Invoke Web Services Asynchronously

Find out how to dispatch asynchronous Web services requests, cache Web services, and call Web services from behind a firewall.

by Juval Löwy and Jonathan Goodyear

Technology Toolbox

- ☑ VB.NET
- ☑ C#
- ❑ SQL Server 2000
- ☑ ASP.NET
- ❑ XML
- ❑ VB6

Go Online!

Use these Locator+ codes at www.visualstudiomagazine.com to go directly to these related resources.

Download

VS02ENQA Download the code for this article, which includes code for caching Web services and for calling Web services from behind a firewall.

Discuss

VS02ENQA_D Discuss this article in the XML & Web Services forum.

Read More

VS02ENQA_T Read this article online.

VSEP020305AN_T "5 Vital Caching Tips" by Jonathan Goodyear

VS0205RT_T "Invoke Asynchronous Magic" by Robert Teixeira

VS0204JL_T "Tame .NET Events" by Juval Löwy

Q: Invoke Web Services Asynchronously

When I add a Web reference in Visual Studio .NET to a Web service and invoke the Web service, the client is blocked until the method is returned. Is there a way to invoke the service asynchronously? If so, how do I get the returned parameters?

A:

Invoking Web services asynchronously with VS.NET is as easy as synchronous calls. The reason is simple: VS.NET generates a wrapper class when you add a Web reference, exposing the original Web methods. At the same time, the wrapper class contains methods allowing you to invoke the service asynchronously. For example, consider the Calculator Web service, written in C#, which provides basic arithmetic operations:

```
using System.Web.Services;

public class Calculator
{
   [WebMethod]
   public int Add(int num1,int num2)
   {
      return = num1+ num2;
   }
   [WebMethod]
   public int Subtract(int num1,int
      num2)
   {
      return = num1- num2;
   }
   //Other methods
}
```

When you add a Web reference to this Web service, the auto-generated wrapper class on the client's side contains the Add() method, which you can use to invoke the call synchronously. In addition, it contains BeginAdd() and EndAdd() methods, used to invoke the Web service asynchronously:

```
using System.Web.Services.Protocols;

public class Calculator :
   SoapHttpClientProtocol
{
   public int Add(int num1,int num2)
   {...}
   public IAsyncResult BeginAdd(int
      num1,int num2,
      AsyncCallback callback,object
      asyncState)
   {...}
   public int EndAdd(IAsyncResult
      asyncResult)
   {...}
   /* Other methods  */
}
```

In .NET, you typically use delegates to invoke calls asynchronously. With the Web service wrapper class, you don't need to use a delegate, because the wrapper class provides the asynchronous methods already. Nonetheless, the programming model when using nondelegate-based asynchronous method calls is similar to using the BeginInvoke() and EndInvoke() methods provided by the delegate class. You dispatch the asynchronous operation using Begin<method name>, and you can either call End<method name> to block until completion, wait for the operation (or multiple operations) to complete, or use a callback method. However, unlike delegates, there's no uniform requirement to call End<method name> on the original object used to dispatch the Begin<method name> call. You can simply create a new instance of the wrapper class and call End<method name> on it. As I mentioned,

C# • Add a Completion Callback to This Asynchronous Web Service Call

```
public class CalculatorWebServiceClient
{
  public void AsynchAdd()
  {
    //Calculator here is the auto-generated
    //wrapper class
    Calculator calc;
    calc  = new Calculator();

    AsyncCallback callback;
    callback = new AsyncCallback
      (OnMethodCompletion);

    calc.BeginAdd(2,3,callback,null);
  }
  protected void OnMethodCompletion(IAsyncResult
    asyncResult)
  {
    //Calculator here is the auto-generated
    //wrapper class
    Calculator calc;
    calc = new Calculator();

    int result;
    result = calc.EndAdd(asyncResult);
    Trace.WriteLine("Operation returned " +
      result.ToString());
  }
}
```

Listing 1 The client provides a delegate targeting a callback method to Begin<Operation>. .NET calls the callback when the Web method returns; meanwhile, the client is free to do concurrent processing. The callback method uses End<Operation> to retrieve the returned value and the outgoing parameters.

several asynchronous programming models are available to the Web service client. The client can issue the call, do some work, then block until the Web method returns by calling End<Operation>, passing in the IAsyncResult object the client got back from Begin-<Operation>:

```
//Calculator is the auto-generated wrapper class
Calculator calc;
calc = new Calculator();

IAsyncResult asyncResult;
asyncResult = calc.BeginAdd(2,3,
  null,null);
/* Do some work */
//Block til completion:
int result = 0;
result = calc.EndAdd(asyncResult);
```

You can also use the AsyncWaitHandle property of the IAsync-Result object returned from Begin<Operation> to dispatch a number of asynchronous Web services requests, then wait for all or any of them to complete:

```
Calculator calc1;
Calculator calc2;
IAsyncResult asyncResult1;
IAsyncResult asyncResult2;

asyncResult1 = calc1.BeginAdd
  (1,2,null,null);
asyncResult2 = calc2.BeginAdd
  (3,4,null,null);

WaitHandle[] array = new WaitHandle[2];
array[0] = asyncResult1.AsyncWaitHandle;
array[1] = asyncResult2.AsyncWaitHandle;

//Wait for all calls to return
WaitHandle.WaitAll(array);
```

The easiest asynchronous programming model involves a client-provided callback method. You provide Begin<method name> with a delegate of type AsyncCallback, targeting a method to be called when the Web service execution is done. The callback is done on a thread from the thread pool. Also, .NET passes to the callback the IAsyncResult object returned from Begin<method name>, so there's no need to save it, and you can have the same callback service for multiple asynchronous calls (see Listing 1). —*J.L.*

Q: Cache Web Services

I'm looking to optimize my Web services. What are some of my caching options?

A:

Several caching options are available to you, depending on your needs. The simplest way to add caching to your Web service is to add the CacheDuration property to your Web service's <WebMethod()> declaration:

```
'set number of seconds to cache
'GetTime1 Web Service
<WebMethod(CacheDuration:=120)> _
   Public Function GetTime1() _
   As DateTime
   Return DateTime.Now
End Function
```

The value you assign to the CacheDuration property sets the number of seconds that your Web service is cached. This approach

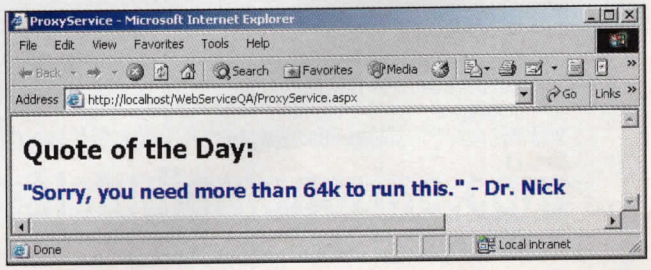

Figure 1 Interact With Web Services Outside Your Firewall. The WebProxy and NetworkCredential classes work together to allow you to interact with Web services outside your firewall. A local Web server displays this quote, but it's being retrieved from the GotDotNet Quote of the Day Web service, which is outside the firewall.

is fairly simple and easy to implement, but it isn't flexible. For instance, there's no way to remove your Web service manually from cache if you have to. In addition, when your Web service expires from cache, the next person to request your Web service must wait while it retrieves new data. This might take a large amount of time in some circumstances.

A better approach would be to manage your Web service's cached data manually. Add a custom CacheItemRemovedHandler to your Web service so you're notified when your Web service data is removed from cache (see Listing 2). If you determine that the Web service data expired and wasn't removed explicitly, then you can refresh the data automatically and reinsert it into cache. Then when the next person requests the Web service, he or she won't have to wait for the data to be refreshed, making for a better experience (see Listing 3 for the Web service method for your manually cached Web service).

First, you get a reference to the Cache object by using a reference to the current HttpContext. Then, use the Cache object to check whether there is an instance of "ServiceTime" in cache already. If it doesn't exist, add it using the Cache.Insert() method, specifying the CacheItemRemovedCallback handler's address to notify you when it's removed from cache. —*J.G.*

Q: Call Web Services Behind a Firewall

I cannot seem to call Web services from behind my firewall. That is supposed to be one of the biggest advantages of Web services. What gives?

A:

Web services are touted as being able to pass through firewalls because they normally operate on port 80 (the port for standard HTTP traffic). Regardless, your Web service calls still need to have security credentials for your firewall's proxy server. Unfortunately, VS.NET's Web Reference feature doesn't handle this well, unless your Windows login account is in the same domain as your proxy server. Many times, this isn't the case. In that circumstance, you need to use the WSDL.exe utility to compile proxy classes manually to call Web services. Use this command to compile a proxy class for the Quote of the Day Web Service on GotDotNet (see Additional Resources):

```
C:\>WSDL.exe /l:VB /n:gotDotNet
   /proxy:proxy.someURL.com
   /proxydomain:myProxyDomain
   /proxypassword:myPassword
   /proxyusername:myUserName
   http://www.gotdotnet.com/playground/
   services/qotd/QotD.asmx?wsdl
```

Make sure you have the directory that you installed VS.NET to in your system path, or that you use the VS.NET Command Prompt located in the Start | Programs | Microsoft Visual Studio .NET | Visual Studio .NET Tools program group. The "/l" switch specifies the language you want your proxy class to use (Visual Basic .NET, in this case). Use the "/n" switch to specify the namespace for your proxy class. You can add the "/o" switch if you want to specify the name of the proxy class file that the WSDL.exe utility emits as output.

The ASP.NET Web application that calls a Web service typically runs without a user logged in, so you need to specify security credentials whenever you call Web service methods from behind a secure firewall. Create a new ASP.NET project named WebServiceQA and import the proxy class you created. Create a new Web Form named ProxyService, and drop a Label control named quoteLabel on it. In the code-behind class for ProxyService, add a reference to

ASP.NET, VB.NET • Get Notified When Your Web Service Data Expires

```
Private Sub MyCacheItemRemovedCallback(ByVal Key _
  As String, ByVal Val As Object, ByVal Reason _
  As CacheItemRemovedReason)
  If Key = "ServiceTime" Then
    'make sure that we did not explicitly
    'remove the item
    If Reason <> _
      CacheItemRemovedReason.Removed Then
      HttpContext.Current.Cache.Insert(Key, _
        DateTime.Now, Nothing, _
        DateTime.Now.AddSeconds _
        (10), Nothing, _
        CacheItemPriority.Normal, _
        AddressOf _
        MyCacheItemRemovedCallback)
    End If
  End If
End Sub
```

Listing 2 If the cache key for the item removed from cache matches the one you're concerned about, check to make sure it wasn't removed from cache intentionally. If it wasn't, then refresh the data and insert it into cache. Be sure to specify the CachedItemRemovedCallback handler's address, so you're notified again when the data is removed from cache.

ASP.NET, VB.NET • See if Data is Present in Cache

```
<WebMethod()> _
  Public Function GetTime2() As DateTime
  Dim theContext As HttpContext = _
    HttpContext.Current
  Dim theCache As Cache = theContext.Cache

  'check to see if there is an
  'instance of ServiceTime in cache
  If theContext.Cache("ServiceTime") _
    = Nothing Then
    'not in cache, so add it
  HttpContext.Current.Cache.Insert _
    ("ServiceTime", DateTime.Now, _
    Nothing,DateTime.Now.AddSeconds _
    (10), Nothing, CacheItemPriority.Normal, _
    AddressOf MyCacheItemRemovedCallback
  End If

  'return ServiceTime from cache,
  'after casting it to DateTime type
  Return Convert.ToDateTime _
    (theContext.Cache("ServiceTime"))
End Function
```

Listing 3 Check to see if the data you need to return from the Web service is present in cache. If not, create a new set of data and store it in cache. Be sure to pass a reference to a CacheItemRemovedCallback delegate to the Cache object's Insert method, so your callback function fires when your cached data expires.

the System.Net and WebServiceQA.gotDotNet namespaces. Then, place this code in the Page_Load event:

```
'instantiate Web service proxy class
Dim qt As Quote = New Quote()

'build network proxy to communicate through firewall
Dim wp As WebProxy = New WebProxy()
wp.Address = New _
   Uri("http://proxy.someURL.com:80")
wp.BypassProxyOnLocal = True
wp.Credentials = New NetworkCredential("userID", _
   "password", "proxyDomain")

qt.Proxy = wp
quoteLabel.Text = qt.GetQuote()
```

At the top, create an instance of the proxy class for the ContentFeed Web service. Then, create an instance of a WebProxy object. Set its Address property to your proxy server's URL. Note the port number at the end. Your proxy server can't use port 80. The BypassProxyOnLocal property allows you to bypass proxy authentication if you'll be calling a Web service that's located inside your firewall's confines. The WebProxy class's Credentials property requires an instance of the NetworkCredential class (see Figure 1). The NetworkCredential constructor used here accepts three parameters: the username, password, and domain name of the account you're using to authenticate to the proxy server. Assign the WebProxy instance to the Proxy property of your Quote Web service proxy class. You can then call the GetQuote() method of your Web service proxy class, which returns a string containing the text of today's quote of the day. Assign this result to the Text property of the quoteLabel ServerControl that you added to your Web Form.

As an alternative to the verbose method just described, you can assign the Proxy property of your Quote Web service proxy class in a single line of code:

```
cf.Proxy = New WebProxy _
   ("http://proxy.someURL.com:80", _
   True,Nothing,New _
   NetworkCredential _
   ("userID","password", _
   "proxyDomain"))
```

You can see that it's possible to call Web services through proxy-guarded firewalls using VS.NET, but it isn't as "plug and play" as Microsoft would have you believe. Hopefully, Microsoft will address this usability issue in a future service pack. —*J.G.*

Additional Resources

Quote Web Service: www.gotdotnet.com/playground/services/qotd/qotd.asmx

Juval Löwy is the principal of IDesign, a consulting and training company focused on .NET design and .NET migration. He is the author of *COM and .NET Component Services* (O'Reilly & Associates). Juval speaks at software development conferences and chairs the program committee of the .NET California Bay Area User Group. Contact him at www.idesign.net.

Jonathan Goodyear is an MCSD and the president of ASPSoft (www.aspsoft.com), an Internet consulting firm based in Orlando, Fla. He is the author of *Debugging ASP.NET* (New Riders Publishing), and speaks regularly at Web development conferences. Reach him by e-mail at jon@aspsoft.com or through his angryCoder eZine at www.angryCoder.com.