C# 笔记

郑华

2018 年 9 月 19 日

第一章　基础

1.1　类型装换

1.2　类

1.3　继承

1.4　多态

1.5　预处理

1.6　C# 属性 (Property)

1.7　泛型

1.8　反射

第二章　高级

2.1　接口

IEumerable 接口　IEnumerable

只包含一个方法GetEnumerator()，它返回一个可用于循环访问集合的IEnumerator 对象，继承实现接口，完成该方法之后，就可以在调用时用foreach 了。

```
public interface IEnumerable{
    // 返回一个循环访问集合的枚举器
    IEnumerator GetEnumerator();
}
```

IEnumerator 接口　IEnumerator

它是一个真正的集合访问器 (枚举器)，没有它，就不能使用foreach语句遍历集合或数组，因为只有IEnumerator 对象才能访问集合中的项，假如连集合中的项都访问不了，那么进行集合的循环遍历是不可能的事情了。

```
public interface IEnumerator{
    // 获取集合中的当前元素
    object Current {get;}
    // 将枚举数推进到集合的下一个元素
    bool MoveNext();
    // 将枚举数设置为其初始位置，改位置位于集合中第一个元素之前
    void Reset()
}
```

参考　https://www.cnblogs.com/aehyok/p/3642103.html

2.2 枚举

2.3 正则表达式

2.4 文件操作

2.5 属性

2.6 集合

2.7 委托

定义委托 delegate result-type Identifier ([parameters]);

- `result-type`: 返回值的类型，和**方法**的返回值类型**一致**
- `Identifier`: 委托的名称
- `parameters`: 参数，要引用的**方法**带的**参数**

实例化委托 Identifier objectName = new Identifier(functionName)

- **Identifier：** 这个是委托名字
- **objectName：** 委托的实例化对象
- **functionName：**是该委托对象所指向的函数的名字

  -> 对于这个函数名要特别注意：定义这个委托对象肯定是在类中定义的，那么如果所指向的函数也在该类中，不管该函数是静态还是非静态的，那么就直接写函数名字就可以了；如果函数是在别的类里面定义的 *public*、*internal*，但是如果是静态，那么就直接用类名. 函数名，如果是非静态的，那么就类的对象名. 函数名，这个函数名与该对象是有关系的，比如如果函数中出现了 *this*，表示的就是对当前对象的调用。

委托推断 Identifier objectName = functionName

当我们需要定义委托对象并实例化委托的时候，就**可以只传送函数的名称**，即函数的地址.

这里面的`functionName` 与实例化委托的`functionName` 是一样的，没什么区别，满足上面的规则。

匿名委托 DelegateTest anonDel = delegate([parameters])//implements

直接定义一个 lambda 来实现临时函数。

多播委托 deleIntifier += functionX

前面使用的每个委托都只包含一个方法调用，调用委托的次数与调用方法的次数相同，如果要调用多个方法，就需要多次给委托赋值，然后调用这个委托。**委托也可以包含多个方法，这时候要向委托对象中添加多个方法，这种委托称为多播委托**，多播委托有一个方法列表，如果调用多播委托，就可以连续调用多个方法，即先执行某一个方法，等该方法执行完成之后再执行另外

一个方法，这些方法的参数都是一样的，**这些方法的执行是在一个线程中执行的**，而不是每个方法都是一个线程，**最终将执行完成所有的方法**。

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一个方法抛出了异常，整个迭代就会停止**。

委托运算符 | = 、+=、-= |

= | Identifier objName= new Identifier(functionName) | 或者 | objName=functionName1 |

这里的“=”号表示**清空** `objectName` 的方法列表，然后将 `functionName` **加入**到 `objectName` 的**方法列表**中

+= | Identifier objName += new Identifier(functionName) | 或者 | objName+=functionName1 |

这里的“+=”号表示**在原有的方法列表不变的情况下**，将 `functionName1` **加入**到 `objectName` 的方法列表中。可以在方法列表中加上多个相同的方法，执行的时候也会执行完所有的函数，哪怕有相同的，就会多次执行同一个方法。

-= | Identifier objName -= new Identifier(functionName) | 或者 | objName-=functionName1 |

这里的“-=”号表示在 `objectName` 的方法列表中减去一个`functionName1`。**可以在方法列表中多次减去相同的方法**，减一次只会减一个方法，如果列表中无此方法，那么减就没有意义，对原有列表无影响，也不会报错。

2.8 事件

2.8.1 自定义事件

声明一个委托 | Delegate result-type delegateName ([parameters]); |

这个委托可以在类 A 内定义也可以在类 A 外定义

```
public delegate void DelegateClick (int a);
```

声明一个基于某个委托的事件 | Event delegateName eventName; |

eventName 不是一个类型，而是一个具体的对象，这个具体的对象只能在类 A 内定义而不能在类 A 外定义

```
public event DelegateClick Click;
```

在类 A 中定义一个触发该事件的方法 ReturnType FunctionName（[parameters]）...

```
ReturnType FunctionName（[parameters]）
{
    If(eventName != null)
    {
        eventName([parameters]);
        或者eventName.Invoke([parameters]);
    }
}

 public class butt
 {
        public event DelegateClick Click;
        public void OnClick(int a)
        {
            if(Click != null)
                Click.Invoke(a);
              //Click(a);//这种方式也是可以的
            MessageBox.Show("Click();");
        }
  }
```

触发事件之后，事件所指向的函数将会被执行。这种执行是通过事件名称来调用的，就像委托对象名一样的。

触发事件的方法**只能在 A 类中定义**，事件的实例化，以及实例化之后的实现体都只能在 A 类外定义。

初始化 A 类的事件 在类 B 中定义一个类 A 的对象，并且让类 A 对象的那个事件指向类 B 中定义的方法，这个方法要与事件关联的委托所限定的方法吻合。

```
butt b = new butt();
b.Click += new DelegateClick (Fm_Click); //事件是基于委托的，所以委托推断一样适用，下面的
    语句一样有效：b.Click += Fm_Click;
```

触发 A 类的事件 在 B 类中去调用 A 类中的触发事件的方法：用 A 类的对象去调用 A 类的触发事件的方法。

```
b.OnClick(10000);
```

2.8.2 控件事件

控件事件委托 EventHandler Public delegate void EventHandler(object sender,EventArgs e);

委托 EventHandler 参数 一般第一个参数都是object sender，第二个参数可以是**任意类型**，不同的委托可以有不同的参数，**只要它派生于 EventArgs 即可**。

实例

```
// 第1步：定义参数
public class TimeInfoEventArgs : EventArgs
{
    public TimeInfoEventArgs(int hour,int minute,int second)
    {
        this.hour = hour;
        this.minute = minute;
        this.second = second;
    }
    public readonly int hour;
    public readonly int minute;
    public readonly int second;
}

// 第2步：定义委托
// 定义名为SecondChangeHandler的委托，封装不返回值的方法，
// 该方法带参数，一个clock类型对象参数，一个TimeInfoEventArgs类型对象
public delegate void SecondChangeHandler(object clock, TimeInfoEventArgs
    timeInformation);


// 第3步：定义事件发送者
// 被其他类观察的钟（Clock）类，该类发布一个事件：SecondChange。观察该类的类订阅了该事
    件。
public class Clock
{
    // 代表小时，分钟，秒的私有变量
    int _hour;
    public int Hour
    {
        get { return _hour; }
        set { _hour = value; }
    }
    private int _minute;
```

```
        public int Minute
        {
            get { return _minute; }
            set { _minute = value; }
        }
        private int _second;
        public int Second
        {
            get { return _second; }
            set { _second = value; }
        }

        // 要发布的事件
        public event SecondChangeHandler SecondChange;

        // 触发事件的方法
        protected void OnSecondChange(object clock,TimeInfoEventArgs timeInformation)
        {
            // Check if there are any Subscribers
            if (SecondChange != null)
            {
                // Call the Event
                SecondChange(clock, timeInformation);
            }
        }
        // 让钟（Clock）跑起来，每隔一秒钟触发一次事件
        public void Run()
        {
            for (; ; )
            {
                // 让线程Sleep一秒钟
                Thread.Sleep(1000);
                // 获取当前时间
                System.DateTime dt = System.DateTime.Now;
                // 如果秒钟变化了通知订阅者
                if (dt.Second != _second)
                {
                    // 创造TimeInfoEventArgs类型对象，传给订阅者
                    TimeInfoEventArgs timeInformation =new TimeInfoEventArgs(dt.Hour, dt.
                        Minute, dt.Second);

                    // 通知订阅者
                    OnSecondChange(this, timeInformation);
                }
```

```
                // 更新状态信息
                _second = dt.Second;
                _minute = dt.Minute;
                _hour = dt.Hour;
            }
        }
    }

/* ======================== Event Subscribers ================================ */
        // 一个订阅者。DisplayClock订阅了clock类的事件。它的工作是显示当前时间。
        public class DisplayClock
        {
            // 传入一个clock对象，订阅其SecondChangeHandler事件
            public void Subscribe(Clock theClock)
            {
                theClock.SecondChange += new SecondChangeHandler(TimeHasChanged);
            }

            // 实现了委托匹配类型的方法
            public void TimeHasChanged(object theClock, TimeInfoEventArgs ti)
              {
                  Console.WriteLine("Current Time: {0}:{1}:{2}",
                    ti.hour.ToString(),
                    ti.minute.ToString(),
                    ti.second.ToString());
              }
        }

        // 第二个订阅者，他的工作是把当前时间写入一个文件
        public class LogClock
        {
            public void Subscribe(Clock theClock)
            {
                theClock.SecondChange += new SecondChangeHandler(WriteLogEntry);
            }
            // 这个方法本来应该是把信息写入一个文件中
            // 这里我们用把信息输出控制台代替
            public void WriteLogEntry(object theClock, TimeInfoEventArgs ti)
            {
                Clock a = (Clock)theClock;
                Console.WriteLine("Logging to file: {0}:{1}:{2}",
                  a.Hour.ToString(),
                  a.Minute.ToString(),
                  a.Second.ToString());
```

```
            }
        }

    /* ======================== Test Application ================================ */
        // 测试拥有程序
        public class Test
        {
            public static void Main()
            {
                // 创建clock实例
                Clock theClock = new Clock();
                // 创建一个DisplayClock实例，让其订阅上面创建的clock的事件
                DisplayClock dc = new DisplayClock();
                dc.Subscribe(theClock);
                // 创建一个LogClock实例，让其订阅上面创建的clock的事件
                LogClock lc = new LogClock();
                lc.Subscribe(theClock);
                // 让钟跑起来
                theClock.Run();
            }
        }
```

https://blog.csdn.net/chaixinke/article/details/45396269