**Programando Serviços Web com SOAP**

**Capítulo 3. Escrevendo Serviços Web SOAP**

No Capítulo 2, examinamos os bastidores do SOAP e o XML subjacente. Neste capítulo, demonstramos como criar, implantar e usar serviços web SOAP usando kits de ferramentas para Java, Perl e a nova plataforma .NET da Microsoft. Cobrimos a instalação, configuração e uso do SOAP::Lite para Perl, Apache SOAP para Java e Microsoft .NET para C#.

A tarefa de criar e implantar serviços web não é tão difícil, nem muito diferente do que os desenvolvedores já fazem em aplicativos web mais tradicionais. A tendência em todas as plataformas é automatizar cada vez mais os detalhes complexos e o trabalho tedioso de criação de serviços web. A maioria dos programadores não precisa conhecer os detalhes exatos de codificações e envelopes; em vez disso, eles simplesmente usarão um kit de ferramentas SOAP como os descritos aqui.

**3.1 Anatomia de um Serviço Web 101**

No Capítulo 1, mencionamos brevemente que um serviço web consiste em três componentes:

1. Um **listener** (ou ouvinte) para receber a mensagem.
2. Um **proxy** para pegar essa mensagem e traduzi-la em uma ação a ser executada (como invocar um método em um objeto Java).
3. O **código da aplicação** para implementar essa ação.

Se implementados corretamente, os componentes listener e proxy devem ser completamente transparentes para o código da aplicação. A situação ideal, na maioria dos casos, é que o código nem saiba que está sendo invocado através de uma interface de serviço web — mas isso nem sempre é possível ou desejável.

Um bom exemplo de implementação simples e transparente é o **SOAP::Lite para Perl**, escrito por Pavel Kulchenko. Esse pacote permite que qualquer módulo Perl instalado seja automaticamente implantado como um serviço web sem qualquer trabalho por parte do desenvolvedor do módulo. O proxy pode carregar e invocar automaticamente qualquer sub-rotina em qualquer módulo.

**3.1.1 Implementações e Kits de Ferramentas SOAP**

Existe uma lista surpreendentemente longa de implementações SOAP disponíveis para desenvolvedores. Neste livro, escolhemos focar em três das ferramentas mais populares: **Apache SOAP para Java**, **SOAP::Lite para Perl** e **Microsoft .NET**. Não importa qual kit de ferramentas você use — o processo fundamental de criar, implantar e usar serviços web SOAP é o mesmo.

Uma lista abrangente e atualizada de todas as implementações e kits de ferramentas SOAP pode ser encontrada em:

* <http://www.soaplite.com/>
* <http://www.soapware.org/>

Há kits SOAP para todas as linguagens e ambientes populares (Java, C#, C++, C, Perl, PHP e Python, para citar alguns).

**3.1.2 Manipulando Mensagens SOAP**

A integração de kits de ferramentas SOAP varia com a camada de transporte.

* Alguns implementam seus próprios servidores HTTP.
* Alguns esperam ser instalados como parte de um servidor web específico, de modo que, em vez de servir uma página web, o daemon HTTP entrega a mensagem SOAP ao componente proxy do kit de ferramentas, que então invoca o código por trás do serviço web.

(Figura 3-1: O daemon HTTP passa a requisição para o proxy SOAP, que então invoca o código por trás do serviço web).



Outros kits SOAP oferecem um mecanismo de transporte **plugável**, permitindo selecionar protocolos diferentes apenas definindo um valor de propriedade. O **SOAP::Lite** é um bom exemplo disso, com suporte para transportes FTP, HTTP, IO, Jabber, SMTP, POP3, TCP e MQSeries.

Independentemente do transporte ser interno ou plugável, todos os kits fornecem o componente proxy, que:

1. **Desserializa** a mensagem XML para um formato nativo.
2. **Invoca** o código da aplicação.
3. **Serializa** a resposta (se houver) de volta para XML e a entrega ao listener para envio ao cliente.

Apesar das diferenças de implementação, todas seguem este padrão básico.

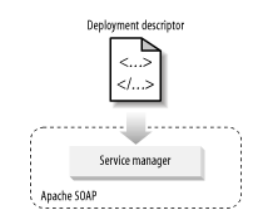
**3.1.3 Implantando Serviços Web**

Implantar um serviço web significa informar ao componente proxy qual código invocar quando receber um determinado tipo de mensagem. Por exemplo, que a mensagem getQuote será tratada pela classe Java samples.QuoteServer ou pelo módulo Perl QuoteServer.pm.

Os mecanismos variam:

* O **SOAP::Lite** requer que o módulo Perl esteja no @INC (caminho de busca de módulos do Perl).
* O **Apache SOAP** requer um **arquivo descritor de implantação** descrevendo a classe Java e o mapeamento entre objetos Java e XML.

(Figura 3-2: Diferente do SOAP::Lite, onde o programa servidor contém a descrição, o Apache SOAP usa um arquivo separado de descritor de implantação).



**3.2 Criando Serviços Web em Perl com SOAP::Lite**

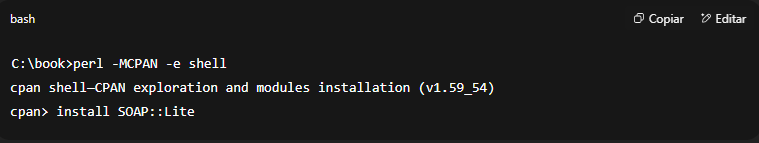
O Perl, como a maioria das linguagens, esconde do programador as complexidades do SOAP com um kit de ferramentas. O **SOAP::Lite** é uma das implementações mais completas do SOAP disponíveis, oferecendo suporte às versões 1.1 e 1.2 do protocolo. Ele tem forte suporte a transportes alternativos (FTP, HTTP, IO, Jabber, SMTP, POP3, TCP e MQSeries), que usaremos mais tarde para demonstrar SOAP sobre Jabber.

**3.2.1 Instalando SOAP::Lite**

O **SOAP::Lite**, como muitos módulos Perl, está disponível no **CPAN** (Comprehensive Perl Archive Network). O CPAN é uma rede de sites web e FTP com conteúdo idêntico — o código-fonte de milhares de módulos Perl. Você pode acessá-lo por linha de comando do Perl ou pelo site <http://www.cpan.org/>.

Para informações sobre instalação de módulos Perl:  
http://www.cpan.org/misc/cpan-faq.html#How\_install\_Perl\_modules

**Exemplo 3-1** mostra a instalação do SOAP::Lite usando o shell interativo do CPAN:



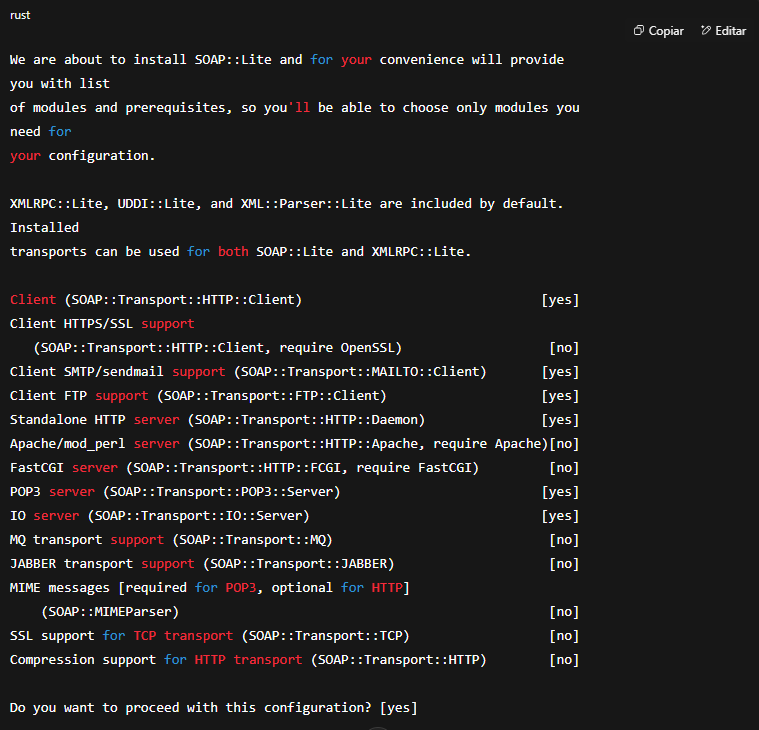
(Se for a primeira vez usando o CPAN, você pode ser guiado por uma configuração inicial.)

O shell CPAN irá:

* Conectar-se a um mirror do CPAN.
* Baixar o código-fonte do SOAP::Lite.
* Tentar compilar o módulo.

Durante a instalação, o **SOAP::Lite** apresenta uma configuração interativa (Exemplo 3-2), na qual é possível usar a configuração padrão ou personalizar.

**Exemplo 3-2 – Configuração interativa do SOAP::Lite**



Na maioria dos casos, a configuração padrão é suficiente.  
No entanto, vamos fazer uma pequena alteração para demonstrar o uso do Jabber como protocolo de transporte para SOAP:

* Responda "no" à pergunta **"Do you want to proceed with this configuration?"**
* Pressione **Enter** para aceitar os valores padrão até chegar à opção **JABBER transport support**, onde você deve responder "yes".

O CPAN então instalará todos os pré-requisitos para o uso do Jabber.

**3.2.2 O Servidor Hello**

Nenhum livro que introduz um novo sistema de programação pode deixar de fora o clássico **Hello World**.

Crie o módulo Perl **Hello.pm** (Exemplo 3-3):

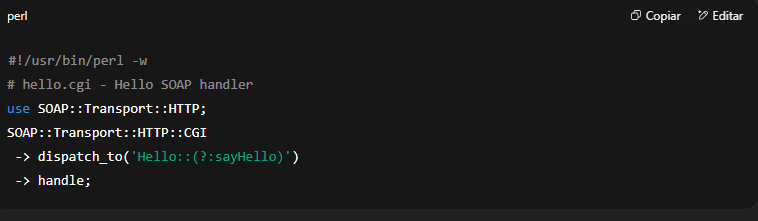
**Exemplo 3-3 – Hello.pm**



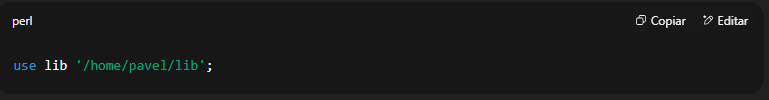
Esse módulo é o código que ficará por trás da interface do serviço web.

Se você já tiver um servidor web com suporte a CGI, pode criar o script **hello.cgi** (Exemplo 3-4):

**Exemplo 3-4 – hello.cgi**



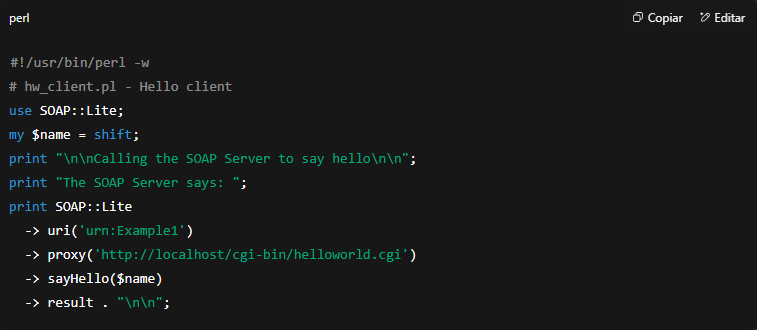
Esse script liga o listener (daemon HTTP) ao proxy (SOAP::Lite).  
Caso o Perl não encontre o módulo Hello nos diretórios padrão (@INC), use:



Com isso, o serviço web SOAP estará pronto para uso.

**3.2.3 O Cliente Hello**

Para testar o serviço, crie o script **hw\_client.pl** (Exemplo 3-5):



Executando:



**3.2.4 Um Cliente Visual Basic**

Para provar que é realmente SOAP que estamos transmitindo aqui, o script em Visual Basic do **Exemplo 3-6** usa a capacidade do Microsoft XML Parser de enviar XML diretamente sobre HTTP para trocar mensagens SOAP com o serviço Hello World.

**Exemplo 3-6 – hw\_client.vbs**



Executar o script Visual Basic deve demonstrar duas coisas:

1. Invocar serviços web SOAP é fácil.
2. Não importa a linguagem utilizada — as strings de Perl e Visual Basic estão sendo trocadas via HTTP.

No próximo exemplo, há duas mensagens trocadas entre o solicitante e o provedor de serviço:

* **Requisição**, codificando o serviço a ser chamado (sayHello) e o parâmetro (James), mostrada no **Exemplo 3-7**.
* **Resposta**, contendo Hello James, mostrada no **Exemplo 3-8**.

**Exemplo 3-7 – Requisição Hello**



**Exemplo 3-8 – Resposta Hello**



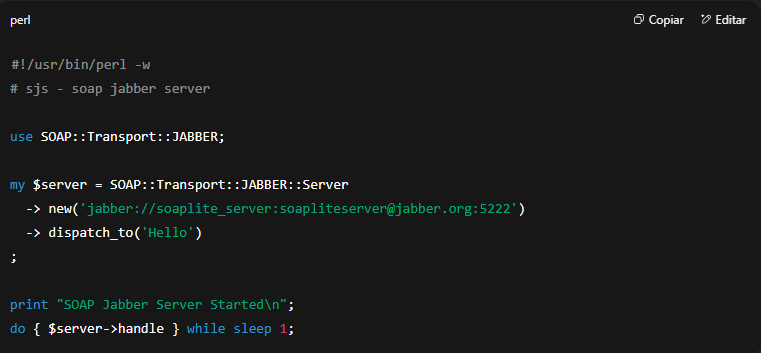
**3.2.5 Alterando Transportes**

O **SOAP::Lite** suporta muitos protocolos de transporte. Vamos modificar o exemplo Hello World para que ele possa ser invocado usando **Jabber**, demonstrando a natureza modular da pilha de serviços web, onde o empacotamento pode ser independente do transporte.

Você pode implantar um serviço web sobre Jabber para aproveitar recursos como presença e identidade fornecidos pelo Jabber.

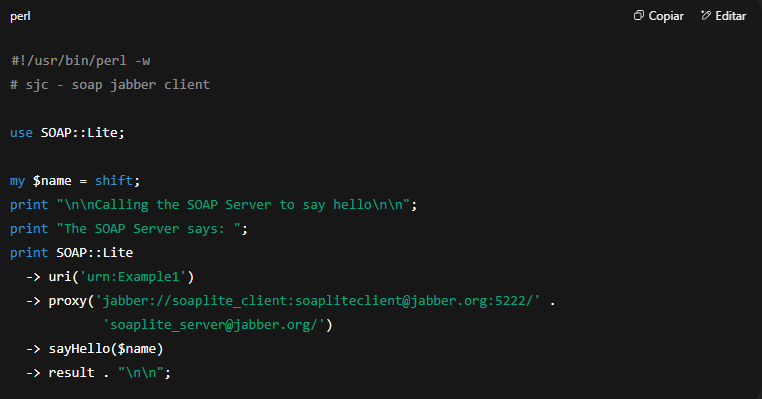
Crie uma instância do servidor Jabber compatível com SOAP incorporado no **SOAP::Lite** usando o script do **Exemplo 3-9**:

**Exemplo 3-9 – sjs, o servidor SOAP Jabber**



Depois, modifique o script cliente para apontar para o endereço Jabber do serviço, como no **Exemplo 3-10**:

**Exemplo 3-10 – sjc, o cliente SOAP Jabber**



As contas soaplite\_server e soaplite\_client estão registradas no **Jabber.org**, portanto o exemplo deve funcionar como está escrito. Para evitar confusão quando vários leitores tentarem ao mesmo tempo, recomenda-se registrar suas próprias IDs Jabber em <http://www.jabber.org/>.

Caso você esteja curioso sobre como o Jabber carrega mensagens SOAP, o **Exemplo 3-11** mostra o texto da mensagem sayHello enviada pelo script anterior. Observe que a mensagem SOAP está embutida no elemento de mensagem Jabber, demonstrando a flexibilidade de ambos os protocolos.

**Exemplo 3-11 – Mensagem Jabber com payload SOAP**



**3.3 Criando Serviços Web em Java com Apache SOAP**

Criar serviços web em Java dá um pouco mais de trabalho do que em Perl com **SOAP::Lite**, mas o processo é essencialmente o mesmo. Para ilustrar como é feito, vamos criar o mesmo serviço web Hello World e implantá-lo usando as ferramentas Apache SOAP.

O **Apache SOAP** é a implementação do protocolo SOAP pela **Apache Software Foundation**. Ele foi projetado para ser executado como um servlet dentro de qualquer Servidor HTTP Java. Como tal, ele implementa apenas a parte de proxy do processo de manipulação de mensagens. Assim como o SOAP::Lite, a lista de recursos do Apache SOAP é impressionante, compartilhando muitos dos mesmos benefícios de seu equivalente em Perl.

**3.3.1 Instalando Apache SOAP**

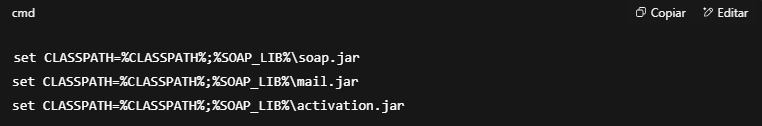
O **Apache SOAP** pode ser usado tanto como cliente quanto como provedor de serviços web SOAP. Uma instalação no lado servidor envolve colocar alguns arquivos .jar no seu **classpath**. É necessário ter um servidor web separado que suporte **Servlets** e **Java Server Pages**, como o **Apache Tomcat** (<http://jakarta.apache.org/tomcat/>).

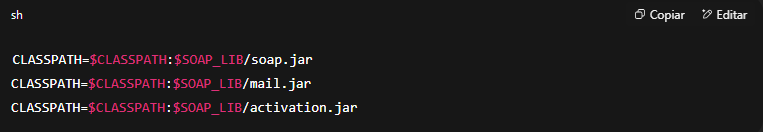
Na página inicial do Apache SOAP (<http://xml.apache.org/soap/index.html>) há links para distribuições apenas de código-fonte ou pré-compiladas. Instalar a versão binária pré-compilada é simples: basta baixar o arquivo ZIP e extrair para um diretório.

No cliente, três arquivos .jar da distribuição (**soap.jar**, **mail.jar** e **activation.jar**) devem estar presentes no seu classpath. Também deve estar presente um analisador XML compatível com **Java API for XML Parsing (JAXP)**, como o **Xerces versão 1.4** (<http://xml.apache.org/xerces-j/>).

Supondo que você instalou os arquivos .jar do Apache SOAP no diretório C:\book\soap, defina a variável de ambiente SOAP\_LIB para C:\book\soap\lib.  
Adicionar os .jar ao classpath envolve:

No Windows:

No Unix (Bourne shell /bin/sh):



Os passos exatos para instalação no servidor dependem do servidor de aplicações web usado, mas o processo é basicamente o mesmo:

* Garantir que os três arquivos .jar estejam no classpath do servidor.

Se o servidor suportar arquivos WAR (**Web Application Archives**), basta usar o arquivo soap.war que acompanha o Apache SOAP (o Tomcat suporta isso). A documentação do Apache SOAP inclui instruções detalhadas para Tomcat e outros ambientes.

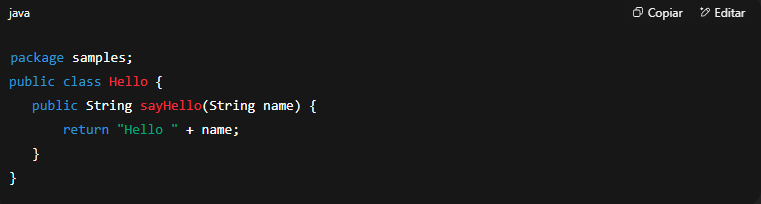
Se você pretende usar o **Bean Scripting Framework (BSF)** para criar serviços baseados em script, também deve garantir que os arquivos bsf.jar e js.jar (uma implementação JavaScript do BSF) estejam no classpath do servidor.

A maioria dos problemas enfrentados por novos usuários do Apache SOAP está relacionada a **classpath incorreto**. Se tiver problemas, comece verificando o classpath.

**3.3.2 O Servidor Hello**

Vamos repetir o que fizemos em Perl: criar o código, implantar o serviço e usar o serviço.

**Exemplo 3-12 – Hello.java**



Compile a classe Java e coloque-a em algum local no classpath do seu servidor web.

**3.3.3 Descritor de Implantação**

Agora precisamos criar um **descritor de implantação** para informar ao Apache SOAP tudo que ele precisa saber para despachar mensagens sayHello para a classe samples.Hello.

**Exemplo 3-13 – Descritor de implantação para samples.Hello**



O descritor contém informações básicas:

* Nome da classe Java (<dd:java class="samples.Hello" static="false" />)
* Escopo da sessão (Application ou Session, conforme especificação Java Servlet)
* faultListener para tratamento de falhas
* Lista de mapeamentos Java-XML

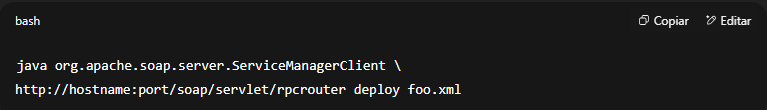
O Apache SOAP suporta provedores plugáveis, permitindo implementar serviços não só como classes Java, mas também como **EJBs**, **COM Classes** e scripts via BSF.

Arquivos de descritor devem ser criados para cada serviço web que se deseja implantar. Existem ferramentas que automatizam isso, mas ainda é preciso selecionar classe, métodos e mapeamentos de tipos (ligação explícita entre um tipo XML e uma classe Java, usada para serializar/desserializar).

Uma vez criado o arquivo, você pode implantá-lo de duas formas:

1. Usando o **Service Manager Client**
2. Modificando diretamente o registro de serviços, se estiver usando o **XML Configuration Manager**

**Método 1 – Service Manager Client**



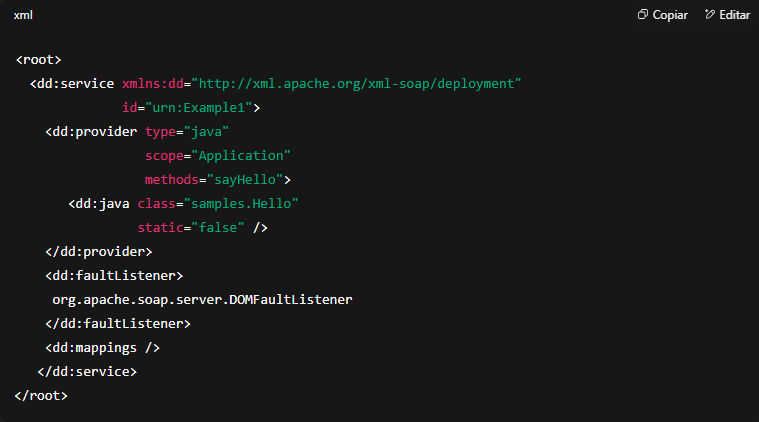
Onde hostname:port é o endereço e porta onde seu serviço web está ouvindo.

Observação: O Apache Service Manager é ele próprio um serviço web, e a implantação ocorre enviando uma mensagem SOAP ao servidor. Para desabilitar esse comportamento (por questões de segurança), defina SOAPInterfaceEnabled como false no arquivo soap.xml.

**Método 2 – XML Configuration Manager**

Neste caso, o arquivo XML de configuração contém todos os descritores de implantação dos serviços. Para implantar, basta adicionar o descritor ao XML e reiniciar o servlet SOAP.

**Exemplo 3-14 – Arquivo de configuração Apache SOAP**



**3.3.4 O Cliente Hello**

Para invocar o serviço Hello World, use a classe Java do **Exemplo 3-15**.

**Exemplo 3-15 – Cliente Hello em Java**

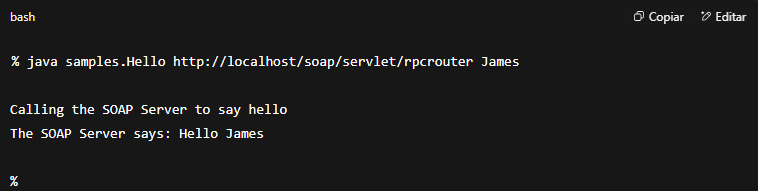


A quantidade de código necessária para realizar essa operação relativamente simples pode parecer surpreendente (nove linhas apenas para inicializar e invocar a chamada do serviço web).

O Java nunca será tão conciso quanto o Perl e outras linguagens de script, mas possui outros pontos fortes. Além disso, vários kits de ferramentas SOAP baseados em Java, como o **GLUE** da *The Mind Electric* e o **Web Services ToolKit** da IBM, oferecem interfaces de proxy dinâmico que reduzem a quantidade de código necessária para invocar serviços web.

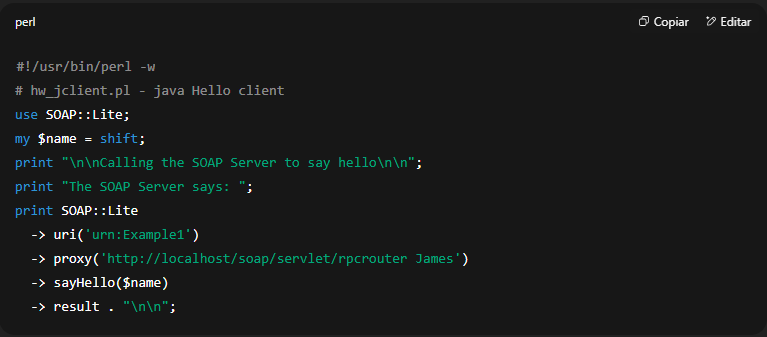
Essas interfaces, porém, geralmente exigem mecanismos adicionais, como **WSDL**, para simplificar a interface de programação. Vamos examinar esses proxies dinâmicos mais adiante no **Capítulo 5**.

Por enquanto, se você compilar e executar essa classe, obterá o mesmo resultado que vimos no exemplo em Perl:

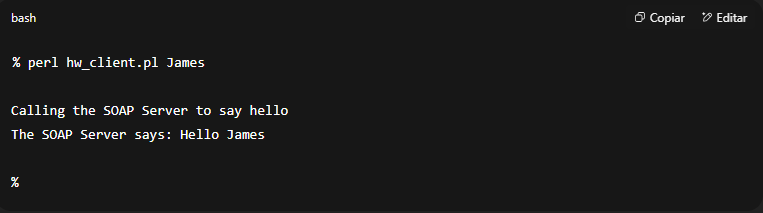


Seu serviço web em Java está finalizado.  
Se você tiver instaladas as versões em Perl e Java, execute novamente o script cliente em Perl, mas apontando-o para a versão Java do serviço Hello World (script modificado mostrado no **Exemplo 3-16**). Você verá que tudo continua funcionando.

**Exemplo 3-16 – hw\_jclient.pl, cliente Perl para o servidor Hello World em Java**



Saída esperada:

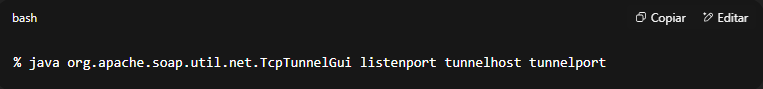


**3.3.5 A Ferramenta TCPTunnelGui**

Uma ferramenta muito útil incluída no Apache SOAP é o **TCPTunnelGui**, um utilitário de depuração que permite visualizar as mensagens SOAP enviadas e recebidas de um serviço web SOAP.

A ferramenta atua como um proxy: ela escuta na máquina local e encaminha o tráfego para o servidor SOAP real. O conteúdo das mensagens que passam pela porta local é exibido na interface gráfica.

Inicie a ferramenta digitando:

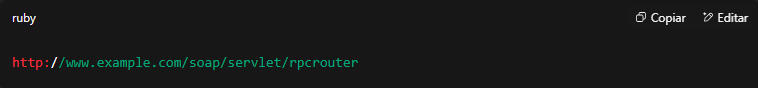


 **listenport** é a porta TCP/IP local que a ferramenta abrirá e escutará.

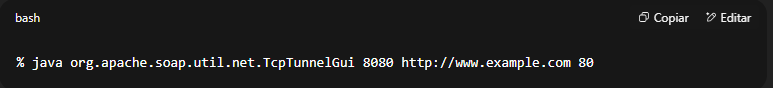
 **tunnelhost** é o endereço do servidor (DNS ou IP) para onde o tráfego será redirecionado.

 **tunnelport** é a porta no servidor de destino.

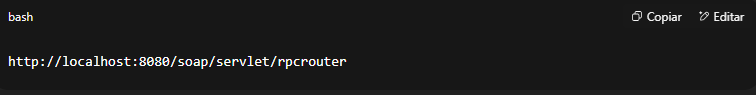
Por exemplo, suponha que seu serviço Hello World esteja implantado em:



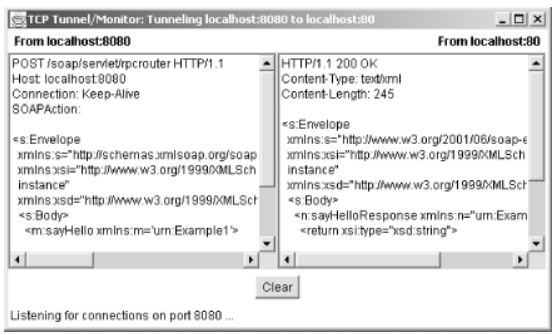
Para visualizar as mensagens enviadas e recebidas por meio da porta TCP/IP local 8080, execute:



Agora direcione as requisições SOAP Hello World para:



(Figura 3-3 – A ferramenta TCPTunnelGui mostrando as mensagens SOAP enviadas e recebidas pelo serviço Hello World)



O **TCPTunnelGui** é extremamente valioso para quem deseja aprender como funcionam os serviços web SOAP (ou depurar por que um serviço não está funcionando).

**3.4 Criando Serviços Web no .NET**

Para desenvolvedores de serviços web que trabalham estritamente na plataforma Windows, a plataforma de desenvolvimento **.NET** da Microsoft oferece suporte integrado para criar e implantar serviços web SOAP de forma simples. Vamos percorrer o processo de criação do serviço Hello World usando **C#**, a nova linguagem de programação semelhante ao Java projetada especificamente para uso com .NET.

**3.4.1 Instalando o .NET**

A primeira coisa que você precisa fazer é baixar e instalar o **Microsoft .NET SDK Beta 2** a partir de <http://msdn.microsoft.com/>.

Essa distribuição gratuita contém tudo o que você precisa para criar e executar qualquer aplicativo .NET, incluindo serviços web .NET.

Há, entretanto, alguns pré-requisitos:

1. É necessário estar executando **Windows 2000**, **Windows NT 4.0**, **Windows 98** ou **Windows Millennium Edition**.
2. Deve ter o **Microsoft Internet Explorer versão 5.01 ou superior**.
3. Deve ter o **Microsoft Data Access Components (versão 2.6 ou superior)** instalado.
4. Deve ter o **Microsoft Internet Information Server (IIS)** instalado e em execução.

Serviços web .NET só podem ser implantados no ambiente IIS.

A instalação do .NET Framework SDK é bastante automática, com um assistente de instalação fácil de usar. Uma vez instalado, podemos criar o serviço Hello World.

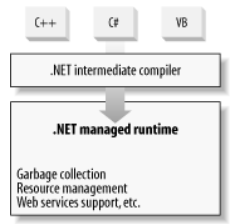
**3.4.2 Apresentando o .NET**

Antes de entrarmos nos detalhes de como criar serviços web no .NET, vamos fazer uma rápida visão geral da arquitetura do .NET.

Primeiramente, o **.NET** é um ambiente de execução semelhante à **Java Virtual Machine**. Pacotes de código, chamados de **assemblies**, podem ser escritos em várias versões específicas do .NET de linguagens populares, como **Visual Basic**, **C++**, **C#**, **Perl**, **Python** etc.

Assemblies são executados em um ambiente de execução gerenciado, hierarquicamente organizado, chamado **Common Language Runtime (CLR)**, que lida com todos os detalhes de gerenciamento de memória e sistema de baixo nível.

(Figura 3-4 – O ambiente gerenciado do .NET)



Atualmente, o .NET funciona basicamente como uma extensão do ambiente **COM** existente no qual as versões atuais do Windows são baseadas. Como tal, o .NET pode ser utilizado em qualquer lugar onde o COM pode ser usado, incluindo dentro do **IIS**.

Serviços web .NET são tipos específicos de assemblies .NET marcados para exportação como serviços web. Esses assemblies estão contidos ou referenciados em um novo tipo de script de servidor chamado **.asmx**.

As extensões .NET para o IIS reconhecem arquivos com final .asmx como serviços web e exportam automaticamente as funções dos assemblies referenciados.

O processo é simples:

1. Escreva o código.
2. Salve o código em um arquivo .asmx.
3. Mova o arquivo .asmx para o servidor IIS.
4. Invoque o serviço web.

**3.4.3 Dizendo Olá**

O .NET introduz a linguagem de programação **C#**. Vamos desenvolver nosso serviço web de exemplo em C#, mas lembre-se de que o .NET facilita igualmente o desenvolvimento em Visual Basic, C++ e outras linguagens.

**Exemplo 3-17 – HelloWorld.asmx, um serviço Hello World em C#**



Note como o código é semelhante à versão em Java que criamos anteriormente.

* As seções entre colchetes angulares <% %> e colchetes [] informam ao ambiente .NET que este código deve ser exportado como serviço web SOAP.
* A linha <%@ WebService Language="C#" Class="Example1" %> indica que estamos exportando um serviço web escrito em C#, implementado pela classe Example1.
* A linha using importa um módulo, neste caso as classes padrão de serviços web.
* [WebService(Namespace="urn:Example1")] define um namespace explícito para o serviço. Caso contrário, o .NET atribuirá um padrão (http://tempuri.org/).
* [WebMethod] marca os métodos da classe que serão expostos como parte do serviço web.

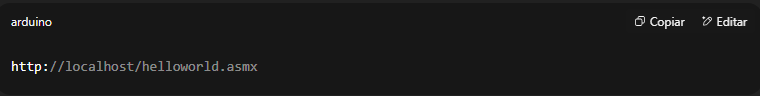
O .NET usa essas informações para:

1. Compilar o código do serviço, caso ainda não tenha feito.
2. Determinar o tipo de requisição:
   * Informações sobre o serviço.
   * Informações sobre um método específico.
   * Invocar a operação (via HTTP-GET, HTTP-POST ou SOAP).

**3.4.4 Implantando o Serviço**

Salve o arquivo HelloWorld.asmx na raiz web do IIS.

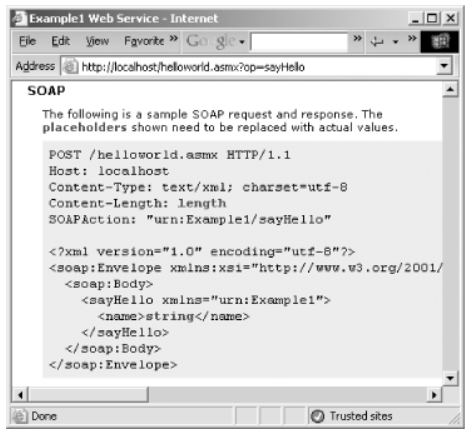
Se o IIS estiver instalado em c:\inetpub\wwwroot, e o arquivo estiver nessa pasta, o endereço será:



Acesse esse endereço em um navegador. Você verá uma página HTML gerada automaticamente com a documentação do serviço (Figura 3-5).

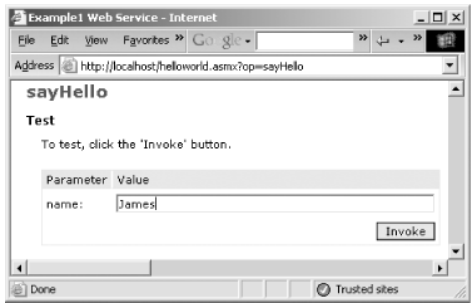


Ao clicar no link **sayHello**, você verá uma descrição detalhada de como invocar essa operação usando SOAP, HTTP-GET e HTTP-POST, bem como um formulário HTML para teste (Figura 3-6).

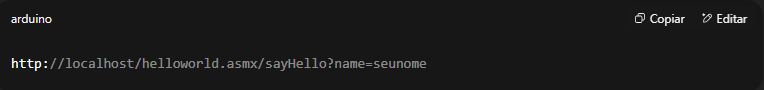


Para testar o serviço:

* Digite seu nome no formulário de teste (Figura 3-7).



* Ou acesse diretamente:



O resultado será uma resposta Hello <nome> (Figura 3-8).

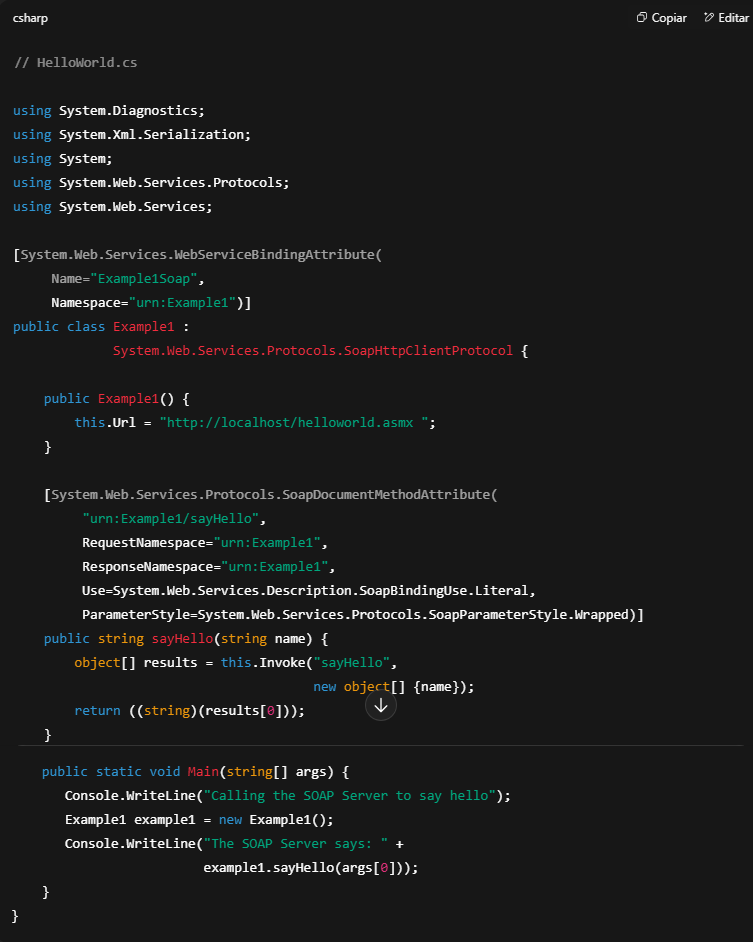


**3.4.5 Invocando o Serviço Usando SOAP**

Criar um cliente SOAP para o serviço Hello World usando .NET é, surpreendentemente, mais difícil do que criar o próprio serviço. Existem ferramentas para facilitar esse processo (que exploraremos brevemente no Capítulo 5), mas, por enquanto, passaremos pelas etapas manualmente para que você saiba o que está acontecendo.

Usando novamente seu editor de texto preferido, crie **HelloWorld.cs** (a extensão .cs indica código-fonte C#) conforme o **Exemplo 3-18**.

**Exemplo 3-18 – HelloWorld.cs, cliente HelloWorld em C#**



A linha [System.Web.Services.WebServiceBindingAttribute] informa ao ambiente gerenciado .NET que este assembly será usado para invocar um serviço web.

Ao compilar o assembly, o .NET fornecerá automaticamente a infraestrutura necessária para que a requisição SOAP funcione.

A herança de System.Web.Services.Protocols.SoapHttpClientProtocol indica ao .NET qual protocolo será usado (SOAP sobre HTTP neste caso).

No construtor, definimos a URL do serviço (this.Url).

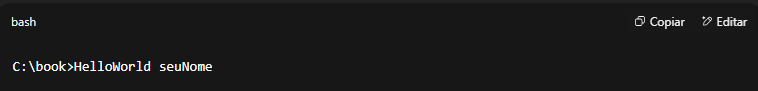
O restante da classe declara um proxy para a operação sayHello, especifica atributos da invocação do serviço, chama o método Invoke e retorna o resultado.

O método Main cria uma instância da classe cliente e chama a operação sayHello, exibindo o resultado no console.

Compile o cliente para gerar o executável HelloWorld.exe:



Para invocar o serviço, digite:



Saída esperada:



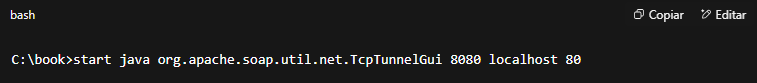
**3.5 Questões de Interoperabilidade**

No momento em que este livro foi escrito, a implementação SOAP do .NET ainda tinha alguns problemas, principalmente na área de interoperabilidade.

Pequenas variações entre a forma como o .NET implementa SOAP e como o SOAP::Lite implementa podem dificultar que ambos funcionem juntos sem ajustes.

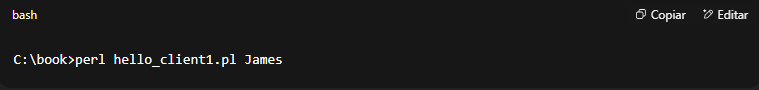
Para ilustrar o problema, siga as etapas abaixo:

1. Inicie a ferramenta **TcpTunnelGui** que acompanha o Apache SOAP, especificando a porta 8080 como porta local e redirecionando para o servidor onde seu HelloWorld.asmx está implantado:



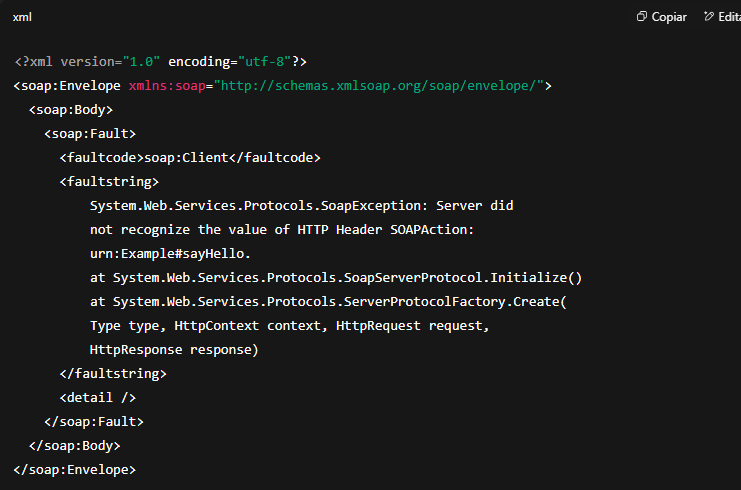
 Modifique o cliente Perl Hello World para apontar para o arquivo .asmx, mas substitua o servidor no URL por localhost:8080.

 Execute o script Perl:

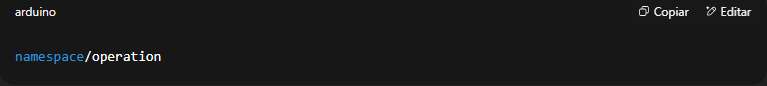


O resultado não será o esperado: o script termina sem exibir Hello James.  
O **TcpTunnelGui** mostrará que a mensagem SOAP foi enviada, mas o .NET rejeitou a requisição e retornou uma *SOAP fault*, como no **Exemplo 3-19**.

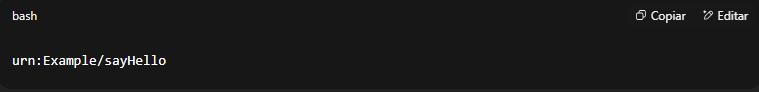
**Exemplo 3-19 – Fault SOAP do .NET**



O .NET exige que o cabeçalho HTTP SOAPAction identifique exatamente a operação, no formato:



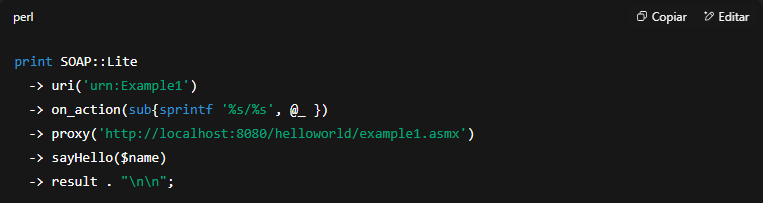
Exemplo:



O SOAP::Lite, por padrão, usa um caractere # para separar namespace e operação, o que não afeta o Apache SOAP (que ignora esse cabeçalho), mas causa falha no .NET.

Para corrigir, é preciso dizer ao SOAP::Lite como formatar o cabeçalho SOAPAction, usando o método on\_action, conforme o **Exemplo 3-20**:

**Exemplo 3-20 – Trecho mostrando a alteração no cliente Perl**



Mesmo com essa alteração, há outro problema:  
O script executa, mas retorna apenas Hello, sem o nome.

Isso ocorre porque o .NET exige que todos os parâmetros de método sejam **nomeados e tipados explicitamente**, enquanto o Perl não faz isso por padrão.

Veja no **TcpTunnelGui** a requisição SOAP enviada pelo SOAP::Lite (Exemplo 3-21):

**Exemplo 3-21 – Requisição SOAP gerada pelo Perl**



O elemento c-gensym3 é um nome gerado automaticamente pelo SOAP::Lite, e todos os parâmetros são tratados como strings (xsd:string).

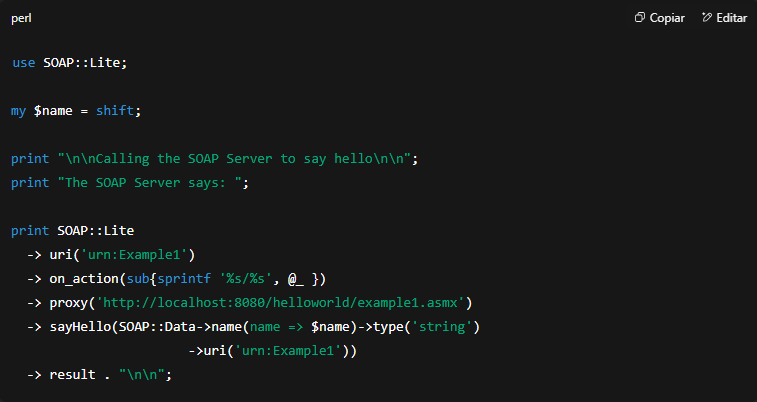
O .NET, ao contrário, espera encontrar um elemento chamado name com tipo xsd:string, como no **Exemplo 3-22**:

**Exemplo 3-22 – Requisição SOAP codificada pelo .NET**



Para que o SOAP::Lite funcione com o .NET, precisamos informar nome, tipo e namespace de cada parâmetro, conforme o **Exemplo 3-23**:

**Exemplo 3-23 – Cliente Perl modificado para funcionar com .NET**



Agora, ao executar o script, tudo funcionará como esperado.

Desenvolvedores que escrevem e utilizam serviços web que possam ser acessados por diferentes implementações SOAP devem estar cientes de que inconsistências como essa existem entre os kits de ferramentas, e precisam estar preparados para lidar com elas.

Com o tempo, as implementações mais populares de SOAP serão refinadas para funcionar juntas de forma transparente, mas, enquanto muitas ainda estão em estado beta (ou até alfa), é preciso atenção.

Felizmente, como veremos no Capítulo 5, existem soluções alternativas para alguns desses problemas.