第 5 章　继承

类的继承，是面向对象编程中一个非常重要的概念，其主要功能就是代码的复用。例如，对于汽车类型，已经创建了一个 CAuto 类，当需要更具体的汽车类型，如 CCar、CSuv 等类型，就可以从 CAuto 类继承而来。

本章就来讨论继承在 C# 中的应用，主要内容包括：

- ❑ 父类与子类；
- ❑ 成员的重写；
- ❑ 抽象类与抽象方法。

在讨论继承关系之前，应该先了解应用继承的前提条件，即，一个类是否允许被继承。

默认情况下，类是可以被继承的，但有些类是不能被继承的，如使用 sealed 关键字定义的类，如下代码中定义的 ClassS 类就不能被继承。

```
public sealed class ClassS
{
    // ...
}
```

接下来继续在 HelloConsole 项目中进行测试。

5.1　父类与子类

类的继承是一个相对的概念，包括继承者与被继承者。其中，被继承的类称为父类，又称为基类（base class）或超类（super class）。继承者称为子类。

在 CAuto 类中，已经定义了 DoorCount 属性、AddPerson() 方法等成员，接下来，再添加两个构造函数和 MaxSpeed 属性，如下面的代码（CAuto.cs 文件）。

```
public class CAuto
{
    // 其他代码
    // 构造函数
    public CAuto()
    {
        DoorCount = 4;
    }
    //
    public CAuto(int doorCount)
    {
        DoorCount = doorCount;
    }
    // 最高时速
    public uint MaxSpeed { get; set; }
}
```

请注意，在无参数的构造函数中，将车门数量（DoorCount 属性）设置为 4。

接下来，创建两个 CAuto 类的子类，分别是 CCar 和 CSuv，将这两个类定义在一个文件中，如下面的代码（CAutoSubClass.cs 文件）。

```
using System;

namespace HelloConsole
{
    //
    public class CCar : CAuto
    {
    }

    //
    public class CSuv : CAuto
    {

    }
}
```

可以看到，CCar 和 CSuv 类继承于 CAuto 类，使用冒号运算符“:”指定继承关系。此外，这两个类中没有定义任何成员，这种情况下，子类会自动继承父类中的非私有成员，包括无参数构造函数和其他类型的成员。

如下代码演示了 CCar 类的应用。

```
CCar car = new CCar();
car.MaxSpeed = 250;
Console.WriteLine("轿车的最高时速为{0}km/h", car.MaxSpeed);
Console.WriteLine("车门数量为{0}", car.DoorCount);
```

其中，MaxSpeed 属性是定义在 CAuto 类中的公共成员，在 CCar 类中可以直接使用。请注意第二条输出语句会显示车门数量 4，这是在构造函数中设置的数据。

CSuv 类也是这样，大家可以自己动手进行测试。

这里简单地演示了继承的概念，其中的 CAuto 就称为 CCar 和 CSuv 的父类，而 CCar 和 CSuv 就是 CAuto 类的子类。此外，在子类中可以直接使用父类中定义的非私有成员，而且，如果在子类中没有定义构造函数，子类会继承父类中的无参数构造函数。

接下来继续了解关于构造函数继承的相关内容。

5.1.1 构造函数的继承

从前面的例子中可以看到，如果在子类中没有创建构造函数，就会继承父类中的无参数构造函数。但是，如果在子类中定义了一个有参数的构造函数，无参数的构造函数就会“消失”，如果需要，就必须重新定义它。

如下代码将在 CCar 类中创建一个构造函数，它包括一个参数，用于指定汽车的引擎类型（CAutoSubClass.cs 文件）。

```
public class CCar : CAuto
{
    // 构造函数
```

```
    public CCar(string engine)
    {
        DoorCount = 4;
        EngineType = engine;
    }
}
```

接下来，试着使用无参数的构造函数来创建对象，如下面的代码。

```
CCar car = new CCar();
```

执行代码，会看到错误提示，如图 5-1 所示。

图 5-1　构造函数调用错误

如果需要一个无参数的构造函数，就应该创建它，如下面的代码。

```
public CCar()
{
    DoorCount = 4;
    EngineType = "汽油发动机";
}
```

在 CCar 类的这两个构造函数中，又看到了重复的代码，这时，就可以考虑新的方法来创建构造函数。

首先，修改有一个参数的构造函数，如下面的代码。

```
public CCar(string engine)
    : base()
{
    EngineType = engine;
}
```

在这里又使用了继承，请注意 base()，它实际上是调用了父类中的无参数构造函数。在 CCar 类的这个构造函数中，首先调用父类中的构造函数，此时会将车门数量设置为 4，然后再执行自己的代码。

接下来修改无参数的构造函数，如下面的代码（CCar.cs 文件）。

```
public CCar() : this("汽油发动机") { }
```

这个构造函数足够简单了，它继承了本类中的 CCar(string engine) 构造函数。当调用 CCar 类的无参数构造函数时，会将车门数量设置为 4，引擎类型设置为汽油发动机。

如下代码演示了 CCar 类的构造函数的使用。

```
CCar car = new CCar();
Console.WriteLine("此车有{0}个车门", car.DoorCount);
Console.WriteLine("此车引擎为{0}", car.EngineType);
```

代码执行结果如图 5-2 所示。

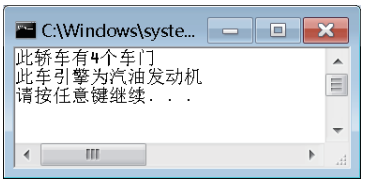

图 5-2　构造函数的继承

在上一章中，通过一系列构造函数创建了构造函数链，从而简化了构造函数的创建过程。在继承体系中，可以同时通过继承父类或本类中的构造函数，进一步简化构造函数的创建。

5.1.2　唯一没有父类的类（Object）

实际应用中，随便在哪个类中都可以使用 ToString() 方法，它的功能就是返回对象的文本描述（string 类型），只不过从来就没有定义过这个方法，那么，它是从哪里来的呢？

隐约中，你可能会感觉这是某个终极类型在起作用，是的，这就是 Object 类，它定义在 System 命名空间。

Object 类是在 .NET Framework 环境下唯一一个没有父类的类，在定义的类型中，如果没有指定其父类，它的父类就是 Object。另一方面，所创建的类，无论继承了多少层，其终极父类都是 Object，这也是为什么可以在所有对象中使用 ToString() 方法。

除了 ToString() 方法，在 Object 类中还有一些方法比较实用，如：

- GetType() 方法会返回对象（或变量）的类型（Type 类型对象），第 9 章会讨论如何使用 Type 类获取更多的类型信息。
- ReferenceEquals() 方法，判断参数中的对象是否与调用此方法的对象为同一个引用。
- Equals() 方法判断参数与调用者是否相等，如果参数为值类型（如枚举或结构）则比较内容是否相等。如果为引用类型（如类），则与 ReferenceEquals() 方法比较结果相同。

5.2　成员的重写

在子类中，如果需要，还可以重写（override）父类中的方法、属性、索引器或事件。下面讨论相关内容。

5.2.1　虚拟成员

在类中定义的方法、属性、索引器或事件，如果需要明确在子类中可以重写，就应该在定义时使用 virtual 关键字定义为虚拟成员。

虚拟成员也可以做具体的工作，即使不重写，也能够正常使用，这一点和普通成员是一样的。

如下代码（CAuto.cs 文件）是在 CAuto 类中定义一个虚拟方法 Fire()。

```
// 虚拟方法
public virtual void Fire()
{
    Console.WriteLine("汽车到底怎么开火呢？？？ ");
}
```

5.2.2　重写

子类中，如果需要重写虚拟成员，需要使用 override 关键字，如下代码（CWarriorJeep.cs 文件）定义的 CWarriorJeep 类继承了 CAuto 类，并添加了 Weapon 属性。请注意，这里重写了其中的 Fire() 方法。

```
public class CWarriorJeep : CAuto
{
    public string Weapon { get; set; }
    //
    public override void Fire()
    {
        base.Fire();
        if (Weapon != "")
        {
            Console.WriteLine("安装了武器{0}，开火试试！ ", Weapon);
        }
    }
}
```

请注意代码中的 base.Fire() 语句，通过 base 关键字调用了父类（CAuto 类）中的 Fire() 方法。这里，即使在子类里重写了父类成员，依然可以使用 base 关键字访问。

如下代码测试了 CWarriorJeep 类的使用。

```
CWarriorJeep jeep = new CWarriorJeep();
jeep.Weapon = "12.7mm 重机枪 ";
jeep.Fire();
```

代码执行结果如图 5-3 所示。

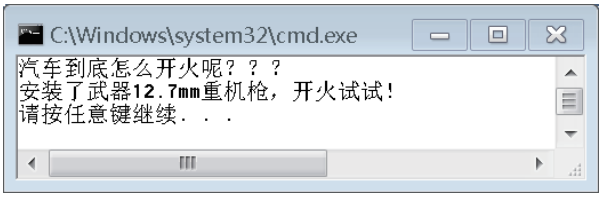

图 5-3　重写方法

5.2.3　隐藏父类成员

如果在父类中的成员没有定义为可重写的（不是虚拟、抽象或重写成员），但又需要在子类中改变这些成员的实现时，可以使用完全覆盖的方式。此时，子类中重新定义成员，并使用 new 关键字，其功能是隐藏父类中的同名成员。

如下代码在 CWarriorJeep 类中重新定义了 MaxSpeed 属性，并指定为只读属性（CWarriorJeep.cs 文件）。

```
new public uint MaxSpeed
{
    get { return 100; }
}
```

请注意，new 关键字和 override 关键字的含义不同，一般来讲，重写（override）的成员是对父类成员的功能实现或扩展。而 new 关键字则表明成员在子类中的全新实现。

5.3 抽象类与抽象方法

抽象（abstract）类是另一种不能被实例化的类，它的功能就是为其子类提供一些标准化结构或通用代码，然后由其子类具体实现。

如果一个类中定义了一个抽象成员（如方法、属性、索引器或事件），这个类就必须被定义为抽象类。

将一个类定义为抽象类时，需要使用 abstract 关键字，如下代码（CPlaneBase.cs 文件）定义了一个飞机类，但由于飞机有很多种，所以，这里只提供了飞机类的基本结构，而不代表某一种具体的飞机类型。

```
// 飞机基类
public abstract class CPlaneBase
{
    // 构造函数
    public CPlaneBase(string sEngine, uint iPilotCount,uint iMaxSpeed)
    {
        EngineType = sEngine;
        PilotCount = iPilotCount;
        MaxSpeed = iMaxSpeed;
    }
    //
    public string EngineType { get; set; }
    public uint PilotCount { get; set; }
    public uint MaxSpeed { get; set; }
    //
    public abstract void Fire();
    //
}
```

在 CPlaneBase 类中定义了一个构造函数、三个属性，以及一个抽象方法 Fire()。不是所有飞机都有武器，所以，必须在具体的飞机类型中实现 Fire() 方法。而且，也正是因为 Fire() 方法是一个抽象成员，所以，CPlaneBase 类也必须定义为抽象类。

在这里，抽象类的名称最后使用 Base 字样只是一个习惯，这表示它是作为基类来使用的。

接下来，创建一个客机类，请注意其中的构造函数，它继承了父类中的构造函数。此外，重写 Fire() 方法时使用了 override 关键字，如下面的代码（CAirliner.cs 文件）。

```
public class CAirliner : CPlaneBase
{
    // 构造函数
    public CAirliner()
        :base("喷气式发动机",2,1000)
    { }
    //
    public override void Fire()
    {
```

```
            Console.WriteLine("亲! 客机是没有武器的! ");
        }
    }
```

然后，可以使用如下代码测试客机类。

```
CAirliner plane = new CAirliner();
plane.Fire();
```

接下来，还可以定义更多的飞机类，如战斗机（CFighter.cs 文件）。

```
using System;

namespace HelloConsole
{
    public class CFighter : CPlaneBase
    {
        // 构造函数
        public CFighter() : base("喷气式发动机", 1, 3000) { }
        //
        public override void Fire()
        {
            Console.WriteLine("使用机炮和导弹开火");
        }
    }
}
```

可以使用以下代码测试战斗机类的使用。

```
CFighter j20 = new CFighter();
j20.Fire();
```

继承提供了代码复用的一种有效方式，可以在层次分明的代码结构中发挥作用，但对于在软件中需要进行组合的组件来讲，继承就显得不是那么灵活了，此时，就需要另一种代码结构的定义方式，这就是下一章将要登场的“接口”。