C#语言中的抽象类与接口的研究及应用

Research and Application of Abstract Class and Interface in C# Language

蓝雯飞 陈淑清 周 俊 (武汉 中南民族大学计算机科学学院 430074)

摘要:抽象类和接口是C#语言中两个重要的对象引用类型,是C#程序设计使用多态性的基础。本文对两者进行比较,并通过例子说明了抽象类和接口的区别和使用场合。

关键词:C#语言 类 抽象类 接口

1 引言

C#是Microsoft近年来推出的一个新的编程语言,像Java一样,C#也是完全面向对象的。也就是说C#中的所以代码都封装在类中。C#也具有面向对象语言所共有的三个特点:继承、封装和多态,其中多态主要有抽象类和接口来实现。抽象类和接口在功能上有许多相似之处,因此许多读者不知道如何区分什么时候该使用抽象类,什么时候该使用接口。本文首先从二者的定义和语法的层面对二者比较分析,接下来举了二个例子进一步说明,最后结合二个例子从语义的层面来讨论二者的区别和适用的场合。目的是让读者在面临选择的时候,有规可循,设计出更加高效、灵活、健壮的C#程序。

2 具体类、抽象类和接口

类是对同类对象的共性加以抽象和封装。类是一种数据结构,具体类中包含数据成员、函数成员以及嵌套类型。C#中的类分具体类(非抽象类)和抽象类。二者最大的区别就是具体类可以直接被实例化,而抽象类不允许直接被实例化。而接口是接口用户(使用接口的程序)与接口的实现者(实现接口的类)之间在建立起一种约定,用户只需关心接口有哪些功能而不必关心它们的实现,接口的实现者即类必须按约定为接口中的各功能,其实就是为抽象方法提供实现代码。

下面,我们从具体类、抽象类和接口的定义和语法层面来分析三者的区别和联系。

2.1 具体类

在C#中,具体类用class关键字声明,是C#程序最基本的单位。具体类可以别的类继承而来,也可以不继承别的类,但在C#语言中类都隐式的继承Object类。具体类可以实现一个或多个接口,也可以不实现任何接口。

具体类的成员可以包括:常量、字段、方法、属性、索引函数、事件、运算符、构造函数、析构函数和嵌套的类型声明等。此外类可以继承,在C#中所有的类,不管是抽象类还是具体类,只能单继承。也就是说,一个子类只能有一个父类,而不允许同时继承来自两个或两个以上不同父类的属性和方法。

2.2 抽象类

在C#中,具体类用abstract class关键字声明抽象类。抽象类最显著的特点在与抽象类不能实例化,即在编程时不能new出一个抽象类的对象。其他特点还有:抽象类不能被密封。当具体类继承抽象类时,具体类必须覆盖抽象类的方法。抽象类中不一定包含抽象方法,但包含抽象方法的类一定要被声明为抽象类,否则编译时将出错。

抽象类在面向对象程序设计中的作用:

(1) 为一些相关的类接供了公共基类以便为下层类功能相似但实现代码不同的那些方法对外提供统一的接口,使C#程序具有多态性;

(2) 为下层相关子类提供一些公用方法的实现代码,以减少代码冗余;

(3) 防止类被意外实例化,这对一些直接实例化没有意义的类,可以加上abstract关键字,这样可以增强代码的安全性。

2.3 接口

在C#中接口用Interface来定义。接口实际上定

义了一个协议,实现接口的类必须带有协议。接口的成员包括方法、特性、索引函数和事件。接口中定义的方法只能声明而不能方法体。接口特性声明只是为了表明此特性是可读写的,或只读的还是只可写的。

与类只能单继承不同,接口和类可以多继承接口。通过接口继承我们可以实现接口的组合与扩充。接口常常被用来为具有相似功能的一组类,对外提供一致的服务接口,这一组类可以是相关的,也可以是不相关的。所以,接口往往比抽象类具有更大的灵活性。

从上面的分析我们可以看出抽象类与接口的区别主要体现在以下 3 个方面:

(1) 抽象类中可以包含方法的声明,也可以提供方法的实现代码,而接口中只能提供方法声明,不可以有任何实现代码;

(2) 抽象类与其子类之间存在层次关系,而接口与实现它的类之间则不存在任何层次关系;

(3) 抽象类只能被单继承,而接口可以被多继承。

现在以表格的方式,对实例类,抽象类和接口三者的特点进行概括如表 1 所示:

表 1　实例类、抽象类和接口的比较

	具体类	抽象类	接口
能否实例化	不能	不能	不能
能否多继承	不能	不能	能
能否为方法提供代码	能	能	不能
能继承哪些类型	具体类、抽象类、接口	具体类、抽象类、接口	接口
能包含哪些成员	常量、字段、方法、属性、索引函数、事件、运算符	常量、字段、方法、属性、抽象函数	方法、特性、索引函数和事件

3　实例

本文所有的例子以能够充分说明抽象类和接口的区别以及何时该用二者中的哪一个为目的。为了能简明的说明问题,代码尽可能的简单,方法体以及对特性的操作没有具体实现。

例 1,这里声明一个通讯工具的抽象类,假设现在的通讯工具都具有收发语音信号和收发电子邮件的功能。

//抽象类

```
abstract class CommunicationTool
{
    public abstract string name
    {
        get;
        set;
    }
    public virtual void SendVoice(){}
    public virtual void ReceiveVoice(){}
    public virtual void SendMail(){}
    public virtual void ReceiveMail(){}
}
class Cellphone:CommunicationTool
{
    public override string name{}
    public override void SendVoice(){}
    public override void ReceiveVoice(){}
    public override void SendMail(){}
    public override void ReceiveMail(){}
}
class Computer:CommunicationTool
{
    public override string name{}
    public override void SendVoice(){}
    public override void ReceiveVoice(){}
    public override void SendMail(){}
    public override void ReceiveMail(){}
}
```

例 2,下面声明了二个接口——电话接口和邮件接口,再用一个手机类来继承和实现这二个接口。//接口

```
interface IPhone
{
    void SendVoice();
    void ReceiVevoice();
}
interface IMail
{
    void SendMail();
    void ReceiveMail();
```

```
}
class Cellphone:IPhone,IMail
{
  string name
  {
    get{return name;}
    set{name = value;}
  }
  public void SendVoice(){}
  public void ReceiVevoice(){}
  public void SendMail(){}
  public void ReceiveMail(){}
}
```

4 讨论

下面我们结合上面的二个例子,透过语法从语义的层面来讨论抽象类和接口的区别。

(1) 从继承背后的语义来分析,抽象类是一种对实体对象类型的抽象,而接口则是对某些功能的抽象。当一个类继承一个抽象类时,其背后所包含的语义是"is a subclass"(是这个抽象类的一个子类)。例一就说明了这一点,计算机类和手机类都属于通信工具,是对这个大类的进一步细分。当一个类继承一个接口时所包含的语义是"has these functions"(具有这些功能)。例二正好说明这一点,电话接口的功能包括接收和发送语音,邮件接口的功能包括接收和发送邮件。手机类继承了这二个接口,说明手机能完成这二个接口所要求的所有功能。

(2) 从继承的结构来看,一个抽象类可以派生出很多子类,这些子类有一个共同的父类。我们可以把它看作是一个正三角的结构。在例一中通信工具类派生出了手机和计算机类,计算机又可以派生出 PC 机、笔记本等其他子类。然而它们都有一个共同的祖先——通信工具类。而一个类在继承接口时可以多继承,那么它可以将自身可以实现的所有接口功能来继承这些接口,此时的结构更像一个倒三角。在例二中手机实现了电话和邮件的功能,其实手记还能实现其他的功能,比如:上网、游戏、下载图片等。抽象这些功能的接口之间又能任意的组合,甚至还可以划分层次。但是这些功能的最终实现者就是手机类。

(3) 从实现的思维模式来看,使用抽象类时其思维模式往往是自底向上的,注重的是软件的实体结构。比如在例一中我们抽象出 Computer 和 Cellphone 二个类,在给这二个类定义属性和方法时发现其中很多的属性和方法相同,于是我们想到了抽象出一个 CommunicationTool 类作为这二个类的抽象基类来封装这些属性和方法,实现代码复用。在这种思维模式下我们考虑更多的是实体而不是功能。

而使用接口时我们的思维模式往往是自顶向下的,注重的是软件所能完成的功能。比如我们设计一个软件要有打电话和发邮件的功能,正如例二所定义的,我们抽象出 Iphone 和 Imail 二个接口来封装打电话和发邮件的功能,要求继承这二个接口的类必须为这些功能提供代码。至于会用那个类来或哪些类来实现这些功能,以及这些类之间的关系是次要的。在这种思维模式下我们更多考虑的是功能而不是实体。

(4) 从抽象类和接口所能完成的功能来看,二者是可以互相代替的。比如例一中,我们也可以定义一个通信接口(ICommunication),然后用 Computer 和 Cellphone 二个类来继承这个接口。同样我们也可以把例二中的接口改为抽象类来定义,然后再继承时再增加方法来完成所有功能。我们之所以没有这样做,是因为例子中的定义更符合抽象类和接口背后所包含的语义。

(5) 从抽象类和接口的特点来看,抽象类更好的体现了多态性,接口更好的体现了封装性。抽象类中可以有实现代码,接口中不能有实现代码。当有大量公共代码时应首先考虑用抽象类,它能实现代码的复用。但是这也是抽象类继承的最大缺陷,当继承的层次过多时,子类中含有许多无用的代码。这使得子类对象过于复杂。而从接口继承时就可以避免这个问题,因为从接口可以多继承,我们可以将接口进行任意的组合,选取其中要实现的功能,继承对应的接口。这样类中的方法就会简单、清晰。

(6) 在对外提供接口(这里的接口是指软件的各个模块或包之间的接口)时,最好用接口而不是抽象类来实现。这样能使内部代码无论怎样改变都不会影响到模块间的调用关系,也就是说其他模块的代码不会随之改变。 (下转第 62 页)

(上接第 73 页)

我们有以下几个方面的理由优先选择接口。一，接口打破了抽象类所要求的层次关系，可以抽象出不相关类的相同行为，所以比抽象类更加灵活；二，它指明了所有实现接口的类所必须实现的方法，这使得错误在编译时就能发现，增加了代码的健壮性；三，通过接口我们只要关心对象之间的交互的方法，而不必关心对象所对应的具体类，体现了面向对象设计的封装性。

5 总结

通过以上的讨论我们对抽象类和接口有了更加深入的认识，了解了二者的差别，明确了二者的适用场合。通常情况下，我们可以按照或参考以上的分析来进行选择。然而实际中的问题总是千变万化、错综复杂的，这些只是原则性规律的总结，我们应该在软件设计和实现的过程中不断的实践和总结，积累经验。这样在设计软件架构时才能做到有的放矢。

参考文献

1 C#语言参考手册，清华大学出版社，微软公司，东方人华编著，2001.7。

2 C# Essentials. 中国电力出版社，Ben Albabari Peter Drayton Brad Merrill 著，刘基诚 译，2001.8。

3 C#实用教程，人民邮电出版社，吴军编著，2001.10。

4 C#和 ASP.NET 程序设计教程，清华大学出版社，木林森编著，2002.1 。