02

O primeiro serviço SOAP

Apresentação do ambiente de desenvolvimento

Nessa aula começaremos a desenvolver uma projeto Java para criar um serviço SOAP. Já temos um ambiente de desenvolvimento pré-configurado baseado no Java JDK 8 (http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) (pode ser JDK 7) e utilizaremos como IDE o Eclipse IDE for Java EE Developers (http://www.eclipse.org/downloads/). Mais para frente usaremos o JBoss Wildfly 8.x (http://wildfly.org/downloads/) e o SoapUI (http://www.soapui.org/).

Alem disso já preparamos algumas classes que representam o modelo da aplicação. O link para **baixar o modelo** se encontra aqui (https://s3.amazonaws.com/caelum-online-public/soap/modelo.zip).

Introdução

A troca de informações faz parte de maioria dos sistema, sendo raro o funcionamento de um sistema isoladamente. Há sistemas de terceiros, como Google Maps ou PayPal, mas também há na mesma empresa aplicações separadas que se conectam e trocam informações. Há vários motivos porque sistemas precisam trocar informações, pode ser que precisamos algum dado especifico que a aplicação cuida ou acessar alguma lógica. De qualquer forma, quando um sistema ou processo acessa o outro para trocar informações falamos de **integração**. O problema é que nem sempre é fácil fazer essa integração funcionar.

A maneira de integração mais difundida hoje em dia está no uso de **Web Services**. Existem várias maneiras de se implementar um Web Service, mas apesar de ser um termo genérico, existe algo muito bem especificado pela W3C:

Um dos quesitos primordiais durante a elaboração dessa especificação era que precisaríamos aproveitar toda a plataforma, arquitetura e protocolos já existentes a fim de minimizar o impacto de integrar sistemas. Criar um novo protocolo do zero era fora de cogitação.

Por esses motivos o Web Service do W3C é baseado em HTTP e XML, duas tecnologias onipresentes e que a maioria das linguagens sabe trabalhar muito bem.Vamos estudar um pouco de XML durante as próximas seções para dar o embasamento do SOAP e do WSDL, assuntos que veremos em breve.

SOAP na JRE

Quando Java nasceu uma das principais características da plataforma era funcionar bem na rede, ou seja, na internet. Por isso a Java já vem com os principais classes para se conectar com recursos na rede. Quando os serviços Web surgiram e ganharam popularidade as primeiras bibliotecas eram exclusivamente do servidor de aplicação. Isso significa que na época era preciso usar um servidor de aplicação para publicar o serviço web. Desde a versão 1.6 do Java isso mudou e as classes para rodar um Web Service SOAP foram embutidas na JRE. Dentro da plataforma Java as bibliotecas são organizadas em especificações. A especificação que trata de SOAP se chama JAX-WS (Java API for XML - Web Service) e a sua implementação referencial, o Metro (https://jax-ws.java.net/ (https://jax-ws.java.net/)), está embutida na JRE.

Nas primeiras seções vamos utilizar apenas a JRE para criar serviços web para simplificar o ambiente de execução. Então não preciso usar o servidor mais sofisticado para Web Services? Claro que não pois dentro de uma aplicações há várias outras preocupações, além da integração na web. Mais para frente veremos como usar um serviço SOAP dentro de servidor de aplicação.

Ambiente de execução

No nosso ambiente de desenvolvimento usaremos JRE na versão 1.8 (mas poderia ser 1.7) e Eclipse Luna na versão Java EE. As duas ferramentas já estão instalados então podemos começar criar um novo projeto Java padrão.

A ideia do projeto é simular uma aplicação de estoque. Vamos imaginar que essa aplicação foi criada em Java e fez bastante sucesso, tanto que outras aplicações também gostariam de acessar informações sobre o estoque. Como não todas as aplicações foram escritos em Java é preciso pensar na integração heterogênea, ideal então para usar um Web Service!

O modelo dessa aplicação já está pronto e possui duas classes. Temos uma classe `Item` e um `ItemDao`. O primeiro representa um item no estoque e possui os atributos `codigo`, `nome`, `tipo` e `quantidade`:

```
public class Item {

    private String codigo;
    private String nome;
    private String tipo;
    private int quantidade;

    //construtores  e gets/sets omitidos
```

O `ItemDao` ainda não acessa o banco de dados e cria apenas alguns objetos em memória. Os métodos do DAO são `encontrar` e `cadastra` itens:

```
public class ItemDao {

    private static Map<String, Item> ITENS = new HashMap<>();

    public ItemDao() {
        // populando alguns itens no estoque
        ITENS.put("MEA", new Item("MEA", "MEAN", "Livro", 5));
        ITENS.put("SEO", new Item("SEO", "SEO na Prática", "Livro", 4));
        ITENS.put("IP4", new Item("IP4", "IPhone 4 C", "Celular", 7));
        ITENS.put("GAL", new Item("GAL", "Galaxy Tab", "Tablet", 3));
        ITENS.put("MOX", new Item("MOX", "Moto X", "Celular" , 6));
    }

    //métodos para cadastrar e procurar Item
```

Criação do serviço web

Vamos chamar a classe que realmente representa a implementação do serviço web de `EstoqueWS`. Ela não tem nada especial e possui por enquanto um método apenas que chamaremos de `getItens()`. O método devolve uma lista de itens que buscaremos do DAO:

```
public class EstoqueWS {

    private ItemDao dao = new ItemDao();

    public List<Item> getItens() {
```

```
        System.out.println("Chamando getItens()");
        return dao.todosItens();
    }
}
```

Para realmente indicar que queremos criar o Web Service devemos usar a anotação `@WebService` . Ou seja, a nossa intenção é chamar aquele método usando HTTP e XML:

```
@WebService
public class EstoqueWS {
    //...
}
```

Pronto, é a forma mais simples possível de criar um serviços web!

Publicando o primeiro Endpoint

Ok, mas ainda falta uma coisa. Como não estamos usando um servidor formal é preciso publicar o serviço programaticamente. No mundo de serviços web isso é chamado de publicar o **Endpoint**. O *Endpoint* é o endereço concreto do serviço. A classe `Endpoint` possui o papel de associar a nossa implementação `EstoqueWS` com uma URL:

```
public class PublicaEstoqueWS {

    public static void main(String[] args) {

        EstoqueWS implementacaoWS = new EstoqueWS();
        String URL = "http://localhost:8080/estoquews";

        System.out.println("EstoqueWS rodando: " + URL);

        //associando URL com implementacao
        Endpoint.publish(URL, implementacaoWS);
    }

}
```

O Contrato do Serviço

Como visto na URL, vamos acessar o serviço através do protocolo HTTP. Também usaremos XML, como já falamos. O nosso serviço se chama `estoquews` na URL e define um método com o nome `getItens` que devolve um lista de itens.

Então a gente poderia criar uma página HTML com as informações de quais métodos estão disponíveis no serviço, além de explicar alguns detalhes aos clientes. Mas será que é preciso criar uma página dessas página para cada serviço?

Não, claro que não. Deve haver alguma forma mais fácil de descrever o nosso serviço, alguma forma automática. Podemos ver a mágica usando um parâmetro especial na URL:

```
http://localhost:8080/estoquews?wsdl
```

WSDL significa *Web Service Description Language* e não é nada mais do que um XML que descreve o nosso serviço! Nele temos todas as informações, independente do Java, que um cliente precisa para chamar o Endpoint. Fácil não? Bom, ainda não entendemos tudo nesse arquivo. E mesmo com nosso serviços simples tem bastante informação, mas saiba que esse arquivo não foi feito para nos humanos e sim para ferramentas interpretarem e criarem o cliente.

Testando o serviço com SoapUI

Para testar o nosso serviço vamos criar um cliente. Ou seja, vamos usar uma ferramenta que irá interpretar o WSDL e gerar um cliente que sabe usar o nosso serviço. Existem várias ferramentas para tal, uma dos mais famosas no mercado é o SoapUI. Que é uma aplicação Java (mas não precisaria ser) que possui uma interface fácil de usar, ideal para testes.

Ao iniciar o SoapUI devemos criar um *New SOAP Project*. No Dialogo basta colocar a URL do WSDL no campo *Initial WSDL*:

```
http://localhost:8080/estoquews?wsdl
```

Ao confirmar é gerado os dados para enviar uma requisição SOAP. Isto é, uma requisição HTTP POST que envia XML. Esse XML é a mensagem SOAP!

O primeiro XML SOAP

Repare que uma mensagem SOAP possui um `Envelope`, um `Header` (cabeçalho opcional) e um `Body` que possui um elemento com o mesmo nome do método no serviço: `getItens`.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws.esto
   <soapenv:Header/>
   <soapenv:Body>
      <ws:getItens/>
   </soapenv:Body>
</soapenv:Envelope>
```

Ao submeter uma **requisição SOAP** recebemos uma **resposta SOAP**. Um XML com a mesma estrutura, apenas o corpo da mensagem (`Body`) muda:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getItensResponse xmlns:ns2="http://ws.estoque.caelum.com.br/">
         <return>
            <codigo>GAL</codigo>
            <nome>Galaxy Tab</nome>
            <quantidade>3</quantidade>
            <tipo>Tablet</tipo>
         </return>
         ...
         <return>
            <codigo>IP4</codigo>
            <nome>IPhone 4 C</nome>
            <quantidade>7</quantidade>
            <tipo>Celular</tipo>
         </return>
      </ns2:getItensResponse>
```

```
    </S:Body>
</S:Envelope>
```

Qualquer mensagem SOAP possui um `Envelope` e um `Body`, apenas o `Header` é opcional. A nossa mensagem SOAP não está perfeita e tem como melhorar muito. Mas agora é a hora dos exercícios!

O que você aprendeu nesse capítulo?

- Serviços Web são utilizados para integrar sistemas
- SOAP é XML que trafega em cima do protocolo HTTP
- o JRE já vem com o JAX-WS (Metro) para usar SOAP
- o contrato do serviço é o WSDL que também é um XML
- uma mensagem SOAP possui um `Envelope` e um `Body`,
- na mensagem SOAP o `Header` é opcional