicle   
    {  
        public override string Run()  
        {  
            return "running a car!";  
        }  
      
    }  
    public class Airplane : Vehicle  
    {  
        public override string Run()  
        {  
            return "fly a Airplance!";  
        }

    }  
      
    public partial class Form1 : Form  
    {  
          
        public Form1()  
        {  
            InitializeComponent();  
        }  
    
        private void Form1\_Load(object sender, EventArgs e)  
        {  
            Vehicle v = new Vehicle();  
            Car c = new Car();  
            Airplane a = new Airplane();  
            Console.WriteLine(v.Run());  
            Console.WriteLine(c.Run());  
            Console.WriteLine(a.Run());  
        }  
         
    }  
}