Java Web サービス: WS-Policy について理解する

WS-Policy の詳細について学び、どの部分が 3 つのオープンソースの Java Web サービス・スタックで動作するかを知る

Dennis Sosnoski
Architecture Consultant and Trainer
Sosnoski Software Solutions, Inc.

2010年 11月 02日

WS-Policy は、Web サービスに適用する機能とオプションを構成するための汎用構造を提供します。皆さんは、この連載の WS-Security 構成で使用されている例の他、WS-ReliableMessaging などの他の拡張技術でも WS-Policy を目にしたことがあるでしょう。今回の記事では、WS-Policy 文書の構造、そして WSDL (Web Service Description Language) でポリシーをサービスに関連付ける方法を説明し、サンプルのセキュリティー構成を Apache Axis2、Metro、Apache CXF のそれぞれで試します。

このシリーズの他の記事を見る

この連載について

Web サービスは、エンタープライズ・コンピューティングにおいて Java 技術が担う重大な役割の一部です。この連載では、XML および Web サービスのコンサルタントである Dennis Sosnoski が、Web サービスを使用する Java 開発者にとって重要になる主要なフレームワークと技術について説明します。この連載から、現場での最新の開発情報を入手して、それらを皆さんのプログラミング・プロジェクトにどのように利用できるかを知っておいてください。

連載「Java Web サービス」では WS-Policy と WS-SecurityPolicy の例をいくつも記載してきましたが、これまでのところ、WS-Policy が実際にどのような動作をするか (少なくとも、意図された動作) については説明していません。今回の記事ではまず WS-Policy の基礎について簡単に説明して、これまで説明していなかった部分を補います。その後、WSDL 文書でポリシーをサービスに関連付ける方法を説明し、最後にサンプル WS-Policy 構成を Axis2、Metro、CXF のそれぞれで試して実際にどのように動作するかを見てみます (完全なサンプル・コードを入手するには「ダウンロード」セクションを参照してください)。

WS-Policy の基礎知識

WS-Policy では、4 種類の要素と 1 組の属性からなる単純な XML 構造を定義します。WS-Policy によって解釈されるこれらの要素と属性を使用すれば、あらゆる複雑度のポリシー・アサーションを編成して組み合わせることができます。ポリシーの構成要素となる実際のアサーションを定義

するには、WS-Policy そのものではなく、ドメイン固有の拡張技術 (WS-SecurityPolicy など) を使用します。

WS-Policy のバージョン

この記事で説明する WS-Policy の詳細は、W3C が公開している「正式」な WS-Policy 1.5 に固有のものですが、その一般規則は、それよりも前に公開されて今でも広く使用されている「サブミッション」バージョンにも当てはまります。ある特定の文書がどのバージョンを使用しているかは、他の WS-* 技術と同じく、XML 名前空間を調べればわかります。WS-Policy 1.5 の名前空間は `http://www.w3.org/ns/ws-policy`、サブミッション・バージョンの名前空間は `http://schemas.xmlsoap.org/ws/2004/09/policy` です。この記事では一貫して `http://www.w3.org/ns/ws-policy` 名前空間を使用し、これを接頭辞 `wsp` で表します。

便宜上、WS-Policy は標準形式でのポリシー表現だけでなく、より簡潔なポリシー表現を作成するために使用できる一連のルールを定義しています。標準形式は冗長になりがちなので、(WS-Policy 勧告では「差し支えない限り、標準形式でのポリシー表現を使用すべきである」と規定してはいますが) ポリシー文書のほとんどの作成者は文書を扱いやすくするために、少なくとも部分的には簡略表現ルールを使用する傾向があります。ただし、ポリシー文書の解釈は標準形式を基本とするため、まずはこの標準形式について説明します。

標準形式での表現

標準形式でのポリシー表現では、最大 3 つの要素を使用します。これらの要素は必ず、特定の順序でネストされていなければなりません。最も外側にある要素は常に <wsp:Policy> で、この要素には `<wsp:ExactlyOne>` 子要素が 1 つだけ含まれている必要があります。`<wsp:Policy>` 内にネストされた `<wsp:ExactlyOne>` には `<wsp:All>` 子要素が含まれますが、その数には制限がありません (まったく含まれないこともあります)。したがって、最も単純な標準形式でのポリシー表現は `<wsp:Policy><wsp:ExactlyOne/></wsp:Policy>` となります。

標準形式でのポリシー・アサーションは、`<wsp:All>` 要素の中にネストしなければなりません。ネストされたポリシー・アサーションの中には、ポリシー表現を含めることができます。連載の過去の記事に記載した WS-SecurityPolicy アサーションでも、その多くにはポリシー表現がネストされています。標準形式でのポリシー表現では、ネストされたポリシー・アサーションも同じく標準形式でなければなりません (実際、ポリシー・アサーションはさらに厳格な標準形式のサブセットであり、最上位以外のすべての `<wsp:ExactlyOne>` 要素には、それぞれ 1 つの `<wsp:All>` 子要素しか含めることができません)。リスト 1 に抜粋したポリシーには、標準形式でのポリシー表現がネストされた例が示されています。

リスト 1. ポリシー表現がネストされた WS-SecurityPolicy の抜粋

```
<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>
            ...
```

何層にも要素がネストされているにも関わらず、この構造の意味は単純明快です。標準形式でのポリシー表現では、<wsp:Policy> はポリシー表現の単なるラッパーに過ぎません。これが最上位のポリシー表現だとすると、ここでポリシーに名前または ID が関連付けられます。<wsp:Policy> 内でネストされた <wsp:ExactlyOne> 要素は、この要素内でネストされた <wsp:All> 要素によって表現されるオルタナティブ (alternative) の OR 結合を表します (つまり、オルタナティブのいずれか 1 つが満たされると、ポリシー表現全体が満たされます)。一方、各 <wsp:All> 要素は、この要素内でネストされたポリシー・アサーションの AND 結合を表します (つまり、すべてのポリシー・アサーションが満たされる場合にのみ、オルタナティブが満たされます)。

標準形式でないポリシー

ポリシー表現は標準形式にしなければならないわけではありません。標準形式でないポリシー表現では、ネストされたオルタナティブを組み込むことも (最上位レベルの下の <wsp:ExactlyOne> 要素に複数の <wsp:All> 子要素を含めること)、次のセクションで説明する簡潔なポリシー表現オプションを利用することもできます。

論理学の理論を学んだことがある方は、標準形式での表現 (ネストされたオルタナティブがない表現) は論理表現の論理和標準形と同じようなものだという認識を持つことでしょう。ポリシー表現は実際のところ、不等号括弧の形式を使用し、アサーションを節とする論理式に過ぎません。論理学の理論では、あらゆる論理式を論理和標準形に変換できることが示されていますが、これと同じ原則が、オルタナティブをネストして作成されるポリシー表現にも当てはまります。つまり、ポリシー表現は、最上位にのみオルタナティブがある標準形式での表現の形にすることができます。標準形式でのポリシー表現が持つ主なメリットは、2 つのポリシーが共存可能かどうかをプログラムによって簡単にチェックできることです。2 つの標準形式ポリシーが共存可能な場合、これらのポリシーは、同じアサーションのセットが含まれる最上位レベルの <wsp:All> 要素を 1 つ以上共有することができます。

簡潔なポリシー表現

特にネストされたオルタナティブが関わってくる場合には尚更のことですが、標準形式でのポリシー表現は冗長になる傾向があります。そこで WS-Policy では、人間が理解しやすいように、標準形式よりも簡潔なポリシー表現を作成するために使用できるオプションを定義しています。WS-Policy の資料では、これらのオプションを使用するポリシーを「簡略形式 (compact form)」と呼んでいることから混乱を招きがちですが、実際には、複数の簡略表現のポリシーが 1 つの標準形式での表現に相当する場合もあります。この記事では単に、表現を簡潔にするための

オプションを 1 つでも使用しているポリシーに対して、「簡略表現 (compact expression)」という言葉を使用します。

簡略表現オプションの 1 つの機能は、基本ポリシー要素 (ポリシー用語ではオペレーター (operator) と呼ばれます。その理由は、基本ポリシー要素のそれぞれが、ネストされたアサーションの特定の解釈を暗に示すからです) を任意の順序でネストしてポリシーを表現できることです。ネストのルールでは、`<wsp:All>` に相当するものとして (標準形式で必要となる 1 つの `<wsp:ExactlyOne>` 子要素を使わずに) 直接使用される `<wsp:Policy>` 要素 (おそらく最も広く使用されている簡略表現の機能) の解釈も定義します。

リスト 1 のポリシーをこの `<wsp:Policy>` を使って表現した簡略表現をリスト 2 に記載します。この表現では長さが標準形式の半分以下になっています。ほとんどの人にとっては、簡略表現のほうが遥かに理解しやすいはずです。

リスト 2. 簡略形式の単純なポリシー

```
<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 では、簡略表現オプションを使用したポリシー表現を標準形式に変換するために適用できる一連の変換を定義しているので、標準形式を直接使用しなければならない理由はほとんどありません。人間が標準形式を解釈するよりも、コンピューターが簡略表現を標準形式に変換するほうが遥かに簡単です。

ポリシーの包含

簡略表現は要素のネストを単純化するだけでなく、ポリシー表現を参照して再利用する手段も提供します。その場合に使用するのが、4 つ目の WS-Policy 要素である `<wsp:PolicyReference>` です。`<wsp:PolicyReference>` 要素は、ポリシー・アサーションを使用できる箇所であれば、どこででも使用することができます。参照されるポリシー表現は効果的にポリシー参照に置き換えられます (厳密には、参照される `<wsp:Policy>` 要素を置き換える `<wsp:All>` 要素が使用されます)。この置き換えは、「ポリシーの包含 (policy inclusion)」と呼ばれます。

リスト 3 は、ポリシーの包含を使用した例です。ここではリスト 2 のポリシー表現を改良して、単独のポリシー表現と参照を使用した形にしています。

リスト 3. ポリシーの参照

```
<!-- 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"/>
      ...
```

ポリシーの参照は、リスト 3 のようにローカル・ポリシー表現に使用することも、外部ポリシー表現に使用することもできます。外部ポリシー表現では、通常は参照の `URI` 属性が外部ポリシーの実際の URL を指定します。

この後に記載するポリシーのテストの例で明らかになりますが、現在、ポリシーの包含は例外なくサポートされているわけではありません。そのため、この素晴らしい機能の有用性は限定されます。

ポリシーのオルタナティブ

前に説明したように、WS-Policy の構造はポリシーの一部としてのオルタナティブの選択をサポートするために、`<wsp:ExactlyOne>` 要素を使用します。簡略表現では、(少なくとも理論上は) 特殊な属性を使用して選択肢を作成することもできます。WS-Policy 勧告によると、ポリシー・アサーションが `<wsp:All>` または `<wsp:Policy>` 要素の子であるとしても、ポリシー・アサーションに `wsp:Optional="true"` 属性を追加することによって、そのアサーションを必須ではなくオプションにすることができます。

ポリシーのオルタナティブは重宝しそうな機能に思えますが、重要な実用例はなかなか思いつきません。この機能が実際に使用されるのは、通常、セキュリティー処理の特定のコンポーネント (`UsernameToken` など) をオプションにする場合です。例えば `UsernameToken` または X.509 証明書という形でクライアントが ID を提供できるようにするなどといった複雑なオルタナティブには、現行の WS-SecurityPolicy 実装では対処しきれないと思います。

ポリシーの関連付け

WSDL 1.1 (多少古くなっていますが、今でも最も広く使われているサービス定義の形) でのサービス定義には、階層構造が使用されます。最初 (最下位) の層は、サービスとの間でやり取りされるメッセージの XML 構造を定義する `<wsdl:message>` 要素からなります。2 番目の層にあるのは、一連の操作を定義する `<wsdl:portType>` 要素で、それぞれの操作は入力、出力、またはエラー・メッセージによって指定されます。3 番目の層には、特定のメッセージ・プロトコル (SOAP など) とアクセス・メソッドを `<wsdl:portType>` に関連付ける `<wsdl:binding>` 要素があります。そして 4 番目の層では `<wsdl:port>` 要素という形のサービス・エンドポイント定義が、`<wsdl:binding>` にアクセスできるアドレスを指定します。

WSDL スキーマ

WSDL 1.1 には XML スキーマ定義に関する長い変化に富んだ歴史があります。元の WSDL 1.1 サブミッションには、WSDL XML 文書を組み立てる方法に関するテキスト記述と、XML スキーマ定義の両方が含まれていましたが、残念ながら、その当時指定されていた XML スキーマはテキストとは一致していませんでした。この不一致はその後のスキーマの変更で修正されたものの、WSDL 1.1 文書はこの変更を反映するように更新されませんでした。WS-I Basic Profile グループは WSDL スキーマにさらに変更を加えることを決定し、この当てにならないスキーマのベスト・プラクティスとも言えるバージョンを作成しました。あるバージョンのスキーマに対して作成された文書は、一般に、他のバージョンには対応しません (まったく同じ名前空間を使っていても対応しません)。しかし幸いなことに、ほとんどの Web サービス・ツールは基本的にスキーマを無視し、妥当だと思われるものであれば何でも受け入れます。「参考文献」に、WSDL の多数のスキーマへのリンクが記載されています。

WS-Policy でポリシーを WSDL サービス定義に関連付けることのできるポイントはいくつかありますが、上記で説明したサービス定義の層と完全には一致していません。これらの層はサービス定義の論理構造を表す一方、WS-Policy はメッセージとメッセージのグループ化に重点を置くからです。WS-Policy は以下の 4 つのレベルでメッセージをグループ化します。

- **メッセージ**: ポリシーは特定のメッセージに適用されます (ポリシーが `<wsdl:message>` 要素によって関連付けられている場合、該当するメッセージが使用されるときには常にポリシーが適用されます。あるいは、`<wsdl:portType>` または `<wsdl:binding>` 要素の中で、操作の入力/出力/エラーの定義を通じてポリシーが関連付けられている場合は、特定の操作がそのメッセージを使用するときに、ポリシーが適用されます)。
- **操作**: ポリシーは、特定の操作でのすべてのメッセージ交換に適用されます (ポリシーは `<wsdl:binding>` または `<wsdl:portType>` に含まれる `<wsdl:operation>` 要素によって関連付けられます)。
- **エンドポイント**: ポリシーは、特定のサービス・バインディングに関するすべてのメッセージ交換 (ポリシーは `<wsdl:port>` または `<wsdl:binding>` によって関連付けられます)、または、ポリシーが `<wsdl:portType>` に関連付けられている場合は、その特定のポート・タイプに基づくすべてのサービス・バインディングに関するすべてのメッセージ交換に適用されます。
- **サービス**: ポリシーは、サービスに関連するエンドポイントおよび操作のすべてに適用されます (ポリシーは `<wsdl:service>` 要素で関連付けられます)。

WSDL に使用される主なポリシー関連付けメカニズムとなるのは、あるポリシーを別のポリシー内で参照するためのメカニズムと同じく、「ポリシーの包含」セクションで説明した `<wsp:PolicyReference>` 要素です。この WS-Policy 要素を前に記載した WSDL 要素の子として追加することで、該当するメッセージ・グループ化レベルで適用するポリシーを指定することができます。ポリシーを適切な内容が含まれた `<wsp:Policy>` 要素として直接埋め込むことも可能ですが、参照を使用すると WSDL 構造を簡潔にしておけるため、通常は参照を使用するほうが賢明です。

あるメッセージ・グループ化レベルで適用したポリシーは、その下の層では `<wsp:All>` 要素内に結合されて継承されます。そのため、各メッセージに適用される実際のポリシー (WS-Policy の用語で言うと、実効 (effective) ポリシー) は、メッセージ、操作、エンドポイント、そしてサービスの各層で適用されるすべてのポリシーの結合になります。したがって、ポリシーを決定するのはメッセージそのものだけでなく、そのメッセージが使用されるコンテキストによっても決定されます。

このメカニズムを説明するために、リスト 4 にポリシー参照を使用したスケルトン WSDL 定義を記載します。

リスト 4. ポリシー関連付けの例

```
<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>
```

上記の例では、`UsernameToken` を必要とするポリシーが `<wsdl:binding>` レベルで関連付けられています。さらに、このバインディングの一部として定義されている `addBook` 操作に、非対称メッセージ暗号化を要件とする追加のポリシーが関連付けられています。この例が WSDL のポリシー参照をすべて示しているとすると、`UsernameToken` は常に必要ですが、メッセージ署名は `addBook` 操作にだけ使用されることになります。

`<wsp:PolicyReference>` を使用して WSDL 要素のポリシーを直接参照する代わりに、`wsp:PolicyURIs` 属性を使用するという方法もあります。ポリシーが関連付けられる任意の WSDL 要素には、この属性を追加することができます。この属性は基本的に、`<wsp:PolicyReference>` 子要素を使用する場合と同じように機能します。

WS-SecurityPolicy の関連付け

WS-SecurityPolicy が指定するメッセージ・グループ化レベルでは、サービス記述に関連付けることのできるポリシー・アサーションのタイプがそれぞれに異なります。例えば、トランスポートのセキュリティーを指定するための `<sp:TransportBinding>` アサーションは、エンドポイント・レベル以外で関連付けることはできません。一方、メッセージ暗号化または署名付与を指定するための `<sp:AsymmetricBinding>` および `<sp:SymmetricBinding>` アサーションは、エンドポイント・レベルまたは操作レベルでのみ使用することができます。

`<sp:AsymmetricBinding>` または `<sp:SymmetricBinding>` をメッセージ・レベルで指定することはできませんが、暗号化または署名付与の対象とするメッセージ・コンポーネントについては、メッセージ・レベルで指定することができます。つまり、少なくとも理論上は、暗号化または署名付与をメッセージごとに指定することができるということです。

ポリシーの例

WS-Policy の原則と、WS-Policy が WSDL とどのように連動するかを学んだところで、今度はこれらの原則を利用するポリシーの例を試してみます。これまでの記事と同じく、今回もオープンソースの Java Web サービス・スタックを代表する Axis2、Metro、CXF の 3 つすべてでコードをテストします。

この記事が最初に公開された時点では、Axis2 の例は正常に動作しませんでしたが、その後すぐに、別の形での構成を使用すると問題を解決できることがわかりました。この変更は、更新した記事の本文およびダウンロード・コードには反映されています。

リスト 5 に、WSDL サービス定義の中で 3 つのレベルのポリシー関連付けを使用した一例 (サンプル・コードのダウンロードに含まれる effective1.wsdl) を記載します。使用した 3 つのポリシーは以下のとおりです。

- `UsernameToken`: ハッシュ化パスワードを使用した UsernameToken を要件とします。
- `SymmEncr`: クライアントが生成する秘密鍵を使用した対称暗号化を要件とします。
- `EncrBody`: メッセージ本体の暗号化を要件とします。

リスト 5. 実効ポリシーの例

```
<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>
```

リスト 5 の WSDL 文書で使用しているポリシー参照 (太字で記載) は、`UsernameToken` ポリシーを `<wsdl:binding>` に関連付けています。したがって、`UsernameToken` はすべての操作に必要ということになります。本の情報を交換するすべての操作について、`SymmEncr` ポリシーが個々の `<wsdl:operation>` に関連付けられていて、`EncrBody` ポリシーが本の情報を含めるメッセージに関連付けられています。つまり、本の情報は常に暗号化された形で送信されるということです。

これが厄介なポリシーとなっているのは、秘密鍵はレスポンスを暗号化するためにしか使用されないにも関わらず、クライアントが秘密鍵を生成して、その鍵をリクエスト・メッセージでサーバーに送信しなければならないという点です。Axis2 1.5.1 (1.5.2 のディストリビューションで試してみましたが、このディストリビューションにはセキュリティー処理に必要なファイルがありません) は、生成されたコードにおいても、サーバー操作においても、完全にポリシーを無視し、セキュリティーをまったく適用せずに実行されます。古いサブミッションの名前空間を使うよう

にポリシーを変更すると、Axis2 はポリシーを認識して正しく実行されましたが、`addBook` 操作のエラー・レスポンスの処理についてはその限りではありません。ポリシーでは、アプリケーション・エラーのレスポンスが返される際には暗号化を使用することになっていますが、Axis2 は暗号化されていないレスポンスを送信しました。

Metro は `UsernameToken` を使用したリクエスト・メッセージを生成しましたが、秘密鍵の情報はメッセージに含まれていないため、最初のメッセージ交換で完全に失敗しました。CXF 2.3.0 は Metro よりも遥かに優秀で、2 つのケースを除けば、ポリシーを正しく処理しました。そのケースとはまず、`addBook` 操作が成功したときに、レスポンス・メッセージが暗号化を使用しないことです。CXF サーバーは `addBook` 操作を正しく処理したものの、レスポンスの処理中にクライアントが例外をスローしました。もう 1 つの CXF のエラーは Axis2 と同じく、アプリケーション・エラーのレスポンスに暗号化を使用しないことです。

リスト 5 のポリシーはメッセージ暗号化を選択的に使用する例で、このポリシーは本の情報が含まれるメッセージだけを暗号化します。ただし、このポリシーが使用するのは対称暗号化です。これと同じことを、クライアントが独自の証明書を持つ非対称暗号化を使用して実現できたとしたら言うことありません (特に、送信側を検証するためにメッセージに署名を付与したい場合には尚更のことです)。リスト 6 に、非対称暗号化を使用するように意図された例 (ダウンロードに含まれる effective2.wsdl) を記載します。この例で使用する WSDL のポリシーは、以下の 2 つだけです。

- `AsymmBinding`: 二重の証明書を使用した非対称暗号化を要件とします。
- `SignBody`: メッセージ本体の署名付与を要件とします。

リスト 6. 非対称暗号化および署名付与の例

```
<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>
```

リスト 6 の例では非対称暗号化を使用して、本の情報を提供するすべてのメッセージに署名を付与します。これは、`getBook` および `getBooksByType` レスポンス・メッセージにはサーバーが署名を付与する一方、`addBook` リクエスト・メッセージにはクライアントが署名を付与することを意味します。ここでは非対称暗号化を使用し、サーバーとクライアントのそれぞれが独自の証明書と秘密鍵を持つことから、リスト 5 の対称暗号化の例よりもいくらか扱いやすいはずです。

外部ポリシー参照

外部ポリシー参照は、この記事に記載した例の構造に使うのに十分適しているとは言えませんが、外部ポリシー参照についてもテストしました。テストで使用したのは、相対参

照 (<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy" URI="./asymm-binding-policy.xml"/> の形) と Web サーバー上でホストされている絶対 URL の両方です。Axis2 と Metro はいずれもこのタイプのポリシー参照を解決できませでしたが、CXF では外部ポリシー参照が正しく機能しました。

Axis2 は前の例と同じように失敗しました。つまり、WS-Policy 1.5 のポリシー名前空間を使用したポリシー・コンポーネントを完全に無視したということです。けれどもサブミッション・バージョンの名前空間に変更したところ、Axis2 はこの例をエラーなしで処理することができました。Axis2 で WS-Policy 1.5 名前空間を使用するとこのような問題があるため、ダウンロード・コードの Axis2 の例ではすべて、サブミッション・バージョンの名前空間を使用しています。

Metro は、このように指定されたポリシー構成を処理することができず、クライアントでは最初のリクエストから早速 `NullPointerException` をスローします。調査してわかったことは、ポリシーを操作レベルとメッセージ・レベルのみで関連付けるのではなく、エンドポイント・レベルで関連付けると問題が解消されることです。effective3.wsdl の Metro の例には `<wsdl:binding>` からだけ参照される `#AsymmBinding` ポリシーを組み込んであります。この例は問題なく実行されます (ただし、対称暗号化の例での CXF のように、Metro はアプリケーション・エラーのレスポンスにはセキュリティーを適用しません)。

CXF もリスト 6 のポリシー構成では失敗しますが、現在のコードに簡単な回避策はありません。CXF のクライアント・コードは `getBook` リクエスト・メッセージを送信する際に誤った署名を生成し、続いてサーバー・コードがリクエスト・メッセージの処理に失敗します。したがって現在の CXF コードでは、署名を付与する対象が何もないとしても、`AsymmetricBinding` がスコープ内にある場合には署名を生成せざるを得ないようです。

ポリシーのまとめ

WS-Policy は、あらゆる形で制約を表現するための柔軟で強力な構造を定義しています。けれども残念ながら、Web サービス・スタックで使用する WS-Policy および WS-SecurityPolicy 処理の実装では、その柔軟性を十分には実装しません。この実装サポートの欠如から、あらゆる種類の Web サービス・スタックと相互作用することを目的とした Web サービスには、有用であるはずの多くの WS-Policy の機能が使用できない状態となっています。

WSDL サービス定義のさまざまなポイントでポリシーを関連付ける基本的な実効ポリシー処理は、ポリシーと WSDL 設計のコア機能であり、サービスに適していれば使用したほうがよいでしょう。けれどもこのタイプの構成に考えられる有用な機能の 1 つ、つまり選択的に個別のメッセージに署名付与と暗号化を適用する機能は、確実には機能しません (テスト・ケースを正しく処理したのは Apache Axis2 だけで、それもポリシー名前空間を変更した上でのことです)。この制約を考えると、最大限の相互運用性を必要とする場合には、とりあえずポリシーの関連付けは `<wsdl:binding>` および `<wsdl:operation>` レベルだけにとどめておくのが最善の策だと思います。

外部ポリシーは、特に SOA 環境に役立つはずです。SOA 環境では、組織全体で使用する一連の共通ポリシーをセットアップし、各サービスが必要に応じたポリシーを参照することができます。現在、外部ポリシーはすべてのオープンソースの Java Web サービス・スタックでサポートされているわけではありませんが (これを適切に処理するのは Apache CXF のみです)、規模の大きな組織では外部ポリシーを利用して、この機能をサポートするサービス・スタックのいずれか (オープン

ソースであるか、商用であるかに関わらず) に Web サービス実装を制限するという方法があります。また、例えば WSDL の include 要素を使用するなどの他の手段で同じ効果をもたらすことができるかもしれません。

この連載の次回の記事では、3 つの主要な Java Web サービス・スタックのパフォーマンスおよび相互運用性に関する問題を要約し、セキュアな Web サービスを実装するという観点でこれらのスタックを比較します。組織の中でセキュアな Web サービスを使用しているとしたら、次回の記事も見逃せません。

ダウンロード

内容	ファイル名	サイズ
Sample code for this article	j-jws18.zip	94KB

著者について

Dennis Sosnoski

Dennis Sosnoski は Java ベースの XML および Web サービスを専門とするコンサルタント兼トレーナーです。専門家としてのソフトウェア開発経験は 30 年以上に渡り、この 10 年間はサーバー・サイドの XML 技術や Java 技術に注力しています。オープンソースのJiBX XML Data Binding フレームワークや、それに関連した JiBX/WS Web サービス・フレームワークの開発リーダーを務め、さらに Apache Axis2 Web サービス・フレームワークのコミッターでもあります。彼は JAX-WS 2.0 および JAXB 2.0 仕様のエキスパート・グループの一員でもありました。連載「Java Web サービス」の内容は Dennis の SOA and web services のトレーニング・コースを基にしています。