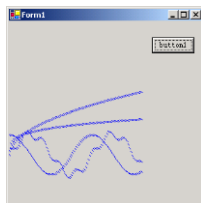


## 委托示例



DelegatePlotFun.cs

## C#4以上版本定义了很多的委托

- Action<T1, T2>
- Func<T1, T2, TResult>
  - 如 new Func<double, double>(Math.Sin);

```
Action 委托
Action() 委托
Action<T> 委托
Action<T1, T2> 委托
Action<T1, T2, T3> 委托
Action<T1, T2, T3, T4> 委托
Action<T1, T2, T3, T4, T5> 委托
Action<T1, T2, T3, T4, T5, T6> 委托
Action<T1, T2, T3, T4, T5, T6, T7> 委托
Action<T1, T2, T3, T4, T5, T6, T7, T8> 委托
Action<T1, T2, T3, T4, T5, T6, T7, T8, T9> 委托
Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10> 委托
Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11> 委托
Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12> 委托

Func(T) 委托
Func(T, TResult) 委托
Func<T> 委托
Func<T, TResult> 委托
Func<T1, T2, TResult> 委托
Func<T1, T2, T3, TResult> 委托
Func<T1, T2, T3, T4, TResult> 委托
Func<T1, T2, T3, T4, T5, TResult> 委托
Func<T1, T2, T3, T4, T5, T6, TResult> 委托
Func<T1, T2, T3, T4, T5, T6, T7, TResult> 委托
Func<T1, T2, T3, T4, T5, T6, T7, T8, TResult> 委托
Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, TResult> 委托
Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, TResult> 委托
Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, TResult> 委托
Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, TResult> 委托
```

## 委托的合并

- 委托的合并----多播MulticastDelegate
  - 一个委托实例中可以“包含”多个函数
  - 调用委托，就是调用其中多个函数
  - 多个函数间的先后顺序是没有意义的
  - 返回值也就没有太多意义
- 运算符 + - += -=
  - 动态地增减其中的函数
  - 提高了程序的灵活性

DelegateMultiTest.cs Delegate温度.cs

## 委托的转换与相等

- 委托的转换
  - 按声明的名称判断
  - 以下两个不能互相转换或加减
    - delegate void Df(int a);
    - delegate void E(int a);
- 委托的相等
  - 按内容（即其中“包含的函数”）来判断
  - 有点点像两个字符串的“相等”与否的判断

DelegateEquals.cs

## 总之

- 委托相当于函数指针
- 但它类型更安全，是引用类型
- 且功能更强大，有多播功能



## C#中的事件

大致上：  
事件-----回调函数

## 用户界面中的事件

- 按钮点击事件
- 基本的写法

```
this.button1.Click += new System.EventHandler(this.button1_Click);  
  
private void button1_Click(object sender, EventArgs e)  
{  
    this.label1.Text = DateTime.Now.ToString();  
}
```

## 事件

- 事件的声明
  - `public event 委托名 事件名;`
- 事件的注册与移除
  - 事件名 `+=` 或 `-=`
    - 在事件所在类的外面，只能用以上两个运算符
- 事件的发生（激发）
  - 事件名（参数列表）
  - 相当于回调所注册的函数

## 实例

- 网络爬虫
- EventWhenDownload.cs

## 定义及使用事件的6步曲

- 公用的
  - 声明事件参数类： `class xxxEventArgs{}`
  - 声明委托： `delegate void xxxEventHandler(obj, args)`
- 在一个类中
  - 定义事件： `public event 类型 名称`
  - 发生事件： 事件名(参数)
- 在别的类中
  - 定义一个方法： `void 方法名(obj, args)`
  - 注册事件： `xxx.事件 += new 委托(方法名)`

## 事件与委托的关系

- 事件有点像委托类型的实例
  - 事件一定有相关的委托类型
  - 与委托实例一样，事件也“包含”多个函数
  - 事件的运算受更多限制（在类外只能用 `+=` 或 `-=`）

## 事件与委托的关系

- 事件比委托实例更复杂：
- 可以定义事件存取器
  - 修饰符 event 委托类型名 事件名
  - {
  - add{ e += value; }
  - remove{ e -= value; }
  - }

## 总之

- 事件是一种消息机制
- 事件源调用事件，别的类注册事件
- 事件的类型是一个委托

## 综合示例

- 使用C#多种语法要素（特别是event及Exception）
- 参见 BankSystem-v1-v2



## 第4章 C#语言高级特性

### 4.3 lambda表达式

## csharp语言新特性

- C#2.0 引入泛型
- C#3.0 引入Lambda及Linq
- C#4.0 更多的动态特性dynmaic

## 泛型(Generic)

- `List<Book> books=new List<Book>();`
- `Book book = books[0];`
- `//以前要用强制类型转换`
- `ArrayList books = new ArrayList();`
- `Book book = (Book) books[0];`

## 匿名方法

- `delegate(参数){ 方法体 }`
- 可以当一个匿名方法
  - ▣ `new Thread(`
    - `new ThreadStart( delegate(){ .....} ) );`
- 可以被隐式转换为一个兼容的委托类型
  - ▣ `new Thread(delegate(){ .....} );`

## Lambda表达式

- 相当于匿名方法的简写
  - ▣ 省略`delegate`, 甚至省略参数类型
  - ▣ 直接用(参数)`=>`{语句或表达式}
- 例
  - ▣ `button1.Click += (sender,e)=>{.....}`
  - ▣ `new Thread( ()=>{....} ).Start();`
  - ▣ `PlotFun( x=>x*x, 0, 100 );`

## Lambda表达式

- Lambda表达式比匿名函数简单
- 匿名函数多一个功能：
  - ▣ 不写 ( 参数 ) 的匿名函数
  - ▣ 可以转成任意多个参数的委托

## Linq

- LINQ : Language Integrated Query
- 常见的形式
  - ▣ `from c in customers`
  - ▣ `where c.Age>10`
  - ▣ `orderby c.Name`
  - ▣ `select new {c.Name, c.Phone}`
- 相当于
  - ▣ `customers`
  - ▣ `.Where(c=> c.Age>10)`
  - ▣ `.OrderBy(c => c.Name).`
  - ▣ `.Select(c => new { c.Name, c.Phone })`

## Linq示例

- `int[] arr = new int[] { 8, 5, 89, 3, 56, 4, 1, 58 };`
- `var m = from n in arr where n < 5 orderby n select n*n;`
- `foreach (var n in m)`
- `{`
- `Console.WriteLine(n);`
- `}`

## 总之

- 匿名函数使用delegate
- Lambda表达式使用=>
- Linq使用from, where, select

## 示例

- 在不同C#版本中使用delegate
  - 注：新版本兼容旧版本
- 参见：MethodDelegateLambda

运算符重载

## 第4章 C#语言高级特性

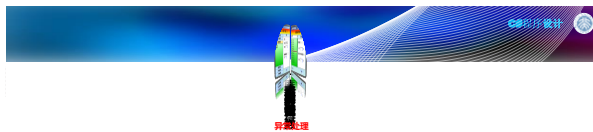
### 4.4 运算符重载

## 运算符

- 使用运算符的例子
  - `this.Location += new Size(10,10);`
  - `TimeSpan diff = date2 - date1;`
  - `String s1, s2; ... if( s1==s2)`
- 运算符有时比方法名更直观
  - 如 两个复数用 `a+b` 比 `a.Add(b)`更直观
  - 但要慎用
- 运算符重载有一些限制
  - 如成对，如类型要求，如有的不能重载
  - 更详细的内容，请参见文档

## 运算符的声明

- 一元运算符声明的形式如下：
  - `public static 类型 operator 一元运算符 ( 类型 参数名 ) { ..... }`
- 二元运算符声明的形式如下：
  - `public static 类型 operator 二元运算符 ( 类型 参数名, 类型 参数名 ) { ..... }`
- 类型转换运算符声明的形式如下：
  - `public static implicit operator 类型 ( 类型 参数名 ) { ..... }`
  - `public static explicit operator 类型 ( 类型 参数名 ) { ..... }`



## 4.5 异常处理

### 异常的概念

- C#中的异常处理
  - try{ ..... }
  - catch(Exception e){ ..... }
  - finally{ ..... }
- System.Exception类
  - public Exception();
  - public Exception(string s);
  - Message属性
  - StackTrace属性

### 几种常用的异常类

- System.OutOfMemoryException
- System.StackOverflowException
- System.NullReferenceException
- System.TypeInitializationException
- System.InvalidCastException
- System.ArrayTypeMismatchException
- System.IndexOutOfRangeException
- System.MulticastNotSupportedException
- System.ArithmeticException
- System.DivideByZeroException
- System.OverflowException

### 捕获和处理异常

- 捕获异常
  - try{ }
  - catch(AException e1){ }
  - catch(BException e2){ }
  - catch(更一般的Exception e){ }
  - finally{ }
  - 注：catch()表示捕获所有种类的异常

### 抛出异常

- throw语句抛出一个异常的语法为：
  - throw expression
- 一般是这样的：
  - if( xxxxxx ) throw new SomeException(信息)

## 创建用户自定义异常类

- 从Exception或ApplicationException继承
- 重抛异常
  - throws;
- 异常链接
  - throw new Exception("msg", e);
  - 这里e称为内部异常
  - InnerException属性
  - 使得外部能进一步知道内部的异常原因

http://www.dongyang.com ExceptionInner.cs

43

## 算术溢出与checked

- 对溢出进行检查
  - 对整个程序 csc /checked XXXX.cs
  - 对部分程序
    - 针对表达式: checked(表达式) 及 unchecked(表达式)
    - 针对块语句: checked{.....} 及 unchecked{.....}
- 对溢出异常进行捕获
  - try{ } catch( OverflowException e ) { }

http://www.dongyang.com CheckedTest.cs

44

## 总之

- C#中的异常处理
  - try{ ..... }
  - catch(Exception e){ ..... }
  - finally{ ..... }
- 自定义异常类
  - 从Exception或ApplicationException继承

http://www.dongyang.com CheckedTest.cs

45



Attribute

http://www.dongyang.com 唐大壮 北京大学

46

## 第4章 C#语言高级特性

### 4.6 Attribute

http://www.dongyang.com 唐大壮 北京大学

47

## 使用Attribute的例子

- 用在类上的
  - [Serializable]
  - public sealed class String : IComparable, ICloneable, IConvertible, IEnumerable
- 用在方法上的
  - [STAThread]
  - static void Main()

http://www.dongyang.com 唐大壮 北京大学

48



## Attribute

Attribute是与类、结构、方法等元素相关的额外信息，是对元信息的扩展。  
通过Attribute可以使程序、甚至语言本身的功能得到增强。

```
[HelpUrl("http://SomeUrl/APIDocs/SomeClass")]
class SomeClass
{
    [WebMethod]
    void GetCustomers() { ... }

    string Test([SomeAttr] string param1) {...}
}
```

## 使用系统定义的Attribute

- 使用Attribute的一般方式
  - 在程序集、类、域、方法等前面用[]表示
  - 可以省略“Attribute”几个字母，只写xxxx
  - 可以带参数
    - 位置参数（相当于构造方法带的参数）
    - 命名参数（域名或属性名=值）
- 示例
  - 在Main()方法使用[STAThread]
  - 使用“过时”：AttributeObsolete.cs
  - 使用“条件”：AttributeConditional.cs
  - 在结构上、枚举上使用：StructLayout, Flag
  - 在程序集级别应用Attribute
    - [assembly: AssemblyCompany("")]

## 自定义Attribute

- 1. 声明Attribute类
  - 从System.Attribute继承而来
  - 名字要用xxxxAttribute
- 2. 使用Attribute类
  - 在类及成员上面使用方括号
  - 可以省略后缀Attribute
- 3. 通过反射访问属性

- Attributes are classes

```
class HelpUrlAttribute : System.Attribute {
    public HelpUrlAttribute(string url) { ... }
    ...
}
```

- Attached to types and members

```
[HelpUrl("http://SomeUrl/APIDocs/SomeClass")]
class SomeClass { ... }
```

- Attributes can be queried at runtime

```
Type type = Type.GetType("SomeClass");
object[] attributes = type.GetCustomAttributes();
```



## 第4章 C#语言高级特性

### 4.7 C#语言中的其他成分

## 编译预处理

- 1. 标识符声明
  - ▣ #define 定义一个标识符;
  - ▣ #undef “取消定义” 一个标识符。
- 2. 条件处理
  - ▣ #if, #elif, #else, #endif
- 3. 信息报告
  - ▣ #error 和 #warning
- 4. 行号标记
  - ▣ #line 行号 “文件名”

## unsafe及指针

- 1. unsafe
  - ▣ 用于修饰类、方法等
- 2. fixed及指针
  - ▣ fixed( 类型 \* 指针名 = 表达式 ) 语句
- 3. sizeof运算符
  - ▣ sizeof( 简单或结构类型名 )
- 4. stackalloc
  - ▣ 在栈上分配的内存，而不是在堆上，因此不会担心内存被垃圾回收器自动回收。

## 指针示例

```
class FileStream: Stream
{
    int handle;

    [DllImport("kernel32", SetLastError=true)]
    static extern unsafe bool ReadFile(int hFile,
        void* lpBuffer, int nBytesToRead,
        int* nBytesRead, overlapped* lpoverlapped);

    public unsafe int Read(byte[] buffer, int index, int count) {
        int n = 0;
        fixed (byte* p = buffer) {
            ReadFile(handle, p + index, count, &n, null);
        }
        return n;
    }
}
```

## 其他关键字

- 1. lock
  - ▣ 多线程程序中，lock可以将某个对象加锁
- 2. volatile
  - ▣ 随时可能被程序以外的其他因素所修改。域被volatile修饰时，会阻止编译器对它的优化。



程序的组织

## 4.8 程序的组织

- 名字空间 程序的逻辑组织
- 嵌套类型 类中嵌套类型
- 程序集 程序的物理组织

## 名字空间

- 1. 名字空间的概念
  - 逻辑划分；避免名字冲突
- 2. 名字空间的声明
  - `namespace xxx.xxxx { }`
  - 可嵌套
- 3. 名字空间的导入
  - `using xxx.xxxx;`
- 4. 使用别名
  - `using 别名 = 名字空间或类名;`

<http://www.ditang.com> NamespaceUsing.cs

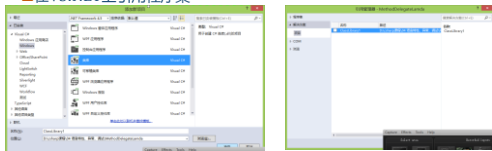
## 嵌套类型

- 嵌套类型的概念
  - 类型中的类型
  - `class A{ public class B{ public struct C{} } }`
  - `new A.B.C();`
- 嵌套类型的可访问性
  - 受各个层次的限制

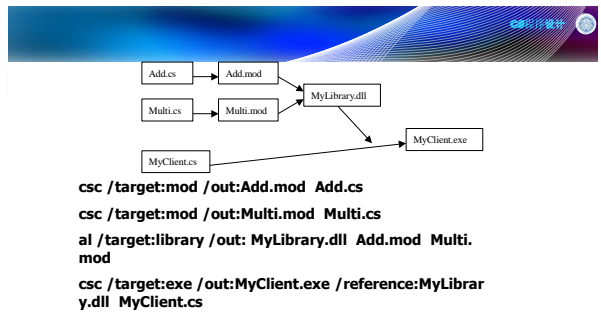
<http://www.ditang.com> NestedAccessibility.cs

## 程序集

- 模块(module)
- 程序集(assembly)
  - 在VS.NET上引用程序集



<http://www.ditang.com> 唐大壮 北京大学



<http://www.ditang.com> 唐大壮 北京大学

## C#语法的小结

- 类型声明
- 类的成员

<http://www.ditang.com> 唐大壮 北京大学

## 类型声明

- 类型声明是C#程序的主体，它可以位于名字空间中，也可以是嵌套的类型。
- 类型声明包括以下几种：
  - 1) 类 class
  - 2) 结构 struct
  - 3) 接口 interface
  - 4) 枚举 enum
  - 5) 委托 delegate

<http://www.ditang.com> 唐大壮 北京大学

## 类的成员

- 1) 常数(const)
  - 它代表了与类相关的常数数据。
- 2) 域(field)
  - 它是类中的变量。
- 3) 方法(method)
  - 它实现了可以被类实现的计算和行为。
- 4) 属性(property)
  - 它定义了命名的属性和与对这个属性进行读写的相关行为。
- 5) 事件(event)
  - 它定义了由类产生的通知
- 6) 索引(indexer)
  - 它允许类的实例通过与数组相同的方法来索引。

<http://www.ditang.com> 唐大仕 北京大学

47

## 类的成员 (续)

- 7) 运算符(operator)
  - 它定义了可以被应用于类的实例上的表达式运算符。
- 8) 实例构造函数(instance constructor)
  - 它执行需要对类的实例进行初始化的动作。
- 9) 析构函数(destructor)
  - 类的实例被清除时实现的动作(结构不能有析构函数)。
- 10) 静态构造函数(static constructor)
  - 它执行对类本身进行初始化的动作。
- 11) 类型(type)
  - 它代表位于类中的类型。

<http://www.ditang.com> 唐大仕 北京大学

48

## 练习

- 1. 练习C#语言中的几个高级特性
  - 参见ch04下的示例
- 2. 练习异常的处理

<http://www.ditang.com> 唐大仕 北京大学

49

## 进一步阅读

- 电子稿  
《C#语言高级特性.doc》
- 文章  
《The Code Project –  
Exception Handling Best Practices in .NET - .NET.htm》

<http://www.ditang.com> 唐大仕 北京大学

50

## 第4章 C#语言高级特性

# 问题与讨论

[dstang2000@263.net](mailto:dstang2000@263.net)

<http://www.ditang.com> 唐大仕 北京大学

51