02

Criando um cliente Java

Aluno, você está começando nesse capitulo? Não tem problema, você pode baixar o projeto web que representar o serviço web aqui (https://s3.amazonaws.com/caelum-online-public/soap/stage/estoquews-web.zip) para importar no Eclipse. Você só precisar baixar o arquivo se você não fez os exercícios do capítulo anterior.

Cliente Java

Conseguimos testar o nosso serviço web com a ajuda do SoapUI. A ferramenta leu o WSDL e criou todo o cliente, incluindo a mensagem SOAP. Agora queremos usar o serviço de estoque em uma aplicação Java.

Como já viemos, com a instalação do JRE já vem também uma implementação do JAX-WS. No caso da JRE da Oracle já vem com a implementação JAX-WS que se chama Metro (https://metro.java.net/).

Como vimos nos capítulos anteriores, nada impediu criar o serviço web usando a JRE apenas, no entanto para gerar o cliente é preciso ter o JDK instalado. Uma vez criadas as classes do cliente basta também o JRE apenas.

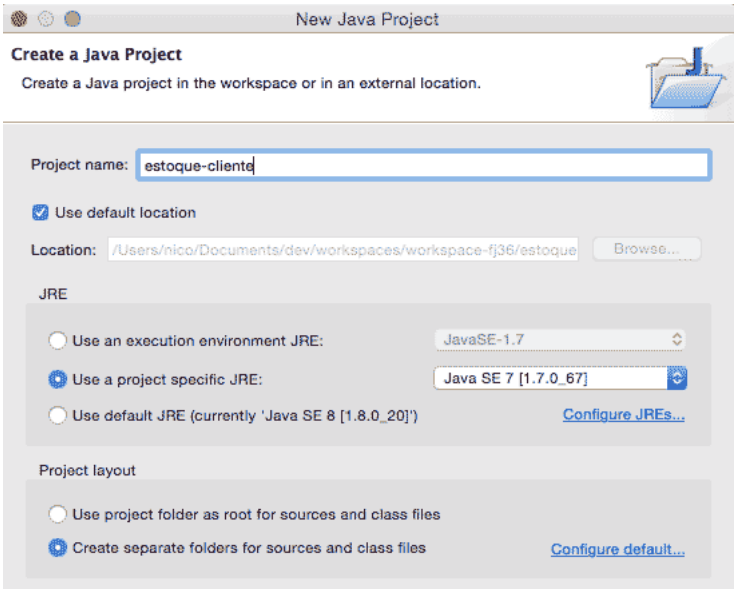
Usando o wsimport

O Metro possui uma ferramenta, o `wsimport` (e justamente essa ferramenta só vem com o JDK), que consegue gerar as classes que acessam o serviço de uma maneira transparente. Objetivo do `wsimport` é o mesmo do `wsdl2java`, gerar as classes para criar o cliente ou servidor.

Para tal, basta passar o endereço do WSDL e automaticamente serão criadas todas as classes necessárias para chamar o serviço remoto. O `wsimport` é um gerador de código para serviços SOAP!

Projeto do cliente

Para não confundir as classes geradas com as classes do servidor vamos criar um novo projeto `estoque-cliente` (projeto padrão java).



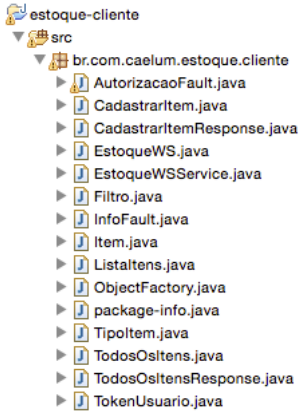
Depois disso abra um terminal e entre na pasta do projeto `estoque-cliente`. Execute na linha de comando:

```
wsimport -s src -p br.com.caelum.estoque.cliente    http://localhost:8080/estoque/EstoqueWS?wsdl
```

Repare nos parâmetros que passamos:

- -s - diretório dos arquivos .java gerados
- -p - pacote das classes geradas

As classes geradas são as classes do mapeamento do JAX-B e algumas outras especificas para chamar o serviço. Entre elas temos uma interface Java `EstoqueWS` e uma classe `EstoqueWSService` que usaremos mais para frente:



Implementando o cliente e o Port

Com essas classes prontas já podemos criar o cliente para nosso serviço. A interface `EstoqueWS` está pronta pra receber algo que se chama de **Port**. O Port é nada mais do que o objeto que se comunica com o serviço! Ele abstrai todos os detalhes como estabelecer a conexão HTTP e gerar a mensagem SOAP. Em alguns casos ele também é chamado de *stub*. De qualquer forma, no mundo de padrões de projeto esse objeto também é chamado de proxy ou remote proxy. 3 nomes para a mesma coisa: Port, Stub ou Remote Proxy!

O código é relativamente simples, basta criar uma instancia da classe `EstoqueWSService` e pedir o *Port*:

```
EstoqueWS cliente = new EstoqueWSService().getEstoqueWSPort();
```

Com o cliente em mãos basta chamar um método do serviço:

```
Filtro filtro = new Filtro();
filtro.setNome("IPhone");
filtro.setTipo(TipoItem.CELULAR);;

ListaItens lista = cliente.todosOsItens(Arrays.<Filtro>asList(filtro));
```

Pronto, isso já basta para criar a mensagem SOAP de ida, enviar um request HTTP e receber a resposta SOAP. Simples não?

Outras ferramentas:

Existem outras ferramentas, como o próprio SoapUI, para gerar as classes do cliente a partir do WSDL, como as que vêm com as outras implementações da especificação do JAX-WS, como por exemplo CXF. Até no Eclipse ou Netbeans existem Wizards para criar as classes pela interface gráfica.

O que você aprendeu nesse capítulo?

- significado do Port
- usar a ferramenta wsimport
- fazer uma chamada SOAP a partir do código Java