Java web 服务: 理解 WS-Policy

**简介：** WS-Policy 提供一个通用结构，用来配置应用到 web 服务的特性和选项。在本[系列](http://www.ibm.com/developerworks/cn/java/j-jws/) 中，您可能看到过它用于 WS-Security 配置，或在其他地方，看到它用于其他扩展技术中，比如 WS-ReliableMessaging。

在 [Java web 服务](http://www.ibm.com/developerworks/cn/java/j-jws/) 系列，您可能已经见到过许多 WS-Policy 和 WS-SecurityPolicy 示例，但是迄今为止，还没有任何关于 WS-Policy 实际工作（至少，是打算 运行 ）的讨论。在文本中，我首先为您提供一点 WS-Policy 的背景知识，弥补您在这方面的不足。然后，您将看到如何在 WSDL 文档中附加策略到服务。最后，我将针对 Axis2、Metro 和 CXF 尝试一些 WS-Policy 配置示例，然后告诉您它们在实践中如何工作。（在 [下载](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#download) 部分获取完整样例代码。）

**WS-Policy 基础**

WS-Policy 定义了一个简单 XML 结构，由 4 个不同元素和一对属性组成。根据 WS-Policy 解释，这些元素和属性提供一种方法来组织和合并任意复杂度的策略断言（policy assertions）。为了定义构成策略的真实断言，您需要使用特定扩展，比如 WS-SecurityPolicy，而不是 WS-Policy 本身。

## WS-Policy 版本

本文中的 WS-Policy 详细资料是专门用于 W3C 发布的 “官方” WS-Policy 1.5，但是相同的通用规则也适用于仍然广泛使用的旧 “提交” 版本。对于其他 WS-\* 技术，您可以通过检查 XML 名称空间了解文档使用的版本。WS-Policy 1.5 的名称空间是http://www.w3.org/ns/ws-policy；提交版是http://schemas.xmlsoap.org/ws/2004/09/policy。在本文中，我使用 http://www.w3.org/ns/ws-policy 名称空间，wsp 作为前缀。

为了方便起见，WS-Policy 定义了一个标准形式的策略表达式和一组可以创建更紧凑的策略表达式的规则。标准形式可能有点繁琐，尽管 WS-Policy 推荐 “应该将策略表达式的标准形式用于实践”，但是，多数策略文件撰写者倾向于使用（至少部分使用）紧凑表达式规则，使文档更人性化。策略文档的解释是基于标准形式的，因此我首先对它做一介绍。

**标准形式表达式**

标准形式策略表达式最多使用 3 个元素，必须按照特定顺序嵌套。最外层的一般是 <wsp:Policy>，它必须且只能包含一个<wsp:ExactlyOne> 子元素。而嵌套的 <wsp:ExactlyOne> 元素，可以包含任意多（可以为 0）的 <wsp:All> 子元素。因此最简单的标准形式策略表达式是 <wsp:Policy><wsp:ExactlyOne/></wsp:Policy>。

任何标准形式的策略断言必须被嵌套在 <wsp:All> 元素中。这些策略断言自身可以包含策略表达式。在本系列之前的文章中，您可能见过 WS-SecurityPolicy 断言包含嵌套的策略表达式。在一个标准形式策略表达式中，所有这些嵌套的策略断言必须是标准形式。（实际上，它们必须在一个更为严格的标准形式子集中，子集中每个 <wsp:ExactlyOne> 元素，除了最外层的那个，都有且仅有一个 <wsp:All> 子元素。） 清单 1 是嵌套策略表达式示例的一个片段，以标准形式表示：

**清单 1. WS-SecurityPolicy 嵌套策略表达式摘录**

|  |
| --- |
| <wsp:Policy>  <wsp:ExactlyOne>  <wsp:All>  <sp:AsymmetricBinding  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">  <wsp:Policy>  <wsp:ExactlyOne>  <wsp:All>  <sp:InitiatorToken>  <wsp:Policy>  <wsp:ExactlyOne>  <wsp:All>  <sp:X509Token  sp:IncludeToken=".../IncludeToken/AlwaysToRecipient">  <wsp:Policy>  <wsp:ExactlyOne>  <wsp:All>  <sp:RequireThumbprintReference/>  </wsp:All>  </wsp:ExactlyOne>  </wsp:Policy>  </sp:X509Token>  </wsp:All>  </wsp:ExactlyOne>  </wsp:Policy>  </sp:InitiatorToken>  ... |

尽管有多层嵌套元素，但语法结构是比较简单的。在一个策略表达式的标准形式中，<wsp:Policy> 只是策略表达式的一个包装，而顶层策略表达式才是附加名称或标识符之处。嵌套的 <wsp:ExactlyOne> 元素表示嵌套的 <wsp:All> 元素代表的替代项的一个 or 组合（因此，如果嵌套的替代项其中的一个符合标准，那整个策略表达式就是符合标准的）。每一个<wsp:All> 元素表示嵌套策略断言的一个 and 组合（因此，只有所有断言符合标准，替代项才符合标准）。

**非标准化策略**

策略表达式不需要都是标准形式的，非标准策略表达式可以包含嵌套替代项（一个 <wsp:ExactlyOne> 元素的多个<wsp:All> 子元素），也可以使用下一节将要讨论的紧凑策略表达式。

如果您学习过逻辑原理，您就会知道标准表示形式（没有嵌套替代项）等同于逻辑表达式的析取范式（disjunctive normal form）。实际上，策略表达式只是使用一个尖括号格式的逻辑表达式，断言是子句。在逻辑理论中，任何逻辑表达式都可以被转换成析取范式 ，而同样的原则适用于用嵌套替代项编写的策略表达式 — 仅在顶层可以被扩展成一个含有替代项的标准格式表示。标准形式策略表达式的一个主要优势是，从程序语法上检查两个策略的兼容性变得很容易 — 如果两个标准策略是兼容的，它们将有一个或多个顶层 <wsp:All> 元素，包含相同的断言集。

**紧凑策略表达式**

标准形式策略表达式是冗长的，特别是如果它包含嵌套替代项。WS-Policy 定义了一些选项，您可以用来创建比标准形式更简明的策略表达式，使策略更易理解。当以紧凑形式 本身使用这些选项时，WS-Policy 文档有时候会变得混乱。实际上，一个策略的多个紧凑表达式等价于一个标准表达式。在本文中，我只是用紧凑表达式 指代使用一个或多个选项的策略。

紧凑表达式选项的一个特性是，拥有通过基本策略元素（在策略术语中称为操作，因为每个元素暗含嵌套断言的一个具体解释）任意次序嵌套来表达策略的能力。嵌套规则也定义一种直接使用的 <wsp:Policy> 元素（不需要标准形式中必须有的 <wsp:ExactlyOne> 子元素）的解释等价于一个 <wsp:All>，这可能是最广泛使用的紧凑表达式特性。

清单 2 显示了（[清单 1](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#listing1) 中）同一个策略的紧凑表达式，使用 <wsp:Policy> 表示。这个比第一个长度减少了一半，而且对大多数人来说都很容易理解 。

**清单 2. 紧凑形式的简单策略**

|  |
| --- |
| <wsp:Policy>  <sp:AsymmetricBinding  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">  <wsp:Policy>  <sp:InitiatorToken>  <wsp:Policy>  <sp:X509Token  sp:IncludeToken=".../IncludeToken/AlwaysToRecipient">  <wsp:Policy>  <sp:RequireThumbprintReference/>  </wsp:Policy>  </sp:X509Token>  </wsp:Policy>  </sp:InitiatorToken>  ... |

WS-Policy 定义了一个变换集，您可以用来将使用紧凑选项的策略表达式转换成标准形式，因此没有理由直接使用标准形式。使用计算机将紧凑表达式转换成标准表达式比起人工解析标准表达式容易得多。

**策略包含**

除了简化元素嵌入，紧凑表达式也提供一种引用和重用策略表达式的方法。您可以使用第 4 个 WS-Policy 元素<wsp:PolicyReference> 来实现。<wsp:PolicyReference> 元素可以在任何策略断言出现的地方出现。引用的策略表达式被策略引用（Policy reference）有效替代（技术上是使用 <wsp:All> 元素替代引用的 <wsp:Policy> 元素。）

清单 3 说明了策略包含的使用，显示使用一个独立的策略表达式和一个策略引用重构 [清单 2](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#listing2) 中的策略表达式：

**清单 3. 策略引用**

|  |
| --- |
| <!-- Client X.509 token policy assertion. -->  <wsp:Policy wsu:Id="ClientX509"  xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd">  <sp:InitiatorToken>  <wsp:Policy>  <sp:X509Token  sp:IncludeToken=".../IncludeToken/AlwaysToRecipient">  <wsp:Policy>  <sp:RequireThumbprintReference/>  </wsp:Policy>  </sp:X509Token>  </wsp:Policy>  </sp:InitiatorToken>  </wsp:Policy>  <wsp:Policy>  <sp:AsymmetricBinding  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">  <wsp:Policy>  <wsp:PolicyReference URI="#ClientX509"/>  ... |

策略引用可用于本地策略表达式，如 [清单 3](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#listing3) 所示，也可用于外部策略表达式。对于外部策略表达式，引用的 URI 属性通常提供外部策略的真实 URL。

正如您在本文后面提供的策略测试示例中所看到的，策略包含目前不是普遍支持的，这限制了这个原本很出色的特性的使用。

**策略替代项**

正如您之前看到的，WS-Policy 结构支持选择替代项作为策略的一部分，使用 <wsp:ExactlyOne> 元素。有了紧凑表达式，您就可以（至少理论上）使用特定属性创建选项。根据 WS-Policy 建议，您可以向任何策略断言添加 wsp:Optional="true"属性，使断言成为一个选择而不是需求，即使断言是 <wsp:All> 或 <wsp:Policy> 元素的子元素。

策略替代项似乎是一个很有用的特性，但是要给出一个非凡的运行示例也比较困难。特性用于实践的情况通常能够对安全性的一个特定组件进行处理，比如一个 UsernameToken，可选。较复杂的替代项，比如允许客户以 UsernameToken 或 X.509 证书格式提供验证，当前 WS-SecurityPolicy 实现功能超出了本书范围。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#ibm-pcon)

**策略附加**

在 WSDL 1.1 （有些过时，但仍然是最广泛使用的服务定义格式），服务定义使用层次结构，第一层（底层）由<wsdl:message> 元素构成，定义消息的 XML 结构到服务，或者从服务定义。第二层是 <wsdl:portType> 元素，定义操作集，每一个操作由输入、输出或默认消息指定。第 3 层是 <wsdl:binding> 元素，使用 <wsdl:portType> 将一个特定消息协议和访问方法联系到一起。第 4 层 <wsdl:portType> 元素格式的服务器端定义，这指定 <wsdl:binding> 的访问路径。

## WSDL 模式

WSDL 1.1 很早之前就开始涉及 XML 架构定义并且其间进行过多次修改。 最初的 WSDL 1.1 提议包括一个 WSDL XML 文档如何构建的文本描述和一个 XML 模式定义。不幸的是，提供您的 XML 模式和文本不匹配。在稍后的修订版中，对此作了校正，但是 WSDL 1.1 文档不能更新，反映不出这一修改。后来，WS-I Basic Profile 小组决定对 WSDL 模式进行更多的修改，所以，它似乎创建了这个不稳定模式的最佳实践版本。写入版本模式中的文档通常同其他版本是不兼容的（尽管使用相同的名称空间），但幸运的是，多数 web 服务工具基本上都忽略这些模式，接受看起来合理的内容。在 [参考资料](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#resources) 部分，获取更多 WSDL 模式的链接。

WS-Policy 允许您在几个点上向 WSDL 服务定义附加策略，这几个点不能精确匹配定义的层。然而，层代表的是服务定义的一个逻辑结构，WS-Policy 更关注消息和消息分组。WS-Policy 使用的 4 个消息分组级别是：

* **消息**：策略适用于一个特定消息（如果策略是通过<wsdl:message> 元素附加的，消息可用于任何地方；或者，如果在 <wsdl:portType> 或 <wsdl:binding> 元素中策略是通过操作的 input/output/fault 定义附加的，当消息被一个特定操作所用时）。
* **操作**：策略适用于一个特定操作的所有消息交换（在<wsdl:binding> 或 <wsdl:portType> 中，通过<wsdl:operation> 元素附加的策略）。
* **端点**：策略适用于一个特定服务绑定的所有消息交换（通过 <wsdl:port> 或 <wsdl:binding> 附加的策略），或者 适用于基于一个特定端口类型的所有服务绑定的消息交换（附加到 <wsdl:portType> 的策略）。
* **服务**：策略适用于所有端点和所有与服务关联的操作（在 <wsdl:service> 元素附加的策略 ）。

WSDL 所用的主要策略附加机制同另一个策略 — 在 [策略包含](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#inclusion) 部分描述的 <wsp:PolicyReference> 元素 — 中引用策略的机制是一样的。WS-Policy 元素可以被添加作为任何之前列出的 WSDL 元素的一个子元素，指定在消息分组层应用的策略。您也可以使用恰当内容直接嵌入一个策略作为 <wsp:Policy> 元素，但是通常最好使用策略引用，便于 WSDL 结构保持整洁。

在消息分组一层中应用的策略，可被下一层继承，在 <wsp:All> 元素中合并。这使得实际（在 WS-Policy 术语中是有效）适用于每个消息的策略成为所有策略（适用于消息、操作、端点和服务层）的连接点。因此，策略不是由消息本身决定的，而是由使用消息的上下文决定的。

清单 4 说明了这些操作如何进行，显示了使用策略引用定义 WSDL 的框架：

**清单 4. 策略附加示例**

|  |
| --- |
| <wsdl:binding name="LibrarySoapBinding" type="wns:Library">  <wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy" URI="#UsernameToken"/>  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>    <wsdl:operation name="addBook">    <wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy" URI="#AsymmEncr"/>  ...  </wsdl:operation>  </wsdl:binding> |

在这种情况下，需要 UsernameToken 的策略被附加在 <wsdl:binding> 层，一个需要不对称消息加密的附加策略被附加到addBook 操作，定义为绑定的一部分。假设这个示例显示了全套 WSDL 策略引用，UsernameToken 永远需要，但是消息签名仅用于 addBook 操作。

作为使用 <wsp:PolicyReference> 直接从 WSDL 元素引用策略的一种替代方法，您也可以使用 wsp:PolicyURIs 属性。您可以将该属性添加到策略被添加到的任何 WSDL 元素。本质上，这同使用 <wsp:PolicyReference> 子元素功能一样。

**WS-SecurityPolicy 附加**

WS-SecurityPolicy 指定消息分组级别，其中不同类型的策略断言可以被附加到一个服务描述中。例如，<sp:TransportBinding> 断言通常用来指定安全性，只能被添加在端点层，而 <sp:AsymmetricBinding> 和<sp:SymmetricBinding> 断言通常用来指定消息加密和签名，只用于端点或操作层。

尽管 <sp:AsymmetricBinding> 或 <sp:SymmetricBinding> 不能在消息层指定，但您可以 在消息层指定加密或者签名的组件。这意味至少从理论上可以在消息令牌基础上指定加密或签名。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#ibm-pcon)

**策略示例**

现在，您已经学习了 WS-Policy 的基本原理，以及它如何同 WSDL 协作，是时候使用这些原理尝试一些策略示例了。在之前的文章中，我用 3 个主要 Java 开源 web 服务栈：Axis2、Metro 和 CXF，对这个代码进行过试验了。

清单 5 是一个示例（effective1.wsdl 在样例代码 [下载](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#download) 部分），在一个 WSDL 服务定义中使用 3 层策略附件。所用的 3 个策略是：

* UsernameToken：需要一个带有散列密码的 UsernameToken。
* SymmEncr：需要使用一个客户端生成您的密钥进行对称加密。
* EncrBody：需要对消息体进行加密。

**清单 5. 有效策略示例**

|  |
| --- |
| <wsdl:definitions targetNamespace="http://ws.sosnoski.com/library/wsdl"...>    <wsp:Policy wsu:Id="UsernameToken" xmlns:wsp="http://www.w3.org/ns/ws-policy"...>  <sp:SupportingTokens>  <wsp:Policy>  <sp:UsernameToken sp:IncludeToken=".../IncludeToken/AlwaysToRecipient">  <wsp:Policy>  <sp:HashPassword/>  </wsp:Policy>  </sp:UsernameToken>  </wsp:Policy>  </sp:SupportingTokens>  </wsp:Policy>    <wsp:Policy wsu:Id="SymmEncr" xmlns:wsp="http://www.w3.org/ns/ws-policy"...>  <sp:SymmetricBinding>  <wsp:Policy>  <sp:ProtectionToken>  <wsp:Policy>  <sp:X509Token sp:IncludeToken=".../IncludeToken/Never">  ...  </sp:X509Token>  </wsp:Policy>  </sp:ProtectionToken>  ...  </wsp:Policy>  </sp:SymmetricBinding>  ...  </wsp:Policy>    <wsp:Policy wsu:Id="EncrBody" xmlns:wsp="http://www.w3.org/ns/ws-policy"...>  <sp:EncryptedParts>  <sp:Body/>  </sp:EncryptedParts>  </wsp:Policy>  ...  <wsdl:binding name="LibrarySoapBinding" type="wns:Library">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#UsernameToken"/>**  ...  <wsdl:operation name="getBook">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#SymmEncr"/>**  <wsdlsoap:operation soapAction="urn:getBook"/>  <wsdl:input name="getBookRequest">  <wsdlsoap:body use="literal"/>  </wsdl:input>  <wsdl:output name="getBookResponse">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#EncrBody"/>**  <wsdlsoap:body use="literal"/>  </wsdl:output>  </wsdl:operation>  <wsdl:operation name="getBooksByType">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#SymmEncr"/>**  <wsdlsoap:operation soapAction="urn:getBooksByType"/>  <wsdl:input name="getBooksByTypeRequest">  <wsdlsoap:body use="literal"/>  </wsdl:input>  <wsdl:output name="getBooksByTypeResponse">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#EncrBody"/>**  <wsdlsoap:body use="literal"/>  </wsdl:output>  </wsdl:operation>  <wsdl:operation name="getTypes">  ...  </wsdl:operation>  <wsdl:operation name="addBook">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#SymmEncr"/>**  <wsdlsoap:operation soapAction="urn:addBook"/>  <wsdl:input name="addBookRequest">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#EncrBody"/>**  <wsdlsoap:body use="literal"/>  </wsdl:input>  <wsdl:output name="addBookResponse">  <wsdlsoap:body use="literal"/>  </wsdl:output>  <wsdl:fault name="addDuplicateFault">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#EncrBody"/>**  <wsdlsoap:fault name="addDuplicateFault" use="literal"/>  </wsdl:fault>  </wsdl:operation>  </wsdl:binding>  ...  </wsdl:definitions> |

[清单 5](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#listing5) 的 WSDL 文件中，策略引用（黑体部分）将 UsernameToken 策略附加到 <wsdl:binding>，以至于 UsernameToken 对所有操作都是必须的。SymmEncr 策略被附加到交换图书信息的所有操作的一个 <wsdl:operation> 中，而 EncrBody 策略被附加到包含图书信息的消息中 — 以至于图书信息总是被加密发送的。

这是一个很麻烦的策略，因为它需要客户生成一个密钥，并使用请求消息将这个密钥发送到客户端，尽管密钥仅作加密用。Axis2 1.5.2 和 Metro 2.0.1 都不能完全处理这种情况。Axis2 使用 WS-Policy 1.5 策略名称空间完全忽略策略组件。当我改为使用提交名称空间时，Axis2 在客户端代码中生成一个合理的配置 — 但是在运行时，在客户端和服务器端同时忽略策略时，运行不安全。Metro 使用一个 UsernameToken 生成一个请求消息，但是没有密钥信息。

CXF 2.3.0 则表现得很好，可以正确处理策略，但以下两个案例除外。addBook 操作成功时，响应消息不使用加密。CXF 服务器可以正确处理这种情况，但是在处理响应时客户端会抛出一个异常。另一个 CXF 错误也出现在使用 addBook 操作时，但这次是在请求失败时，返回的应用程序错误响应应该使用加密，但是 CXF 服务器发送它时不使用加密，而且客户端也毫无异议地接受了。

[清单 5](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#listing5) 中的策略演示了选择性使用消息加密，仅加密那些包含订单信息的消息。然而，这个策略使用对称加密。能够使用非对称加密处理同类型的事情就太好了，其中客户端有自己的证书（特别是当您想要签署发送者证书消息时）。清单 6 是专为此设计的示例（在 [下载](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#download) 部分下载 effective2.wsdl）。该示例只使用 WSDL 的两个策略：

* AsymmBinding：需要非对称加密，使用双重证书。
* SignBody：需要消息体签名。

**清单 6. 非对称签署示例**

|  |
| --- |
| <wsdl:definitions targetNamespace="http://ws.sosnoski.com/library/wsdl"...>  <wsp:Policy wsu:Id="AsymmBinding" xmlns:wsp="http://www.w3.org/ns/ws-policy" ...>  <sp:AsymmetricBinding>  <wsp:Policy>  <sp:InitiatorToken>  <wsp:Policy>  <sp:X509Token sp:IncludeToken=".../IncludeToken/AlwaysToRecipient">  ...  </sp:X509Token>  </wsp:Policy>  </sp:InitiatorToken>  <sp:RecipientToken>  <wsp:Policy>  <sp:X509Token sp:IncludeToken=".../IncludeToken/Never">  ...  </sp:X509Token>  </wsp:Policy>  </sp:RecipientToken>  ...  </wsp:Policy>  </sp:AsymmetricBinding>  </wsp:Policy>    <wsp:Policy wsu:Id="SignBody" xmlns:wsp="http://www.w3.org/ns/ws-policy" ...>  <sp:SignedParts>  <sp:Body/>  </sp:SignedParts>  </wsp:Policy>  ...  <wsdl:binding name="LibrarySoapBinding" type="wns:Library">  ...  <wsdl:operation name="getBook">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#AsymmBinding"/>**  <wsdlsoap:operation soapAction="urn:getBook"/>  <wsdl:input name="getBookRequest">  <wsdlsoap:body use="literal"/>  </wsdl:input>  <wsdl:output name="getBookResponse">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#SignBody"/>**  <wsdlsoap:body use="literal"/>  </wsdl:output>  </wsdl:operation>  <wsdl:operation name="getBooksByType">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#AsymmBinding"/>**  <wsdlsoap:operation soapAction="urn:getBooksByType"/>  <wsdl:input name="getBooksByTypeRequest">  <wsdlsoap:body use="literal"/>  </wsdl:input>  <wsdl:output name="getBooksByTypeResponse">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#SignBody"/>**  <wsdlsoap:body use="literal"/>  </wsdl:output>  </wsdl:operation>  <wsdl:operation name="getTypes">  ...  </wsdl:operation>  <wsdl:operation name="addBook">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#AsymmBinding"/>**  <wsdlsoap:operation soapAction="urn:addBook"/>  <wsdl:input name="addBookRequest">  **<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"**  **URI="#SignBody"/>**  <wsdlsoap:body use="literal"/>  </wsdl:input>  <wsdl:output name="addBookResponse">  <wsdlsoap:body use="literal"/>  </wsdl:output>  <wsdl:fault name="addDuplicateFault">  <wsdlsoap:fault name="addDuplicateFault" use="literal"/>  </wsdl:fault>  </wsdl:operation>  </wsdl:binding>  ...  </wsdl:definitions> |

[清单 6](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#listing6) 中的示例使用非对称加密技术签署所有消息，提供图书信息。这意味着 getBook 和 getBooksByType 响应消息应该由服务器签署，而 addBook 请求消息应该由客户端签署。因为这是使用非对象加密技术，而每一端有自己的证书和私有密钥，比起 [清单 5](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#listing5) 对称加密技术示例，它处理起来更为简单。

## 外部策略引用

外部策略引用不能很好地融入提供的示例结构，但是我想试试看。对于我的测试，我使用两个关系引用，<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy" URI="./asymm-binding-policy.xml"/> 形式的关系引用和位于 web 服务器的绝对 URL。Axis2 和 Metro 都不能解决这类策略引用，但它们在 CXF 上可以正确运行。

在早期示例中 Axis2 同样失败，当我修改提交名称空间时，除了在两端都忽略策略组件，还完全忽略使用 WS-Policy 1.5 策略名称空间的策略组件，并生成客户端策略。（对于使用 Axis2 运行的高效策略简单示例，见 “[WS-Security 的细粒度使用](http://www.ibm.com/developerworks/cn/java/j-jws7.html)”。）

Metro 不能运行此策略配置，在客户端使用第一个请求抛出一个NullPointerException。经过一番调查，我发现，如果不仅仅在操作和消息层附加策略，在端点层也附加一个策略，那么问题将会变得更糟糕。Metro 示例包括一个 effective3.wsdl，其中有一个来自 <wsdl:binding> 的 #AsymmBinding 策略引用，该示例运行没有问题（尽管，在对称案例中像 CXF，但是 Metro 不能应用安全性到应用程序错误响应）。

使用 [清单 6](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#listing6) 策略配置，CXF 也可能失败，但是在目前代码中没有轻松的工作环境。在发送 getBook 请求消息时，CXF 客户端代码不能正确生成一个签名，而服务器端代码也在处理请求消息时失败。因此，当 AsymmetricBinding 在作用域中时，它和当前 CXF 代码一样坚持生成签名，即使没有什么需要签署。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-jws18/index.html#ibm-pcon)

**策略包装**

WS-Policy 定义一个灵活且功能强大的结构，用于表达任何形式的约束。不幸的是，web 服务栈使用的 WS-Policy 和 WS-SecurityPolicy 处理的实施不能实现较大的灵活性。因为缺乏实现支持，WS-Policy 的其他有用特性不能用于想要同全方位的 web 服务栈交互的 web 服务。

基本高效策略处理，在 WSDL 服务定义中策略依此添加到不同端点，是一个策略和 WSDL 设计的核心特性，应该用于适合您的服务。但是这类配置的一个潜在有用特性 — 可以根据情况选择性地签署和加密消息 — 不能可靠地进行工作。考虑到这一限制，如果您想要最好的互操作性，目前最好同时在 <wsdl:binding> 和 <wsdl:operation> 层附加策略。

外部策略对于 SOA 类环境是特别有用的，在这类环境中，可以构建一组通用策略，便于整个组织使用，且每个服务可以引用满足其需求的策略。尽管，目前这一特性并不被所有开源 Java web 服务栈所支持（只有 Apache CXF 可以正确处理），大型组织可能想要使用这一特性，且限制 web 服务实现来使用一个支持它的栈（不论开源还是商业）。通过其他方法您也可能达到同样的效果，比如通过使用 WSDL。