C# 多态性 C# 接口（ Interface ）

C# 运算符重载

您可以重定义或重载 C# 中内置的运算符。因此，程序员也可以使用用户自定义类型的运算符。重载运算符是具有特殊名称的函数，是通过关键字 **operator** 后跟运算符的符号来定义的。与其他函数一样，重载运算符有返回类型和参数列表。

例如，请看下面的函数：

```
public static Box operator+ (Box b, Box c)
{
   Box box = new Box();
   box.length = b.length + c.length;
   box.breadth = b.breadth + c.breadth;
   box.height = b.height + c.height;
   return box;
}
```

上面的函数为用户自定义的类 Box 实现了加法运算符（ + ）。它把两个 Box 对象的属性相加，并返回相加后的 Box 对象。

运算符重载的实现

下面的程序演示了完整的实现：

实例

```
using System;

namespace OperatorOvlApplication
{
   class Box
   {
      private double length;      // 长度
      private double breadth;     // 宽度
      private double height;      // 高度

      public double getVolume()
      {
         return length * breadth * height;
      }
      public void setLength( double len )
      {
         length = len;
      }

      public void setBreadth( double bre )
      {
         breadth = bre;
      }
```

```
      public void setHeight( double hei )
      {
         height = hei;
      }
      // 重载 + 运算符来把两个 Box 对象相加
      public static Box operator+ (Box b, Box c)
      {
         Box box = new Box();
         box.length = b.length + c.length;
         box.breadth = b.breadth + c.breadth;
         box.height = b.height + c.height;
         return box;
      }

   }

   class Tester
   {
      static void Main(string[] args)
      {
         Box Box1 = new Box();          // 声明 Box1，类型为 Box
         Box Box2 = new Box();          // 声明 Box2，类型为 Box
         Box Box3 = new Box();          // 声明 Box3，类型为 Box
         double volume = 0.0;           // 体积

         // Box1 详述
         Box1.setLength(6.0);
         Box1.setBreadth(7.0);
         Box1.setHeight(5.0);

         // Box2 详述
         Box2.setLength(12.0);
         Box2.setBreadth(13.0);
         Box2.setHeight(10.0);

         // Box1 的体积
         volume = Box1.getVolume();
         Console.WriteLine("Box1 的体积:  {0}", volume);

         // Box2 的体积
         volume = Box2.getVolume();
         Console.WriteLine("Box2 的体积:  {0}", volume);

         // 把两个对象相加
         Box3 = Box1 + Box2;

         // Box3 的体积
         volume = Box3.getVolume();
         Console.WriteLine("Box3 的体积:  {0}", volume);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Box1 的体积： 210
Box2 的体积： 1560
Box3 的体积： 5400
```

可重载和不可重载运算符

下表描述了 C# 中运算符重载的能力：

运算符	描述
+, -, !, ~, ++, --	这些一元运算符只有一个操作数，且可以被重载。
+, -, *, /, %	这些二元运算符带有两个操作数，且可以被重载。
==, !=, <, >, <=, >=	这些比较运算符可以被重载。
&&, ||	这些条件逻辑运算符不能被直接重载。
+=, -=, *=, /=, %=	这些赋值运算符不能被重载。
=, ., ?:, ->, new, is, sizeof, typeof	这些运算符不能被重载。

实例

针对上述讨论，让我们扩展上面的实例，重载更多的运算符：

实例

```
using System;

namespace OperatorOvlApplication
{
    class Box
    {
       private double length;      // 长度
       private double breadth;     // 宽度
       private double height;      // 高度

       public double getVolume()
       {
         return length * breadth * height;
       }
      public void setLength( double len )
      {
          length = len;
      }

      public void setBreadth( double bre )
      {
```

```
        breadth = bre;
    }

    public void setHeight( double hei )
    {
        height = hei;
    }
    // 重载 + 运算符来把两个 Box 对象相加
    public static Box operator+ (Box b, Box c)
    {
        Box box = new Box();
        box.length = b.length + c.length;
        box.breadth = b.breadth + c.breadth;
        box.height = b.height + c.height;
        return box;
    }

    public static bool operator == (Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.length == rhs.length && lhs.height == rhs.height
           && lhs.breadth == rhs.breadth)
        {
            status = true;
        }
        return status;
    }
    public static bool operator !=(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.length != rhs.length || lhs.height != rhs.height
            || lhs.breadth != rhs.breadth)
        {
            status = true;
        }
        return status;
    }
    public static bool operator <(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.length < rhs.length && lhs.height
            < rhs.height && lhs.breadth < rhs.breadth)
        {
            status = true;
        }
        return status;
    }

    public static bool operator >(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.length > rhs.length && lhs.height
            > rhs.height && lhs.breadth > rhs.breadth)
        {
```

```
            status = true;
        }
        return status;
    }

    public static bool operator <=(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.length <= rhs.length && lhs.height
            <= rhs.height && lhs.breadth <= rhs.breadth)
        {
            status = true;
        }
        return status;
    }

    public static bool operator >=(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.length >= rhs.length && lhs.height
           >= rhs.height && lhs.breadth >= rhs.breadth)
        {
            status = true;
        }
        return status;
    }
    public override string ToString()
    {
        return String.Format("({0}, {1}, {2})", length, breadth, height);
    }

}

class Tester
{
    static void Main(string[] args)
    {
      Box Box1 = new Box();          // 声明 Box1，类型为 Box
      Box Box2 = new Box();          // 声明 Box2，类型为 Box
      Box Box3 = new Box();          // 声明 Box3，类型为 Box
      Box Box4 = new Box();
      double volume = 0.0;   // 体积

      // Box1 详述
      Box1.setLength(6.0);
      Box1.setBreadth(7.0);
      Box1.setHeight(5.0);

      // Box2 详述
      Box2.setLength(12.0);
      Box2.setBreadth(13.0);
      Box2.setHeight(10.0);

     // 使用重载的 ToString() 显示两个盒子
```

```
            Console.WriteLine("Box1:  {0}", Box1.ToString());
            Console.WriteLine("Box2:  {0}", Box2.ToString());

            // Box1 的体积
            volume = Box1.getVolume();
            Console.WriteLine("Box1 的体积:  {0}", volume);

            // Box2 的体积
            volume = Box2.getVolume();
            Console.WriteLine("Box2 的体积:  {0}", volume);

            // 把两个对象相加
            Box3 = Box1 + Box2;
            Console.WriteLine("Box3:  {0}", Box3.ToString());
            // Box3 的体积
            volume = Box3.getVolume();
            Console.WriteLine("Box3 的体积:  {0}", volume);

            //comparing the boxes
            if (Box1 > Box2)
              Console.WriteLine("Box1 大于 Box2");
            else
              Console.WriteLine("Box1 不大于 Box2");
            if (Box1 < Box2)
              Console.WriteLine("Box1 小于 Box2");
            else
              Console.WriteLine("Box1 不小于 Box2");
            if (Box1 >= Box2)
              Console.WriteLine("Box1 大于等于 Box2");
            else
              Console.WriteLine("Box1 不大于等于 Box2");
            if (Box1 <= Box2)
              Console.WriteLine("Box1 小于等于 Box2");
            else
              Console.WriteLine("Box1 不小于等于 Box2");
            if (Box1 != Box2)
              Console.WriteLine("Box1 不等于 Box2");
            else
              Console.WriteLine("Box1 等于 Box2");
            Box4 = Box3;
            if (Box3 == Box4)
              Console.WriteLine("Box3 等于 Box4");
            else
              Console.WriteLine("Box3 不等于 Box4");

            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Box1： (6, 7, 5)
Box2： (12, 13, 10)
Box1 的体积： 210
Box2 的体积： 1560
Box3： (18, 20, 15)
Box3 的体积： 5400
Box1 不大于 Box2
Box1 小于 Box2
Box1 不大于等于 Box2
Box1 小于等于 Box2
Box1 不等于 Box2
Box3 等于 Box4
```

C# 多态性 C# 接口（ Interface ）

1 篇笔记

写笔记

operator 关键字用于在类或结构声明中声明运算符。运算符声明可以采用下列四种形式之一：

```
public static result-type operator unary-operator ( op-type operand )
public static result-type operator binary-operator ( op-type operand, op-type2 operand2 )
public static implicit operator conv-type-out ( conv-type-in operand )
public static explicit operator conv-type-out ( conv-type-in operand )
```

参数：

- result-type 运算符的结果类型。
- unary-operator 下列运算符之一：+ - ! ~ ++ — true false
- op-type 第一个（或唯一一个）参数的类型。
- operand 第一个（或唯一一个）参数的名称。
- binary-operator 其中一个：+ - * / % & | ^ << >> == != > < >= <=
- op-type2 第二个参数的类型。
- operand2 第二个参数的名称。
- conv-type-out 类型转换运算符的目标类型。
- conv-type-in 类型转换运算符的输入类型。

注意：

前两种形式声明了用户定义的重载内置运算符的运算符。并非所有内置运算符都可以被重载（请参见可重载的运算符）。op-type 和 op-type2 中至少有一个必须是封闭类型（即运算符所属的类型，或理解为自定义的类型）。例如，这将防止重定义整数加法运算符。

后两种形式声明了转换运算符。conv-type-in 和 conv-type-out 中正好有一个必须是封闭类型（即，转换运算符只能从它的封闭类型转换为其他某个类型，或从其他某个类型转换为它的封闭类型）。

运算符只能采用值参数，不能采用 ref 或 out 参数。

C# 要求成对重载比较运算符。如果重载了==，则也必须重载!=，否则产生编译错误。同时，比较运算符必须返回bool类型的值，这是与其他算术运算符的根本区别。

C# 不允许重载=运算符，但如果重载例如+运算符，编译器会自动使用+运算符的重载来执行+=运算符的操作。

运算符重载的其实就是函数重载。首先通过指定的运算表达式调用对应的运算符函数，然后再将运算对象转化为运算符函数的实参，接着根据实参的类型来确定需要调用的函数的重载，这个过程是由编译器完成。

任何运算符声明的前面都可以有一个可选的属性（C# 编程指南）列表。

```
using System;
using System.Collections.Generic;
using System.Text;

namespace OperatorOverLoading
{
    class Program
    {
        static void Main(string[] args)
        {
            Student s1 = new Student(20, "Tom");
            Student s2 = new Student(18, "Jack");
            Student s3 = s1 + s2;

            s3.sayPlus();
            (s1 - s2).sayMinus();
            Console.ReadKey();
        }
    }
    public class Student
    {
        public Student() { }
        public Student(int age, string name)
        {
            this.name = name;
            this.age = age;

        }
        private string name;
        private int age;

        public void sayPlus()
        {
            System.Console.WriteLine("{0} 年龄之和为：{1}", this.name, this.age);

        }
        public void sayMinus() {
            System.Console.WriteLine("{0} 年龄之差为：{1}", this.name, this.age);
        }
```

```
        //覆盖“+”操作符
        public static Student operator +(Student s1, Student s2)
        {
            return new Student(s1.age + s2.age, s1.name + " And " + s2.name);

        }
        //覆盖“-”操作符
        public static Student operator -(Student s1, Student s2) {
            return new Student(Math.Abs(s1.age - s2.age), s1.name + "And" + s2.name);
        }
    }
}
```

人菜自尊强 1年前 (2017-09-26)