**委托：delegate**

**委托的理解：**

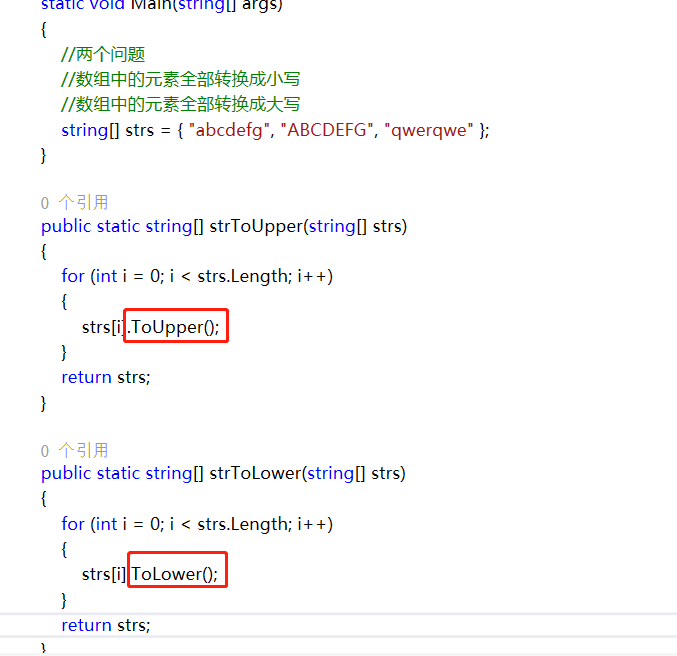
1. 委托，就是一种**函数的类型**，以计算机原理来讲，它是一种**安全的函数指针**，以面向对象编程设计的思想来讲，它是一种不安全的引用类型(因为委托能够在命名空间下声明，所以客户端可以对它进行随意的赋值等操作，严重破坏对象的封装性)
2. 当一个委托指向某个方法的时候，**那么调用委托的对象就是在调用该委托所指向的方法**
3. 相同类型的委托/事件的实例 可以直接赋值给另一个 相同类型的委托/事件的实例，当赋值成功后，被赋值的 委托/事件的实例 指向赋值的 委托/事件的实例 中所指向的函数（引用类型，传递地址）
4. 因为方法中的参数都是拥有类型的，当我们要在一个方法中的参数传另一个方法的时候，我们却不知道需要传参的方法是什么类型，这时候我们就能申明一个签名与需要传参方法的签名相同的委托，并让委托指向一个方法，我们在参数中写委托的类形与对象，这样就能达到在传参中传一个方法的目的，也可以理解为，**被指向的方法的类型是委托类型**

**委托的注意事项：**

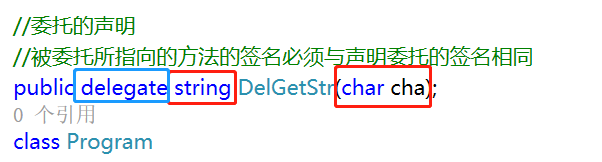
1. 一个方法可以直接赋值给委托对象，使委托对象指向该方法，前提是两者的**签名**（方法的参数、返回值）必须相同

**什么时候使用委托和如何使用委托来解决问题：**

1. 当我们在编程时侯针对某个问题写了多个方法，但是我们发现这三个方法都有相似的地方，这时候我们就能通过声明委托来减少代码，如下图所示：我们可以看到，画红框的部分是不同的，其余全部是相同的，那我们可以考虑通过委托来减少代码的写入



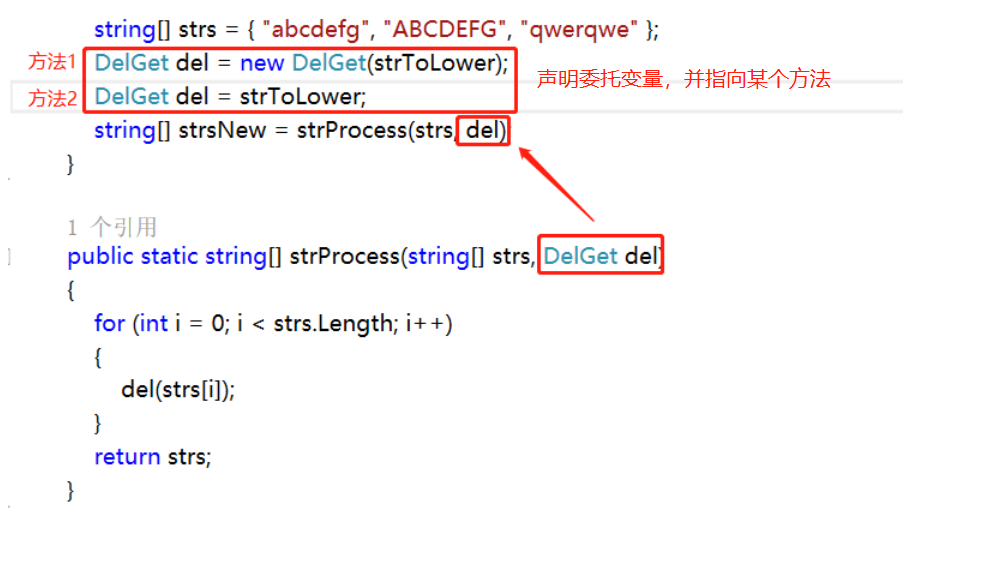
1. 我们先查找不同方法之间的特性（参数和返回值），然后我们声明一个委托：



1. 然后通过在一个方法中写委托的类型和对象，并在其他地方对该对象赋值所需要指向的方法，这样就能减少代码，并且能实现了在方法中通过传参的方式调用另一个方法



1. 我们也能通过这种手段来让委托指向一个方法，与上图一样的问题：



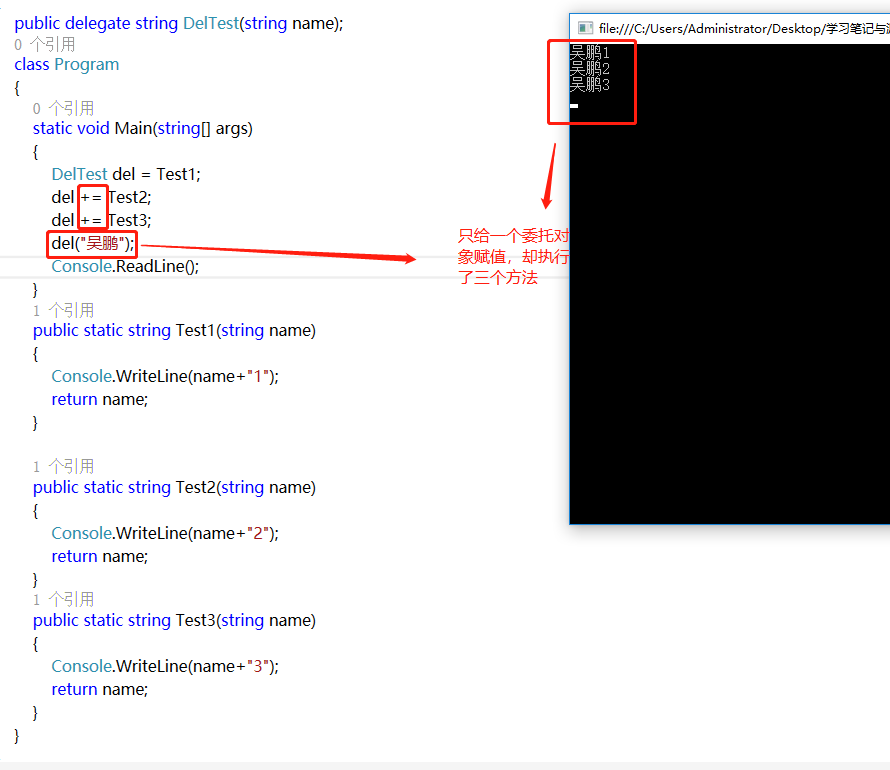
**多播委托的使用：**

**多播委托的详解：**

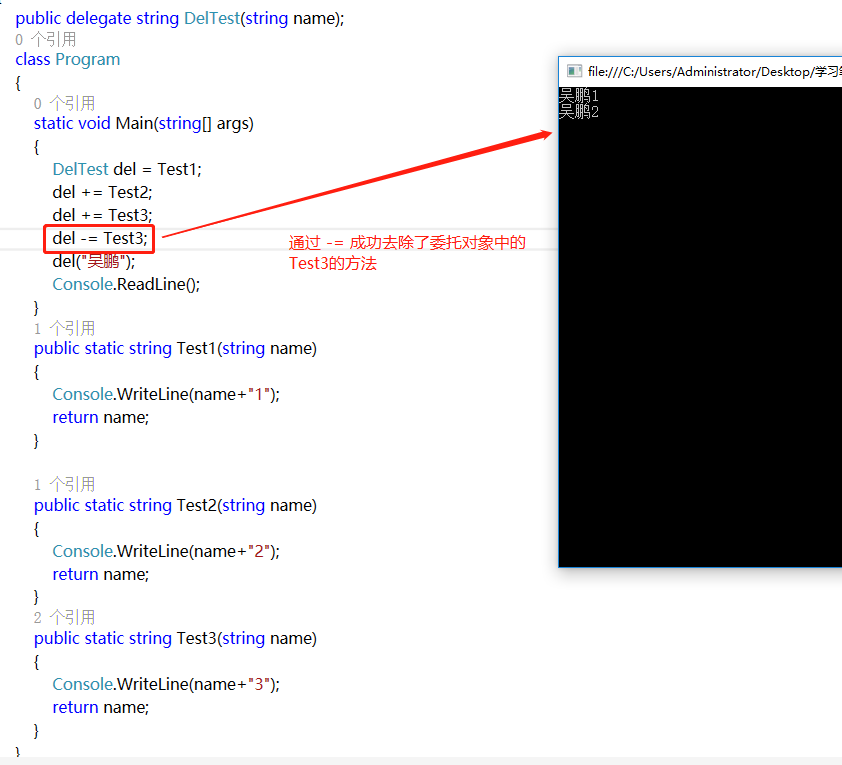
一个委托能够同时指向多个与委托签名相同的方法，并只用一个委托对象来执行这些方法，这就是多播委托

**多播委托的语法：**

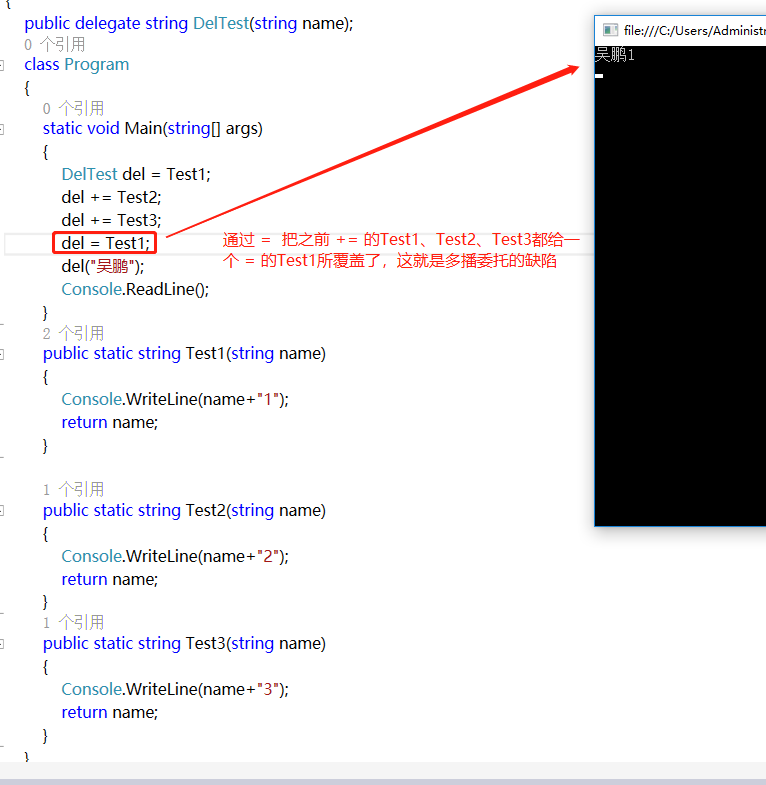
我们通过 **委托对象 “+=” 方法** 来把需要给委托指向的方法给委托，并且通过该对象能够同时执行我们给委托 “+=” 的方法，如下图所示，我们只通过一个委托对象，却执行了三个方法：



反之，当我们想把某个方法从委托对象中去除，我们可以通过 **委托对象“-=”需要被减去的方法**，来去除某个存在于委托对象中的方法，如下图所示：



但是多播委托有一个不好的地方，当我们在 += 给委托对象赋值方法后，再 = 一次委托对象，那么多播委托之前所 += 的方法都会给我们 = 的方法所替代掉，如下图所示：



**委托实现同步调用、异步调用、异步回调：**

**简介：**

1. .NET可以实现让我们 异步或同步 的去调用任何函数，为实现这种机制，我们需要定义一个具有与调用函数相同签名的一个委托，并通过这个委托的 invoke、begininvoke/endinvoke方法 去实现 异步或同步 的调用指定的函数
2. 用委托来实现异步调用的时候，.NET底层是在线程池中取出一个线程来实现异步调用，由此可得出，实现异步的线程的线程优先级都为【Normal】，并且是【后台线程】

**异步/同步调用所使用的函数和返回的类的详解：**

1. object result delegate.invoke ( MethodParameter )：Invoke方法的主要功能就是帮助我们在UI线程上调用委托所指定的方法，当调用此方法的时候，会优先检测调用invoke()函数的是否是UI线程，如果是则直接执行委托所指向的函数，如果不是，则切换至UI线程去执行委托所指向的函数，并阻塞当前调用invoke()函数的线程，直至函数执行完毕则恢复线程的继续使用

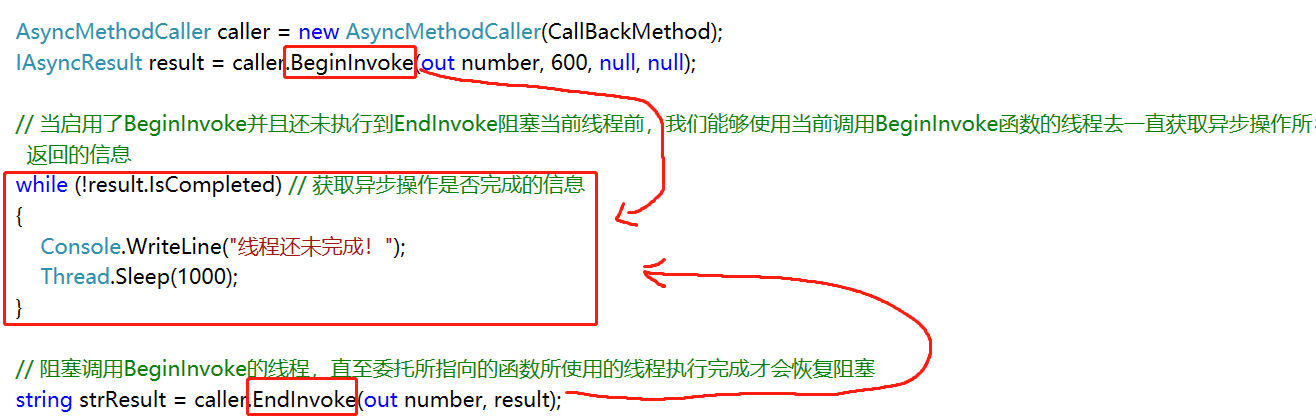
-> object result：委托所指向的函数的返回值

-> MethodParameter：委托所指向函数的所需要的参数，支持ref/out参数

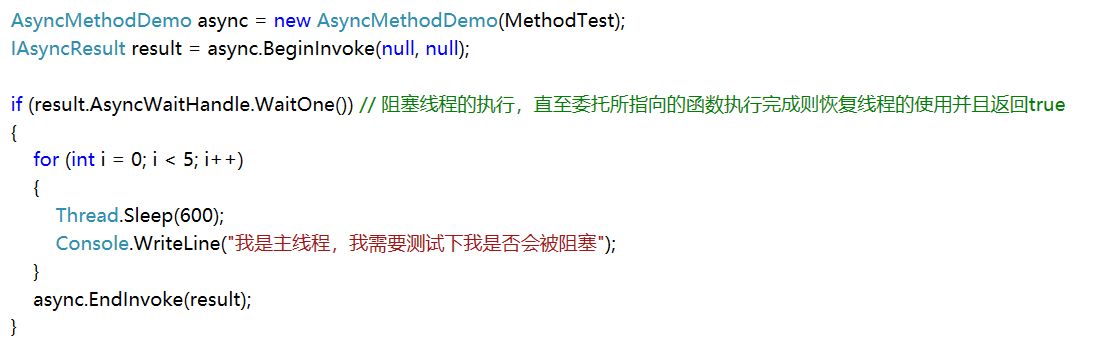
1. IAsnycResult result delegate.BeginInvoke ( MethodParameters, AsnycCallBackMethod , Object AsnycState )：开启一个异步请求，并从线程池中取出一个线程去执行委托所指向的函数，可以在MethodParameters中指定委托所指向的函数所需的参数

-> IAsnycResult result：一个接口，BeginInvoke函数能够返回该接口的实现，**它是异步的核心**，该接口存储了异步操作的一些状态信息，并且提供了一些函数来操作和阻塞线程，如下所示：

1. IsCompleted属性：因为BeginInvoke至EndInvoke之间的代码不会发生阻塞的机制，我们能够使用该属性去不断的获取异步操作是否已完成



1. AsyncWaitHandle.WaitOne()函数：该函数会阻塞当前线程的执行（区别于IsCompleted属性），直至委托所指向的函数执行完成则恢复线程的使用，并且返回true



-> MethodParameters：委托所指向的函数所需的参数，支持ref/out参数

-> AsnycCallBackMethod：线程执行完成委托所指向的函数所执行的回调函数，如不需要使用异步回调的编程模式，则可以赋值为null**（回调函数的参数必须为 IAsnycResult result，因为当线程执行完委托所指向的函数后，系统除了在BeginInvoke中返回IAsnycResult result的实现外，还会把IAsnycResult result的实现也一并赋值给我们所指定的回调函数，然后再执行回调函数）**

-> Object AsnycState：获取用户定义的对象，它限定或包含关于异步操作的信息，如不需要可以赋值为null

1. object MethodResult delegate.EndInvoke ( MethodParametersForOutOrRef, IAsnycResult result )： 阻塞当前调用BeginInvoke函数的线程，直至它的完成才恢复该线程的使用

-> object MethodResult：委托所指向的函数的返回值

-> MethodParametersForOutOrRed：当委托所指向的参数存在 out/ref 的时候，我们需要在这里来接收 out/ref参数 的返回

-> IAsnycResult result：该参数必须接收delegate.BeginInvoke函数所返回的IAsnycResult result才能正常的去阻塞其他线程的执行

**异步/同步调用的一些实例demo：**

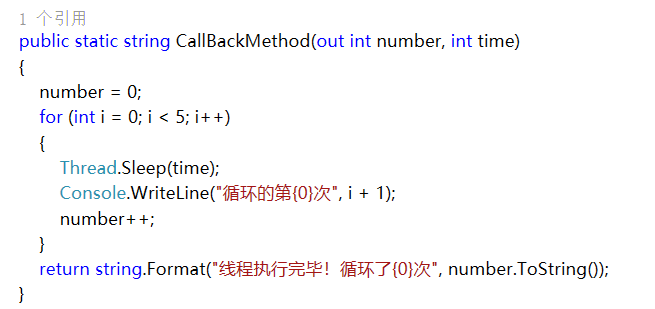
1. 同步调用：使用委托的invoke方法来实现同步调用，其实先机制其实是当代码运行到invoke函数那一块的时候会判断当前调用invoke函数的线程是否为UI线程，如果是则直接调用，如果不是则切换至UI线程并完成该函数的调用，并且会阻塞当前线程的运行直至委托所指向的函数执行完成。



使用同步机制虽能够处理一些针对的问题解决不同步调用带来对数据或显示结果出现错误的问题，但是由于会造成阻塞，所以一些问题还是需要使用异步机制来解决

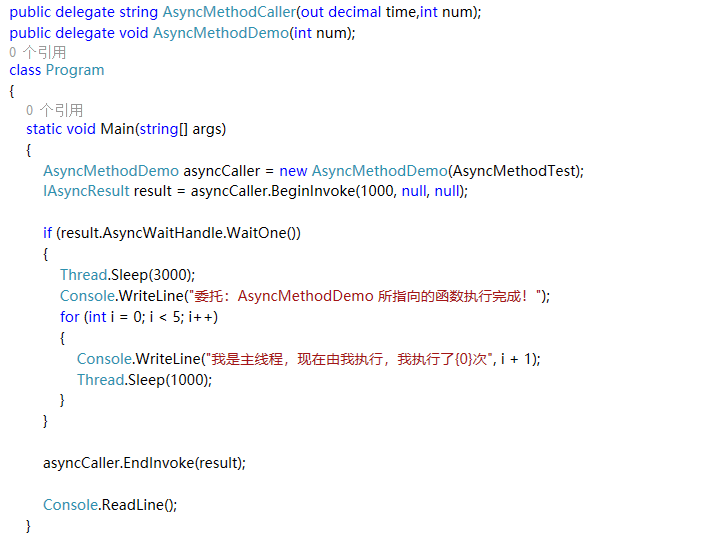
1. 异步调用：使用委托的BeginInvoke函数和EndInvoke函数来实现异步调用（**BeginInvoke和EndInvoke必须联合使用**），其原理则是代码运行至BeginInvoke函数的时候，会从线程池中抽取一个线程来完成委托所指向的函数的调用，直至代码运行到EndInvoke函数的时候，如果线程的调用还未结束，则EndInvoke函数就会阻塞当前调用者线程的继续使用直至委托所指向的函数调用完成，也就是说，BeginInvoke函数至EndInvoke函数之前的代码块的运行是不会发生线程阻塞的，这时候我们就能够利用BeginInvoke函数所返回的IAsyncResult接口（异步核心）中的IsCompleted属性来判断当前的异步委托是否是否执行完成（该属性能够不断的返回异步委托的执行结果），或者利用AsyncWaitHandle属性的WaitOne()函数来阻塞当前线程直至异步调用结束（区别于IsCompleted属性，IsCompleted属性是能够不断地返回异步执行结果的，而WaitOne()函数则会阻塞线程的使用，直至异步调用结束才会恢复线程的使用并且返回true）





但是使用这种方式还是有个缺陷，因为代码执行至EndInvoke的时候还是会发生线程的阻塞、等待的问题，所以我们只有在BeginInvoke至EndInvoke之间写一些处理代码才能够不让线程阻塞，但是我么利用异步回调就能够很完美的解决这个缺陷

1. 异步回调：当我们使用委托的BeginInvoke函数的时候，我们能够指定一些参数，第一个参数为委托所指向的函数所需的参数，第二个参数就是回调函数，第三个参数便是我们所指定的异步调用的一些信息，当我们指定了第二个参数的回调函数后，我们就能够使用异步回调了，这个回调函数会在BeginInvoke后、委托所指向的函数执行完成，才会去调用回调函数，需要注意的是，回调函数的参数我们必须设置为：IAsyncResult result，其原理就是当我们指定了第三个参数后，.NET除了会把IAsyncResult result的实现给BeginInvoke函数的返回值，还会把IAsyncResult result的实现赋值给我们所指定的回调函数，**所以BeginInvoke所返回的IAsyncResult result的实现，和回调函数中的IAsyncResult result参数其实是一样的值**，通过这种机制，我们就能把IAsyncResult result的实现转换成BeginInvoke所使用的委托变量，并通过转换后的委托变量调用EndInvoke函数并赋值该函数的参数为回调函数中的参数IAsyncResult result的实现，通过这种方式我们就实现了 异步的回调、委托的BeginInvoke和EndInvoke都调用完成，当使用了异步回调后，就不会造成线程阻塞的局面





详情可以参考以下网址：

1. <https://www.cnblogs.com/arxive/p/6528915.html>
2. <https://www.cnblogs.com/wolf-sun/p/5675791.html>
3. <https://www.cnblogs.com/SuperMetalMax/p/6211113.html>