委托,Lambda表达式和事件

siki 微信：devsiki QQ:804632564

委托

什么是委托？

如果我们要把方法当做参数来传递的话，就要用到委托。简单来说委托是一个类型，这个类型可以赋值一个方法的引用。

声明委托

在C#中使用一个类分两个阶段，首选定义这个类，告诉编译器这个类由什么字段和方法组成的，然后使用这个类实例化对象。在我们使用委托的时候，也需要经过这两个阶段，首先定义委托，告诉编译器我们这个委托可以指向哪些类型的方法，然后，创建该委托的实例。

定义委托的语法如下：

```
delegate void IntMethodInvoker(int x);
```

定义了一个委托叫做IntMethodInvoker,这个委托可以指向什么类型的方法呢？

这个方法要带有一个int类型的参数，并且方法的返回值是void的。

定义一个委托要定义方法的参数和返回值，使用关键字delegate定义。

定义委托的其他案例：

```
delegate double TwoLongOp(long first,long second);
delegate string GetAString();
```

使用委托

```
private delegate string GetAString();

static void Main(){
        int x = 40;
        GetAString firstStringMethod = new GetAString(x.ToString);
        Console.WriteLine(firstStringMethod());
}
```

在这里我们首先使用GetAString委托声明了一个类型叫做fristStringMethod,接下来使用new 对它进行初始化，使它引用到x中的ToString方法上，这样firstStringMethod就相当于x.ToString，我们通过firstStringMethod()执行方法就相当于x.ToString()

通过委托示例调用方法有两种方式

```
        fristStringMethod();
        firstStringMethod.Invoke();
```

委托的赋值

GetAString firstStringMethod = new GetAString(x.ToString);只需要把方法名给一个委托的构造方法就可以了

GetAString firstStringMethod = x.ToString;也可以把方法名直接给委托的实例

简单委托示例

定义一个类MathsOperations里面有两个静态方法，使用委托调用该方法

```
class MathOperations{
        public static double MultiplyByTwo(double value){
                return value*2;
        }
        public static double Square(double value){
                return value*value;
        }
}
delegate double DoubleOp(double x);
static void Main(){
        DoubleOp[] operations={ MathOperations.MultiplyByTwo,MathOperations.Square };
        for(int i =0;i<operations.Length;i++){
                Console.WriteLine("Using operations "+i);
                ProcessAndDisplayNumber( operations[i],2.0 );
        }
}
static void ProcessAndDisplayNumber(DoubleOp action,double value){
        double res = action(value);
        Console.Writeline("Value :"+value+" Result:"+res);
}
```

Action委托和Func委托

除了我们自己定义的委托之外，系统还给我们提供过来一个内置的委托类型，Action和Func

Action委托引用了一个void返回类型的方法，T表示方法参数，先看Action委托有哪些

Action

Action<in T>

Action<in T1,in T2>

Action<in T1,in T2 .... inT16>

Func引用了一个带有一个返回值的方法，它可以传递0或者多到16个参数类型，和一个返回类型

Func<out TResult>

Func<in T,out TResult>

Func<int T1,inT2,,,,,,in T16,out TResult>

案例1

```
delegate double DoubleOp(double x);
```

如何用Func表示

```
Func<double,double>
```

案例2-对int类型排序

对集合进行排序，冒泡排序

```
bool swapped = true;
do{
        swapped = false;
        for(int i =0;i<sortArray.Length -1;i++){
                if(sortArray[i]>sortArray[i+1]){
                        int temp= sortArray[i];
                        sortArray[i]=sortArray[i+1];
                        sortArray[i+1]=temp;
                        swapped = true;
                }
        }
}while(swapped);
```

这里的冒泡排序只适用于int类型的，如果我们想对他进行扩展，这样它就可以给任何对象排序。

案例2-雇员类

```
class Employee{
        public Employ(string name,decimal salary){
                this.Name = name;
                this.Salary = salary;
        }
        public string Name{get;private set;}
        public decimal Salary{get;private set;}
        public static bool CompareSalary(Employee e1,Employee e2){
                return e1.salary>e2.salary;
        }
}
```

案例2-通用的排序方法

```
public static void Sort<T>( List<T> sortArray,Func<T,T,bool> comparision ){
        bool swapped = true;
        do{
                swapped = false;
                for(int i=0;i<sortArray.Count-1;i++){
                        if(comparision(sortArray[i+1],sortArray[i])){
                                T temp = sortArray[i];
                                sortArray[i]=sortArray[i+1];
                                sortArray[i+1]=temp;
                                swapped = true;
                        }
                }
        }while(swapped);
}
```

案例2-对雇员类排序

```
static void Main(){
        Employee[] employees = {
                new Employee("Bunny",20000),
                new Employee("Bunny",10000),
                new Employee("Bunny",25000),
                new Employee("Bunny",100000),
                new Employee("Bunny",23000),
                new Employee("Bunny",50000),
                };
        Sort(employees,Employee.CompareSalary);
        输出
}
```

多播委托

前面使用的委托都只包含一个方法的调用，但是委托也可以包含多个方法，这种委托叫做多播委托。使用多播委托就可以按照顺序调用多个方法，多播委托只能得到调用的最后一个方法的结果，一般我们把多播委托的返回类型声明为void。

```
Action action1 = Test1;
action2+=Test2;
action2-=Test1;
```

多播委托包含一个逐个调用的委托集合，如果通过委托调用的其中一个方法抛出异常，整个迭代就会停止。

取得多播委托中所有方法的委托

```
Action a1 = Method1;
a1+=Method2;

Delegate[] delegates=a1.GetInvocationList();
foreach(delegate d in delegates){
        //d();
        d.DynamicInvoke(null);
}
```

遍历多播委托中所有的委托，然后单独调用

匿名方法

到目前为止，使用委托，都是先定义一个方法，然后把方法给委托的实例。但还有另外一种使用委托的方式，不用去定义一个方法，应该说是使用匿名方法（方法没有名字）。

```
Func<int,int,int> plus = delegate (int a,int b){
        int temp = a+b;
        return temp;
};
int res = plus(34,34);
Console.WriteLine(res);
```

在这里相当于直接把要引用的方法直接写在了后面，优点是减少了要编写的代码，减少代码的复杂性

Lambda表达式-表示一个方法的定义

从C#3.0开始，可以使用Lambda表达式代替匿名方法。只要有委托参数类型的地方就可以使用Lambda表达式。刚刚的例子可以修改为

```
Func<int,int,int> plus = (a,b)=>{ int temp= a+b;return temp; };
int res = plus(34,34);
Console.WriteLine(res);
```

Lambda运算符“=>”的左边列出了需要的参数，如果是一个参数可以直接写　a=>(参数名自己定义),如果多个参数就使用括号括起来，参数之间以，间隔

多行语句

1，如果Lambda表达式只有一条语句，在方法快内就不需要花括号和return语句，编译器会自动添加return语句

```
Func<double,double> square = x=>x*x;
```

添加花括号，return语句和分号是完全合法的

```
Func<double,double> square = x=>{
        return x*x;
        }
```

2，如果Lambda表达式的实现代码中需要多条语句，就必须添加花括号和return语句。

Lambda表达式外部的变量

通过Lambda表达式可以访问Lambda表达式块外部的变量。这是一个非常好的功能，但如果不能正确使用，也会非常危险。示例：

```
int somVal = 5;
Func<int,int> f = x=>x+somVal;
Console.WriteLine(f(3));//8
somVal = 7;
Console.WriteLine(f(3));//10
```

这个方法的结果，不但受到参数的控制，还受到somVal变量的控制，结果不可控，容易出现编程问题，用的时候要谨慎。

事件

事件(event)基于委托，为委托提供了一个发布/订阅机制，我们可以说事件是一种具有特殊签名的委托。

什么是事件？

事件（Event）是类或对象向其他类或对象通知发生的事情的一种特殊签名的委托.

事件的声明

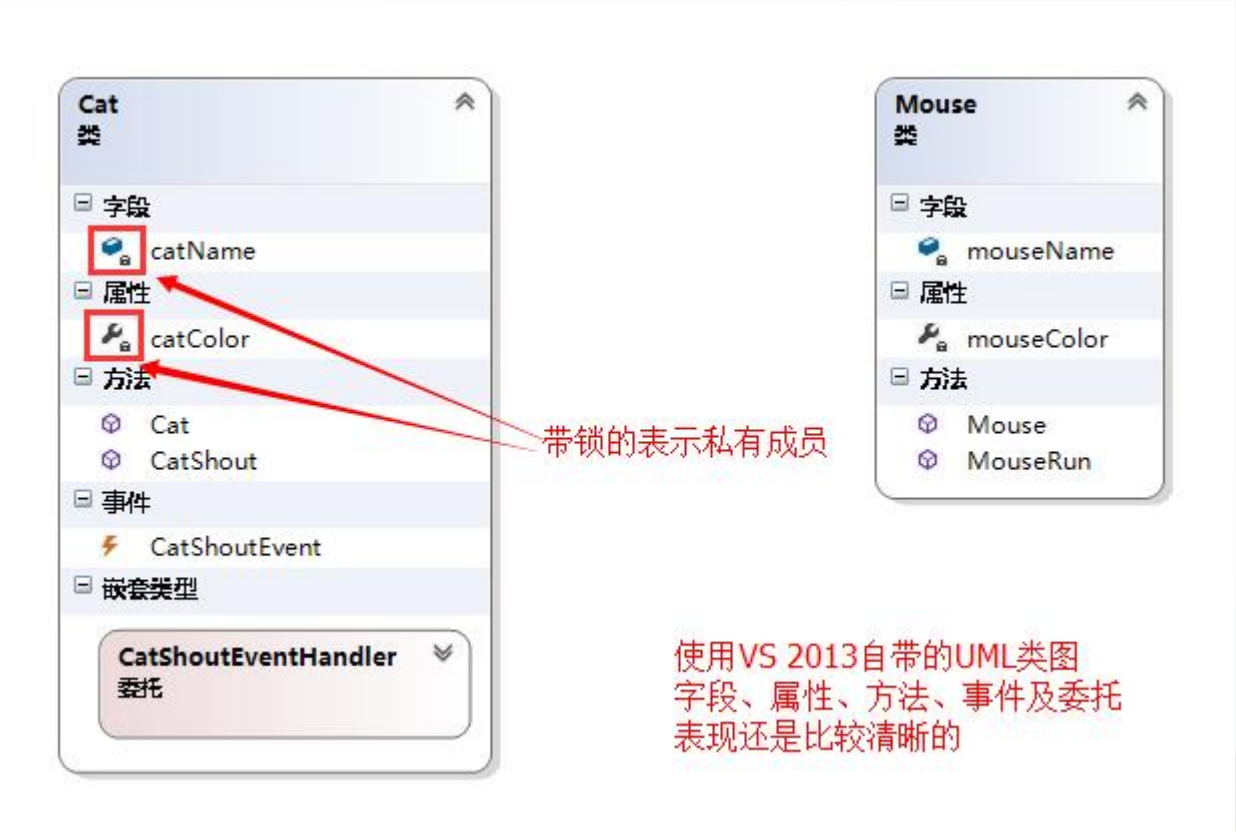
public event 委托类型 事件名;

事件使用event关键词来声明，他的返回类值是一个委托类型。

通常事件的命名，以名字+Event 作为他的名称，在编码中尽量使用规范命名，增加代码可读性。

为了更加容易理解事件，我们还是以前面的动物的示例来说明，有三只动物，猫(名叫Tom),还有两只老鼠（Jerry和Jack）,当猫叫的时候，触发事件(CatShout)，然后两只老鼠开始逃跑(MouseRun)。接下来用代码来实现。(设计模式-观察者模式)

猫捉老鼠的UML图


Cat
类
字段
catName
属性
catColor
方法
Cat
CatShout
事件
CatShoutEvent
嵌套类型
CatShoutEventHandler
委托
带锁的表示私有成员
Mouse
类
字段
mouseName
属性
mouseColor
方法
Mouse
MouseRun
使用VS 2013自带的UML类图
字段、属性、方法、事件及委托
表现还是比较清晰的

Cat类和Mouse类

```
class Cat
{
    string catName;
    string catColor { get; set; }
    public Cat(string name, string color)
    {
        this.catName = name;
        catColor = color;
    }
    public void CatShout()
    {
        Console.WriteLine(catColor+" 的猫 "+catName+" 过来了，喵！喵！喵！\n");
        //猫叫时触发事件
        //猫叫时，如果CatShoutEvent中有登记事件，则执行该事件
        if (CatShoutEvent != null)
            CatShoutEvent();
    }
    public delegate void CatShoutEventHandler();
    public event CatShoutEventHandler CatShoutEvent;
}
class Mouse
{
    string mouseName;
    string mouseColor { get; set; }
    public Mouse(string name, string color)
```

运行代码

```
Console.WriteLine("[场景说明]: 一个月明星稀的午夜,有两只老鼠在偷油吃\n");
Mouse Jerry = new Mouse("Jerry", "白色");
Mouse Jack = new Mouse("Jack", "黄色");

Console.WriteLine("[场景说明]: 一只黑猫蹑手蹑脚的走了过来\n");
Cat Tom = new Cat("Tom", "黑色");
Console.WriteLine("[场景说明]: 为了安全的偷油，登记了一个猫叫的事件\n");

Tom.CatShoutEvent += new Cat.CatShoutEventHandler(Jerry.MouseRun);
Tom.CatShoutEvent += new Cat.CatShoutEventHandler(Jack.MouseRun);


Console.WriteLine("[场景说明]: 猫叫了三声\n");
Tom.CatShout();
Console.ReadKey();
```

事件与委托的联系和区别

-事件是一种特殊的委托，或者说是受限制的委托，是委托一种特殊应用，只能施加+=,-=操作符。二者本质上是一个东西。

-event ActionHandler Tick; // 编译成创建一个私有的委托示例, 和施加在其上的add, remove方法.

-event只允许用add, remove方法来操作，这导致了它不允许在类的外部被直接触发，只能在类的内部适合的时机触发。委托可以在外部被触发，但是别这么用。

-使用中，委托常用来表达回调，事件表达外发的接口。

-委托和事件支持静态方法和成员方法, delegate(void * pthis, f_ptr), 支持静态返方法时, pthis传null.支持成员方法时, pthis传被通知的对象.

-委托对象里的三个重要字段是, pthis, f_ptr, pnext, 也就是被通知对象引用, 函数指针/地址, 委托链表的下一个委托节点.

关注公众号

发布最新的视频，文章和教学资源

THANK

@siki 微信：devsiki QQ:804632564