**Capítulo 2. Introduzindo o SOAP**

O lugar do SOAP na pilha de tecnologias de serviços web é o de um protocolo de empacotamento padronizado para as mensagens compartilhadas por aplicações. A especificação define nada mais que um simples envelope baseado em XML para a informação que está sendo transferida, e um conjunto de regras para traduzir tipos de dados específicos de aplicação e plataforma em representações XML. O design do SOAP o torna adequado para uma grande variedade de padrões de mensageria e integração de aplicações. Isso, em grande parte, contribui para sua crescente popularidade.

Este capítulo explica as partes do padrão SOAP. Ele cobre o formato da mensagem, o mecanismo de relatório de exceções (falhas), e o sistema para codificação de valores em XML. Discute o uso do SOAP sobre transportes que não sejam HTTP, e conclui com reflexões sobre o futuro do SOAP. Você aprenderá o que o SOAP faz e como ele faz, e obterá uma firme compreensão da flexibilidade do SOAP. Capítulos posteriores se basearão nisso para mostrar como programar com SOAP usando ferramentas que abstraem detalhes do XML.

**2.1 SOAP e XML**

SOAP é XML. Ou seja, SOAP é uma aplicação da especificação XML. Ele depende fortemente de padrões XML como XML Schema e XML Namespaces para sua definição e funcionamento. Se você não está familiarizado com qualquer um deles, provavelmente vai querer se atualizar antes de prosseguir com as informações deste capítulo (você pode encontrar informações sobre cada uma dessas especificações no site do World Wide Web Consortium em <http://www.w3c.org/>). Este livro assume que você está familiarizado com essas especificações, ao menos de forma superficial, e não gastará tempo discutindo-as. A única exceção é uma breve introdução aos tipos de dados do XML Schema no Apêndice B.

**2.1.1 Mensageria XML**

Mensageria XML é onde aplicações trocam informações usando documentos XML (veja Figura 2-1). Ela fornece uma maneira flexível para aplicações se comunicarem e forma a base do SOAP.

Uma mensagem pode ser qualquer coisa: uma ordem de compra, uma solicitação de preço de ação atual, uma consulta para um mecanismo de busca, uma lista de voos disponíveis para Los Angeles, ou qualquer outro conjunto de informações que possa ser relevante para uma aplicação específica.



Como o XML não está vinculado a uma aplicação, sistema operacional ou linguagem de programação específica, mensagens XML podem ser usadas em todos os ambientes. Um programa em Perl no Windows pode criar um documento XML representando uma mensagem, enviá-lo para um programa Java baseado em Unix, e afetar o comportamento desse programa Java.

A ideia fundamental é que duas aplicações, independentemente do sistema operacional, linguagem de programação ou qualquer outro detalhe técnico de implementação, possam compartilhar informações usando nada mais que uma simples mensagem codificada de uma forma que ambas entendam. O SOAP fornece uma maneira padronizada de estruturar mensagens XML.

**2.1.2 RPC e EDI**

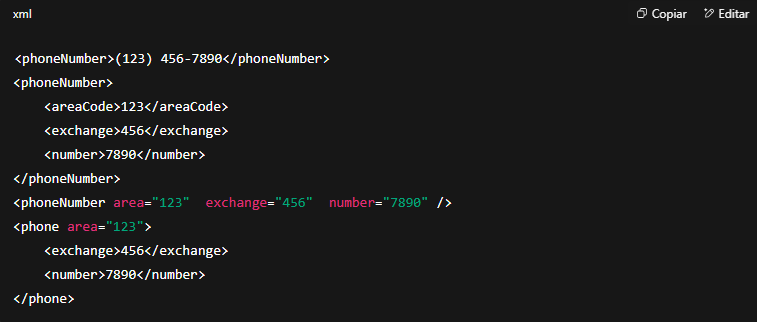
Mensageria XML, e portanto o SOAP, tem duas aplicações relacionadas: RPC e EDI. Chamada de Procedimento Remoto (RPC - Remote Procedure Call) é a base da computação distribuída, a forma pela qual um programa realiza uma chamada de procedimento (ou função, ou método) em outro, passando argumentos e recebendo valores de retorno. Intercâmbio Eletrônico de Documentos (EDI - Electronic Document Interchange) é a base das transações comerciais automatizadas, definindo um formato padrão e a interpretação de documentos e mensagens financeiras e comerciais.

Se você usar SOAP para EDI (conhecido como "SOAP estilo documento"), então o XML será uma ordem de compra, reembolso de impostos ou documento semelhante. Se você usar SOAP para RPC (conhecido, sem surpresa, como "SOAP estilo RPC"), então o XML será uma representação de parâmetros ou valores de retorno.

**2.1.3 A necessidade de uma codificação padrão**

Se você está trocando dados entre sistemas heterogêneos, precisa concordar com uma representação comum. Como você pode ver no Exemplo 2-1, um único pedaço de dado como um número de telefone pode ser representado de muitas maneiras diferentes, e igualmente válidas, em XML.

**Exemplo 2-1. Muitas representações XML de um número de telefone**



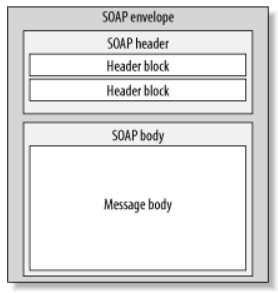
Qual é a codificação correta? Quem sabe! A correta é aquela que a aplicação está esperando. Em outras palavras, simplesmente dizer que o servidor e o cliente estão usando XML para trocar informações não é suficiente. Precisamos definir:

* Os tipos de informação que estamos trocando
* Como essa informação deve ser expressa em XML
* Como realmente enviar essa informação

Sem essas convenções acordadas, programas não podem saber como decodificar a informação recebida, mesmo que esteja codificada em XML. O SOAP fornece essas convenções.

**2.2 Mensagens SOAP**

Uma mensagem SOAP consiste em um envelope contendo um cabeçalho opcional e um corpo obrigatório, como mostrado na Figura 2-2. O cabeçalho contém blocos de informações relevantes sobre como a mensagem deve ser processada. Isso inclui configurações de roteamento e entrega, declarações de autenticação ou autorização e contextos de transação. O corpo contém a mensagem real a ser entregue e processada. Qualquer coisa que possa ser expressa em sintaxe XML pode ir no corpo de uma mensagem.



A sintaxe XML para expressar uma mensagem SOAP é baseada no namespace http://www.w3.org/2001/06/soap-envelope. Esse identificador de namespace XML aponta para um XML Schema que define a estrutura de como é uma mensagem SOAP.

Se você estivesse usando SOAP no estilo documento, poderia transferir uma ordem de compra com o XML do Exemplo 2-2.

**Exemplo 2-2. Uma ordem de compra em SOAP estilo documento**



Este exemplo ilustra todos os componentes principais da especificação SOAP Envelope. Há o <s:Envelope>, o contêiner de nível mais alto que compõe a mensagem SOAP; o opcional <s:Header>, que contém blocos adicionais de informações sobre como a carga útil do corpo deve ser processada; e o elemento obrigatório <s:Body> que contém a mensagem real a ser processada.

**2.2.1 Envelopes**

Todo elemento Envelope deve conter exatamente um elemento Body. O elemento Body pode conter quantos nós filhos forem necessários. O conteúdo do elemento Body é a mensagem. O elemento Body é definido de forma a poder conter qualquer XML válido e bem-formado que tenha qualificação de namespace e não contenha instruções de processamento ou referências a DTD (Document Type Definition).

Se um Envelope contiver um elemento Header, ele não deve conter mais que um, e ele deve aparecer como o primeiro filho do Envelope, antes do Body. O cabeçalho, assim como o corpo, pode conter qualquer XML válido, bem-formado e qualificado por namespace que o criador da mensagem SOAP queira inserir.

Cada elemento contido pelo Header é chamado de bloco de cabeçalho (*header block*). A finalidade de um bloco de cabeçalho é comunicar informações contextuais relevantes para o processamento de uma mensagem SOAP. Um exemplo pode ser um bloco de cabeçalho contendo credenciais de autenticação ou informações de roteamento de mensagem. Blocos de cabeçalho serão destacados e explicados em mais detalhes ao longo do restante do livro. No Exemplo 2-2, o bloco de cabeçalho indica que o documento possui um ID de transação "1234".

**2.2.2 Mensagens RPC**

Agora vamos ver uma mensagem no estilo RPC. Tipicamente, mensagens vêm em pares, como mostrado na Figura 2-3: a solicitação (o cliente envia informações de chamada de função para o servidor) e a resposta (o servidor envia de volta o(s) valor(es) de retorno para o cliente). O SOAP não exige que toda solicitação tenha uma resposta ou vice-versa, mas é comum ver o pareamento solicitação-resposta.



Imagine que o servidor oferece esta função, que retorna o preço de uma ação, como um serviço SOAP:



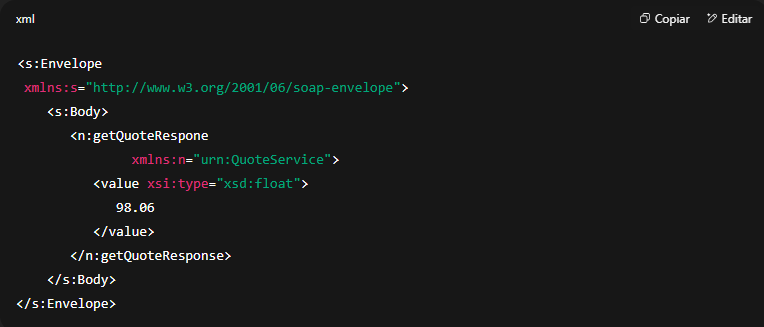
O Exemplo 2-3 ilustra uma mensagem SOAP simples no estilo RPC que representa uma solicitação para o preço atual da ação da IBM. Novamente, mostramos um bloco de cabeçalho que indica um ID de transação "1234".

**Exemplo 2-3. Mensagem SOAP no estilo RPC**



O Exemplo 2-4 é uma possível resposta que indica a operação sendo respondida e o valor solicitado da cotação da ação.

**Exemplo 2-4. Resposta SOAP à solicitação no Exemplo 2-3**



**2.2.3 O atributo mustUnderstand**

Quando uma mensagem SOAP é enviada de uma aplicação para outra, há um requisito implícito de que o destinatário deve entender como processar essa mensagem. Se o destinatário não entender a mensagem, ele deve rejeitá-la e explicar o problema ao remetente. Isso faz sentido: se a Amazon.com enviasse para a O'Reilly uma ordem de compra de 150 furadeiras elétricas, alguém da O'Reilly ligaria para alguém da Amazon.com e explicaria que a O'Reilly and Associates vende livros, não furadeiras elétricas.

Blocos de cabeçalho são diferentes. Um destinatário pode ou não entender como lidar com um bloco de cabeçalho específico e ainda assim ser capaz de processar a mensagem principal adequadamente. Se o remetente da mensagem quiser exigir que o destinatário entenda um bloco específico, ele pode adicionar o atributo mustUnderstand="true" ao bloco de cabeçalho. Se essa marca estiver presente e o destinatário não entender o bloco ao qual ela está anexada, o destinatário deve rejeitar a mensagem inteira.

No envelope getQuote que vimos anteriormente, o cabeçalho da transação contém a marca mustUnderstand="true". Como essa marca está definida, independentemente de o destinatário entender ou ser capaz de processar o corpo da mensagem (a mensagem getQuote), se ele não souber lidar com o bloco de cabeçalho da transação, a mensagem inteira deve ser rejeitada. Isso garante que o destinatário entenda transações.

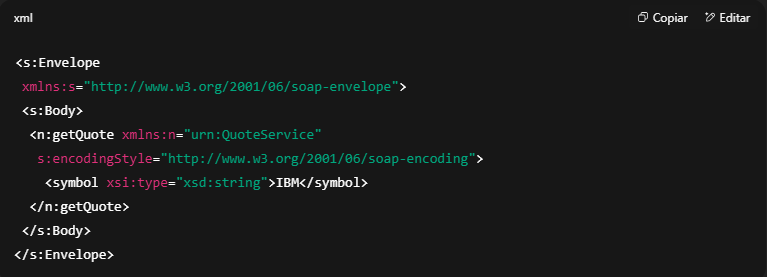
**2.2.4 Estilos de codificação**

Como parte da especificação geral, a Seção 5 do padrão SOAP introduz um conceito conhecido como *estilos de codificação*. Um estilo de codificação é um conjunto de regras que define exatamente como tipos de dados nativos de aplicação e plataforma devem ser codificados em uma sintaxe XML comum. Estes são, obviamente, para uso com SOAP no estilo RPC.

O estilo de codificação para um determinado conjunto de elementos XML é definido por meio do uso do atributo encodingStyle, que pode ser colocado em qualquer parte do documento e se aplica a todos os filhos subordinados do elemento no qual ele está localizado.

Por exemplo, o atributo encodingStyle no elemento getQuote no corpo do Exemplo 2-5 indica que todos os filhos do elemento getQuote seguem as regras do estilo de codificação definidas na Seção 5.

**Exemplo 2-5. O atributo encodingStyle**



Embora a especificação SOAP defina um estilo de codificação na Seção 5, foi declarado explicitamente que nenhum estilo único é o esquema de serialização padrão. Por que isso é importante?

Estilos de codificação são como aplicações em diferentes plataformas compartilham informações, mesmo que possam não ter tipos de dados ou representações em comum. A abordagem que o estilo de codificação da Seção 5 do SOAP adota é apenas um possível mecanismo para fornecer isso, mas não é adequada para todas as situações.

Por exemplo, no caso em que uma mensagem SOAP é usada para trocar uma ordem de compra que já possui uma sintaxe XML definida, não há necessidade de aplicar as regras de codificação da Seção 5. A ordem de compra seria simplesmente inserida na seção Body do envelope SOAP como está.

O estilo de codificação da Seção 5 do SOAP será discutido com muito mais detalhes mais adiante neste capítulo, pois a maioria das aplicações e bibliotecas SOAP o utiliza.

**2.2.5 Controle de versões**

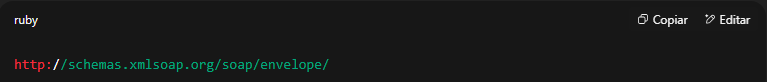
Houve várias versões da especificação SOAP colocadas em produção. O rascunho de trabalho mais recente, SOAP Versão 1.2, representa os primeiros resultados do esforço do World Wide Web Consortium (W3C) para padronizar um protocolo de empacotamento baseado em XML para serviços web. O W3C escolheu o SOAP como base para esse esforço.

A versão anterior do SOAP, Versão 1.1, ainda é amplamente utilizada. Na verdade, no momento em que estamos escrevendo isto, existem apenas três implementações da especificação SOAP 1.2 disponíveis: SOAP::Lite para Perl, Apache SOAP Versão 2.2 e Apache Axis (que nem mesmo está em status beta).

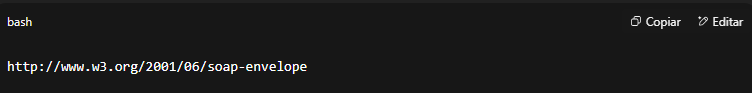
Embora SOAP 1.1 e 1.2 sejam em grande parte iguais, as diferenças existentes são suficientemente significativas para merecerem menção. Para evitar problemas sutis de incompatibilidade, o SOAP 1.2 introduz um modelo de controle de versão que trata de como processadores compatíveis com SOAP Versão 1.1 e SOAP Versão 1.2 podem interagir. As regras são bastante diretas:

1. Se uma aplicação compatível com SOAP Versão 1.1 receber uma mensagem SOAP Versão 1.2, um erro de "incompatibilidade de versão" (*version mismatch*) será acionado.
2. Se uma aplicação compatível com SOAP Versão 1.2 receber uma mensagem SOAP Versão 1.1, a aplicação pode optar por processá-la de acordo com a especificação SOAP Versão 1.1 ou acionar um erro de "incompatibilidade de versão".

A versão de uma mensagem SOAP pode ser determinada verificando o namespace definido para o envelope SOAP. A Versão 1.1 usa o namespace:

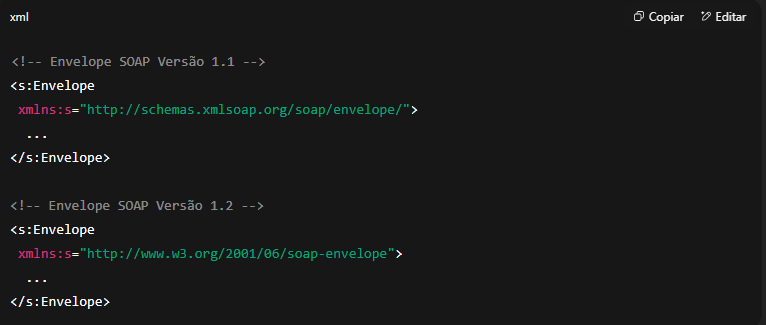


enquanto a Versão 1.2 usa o namespace:



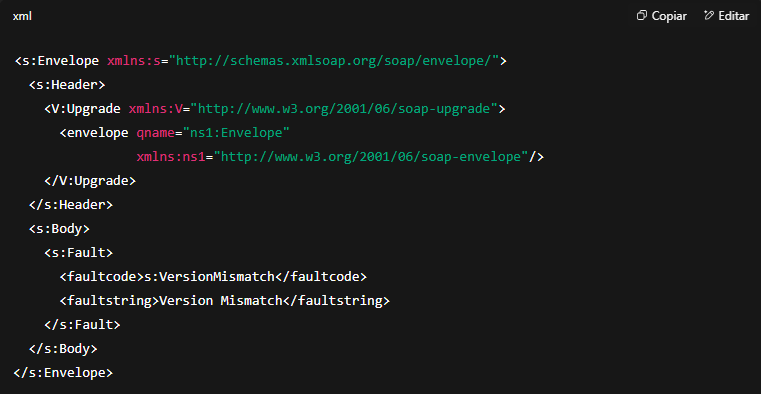
O Exemplo 2-6 ilustra a diferença.

**Exemplo 2-6. Distinguindo entre SOAP 1.1 e SOAP 1.2**



Quando as aplicações relatam um erro de incompatibilidade de versão de volta ao remetente da mensagem, elas podem incluir opcionalmente um bloco de cabeçalho Upgrade que informa ao remetente qual versão do SOAP é suportada. O Exemplo 2-7 mostra o cabeçalho Upgrade em ação.

**Exemplo 2-7. O cabeçalho Upgrade**



Para compatibilidade retroativa, erros de incompatibilidade de versão devem estar em conformidade com a especificação SOAP Versão 1.1, independentemente da versão do SOAP utilizada.

**2.3 Falhas SOAP (*SOAP Faults*)**

Uma falha SOAP (mostrada no Exemplo 2-8) é um tipo especial de mensagem voltada especificamente para comunicar informações sobre erros que possam ter ocorrido durante o processamento de uma mensagem SOAP.

**Exemplo 2-8. Falha SOAP**



As informações comunicadas na falha SOAP são as seguintes:

**O código de falha** (*fault code*)  
Um valor gerado de forma algorítmica para identificar o tipo de erro que ocorreu. O valor deve ser um *XML Qualified Name*, significando que o nome do código só tem significado dentro de um namespace XML definido.

**A string de falha** (*fault string*)  
Uma explicação legível por humanos sobre o erro.

**O ator da falha** (*fault actor*)  
O identificador único do nó de processamento da mensagem no qual o erro ocorreu (atores serão discutidos mais adiante).

**Os detalhes da falha** (*fault details*)  
Usados para expressar detalhes específicos da aplicação sobre o erro ocorrido. Devem estar presentes se o erro ocorrido estiver diretamente relacionado a algum problema com o corpo da mensagem. Não devem ser usados, entretanto, para expressar informações sobre erros que ocorram em relação a qualquer outro aspecto do processamento da mensagem.

**2.3.1 Códigos de falha SOAP padrão**

O SOAP define quatro tipos padrão de falhas que pertencem ao namespace http://www.w3.org/2001/06/soap-envelope. Estes são descritos a seguir:

**VersionMismatch**  
O envelope SOAP está usando um namespace inválido para o elemento Envelope.

**MustUnderstand**  
Um bloco de cabeçalho continha a marca mustUnderstand="true" que não foi entendida pelo destinatário da mensagem.

**Server**  
Ocorreu um erro que não pode ser diretamente vinculado ao processamento da mensagem.

**Client**  
Há um problema na mensagem. Por exemplo, a mensagem contém credenciais de autenticação inválidas ou há uma aplicação incorreta das regras de estilo de codificação da Seção 5.

Esses códigos de falha podem ser estendidos para permitir tipos de falha mais expressivos e granulares, mantendo ainda a compatibilidade retroativa com os códigos principais.

O exemplo de falha SOAP demonstra como essa extensibilidade funciona. O código de falha Client.Authentication é um derivado mais granular do tipo de falha Client. A notação com “.” indica que a parte à esquerda do ponto é mais genérica que a parte à direita.

**2.3.2 Falhas MustUnderstand**

Como mencionado anteriormente, um bloco de cabeçalho contido em uma mensagem SOAP pode indicar, por meio da marca mustUnderstand="true", que o destinatário da mensagem deve entender como processar o conteúdo do bloco de cabeçalho. Se não puder, o destinatário deve retornar uma falha MustUnderstand ao remetente da mensagem. Ao fazer isso, a falha deve comunicar informações específicas sobre os blocos de cabeçalho que não foram compreendidos pelo destinatário.

A estrutura da falha SOAP não permite expressar quaisquer informações sobre quais cabeçalhos não foram entendidos. O elemento details seria o único lugar para colocar essa informação, mas ele é reservado exclusivamente para o propósito de expressar informações de erro relacionadas ao processamento do corpo, e não do cabeçalho.

Para resolver esse problema, a especificação SOAP Versão 1.2 define um bloco de cabeçalho padrão chamado *Misunderstood* que pode ser adicionado à mensagem de falha SOAP para indicar quais blocos de cabeçalho na mensagem recebida não foram compreendidos. O Exemplo 2-9 mostra isso.

**Exemplo 2-9. O cabeçalho Misunderstood**

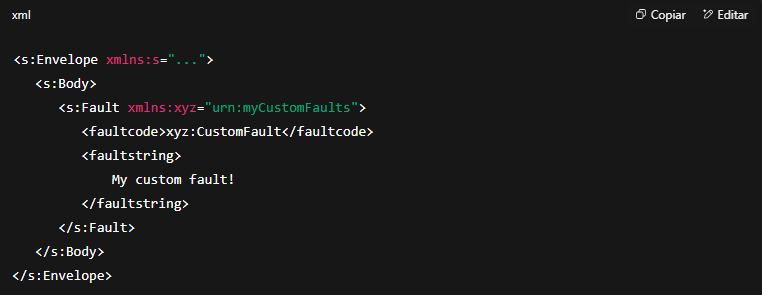


O bloco de cabeçalho Misunderstood é opcional, o que o torna pouco confiável para uso como método primário de determinar quais cabeçalhos causaram a rejeição da mensagem.

**2.3.3 Falhas personalizadas (*Custom Faults*)**

Um serviço web pode definir seus próprios códigos de falha personalizados que não derivem daqueles definidos pelo SOAP. O único requisito é que essas falhas personalizadas sejam qualificadas por namespace. O Exemplo 2-10 mostra um código de falha personalizado.

**Exemplo 2-10. Uma falha personalizada**



Aborde falhas personalizadas com cautela: um processador SOAP que entenda apenas os quatro códigos de falha padrão não poderá agir de forma inteligente ao receber uma falha personalizada.

No entanto, falhas personalizadas ainda podem ser úteis em situações em que os códigos de falha padrão sejam genéricos demais ou inadequados para expressar o erro ocorrido.

Na maioria dos casos, a extensibilidade dos quatro códigos de falha existentes torna os códigos personalizados amplamente desnecessários.

**2.4 O Modelo de Troca de Mensagens SOAP**

O processamento de uma mensagem SOAP envolve desmontar o envelope e fazer algo com as informações que ele carrega. O SOAP define uma estrutura geral para tal processamento, mas deixa os detalhes de como isso é implementado a cargo da aplicação.

O que a especificação SOAP tem a dizer sobre processamento de mensagens trata principalmente de como as aplicações trocam mensagens SOAP. A Seção 2 da especificação descreve um modelo muito específico de troca de mensagens.

**2.4.1 Caminhos de mensagem e atores**

No cerne desse modelo de troca está a ideia de que, embora uma mensagem SOAP seja fundamentalmente uma transmissão unidirecional de um envelope de um remetente para um destinatário, essa mensagem pode passar por vários processadores intermediários que, cada um por sua vez, façam algo com a mensagem. Isso é análogo a um *pipeline* do Unix, onde a saída de um programa se torna a entrada de outro, e assim por diante, até que se obtenha o resultado desejado.

Um intermediário SOAP é um serviço web projetado especificamente para ficar entre um consumidor de serviço e um provedor de serviço e agregar valor ou funcionalidade à transação entre os dois. O conjunto de intermediários pelos quais a mensagem passa é chamado de **caminho da mensagem** (*message path*). Cada intermediário ao longo desse caminho é conhecido como um **ator** (*actor*).

A construção de um caminho de mensagem (a definição de quais nós uma mensagem atravessa) não é coberta pela especificação SOAP. Várias extensões ao SOAP, como o **SOAP Routing Protocol** da Microsoft (*WS-Routing*), surgiram para preencher essa lacuna, mas ainda não existe um método padrão (de fato ou formal) para expressar o caminho da mensagem. O WS-Routing será abordado mais adiante.

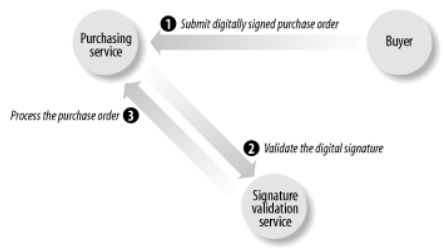
O que o SOAP especifica, entretanto, é um mecanismo para identificar quais partes da mensagem SOAP se destinam ao processamento por atores específicos em seu caminho de mensagem. Esse mecanismo é conhecido como **targeting** (endereçamento) e só pode ser usado em relação a blocos de cabeçalho (o corpo do envelope SOAP não pode ser explicitamente direcionado a um nó específico).

Um bloco de cabeçalho é direcionado a um ator específico em seu caminho de mensagem por meio do uso do atributo especial actor. O valor do atributo actor é o identificador único do intermediário que está sendo direcionado. Esse identificador pode ser a URL onde o intermediário pode ser encontrado ou algo mais genérico. Intermediários que não correspondem ao atributo actor devem ignorar o bloco de cabeçalho.

Por exemplo, imagine que eu seja um atacadista de finos cardigãs. Eu configuro um serviço web que me permite receber ordens de compra dos meus clientes na forma de mensagens SOAP. Você, um dos meus melhores clientes, quer enviar um pedido de 100 cardigãs. Então, você me envia uma mensagem SOAP que contém a ordem de compra.

Para nossa proteção mútua, estabeleci uma relação com um serviço web de terceiros confiável que pode me ajudar a validar se a ordem de compra enviada realmente veio de você. Esse serviço funciona verificando se o bloco de cabeçalho de assinatura digital incorporado na mensagem SOAP é válido.

Quando você envia essa mensagem para mim, ela será roteada através desse serviço intermediário de verificação de assinatura, que, por sua vez, extrairá a assinatura digital, validará e adicionará um novo bloco de cabeçalho que me dirá se a assinatura é válida. A transação está representada na Figura 2-4.



Agora, o intermediário de verificação de assinatura precisa de alguma forma saber qual bloco de cabeçalho contém a assinatura digital que ele deve verificar. Isso é feito direcionando o bloco de assinatura digital para o serviço de verificação, conforme no Exemplo 2-11.

**Exemplo 2-11. O cabeçalho actor**



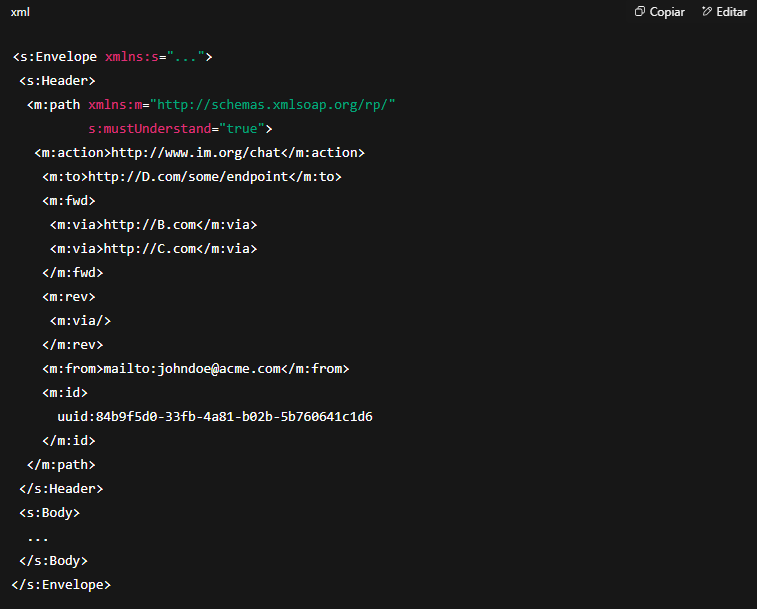
O atributo actor no bloco de cabeçalho signature é como o intermediário verificador de assinatura sabe que é responsável por processar aquele bloco de cabeçalho. Se a mensagem não passar pelo verificador de assinatura, então o bloco signature é ignorado.

**2.4.2 O Protocolo de Roteamento SOAP**

Lembre-se, o SOAP não especifica como a mensagem deve ser roteada até o serviço de verificação de assinatura, apenas que isso deve acontecer em algum momento durante o processamento da mensagem SOAP. Isso torna a implementação de caminhos de mensagens SOAP uma proposta relativamente difícil, já que não há um único padrão para representar esse caminho. O **SOAP Routing Protocol (WS-Routing)** é a proposta da Microsoft para resolver esse problema.

O WS-Routing define um bloco de cabeçalho SOAP padrão (veja o Exemplo 2-12) para expressar informações de roteamento. Sua função é definir a sequência exata de intermediários pelos quais uma mensagem deve passar.

**Exemplo 2-12. Uma mensagem WS-Routing**



Neste exemplo, vemos que a mensagem SOAP deve ser entregue a um destinatário localizado em http://d.com/some/endpoint, mas que deve primeiro passar pelos intermediários http://b.com e http://c.com.

Para garantir que o caminho de mensagem definido pelo bloco de cabeçalho WS-Routing seja seguido corretamente, e porque o WS-Routing é uma extensão de terceiros ao SOAP que nem todo processador SOAP entenderá, a marca mustUnderstand="true" pode ser configurada no bloco de cabeçalho path.

**2.5 Usando SOAP para Web Services no Estilo RPC**

RPC é a aplicação mais comum do SOAP no momento. As seções a seguir mostram como chamadas de método e valores de retorno são codificados em corpos de mensagens SOAP.

**2.5.1 Invocando métodos**

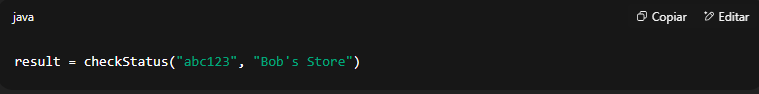
As regras para empacotar uma solicitação RPC em um envelope SOAP são simples:

* A chamada de método é representada como uma única estrutura, com cada parâmetro de entrada (*in*) ou entrada-saída (*in-out*) modelado como um campo nessa estrutura.
* Os nomes e a ordem física dos parâmetros devem corresponder aos nomes e à ordem física dos parâmetros no método que está sendo invocado.

Isso significa que um método Java com a seguinte assinatura:



pode ser invocado com estes argumentos:



usando o seguinte envelope SOAP:



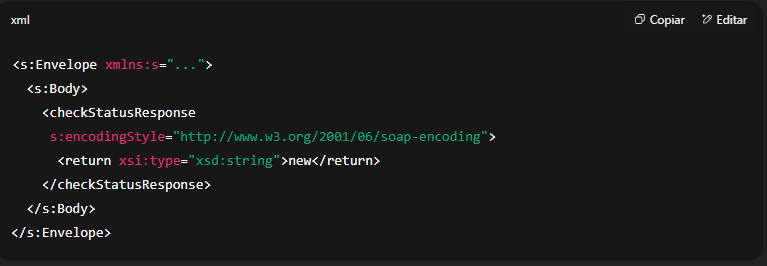
As convenções RPC do SOAP não exigem o uso do estilo de codificação da Seção 5 do SOAP e da tipagem explícita com xsi:type. No entanto, eles são amplamente usados e serão o que descreveremos.

**2.5.2 Retornando respostas**

Respostas de método são semelhantes às chamadas de método no sentido de que a estrutura da resposta é modelada como uma única estrutura com um campo para cada parâmetro *in-out* ou *out* na assinatura do método.

Se o método checkStatus que chamamos anteriormente retornasse a string "new", a resposta SOAP poderia ser algo como o Exemplo 2-13.

**Exemplo 2-13. Resposta à chamada de método**



O nome da estrutura de resposta da mensagem (checkStatusResponse) não é importante, mas a convenção é nomeá-la com base no método, adicionando *Response* no final. Da mesma forma, o nome do elemento return é arbitrário — o primeiro campo na estrutura de resposta da mensagem é assumido como o valor de retorno.

**2.5.3 Relatando erros**

As convenções RPC do SOAP utilizam a falha SOAP (*SOAP fault*) como método padrão para retornar respostas de erro a clientes RPC. Assim como nas mensagens SOAP padrão, a falha SOAP é usada para transmitir a natureza exata do erro que ocorreu e pode ser estendida para fornecer informações adicionais por meio do uso do elemento detail.

Não há muito sentido em personalizar mensagens de erro em falhas SOAP quando se está fazendo RPC, pois a maioria das implementações de SOAP RPC não saberá como lidar com informações de erro personalizadas.

**2.6 Codificação de Dados do SOAP**

A primeira parte da especificação SOAP descreve um formato de envelope padrão para empacotar dados. A segunda parte da especificação (especificamente, a Seção 5) descreve um possível método de serialização dos dados destinados ao empacotamento. Essas regras descrevem em detalhes específicos como tipos de dados básicos de aplicação devem ser mapeados e codificados no formato XML quando incorporados a um Envelope SOAP.

A especificação SOAP introduz o estilo de codificação SOAP como “um sistema de tipos simples que é uma generalização dos recursos comuns encontrados em sistemas de tipos de linguagens de programação, bancos de dados e dados semiestruturados”. Como tal, essas regras de codificação podem ser aplicadas em praticamente qualquer ambiente de programação, independentemente das pequenas diferenças que existam entre esses ambientes.

Os estilos de codificação são completamente opcionais e, em muitas situações, não úteis (lembre-se do exemplo da ordem de compra que demos anteriormente neste capítulo, onde fazia sentido enviar um documento e não uma chamada/resposta de método codificada). Os envelopes SOAP são projetados para transportar quaisquer documentos XML arbitrários, não importando como o corpo da mensagem seja ou se ele segue ou não um conjunto específico de regras de codificação de dados. As regras de codificação da Seção 5 são oferecidas apenas como uma conveniência para permitir que aplicações troquem informações dinamicamente sem conhecimento prévio dos tipos de informações a serem trocadas.

**2.6.1 Entendendo a terminologia**

Antes de prosseguir, é importante compreender bem o vocabulário usado para descrever o processo de codificação. Particularmente importantes são os termos **valor** (*value*) e **acessador** (*accessor*).

Um **valor** representa uma única unidade de dados ou uma combinação de unidades de dados. Isso pode ser o nome de uma pessoa, a pontuação de um jogo de futebol ou a temperatura atual. Um **acessador** representa um elemento que contém ou permite acesso a um valor. No exemplo abaixo, firstname é um acessador, e Joe é um valor:



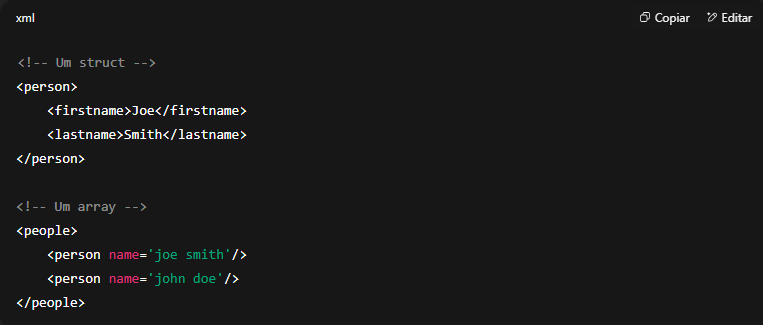
Um **valor composto** representa uma combinação de dois ou mais acessadores agrupados como filhos de um único acessador, e é demonstrado no Exemplo 2-14.

**Exemplo 2-14. Um valor composto**



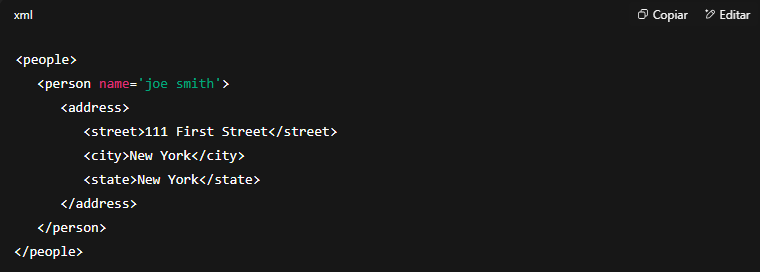
Existem dois tipos de valores compostos: *structs* (as estruturas de que falamos anteriormente) e *arrays* (matrizes). Um *struct* é um valor composto em que cada acessador tem um nome diferente. Um *array* é um valor composto em que os acessadores têm o mesmo nome (valores são identificados por suas posições no array). Um *struct* e um *array* são mostrados no Exemplo 2-15.

**Exemplo 2-15. Structs e arrays**



Por meio do uso dos atributos especiais id e href, o SOAP define que acessadores podem ser de referência única (*single-referenced*) ou de múltipla referência (*multireferenced*). Um acessador de referência única não tem identidade, exceto como filho de seu elemento pai. No Exemplo 2-16, o elemento <address> é um acessador de referência única.

**Exemplo 2-16. Um acessador de referência única**



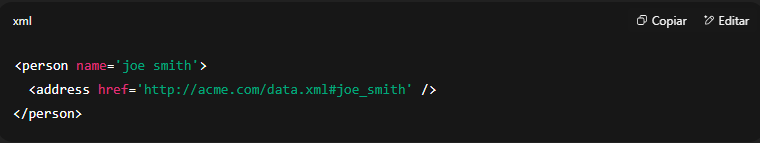
Um acessador de múltipla referência usa id para dar uma identidade ao seu valor. Outros acessadores podem usar o atributo href para referenciar seus valores. No Exemplo 2-17, cada pessoa tem o mesmo endereço, pois fazem referência ao mesmo acessador de endereço de múltipla referência.

**Exemplo 2-17. Um acessador de múltipla referência**



Essa abordagem também pode ser usada para permitir que um acessador faça referência a fontes de informações externas que não fazem parte do Envelope SOAP (dados binários, por exemplo, ou partes de um envelope MIME multipart). O Exemplo 2-18 referencia informações contidas em um documento XML externo.

**Exemplo 2-18. Referência a um documento externo**



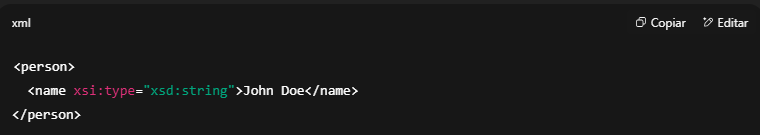
**2.6.2 XML Schemas e xsi:type**

A regra de codificação SOAP na Seção 5.1 descreve como expressar tipos de dados dentro do envelope SOAP, e tem causado bastante confusão e desafios para os implementadores de SOAP. Leia por si mesmo:

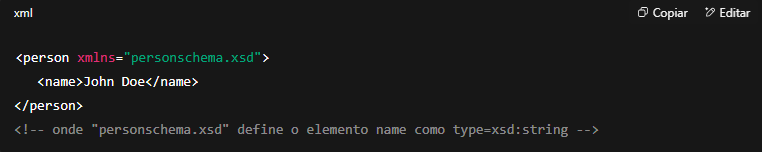
Embora seja possível usar o atributo xsi:type de modo que um grafo de valores seja auto-descritivo tanto em sua estrutura quanto nos tipos de seus valores, as regras de serialização permitem que os tipos de valores possam ser determináveis apenas por referência a um schema. Tais schemas podem estar na notação descrita por *XML Schema Part 1: Structures* e *XML Schema Part 2: Data types* ou podem estar em qualquer outra notação.

**Tradução para o inglês simples:** o SOAP define três maneiras diferentes de expressar o tipo de dado de um acessador:

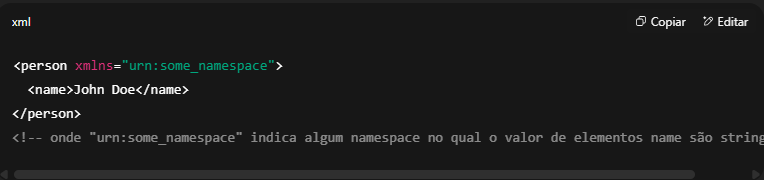
1. Usar o atributo xsi:type em cada acessador, referenciando explicitamente o tipo de dado de acordo com a especificação XML Schema, como neste exemplo:



1. Referenciar um documento XML Schema que define o tipo de dado exato de um elemento específico dentro de sua definição, como neste exemplo:



1. Referenciar algum outro tipo de documento de schema que defina o tipo de dado de um elemento específico dentro de sua definição, como neste exemplo:



As primeiras implementações do SOAP variaram em suas interpretações dessa parte da especificação, causando problemas de integração desagradáveis e irônicos (já que o objetivo principal do SOAP é permitir a interoperabilidade).

Em particular, a implementação SOAP da IBM (posteriormente Apache) optou por exigir a tipagem baseada em xsi:type (ignorando completamente as outras duas opções), enquanto a implementação SOAP da Microsoft optou por ignorar completamente a opção xsi:type em favor de usar schemas baseados em um documento de descrição de serviço externo. Como nenhuma das ferramentas foi implementada como uma implementação completa das regras de codificação SOAP, nenhuma delas conseguia interpretar os tipos de dados codificados pela outra, embora ambas fossem esquemas de codificação SOAP legalmente válidos.

Felizmente, isso foi resolvido.

De fato, existe um grande esforço contínuo para melhorar a interoperabilidade entre implementações SOAP. Para mais informações sobre esse esforço, consulte o grupo “SOAPBuilders” em <http://groups.yahoo.com/>.

**2.7 Tipos de Dados SOAP**

Os tipos de dados suportados pelo estilo de codificação SOAP são os definidos pela especificação *XML Schema data types*. Todos os tipos de dados usados dentro de um bloco XML codificado em SOAP devem ser retirados diretamente da especificação XML Schema ou derivados de tipos nela contidos.

A codificação SOAP fornece duas sintaxes alternativas para expressar instâncias desses tipos de dados dentro do envelope SOAP. O Exemplo 2-19 mostra duas expressões equivalentes de um inteiro com valor "36".

**Exemplo 2-19. Sintaxes alternativas de codificação SOAP para tipagem de valores**



O primeiro método é conhecido como **acessador anônimo** (*anonymous accessor*) e é comumente encontrado em arrays codificados SOAP (como veremos mais adiante neste capítulo). É “anônimo” porque o nome do acessador é seu tipo, e não uma identificação significativa para o valor.

A segunda abordagem é a sintaxe de acessador nomeado que já vimos. Ambas são válidas, já que ambas podem ser diretamente vinculadas aos tipos de dados do XML Schema.

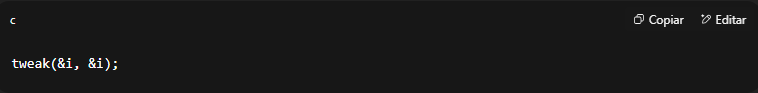
**2.7.1 Múltiplas referências em dados codificados em XML**

Os valores com que um programa trabalha são armazenados na memória. Variáveis são como as linguagens de programação permitem que você manipule esses valores na memória. Duas variáveis diferentes podem ter o mesmo valor; por exemplo, duas variáveis inteiras podem estar ambas configuradas para o valor 42. A codificação XML SOAP para isso usaria XML de referência única (*single-reference XML*), como no Exemplo 2-20.

**Exemplo 2-20. Duas variáveis inteiras configuradas para 42**

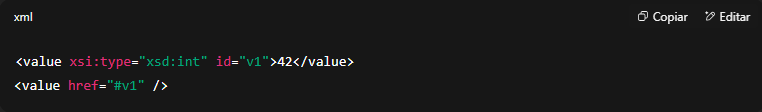


Às vezes, porém, você precisa indicar que duas variáveis separadas estão armazenadas na mesma posição de memória. Por exemplo, se esta chamada de sub-rotina for codificada em XML para SOAP, você precisará identificar o primeiro e o segundo parâmetros como sendo o mesmo:



Você faz isso com as regras de codificação da Seção 5 usando tipos de múltipla referência. Ou seja, você usa o atributo id para nomear o valor em i, e depois usa o atributo href para identificar outras ocorrências desse valor, como no Exemplo 2-21.

**Exemplo 2-21. Múltipla referência para indicar que dois parâmetros são o mesmo**



É importante entender que, mesmo que “SOAP” originalmente significasse “Simple Object Access Protocol”, ele na verdade não tem conceito do que seja um objeto. Para o SOAP, tudo é dado codificado em XML. Portanto, não existe tal coisa como uma “referência de objeto” no SOAP.

Em vez disso, a Codificação da Seção 5 do SOAP especifica um conjunto de regras para transformar um objeto em XML representando esse objeto. Todas as referências a esse objeto que também precisam ser codificadas seriam feitas por meio do uso dos atributos id e href.

Dado o Exemplo 2-22, a serialização codificada SOAP do objeto Person poderia se parecer com o Exemplo 2-23.

**Exemplo 2-22. Código Java para construir um objeto**



Exemplo 2-23. Serialização SOAP do objeto



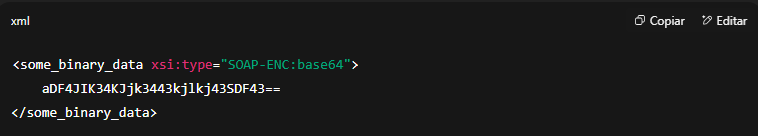
**2.7.2 Structs, Arrays e outros tipos compostos**

Mencionamos anteriormente que a diferença entre um array e um struct no SOAP é que, em um array, cada acessador no grupo é diferenciado apenas por sua posição ordinal no grupo, enquanto no struct cada acessador é diferenciado por nome. Isso foi mostrado no Exemplo 2-15.

Embora muitas linguagens de programação considerem strings como um array de bytes, o SOAP não. Uma string é representada com o tipo de dado string, e não como um array de bytes.

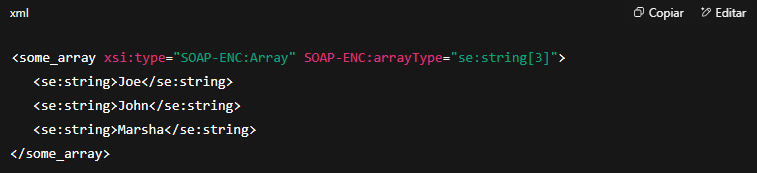
Se você tiver uma coleção de bytes que deseja transmitir e esses bytes não representarem uma string de texto, a Codificação da Seção 5 do SOAP determina que você deve usar uma string base64, conforme definido pela especificação XML Schemas. A serialização correta de um array de bytes arbitrários, então, é mostrada no Exemplo 2-24.

**Exemplo 2-24. Um array de bytes codificado em SOAP**



Arrays regulares, entretanto, são indicados como acessadores do tipo SOAP-ENC:Array ou um tipo derivado desse. O tipo de elementos que um array pode conter é indicado por meio do atributo definido pelo SOAP arrayType, mostrado no Exemplo 2-25.

**Exemplo 2-25. O atributo arrayType**



Observe o [3] anexado ao final do valor do tipo de dado no atributo arrayType. Os colchetes ([]) indicam as dimensões do array, enquanto os números internos representam o número de elementos por dimensão.

Em outras palavras:

* [3] indica uma única dimensão de 3 elementos.
* [3,2] indica um array bidimensional de três elementos cada.

A Codificação SOAP suporta um número ilimitado de dimensões por array, além de permitir arrays de arrays.

Por exemplo, um arrayType de xsd:string[2][] indica um array ilimitado de arrays unidimensionais de string, cada um dos quais contém dois elementos.

No Exemplo 2-26, o acessador data é um array que contém ambos os arrays names.

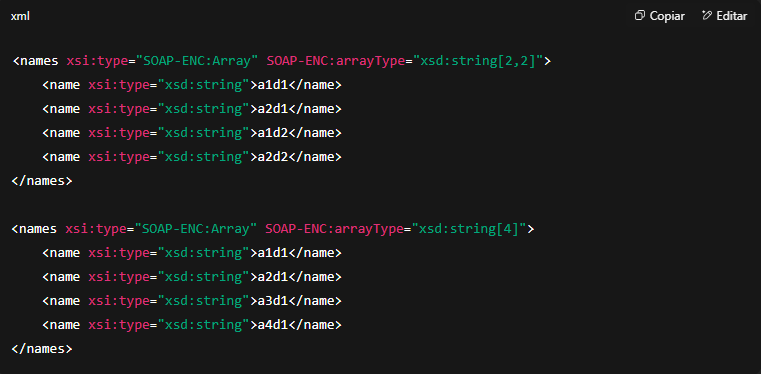
**Exemplo 2-26. Um array bidimensional**



Arrays multidimensionais, expressos como XML, são sintaticamente idênticos a arrays unidimensionais, exceto pelo valor indicado no atributo arrayType.

Por exemplo, um array bidimensional de duas strings é quase idêntico a um array unidimensional de quatro strings (mostrado no Exemplo 2-27).

**Exemplo 2-27. Comparação de array bidimensional e unidimensional**



O valor do atributo arrayType distingue a verdadeira natureza do array serializado.

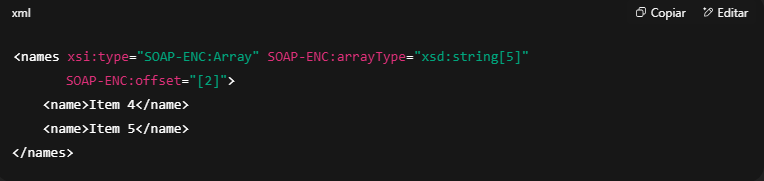
**2.7.3 Arrays parcialmente transmitidos e arrays esparsos**

A Codificação SOAP também inclui suporte para arrays parcialmente transmitidos e arrays esparsos por meio de um conjunto de definições de atributos adicionais.

Um **array parcialmente transmitido** é aquele no qual apenas parte do array é serializada dentro do envelope SOAP. Isso é indicado por meio do atributo SOAP-ENC:offset, que fornece o número de ordinais, contando a partir de zero, até a primeira posição ordinal transmitida.

Em outras palavras, se você tiver um array unidimensional de cinco elementos e quiser transmitir apenas os dois últimos, você usaria a sintaxe do Exemplo 2-28.

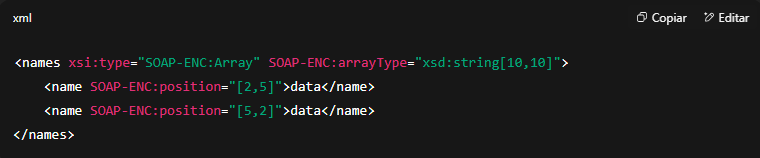
**Exemplo 2-28. Usando SOAP-ENC:offset para arrays parcialmente transmitidos**



**Arrays esparsos** representam uma grade de valores com dimensões especificadas que podem ou não conter dados.

Por exemplo, se você tiver um array bidimensional de dez itens cada, mas apenas os elementos na posição [2,5] e [5,2] contiverem dados, a serialização do Exemplo 2-29 seria apropriada.

**Exemplo 2-29. Serialização SOAP de arrays esparsos**

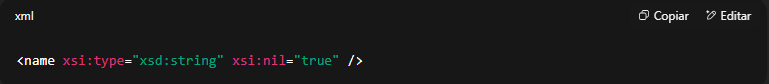


**2.7.4 Acessadores nulos**

No exemplo de array esparso, a ausência de um acessador indica que o valor do acessador é nulo ou algum outro valor padrão.

Um problema com isso é o fato de que o receptor da mensagem não tem como saber se o valor do acessador realmente era nulo ou se o remetente simplesmente não serializou a mensagem corretamente.

Se o receptor espera encontrar o acessador na mensagem, um método melhor de indicar se um acessador contém um valor nulo é usar o atributo xsi:nil="true", definido pelo XML Schema:



Isso permite que você seja muito mais expressivo na sua codificação de dados de aplicação e elimina a confusão sobre o significado de elementos ausentes.

**2.8 Transportes SOAP**

Como mencionado anteriormente, o SOAP se encaixa na pilha de tecnologias de serviços web como um protocolo de empacotamento padronizado, em camadas sobre as camadas de rede e transporte.

Como protocolo de empacotamento, o SOAP não se importa com quais protocolos de transporte são usados para trocar as mensagens. Isso torna o SOAP extremamente flexível em como e onde ele é usado.

Como ilustração dessa flexibilidade, o **SOAP::Lite** — a implementação de serviços web SOAP em Perl escrita por Pavel Kulchenko — suporta a capacidade de trocar mensagens SOAP através de HTTP, FTP, TCP bruto, SMTP, POP3, MQSeries e Jabber. Mostraremos SOAP sobre Jabber no Capítulo 3.

**2.8.1 SOAP sobre HTTP**

Devido à sua onipresença na Internet, o HTTP é, de longe, o transporte mais comum usado para trocar mensagens SOAP.

A especificação SOAP chega a dar um tratamento especial ao HTTP dentro da própria especificação — descrevendo em detalhes como a semântica do modelo de troca de mensagens SOAP se mapeia para o HTTP.

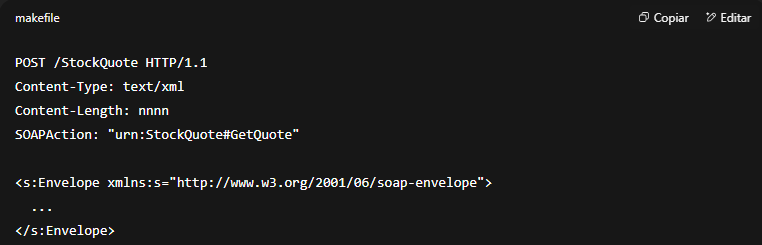
SOAP-sobre-HTTP é uma combinação natural com as convenções RPC do SOAP (requisição-resposta), porque o HTTP é um protocolo baseado em requisição-resposta.

A mensagem de requisição SOAP é enviada ao servidor HTTP junto com a requisição HTTP, e o servidor retorna a mensagem de resposta SOAP na resposta HTTP (veja a Figura 2-5).

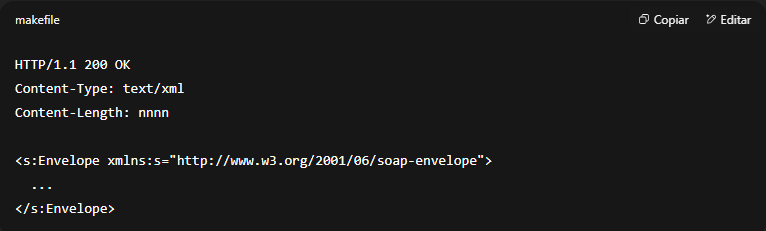


Os Exemplos 2-30 e 2-31 ilustram uma mensagem HTTP de requisição e uma mensagem HTTP de resposta que contêm uma mensagem SOAP.

**Exemplo 2-30. Requisição HTTP contendo uma mensagem SOAP**



Exemplo 2-31. Resposta HTTP contendo uma mensagem SOAP



O cabeçalho HTTP SOAPAction é definido pela especificação SOAP e indica a intenção da requisição HTTP SOAP.

Seu valor é completamente arbitrário, mas tem como objetivo informar ao servidor HTTP o que a mensagem SOAP deseja fazer antes que o servidor HTTP decodifique o XML.

Servidores podem então usar o cabeçalho SOAPAction para filtrar requisições inaceitáveis.

**2.8.2 Questões controversas**

As convenções para envio de SOAP sobre HTTP sempre causaram dificuldades na comunidade de desenvolvimento SOAP. Há uma série de questões que surgem repetidas vezes. Entre elas estão:

**O SOAP deve realmente usar a porta 80 do HTTP ou uma porta específica do SOAP?**  
Como as mensagens SOAP se disfarçam de tráfego web tradicional na porta 80, firewalls geralmente as deixam passar sem bloqueio. Obviamente, administradores de segurança podem ter problemas com isso.

Não há requisitos para que o SOAP sobre HTTP use a porta 80, mas muitas pessoas o utilizam especificamente para evitar serem bloqueadas por firewalls.

**O cabeçalho SOAPAction é realmente útil?**  
Como seu valor é arbitrário, não há como um servidor sempre saber a intenção de uma requisição sem analisar o XML. Essa questão é debatida desde que o cabeçalho SOAPAction foi introduzido no SOAP Versão 1.1.

O grupo de trabalho do W3C que está padronizando o SOAP tende a descontinuar o cabeçalho SOAPAction na próxima versão do protocolo.

**Quando ocorre uma falha de cliente ao processar uma mensagem SOAP, o servidor deve enviar um HTTP 500 "Server Error" ou um HTTP 200 "OK" com uma falha SOAP incluída?**  
Esta é uma questão interessante de semântica. Uma falha de cliente no SOAP é obviamente um erro no nível de aplicação, e não resultado de um erro de servidor.

No entanto, o código HTTP 500 "Server Error" é a resposta padrão exigida para todas as falhas SOAP, independentemente do código de falha.

O consenso geral sobre essa questão é que a consistência é mais importante. Apesar de tipos de falha de cliente não serem *Server Errors*, o código 500 ainda é a resposta correta quando o HTTP é usado como transporte.

**Deve-se usar um esquema de URL específico para SOAP, em vez do tradicional http:// usado para páginas web?**  
Essa questão, como a que trata do uso da porta 80, aborda diretamente se os serviços web SOAP devem se disfarçar como serviços HTTP tradicionais.

Alguns defendem que é necessário um novo esquema soap://. O Protocolo de Roteamento SOAP da Microsoft chega a definir tal esquema.

Embora o HTTP seja o transporte mais popular para mensagens SOAP, ele não está livre de problemas. O HTTP não foi projetado como um transporte para mensagens XML, e há momentos em que os dois protocolos não se encaixam perfeitamente.

Ainda assim, ele continua sendo o transporte mais popular para SOAP, embora o .NET da Microsoft faça uso intenso de SOAP sobre Mensagens Instantâneas, o que pode desafiar a supremacia do HTTP.