Delegate,Action,Func,Predicate的使用与区别

在书写代码时，常常会用到委托，这个在winform下较常见，但自定义Delegate时，我们常常发现Delegate必须全局可见，才能在需要的地方进行使用，而对于私有的delegate对象，在本类中进行使用，这似乎是不方便的。下边我们来看传统的Delegate的写法。

```
public delegate void MyDelegate(string name);
    public class MyBlogBase
    {
        private MyDelegate mydelegate;
    }
```

必须保证MyDelegate放在类的外边，才能在其他地方可见，并使用

Action，Func的出现改变了这一局面，这两个其实说白了就是系统定义好的Delegate，他有很多重载的方法，便于各种应用情况下的调用。他在系统的System命名空间下，因此全局可见。

Action<T>:封装一个方法，该方法只有一个参数并且不返回值。其中T是可接收的任何类型。使用代码如下：

```
public class MyBlogBase
    {
        public string myName;
        Action<string> myAction;
        public MyBlogBase()
        {
            //1. myAction = delegate(string curName) { myName = curName; };
            //2. myAction = new Action<string>(SetAction);
            //3.
            myAction = curname => { myName = curname; };

        }
        private void SetAction(string name)
        {
            myName = name;
        }
    }
```

在上例中，给出了3种使用Action的方法，

方法一：采用匿名委托，

方法二：指定一个实际的方法。

方法三：使用Lamda表达式。以上3中用法均可运行。

在实际应用中要比原始的定义Delegate方便，灵活。

Func<T in,T Tresult>:封装一个具有一个参数并返回 TResult 参数指定的类型值的方法。其实个人感觉，Func和Action的区别很明显，也很直接。二者都是委托，但Func能返回函数执行结果，而Action返回类型是Void，这个区别很明显，在具体的项目中，也很容易确定该使用那个。下文就说明具体Func的代码调用：

```
public string myName;
        Func<string, string> myFunc;
        public MyBlogBase()
        {
            //myFunc = delegate(string curName) { return curName.ToUpper(); };
            //myFunc = new Func<string, string>(SetFunc);
            myFunc = name => { return name.ToUpper(); };
        }
        private string SetFunc(string name)
        {
            return name.ToUpper();
        }
        public void StartFun(string curName)
        {
            myName = myFunc(curName);
        }
```

如上3种写法，都是合适的Func定义，大家可以选择适合自己的编程模式，其实匿名方法，有个优点，就是可以直接使用当前函数出现的变量，代码更简洁，但可能有些人觉得不易读。

Predicate<T>：也是一种委托，表示定义一组条件并确定指定对象是否符合这些条件的方法.此方法常在集合的查找中被用到，如：数组，正则拼配的结果集中被用到。使用此方法快捷方便，使用代码如下：

```
        Predicate<int> myPredicate;
        int[] myNum = new int[8] { 12, 33, 89, 21, 15, 29, 40, 52 };
        public int[] myResult;

        public MyBlogBase()
        {
            myPredicate = delegate(int curNum)
                    { if (curNum % 2 == 0) return true;
                      else return false;
                    };
        }
        public void StartPredicate()
        {
            myResult = Array.FindAll(myNum, myPredicate);
        }
```

上例中说明了Predicate的使用，FindAll方法中，参数2即是一个Predicate，在具体的执行中，每一个数组的元素都会执行指定的方法，如果满足要求返回true，并会被存放在结果集中，不符合的则被剔除，最终返回的集合，即是结果判断后想要的集合，此方法应用场景感觉像迭代中的yield。当然此方法也可以书写上边类似Action和Func的3中方式，此处省略。

为了更好的验证运行效果，添加Test项目及进行测试，把代码粘帖出来分享一下：

```
        [TestMethod]
        public void TestAction()
        {
            MyBlogBase blogObj = new MyBlogBase();
            blogObj.StartAction("ywg369");
            Assert.AreEqual("ywg369", blogObj.myName);
        }
        [TestMethod]
        public void TestFunc()
        {
            MyBlogBase blogObj = new MyBlogBase();
            blogObj.StartFun("ywg369");
            Assert.AreEqual("YWG369", blogObj.myName);
        }
        [TestMethod]
        public void TestPredicate()
        {
            MyBlogBase blogObj = new MyBlogBase();
            blogObj.StartPredicate();
            Assert.AreEqual(3, blogObj.myResult.Length);
        }
```