利用 MTOM 编码和异步调用改进流传输的性能①

罗小平，胡卫东，闫三虎

(西南石油大学 计算机科学与工程学院，成都 610500)

摘 要: 采用文本编码来格式化消息时会增加客户和服务信道之间的数据传输量，且随着文件长度的增加传输量会线性增长，其性能会大打折扣。WCF(Windows Communication Foundation)作为新一代的网络传输开发架构，其在构建面向服务的应用程序方面具有无比强大的生产力，本文以这个架构为蓝本，在消息层采用 MTOM(Message Transmission Optimization Mechanism)编码构建 SOAP(Simple Object Access Protocol)包，在传输层实现流模式，并利用异步调用提高应用程序的响应性能，可以显著提高文件流的传输性能。

关键词:WCF；MTOM 编码；流模式；异步调用

Improving the Performance of Streaming by Using MTOM Encoding and Asynchronous Calls

LUO Xiao-Ping, HU Wei-Dong,YAN San-Hu

(College of Computing Sciences, Southwest Petroleum University, Chengdu 610500, China)

Abstract: When formatting message using Text Encoding, the data transmission will increase between the client and service channel capacity. With the increment of the file length, throughput will increase linearly and its performance will be greatly reduced. As a new generation development architecture of network transmission, WCF (Windows Communication Foundation) has extremely powerful productivity in building the service-oriented application . This paper is to be modeled on this structure. The message layer constructs SOAP (Simple Object Access Protocol) package using MTOM (Message Transmission Optimization Mechanism) and the transport layer achieve streamming mode. The use of asynchronous calls can improve applications' performance of response. This program can significantly improve file stream transmission performance.

Keywords: WCF; MTOM encoding; streamming mode; asynchoronous calls

在个人主页上传一张照片到服务器的相册中，或者把一部电影通过网络传给另一台 PC，这些都是利用了网络实现两个应用程序之间的数据流的传输，也就涉及到鲜为人知的 Socket 编程模型。不过这种模型存在一个很大的缺陷，那就是它不能方便地重构原型。为了克服这种缺陷，世界软件的巨头如微软、IBM、Oracle 等便联合在一起组成了一个名为 WS-I(Web Service Interoperability)的组织，制定了远程信息交换的 WS-* 规则，这种新的编程模型称为 SOA (Service-oriented Architecture)，它使用 SOAP(Simple Object Access Protocol)消息来方便地在网络异端重构类型。于是微软 WCF(Windows Communication Foundation)便应运而生，它不仅规定了网络中信息流的编码方式，也定义了一系列共同遵守的契约，形成了该模型中三个必不可少的要素，即地址(Address)、绑定(Binding)、契约(Contract)[1]。地址定义了网络消息送达之处；绑定定义端点通信的信道；契约定义端点提供的功能集合。为了克服大容量二进制文件流传输性能差的缺点，需要改变 WCF 模型中契约和绑定的方式。通过改变信道层的消息编码和服务调用的方式来改进大容量二进制文件流的传输性能是本文着重探讨的两个方面。

1 通过MTOM编码提高消息传输效率

在 WC 内部提供了三种编码方式：Binary

① 收稿时间:2010-04-19;收到修改稿时间:2010-05-27

MessageEncoder， TextMessageEncoder， Mtom MessageEncoder。其中 BinaryMessageEncoder 提供紧凑的二进制格式来串行化消息，不过这种格式化的消息是不可互操作的；TextMessageEncoder 具有互操作能力，它在对二进制数据进行编码时，先将二进制流转换成字节数组，再将这些字节数组以 Base64 编码得到 XML 表示，最后再以 SOAP 消息格式在网络中进行传输。由于 Base64 编码将使用三个字节来表示四个字符的数据，这使得编码后的数据量最多可能达到原来的 133%，这在大文件流的传输中是一个不小的开销。而 MTOM 编码介于二者之间，它不仅具有互操作能力，还可以大大减少大文件流的传输消息量。

1.1 MTOM 编码

MTOM(Message Transmission Optimization Mechanism)编码，即消息传输优化机制[2]，原理如图 1 所示。这种机制的做法是不对二进制流进行编码，而是把它作为 SOAP 消息的附件发送，于是这些大二进制流就没有 Base64 编码的开销。当然传输的数据量就大大地减少了。因此在传输大文件流时选择 MTOM 编码消息可以有效地提高传输性能。

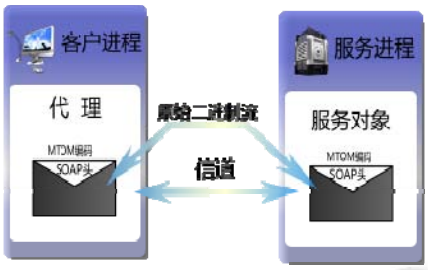

图 1　MTOM 消息包传输

为了在 WCF 中使用 MTOM 编码，需要在配置文件（app.config 或 web.config）里将 messageEncoding 设置成 Mtom 即可，相关的配置节信息如下：

```
<bindings>
<basicHttpBinding>
<binding name="MTMOBinding"
messageEncoding="Mtom"/>
</basicHttpBinding>
</bindings>
```

1.2 激活流传输模式

流传输是特指使用文件流的方式来操作 SOAP 消息包，这对大文件的传输很有效果[3]。相对于传统的缓存系统而言，这种方式在发送和接收时不必缓存消息，而是以管道流的方式将数据流从一端发送到另一端。这样可以防止当整个文件实例装入到内存时由于缓存不够而导致内存异常，对于服务客户来说，就不用足够的内存来缓存整个文件了。所以为了进一步提高传输的效率，采用了流传输模式。

1.2.1 操作契约约定

对于一个以消息流方式发送内容的操作来说，它必须以一个单一的 Stream 对象作为操作契约的输入输出参数，这也意味着如下形式的方法签名会被许可：

```
void SendStream(Stream inStream);
Stream ReceiveStream();
Stream SendAndReceiveStream(Stream
inStream);
void SendAndReceiveStream(Stream inStream，out
Stream outStream);
void ReceiveStream(out Stream outStream);
```

为了使用流模式，操作契约使用了如下的签名方法：

```
[OperationContract]
void SendStream(Stream stream);
```

1.2.2 绑定设置

经典的 BasicHttpBinding，还有 NetTcpBinding，NetNamedPipeBinding 都是支持流传输模式，为了激活流传输模式，在应用程序的配置文件的绑定设置中必须设置 TransferMode 属性为下列值之一：

Buffered 缓存消息，即不是消息流的传输模式；

Streamed 激活一个双向的消息流；

StreamdRequest 仅在请求方向激活消息流；

StreamdResponse 仅在回复方向上激活消息流；

这儿考虑到使用 MTOM 编码和流传输模式，所以选择使用经典的 BasicHttpBinding，因为这个绑定同时支持这两个属性，而其他两个只支持流传输模式，同时为了要传输大容量的文件流，所以还得修改相应的 maxReceivedMessageSize 和 maxArrayLength 属性，下面的设置可以支持 700M 以下的文件流的传输：

```
<basicHttpBinding>
  <binding name=" MTMOBinding "
maxReceivedMessageSize="734003200"
```

```
messageEncoding="Mtom"
transferMode="StreamedRequest">
    <readerQuotas maxArrayLength="734003200" />
    </binding>
</basicHttpBinding>
```

2 通过异步调用改进响应性能

程序设计中，方法的调用可以采用同步和异步两种方式。同步调用是指方法在调用过程中一直阻塞，直到方法调用结束返回结果才会让程序继续向前执行，这种行为比较耗费资源，因为网络访问在等待方法完成的时间内是阻塞的。而且如果是远程对象的调用则花费的时间会更长，所以这种时间的浪费让人是不可接受的，这在大文件传输中尤为明显。于是在大文件流传输中采用异步调用的机制不失为一种好的方式。

2.1 异步调用机制

异步调用的机制在于调用方法一启动就立即返回，而不等待调用完成。在内部处理过程中，该调用会立即被线程池中的一个线程接管。等到调用完成后程序得到返回信息，而不用程序阻塞等待返回结果，其管理由.NET Framework 的 CLR(Common Lauguage Runtime)接任。在 WCF 中引入了异步调用机制，这里面任何服务在本质上都是支持异步调用的，可以同时采用同步和异步两种方式来调用同样的服务，这就意味着在服务实现中只关心同步操作的方法，而不用专门实现相应的异步操作。有两种方式来实现异步调用:一种选择是在设计 WCF 时由 Visual studio 生成服务引用时选择生成异步调用，这样在生成客户端的类型将同时生成相应的异步方法，它是形如：IAsyncResult Begin<Operation> (params list，AsyncCallback callback，object state)和 Type End<Operation> (IAsyncResult result)的签名方法；另一种是在客户端的操作契约中显式地指明生成相应的异步方法的签名(必须设置 AsyncPattern 为 true)[4]。为了减少强类型的藕合性，使用第二种方式更具有灵活性：

```
[OperationContract(AsyncPattern=true)] //声明一个异步调用的签名
IAsyncResult BeginSendStream(Stream inputStream，AsyncCallback callback，object state);
//结果调用方法
void EndSendStream(IAsyncResult result);
```

2.2 异步调用过程

在前面的契约中定义了一个操作契约

void SendStream(Stream stream)，签名方法实际上是一个同步方法。而在声明一个对应的异步调用时，WCF 内部却调用了相应的同步方法来执行操作，只是在调用了同步方法后就立即返回，让调用线程继续执行后面的代码，在服务操作完成时会调用 AysncCallback 的回调方法告知客户端，然后由客户端执行剩下的工作。调用过程如图 2 所示。

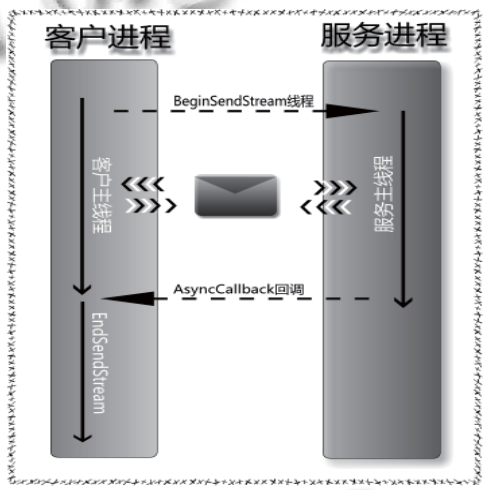

图 2 异步调用时序

2.3 实现服务器的同步方法

在上面定义了服务契约的接口后，接下来就实现服务类型，服务类型的目的是为了传输 Stream 对象，由于 Stream 是一个抽象类，所以这儿采用文件流为操作对象来使用 SendStream()这个方法，同时把服务行为设置为单调服务和并发使用，这样可以得到灵活的扩展性能，此外将文件名信息附加到消息头上以消除不能在操作方法上传递参数的限制。为了运用服务方法，要同时按照上面的配置设置更新配置文件。相应的实现如下：

```
[ServiceBehavior
(InstanceContextMode=InstanceContextMode.PerCall,C
oncurrencyMode=ConcurrencyMode.Multiple)]
public class SendStreamService:ISendStreamService {
 public void SendStream(System.IO.Stream stream){
 int maxLength = 8192;
```

```
 int length=0;
 int pos=0;
 byte[] buffer =new byte[maxLength];
 //从附加的消息头来得到文件路径信息
 int index =
OperationContext.Current.IncomingMessageHeaders.Find
dHeader("fileName", "http://tempuri.org");
 string file=OperationContext.Current.
IncomingMessageHeaders.GetHeader<string>(index).To
String();
 FileStream outStream = new FileStream(file,
FileMode.OpenOrCreate, FileAccess.Write);
 while ((pos = stream.Read(buffer, 0, maxLength)) > 0){
  outStream.Write(buffer, 0, pos);
  length += pos;
  }
 }
}
```

2.4 定义客户类和调用服务方法

为了使用客户端的异步调用，除了在客户端契约中指明上面的异步方法签名以外，还要调用工厂方法来得到一个客户端实例：

```
    public class SendStreamClient
    :ClientBase<Service.ISendStreamService>,
Service.ISendStreamService{    //定义客户类
    public SendStreamClient() { }
    public SendStreamClient(string endpointName) :
base(endpointName) { }
    public SendStreamClient(Binding binding,
EndpointAddress endpointAddress)
    : base(binding, endpointAddress) { }
    public void SendStream(System.IO.Stream
stream){
    Channel.SendStream(stream);
    }
    //定义异步签名
    public IAsyncResult BeginSendStream(
System.IO.Stream stream, AsyncCallback callback,
object asyncState){
    return Channel.BeginSendStream(stream,
callback,asyncState);
    }
    ……
    }
    }
    //下面是调用客户对象执行相应的调用
    SendStreamClient client;  //客户对象
    FileStream inStream;  //文件流对象
    protected void Button1_Click(object sender,
EventArgs e){ //调用文件对话框得到文件名
    string file=FileUpload1.PostedFile.FileName;
    inStream = new FileStream(file, FileMode.Open,
FileAccess.Read);
    file = file.Substring(file.LastIndexOf("\\") + 1);
    string path = Server.MapPath(Request.Path);
    path = path.Substring(0, path.LastIndexOf("\\"));
    //实例化客户对象
    client = new SendStreamClient("setEndpoint");
    using (OperationContextScope scope = new
OperationContextScope(client.InnerChannel)){
    //创建传递文件名消息头
    MessageHeader fileHeader =
MessageHeader.CreateHeader("fileName",
"http://tempuri.org", string.Format("{0}//{1}", path,
file));
OperationContext.Current.OutgoingMessageHeaders.Ad
d(fileHeader);
    client.Open();
    client.BeginSendStream(inStream, new
AsyncCallback(Callback), file);  //执行异步调用
    }}
    //回调方法
    private void Callback(IAsyncResult result){
    string file= result.AsyncState as string;
    client.EndSendStream(result);
    result.AsyncWaitHandle.Close();
    inStream.Close();
    client.Close();
    }
```

2.5 实现性能对比

为了测试性能的改进情况，这儿以一个 288M 的电影文件做测试，用同步方法加文本编码和异步方法

加 MTOM 编码进行传输，并生成消息日志，通过 Service Trace Viewer 跟踪得到如图 3 的结果。

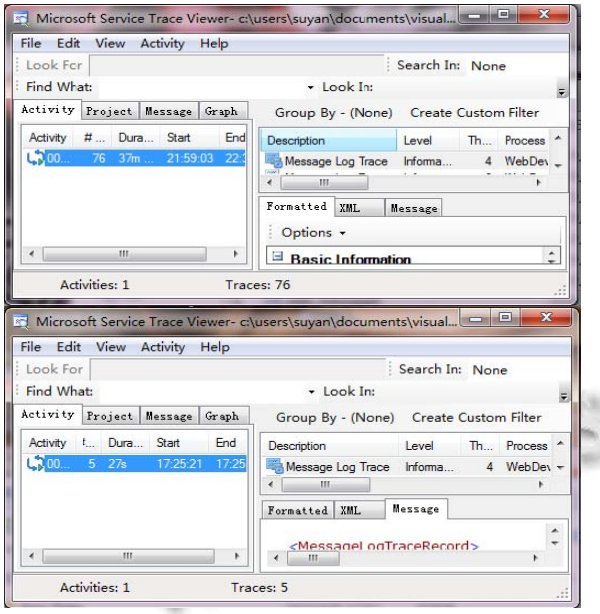

图 3 性能分析对比

在时间和消息次数两个方面有明显的差别：

时间方面：前一种是耗时 37 秒,而后一种只耗时 27 秒。

消息传输次数：前一种达到了 76 次往返，而后一种却只有 5 次往返。

综合以上的结果发现，在异步和 MTOM 编码方式下，文件流作为一个附件附加在 SOAP 消息中，所在传输的数据量和次数都大大地减小了，整体性能只有普通方式的 72.97%。这也证实了性能改善是很明显的。

3 结束语

通过以上的分析和设计，采用 WCF 的 MTOM 编码和异步调用模式，可以有效地提高服务和客户的传输性能。在实际应用中，如在 ASP.NET 编程中可以突破 4M 规则，将本地文件方便快捷地上传到 Web 服务器，在传统的客户/服务模型中可以迅速地在本地和服务器之间上传和下载文件。随着应用的深入，这种方式必将成为大文件流传输的标准的设计方式之一。

参考文献

1 Lowy J, Vasters C. Programming WCF Services. 2nd. O'Reilly Media Inc, 2007. 13－22.

2 McMurtry C. WCF3.5 揭秘. 赵科平,龚岑,等译. 北京:人民邮电出版社, 2009.415－421.

3 Bustamante ML. Learning WCF. O'Reilly Media Inc, 2007. 60－75.

4 Resnick S, Crane R, Browen C. Essential Windows Communication Foundation. Pearson Education Inc, 2008. 321－325.