C#异步调用

让我们首先了解下什么时候用到C#异步调用：

.NET Framework 允许您C#异步调用任何方法。定义与您需要调用的方法具有相同签名的委托；公共语言运行库将自动为该委托定义具有适当签名的 BeginInvoke 和 EndInvoke 方法。BeginInvoke 方法用于启动C#异步调用。它与您需要异步执行的方法具有相同的参数，只不过还有两个额外的参数（将在稍后描述）。BeginInvoke 立即返回，不等待C#异步调用完成。BeginInvoke 返回 IasyncResult，可用于监视调用进度。

EndInvoke 方法用于检索C#异步调用结果。调用 BeginInvoke 后可随时调用 EndInvoke 方法；如果C#异步调用未完成，EndInvoke 将一直阻塞到C#异步调用完成。EndInvoke 的参数包括您需要异步执行的方法的 out 和 ref 参数（在 Visual Basic 中为 ByRef 和 By Ref）以及由 BeginInvoke 返回的 IAsyncResult。

注意　Visual Studio .NET 中的智能感知功能会显示 BeginInvoke 和 EndInvoke 的参数。如果您没有使用 Visual Studio 或类似的工具，或者您使用的是 C# 和 Visual Studio .NET，请参见异步方法签名获取有关运行库为这些方法定义的参数的描述。

本主题中的代码演示了四种使用 BeginInvoke 和 EndInvoke 进行C#异步调用的常用方法。调用了 BeginInvoke 后，可以：

· 进行某些操作，然后调用 EndInvoke 一直阻塞到调用完成。

· 使用 IAsyncResult.AsyncWaitHandle 获取 WaitHandle，使用它的 WaitOne 方法将执行一直阻塞到发出 WaitHandle 信号，然后调用 EndInvoke。

· 轮询由 BeginInvoke 返回的 IAsyncResult，确定C#异步调用何时完成，然后调用 EndInvoke。

· 将用于回调方法的委托传递给 BeginInvoke。该方法在C#异步调用完成后在 ThreadPool 线程上执行，它可以调用 EndInvoke。

警告：始终在C#异步调用完成后调用 EndInvoke。

测试方法和异步委托

四个示例全部使用同一个长期运行的测试方法 TestMethod。该方法显示一个表明它已开始处理的控制台信息，休眠几秒钟，然后结束。TestMethod 有一个 out 参数（在 Visual Basic 中为 ByRef），它演示了如何将这些参数添加到 BeginInvoke 和 EndInvoke 的签名中。

您可以用类似的方式处理 ref 参数（在 Visual Basic 中为 ByRef）。

下面的代码示例显示 TestMethod 以及代表 TestMethod 的委托；若要使用任一示例，请将示例代码追加到这段代码中。

注意　为了简化这些示例，TestMethod 在独立于 Main() 的类中声明。或者，TestMethod 可以是包含 Main() 的同一类中的 static 方法（在 Visual Basic 中为 Shared）。

```
using System;
using System.Threading;

public class AsyncDemo
{
   // The method to be executed asynchronously.
   //
   public string TestMethod(
   int callDuration, out int threadId)
   {
      Console.WriteLine("Test method begins.");
      Thread.Sleep(callDuration);
      threadId = AppDomain.GetCurrentThreadId();
      return "MyCallTime was " + callDuration.ToString();
   }
}

// The delegate must have the same signature as the method
// you want to call asynchronously.
public delegate string AsyncDelegate(int callDuration, out int threadId);
```

C#异步调用四大方法之使用 EndInvoke 等待异步调用

异步执行方法的最简单方式是以 BeginInvoke 开始，对主线程执行一些操作，然后调用 End

Invoke。EndInvoke 直到 C#异步调用完成后才返回。这种技术非常适合文件或网络操作，但是由于它阻塞 EndInvoke，所以不要从用户界面的服务线程中使用它。

```
public class AsyncMain
{
  static void Main(string[] args)
  {
    // The asynchronous method puts the thread id here.
    int threadId;

    // Create an instance of the test class.
    AsyncDemo ad = new AsyncDemo();

    // Create the delegate.
    AsyncDelegate dlgt = new AsyncDelegate(ad.TestMethod);

    // Initiate the asychronous call.
    IAsyncResult ar = dlgt.BeginInvoke(3000,
    out threadId, null, null);

    Thread.Sleep(0);
    Console.WriteLine("Main thread {0} does some work.",
    AppDomain.GetCurrentThreadId());

    // Call EndInvoke to Wait for
    //the asynchronous call to complete,
    // and to retrieve the results.
    string ret = dlgt.EndInvoke(out threadId, ar);

    Console.WriteLine("The call executed on thread {0},with return value \"{1}\".", threadId, ret);
```

```
  }
}
```

C#异步调用四大方法之使用 WaitHandle 等待异步调用

等待 WaitHandle 是一项常用的线程同步技术。您可以使用由 BeginInvoke 返回的 IAsyncResult 的 AsyncWaitHandle 属性来获取 WaitHandle。C#异步调用完成时会发出 WaitHandle 信号，而您可以通过调用它的 WaitOne 等待它。

如果您使用 WaitHandle，则在 C#异步调用完成之后，但在通过调用 EndInvoke 检索结果之前，可以执行其他处理。

```
public class AsyncMain
{
   static void Main(string[] args)
   {
      // The asynchronous method puts the thread id here.
      int threadId;

      // Create an instance of the test class.
      AsyncDemo ad = new AsyncDemo();

      // Create the delegate.
      AsyncDelegate dlgt = new AsyncDelegate(ad.TestMethod);

      // Initiate the asychronous call.
      IAsyncResult ar = dlgt.BeginInvoke(3000,
      out threadId, null, null);

      Thread.Sleep(0);
      Console.WriteLine("Main thread {0} does some work.",
      AppDomain.GetCurrentThreadId());
```

```
        // Wait for the WaitHandle to become signaled.
        ar.AsyncWaitHandle.WaitOne();

        // Perform additional processing here.
        // Call EndInvoke to retrieve the results.
        string ret = dlgt.EndInvoke(out threadId, ar);

        Console.WriteLine("The call executed on thread {0}, with return value \"{1}\".", threadId, ret);
    }
}
```

C#异步调用四大方法之轮询异步调用完成

您可以使用由 BeginInvoke 返回的 IAsyncResult 的 IsCompleted 属性来发现 C#异步调用何时完成。从用户界面的服务线程中进行 C#异步调用时可以执行此操作。轮询完成允许用户界面线程继续处理用户输入。

```
public class AsyncMain
{
    static void Main(string[] args)
    {
        // The asynchronous method puts the thread id here.
        int threadId;

        // Create an instance of the test class.
        AsyncDemo ad = new AsyncDemo();

        // Create the delegate.
        AsyncDelegate dlgt = new AsyncDelegate(ad.TestMethod);
```

```
        // Initiate the asychronous call.
        IAsyncResult ar = dlgt.BeginInvoke(3000,
        out threadId, null, null);

        // Poll while simulating work.
        while (ar.IsCompleted == false)
        {
            Thread.Sleep(10);
        }

        // Call EndInvoke to retrieve the results.
        string ret = dlgt.EndInvoke(out threadId, ar);

        Console.WriteLine("The call executed on thread {0}, with return value \"{1}\".", threadId, ret);
    }
}
```

C#异步调用四大方法之异步调用完成时执行回调方法

如果启动异步调用的线程不需要处理调用结果，则可以在调用完成时执行回调方法。回调方法在 ThreadPool 线程上执行。

要使用回调方法，必须将代表该方法的 AsyncCallback 委托传递给 BeginInvoke。也可以传递包含回调方法将要使用的信息的对象。例如，可以传递启动调用时曾使用的委托，以便回调方法能够调用 EndInvoke。

```
public class AsyncMain
{
    // Asynchronous method puts the thread id here.
    private static int threadId;

    static void Main(string[] args)
```

```
{
    // Create an instance of the test class.
    AsyncDemo ad = new AsyncDemo();

    // Create the delegate.
    AsyncDelegate dlgt = new AsyncDelegate(ad.TestMethod);

    // Initiate the asychronous call.    Include an AsyncCallback
    // delegate representing the callback method, and the data
    // needed to call EndInvoke.
    IAsyncResult ar = dlgt.BeginInvoke(3000,
    out threadId,
    new AsyncCallback(CallbackMethod),
    dlgt);

    Console.WriteLine("Press Enter to close application.");
    Console.ReadLine();
}

// Callback method must have the same signature as the
// AsyncCallback delegate.
static void CallbackMethod(IAsyncResult ar)
{
    // Retrieve the delegate.
    AsyncDelegate dlgt = (AsyncDelegate)ar.AsyncState;

    // Call EndInvoke to retrieve the results.
    string ret = dlgt.EndInvoke(out threadId, ar);

    Console.WriteLine("The call executed on thread {0}, with return value \"{1}\".", threadId, ret);
```

```
  }
}
```

C#异步调用四大方法的基本内容就向你介绍到这里，希望对你了解和学习 C#异步调用有所帮助。