WSDL Binding Styles

October 25, 2009 by jdslo

Contents

Introduction

A WSDL document is a Web service interface. A WSDL binding describes how the service is bound to a messaging protocol, generally the SOAP messaging protocol. A WSDL SOAP binding can be either Remote Procedure (RPC) style binding or a document style binding. A SOAP binding can also have an encoded or literal use.

The RPC style specifies that the <soap:body> contains an element with the name of the Web method being invoked. This element in turn contains an entry for each parameter and the return value of this method. RPC style more closely reflects the style used when defining a Java interface, which is generally a fine-grained operation.

With Document Style, the message parts appear directly under the <soap:body> element. There are no SOAP formatting rules for what the <soap:body> contains. The server application is responsible for mapping the server objects (parameters, method calls, and so forth) and the values of the XML documents.

Encoding use, each message part references an abstract type using the type attribute instead of a schema. Applications using SOAP encoding are focused on remote procedure calls and will likely use the RPC message style.

Literal use, each part references a concrete schema definition using either the element or type attribute; in other words, data is serialized according to a given schema.

Table Summary

WSDL Style	Advantages	Disadvantages
RPC/encoded	○ Simple structure ○ Operation name appears in the message	○ Type encoding degrades throughput performance ○ Not WS-I compliant ○ Difficult to validate ○ Not WS-I compliant
RPC/literal	○ Simple structure ○ Operation name appears in the message ○ WS-I compliant	○ Difficult to validate
Document/literal	○ Can be validated	○ The operation name in the SOAP message is lost. Without the name, dispatching can be difficult, and sometimes impossible. ○ WS-I only allows one child of the soap:body in a SOAP message, which restricts message content.

Document/literal wrapped	○ Easiest binding to Java since the parameters are enclosed within a wrapper object ○ Easy to validate ○ Operation name in the SOAP message, easy to dispatch ○ WS-I compliant, meets the WS-I restriction that the SOAP message's soap:body has only one child ○ De-facto standard for defining interoperable Web services ○ Encourages course-grain operations, because it passes XML documents, potentially reducing the number of service calls	○ WSDL slightly more complex ○ Messages somewhat more verbose

Example Java method for use in our comparison.

```
public void myMethod(int x, float y);
```

RPC/encoded WSDL

The complete method is specified in the WSDL file and in the SOAP body. This includes the sent parameters and the return values. The Encoded- USE indicates that the SOAP message will contain the encoded type information such as `xsi:type="xsd:int", xsi:type="xsd:string", xsi:type="xsd:double"`.

```
<message name="myMethodRequest">
 <part name="x" type="xsd:int"/>
 <part name="y" type="xsd:float"/>
</message>

<message name="empty"/>

<portType name="PT">
 <operation name="myMethod">
  <input message="myMethodRequest"/>
  <output message="empty"/>
 </operation>
</portType>
```

```
<binding name="MyBinding" type="tns:PT">
 <soap:binding transport="http://schemas.xmlsoap.org/soap/http&quot
(http://schemas.xmlsoap.org/soap/http&quot); style="rpc"/>
  <operation name="myMethod">
 <soap:operation soapAction=""/>
 <input>

  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/&quot
(http://schemas.xmlsoap.org/soap/encoding/&quot); use="encoded" namespace="urn:Foo"/>
   ...
```

RPC/encoded SOAP message

```
<soap:envelope>
 <soap:body>
  <myMethod>
   <x xsi:type="xsd:int">5</x>
   <y xsi:type="xsd:float">5.0</y>
  </myMethod>
 </soap:body>
</soap:envelope>
```

Strengths

Simple structure.
The operation name appears in the message, so the receiver has an easy time dispatching this message to the implementation of the operation.

Weaknesses

The type encoding info (such as xsi:type="xsd:int") is usually just overhead which degrades throughput performance.
You cannot easily validate this message since only the <x ...>5</x> and <y ...>5.0</y> lines contain values defined in a schema; the rest of the soap:body contents comes from WSDL definitions.
Any changes to the interface would break the contract between the service and the client since there is a tight coupling between service provider and client.
Although it is legal WSDL, RPC/encoded is not WS-I compliant.

RPC/literal WSDL

The style will remain RPC but the WSDL will now have the ‘use’ setting changed from ‘encoded’ to ‘literal’ in the soap:body. Which means the data is serialized according to the given schema.

```
<message name="myMethodRequest">
 <part name="x" type="xsd:int"/>
 <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>


<portType name="PT">
 <operation name="myMethod">
  <input message="myMethodRequest"/>
  <output message="empty"/>
 </operation>
</portType>


<binding name="MyBinding" type="tns:PT">
 <soap:binding transport="http://schemas.xmlsoap.org/soap/http&quot
(http://schemas.xmlsoap.org/soap/http&quot); style="rpc"/>
 <operation name="myMethod">
  <soap:operation soapAction=""/>
  <input>
   <soap:body use="literal" namespace="urn:Foo"/>
    ...
```

RPC/literal SOAP message

```
<soap:envelope>
 <soap:body>
  <myMethod>
   <x>5</x>
   <y>5.0</y>
  </myMethod>
 </soap:body>
</soap:envelope>
```

Strengths

The WSDL is still simple and straightforward.
The operation name still appears in the message.
The type encoding info is eliminated.
RPC/literal is WS-I compliant.

Weaknesses

You still cannot easily validate this message since only the 5 and 5.0 lines contain values defined in a schema; the rest of the soap:body contents comes from WSDL definitions.
Document/literal WSDL
Similar to RPC/literal, but types must now reference elements. Document literal encourages the use of courser-grained operations by packaging SOAP payloads within XML documents. Course-grained operations can reduce the number of messages being passed between provider and consumer.

Document/literal WSDL

```
<types>
 <schema>
  <element name="xElement" type="xsd:int"/>
  <element name="yElement" type="xsd:float"/>
 </schema>
</types>

<message name="myMethodRequest">
 <part name="x" element="xElement"/>
 <part name="y" element="yElement"/>
</message>
<message name="empty"/>

<portType name="PT">
 <operation name="myMethod">
  <input message="myMethodRequest"/>
  <output message="empty"/>
 </operation>
</portType>

<binding name="MyBinding" type="tns:PT">
 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/&gt
(http://schemas.xmlsoap.org/soap/http"/&gt);
  <operation name="myMethod">
   <input>
    <soap:body use="literal"/>
     ...
```

Document/literal SOAP message

```
<soap:envelope>
 <soap:body>
  <xElement>5</xElement>
  <yElement>5.0</yElement>
 </soap:body>
</soap:envelope>
```

Strengths

There is no type encoding info.
You can finally validate this message with any XML validator. Everything within the soap:body is defined in a schema.
Document/literal is WS-I compliant, but with restrictions

Weaknesses

The WSDL is getting a bit more complicated.
The operation name in the SOAP message is lost. Without the name, dispatching can be difficult, and sometimes impossible.
WS-I only allows one child of the soap:body in a SOAP message.
Document/literal wrapped WSDL
Most SOAP implementations now support the "wrapped" convention. .NET uses it by default (and it doesn't support rpc/literal). All JAX-RPC compliant implementations must also support "wrapped". The "wrapped" document-literal has become the de-facto standard for defining interoperable Web services.

Document/literal wrapped WSDL

```
<types>
 <schema>
  <element name="myMethod">
   <complexType>
    <sequence>
     <element name="x" type="xsd:int"/>
     <element name="y" type="xsd:float"/>
    </sequence>
   </complexType>
  </element>
  <element name="myMethodResponse">
   <complexType/>
  </element>
 </schema>
</types>

<message name="myMethodRequest">
 <part name="parameters" element="myMethod"/>
</message>
<message name="empty">
 <part name="parameters" element="myMethodResponse"/>
</message>

<portType name="PT">
 <operation name="myMethod">
  <input message="myMethodRequest"/>
 <output message="empty"/>
 </operation>
</portType>
```

```
<binding name="MyBinding" type="tns:PT">
 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/&gt
(http://schemas.xmlsoap.org/soap/http"/&gt);
  <operation name="myMethod">
   <input>
    <soap:body use="literal"/>
     ...
```

Document/literal wrapped SOAP message

```
<soap:envelope>
 <soap:body>
  <myMethod>
   <x>5</x>
   <y>5.0</y>
  </myMethod>
 </soap:body>
</soap:envelope>
```

These are the basic characteristics of the document/literal wrapped pattern:

- The input message has a single part.
- The part is an element.
- The element has the same name as the operation.
- The element's complex type has no attributes.

Strengths

There is no type encoding info.
Everything that appears in the soap:body is defined by the schema, so you can easily validate this message.
Once again, you have the method name in the SOAP message.
Document/literal is WS-I compliant, and the wrapped pattern meets the WS-I restriction that the SOAP message's soap:body has only one child.

Weaknesses

The WSDL is a bit more complicated.

Rules for implementing Document/Literal Wrapped

"Wrapped" is a form of document/literal, therefore it must follow all the rules defined for document/literal. When defining a document/literal service, there can be at most one body part in your input message and at most one body part in your output message. You do *not* define each method parameter as a separate part in the message definition. (The parameters are defined in the types section, instead.)
Each part definition must reference an element (not a type) defined, imported, or included in the types section of the WSDL document. These element definitions are "wrapper" elements (hence the name of this convention). You define your input and output parameters as element structures within these wrapper elements.
A wrapper element must be defined as a complex type that is a sequence of elements. Each child element in that sequence will be generated as a parameter in the service interface.

The name of the input wrapper element must be the same as the operation name.
The name of the output wrapper element should be (but doesn't have to be) the operation name appended with "Response" (e.g., if the operation name is "add", the output wrapper element should be called "addResponse").
In the binding definition, the soap:binding should specify style="document" (although this is the default value, so the attribute may be omitted), and the soap:body definitions must specify use="literal" and nothing else. You must not specify the namespace or encodingStyle attributes in the soap:body definition.

Further Information / Article source

http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/ (http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/)

http://labs.icodeon.com/projects/spd/html/index.html (http://labs.icodeon.com/projects/spd/html/index.html)

Posted in SOA | 2 Comments

2 Responses

Kenneth *on May 9, 2010 at 3:25 am | Reply*
Thanks for sharing I will try it to my site.

Kenneth

Biju *on April 19, 2011 at 10:20 pm | Reply*
Thank you. Very well explained.. after so many days of searching, at last I found the exact answers I was looking for.

Comments RSS