黄蜡带

产品名称	黄蜡带
公司名称	厦门日华机电成套有限公司
价格	.00/个
规格参数	
公司地址	福建厦门火炬高新技术开发区新丰2路8号日华大厦三楼AB单元
联系电话	0592-5701778-1029

产品详情

黄蜡带编程过程中使用委派使得在第二个线程中对某个方法进行异步调用变的非常简单。这意味着你可以在一个 Windows 窗体程序中调用一个耗时很长的网络调用方法而不必使用户界面失去响应。 异步调用对服务器端程序也是一个很有用的方法。假设你正黄蜡带在编写一个提供用户黄蜡带服务的ASP.NET页面的代码，当你的代码需要在一条网络上处理两个或更黄蜡带多的耗时调用，你将如何处理？举个例子，你写的代码要对远程数据进行查询并调用远程网络服务的SOAP方法来完成工作。如果你不使用异步调用，在同一时间内你只能在你的那一条网络上调用一个方法，这意味着在第一个网络调用结束并返回之前你不能启动第二个网络调用。通过委派进行异步调用使你能够在同一时间内进行两个或更多的网络调用。这将极大的减少服务端ASP.NET页面处理用户要求的时间，提高了你的程序对用户的响应能力。当Visual Basic .Net的编译器生成一个委派类型的定义时，它添加Invoke方法以支持同步调用。编译器还添加了两个方法：BiginInvoke和EndInvoke，以支持异步调用。 让我们修改上面的代码，以便通过委派对象以异步模式调用GetCustomerList方法。不再调用Invoke方法，取而代之的是BeginInvoke方法，BeginInvoke方法需要你传递在委派类型对象中定义的所有参数。在下面的例子中，参数是一个表示美国州名的string类型的参数，值为"CA"。 ''*** create delegate object and bind to target method Dim handler1 As GetCustomerListHandler handler1 = AddressOf DataAccessCode.GetCustomerList ''*** execute method asynchronously handler1.BeginInvoke("CA", Nothing, Nothing)
与Invoke方法相比，BeginInvoke需要额外的两个参数，我将在这篇文章中详细解释。 当你调用一个委派对象的BeginInvoke方法时，本质上是向公共语言运行时（CLR）提出请求，以异步方式，也就是在在第二个线程中执行绑定的方法。通过BeginInvoke方法实现异步调用是非常有效的，该方法非常简单，因为你不必手黄蜡带工建立并管理第二个线黄蜡带程，CLR会自动为你完成这些工作。 Figure 1显示了CLR处理处理BeginInvoke调用。当你调用BeginInvoke方法时，委派对象向一个特别的内部队列发送请求，而CLR维护着一个响应该队列中请求的工作者线程池。异步调用之所以能够实现，就在于调用BeginInvoke方法的线程和执行绑定方法的线程是不同的。 Figure 1 调用异步方法 CLR通过检查一系列因素来动态地决定工作者线程池中的线程数量。在一个典型的桌面应用程序中，受CLR限制，池中的线程数目屈指可数。在一个繁忙的服务器端程序中，由于后台代码中含有很多对BeginInvoke的并发调用，CLR将池中线程数目大幅提高，最大数目为每个处理器25个线程。也就是说池中线程数目与处理器数目按一定比例变化，对于拥有4个处理器的计算机，CLR维护的工作者线程池将拥有100个线程。 下面来思考一下调用Invoke和调用BeginInvoke之间的不同。假设你想洗涤积累了一个月的脏衣服，你将有两个选择：去自助洗衣房自己完成，或把它们交给洗衣房黄蜡带让别人为你完成。 调用Invoke就像去自助洗衣房并自己完成一样

，是一个同步操作。在工作完成之间，你必须呆在自助洗衣房里。当你离开时，你会带着你完成的工作：洗干净的衣服。 调用BeginInvoke就像把脏衣服送到洗衣店让别人完成，是一个异步操作。当你把脏衣服送到洗衣店，你就可以离开并立即投入到其他工作中。但是，与自助洗衣店相比，当你离开时，你无法拿着那些洗干净的衣服。你会有一张洗衣店的票据，过一段时间后，你必须返回洗衣店领取你的已经洗干净的衣服。而且，你必须保管好票据，因为如果你丢失了，洗衣店的人可能会拒绝把衣服还给你。当你调用BeginInvoke时，你必须将一个调用某方法（上面代码中的GetCustomerList方法）的请求（脏衣服）提交给CLR（洗衣店）。对BeginInvoke的调用会立刻返回，因此，调用BeginInvoke的线程可以执行其他操作，而不必等待CLR执行GetCustomerList方法。当BeginInvoke返回后，你可以确定的是CLR会在第二个线程（相对调用BeginInvoke的线程而言）中调用GetCustomerList方法，但你不能确切的知道何时才会执行。过一段时间之后，你可以向CLR询问这个异步调用的方法是否已经完成，也可以得到这个方法的返回值。但是，对于异步调用的方法，不论是询问其状态，还是获得其返黄蜡带回值，都必须向CLR提供你提出黄蜡带异步调用请求时的信息。 IAsyncResult BeginInvoke方法的返回值与洗衣店提供的票据十分相似。实际上，BeginInvoke方法的返回值是一个实现了IAsyncResult接口的对象。下面是一个调用委派对象的BeginInvoke方法并返回IAsyncResult对象的例子： ''*** create delegate object and bind to target method Dim handler1 As GetCustomerListHandler handler1 = AddressOf DataAccessCode.GetCustomerList ''*** execute method asynchronously and capture IAsyncResult object Dim ar As System.IAsyncResult ar = handler1.BeginInvoke("CA", Nothing, Nothing) IAsyncResult对象使用户能够监视一个异步调用的进度。通过IAsyncResult对象，当异步方法运行结束时，用户还能够获得方法的返回值和所有的输出参数。 委派对象与返回的IAsyncResult对象之间并不是一对一关系！明白这黄蜡带一点非常重要。例如，一个委派对象可以被用来同时启动2个或更多的异步调用： ''*** create delegate object and bind to target method Dim handler1 As GetCustomerListHandler handler1 = AddressOf DataAccessCode.GetCustomerList ''*** execute multiple methods asynchronously Dim ar1, ar2 As System.IAsyncResult ar1 = handler1.BeginInvoke("CA", Nothing, Nothing) ar2 = handler1.BeginInvoke("WA", Nothing, Nothing) 在这个例子中，两个不同的异步调用被同时启动。针对每一个异步调用，CLR都从工作者线程池中分配一个独立的线程，异步调用即在此线程中运行，见Figure 2。 Figure 2 两个异步调用 通过该方法，你可以在网络上进行2个或更多的并发调用。但是，需要牢记的是：每一次的异步调用都有其独自的IAsyncResult对象。 IAsyncResult接口有一个名为IsComplete的属性，允许你监测异步调用是否运行完毕。尽管轮询在很多情况下效率低下，但在少数情况下也是有用的。下面的代码演示了如何使用IsComplete属性来判断一个异步调用是否运行完毕： Dim ar As IAsyncResult ar = handler1.Begin黄蜡带Invoke("CA", Nothing, Nothing) ''*** allow some time to pass ''*** check to see if the async call has completed If (ar.IsCompleted) Then ''*** retrieve method return value End If 调用EndInvoke方法 通过委派对象的EndInvoke方法，你可以获得返回值和输出参数。EndInvoke是一个和委派类型对象具有相同返回值类型的方法： Dim ar As IAsyncResult ar = handler1.BeginInvoke("CA", Nothing, Nothing) ''*** allow some time to pass Dim retval As String() retval = handler1.EndInvoke(ar) 从上面的代码中可以看出：通过EndInvoke方法可以获得一个异步调用的返回值。需要注意的是，EndInvoke方法要求你提供该异步调用的IAsyncResult对象。这就像当你到洗衣店取干净衣服时，需要向店员提供你的票据。IAsyncResult对象使CLR可以决定哪一个异步调用是你想要的。 还有一点值得注意，EndInvoke方法的参数列表中还可以包括定义委派类型时以ByRef方式调用的参数。但是我采用的例子（GetCustomerList）不包括任何输出参数，所以调用EndInvoke方法只需要你提供一个实现了IAsyncResult接口的对象。 除了获得返回值和输出参数，EndInvoke方法还可以让你知道异步调用是否成功。当异步调用的方法抛出一个自身不能处理的异常时，这个异常对象会被CLR截获，当你调用EndInvoke方法时，该异常对象会被抛出。如下代码： Dim retval As String() Try retval = handler1.EndInvoke(ar) Catch ex As Exception ''*** deal with exception here End Try 将BeginInvoke和EndInvoke方法成对调用是非常重要的。如果你没有调用EndInvoke，你就不能确切知道异步调用是否运行成功。忘记调用EndInvoke将会导致不引人注意的运行时异常，会使你的代码中具有bugs，并且难以跟踪黄蜡带和修复。 上面几个理由充分说明了调用BeginInvoke就要调用EndInvoke黄蜡带。这里还有一个必须调用EndInvoke的更重要的理由：这是规则。忘记调用EndInvoke将导致CLR无法释放为异步调用分配的资源。因此，当你在你的程序中调用了BeginInvoke而没有相应地调用EndInvoke时，你的程序可能会有资源泄漏。 下面让我们看一下调用EndInvoke时程序是如何工作的。如果异步调用，即第二线程，已经完成，则EndInvoke立即返回。但是，如果异步调用还没有启动，或者仍然在运行，对EndInvoke的调用就会阻塞。当你设计程序时，你要考虑到这种情况的后果。 设想一下这种情况：你编写的ASP.Net页面的后台代码需要同时进行两个网络调用，EndInvoke的阻塞方式使得对某一方法的多种参数调用之间的和谐相处变得灵巧而简单。代码Figure 3中同时进行两个异步调用来产生两个并发的网

络调用，前提条件是在请求作为一个整体被完成之前，两个网络调用都要成功完成。在第一个异步调用完成之前，对EndInvoke的第一次调用一直处在阻塞状态。在第二个异步调用完成之前，对EndInvoke的第二次调用也一直黄蜡带处在阻塞状态。 如果第二个异步调用在第一个异步调用之前完成，会发生什么情况呢？实际上这种考虑并不重要。在代码Figure 3中，两个网络调用中哪个先完成是无关紧要的。在服务器端的请求完成其工作之前，对GetCustomerList的两次调用必须完成。因此，EndInvoke的阻塞方式发挥了重要规则。它使对一个方法的多种参数调用合并成一个逻辑上的调用。 与此同时，CLR在后台完成了所有复杂的工作。包括管理线程，发送异步调用，在其他线程完成之前阻塞某些线程。所有的这些工作，你只需要调用BeginInvoke和EndInvoke即可解决。现在，你可以清楚的看到委派对象针对异步调用执行提供了一个简单，强大的抽象。 现在让我们把注意力转到在一个桌面程序中的Windows窗体上，与服务端程序相比，这是一个区别很大的情况。因为在桌面程序中，通常所有的代码都在主用户界面线程中运行。如果你调用一个耗时很长的网络操作，用户界面线程就会冻结，这个程序就会对用户操作失去响应，直到网络操作返回。 在这种情况下，使用异步调用是非常必要的，它使你不必冻结用户界面。你已经看到通过调用BeginInvoke方法产生一个异步调用是非常简单的事情，但一个问题出现了：当异步调用完成时，你的程序如何作出反应？当一个异步调用完成时，你的程序如何更新用户界面？当然，你可以通过轮询技术来循环调用IsComplete方法，但这是非常低效的。作为轮询技术的取代，你可以充分利用委派的一个便利的特点：设置一个回调方法，当异步调用完成时，CLR会自动调用该回调方法。 代码Figure 4演示了在一个Windows窗体程序中异步调用GetCustomerList方法。 所有的代码都被包括在一个名为Form1的类中。在这个类的定义中，你可以看到通过调用BeginInvoke来实现异步调用，并对该异步调用设置一个回调方法。 让我们仔细研究上面的代码，当你想设置一个回调方法时，你必须通过AsyncCallBack类型的委派类型来实现，AsyncCallBack的定义在Framework Class Library的System命名空间中。当你想创建一个回调方法时，你的回调方法的声明必须与AsyncCallBack委托声明具有相同的参数，上面代码中的MyCallbackMethod方法就是一个具有正确声明的方法。从代码中可以黄蜡带看到，回调方黄蜡带法必须被声明为Sub，并且只有一个IAsyncResult类型的参数。 你会发现类Form1具有两个变量，都是对委派对象的引用。第一个变量TargetHandler指向异步调用GetCustomerList方法的委派对象，第二个变量CallbackHandler指向绑定MyCallbackMethod方法的委派对象。这些委派对象在调用BeginInvoke时都被使用。 下面来看一下调用BeginInvoke的事件处理方法cmdExecuteTask。变量TargetHandler作为BeginInvoke的参数来启动异步调用，BeginInvoke的第二个参数是绑定了MyCallbackMethod方法的委派对象的引用。也就是说，你向BeginInvoke方法传递一个委派对象的引用，使CLR知道你想使用哪一个回调方法。这些就是在异步方法调用之后设置回调方法所需要的全部工作。 下面让我们看一下黄蜡带Figure 4中当异步调用请求被发送之后代码是如何工作的。当应用程序调用BeginInvoke的时候，CLR就从其线程池中分配一个工作者线程，并在这个工作者线程中运行要调用的方法。然后，在这个线程中运行回调方法。当这些工作完成之后，CLR将这个工作者线程释放到线程池中以便响应其他的异步调用请求。 回调方法并不在调用BeginInvoke方法的用户主界面线程中运行，明白这一点非常重要。在此强调一下，回调方法和异步调用的方法都运行在同一个线程中。 传递给BeginInvoke的最后一个参数是AsyncState。AsyncState是一个非常灵活的参数，你可以用它来传递任何东西。它被定义成Object的形式以传递任何值和对象。在我提供的Figure 4例子中，没有任何必要使用AsyncState参数，所以传递Nothing值。 在其他的设计中，通过AsyncState参数，可以在调用BeginInvoke的时候很轻易的将值或对象传递给回调方法。例如，假设你的回调方法需要一个Integer类型的值，当你调用BeginInvoke时，将Integer类型的值作为AsyncState参数即可： Dim MyValue As Integer = 100 TargetHandler.BeginInvoke("CA", CallbackHandler, MyValue) 作为AsyncState传递的值或对象都可以通过IAsyncResult接口的AsyncState属性获得。下面是使用Integer类型值的回调方法例子： Sub MyCallbackMethod(ByVal ar As IAsyncResult) Dim y As Integer = CInt(ar.AsyncState) ''*** other code omitted for brevity End Sub 一定要记住AsyncState被定义为Object形式，所以在你对AsyncState容纳的值或对象进行操作之前，一定要将它显式转换成更详细的类型。 从上面的讲解中你可以看到，在窗体的代码中通过BeginInvoke进行异步调用是非常简单的。你还学到了如何设定当异步调用方法完成时自动调用的回调事件。你需要学习的最后一项内容是如何更新用户界面使用户知道异步调用已经完成了。这是有相当难度的工作，写出正确的代码是比较困难的，如果写错了，将导致严重的错误。 编写基于窗体的程序时一个要遵守的重要的线程规则就是：只有主用户界面线程才能拥有窗体对象以及窗体上的所有子控件。也就是说，在第二个线程中访问窗体及其控件的方法或属性是非法的。而且你已经知道，回调方法，例如Figure 4中的MyCallbackMethod方法，运行在第二个线程中而不是主用户界面线程。这意味着你绝对不能在回调方法中直接更新用户界面。 简单的说，运行在第二个线程中的回调方法必黄蜡带须强迫主用户界面线程调用它的更新用户界面的代码。我将在下个月的专栏中详细解释如何操作。目前只需要明白在回调方法中

直接更新用户界面是不可取的编程技巧，因为Framework（框架）不允许这样做。 结论 通过委派进行异步调用时，每一种委派类型都提供了一个BeginInvoke方法来启动一个异步调用，异步调用在第二个线程中运行，而线程来自由CLR维护的线程池。每一种委派类型提供响应的EndInvoke方法，通过EndInvoke方法可以获得异步调用方法的返回值，也可以检查异步调用方法执行过程中是否抛出了未处理的错误。 通过委派进行异步调用最有价值的特点就是你不必在创建和管理第二线程上花费心思。CLR维护了一个工作者线程池，这种不必直接编写代码来处理线程就能够进行异步和并发调用的技术使你获益非浅。唯一需要学习的就是如何在恰当的时候调用BeginInvoke和EndInvoke方法。