第 3 章　C# 面向对象基础

C# 是一种面向对象语言，在第 2 章学习了 C# 基本语法知识之后，本章将介绍 C# 语言面向对象的知识。本章除了对一般的面向对象知识（如类、对象、继承和多态等）进行介绍外，还对 C# 本身特有的概念（如属性、索引器等）进行介绍。通过第 2 章和本章的学习，读者就可以基本掌握 C# 的基础知识，为后续章节的学习打下语言基础。

学习目标：

- 掌握命名空间的定义和引用。
- 掌握类的概念。
- 掌握属性和索引器的使用。
- 掌握方法和接口的相关操作。
- 理解继承和多态。
- 理解委托和事件。

3.1　面向对象编程概述

3.1.1　面向对象编程方法学

C# 语言的设计哲学是面向对象编程方法学。面向对象编程将数据和代码看成是相互关联、不可分割的整体，采用信息抽象和数据隐蔽技术，用符合人们思维习惯的方法来设计软件，以提高软件的重用性和可维护性。面向对象编程的主要优点如下：

（1）设计和代码易于理解，面向对象的编程方法符合人们的思维习惯。

（2）代码可重用性高，能够直接利用他人已经设计并编写成功的程序。

（3）可扩展性好，能够由基本而通用的解决方案派生出新的解决方案。

3.1.2　面向对象的程序设计步骤

面向对象的程序设计步骤大概包含以下几点：

1. 类的认定

关于类的认定还没有严格的准则，但有些经验准则可以参考，例如：

（1）对于问题空间中自然出现的实体，用类进行模型化。

（2）将方法设计成单用途的。

（3）如果需要对已有方法进行扩展，就设计一个新的方法。

（4）避免冗长的方法。

（5）把那些为多个方法或某个子类所需要的数据，存储在实例变量中。

（6）为类库设计，不要只为你自己或者你目前的应用设计。

2. 类的设计

在任何的面向对象应用中，类实例是系统的主要部分，而且如果采用纯面向对象的方法，那么整个系统就是由类实例组成的。因此，每个独立的类的设计对整个应用系统都有影响。在进行类的设计时，应考虑下面一些因素：

（1）类的公共接口的单独成员应该是类的操作符。

（2）类 A 的实例不应该直接发送消息给类 B 的成员。

（3）操作符是公共的当且仅当类实例的用户可用。

（4）属于类的每个操作符要么访问要么修改类的某个数据。

（5）类必须尽可能少地依赖其他类。

（6）两个类之间的互相作用应该是明显的。

（7）采用子类继承超类的公共接口，开发子类成员为超类的具体实现。

（8）继承结构的根类应该是目标概念的抽象模型。

3. 类层次结构的组织

支持重用是面向对象程序设计的主要任务，继承机制支持两种层次的重用。第一，在高层设计阶段，使用继承机制可以开发出有意义的高级抽象，进而有助于重用。继承关系的重用性使得设计者能够在抽象中识别一般性，并从一般性中产生高级抽象。通过识别这种一般性，并把它从较高的抽象中移出来，它就在当前或今后的设计中变成可重用。第二，在详细设计阶段，继承性支持将已有的类作为新定义类的重用基础，可以把已有的部分代码复制到新的子类中并修改，以适应其新的目的。继承性在已有类和新类之间建立了一种依赖关系，子类的新代码不引起旧代码失效，继承的代码被自动地包含在新定义中，并作为新类的定义被编译。对已有的类的任何修改都被归并到下次编译的新类中。

4. 类模块之间的接口技术

接口使得程序的可移植性和可扩展性更强。例如，可以通过继承机制实现类之间的接口。

5. 对类库和应用构架的支持

类库中的类就好像是一般建筑预制件，可以复杂到整个单元居室，也可以简单到梁柱，规格比较标准，容易被独立使用。但需要应用开发人员自己根据应用特征进行组装，因此类库本身并不是重用的基本单位。相对地，构架则是以构件之间有密切的联系为特征，面向特定的应用范畴，以整个构架而不是其中的单个构件来体现它的能量，因此构架本身是重用的基本单位，一旦与应用特征相符，就可以整体被重用。所以，基于构架的设计是面向对象程序设计的理想目标。

3.2 命名空间

3.2.1 命名空间的概念

在 Microsoft.NET 中，.NET Framework 提供了丰富的类库，C＃程序是利用命名空间将这些类库组织起来并在程序中加以引用的。下面简单介绍几种常用的命名空间，以便于读者在使用的时候查找。

- System：根空间，包含一些基本的类库。
- System. Collections：主要是和集合类相关的类库空间。
- System. Collections. Genaric：泛型集合类库空间。
- System. Data：数据处理类库空间。
- System. Data. Linq：LINQ 技术的类库空间
- System. Data. Odbc：以 ODBC 方式连接数据库的类库空间。
- System. Data. OracleClient：连接 Oracle 数据库的类库空间。
- System. Data. OleDb：以 OLEDB 方式连接数据库的类库空间。
- System. Data. SqlClient：连接 SQL Server 数据库的类库空间。
- System. Diagnostics：用于应用程序诊断的类库空间。
- System. Drawing：用于绘图的类库空间。
- System. Drawing. Drawing2D：专门用于 2D 绘图的类库空间。
- System. Drawing. Printing：用于绘图打印的类库空间。
- System. IO：输入输出的类库空间。
- System. Net：网络应用类库空间。
- System. Runtime. Remoting：远程调用的类库空间。
- System. Security：安全方面应用的类库空间。
- System. Threading：线程相关的类库空间。
- System. Web：Web 相关的类库空间。
- System. Windows. Forms：Windows 窗体相关的类库空间。
- System. Xml：XML 文件处理相关的类库空间。

3.2.2 命名空间的定义和引用

利用关键字 namespace 可以创建新的命名空间，例如：

```
namespace IBMP.Web
{
    ...
}
```

IBMP. Web 为命名空间的名称。命名空间的成员可以在多个文件中定义，即没有必

要将整个命名空间都放在一个单独的源文件中。当编译器遇到多个使用相同命名空间的源文件时，会自动把它们累加到一起。作为一种良好的编程习惯，命名空间的定义也应该反映出一种层次关系，例如，可将公司名称或项目名称（例如上面例子中的 IBMP）等具有唯一性的标识作为根命名空间，这样可保证定义的命名空间不会和其他开发者定义的冲突。在较大型项目中，创建层次型的命名空间是一种常用的做法。例如，可以按照项目的功能模块来组织命名空间：公司名称.项目名称.子项目名称……

引用命名空间有两种方式。第一种方式是使用包含命名空间名称的完整类型名称，例如在程序中书写：

```
System.Xml.XmlTextReader xtr;
```

第二种方式是使用关键字 using 把该命名空间添加到当前命名空间中，即：

```
using System.Xml;
```

添加完该命名空间后就可在程序中直接引用 XmlTextReader 方法了。如果按照第二种方法，不同的命名空间下有相同名称的类要使用，则编译器会提示错误，此时必须按照第一种方法添加命名空间来防止二义性的出现。

3.3　类和对象

3.3.1　类和对象的关系

类是面向对象的程序设计的基本组成单位。类是一种抽象的数据类型，现实生活中的任何东西都可以抽象成一个类。类的定义由三个重要项组成：属性、操作和约束。属性是类或对象的特征，操作则是类或对象能够执行的动作，约束则是类或对象必须遵循的参数。例如，我们把人用类来描述，那么这个类具有一些属性如年龄、性别、受教育程度等，同时这个类也需要一组操作，如说话、跑步、睡觉等。最后这个类要具有一组约束，如规定人一定要吃饭、睡觉等。

对象就是类的实例。例如针对人定义一个类，而张三、李四等都是人这个类的实例，也就是对象。总之，类是对具有相同数据和特性的“一组对象”的抽象，而对象是类的具体实例。在 C# 中，对象的类型定义为“类（class）”，并且总是先定义一个“类”类型，然后用它去定义若干个同类型的对象，即“对象”就是“类”类型的变量。

3.3.2　类的定义

类是一种数据结构，它可以封装数据成员、函数成员和其他的类。类是从实际对象中抽象出来的一种完整自包含的数据结构，它封装了一类对象共有的属性和功能。C# 中的一切类型都是类，所有的语句都必须位于类内部。任何数据类型使用之前都必须先声明。一个类一旦被成功地声明后就可以当做一种新类型来使用。C# 中使用关键字 class 来定义类，格式如下：

```
[修饰符] class<类名>[:<基类或接口>]
{
    [类体]
}[;]
```

[ ]表示内容是可选的，<>表示内容是必需的。修饰符、类体和“;”都是可选的，关键字 class 和类名是必需的，“基类或接口”部分也是可选的，但如果存在，则“基类或接口”是必需的。下面给出了一个最简单的类的定义：

```
class  person
{
}
```

3.3.3 类的成员和访问控制

1. 类的成员

定义在类体内的元素都是类的成员。类的成员主要包含两类：数据成员和函数成员。数据成员是描述状态的，函数成员是描述操作的。类的数据成员包括：

- 字段：字段是在类内定义的成员变量，用来存储描述类的特征的值。
- 常量：常量是类的常量成员。

类的函数成员主要包括：

- 方法：用于实现类执行的计算和操作，方法是以函数的形式来定义的。
- 属性：属性是字段的自然扩展，也具有数据类型，访问属性和字段的语法也一样。但与字段不同，属性不表示存储位置，属性具有访问器，访问器指定在它们的值被读取或写入是需要执行的语句。
- 索引器：提供了一种通过索引的方式访问类的数据的方法。
- 事件：用于定义可由类生成的通知或消息。
- 运算符：用于定义对该类的实例进行运算的运算符，可以对预定义的运算符进行重载。
- 构造函数和析构函数：构造函数是名称与类名相同的方法，类可以有多个接受不同参数的构造函数。构造函数在类的实例初始化时被执行，它使得开发人员可以设置默认值、限制实例化以及编写灵活便于阅读的代码。下面代码定义了一个没有参数的构造函数：

```
class Person
{
    public Person()
    {
      Console.WriteLine("this is a new class");
    }
}
```

同样也可以定义其他构造函数：

```
public person(string name)
{
    Console.WriteLine("this is a new class named {0}",name);
}
```

析构函数也是一类特殊的方法，其名称由类名前加上“～”构成，它用于回收对象中无用的资源。C＃中一个类只能有一个析构函数，并且析构函数是自动执行的，因为.NET Framework 的垃圾回收机制决定何时回收对象中的资源。析构函数的定义方式如下：

```
class person
{
    ~person()
      {
      }
}
```

开发人员可以在～person()中编写代码，但通常是不必要的，因为.NET Framework 会代替开发人员完成资源的释放工作。

2. 访问控制

访问控制的主要作用是指定类和类的成员的可访问性，它是通过访问控制修饰符来定义的，具体来说，访问控制修饰符包含如下几种：

- public：可以从任何程序集访问该类，这是限制最少的一种访问方式。
- protected：为了方便派生类的访问，又希望成员对于外界是隐藏的，这时可使用 protected 修饰符。
- private：仅限于类中的成员可以访问，从类的外部访问其私有成员是不合法的。
- internal：同一个程序集中的类能访问。
- protected internal：这是唯一能使用多个修饰符的情况。只有当前程序集或从基类派生出来的类型能访问。

上述 5 种访问修饰符都可以用于类的成员，如果在声明类的成员时没有出现访问修饰符，则默认成员是私有的。对于类而言，如果类不是在某个类内部声明的，那么这个类就是顶级类。顶级类只能使用 public 和 internal 两种修饰符，并且它们的含义如下：

- public：所修饰的顶级类的可访问域是它所在的程序和任何引用该程序的程序。
- internal：所修饰的顶级类的可访问域是定义它的程序。

3.4　属性和索引器

面向对象编程的封装性原则决定了类中的数据成员是不能直接访问的。不能直接访问的原因很简单：直接访问类的数据成员的前提是必须充分了解类的实现细节，而这有悖于隐藏设计细节的思想，会限制代码的重用性和维护性。另外，如果直接访问类的数据

成员，则类的数据安全性得不到保障，加大调试程序的难度。

为了保证数据成员不被外界直接访问，最好的做法是将数据成员的访问方式都设置为 private。那么在 C＃中又是如何实现对数据成员的访问的呢？C＃定义了一种名为属性的访问器来访问数据成员。

3.4.1 属性

属性提供了对类或对象性质的访问，是对现实世界中实体特征的抽象。类的属性成员用来描述对象的特征，属性本身不存储任何数据，它只是提供了一种数据交换的方式。属性成员是对变量成员的扩展，它更好地体现了类的封装性。属性的声明方式如下：

```
[修饰符]<类型><属性名>
{
    [get {<get 访问器体>}]
    [set {<set 访问器体>}]
}
```

“修饰符”是可选的，修饰符可以是 public、private、static、protected、internal、virtual、override、abstract 等。其中 virtual、static、override 和 abstract 不能同时使用。属性名的命名规则和字段成员的命名规则相同，但是属性名的第一个字母通常都大写。

C＃中的属性是通过 get 和 set 访问器来进行属性值的读写。get 和 set 访问器是可选的，它们分别是由关键字 get 和 set，以及位于一对大括号内的“get 访问器体”和“set 访问器体”代码块组成。根据 get 和 set 访问器的存在情况，属性可分为如下几种类型：

- 只读(read-only)属性：只有 get 访问器的属性。
- 只写(write-only)属性：只有 set 访问器的属性。
- 读写(read-write)属性：同时具有 get 和 set 访问器的属性。

下例是 Person 类中的属性的声明：

```
public  class Person
{
    private int age;
    private string name;
    …
    public string Name                    //定义属性成员 Name
      {
        get
          {return name;}
        set
          {
              if(name !=value)
                  name=value;
          }
        …
```

```
        }
    ...
}
```

从上面的例子可以看出，“get 访问器体”必须用 return 语句来返回，且返回的值的类型必须可以隐式转换为属性类型。set 访问器相当于一个具有单个属性类型值参数和 void 返回类型的方法，用于处理类外部的写入值。set 访问器带有一个特殊的关键字 value，value 就是 set 访问函数的隐式参数，在 set 访问器中通过 value 将外部的输入传递进来，然后赋值给类中的某个成员变量（本例中的 name）。

在 C# 中可以通过属性成员的访问函数来访问类中的某个变量，这样就可以通过属性成员把类本身和调用该类的程序分割开，实现面向对象封装性的要求。Person 类的成员变量 name 外部是无法访问的，但属性 Name 具有 public 的访问权限，外部可以访问属性 Name。因此外部程序可以通过对 Name 属性的访问得到 Person 类中私有成员变量 name 的值。

外部程序访问如下：

```
Person myPerson=new Person();
string myName=myPerson.Name;
```

而不能进行如下操作：

```
string myName=myPerson.name;
```

假设程序中已经实例化了一个对象 myPerson，可以使用下列语句对私有变量 name 进行赋值：

```
myPerson.Name="Tom";
```

程序在运行时若发现成员变量 name 未被赋值，则会自动执行 set 访问器，将“Tom”赋值给 myPerson 的私有变量成员 name。如果此时再执行：

```
string myName=myPerson.Name;
```

则 myName 的值就是“Tom”。

需要说明的是，随着 C# 版本的不断发展，对属性的定义也越来越简单和灵活。C# 3.0 中引入的自动属性将使得创建属性更加方便，自动属性的内容将在第 5 章进行介绍。

3.4.2　索引器

一个类往往会包含数组类型的对象，而索引器（indexer）正好提供了一种通过索引方式访问类的数据信息的方法。C# 中的索引器的工作方式和属性类似，它们的不同之处在于：索引器用来访问数组类型对象元素，而属性用来访问类中的私有成员变量。在定义索引器的时候也要用到 get 访问器和 set 访问器。与属性不同，索引器访问的对象是对象中各元素的值，而不是特定的私有成员变量。定义索引器时不需要给出名称，只要使用关键字 this 即可，它用于引用当前的对象实例。

索引器的定义

```
[修饰符]<类型>this[<参数列表>]
{
    [get{[get访问器体]}]
    [set{[set访问器体]}]
}
```

与属性的定义不同，索引器的名称必须用关键字 this，this 后面一定跟一对方括号，且方括号内至少要有一个参数。修饰符可以是 new、public、private、protected、internal、virtual、override、abstract 和 sealed 等。索引器可以方便地实现对数组的访问，例如下面的程序：

```
using System;
class Person
{
    private string[] name;
    public Person()
    {
        name=new string[] {"Tom","Jack","Rose","John"};
    }
public string  this[int index]              //定义索引器
{
    get
    {
        return name[index];
    }
    set
    {
        name [index]=value;
    }
}
public int GetNameNum()
    {
        return name.Length;
    }
    …
}
public class PersonApp
{
    public static void Main()
    {
        Person myPerson=new Person();
        for(int i=0;i<myPerson. GetNameNum();i++)
```

```
            Console.WriteLine("{0}", myPerson[i]);
    }
}
```

程序运行结果如下所示：

```
Tom
Jack
Rose
John
```

从上面的例子可以看出，索引器的使用很简单，只需在对象名后加上索引操作符和索引即可。

3.5　方　　法

方法又称函数，是用来完成某些操作的算法。在面向对象的语言中，类或对象是通过方法与外界交互的，因此方法是类与外界交互的基本方式。

3.5.1　方法的声明

C＃中方法的声明如下所示：

```
[修饰符]<返回类型><方法名>([形式化参数列表])
{
    <方法体>
}
```

“返回类型”和“方法名”是必需的，“形式化参数列表”用来指定方法的参数，是可选的；“方法体”是必需的，但可以为空。“修饰符”是可选的，除了访问控制修饰符外，它还包括如下几种修饰符：

- static：该方法是类的一部分，而不是类实例的一部分，静态方法只能对静态成员进行访问。
- virtual：指示该方法可以在子类中被覆盖，它不能与 static 或 private 一同使用。
- override：指示该方法覆盖了基类中的同名方法，这样它就能定义子类特有的行为，基类中被覆盖的方法必须是 virtual 方法。
- new：允许继承类中的一个方法“隐藏”基类中同名的非 virtual 方法。它会取代原方法，而不是覆盖。
- sealed：指示禁止派生类覆盖此方法。
- abstract：指示该方法不包含具体实现细节，且必须由子类来实现。
- extern：指示该方法是在外部实现的。