什么是回调函数？

回调函数是一种特殊的函数，它作为参数传递给另一个函数，并在被调用函数执行完毕后被调用。回调函数通常用于事件处理、异步编程和处理各种操作系统和框架的API。

.Net Framework通过委托提供回调函数机制。委托确保回调函数是安全的，并且允许顺序调用多个方法、支持调用静态方法和实例方法。

Ps：保证类型安全是指它确保在运行时执行的代码不会意外地执行无效的操作，例如访问无效的内存地址或执行无效的类型转换。CLR通过在编译时和运行时检查数据类型来实现这一目标，以防止类型错误。（第一章笔记中有）

如何声明委托：

委托签名的结构：最前面是修饰符，后面是委托的关键字delegate，delegate之后是该委托的回调方法的返回值，括号前是委托方法名，括号中是回调方法的形参信息。



Ps：委托可以定义在命名空间下，也可以定义在类里面。委托属于一个定义，是和类、接口类似的，通常放在外部。

如何使用委托

下面是一个例子：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1.使用new操作符创建一个委托，在括号中加入一个函数的签名。

2.委托的变量的括号里面传指定类型的参数，调用委托中的所有方法

如何使用委托回调静态方法：

下面是例子：



如何使用委托回调实例方法：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Ps：在一个类型中通过委托来调用另一个类型的私有成员，只要委托对象是由具有足够安全性的代码创建的。

例子：

文本

描述已自动生成

将方法绑定到委托时，有协变性和逆变性。协变性是指方法能返回从委托的返回类型派生的一个类型；逆变性是指方法获取的参数可以是委托的参数类型的基类。

即调用的方法的返回类型可以是委托规定的返回类型的子类（协变性，子类拥有父类的所有方法），调用的方法需要传入的参数的类型可以是委托规定的函数形参类型的父类的类型（逆变性，装箱）

下面是例子：

协变：

图形用户界面

中度可信度描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

逆变：

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本

描述已自动生成

委托的运作原理：

文本, 信件

描述已自动生成

编译器实际上是把委托当成一个public类（如果类的访问性是interna和private则为private类），进行编译的，类名即委托的名字，这个类包含四个方法——实例构造器、BeginInvoke，EndInvoke，Invoke。这个构造器获取两个参数，一个object类型的是对象引用，一个是回调方法的序号，也就是委托里的第几个方法（native int和平台即CPU的位数有关）

Ps：

报纸上的文字

描述已自动生成

所有委托都会继承MulticastDelegate类。这是System.MulticastDelegate类里的部分，这几个非公共字段这几个字段很重要

表格

低可信度描述已自动生成

在对象引用（也就是方法）被传给object参数时，标识了一个从MethodDef或MemberDef元数据token获得的特殊IntPtr值，这个IntPtr被传给构造器的method参数（而对于静态方法的object传入null）。Object参数被传给\_target字段，intptr被传给\_methodPtr字段，而invocationList是存\_object参数代表的方法的，也就是多播委托要用到的

书上对于这段话的解释：

对于以下代码



两个不同的Feedback类型实例，一个包装Program的静态方法，一个包装Program类的实例方法。相应的fbStatic包装静态方法，\_target为null；fbinstance包装实例方法，因此\_target为Progtram实例对象维护的一个实例方法。

图示

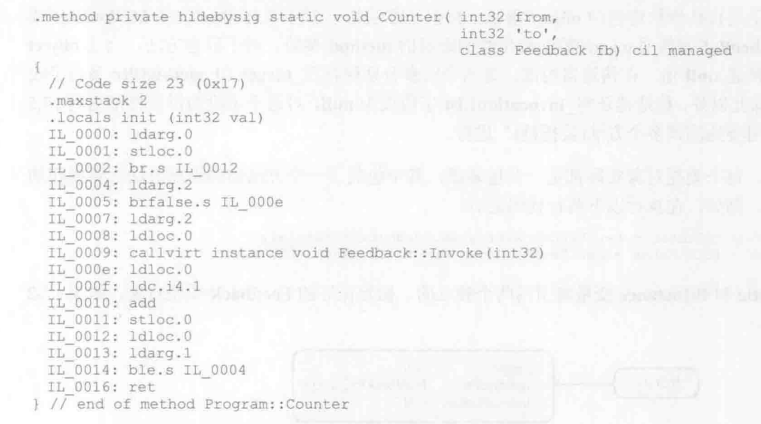
描述已自动生成

委托包装的方法的调用机制类似于本章一开始的代码中的Counter方法：

文本

描述已自动生成

先检查委托的实例是否为null，如果\_invocationList不为null则调用。调用方式分为隐式和显式两种，上图Counter方法的委托实例+括号+括号内指定类型参数，为隐式调用，如果是显式调用则为fb.Invoke(val)，两种方式生成的IL代码完全一致。



IL\_0009为调用Invoke函数的IL代码

文本

描述已自动生成

如何用委托回调多个方法——多播委托（委托链）

对于下面的方法：

文本

描述已自动生成

首先是fb1，fb2，fb3三个Feedback类型委托的实例，其中fb1、fb2包装静态方法，fb3包装实例方法

图形用户界面, 图示, 文本

描述已自动生成

然后声明一个委托类型变量fbChain作为委托集合，通过Delegate.Combine函数把fb1、fb2、fb3添加进委托fbChain的\_invocationList字段中，fbChain作为委托集合自然也不需要\_target和\_methodPtr（初始化成什么样都无所谓），如图

图示

描述已自动生成

最后fbChain的\_inovcationList<Feedback>中会有三个委托，深色部分是加入第三个委托之前的状态，也就是说向委托类型变量中添加委托时并不是在原有的\_invocationList字段中开辟新的空间，而是创建一个含有3个空间\_invocationList的新的委托，原来的图中神色部分的旧委托因此没有被一个变量引用，将被垃圾回收

图示

描述已自动生成

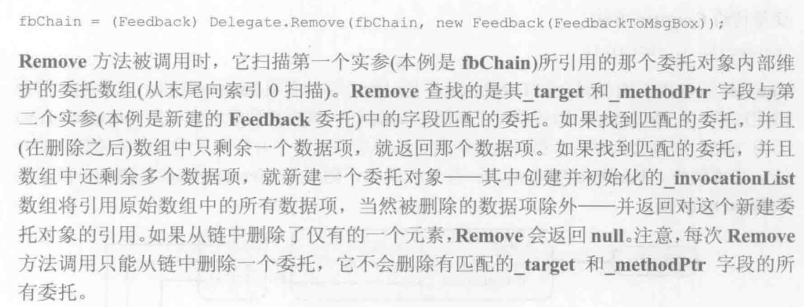
而调用委托链时，该委托对象的实例判断\_invocationList不为null时，就执行一个循环遍历这个数组并 依次调用其中所有的委托

文本, 信件

描述已自动生成

删除\_invocationList中的委托，使用Delegate.Remove方法，查找\_invocationList数组中的末尾元素，然后把它删去

具体解释



Ps：C#为委托实例重载了+和-操作符，添加委托即调用Delegate.Combine时可以使用+或者+=，调用Delegate.Remove时可以使用-或者-=操作符

如果委托调用Invoke是遍历\_invocationList中所有项，那么这个代码是不健壮的，假设调用中间的某一项时抛出了异常，之后的所有数组元素都不会被调用。

因此MulticaseDelegate提供了一个GetInvocationList方法显式调用每一个委托。

GetInvocationList是一个委托类型的方法，它返回一个委托数组，表示当前委托的调用列表。它的函数签名为：public virtual Delegate[] GetInvocationList();

文本

描述已自动生成

下面是例子：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

泛型委托：

这是一个泛型委托的例子：



.Net Framework提供了两个泛型委托，一个Action委托，另一个是Func委托

Action和Func委托的区别：

Action 委托：表示一个没有返回值（void）但可以传入参数的方法，它可以接受 0 到 16 个泛型参数。例如，Action 表示一个没有参数且没有返回值的方法，而 Action<int, string> 表示一个接受两个参数（一个 int 类型和一个 string 类型）且没有返回值的方法。

例子：

文本, 信件

描述已自动生成

Func 委托：表示一个带有返回值（必须不为void）但不可以传入参数的方法，它可以接受 0 到 16 个泛型参数。最后一个泛型参数表示委托的返回类型，其余的泛型参数表示委托的输入参数类型。例如，Func<int> 表示一个没有输入参数且返回值为 int 类型的方法，而 Func<int, string, bool> 表示一个接受两个参数（一个 int 类型和一个 string 类型）且返回值为 bool 类型的方法

例子：

文本

描述已自动生成

如果委托要接收一个变长参数，要为委托的任何参数指定默认值，或者对委托的泛型类型参数进行约束。

C#为委托提供的简化语法：

1.不需要构造委托对象。

即使用+=和-=来向委托链中添加委托对象（对于+=和-=，C#编译器会生成IL代码来新建对应类型的委托对象）

2.不需要定义回调方法

使用lambda匿名函数

例子：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Ps：在传入参数仅有一个时可以不用在参数外加上括号，而传入的参数多于一个时就需要加小括号。匿名函数在编译时会被标记为private和static

3.局部变量不需要手动包装到类中即可传给回调方法

在一个方法中定义一个委托或 lambda 表达式时，如果该委托或 lambda 表达式引用了该方法中的局部变量，那么编译器会自动为您生成一个类来包装这些局部变量。这个过程被称为变量捕获。

例子：

图形用户界面, 文本

描述已自动生成

“<>”开头的哪个类是编译器创建的类，里面包装了局部变量x。

Ps：C#禁止标识符包含”<”符号，然而CLR并不禁止，因此以“<”开头的方法总是C#编译器生成的。但是无论匿名函数还是匿名变量，编译器为它们生成的名字是随机的，这不利于反射调用。

委托和反射

CreateDelegate 是 MethodInfo 类的一个方法，它可以用来从方法信息创建委托实例，该方法允许在编译时不需要知道委托所有必要信息的情况下创建委托。这个方法接受一个委托类型和一个 MethodInfo 对象作为参数，然后返回一个绑定了指定方法的委托实例。这样，您就可以使用反射来动态创建和绑定委托，而无需在编译时确定委托的类型和绑定的方法。

CreateDelegate方法的签名以及重载：

图片包含 文本

描述已自动生成

Object target指定作为this参数传给实例方法的对象（方法的实例）

图片包含 图示

描述已自动生成

例子：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Ps：DynamicInvoke和Invoke的区别

DynamicInvoke 是一个非泛型方法，它接受一个 object[] 类型的参数数组，并返回一个 object 类型的值。当使用 DynamicInvoke 调用委托时，需要将参数封装到一个 object[] 数组中，并在调用后将返回值从 object 类型转换回实际类型。这会带来一些性能损失，因为需要进行装箱和拆箱操作。

Invoke 是一个泛型方法，它的参数类型和返回类型与委托的签名相同。当使用 Invoke 调用委托时，可以直接传递参数，并直接获取返回值，无需进行装箱和拆箱操作。这样可以提高性能。