HW3 2150276 沈卓成

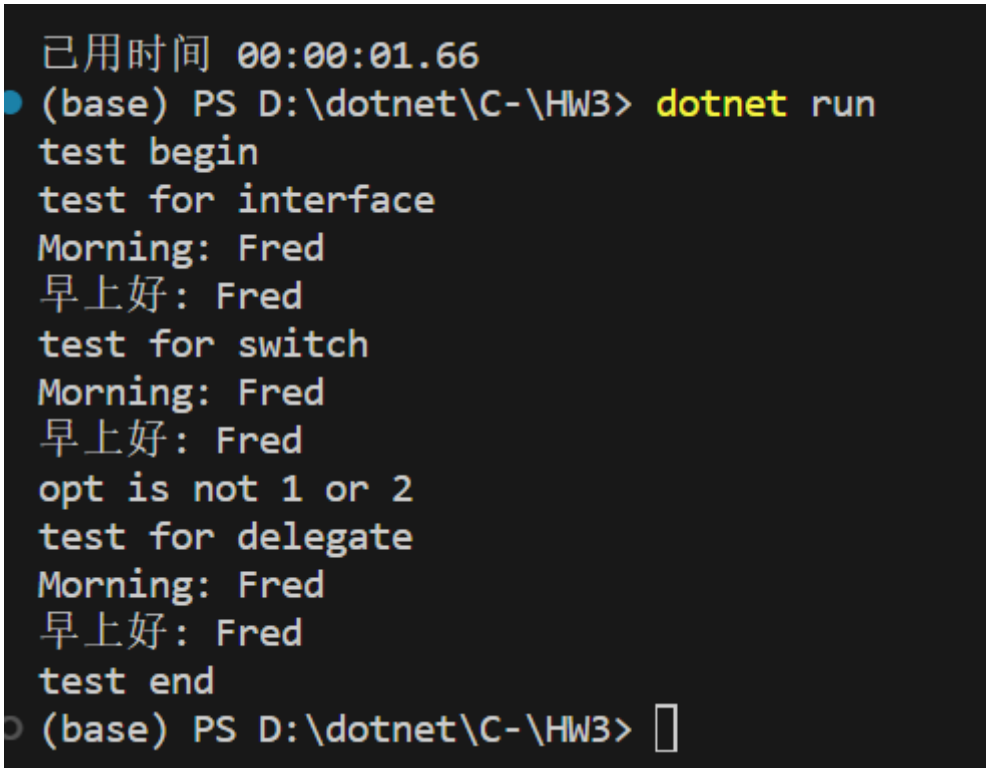
1 委托与事件

Code:

```
using System;
namespace test_delegate
{
    public delegate void GreetingDelegate(string name);

    public interface IGreeting
    {
        void GreetingPeople(string name);
    }

    public class EnglishGreeting_cls : IGreeting
    {
        public void GreetingPeople(string name)
        {
            Console.WriteLine("Morning: "+ name);
        }

    }

    public class ChineseGreeting_cls :IGreeting
    {
        public void GreetingPeople(string name)
        {
            Console.WriteLine("早上好: "+ name);
        }

    }

    public class Program
    {
        static void EnglishGreeting(string name)
        {
            Console.WriteLine("Morning: "+ name);
        }

        static void ChineseGreeting(string name)
        {
            Console.WriteLine("早上好: "+ name);
        }
        static void GreetPeople(string name, GreetingDelegate fn)
        {
            fn(name);
        }
        public static void Main()
        {
            Console.WriteLine("test begin");
            TestInterface();
```

```
            TestSwitch();
            TestDelegate();
            Console.WriteLine("test end");
        }

        static void GreetPeopleWithInterface(string name, IGreeting
makegreeting)
        {
            makegreeting.GreetingPeople(name);
        }
        static void TestInterface()
        {
            Console.WriteLine("test for interface");
            GreetPeopleWithInterface("Fred", new EnglishGreeting_cls());
            GreetPeopleWithInterface("Fred", new ChineseGreeting_cls());

        }


        static void TestSwitch()
        {
            Console.WriteLine("test for switch");
            GreetPeopleWithSwitch("Fred", 1);
            GreetPeopleWithSwitch("Fred", 2);
            GreetPeopleWithSwitch("Fred", 3);
        }
        static void GreetPeopleWithSwitch(string name, int opt)
        {
            switch (opt)
            {
                case 1:
                    Console.WriteLine("Morning: "+ name); break;
                case 2:
                    Console.WriteLine("早上好: "+ name); break;
                default:
                    Console.WriteLine("opt is not 1 or 2"); break;
            }

        }

        static void TestDelegate()
        {
            Console.WriteLine("test for delegate");
            GreetingDelegate d;
            d = EnglishGreeting;
            d += ChineseGreeting;
            GreetPeople("Fred", d);
        }


    }

};
```

Ans:

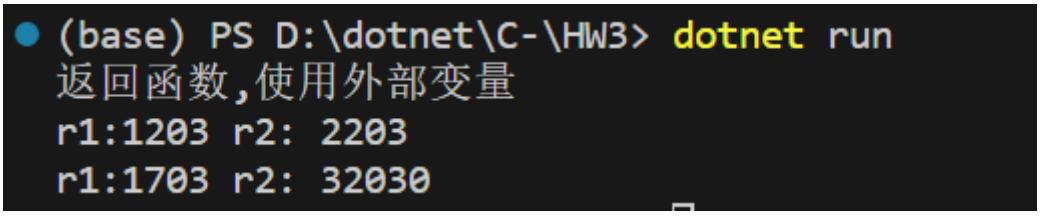


2 委托与闭包

Code:

```
using System;
public class Test
{
    static Func<int, int, int> SomeFunc(int one)
    {
        int temp = 200;
        return (two, three) =>
        {
            temp += one;
            return one + two + three + temp;
        };

    }

    public static void Main()
    {
        Console.WriteLine("返回函数,使用外部变量");
        Func<int, int, int> Func1 = SomeFunc(500);
        Func<int, int, int> Func2 = SomeFunc(1000);
        int r1 = Func1(1,2);
        int r2 = Func2(1,2);
        Console.WriteLine("r1:{0} r2: {1}",r1,r2);
        r1 = Func1(1, 2);
        r2 = Func2(1,2);
        Console.WriteLine("r1:{0} r2: {1}0",r1, r2);
    }

}
```

Ans:



调用r1或r2一次，都会比上次多一个one变量的值，所以，闭包内的变量长期贮存在内存中，随Func的实例对象一直存在。

3 协变和逆变

Code:

```
public class Test{
    public static void Main()
    {
        Car c= new Car();
        Vehicle v=c;
        v.drive();
    }

}
```

Output: Car drive

子类向上可以转为基类，但是基类不能向下转为子类。

Code:

```
public class Test
{
    delegate T MyDelegate<T>();
    public static void Main()
    {
        MyDelegate<Car> carDelegate = () => new Car();
        MyDelegate<Vehicle> vehicleDelegate = carDelegate;
        vehicleDelegate().drive();

    }
}
```

Error: 无法进行隐式转换

但是如果使用out关键词：

Code:

```
public class Test
{
    delegate T MyDelegate<out T>();
    public static void Main()
    {
        MyDelegate<Car> carDelegate = () => new Car();
        MyDelegate<Vehicle> vehicleDelegate = carDelegate;
        vehicleDelegate().drive();

    }
}
```

Output: Car drive

同理，下面两组例子也将进行比较：

Code:

```
public class Test
{
    delegate void OtherDelegate<T>(T _);
    public static void Main()
    {
        OtherDelegate<Car>carDelegate =(t)=>
        {
            Console.WriteLine(t.GetType().Name);
        };

        carDelegate(new Vehicle());
    }

}
```

Error: Vehicle无法转换为Car

下面这段代码使用in关键词：

Code:

```
public class Test
{
    delegate void OtherDelegate<in T>(T _);
    public static void Main()
    {
        OtherDelegate<Car>carDelegate =(t)=>
        {
            Console.WriteLine(t.GetType().Name);
        };

        carDelegate(new Vehicle());
    }

}
```

这里多了一个in关键字，它告诉编译器，要么传递T作为委托的参数类型，要么传递T的派生类型。Car是Vehicle的派生类，因此可以通过编译。

Summary

在C#中，协变（covariance）和逆变（contravariance）是与泛型类型参数的关系相关的概念，特别是在委托和接口上。这些概念允许在保持类型安全的同时提供灵活的类型转换。

协变 (Covariance)

- **定义**: 允许方法返回更派生的类型作为结果。
- **场景**: 当你有一个泛型接口或委托，它的泛型类型参数在输出位置时，你可以使用派生类型作为返回类型。
- **关键字**: `out`（在接口或委托的泛型参数前使用）。

逆变 (Contravariance)

- **定义**: 允许方法接受更基类的类型作为参数。
- **场景**: 当你有一个泛型接口或委托，它的泛型类型参数在输入位置时，你可以使用基类作为参数类型。
- **关键字**: `in`（在接口或委托的泛型参数前使用）。

Note

在C#中，支持协变的参数只能用于方法的返回值，而支持逆变的参数只能用于方法的参数，这是由它们各自的安全性和设计决定的。下面分别解释为什么会这样。

协变（Covariance）

在C#中，支持协变的参数只能用于方法的返回值，而支持逆变的参数只能用于方法的参数，这是由它们各自的安全性和设计决定的。

协变允许你将一个派生类型当作更一般的基类型来使用。在方法返回值的情况下，如果一个方法应该返回一个基类型对象，它实际上返回一个派生类型的对象是安全的，因为派生类型继承了基类型的所有属性和方法。因此，客户端期待的任何操作都可以在派生类型的对象上执行。

逆变则是相反的情况。它允许你将一个更一般的基类型当作派生类型来使用。在方法参数的情况下，如果一个方法期望一个派生类型作为参数，而你传入一个基类型，这是安全的。因为基类型不包含派生类型特有的任何属性或方法，所以方法可以安全地对基类型的对象执行操作，而不会触及不存在的派生类型特有的成员。

- 协变保证了返回类型的安全性。它确保了你总是得到一个你期望的类型或其派生类型的对象。
- 逆变保证了参数类型的安全性。它确保了你可以传递一个更广泛的类型给期望特定类型的方法。