迁移到 WSE 3.0

发布日期： 2006-5-9 | 更新日期： 2006-5-9

下载本文的代码： ServiceStation0604.exe (139KB)

本页内容

您可能已经听说，Web Services Enhancements (WSE) for Microsoft® .NET Framework 的新版本简化了构建安全 Web 服务的过程。但您可能不知道，其中大多数改进源自 WSE 3.0 中所做的一些核心体系结构更改。本专栏讨论更改的内容，并研究转移到 WSE 3.0 的过程中将面临的主要迁移问题。

WSE 3.0 中的主要改动

MSDN® 最近发表了一篇题为"*What's New in WSE 3.0*"的文章，作者是 WSE 团队的首席项目经理 Mark Fussell。本文概括自 WSE 2.0 以来对该产品的主要改动和改进，包括与 .NET Framework 2.0 和 Visual Studio® 2005 集成，通过新增和改进的策略体系结构简化了安全性，以及支持使用消息传输优化机制（Message Transmission Optimization Mechanism，MTOM）发送大量数据。其他改动包括：改进的会话管理（通过 WS-SecureConversation），支持在 IIS 外宿主 ASMX 服务，以及支持最新的 WS-* 规范。

尽管 WSE 团队竭尽全力确保向后兼容性，但是其中某些改动会影响核心编程模型，因此您必须解决这些问题才能更新现有的 WSE 2.0 解决方案。

好消息是：一旦迁移到 WSE 3.0，您将拥有更多的通讯选择。WSE 3.0 支持最新的 WS-* 规范（Windows® Communication Foundation (WCF) 也支持这些规范），这使得 WSE 3.0 和 WCF 是网络兼容的。这意味着 WCF 客户端可以与 WSE 3.0 服务互操作；反之亦然，无需继续迁移。

↑ 返回页首

将项目从 **WSE 2.0** 迁移到 **WSE 3.0**

谈到将现有代码向前迁移到 WSE 3.0，与 .NET Framework 2.0 和 Visual Studio 2005 的集成就显得非常重要。WSE 3.0 构建在 .NET Framework 2.0 之上，并且与 Visual Studio 2005 集成 - 也就是说，它既不运行在 .NET Framework 1.1 上，也不与 Visual Studio .NET 2003 集成。因此，向前迁移 WSE 2.0 解决方案时，第一步就是将 .NET Framework 1.1 解决方案向前迁移到 .NET Framework 2.0。幸运的是，Visual Studio 2005 提供了自动迁移工具，大大地简化了该步骤。

您只需在 Visual Studio 2005 中打开一个现有的 Visual Studio .NET 2003 解决方案即可开始迁移它。该操作启动了 Visual Studio 转换向导，在该向导中开始自动转换到新的 Visual Studio 2005 格式。转换完成后，该向导生成一份报告，这样您就可以了解该向导进行了哪些操作，并检查任何错误或警告。现在您可以构建该解决方案，编译器将帮助您需要着手进行的代码更改，以及在 .NET Framework 2.0 上运行。

接下来，您需要将 WSE 3.0 与您的解决方案重新集成。删除对 Microsoft.Web.Services2.dll 的引用，添加对 Microsoft.Web.Services3.dll 的引用，更新您的配置文件，并修订所有受改动的 WSE 3.0 类库影响的代码。

最简单的开始方法是通过右键单击每个项目，选择 WSE Settings 3.0 来打开 WSE 配置工具。如果该工具在现有配置文件中检测到一个 WSE 2.0 配置部分，那么该工具显示一个对话框，询问您是否愿意自动将设置导入新的 WSE 3.0 配置部分。回答"是"，该工具则对此文件进行必要的更改（至少尽其所能）。该工具进行诸如以下这些操作：注册 WSE 3.0 配置部分，将 <soapExtensionType> 元素更改为新的 <soapServerProtocolFactory> 元素，以及将大多数现有 WSE 2.0 设置转移到新的 WSE 3.0 配置部分。而且，该工具自动为您更新对 Microsoft.Web.Services3.dll 的引用。

您还可以在"开始"菜单中以独立模式（在 Visual Studio .NET 外部）启动 WSE 3.0 配置工具。您需要手动选择一个配置文件（使用"文件"菜单），但该文件仍然要向前迁移 WSE 2.0 设置。

尽管 WSE 3.0 配置工具在迁移过程中为您提供了一个良好的开端，但它照顾不到所有的情况。当该工具检测到没有自动向前迁移的配置项时，它将一切保持原样。然后显示警告对话框，说明发现了一些错误，建议您查看日志，然后询问您是否继续。它编写日志文件 (WseSettingsErrors.log) 中需要注意的具体详细信息。这暗示了您要做很多工作。

该过程中的下一步是处理破坏代码的任何改动。破坏性改动可分为四类：常规类库、直接 Internet 消息封装 (DIME)/MTOM、安全性和策略/筛选器。

↑ 返回页首

常规类库

尽管 WSE 3.0 配置工具将用新程序集引用和新配置信息自动更新您的项目，但您仍必须手动更新源代码以处理对 WSE 3.0 类库的一些改动。例如，您必须更新所有源文件以使用新的 WSE 3.0 命名空间 (Microsoft.Web.Services3)。全局搜索和替换通常用来执行该操作。

图 1 突出显示 WSE 2.0 和 WSE 3.0 之间主要的命名空间差别。由于大多数体系结构改动发生在策略领域，因此 WSE 3.0 将策略命名空间更改为 Microsoft.Web.Services3.Design。您可能注意到删除了几个 WSE 2.0 命名空间，添加了几个新的命名空间。

除了对 WSE 3.0 所做的一些命名空间改动，许多核心库类仍然保持原样。例如，典型的 WSE 2.0 消息处理应用程序（那些使用 SoapSender/SoapReceiver 的应用程序）应该干净利落地进行迁移。现在，通过对 ASMX 的新支持，您只不过多了一些选择。以下部分讨论已更改的类库区域。如果您的解决方案在进行了必要的命名空间更改后进行编译，它应该像在 WSE 3.0 上一样运行，如果未对应用程序应用任何策略，默认情况下仍使用 WSE 2.0 管道。

↑ 返回页首

DIME/MTOM

另一个需要注意的区域是任何涉及 DIME 附件的 WSE 2.0 代码。例如，以下 WSE 2.0 代码将一个 JPEG 文件作为 DIME 附件返回给调用方：

```
[WebMethod]
public void GetFile(string fileName)
{
    SoapContext respContext = ResponseSoapContext.Current;
    DimeAttachment dimeAttach = new DimeAttachment(
      "image/jpeg", TypeFormat.MediaType, fileName);
    respContext.Attachments.Add(dimeAttach);
}
```

在 WSE 3.0 中，Microsoft.Web.Services2.Dime 命名空间已经从程序集中删除（不再有 DimeAttachment 类），Attachments 属性已经从 SoapContext 类中删除。因此，必须将这段代码修改为使用新的 WSE 3.0 MTOM 模型。

使用 MTOM，您只需将二进制数据（您过去将这些数据作为 DimeAttachment 传输）建模为签名中的二进制数组，如下所示：

```
[WebMethod]
public byte[] GetFile(string fileName)
{
     byte[] response = File.ReadAllBytes(filePath);
     return response;
}
```

接下来，通过 WSE 3.0 配置部分或通过使用某个代理上的 RequireMtom 属性来启用 MTOM（看看配置工具中的 Messaging 选项卡）。启用后，使用 MTOM 优化对所有已传输的字节数组进行自动编码。

如果您目前正在用一条消息传送多个 DimeAttachment，通过将多个字节数组封装在一个您在签名中使用的类中，即可将该方案传递给 MTOM。MTOM 支持对它在一条消息中发现的所有字节数组进行自动编码。

↑ 返回页首

安全性

WSE 3.0 大大简化了保护 Web 服务安全的过程。一组完备的安全策略断言涵盖了最常见的情况和方案。这种安全性新方法建立在改进的 WSE 3.0 策略框架（后面将对此进行详细介绍）基础之上。

如果您目前使用 WSE 2.0 策略框架，请将重点从当前的策略文件转移到新的 WSE 3.0 策略格式和新的完备断言上。手动运行 WSE 3.0 安全策略向导是实现该操作的最简单方法。由于 WSE 2.0 策略方法需要的代码很少，因此您不必接触许多源文件，只需处理在运行时提供令牌的代码。例如，您将不再使用 PolicyEnforcementSecurityTokenCache（在 WSE 3.0 中已经删除）；而将使用 SetClientCredential<T>。

如果通过直接与 SoapContext 的 Security 属性交互来编写命令性安全代码，您还需要多做一些工作。实际上，在 WSE 3.0 中已经不使用 Security 属性了，因此，如果代码引用该属性，您将收到一个编译器警告，指出您应该考虑编写一个安全筛选器来代替该属性。稍后，我将对此进行解释。

安全类本身实际上没有多大变化，只是您与它们交互的方式发生了变化。WSE 鼓励您在 WSE 管道中驻留的安全筛选器实现中执行安全任务，而不是在 WebMethod 中直接执行安全任务，这样就提高了安全代码的可重用性。大多数实际的安全代码无需更改即可迁移到一个安全筛选器类中。我将在后面的部

借 鑼 /a>
鍒涘缓鍙 嘿瀹氬箟鐨 Web 鏈嶅姟
淇濇姢 XML Web 鏈嶅姟鍏嶅彈棒戝 鏀诲嚮
浣跨敤 Microsoft SOAP Toolkit 2.0 瘑虹琦瀹夊叏 Web 鏈嶅姟
浣跨敤.NET 妗嗘灦灏嗗帮鏈变唬鐮佷繚淐 Web 鏈嶅姟鍞惧緵
浣跨敤SOAP Toolkit 2.0 灏嗗帮鏈变唬鐮佷繚淐 Web 鏈嶅姟鍞惧緵
浣跨敤SQL Server 2000 灏嗗帮鏈变唬鐮佷繚淐 Web 鏈嶅姟鍞惧緵
鐨夋 锛氭滐鍙 Web 鏈嶅姟 (disco)
绠€鍗曞 璞¤ 闀 崗瑻 /a>
瑙掓 鍚堝悓
浣跨敤 Web Services Enhancements 2.0 杩涜 缂栧▼
Web鏈嶅姟浜掓搷浣滄€у拰 SOAP
WindowsServer 2003 鍜 Visual Studio .NET 2003 涓 Web 鏈嶅姟鐨€鍙戜汉鍛樺彲浠ヤ娇鐢ㄧ殑鏂板姛鑳 /a>
缁撳悎 Web 鏈嶅姟鍜孭ocket PC Phone Edition 鍒涢€犱环鍊 /a>
W3C XML 鏋舵瀯瑙掓 妯″紡锛 仪鍏嶅 鏉傛€ /a>
W3C XML 鏋舵瀯瑙掓 妯″紡锛 氬 鐞嗘敼鍙 /a>
浣跨敤 Web Services Enhancements 鍙戦€佸甫鏈夐檮浠剁殑 SOAP 娑堟伅
Web Services Enhancements 绠ㄧ悊鍣€鏈 咾瀵 /a>
浜嗚В DIME 鍜 WS-Attachments
鏈嶅姟鍣ㄧ 鍨嬪 Web 鏂规硶
杞 欢鍗虫湇鍔 /a>
UDDI 锛氭簿绉 XML Web 鏈嶅姟
UDDI鏌ヨ 绀轰緥
XML缃戠粶鏈嶅姟瀹夊叏

分中讨论如何编写自定义筛选器和策略断言。

安全前端上唯一的主要类库更改就是已由等效类型（现在随 .NET Framework 2.0 一起提供）替换的 Microsoft.Web.Services2.Security.X509 类型（请参见 System.Security.Cryptography.X509Certificates）。

返回页首

策略 / 筛选器

级别较低的扩展性点好像总是最难以迁移，这同样也发生在 WSE 3.0 中。WSE 3.0 类库的最大变化是由新的策略体系结构造成的。WSE 3.0 策略框架不仅代替 WSE 2.0 引入的策略框架，还代替与管道中的筛选器协作所用的旧机制。您需要了解 WSE 2.0 和 WSE 3.0 策略框架之间的区别，以及 WSE 3.0 管道如何工作才能迁移安全性、策略以及与筛选器有关的代码。

WSE 2.0 策略框架依靠 WS-Policy 作为表达策略的格式。它支持 WS-PolicyAssertion 和 WS-SecurityPolicy 规范中定义的大多数策略断言。除了这些内置断言外，还有一个用于自定义策略断言的可扩展性机制。

WSE 2.0 向导生成了包含内置安全断言（主要是 <Integrity> 和 <Confidentiality>）的安全策略。然后，策略通过其配置文件与应用程序相关联。策略文件本身包含一个映射部分，以便根据策略元素的 URI 将策略元素与特定的服务终结点相关联。在运行时，WSE 2.0 检查配置的策略文件，以确定在特定 URI 处发送/接收消息时执行哪个策略。

WSE 2.0 策略框架作为距离 WSE 管道中应用程序最近的筛选器实现（请参见图 2）。所有策略逻辑都是在管道的这个特定步骤期间执行的。WSE 2.0 允许通过编写和配置自定义策略断言将自定义代码插入该策略步骤中。WSE 2.0 还允许您直接控制整个 WSE 管道的配置和操作。通过编写和配置自定义筛选器，您可以在该策略步骤之前或之后将代码插入管道中。

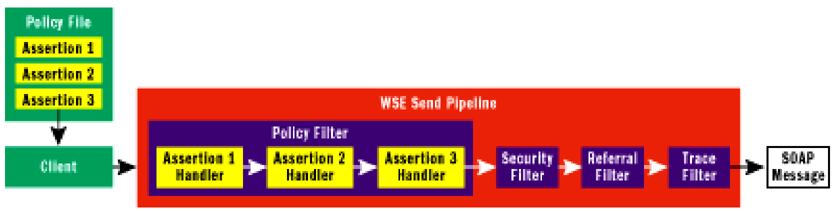

图 2 WSE 2.0 管道中的策略

WSE 2.0 策略框架有一些缺点。首先，不同的可扩展性选择经常会将事情搞乱。不清楚何时编写自定义筛选器，也不清楚何时编写自定义策略断言。

其次，将 WS-Policy 用作配置策略的语法对大多数开发人员而言过于复杂和麻烦。当开发人员必须调整 UI 没有公开的设置时，这迫使开发人员严重依赖安全策略向导，并且会使他们有点儿进退两难。

第三，用于将策略与终结点相关联的映射机制有时会导致不匹配。WSE 2.0 试图使用传入/传出消息的目标 URI 找到适当的策略（从配置的策略文件中）。通常，由于可能的 URI 差别，您不确定实际上是否应用了正确的策略。您每次更改服务的位置时（例如，在部署后或通过在一个目录上要求 HTTPS），您必须更新所有策略文件以匹配当前地址。

最后的一个主要缺点是，支持的断言需要您根据低级别的构造块进行思考，即使是在使用很好的安全策略向导时。例如，您必须独立于对应的响应消息指定请求消息的要求。在每种情况下，您都必须决定签名和加密，以及用于每个签名的令牌类型，无需太多指导或最佳做法。这通常会导致不太安全的安全策略。

返回页首

WSE 3.0 中的策略

WSE 3.0 中最基本的改变是，策略框架不再作为管道中的筛选器实现。相反，WSE 3.0 实际上使用策略驱动管道的整体创建（请参见图 3）。它本质上合并了策略断言和管道筛选器这两个领域，提供了一个用于配置和扩展 WSE 行为的统一模型。

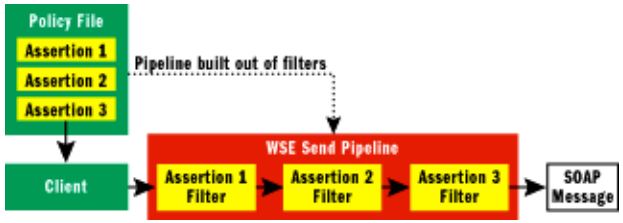

图 3 WSE 3.0 管道中的策略

WSE 3.0 通过使策略断言和管道筛选器成为一体，实现了该模型。WSE 3.0 初始化时，它检查提供的策略，并根据指定的断言从筛选器中构建管道。如果该应用程序没有配置为使用策略，它只创建一个与 WSE 2.0 中使用的管道配置等效的默认管道配置。

由于 WSE 3.0 使用策略配置管道，因此它在其配置部分中不再包括 <filters> 元素，配置工具也不再包括 Filters 选项卡。现在，已经通过策略完成了所有管道配置。对应的 WSE 3.0 类也进行了修改，以适应这种体系结构转变。

另一个显著的变化是您在 WSE 3.0 中表示策略的方法。WSE 3.0 决定定义自己的词汇表来表示策略和内置断言，而不使用 WS-Policy 和朋友定义的词汇表。产生的 XML 更干净且更易于使用，某些开发人员甚至会觉得无需工具帮助直接编辑它也很惬意。然而，WSE 3.0 仍然提供一个生成新格式策略的安全策略向导，这是迁移现有 WSE 2.0 策略的最简单方法。

WSE 3.0 还定义了一种新机制，用于将策略映射到服务终结点。WSE 3.0 机制更清楚在运行时将应用哪个策略。您将一个策略标识符（名称或类型）与服务或客户端代码相关联。在运行时，WSE 将使用提供的没有二义性的标识符找到指定的策略。这种转变缓解了通过 URI 将策略映射到服务终结点所产生的问题。

您将注意到的最后一个主要变化是，WSE 3.0 支持一组不同的内置安全断言。新的安全断言不再将重点放在低级别完整性和保密性的构造块上。相反，它们着重于代表当今最佳做法（如相互证书身份验证）的、较高级别的完备安全方案上。一个安全断言在使用最佳做法方式的同时描述请求和响应消息。按以下方法进行考虑：WSE 2.0 断言将重点放在消息级别的安全要求上，而新的 WSE 3.0 断言将重点放在操作级别的安全要求上。

这些 WSE 3.0 改动解决了前面描述的缺点，但其中大多数改动需要对代码和策略文件进行重大更改。我们来研究一些代码，看看这些代码是如何工作的。

返回页首

迁移自定义筛选器代码

迄今为止，我讨论过的主要策略概念直接映射到图 4 中概括的 WSE 3.0 类库中的类。

如果您在现有的 WSE 2.0 解决方案中使用自定义筛选器或策略断言，则需要进行一些更改，以便将您的代码引入到新的类设计中。我们看看详细信息。

假定您的服务需要根据对应的架构定义验证所有的传入请求，并且您目前已经通过一个 WSE 2.0 自定义筛选器实现了该验证。

在 WSE 3.0 中编写自定义筛选器与在 WSE 2.0 中编写自定义筛选器几乎一样。在 WSE 2.0 中，针对输入筛选器和输出筛选器（SoapInputFilter 和 SoapOutputFilter）有不同的基类。在 WSE 3.0 中，只有一个您可以从中派生的基类：SoapFilter。

SoapFilter 包含一个名为 ProcessMessage 的抽象方法，您需要重写该方法，如下所示：

```
public class ValidationFilter : SoapFilter
{
```

```
        public override SoapFilterResult ProcessMessage(
            SoapEnvelope envelope)
        {
            ... // process the message here
        }
    }
```

编写该代码是为了在 ProcessMessage 中就近实现要求。您可能希望通过不同的方式检查消息，传送它，或者修改消息上下文。ProcessMessage 的返回类型在 WSE 3.0 中也发生了改变。您返回一个 SoapFilterResult 值（SoapFilterResult.Continue 或 SoapFilterResult.Stop）以指明管道是否应该继续处理消息。

WSE 3.0 还提供了一些新的语句管理集合（作为 SoapContext 的一部分提供）。您将发现名为 MessageState、OperationState 和 SessionState 的属性，它们提供不同的作用域，用于在筛选器之间、跨消息交换甚至跨多个操作共享状态。

为了继续该验证示例，我需要根据其架构验证传入的 SoapEnvelope。我将要求筛选器的用户提供 XmlSchemaSet 对象，该对象包含本例中的验证所需的架构。如果消息通过架构验证，我将只允许消息在管道中继续。图 *5* 显示完整的 ValidationFilter 实现。

如果自定义筛选器涉及安全性，则需要从 ReceiveSecurityFilter 或 SendSecurityFilter 派生，这取决于通信的方向。这两个类派生自 SoapFilter，它们提供与消息安全相关的额外方法。ReceiveSecurityFilter 定义一个抽象的 ValidateMessageSecurity 方法，SendSecurityFilter 定义一个抽象的 SecureMessage 方法。这两个方法为您提供 Microsoft.Web.Services3.Security.Security 对象，您可以使用该对象执行命令性安全任务。查看文章"*Using the Policy Framework to Secure Web Apps with Web Services Enhancements*"以了解有关如何实现安全筛选器的完整示例。

为了引入现有的自定义筛选器，您必须进行的改动相当简单：只需更改基类名，将 ProcessMessage 的返回类型修改为 SoapFilterResult，并相应地修改实现。

ValidationFilter 示例已经准备好，您现在需要将它插入服务的接收管道。在 WSE 2.0 中，通过 <filters> 配置元素或在运行时通过编程方式将一个实例插入管道中，您可以实现该操作。在 WSE 3.0 中，使用策略框架配置管道。首先，您需要编写一个自定义策略断言类来包装筛选器。

返回页首

迁移自定义策略断言

在 WSE 2.0 和 WSE 3.0 中，策略断言是从 PolicyAssertion 派生的类。然而，PolicyAssertion 类的结构及其在管道中所起的作用完全不同。

在 WSE 2.0 中，PolicyAssertion 类的作用是验证并实现对传入消息和传出消息的要求。在 WSE 3.0 中，PolicyAssertion 类的作用是充当 SoapFilter 对象的工厂，在本例中，SoapFilter 对象验证并实现要求。WSE 3.0 在客户端/服务中构建发送/接收管道时，调用配置的策略中的每个 PolicyAssertion 类。

在 WSE 3.0 中，通过从 PolicyAssertion 派生一个新类，然后覆盖其抽象成员，您可以编写一个自定义策略断言。图 *6* 中的代码说明如何开始 ValidationAssertion。

PolicyAssertion 定义四个必须实现的抽象方法：CreateClientInputFilter、CreateClientOutputFilter、CreateServiceInputFilter 和 CreateServiceOutputFilter。当策略断言在某个客户端策略中使用时，WSE 调用 CreateClientXxx 格式，其中，构建接收管道时 *Xxx* 为 InputFilter，创建发送管道时，*Xxx* 为 OutputFilter。当该断言用于某个服务策略时，调用采用 CreateServiceXxx 格式。

这些方法为您提供了一个机会：在运行时构建代码时可以将代码插入各种 WSE 管道中。请注意，每个方法返回一个 SoapFilter。WSE 使用返回的 SoapFilter 对象，并按照其在策略中的发生顺序将其放在对应的管道中。因此，策略中断言的顺序很重要。第一个断言始终在距离管道中应用程序最近的地方结束，而最后一个断言始终在靠近网络的地方结束（请参见图 *4*）。

您不必从每个方法返回一个 SoapFilter，除非您的确在相应的 WSE 管道中需要它。例如，如果特定的策略断言绝不会应用于客户端，您就可以从 CreateClientInputFilter 和 CreateClientOutputFilter 方法返回空。在验证示例中，您只需注意验证服务接收到的消息，因此，CreateServiceInputFilter 是唯一一个需要返回 ValidationFilter 对象的工厂。

在这个特别的示例中，您需要允许 ValidationAssertion 类的用户指定验证所需架构的位置。因为如此，您可以将这些架构加载到一个 XmlSchemaSet 对

象中，并将该对象提供给新的 ValidationFilter。实现此目的的一种方法是让用户在构造函数中指定位置。图 7 中的代码显示 ValidationAssertion 示例的一个示例实现。

如果现有的 WSE 2.0 解决方案中有自定义的策略断言，您需要将它们作为 SoapFilter 实现进行重写，然后用一个 PolicyAssertion 类（如图 7 所示）包装它们。

一旦自定义的 PolicyAssertion 类就绪，就可以开始在客户端和服务策略中使用它们。在 WSE 2.0 中，您必须使用 WS-Policy 语法定义策略。然而，新的 WSE 3.0 策略框架允许您在代码中以命令方式定义策略，或者使用 XML 词汇以声明方式定义策略。

返回页首

在代码中定义策略

在 WSE 3.0 中，Policy 类对一个包含断言的策略建模。它有一个名为 Assertions 的公共属性（Collection<PolicyAssertion> 类型的），用于管理一个 PolicyAssertion 实例的集合。通过实例化一个 Policy 对象并将 PolicyAssertion 对象添加到它的 Assertions 集合中，您可以通过编程方式定义一个策略，如下所示：

```
Policy myPolicy = new Policy();
myPolicy.Assertions.Add(new RequireActionHeaderAssertion());
myPolicy.Assertions.Add(new RequireSoapHeaderAssertion(
    "MyHeader", "http://example.org/myheader"));
myPolicy.Assertions.Add(new ValidationAssertion(
    HttpContext.Current.Server.MapPath("xsd")));
```

该策略包含三个断言。前两个是内置断言（WSE 3.0 类库的一部分），最后一个是自定义的验证断言。第一个断言要求消息中存在 WS-Addressing<Action> 头。第二个断言要求消息中存在名为 MyHeader 的自定义头。最后一个断言要求消息传递架构验证。这三个断言在 Assertions 集合中的顺序决定了它们在管道中出现的顺序。

您也可以从 Policy 派生一个类，并在其构造函数中添加 PolicyAssertion 对象，而不是在每次需要时从头开始构建策略，如下所示：

```
public class MyPolicy : Policy
{
    public MyPolicy()
    {
        this.Assertions.Add(new RequireActionHeaderAssertion());
        this.Assertions.Add(new RequireSoapHeaderAssertion("MyHeader",
            "http://example.org/myheader"));
        this.Assertions.Add(new ValidationAssertion(
            HttpContext.Current.Server.MapPath("xsd")));
    }
}
```

这段代码将策略定义包装到一个类中，用户可以仅在需要该类时对其进行实例化。既然已经有了一个策略，您就可以将它应用于服务或客户端代理。

返回页首

将策略应用于服务

通过在您的 Web 服务类上使用新的 [Policy] 属性，您将一种策略类型应用于特定的服务，这就是我在前面提到过的新映射机制。该属性可应用于 ASMX 派生的类或 SoapReceiver 派生的类。下面是一个将 MyPolicy 与简单的 ASMX 类相关联的示例：

```
[Policy(typeof(MyPolicy))]
[WebService(Namespace = "http://example.org/math")]
public class MathService: WebService
{
    [WebMethod]
    public double Add(double x, double y) { return x + y; }
    ...
```

这段代码告诉 WSE，MyPolicy 类应该用于为该服务构建管道。为使这段代码生效，您需要使用 WSE 设置工具在应用程序中启用 WSE 3.0。由于本例中的服务是一个 ASP.NET Web 站点，因此，您还需要启用 WSE Soap Protocol Factory（请参见 General 选项卡）。但是，您不需要在 WSE 设置工具中启用策略（通过 Policy 选项卡），只需在使用基于 XML 的策略文件时启用策略。在本例中，我已经在代码中定义了策略，并将其与 Web 服务类相关联，这样 WSE 就具备它所需要的一切功能了。

返回页首

将策略应用于客户端

您可以使用以下任何一种技术将策略应用于客户端。第一种技术是调用您将在所有 WSE 3.0 生成的代理类上发现的新 SetPolicy 方法。下面是一个示例：

```
MathServiceWse proxy = new MathServiceWse();
proxy.SetPolicy(new MyPolicy());
double sum = proxy.Add(2, 3);
```

如果您希望将策略与代理类永久关联，就可以像在服务上一样用 [Policy] 属性来标注它，这样就无需调用 SetPolicy 了。下面是该技术的一个示例：

```
// attributes omitted for clarity
[Policy(typeof(MyPolicy))]
[WebServiceBinding(Name="MathService",
   Namespace="http://example.org/math")]
public partial class MathServiceWse : WebServicesClientProtocol { ... }
```

在代码中定义策略以及像上面这样应用策略的能力提供了很大的运行时灵活性。这允许您动态构建策略，根据应用程序状态或配置设置选择是包括断言还是省略断言。

不难想象您如何使用前面几部分中介绍的技术在应用程序配置文件中指定整个策略断言列表，以及如何动态加载它们。这将允许您对策略本身进行更改，无需重新编译和重新部署代码。为了提供这种灵活性，WSE 3.0 提供了一个内置解决方案，用于在 XML 中定义策略。

返回页首

在 **XML** 中定义策略**XML**

与 WSE 2.0 不同，WSE 3.0 不使用 WS-Policy 词汇表示策略。相反，它使用以下词汇：

```
<policies
   xmlns="http://schemas.microsoft.com/
   wse/2005/06/policy">
  <policy name="Policy1">
     <!-- assertions go here -->
  </policy>
  <policy name="Policy2">
     <!-- assertions go here -->
  </policy>
     <!-- additional policies can go here -->
</policies>
```

策略文件必须包含一个来自 http://schemas.microsoft.com/wse/2005/06/policy 命名空间的根 <policies> 元素。它包含一个 <policy> 元素（来自相同的命名空间）的列表。您使用名称属性标识每个 <policy>，即，您将使用该名称在代码中引用策略。

每个 <policy> 元素包含一个断言元素的列表。每个断言元素对应一个 PolicyAssertion 派生的类。下面的示例定义一个名为 MyXmlPolicy 的策略，该策略以前使用了两个相同的内置断言：

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <policy name="MyXmlPolicy">
    <requireActionHeader/>
    <requireSoapHeader name="MyHeader"
        namespace="http://example.org/myheader"/>
  </policy>
  <!-- additional policies can go here -->
  ...
</policies>
```

因为我在使用内置断言，所以不需要指定这些元素代表哪些 PolicyAssertion 类，因为 WSE 知道。如果您正在 Visual Studio 2005 中编辑策略文件，您甚至能获得智能感知®提示，向您显示 IDE 了解的所有内置断言元素（请参见图 8）。WSE 3.0 还提供了一个新向导，用于生成该格式的安全策略；这是向前迁移 WSE 2.0 安全策略的最简单的方法。

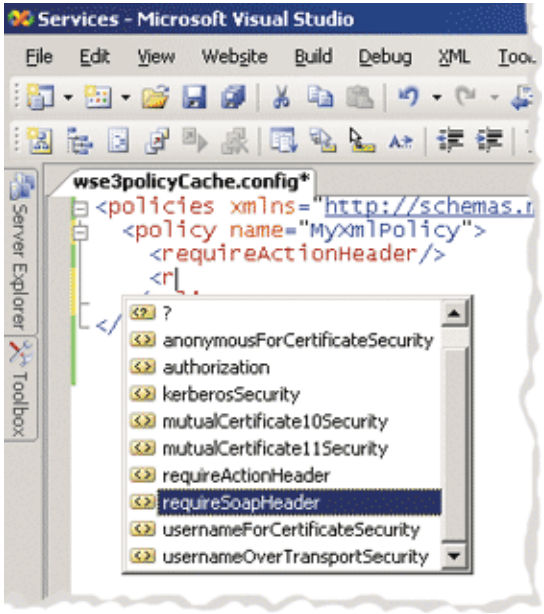

图 8 策略智能感知

返回页首

将断言元素映射到代码

您可以显式控制断言元素名和相应的 PolicyAssertion 类之间的映射。新的 <extension> 元素将一个元素名映射为一个 PolicyAssertion 类。以下代码显示如何针对自定义的 ValidationAssertion 实现该操作：

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <extensions>
    <extension name="validation"
        type="ValidationAssertion, ValidationAssertion" />
  </extensions>
  <policy name="ValidationPolicy">
    <requireActionHeader/>
    <requireSoapHeader name="MyHeader"
        namespace="http://example.org/myheader"/>
    <validation schemaLocation="C:\services\mathservice\xsd"/>
  </policy>
</policies>
```

您必须更新 ValidationAssertion 类才能从 <validation> 元素读取 schemaLocation 属性。PolicyAssertion 为此定义了 ReadXml 方法。图 9 显示如何在 ValidationAssertion 类中实现 ReadXml。

该实现希望在名为 schemaLocation 的属性中找到架构位置。注意它如何推动阅读器，以便它将针对下一个断言进行适当的定位。一切已经就绪，用户现在可以选择是在代码中使用该断言还是在基于 XML 的策略中使用该断言。

返回页首

配置并应用 **XML** 策略

要将一个基于 XML 的策略应用于您的服务和客户端，您必须先用策略文件的位置配置应用程序。通过使用 WSE 设置工具中的 Policy 选项卡，您可以达到此目的，或者您可以只将以下部分添加到您的应用程序配置文件中：

```
<configuration>
  ...
  <microsoft.web.services3>
    <policy fileName="wse3policyCache.config" />
  </microsoft.web.services3>
</configuration>
```

一旦 WSE 知道在何处查找基于 XML 的策略，您就可以在使用 [Policy] 属性时，通过指定策略名（在策略文件中）将其应用于您的代码，如下所示：

```
[Policy("MyXmlPolicy")]
[WebService(Namespace = "http://example.org/math")]
public class MathService: WebService
{
    [WebMethod]
    public double Add(double x, double y) { return x + y; }
    ...
```

在一个策略类上调用 SetPolicy 时，您可以进行相同的操作，如下所示：

```
MathServiceWse proxy = new MathServiceWse();
proxy.SetPolicy("MyXmlPolicy");
double sum = proxy.Add(2, 3);
```

在 XML 中定义策略并且以这种方式配置代码在灵活性和开发人员效率之间提供最佳的平衡，这可能是大多数开发人员将采用的方法。

如您所见，一旦将自定义筛选器和策略断言迁移到 WSE 3.0，您就必须决定如何定义您的策略（用代码还是用 XML）。然后，必须更新您的客户端/服务代码以应用使用 [Policy] 属性或 SetPolicy 方法的策略，因为映射不再是策略文件本身的一部分。所有这些改变都需要一些手动干预，甚至要重新设计。

返回页首

现在需要做什么？

要了解有关 WSE 3.0 策略框架

在某些情况下，将现有解决方案迁移到 WSE 3.0 需要要比其他情况付出更多的努力。现在，对于更简单的迁移方案

的详细信息，重点关注安全性的新方法，请查看 *MSDN Magazine* 2006 年 2 月那期中 Tomasz Janczuk 的文章"*WSE Security: Protect Your Web Services Through the Extensible Policy Framework in WSE*"。

还要阅读 Mark Fussell 撰写的 *What's New in WSE 3.0*，以及 Keith Brown 撰写的 *Security Features in WSE 3.0*。

您还将发现以下资源很有帮助：*WSE 3.0 Hands-On Lab-Exploring Security* 和 *Web Services Policy Framework (WS-Policy)*。

返回页首

和改进的 WSE 3.0 策略框架，您已经具备了相关的知识（自定义筛选器和策略断言方面存在更困难的迁移问题），那么您就可以准备进行自己的迁移了。

请您将对 Aaron 的问题和意见发送至 *sstation@microsoft.com*。

Aaron Skonnard 是 Pluralsight（Microsoft .NET 培训提供商）的创始人之一。Aaron 著有 Pluralsight 的 Applied Web Services 2.0、Applied BizTalk Server 2006 和 Introducing Windows Communication Foundation 教程。Aaron 花费数年时间开发课程，在会议上演讲，以及教授专业开发人员。他的联系方式为*pluralsight.com/aaron*。

转到原英文页面