SOAP协议规范

by wxyxl

Fri, 04 May 2001 13:58:32 GMT

from: CSDN技术中心 http://dev.csdn.net/article/6/6763.shtm

整理: 范晨鹏 p_168@163.com

1. 简介

SOAP以XML形式提供了一个简单、轻量的用于在分散或分布环境中交换结构化和类型信息的机制。SOAP本身并没有定义任何应用程序语义，如编程模型或特定语义的实现；实际上它通过提供一个有标准组件的包模型和在模块中编码数据的机制，定义了一个简单的表示应用程序语义的机制。这使SOAP能够被用于从消息传递到RPC的各种系统。

SOAP包括三个部分

- SOAP封装（见第4节）结构定义了一个整体框架用来表示消息中包含什么内容，谁来处理这些内容以及这些内容是可选的或是必需的。
- SOAP编码规则（见第5节）定义了用以交换应用程序定义的数据类型的实例的一系列机制。
- SOAP RPC表示（见第7节）定义了一个用来表示远程过程调用和应答的协定。

虽然这三个部分都作为SOAP的一部分一起描述，但它们在功能上是相交的。特别的，封装和编码规则是在不同的名域中定义的，这种模块性的定义方法增加了简单性在SOAP封装，SOAP编码规则和SOAPRPC协定之外，这个规范还定义了两个协议的绑定，描述了在有或没有HTTP扩展框架[6]的情况下，SOAP消息如何包含在HTTP消息[5]中被传送。

1.1 设计目标

SOAP的主要设计目标是简单性和可扩展性，这意味着传统的消息系统和分布对象系统的某些性质不是SOAP规范的一部分。这些性质包括：

- 分布式碎片收集
- 成批传送消息
- 对象引用（要求分布式碎片收集）
- 激活机制（要求对象引用）

1.2 符号约定

这篇文章中的关键字 "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT","SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", 和"OPTIONAL"的解释在RFC-2119 [2]中。 这篇文章中用到的名域前缀 "SOAP-ENV" 和"SOAP-ENC"分别与"http://schemas.xmlsoap.org/soap/envelope/"

和"http://schemas.xmlsoap.org/soap/encoding/"关联。整篇文档中，名域前缀“xsi”被假定为与URI"http://www.w3.org/1999/XMLSchema-instance“（在XMLSchema规范[11]定义）相连。类似的，名域前缀“xsd”被假定为与URI"http://www.w3.org/1999/XMLSchema"（在[10]中定义）相连。名域前缀“tns”用来表示任意名域。所有其它的名域前缀都只是例子。
名域URI的基本形式“some－URI”表示某些依赖于应用程序或上下文的URI[4]。这个规范用扩展BNF（在RFC－2616[5] 描述）描述某些结构。

1.3 SOAP消息举例

在这个例子中，GetLastTradePrice SOAP 请求被发往StockQuote服务。这个请求携带一个字符串参数和ticker符号，在SOAP应答中返回一个浮点数。XML名域用来区分SOAP标志符和应用程序特定的标志符。这个例子说明了在第6节中定义的HTTP绑定。如果SOAP中管理XML负载的规则完全独立于HTTP是没有意义的，因为事实上该负载是由HTTP携带的。在Appendix A中有更多的例子。

例1 在HTTP请求中嵌入SOAP消息

```
POST /StockQuote HTTP/1.1
Host:
www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction:
"Some-URI"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>DIS</symbol>
</m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

下面是一条应答消息，包括HTTP消息，SOAP消息是其具体内容 ：

例2 在HTTP应答中嵌入SOAP消息

```
HTTP/1.1 200 OK
Content-Type: text/xml;
```

```
charset="utf-8"
Content-Length:
nnnn
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
<SOAP-ENV:Body>
<m:GetLastTradePriceResponse xmlns:m="Some-URI">
<Price>34.5</Price>
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2. SOAP消息交换模型

SOAP消息从发送方到接收方是单向传送，但正如上面显示的，SOAP消息经常以请求/应答的方式实现。SOAP实现可以通过开发特定网络系统的特性来优化。例如，HTTP绑定（见第6节）使SOAP应答消息以HTTP应答的方式传输，并使用同一个连接返回请求。不管SOAP被绑定到哪个协议，SOAP消息采用所谓的“消息路径”发送，这使在终节点之外的中间节点可以处理消息。一个接收SOAP消息的SOAP应用程序必须按顺序执行以下的动作来处理消息：识别应用程序想要的SOAP消息的所有部分 （见4.2.2节）检验应用程序是否支持第一步中识别的消息中所有必需部分并处理它。如果不支持，则丢弃消息（见4.4节）。在不影响处理结果的情况下，处理器可能忽略第一步中识别出的可选部分。如果这个SOAP应用程序不是这个消息的最终目的地，则在转发消息之前删除第一步中识别出来的所有部分。为了正确处理一条消息或者消息的一部分，SOAP处理器需要理解：所用的交换方式（单向，请求/应答，多路发送等等），这种方式下接收者的任务，RPC机制（如果有的话）的使用（如第7节中所述），数据的表现方法或编码，还有其它必需的语义。尽管属性（比如SOAP encodingstyle，见4.1.1节）可以用于描述一个消息的某些方面，但这个规范并不 强制所有的接收方也必须有同样的属性并取同样的属性值。举个例子，某一特定的应用可能知道一个元素表示一条遵循第7节约定的RPC请求，但是另外一些应用可能认为指向该元素的所有消息都用单向传输，而不是类似第7节的请求应答模式。
（译者注：交互双方的SOAP消息并不一定要遵循同样的格式设定，而只需要以一种双方可理解的格式交换信息就可以了）

3. 与XML的关系

所有的SOAP消息都使用XML形式编码（更多有关XML的信息请见[7]）一个SOAP应用程序产生的消息中，所有由SOAP定义的元素和属性中必须包括正确的名域。SOAP应用程序必须能够处理它接收到的消息中的SOAP名域（见4.4节），并且它可以处理没有SOAP名域的SOAP消息，就象它们有正确的名域一样。SOAP定义了两个名域（更多有关XML名域的信息请见[8]）

- SOAP封装的名域标志符是"http://schemas.xmlsoap.org/soap/envelope/"
- SOAP的编码规则的名域标志符是"http://schemas.xmlsoap.org/soap/encoding/"

SOAP消息中不能包含文档类型声明，也不能包括消息处理指令。[7] SOAP使用"ID"类型"id"属性来指定一个元素的唯一的标志符，同时该属性是局部的和无需校验的。SOAP使用"uri-reference"类型的"href"属性指定对这个值的引用，同时该属性是局部的和无需校验的。这样就遵从了XML规范[7]，XMLSchema规范[11]和XML连接语言规范[9]的风格。除了SOAP mustUnderstand 属性(见4.2.3节）和SOAPactor属性(见4.2.2节）之外，一般允许属性和它们的值出现在XML文档实例或Schema中（两者效果相同）。也就是说，在DTD或Schema中声明一个缺省值或固定值和在XML文档实例中设置它的值在语义上相同。

4. SOAP封装

SOAP消息是一个XML文档，包括一个必需的SOAP封装，一个可选的SOAP头和一个必需的SOAP体。在这篇规范剩余部分中，提到SOAP消息时就是指这个XML文档。这一节中定义的元素和属性的名域标志符为：

"http://schemas.xmlsoap.org/soap/envelope/" 。一个SOAP消息包括以下部分：1.在表示这个消息的XML文档中，封装是顶层元素。2.应用SOAP交换信息的各方是分散的且没有预先协定，SOAP头提供了向SOAP消息中添加关于这条SOAP消息的某些要素(feature)的机制。SOAP定义了少量的属性用来表明这项要素（feature）是否可选以及由谁来处理。（见4.2节）3.SOAP体是包含消息的最终接收者想要的信息的容器（见4.3节）。SOAP为SOAP体定义了一个Fault元素用来报告错误信息。语法规则如下所示：

封装

1. 元素名是 "Envelope"

2. 在SOAP消息中必须出现。

3. 可以包含名域声明和附加属性。如果包含附加属性，这些属性必须限定名域。类似的，"Envelope"可以包含附加子元素，这些也必须限定名域且跟在SOAP体元素之后。

SOAP头 （见4.2节）

1. 元素名是"Header"

2. 在SOAP消息中可能出现。如果出现的话，必须是SOAP封装元素的第一个直接子元素。

3. SOAP头可以包含多个条目，每个都是SOAP头元素的直接子元素。所有SOAP头的直接子元素都必须限定名域。

SOAP体 （见4.3节）

1. 元素名是"Body"

2. 在SOAP消息中必须出现且必须是SOAP封装元素的直接子元素。它必须直接跟在SOAP头元素（如果有）之后。否则它必须是SOAP封装元素的第一个直接子元素。

3. SOAP体可以包括多个条目，每个条目必须是SOAP体元素的直接子元素。SOAP体元素的直接子元素可以限定名域。SOAP定义了SOAPFault元素来表示错误信息。

4.1.1 SOAP encodingStyle属性

EncodingStyle全局属性用来表示SOAP消息的序列化规则。这个属性可以在任何元素中出现，作用范围与名域声明的作用范围很相似，为这个元素的内容和它的所有没有重载此属性的子元素。SOAP消息没有定义缺省编

码。属性值是一个或多个URI的顺序列表，每个URI确定了一种或多种序列化规则，用来不同程度反序列化SOAP消息，举例如下：

"http://schemas.xmlsoap.org/soap/encoding/"
"http://my.host/encoding/restricted http://my.host/encoding/"
""

第5节中定义的序列化规则由URI"http://schemas.xmlsoap.org/soap/encoding/" 确定。使用这个特定序列化规则的消息应该用encodingStyle属性说明这一点。另外，所有以"http://schemas.xmlsoap.org/soap/encoding/"开头的URI中的序列化规则与第5节中定义的SOAP编码规则相一致。一个零长度的URI（""）明确显示所含元素没有任何编码形式。这可以用来取消上一级元素的所有编码声明。

4.1.2 封装版本模型

SOAP没有定义常规的基于主版本号和辅版本号的版本形式。SOAP消息必须有一个封装元素与名域"http://schemas.xmlsoap.org/soap/envelope/"关联。如果SOAP应用程序接收到的SOAP消息中的SOAP封装元素与其他的名域关联，则视为版本错误，应用程序必须丢弃这个消息。如果消息是通过HTTP之类的请求/应答协议收到的，应用程序必须回答一个SOAP VersionMismatch 错误信息（见4.4节）。

4.2 SOAP头

SOAP为相互通信的团体之间提供了一种很灵活的机制：在无须预先协定的情况下，以分散但标准的方式扩展消息。可以在SOAP头中添加条目实现这种扩展，典型的例子有认证，事务管理，支付等等。头元素编码为SOAP封装元素的第一个直接子元素。头元素的所有直接子元素称作条目。条目的编码规则如下：

一个条目有它的完整的元素名（包括名域URI和局部名）确定。SOAP头的直接子元素必须有名域限制。
SOAP encodingStyle属性可以用来指示条目所用的编码形式（见4.1.1节）
SOAP mustUnderstand属性（见4.2.3节）和SOAPactor属性（见4.2.2节）可以用来指示如何处理这个条目以及由谁来处理。（见4.2.1节）

4.2.1 使用头属性

这一节中定义的SOAP头属性确定了SOAP消息的接收者应该怎样按第2节中所述的方式处理消息。产生SOAP消息的SOAP应用程序，应该仅仅在SOAP头元素的直接子元素中使用这些SOAP头属性。SOAP消息的接收者必须忽略所有不在SOAP头元素的直接子元素中SOAP头属性。下面的例子是一个SOAP头，包括一个元素标志符"Transaction"，"mustUnderstand"取值为"1"和数值5。这应该以如下方式编码：

<SOAP-ENV:Header>
<t:Transaction
xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
5

```
    </t:Transaction>
</SOAP-ENV:Header>
```

4.2.2 SOAP actor属性

一个SOAP消息从始节点到终节点的过程中，可能沿着消息路径经过一系列SOAP中间节点。一个SOAP中间节点是一个可以接收转发SOAP消息的应用程序。中间节点和终节点由URI区分。可能SOAP消息的终节点并不需要所有部分，而在消息路径上的一个和几个中间节点可能需要这些内容。头元素的接收者扮演的角色类似于一个过滤器，防止这些只发给本接受者的消息部分扩散到其它节点。即一个头元素的接收者必须不转发这些头元素到SOAP消息路径上的下一个应用程序。同样的，接收者可能插入一个相似的头元素。SOAP actor全局属性可以用于指示头元素的接收者。SOAP actor属性的值是一个URI。

URI "http://schemas.xmlsoap.org/soap/actor/next"指出了第一个处理这个消息的SOAP应用程序需要这个头元素。这类似于HTTP头中用Connection域表示hop-by-hop范围模型。省略SOAP actor属性表示接收者是SOAP消息的终节点。如果这个属性要生效，它必须出现在SOAP消息实例中。（见第3节和4.2.1节）

4.2.3 SOAP mustUnderstand属性

SOAP mustUnderstand全局属性用来指示接受者在处理消息时这个条目是否必须处理。条目的接收者由SOAP actor属性定义（见4.2.2节）。MustUnderstand属性的值是"1" 或 "0"。缺少SOAP mustUnderstand属性在语义上等同于它的值为"0"。如果一个头元素的SOAP mustUnderstand属性的值是"1",那么条目的接受者必须或者遵守语义（如以元素的全名传送）并按照语义正确的处理，或者放弃处理消息（见4.4节）。SOAP mustUnderstand属性考虑了消息演变的准确性（robust evolution）。必须假定包含SOAP mustUnderstand属性且值为"1"的元素以某种方式修改了它们的父元素或同层元素的语义。以这种方式连接元素确保了语义上的变化不会被那些不能完全理解它的接收者忽略。如果这个属性要生效，它必须出现在SOAP消息实例中。（见第3节和4.2.1节）

4.3 SOAP体

SOAP体元素提供了一个简单的机制，使消息的最终接收者能交换必要的信息。使用体元素的典型情况包括配置RPC请求和错误报告。体元素编码为SOAP封装元素的直接子元素。如果已经有一个头元素，那么体元素必须紧跟在头元素之后，否则它必须是SOAP封装元素的第一个直接子元素。体元素的所有直接子元素称作体条目，每个体条目在SOAP体元素中编码为一个独立的元素。条目的编码规则如下：

- 一个条目由它的元素全名（包括名域URI和局部名）确定。SOAP体元素的直接子元素可能是名域限制的。
- SOAP encodingStyle属性可能用来指示条目（见4.1.1节）的编码方式。
- SOAP定义了一个Fault条目用来报告错误信息。（见4.4节）

4.3.1 SOAP头和体的关系

虽然头和体定义为独立的元素，它们实际上是有关系的。体条目和头条目的关系如下：体条目在语义上等同于actor属性为缺省值且mustUnderstand属性值为"1"的头条目。不使用actor属性则表示缺省的actor。（见4.2.2节）

4.4 SOAP错误

SOAP错误元素用于在SOAP消息中携带错误和（或）状态信息。如果有SOAP错误元素，它必须以以体条目的方式出现，并且在一个体元素中最多出现一次。SOAP错误元素定义了以下四个子元素：

- faultcode

faultcode元素给软件提供了一个识别此错误的算法机制。SOAP错误元素必须有faultcode子元素，并且它的值必须是一个合法的名（在[8]节定义）。SOAP定义一些SOAP faultcode描述基本的SOAP错误（见4.4.1节）。

- faultstring

faultstring元素提供了一个错误解释，而不是为了软件处理。faultstring元素类似于HTTP中定义(见[5]，第6.1节）的'Reason-Phrase'。SOAP错误元素必须有faultstring子元素，并且它应该提供一些错误本质的解释信息。

- faultactor

faultactor元素提供了在消息路径上是谁导致了错误发生的信息（见第2节）。它类似于SOAP actor属性（见4.2.2节），只是SOAP actor指的是头条目的目的地，faultactor指的是错误的来源。faultactor属性的值是用来区分错误来源的URI。不是SOAP消息的最终目的地的应用程序必须在SOAP Fault元素中包含faultactor元素。消息的最终目的地可以使用faultactor元素明确的指示是它产生了这个错误（参见下面的detail元素）

- detail

detail元素用来携带与Body元素有关的应用程序所要的错误信息。如果Body元素的内容不能被成功的处理，则必须包含detail子元素。它不能用来携带属于头条目的错误信息。头条目的详细出错信息必须由头条目携带。Fault元素中没有detail元素表示这个错误与Body元素的处理无关。在有错误的时候，这可以用来区分Body元素有没有被正确的处理。detail元素的所有直接子元素称作detail条目，并且每个detail条目在detail元素中编码为独立的元素。detail条目的编码规则如下（参见例10）： 一个detail条目由它的元素全名（包括名域URI和局部名）确定。SOAP体元素的直接子元素可能是名域限制的。SOAP encodingStyle属性可能用来指示detail条目（见4.1.1节）的编码方式。也可以有其它的Fault子元素，只要它们是名域限制的。

4.4.1 SOAP 错误代码

在描述这个规范中定义的错误时，这一节中定义的Faultcode值必须用在faultcode元素中。这些faultcode值得名域标志符为"http://schemas.xmlsoap.org/soap/envelope/"。定义这个规范之外的方法时推荐（不要求）使用这个名域。缺省的SOAP faultcode值以可扩展的方式定义，允许定义新的SOAP faultcode值，并与现有的faultcode值向后兼容。使用的机制类似于HTTP中定义的1xx, 2xx,3xx等基本的状态类（见[5]第10节），不过，它们定义为XML合法名（见 [8] 第3节 ），而不是整数。 字符"."（点）作为faultcode的分隔符，点左边的错误代码比右边的错误代码更为普通。如：

Client.Authentication

这篇文档中定义的faultcode值是：

名称	含义
VersionMismatch	处理方发现SOAP封装元素有不合法的名域（见4.1.2节）
MustUnderstand	处理方不理解或者不服从一个包含值为"1"的

mustUnderstand	属性的 SOAP头元素的直接子元素。（见4.2.3节）

Client

Client错误类表示消息的格式错误或者不包含适当的正确信息。例如，消息可能缺少正确的认证和支付信息。一般地，它表示消息不能不作修改就重发。参见4.4节

SOAP Fault detail子元素的描述。

Server

Server错误类表示由于消息的处理过程而不是消息的内容本身使得消息消息不能正确的处理。例如，处理消息时可能要与其它处理器通信，但它没有响应。这个消息可能在迟一点的时间处理成功。 SOAP Fault子元素的详细信息参见4.4节

5. SOAP编码

SOAP编码格式基于一个简单的类型系统，概括了程序语言，数据库和半结构化数据等类型系统的共同特性。一个类型或者是一个简单的（标量的）类型，或者是由几个部分组合而成的复合类型，其中每个部分都有自己的类型。以下将详细描述这些类型。这一节定义了类型化对象的序列化规则。它分两个层次。首先，给定一个与类型系统的符号系统一致的Schema（译者注：这里的schema不是符合XML语法的schema，而仅仅表示广义的用于表示消息结构的定义方式），就构造了XML语法的Schema。然后，给定一个类型系统的Schema和与这个Schema一致的特定的值，就构造了一个XML文档实例。反之，给定一个依照这些规则产生的XML文档实例和初始的Schema，就可以构造初始值的一个副本。这一节中定义的元素和属性的名域标志符
为"http://schemas.xmlsoap.org/soap/encoding/"。下面的例子都假定在上一层的元素中声明了名域。
鼓励使用这一节中描述的数据模型和编码方式，但也可以在SOAP中使用其他的数据模型和编码方式。（见4.1.1节）

5.1 XML中的编码类型规则

XML允许非常灵活的数据编码方式。SOAP定义了一个较小的规则集合。这一节在总的层次上定义了这些编码规则，下一节将描述特定类型的编码规则的细节。这一节定义的编码规则可以与第7节中所述的RPC调用和应答映射结合使用。下面的术语用来描述编码规则:

- 一个"value"是一个字符串，类型（数字，日期，枚举等等）的名或是几个简单值的组合。所有的值都有特定的类型。
- 一个"simple value"没有名部分， 如特定的字符串，整数，枚举值等等。
- 一个"compound value"是相关的值的结合，如定单，股票报表，街道地址等等。在"compound value"中，每个相关的值都潜在的以名，序数或这两者来区分。这叫作"a ccessor"。复合值的例子有定单和股票报表等等。数组也是复合值。在复合值中，多个accessor有相同的名是允许的，例如RDF就是这样做的。
- 一个"array"是一个复合值，成员值按照在数组中的位置相互区分。
- 一个"struct"也是一个复合值，成员值之间的唯一区别是accessor名，accessor名互不相同。

- 一个"simple type"是简单值的类，如叫做"string" "integer"的类，还有枚举类等等。
- 一个"compound type"是复合值的类。复合类型的例子有定单类，它们有相同的accessor名（shipTo, totalCost等），但可能会有不同的值（可能以后被设置为确定的值）。

在复合类型中，如果类型内的accessor名互不相同，但是可能与其他类型中的accessor名相同，即，accessor名加上类型名形成一个唯一的标志符，这个名叫作"局部范围名"。如果名是直接或间接的基于URI的一部分，那么不管它出现在什么类型中，这个名本身就可以唯一标志这个accessor，这样的名叫作"全局范围名"。给定了schema中相关的值的序列化信息，就可能确定某些值只与某个accessor的一个实例有关。其它情况下则无法确定。当且仅当一个accessor引用一个值，这个值才能被视为"single-reference"，如果有不止一个accessor引用它，那么就将它视为"multi-reference"。注意，可能一个确定的值在一个schema中是"single-reference"，而在另一个schema中是"multi-reference"。在语句构成上，一个元素可能是"independent" 或"embedded"。一个独立的元素指出现在序列化最顶层的任何元素。所有其它元素都是嵌入元素。虽然用xsi:type属性可以使值的结构和类型变为自描述的，但是序列化规则允许值的类型仅仅参照schema而定。这样的schema可能使用"XML Schema Part 1: Structures" [10]和"XML Schema Part 2: Datatypes" [11]中描述的符号系统，也可能使用其它符号系统。注意，虽然序列化规则可以用于除了数组和结构之外的复合类型，但是许多schema仅仅包含数组和结构类型。序列化规则如下:

所有的值以元素内容的形式表示。一个multi-reference值必须表示为一个独立元素的内容，而一个single-reference值最好不要这样表示（也可以这样表示）。对于每个具有值的元素，值的类型时必须用下述三种方式之一描述：

- 所属元素实例有xsi:type属性
- 所属元素是一个有SOAP-ENC:arrayType 属性（该属性可能是缺省的）的元素的子元素，或者
- 所属元素的名具有特定的类型，类型可以由schema确定。

一个简单值表示为字符数据，即没有任何子元素。每个简单值必须具有一个类型，这个类型或者是XML Schemas Specification, part 2 [11]有的类型，或者具有源类型（参见5.2节）。一个复合值编码成一个元素的序列，每个accessor用一个嵌入元素表示，该元素的元素名和accessor的名一致。如果accessor的名是局部于其所属的类型的，则该元素的元素名不是合格的，否则对应的元素名是合格的。(参见5.4节)
一个multi-reference的简单值或复合值编码成一个独立的元素，这个元素包含一个局部的无需校验的属性，属性名为"id"，类型为"ID"（依照XML Specification [7]）。值的每个accessor对应一个空元素，该元素有一个局部的，无需校验的属性，属性名为"href"，类型为" uri-reference "（依照XML Schema Specification [11]），"href"属性的值引用了相对应的独立元素的URI标志符。字符串和字符数组表示为multi-reference的简单类型，但是特殊的规则使它们在普通的情况下能被更有效的表示（参见5.2.1节和5.2.3节）。字符串和字符数组值的accessor可能有一个名字为"id"，类型为"ID"（依照XML Specification [7]）的属性。如果这样，所有这个值的所有其它accessor编码成一个空元素，这个元素有一个局部的，无需校验的属性，属性名为"href"，类型为" uri-reference "（依照XML Schema Specification [11]），"href"属性的值引用了包含这个值的元素的URI标志符。编码时允许一个值有多个引用，就像多个不同的值有多个引用一样，但这仅在从上下文可以知道这个XML文档实例的含义没有改变时才可使用。数组是复合值（参见5.4.2节）。SOAP数组定义为具有类型"SOAP-ENC:Array"或从它衍生的类型.

SOAP数组可以时一维或多维，它们的成员以序数位置相互区分。一个数组值表示为反映这个数组的一系列元素，数组成员按升序出现。对多维数组来说，右边的这一维变化最快。每个成员元素命名为一个独立元素。（见规则2）SOAP数组可以是single-reference 或multi-reference值，因此可以表示为嵌入元素或独立元素的内容。SOAP数组必须包含一个"SOAP-ENC:arrayType"属性，它的值指定了包含元素的类型和数组的维数。"SOAP-ENC:arrayType"属性的值定义如下：

```
    arrayTypeValue = atype asize
atype = QName *( rank )
rank = "[" *( "," ) "]"
asize = "[" #length "]"
length = 1*DIGIT
```

- "atype"结构是被包含元素的类型名，它表示为QName并且作为类型限制在XML元素声明的
- "type"属性中出现（这意味着被包含元素的所有值都要与该类型一致，即在SOAP-ENC:a rrayType中引用的类型必须是每个数组成员的类型或超类型）。在arrays of arrays or "jagged arrays"的情况下，类型组件编码为"innermost"类型且在从第一层开始的嵌套数组的每一层中，类型名后都跟随一个rank结构。多维数组编码时从第一维起，每一维之间用逗号隔开。
- "asize"结构包含一个以逗号分隔的列表，数值0，1或其它整数表示数组每一维的长度。整数0表示没有指定详细的大小，但是可能在检查数组实际成员的大小后确定。例如，一个5个成员的整型数组的arrayTypeValue值为"int[][5]"，它的atype值是int[]"，asize值是"[5]"。同样，一个3个成员的两维整型数组的arrayTypeValue值为"int[,][3]"，它的atype值是int[,]"，asize值是"[3]"。

一个SOAP数组成员可能包含一个"SOAP-ENC:offset"属性表示这一项在整个数组中的位置偏移值。这被用来指示一个部分储值数组（见5.4.2.1节）的位置偏移值。同样，一个数组成员可能包含一个"SOAP-ENC:position"属性表示这一项在整个数组中的位置，这被用来描述稀疏数组（见5.4.2.2节）的成员。"SOAP-ENC:offset" 和"SOAP-ENC:position"属性值的定义如下：

```
    arrayPoint = "[" #length "]"
```

偏移值和位置从0开始

NULL值或缺省值可能通过省略accssor元素来表示。NULL值也可能通过一个包含值为'1'的xsi:null属性的accssor元素来表示，其它的依赖于应用程序的属性和值也可能用来表示NULL值。注意，规则2允许独立的元素和数组成员名不同于值类型的元素。

5.2 简单类型

SOAP采用了"XML Schema Part 2: Datatypes"规范[11]"Built-in datatypes"节中的所有类型作为简单类型，包括值和取值范围。例如：

类型	举例
int	58502
float	3141592653589 79E+1

negativeInteger	-32768
string	Louis "Satchmo" Armstrong

在XML Schema规范中声明的数据类型可以直接用在元素schema中，也可以使用从这些类型衍生的新类型。一个schema和对应的具有这些类型的元素的数据实例的例子如下所示：

```
<element name="age" type="int"/>
<element name="height" type="float"/>
<element name="displacement" type="negativeInteger"/>
<element name="color">
<simpleType base="xsd:string">
<enumeration value="Green"/>
<enumeration value="Blue"/>
</simpleType>
</element>
<age>45</age>
<height>5.9</height>
<displacement>-450</displacement>
<color>Blue</color>
```

所有简单值必须编码为元素的内容，它的类型或者在"XML Schema Part 2: Datatypes"规范[11]中定义过，或者是基于一个用XML Schema规范提供的机制能推衍生出的类型。如果一个简单值编码为独立元素或异质数组成员，那么有一个对应于数据类型的元素声明将会很方便。因为"XML Schema Part 2: Datatypes"规范[11]包括了类型定义，但是不包括对应的元素声明，SOAP-ENC schema和名域为每个简单数据类型声明了一个元素，如<SOAP-ENC:int id="int1">45</SOAP-ENC:int>

5.2.1 字符串

字符串数据类型的定义在"XML Schema Part 2: Datatypes"规范[11]中。注意，这不同于许多数据库和程序语言中的"string"类型，特别的，字符串数据类型可能禁止某些在那些语言中允许的字符。（这些值必须用xsd:string之外的数据类型表示）一个字符串可能编码为一个single-reference 或 multi-reference值。包含字符串值的元素可能有一个"id"属性。附加的accessor元素可能有对应的"href"属性。
例如，同一字符串的两个accessor可能以如下形式出现：

```
<greeting id="String-0">Hello</greeting>
<salutation href="#String-0"/>
```

但是，如果两个accessor参考同一字符串实例（或字符串的子类型），这不是一个实质问题，它们可以编码为两个single-reference值，如下所示：

```
<greeting>Hello</greeting>
```

```
<salutation>Hello</salutation>
```

这个例子的schema片断如下所示：

```
<element name="greeting" type="SOAP-ENC:string"/>
<element name="salutation" type="SOAP-ENC:string"/>
```

在这个例子中，SOAP-ENC:string类型用作元素的类型，这是声明数据类型是"xsd:string"且允许"id"和"href"属性的元素的简便方法。精确定义参见SOAP编码schema。Schemas可以使用这些源自SOAP编码schema的声明，但也可以不这样做。

5.2.2 Enumerations

"XML Schema Part 2: Datatypes"规范 [11] 定义了"enumeration."机制。SOAP数据模型直接采用了这种机制。但是，由于程序语言和其它语言在定义枚举时通常有些不同，所以我们在这里详细阐述了它的概念并描述了一个列表成员的可能取的值是如何编码的。"Enumeration"作为一个概念表示不同的名字的集合。一个特定的枚举就是对应于特定的基类型的不同的值的列表。例如，颜色集合("Green", "Blue", "Brown")可以定义为基于字符串类型的枚举，("1", "3", "5")可能是一个基于整型数的枚举，等等。"XML Schema Part 2: Datatypes" [11]支持除了布尔型以外所有简单类型的枚举。"XML Schema Part 1: Structures"规范[10]的语言可以用来定义枚举类型。如果schema由另一个没有特定基类型适用的符号系统生成，就使用"string"。在下面schema的例子中，"EyeColor"定义为字符串，可能的值是"Green", "Blue", 或"Brown"的枚举，数据实例按照schema显示如下。

```
<element name="EyeColor" type="tns:EyeColor"/>
<simpleType name="EyeColor" base="xsd:string">
<enumeration value="Green"/>
<enumeration value="Blue"/>
<enumeration value="Brown"/>
</simpleType>
<Person>
<Name>Henry Ford</Name>
<Age>32</Age>
<EyeColor>Brown</EyeColor>
</Person>
```

5.2.3 字符数组

一个字符数组可能编码为single-reference 或multi-reference值。字符数组的编码规则与字符串的编码规则类似。特别的，包含字符数组的元素值可能由一个"id"属性，附加的accssor元素可能有相应的"href"属性。推荐使用定义在XML Schemas [10][11]中的'base64'编码（使用在2045 [13]中定义的base64编码算法）表示模糊字符数组。不过，由于行长度（line length）的限制，通常在MIME中应用base64编码，SOAP中一般不应用

base64编码。但是提供了"SOAP-ENC:base64"子类型使之能用于SOAP。

```
<picture xsi:type="SOAP-ENC:base64">
aG93IG5vDyBicm73biBjb3cNCg==
</picture>
```

5.3 多态accessor

许多语言允许能够多态访问多种类型值的accessor，每种类型在运行时可用。一个多态accessor实例必须包含一个"xsi:type"属性描述实际值的类型。例如，一个名为"cost"类型值为"xsd:float"的多态accessor编码如下：

<cost xsi:type="xsd:float">29.95</cost>与之对比，类型值不变的accessor编码如下：

<cost>29.95</cost>

5.4 Compound types复合类型

SOAP定义了与下列常在程序语言中出现的结构性模式对应的类型：

- 结构:一个"struct"是一个复合值，它的成员值的唯一区别是accessor名称，任意两个accessor名称都不相同。
- 数组:一个"array"是一个复合值，它的成员值的唯一区别是序数位置。

SOAP也允许结构和数组之外的其它数据的序列化，例如Directed-Labeled-Graph Data Model之类的数据中，单个节点有许多不同的accssor，有些不止出现一次。SOAP序列化规则不要求底层的数据模型在accssor之间区分次序，但如果有这样的次序的话，这些accssor必须按照这个顺序编码。

5.4.1 复合值，结构和值引用

复合值的成员编码为accessor元素。当accessor由名区分时（如结构），accessor名即作为元素名。名局部于类型的accessor有不受限的名，其它的accessor则有受限的名。下面的例子是类型为"Book"的结构：

```
<e:Book>
<author>Henry Ford</author>
<preface>Prefatory text</preface>
<intro>This is a book.</intro>
</e:Book>
```

以下是描述上面结构的schema片断：

```
<element name="Book">
<complexType>
```

```
<element name="author" type="xsd:string"/>
<element name="preface" type="xsd:string"/>
<element name="intro" type="xsd:string"/>
</complexType>
</e:Book>
```

以下是一个同时具有简单和复杂成员类型的例子。它显示两层引用。注意"Author"accssor元素的"href"属性是对相应具有"id"属性的值的引用。"Address"与之类似。

```
<e:Book>
<title>My Life and Work</title>
<author href="#Person-1"/>
</e:Book>
<e:Person id="Person-1">
<name>Henry Ford</name>
<address href="#Address-2"/>
</e:Person>
<e:Address id="Address-2">
<email>mailto:henryford@hotmail.com</email>
<web>http://www.henryford.com</web>
</e:Address>
```

当"Person"的值和"Address"的值是multi-reference时，上面的形式是正确的。如果它们是single-reference，就必须用嵌入的形式，如下所示：

```
<e:Book>
<title>My Life and Work</title>
<author>
<name>Henry Ford</name>
<address>
<email>mailto:henryford@hotmail.com</email>
<web>http://www.henryford.com</web>
</address>
</author>
</e:Book>
```

如果添加一个限制，任意两个人都不会有相同的地址，并且地址可以是街道或Email地址，一本书可以有两个作者，编码如下：

```
<e:Book>
```

```
<title>My Life and Work</title>
<firstauthor href="#Person-1"/>
<secondauthor href="#Person-2"/>
</e:Book>
<e:Person id="Person-1">
<name>Henry Ford</name>
<address xsi:type="m:Electronic-address">
<email>mailto:henryford@hotmail.com</email>
<web>http://www.henryford.com</web>
</address>
</e:Person>
<e:Person id="Person-2">
<name>Samuel Crowther</name>
<address xsi:type="n:Street-address">
<street>Martin Luther King Rd</street>
<city>Raleigh</city>
<state>North Carolina</state>
</address>
</e:Person>
```

序列化可以包含对不在同一个资源的值的引用：

```
<e:Book>
<title>Paradise Lost</title>
<firstauthor href="http://www.dartmouth.edu/~milton/"/>
</e:Book>
```

以下是描述上面结构的schema片断：

```
<element name="Book" type="tns:Book"/>
<complexType name="Book">
<!-- Either the following group must occur or else the
href attribute must appear, but not both. -->
<sequence minOccurs="0" maxOccurs="1">
<element name="title" type="xsd:string"/>
<element name="firstauthor" type="tns:Person"/>
<element name="secondauthor" type="tns:Person"/>
</sequence>
<attribute name="href" type="uriReference"/>
```

```
<attribute name="id" type="ID"/>
<anyAttribute namespace="##other"/>
</complexType>
<element name="Person" base="tns:Person"/>
<complexType name="Person">
<!-- Either the following group must occur or else the
href attribute must appear, but not both. -->
<sequence minOccurs="0" maxOccurs="1">
<element name="name" type="xsd:string"/>
<element name="address" type="tns:Address"/>
</sequence>
<attribute name="href" type="uriReference"/>
<attribute name="id" type="ID"/>
<anyAttribute namespace="##other"/>
</complexType>
<element name="Address" base="tns:Address"/>
<complexType name="Address">
<!-- Either the following group must occur or else the
href attribute must appear, but not both. -->
<sequence minOccurs="0" maxOccurs="1">
<element name="street" type="xsd:string"/>
<element name="city" type="xsd:string"/>
<element name="state" type="xsd:string"/>
</sequence>
<attribute name="href" type="uriReference"/>
<attribute name="id" type="ID"/>
<anyAttribute namespace="##other"/>
</complexType>
```

5.4.2 数组

SOAP数组定义为具有"SOAP-ENC:Array"类型或一个从"SOAP-ENC:Array"衍生的类型（参见规则8）。数组表示为元素值，对元素的名没有特别的约束（正如元素值并不约束它们所属的元素）。数组可以包含任意类型的元素，包括嵌套数组。可以创建新的类型（受SOAP-ENC:Array
类型限制）来表示数组，如整数数组或某些用户定义的枚举。数组值表示为组成这个数组的项的元素的规则序列。在数组值中，元素名对于区分accesor并不重要。元素可以有任意的名。实际上，元素常常用它们在schema中暗示或确定的数组类型来命名元素。并且一般情况下对于复合值来说，如果数组中数组项的值是single-reference值，则这个数组项包含它的值，否则，该数组项通过"href"属性引用这个值。下面的例子是一个整型数组的schema片断：

```
<element name="myFavoriteNumbers"
type="SOAP-ENC:Array"/>
<myFavoriteNumbers
SOAP-ENC:arrayType="xsd:int[2]">
<number>3</number>
<number>4</number>
</myFavoriteNumbers>
```

在这个例子中，数组"myFavoriteNumbers"包括几个成员，每个成员是一个类型为SOAP-ENC:int的值。注意SOAP-ENC:Array允许不受限制的元素名，它们不传达任何类型信息，所以在使用时，或者它们有xsi:type属性，或者它们所属的元素有SOAP-ENC:arrayType属性。自然，由SOAP-ENC:Array衍生的类型可以声明局部元素，但这种情况下要包括类型信息。上面已经提到，SOAP-ENC schema包含了元素的声明，元素名与"XML Schema Part 2: Datatypes"规范[11]中的简单类型一致。其中包括了对"Array"的声明。于是，我们可以这样写：

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:int[2]">
<SOAP-ENC:int>3</SOAP-ENC:int>
<SOAP-ENC:int>4</SOAP-ENC:int>
</SOAP-ENC:Array>
```

数组可以包含特定arrayType的任意子类型的实例。即，数组成员可以是arryType属性值指定的类型的任意子类型，这个类型对于arrayType属性中指定的类型来说是可替换的（根据schema中的替换规则）。例如，一个整型数组可以包含从整型衍生的任意类型（如"int"或任意用户定义的从整型衍生的类型）。同样，一个"address"数组可能包含一个address的受限类型或扩展类型如"internationalAddress"。因为提供的SOAP-ENC:Array类型允许任意类型的成员，所以可以包含任意类型的混合除非使用arrayType属性加以特别的限制。在实例中，可以使用xsi:type指定成员元素的类型，或通过schema中成员元素的声明来指定。下面是两个例子。

```
<SOAP-ENC:Array SOAP-ENC:arrayType="SOAP-ENC:ur-type[4]">
<thing xsi:type="xsd:int">12345</thing>
<thing xsi:type="xsd:decimal">6.789</thing>
<thing xsi:type="xsd:string">
Of Mans First Disobedience, and the Fruit
Of that Forbidden Tree, whose mortal tast
Brought Death into the World, and all our woe,
</thing>
<thing xsi:type="xsd:uriReference"> http://www.dartmouth.edu/~milton/reading_room/
</thing>
</SOAP-ENC:Array>
<SOAP-ENC:Array SOAP-ENC:arrayType="SOAP-ENC:ur-type[4]">
<SOAP-ENC:int>12345</SOAP-ENC:int>
```

```
<SOAP-ENC:decimal>6.789</SOAP-ENC:decimal>
<xsd:string>
Of Mans First Disobedience, and the Fruit
Of that Forbidden Tree, whose mortal tast
Brought Death into the World, and all our woe,
</xsd:string>
<SOAP-ENC:uriReference> http://www.dartmouth.edu/~milton/reading_room/ </SOAP-
ENC:uriReference >
</SOAP-ENC:Array>
```

数组值可以是结构或其它复合值。例如"xyz:Order"结构数组：

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xyz:Order[2]">
<Order>
<Product>Apple</Product>
<Price>1.56</Price>
</Order>
<Order>
<Product>Peach</Product>
<Price>1.48</Price>
</Order>
</SOAP-ENC:Array>
```

数组成员值也可以是数组。下例是两个字符串数组组成的数组：

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[][2]">
<item href="#array-1"/>
<item href="#array-2"/>
</SOAP-ENC:Array>
<SOAP-ENC:Array id="array-1" SOAP-ENC:arrayType="xsd:string[2]">
<item>r1c1</item>
<item>r1c2</item>
<item>r1c3</item>
</SOAP-ENC:Array>
<SOAP-ENC:Array id="array-2" SOAP-ENC:arrayType="xsd:string[2]">
<item>r2c1</item>
<item>r2c2</item>
</SOAP-ENC:Array>
```

包含数组的元素无需命名为"SOAP-ENC:Array"。它可以有任意的名，只要元素的类型是SOAP-ENC:Array或由之衍生的类型。例如，下面是一个schema片断和与之一致的数组实例。

```
<simpleType name="phoneNumber" base="string"/>
<element name="ArrayOfPhoneNumbers">
<complexType base="SOAP-ENC:Array">
<element name="phoneNumber" type="tns:phoneNumber" maxOccurs="unbounded" />
</complexType>
<anyAttribute/>
</element>
<xyz:ArrayOfPhoneNumbers SOAP-ENC:arrayType="xyz:phoneNumber[2]">
<phoneNumber>206-555-1212</phoneNumber>
<phoneNumber>1-888-123-4567</phoneNumber>
</xyz:ArrayOfPhoneNumbers>
```

数组可能是多维的。在这种情况下，在arrayType属性的asize部分将不止有一个值：

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[2,3]">
<item>r1c1</item>
<item>r1c2</item>
<item>r1c3</item>
<item>r2c1</item>
<item>r2c2</item>
<item>r2c3</item>
</SOAP-ENC:Array>
```

虽然上面的例子把数组编码为独立的元素，但元素值也可以是嵌入形式，而且若元素值是single reference时，必须编码为嵌入形式。下例是一个schema片断，电话号码数组嵌入到一个类型为"Person"的结构中，并且通过accessor "phone-numbers"访问它：

```
<simpleType name="phoneNumber" base="string"/>
<element name="ArrayOfPhoneNumbers">
<complexType base="SOAP-ENC:Array">
<element name="phoneNumber" type="tns:phoneNumber" maxOccurs="unbounded"/>
</complexType>
<anyAttribute/>
</element>
<element name="Person">
<complexType>
```

```
<element name="name" type="string"/>
<element name="phoneNumbers" type="tns:ArrayOfPhoneNumbers"/>
</complexType>
</element>
<xyz:Person>
<name>John Hancock</name>
<phoneNumbers SOAP-ENC:arrayType="xyz:phoneNumber[2]">
<phoneNumber>206-555-1212</phoneNumber>
<phoneNumber>1-888-123-4567</phoneNumber>
</phoneNumbers>
</xyz:Person>
```

下面的例子中，数组值为single-reference，被编码为嵌入元素，包含它的元素名即为入口名：

```
<xyz:PurchaseOrder>
<CustomerName>Henry Ford</CustomerName>
<ShipTo>
<Street>5th Ave</Street>
<City>New York</City>
<State>NY</State>
<Zip>10010</Zip>
</ShipTo>
<PurchaseLineItems SOAP-ENC:arrayType="Order[2]">
<Order>
<Product>Apple</Product>
<Price>1.56</Price>
</Order>
<Order>
<Product>Peach</Product>
<Price>1.48</Price>
</Order>
</PurchaseLineItems>
</xyz:PurchaseOrder>
```

5.4.2.1 部分储值（partially transmitted）数组

SOAP提供了对部分储值（partially transmitted）数组的支持，如某些上下文中的可变数组。一个partially transmitted 数组由一个"SOAP-ENC:offset"属性（从第一个transmitted的元素开始的偏移量，基于0）指示。如果省略，偏移量取0。下面的例子中数组的大小为5，但只有从0起，第三和第四个元素被储值。

```
<SOAP-ENC:Array ;SOAP-ENC:arrayType="xsd:string[5]" ;SOAP-ENC:offset="[2]">
<item>The third element</item>
<item>The fourth element</item>
</SOAP-ENC:Array>
```

5.4.2.2 稀疏数组Sparse Arrays

SOAP提供了对稀疏数组的支持。每个表示成员值的元素包含一个"SOAP-ENC:position"属性，用来指示它在数组中的位置。下例是两维字符串稀疏数组的例子，数组大小是4，但只用到第2个。

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[,][4]">
<SOAP-ENC:Array href="#array-1" SOAP-ENC:position="[2]"/>
</SOAP-ENC:Array>
<SOAP-ENC:Array id="array-1" SOAP-ENC:arrayType="xsd:string[10,10]">
<item SOAP-ENC:position="[2,2]">Third row, third col</item>
<item SOAP-ENC:position="[7,2]">Eighth row, third col</item>
</SOAP-ENC:Array>
```

如果对array-1的引用仅发生在数组内部，上例也可以编码如下：

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[,][4]">
<SOAP-ENC:Array SOAP-ENC:position="[2]" SOAP-ENC:arrayType="xsd:string[10, 10]>
<item SOAP-ENC:position="[2,2]">Third row, third col</item>
<item SOAP-ENC:position="[7,2]">Eighth row, third col</item>
</SOAP-ENC:Array>
</SOAP-ENC:Array>
```

5.4.3 一般复合类型

在这里提到的编码规则不仅仅限于accessor名已知的情况，如果accessor名是运行环境下实时获得的，编码规则同样适用，也就是说accessor编码成一个元素名与accessor名匹配的元素，同时accessor可能包含或者引用该元素的值。如果accessor包含类型不能事先确定的值，它必须包含一个合适的属性xsi:type 。类似地，上述引用的规则已经足够用于复合类型的序列化，这些复合类型可能包含用名区分的accessors（结构）和用名及序数位置区分的accessors。（可能包含重复的accessor） 实际上这并不要求任何schema模式包含这些类型，但更为准确的说法是：一个类型模型（type-model）schema如果有这些类型，就可以构造一个符合XML句法规则的schema和XML文档实例。

```
<xyz:PurchaseOrder>
<CustomerName>Henry Ford</CustomerName>
<ShipTo>
<Street>5th Ave</Street>
```

```
<City>New York</City>
<State>NY</State>
<Zip>10010</Zip>
</ShipTo>
<PurchaseLineItems>
<Order>
<Product>Apple</Product>
<Price>1.56</Price>
</Order>
<Order>
<Product>Peach</Product>
<Price>1.48</Price>
</Order>
</PurchaseLineItems>
</xyz:PurchaseOrder>
```

类似地，将一个结构上类似数组但实际上不是一个 SOAP-ENC:Array类型或者 SOAP-ENC:Array子类型的复合值序列化同样是允许的，例如：

```
<PurchaseLineItems>
<Order>
<Product>Apple</Product>
<Price>1.56</Price>
</Order>
<Order>
<Product>Peach</Product>
<Price>1.48</Price>
</Order>
</PurchaseLineItems>
```

5.5 缺省值

省略accessor元素意味着或者有一个缺省值或者值不知道。具体细节依靠这个accessor，方法和上下文。例如，对于多态accessor，省略accessor一般意味着一个Null值。同样，省略布尔accessor一般意味着False值或者值不知道，省略数字accessor一般意味着值为零或者值不知道。

5.6 SOAP root属性

SOAP root 属性可用于标记一个序列化root，从而一个对象可以反序列化（deserialized），而实际上该root并不是真正的对象root。这个属性有两个可选值"1" or "0"。对象真正的roots属性值为“1” ，序列化

root但不是真正的root属性值也为“1”，元素如果要显式地指定不能为序列化root，只需将该属性设置为“0” SOAP root属性可以出现在SOAP头和SOAP体元素的任意子元素中。（译者注：SOAP root属性为0的元素不是一个独立的实体，外部的应用不能访问到该元素，但该元素可以被SOAP文档本身的其它元素访问到）SOAP root属性可以出现在SOAP头和SOAP体元素的任意子元素中。这个属性没有缺省值。

6. 在HTTP中使用SOAP

这一节讲述了如何在HTTP中使用SOAP。把SOAP绑定到HTTP，无论使用或不用HTTP扩展框架，都有很大的好处：在利用SOAP的形式化和灵活性的同时，使用HTTP种种丰富的特性。在HTTP中携带SOAP消息，并不意味着SOAP改写了HTTP已有的语义，而是将构建在HTTP之上SOAP语义自然地对应到HTTP语义。SOAP自然地遵循HTTP的请求/应答消息模型使得SOAP的请求和应答参数可以包含在HTTP请求和应答中。注意，SOAP的中间节点与HTTP的中间节点并不等同，即，不要期望一个根据HTTP连接头中的域寻址到的HTTP中间节点能够检查或处理HTTP请求中的SOAP消息。
在HTTP消息中包含SOAP实体时，按照RFC2376[3] HTTP应用程序必须使用媒体类型 "text/xml"。

6.1 SOAP HTTP请求

虽然SOAP可能与各种HTTP请求方式相结合，但是绑定仅定义了在HTTP POST请求中包含SOAP消息。（第7节中描述了如何在RPC中使用SOAP，第6.3节描述了如何使用HTTP扩展框架）

6.1.1 HTTP头中SOAPAction域

一个HTTP请求头中的SOAPAction域用来指出这是一个SOAP HTTP请求，它的值是所要的URI。在格式、URI的特性和可解析性上没有任何限制。当HTTP客户发出SOAP HTTP请求时必须使用在HTTP头中使用这个域。

```
    soapaction = "SOAPAction" ":" [ <"> URI-reference <"> ]
URI-reference = <as defined in RFC 2396 [4]>
```

HTTP头中SOAPAction域使服务器（如防火墙）能正确的过滤HTTP中SOAP请求消息。如果这个域的值是空字符串（""），表示SOAP消息的目标就是HTTP请求的URI。这个域没有值表示没有SOAP消息的目标的信息。例子:

```
SOAPAction: "http://electrocommerce.org/abc#MyMessage"
SOAPAction: "myapp.sdl"
SOAPAction: ""
SOAPAction:
```

6.2 SOAP HTTP应答

SOAP HTTP遵循HTTP 中表示通信状态信息的HTTP状态码的语义。例如，2xx状态码表示这个包含了SOAP组件的客户请求已经被成功的收到，理解和接受。在处理请求时如果发生错误，SOAP HTTP服务器必须发出应答HTTP 500 "Internal Server Error"，并在这个应答中包含一个SOAP Fault元素（见4.4节）表示这个SOAP处理错误。

6.3 HTTP扩展框架

一个SOAP消息可以与HTTP扩展框架 [6]一起使用以区分是否有SOAP HTTP请求和它的目标。是使用扩展框架或是普通的HTTP关系到通信各方的策略和能力。通过使用一个必需的扩展声明和"M-"HTTP方法名前缀，客户可以强制使用HTTP扩展框架。服务器可以使用HTTP状态码510 "Not Extended"强制使用HTTP扩展框架。也就是说，使用一个额外的来回，任何一方都可以发现另一方的策略并依照执行。用来表示SOAP使用了扩展框架的扩展标志符是：http://schemas.xmlsoap.org/soap/envelope/

6.4 SOAP HTTP举例

例3 使用POST的SOAP HTTP

```
POST /StockQuote HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://electrocommerce.org/abc#MyMessage"
<SOAP-ENV:Envelope...
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope...
```

例4 使用扩展框架的SOAP HTTP

```
M-POST /StockQuote HTTP/1.1
Man: "http://schemas.xmlsoap.org/soap/envelope/"; ns=NNNN
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
NNNN-SOAPAction: "http://electrocommerce.org/abc#MyMessage"
<SOAP-ENV:Envelope...
HTTP/1.1 200 OK

Ext:
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope...
```

7. 在RPC中使用SOAP

设计SOAP的目的之一就是利用XML的扩展性和灵活性来封装和交换RPC调用。这一节定义了远程过程调用和应答的统一表示形式。虽然可以预计到这种表示形式最可能被用于与第5节中定义的编码方式相结合，但也可能有其它的表示形式。SOAP的encodingStyle属性（见4.3.2节）可以用来表明方法调用和应答都使用这一节所指定的表示方式。在RPC中使用SOAP和SOAP协议绑定（见第6节）是紧密相关的。在使用HTTP作为绑定协议时，一个RPC

调用自然地映射到一个HTTP请求，RPC应答同样映射到HTTP应答。但是，在RPC中使用SOAP并不限于绑定HTTP协议。

要进行方法调用，以下的信息是必需的：

- 目标对象的URI
- 方法名
- 方法signature（可选）
- 方法的参数
- 头数据（可选）

SOAP依靠协议绑定提供传送URI的机制。例如，对HTTP来说，请求的URI指出了调用的来源 。除了必须是一个合法的URI之外，SOAP对一个地址的格式没有任何限制。（更多URI的信息参见 [4]）

7.1 RPC和SOAP体

RPC方法调用和应答都包含在SOAP Body元素中（见4.3节），它们使用如下的表示形式：

- 一个方法调用用一个结构表示
- 一个方法调用被看作一个单个的结构，每个[in]和[in/out]参数有一个accessor。结构的名和类型与方法相同。每个[in]和[in/out]参数都被看作一个accessor，这个accessor的名和类型与参数的名和类型相对应。它们的出现顺序和方法中定义的参数顺序相同。
- 一个方法应答用一个结构表示。
- 一个方法应答被看作一个单个的结构，返回值和每个[in]和[in/out]参数有一个accessor。第一个accessor是返回值，之后是参数accessor，参数accessor的出现顺序和方法中定义的参数顺序相同。每个参数accessor的名称和类型与参数的名称和类型相对应。返回值accessor的名称并不重要。同样，结构的名称也不重要，不过，通常在方法名称的后面加上字符串"Response"作为结构的名称。

方法错误使用SOAP Fault元素（见4.4节）表示。如果绑定的协议有额外的规则表示错误，则这些规则也必须要遵从。正如上面所述，方法调用和应答结构可以按照第5节中规则编码，或者用encodingStyle属性（见4.1.1节）指定编码方式。应用程序可以处理缺少参数的请求，但是可能返回一个错误。因为返回结果表示调用成功，错误表示调用失败，所以，在方法应答中同时包含返回结果和错误是错误的。

7.2 RPC和SOAP头

在RPC编码中，可能会有与方法请求有关但不是正规的方法signature的附加信息。如果这样，它必须作为SOAP头元素的子元素。使用这种头元素的一个例子是在消息中传递事务ID。由于事务ID不是方法signature的一部分，通常由底层的组件而不是应用程序代码控制，所以没有一种直接的方法在调用中传递这个必要的信息。通过在头中添加一个给定名字的条目，接收方的事务管理器就可以析取这个事务ID，而且不影响远程过程调用的代码。

8. 安全性考虑

这篇文档中没有涉及完整性和保密性，这些问题将在以后的版本中描述。

9. 参考文献

[1] S. Bradner, "The Internet Standards Process -- Revision 3", RFC2026, Harvard University, October 1996

[2] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997

[3] E. Whitehead, M. Murata, "XML Media Types", RFC2376, UC Irvine, Fuji Xerox Info. Systems, July 1998

[4] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, MIT/LCS, U.C.Irvine, Xerox Corporation, A ugust 1998.

[5] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee, "Hypert ext Transfer Protocol -- HTTP/1.1", RFC 2616, U.C. Irvine, DEC W3C/MIT, DEC,W3C/MIT, W3C/MIT, January 1997

[6] H. Nielsen, P. Leach, S. Lawrence, "An HTTP Extension Framework", RFC 2774, Microsoft, Microsoft, Agranat Systems

[7] W3C Recommendation "The XML Specification"

[8] W3C Recommendation "Namespaces in XML"

[9] W3C Working Draft "XML Linking Language". This is work in progress.

[10] W3C Working Draft "XML Schema Part 1: Structures". This is work in progress.

[11] W3C Working Draft "XML Schema Part 2: Datatypes". This is work in progress.

[12] Transfer Syntax NDR, in "DCE 1.1: Remote Procedure Call"

[13] N. Freed, N. Borenstein, "Multipurpose Internet Mail Extensions (MIME)Part One: Format of Internet Message Bodies", RFC2045, Innosoft, First Virtu al, November 1996

10。 附录

A. SOAP封装举例

A.1 请求编码举例

例5 类似于例1，但有一个必要的头

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
```

```
<SOAP-ENV:Header>
<t:Transaction
xmlns:t="some-URI"
SOAP-ENV:mustUnderstand="1">
5
</t:Transaction>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>DEF</symbol>
</m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

例6 类似于例1，但有多个请求参数

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
<SOAP-ENV:Body>
<m:GetLastTradePriceDetailed
xmlns:m="Some-URI">
<Symbol>DEF</Symbol>
<Company>DEF Corp</Company>
<Price>34.1</Price>
</m:GetLastTradePriceDetailed>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A.2 应答编码举例

例7 与例2类似，但有必要的头部

```
HTTP/1.1 200 OK
```

```
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
<SOAP-ENV:Header>
<t:Transaction xmlns:t="some-URI" xsi:type="xsd:int" mustUnderstand="1"> 5
</t:Transaction>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<m:GetLastTradePriceResponse xmlns:m="Some-URI">
<Price>34.5</Price>
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

例8 与例2类似，但有一个结构

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
<SOAP-ENV:Body>
<m:GetLastTradePriceResponse
xmlns:m="Some-URI">
<PriceAndVolume>
<LastTradePrice> 34.5 </LastTradePrice>
<DayVolume> 10000 </DayVolume>
</PriceAndVolume>
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

例9 与例2类似，但处理必要的头出错

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
```

```
Content-Length: nnnn
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode>SOAP-ENV:MustUnderstand</faultcode>
<faultstring>SOAP Must Understand Error</faultstring>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

例10 与例2类似，但处理Body出错

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode>SOAP-ENV:Server</faultcode>
<faultstring>Server Error</faultstring>
<detail>
<e:myfaultdetails xmlns:e="Some-URI">
<message>
My application didn't work
</message>
<errorcode> 1001 </errorcode>
</e:myfaultdetails>
</detail>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```