Criando um Web Service SOAP com Java EE 6

www.javaeenacaixa.com/2015/02/criando-um-web-service-soap-com-java-ee.html

Olá a todos! Já foi apresentado neste post anterior como criar um Web Service REST utilizando a tecnologia Java EE 6. Nesta publicação serão abordados os passos necessários para a criação de um novo Web Service, mas desta vez elaborado através da utilização do protocolo SOAP - Simple Object Access Protocol.

Será utilizado também o template de projeto Java EE que foi criado neste post. O objetivo do Web Service que será apresentado será expor o método de negócio existente nesta aplicação de exemplo para clientes diferentes do módulo WEB do projeto.

Estrutura do projeto

Apenas para refrescar a memória do que foi apresentado no post de construção do templante do projeto Java EE, nosso projeto é um projeto multi módulo Maven composto por quatro módulos, mas para elaboração do conteúdo deste post estamos interessados em apenas dois deles:

- **O módulo EJB** (teste-ejb): Este projeto contém os EJBs com os métodos de negócio da aplicação. Estes EJB serão invocados pelo Web Service para acesso a camada de negócio da aplicação. É empacotado em um módulo JAR e compõe o módulo EAR do projeto.
- **O módulo WEB** (teste-war): Este projeto contém as páginas e controllers da camada WEB. É neste projeto que adicionaremos o Web Service para exposição dos métodos de negócio aos clientes externos à aplicação. É empacotado em um módulo WAR e compõe o módulo EAR do projeto.

Definindo o contrato do serviço

O primeiro passo na criação de um Web Service é a definição do contrato do serviço. É através deste contrato que o provedor do serviço divulga aos consumidores quais informação devem ser enviadas na requisição e o que eles devem esperar como retorno após o processamento.

Como no protocolo SOAP as mensagens trocadas entre provedores e consumidores são escritas através da linguagem XML, o contrato de um serviço pode ser elaborado através de uma coleção de arquivos XSD - XML Schema Definition. Estes arquivos são documentos XML que descrevem a estrutura a ser seguida por um documento XML e podem ser utilizados para validação das mensagens.

Para definição do contrato do nosso Web Service, iremos definir dois XSDs, uma para a mensagem de entrada e outro para a mensagem de retorno. Estes arquivos serão alocados na pasta /src/main/resources/schemas do módulo WEB teste-war.

Abaixo segue o XSD da mensagem de entrada, nomeado EfetuarOperacaoDeNegocioRequest.xsd:

```
targetNamespace="http://testewar.javaeenacaixa.com/ws/types"
xmlns:tns="http://testewar.javaeenacaixa.com/ws/types"
elementFormDefault="qualified">
```

Neste schema definimos que nossa mensagem de entrada, chamada EfetuarOperacaoDeNegocioRequest será composta por um único atributo do tipo string, denominado texto.

O XSD da mensagem de retorno, nomeado EfetuarOperacaoDeNegocioResponse.xsd, é apresentado abaixo:

```
targetNamespace="http://testewar.javaeenacaixa.com/ws/types"
xmlns:tns="http://testewar.javaeenacaixa.com/ws/types"
elementFormDefault="qualified">
```

O schema de retorno define que a mensagem de saída, batizada de EfetuarOperacaoDeNegocioResponse também será composta por um único atributo do tipo string, denominado resultado.

Criando as classes Java do contrato do serviço

Com os arquivos XSD que definem as mensagens do serviço criados, podemos gerar as classes Java correspondentes a estas mensagens através do plugin Maven para JAXB.

Para isso, precisamos configurar no módulo WEB o uso deste plugin através da adição do trecho abaixo no arquivo POM deste projeto (teste-war/pom.xml):

```
org.jvnet.jaxb2.maven2
maven-jaxb2-plugin
0.8.2

        generate

    generate-sources
```

```
${basedir}/src/main/resources/schemas

${project.build.directory}/generated-sources/jaxb2

-npa
-Xcollection-setter-injector

true
true

    net.java.dev.vcc.thirdparty
    collection-setter-injector
    0.5.0-1
```

Note pelas linhas destacadas que, através da propriedade de configuração schemaDirectory é definido o local onde estão os arquivos XSD. Já a propriedade generateDirectory define o local onde serão armazenadas as classes Java após serem geradas pelo plugin. Uma boa prática é alocar estas classes geradas automaticamente na pasta de build do projeto, para que este código gerado automaticamente não seja misturado com código mantido pelos desenvolvedores.

Para efetuar a geração das classes basta executar os goals Maven abaixo na pasta do módulo teste-war:

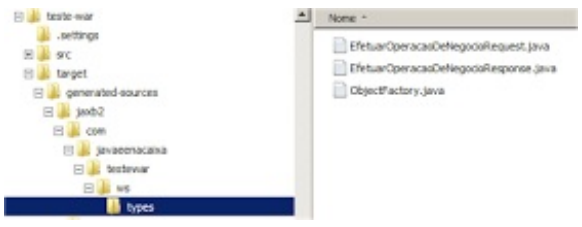
mvn clean generate-sources

Se tudo estiver correto a saída abaixo deve ser visualizada:

Após a execução as classes equivalentes aos schemas definidos devem ser visualizadas na pasta /teste-war/target/generated-sources/jaxb2 conforme demostra a imagem abaixo:

Implementando o Web Service

Com as classes das mensagens criadas podemos agora criar a implementação do Web Service SOAP. Para isso primeiramente foi criado o pacote no qual a classe do Web Service será alocada: com.javaeenacaixa.testewar.ws. Em seguida foi criada a classe o Web Service, cuja implementação segue abaixo:



```
package com.javaeenacaixa.testewar.ws;

import javax.ejb.EJB;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

import com.javaeenacaixa.testeejb.sessionbean.TesteBeanLocal;
import com.javaeenacaixa.testewar.ws.types.EfetuarOperacaoDeNegocioRequest;
import com.javaeenacaixa.testewar.ws.types.EfetuarOperacaoDeNegocioResponse;
import com.javaeenacaixa.testewar.ws.types.ObjectFactory;

@WebService(name="EfetuarOperacaoDeNegocioWS",serviceName="EfetuarOperacaoDeNegocioWSService")
public class EfetuarOperacaoDeNegocioWS {

    //Injecao do EJB da camada de negocio
    @EJB
    private TesteBeanLocal testeBean;

    @WebMethod(action="process")
    public EfetuarOperacaoDeNegocioResponse efetuarOperacaoDeNegocio(@WebParam(name="request")
EfetuarOperacaoDeNegocioRequest request){

        //Instanciacao do objeto de retorno
        ObjectFactory factory = new ObjectFactory();
        EfetuarOperacaoDeNegocioResponse response = factory.createEfetuarOperacaoDeNegocioResponse();

        //Chamada ao metodo de negocio
        response.setResultado(
            testeBean.efetuarOperacaoDeNegocio(request.getTexto())
        );
```

```
        return response;
    }
}
```

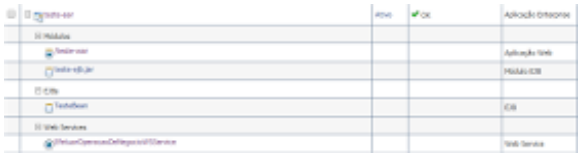
Nas linhas destacadas estão as anotações necessárias para transformar nosso POJO em um Web Service SOAP. O primeiro destaque mostra a definição do nome do serviço. O segundo destaque exibe a definição do método do serviço e os parâmetros utilizados.

Testando o Web Service

Após a elaboração do Web Service estamos prontos para testá-lo. O primeiro passo para isso é efetuarmos o deploy da aplicação em um servidor de aplicações Java EE 6. No meu caso, estou utilizando o Weblogic 12c (12.1.3).

Após o deploy, ao acessarmos a aba de implantações do domínio, podemos visualizar as informações sobre o Web Service implantado, conforme apresentado na imagem abaixo:

Ao clicarmos sobre o nome do Web Service, na aba Teste temos o link para o WSDL do serviço:

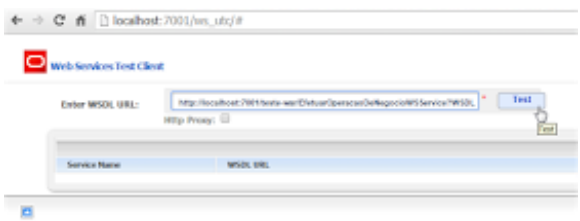


O WSDL - Web Services Description Language - apresenta todos os detalhes sobre o Web Service, como métodos disponíveis, composição das mensagens e endereço da implementação. Através dele podemos enviar mensagens ao Web Service através da construção de um cliente de testes, ou utilizando um ferramenta de testes como o SOAP UI. No meu caso, vou utilizar a ferramenta de testes embutida ao Weblogic, mas o processo seria similar para qualquer outra ferramenta de testes.

Após copiar o endereço do WSDL, no meu caso http://localhost:7001/teste-war/EfetuarOperacaoDeNegocioWSService?WSDL vamos acessar a ferramenta de testes em http://localhost:7001/ws_utc.

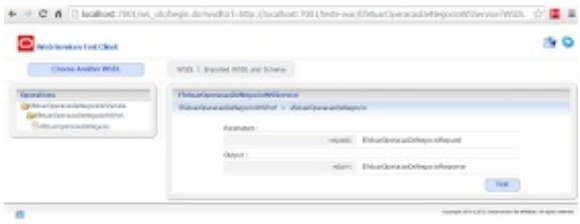
Cole o endereço do wsdl no campo adequado e clique em Test.

A ferramenta efetuará o parse do WSDL e apresentará a lista de operações disponíveis:

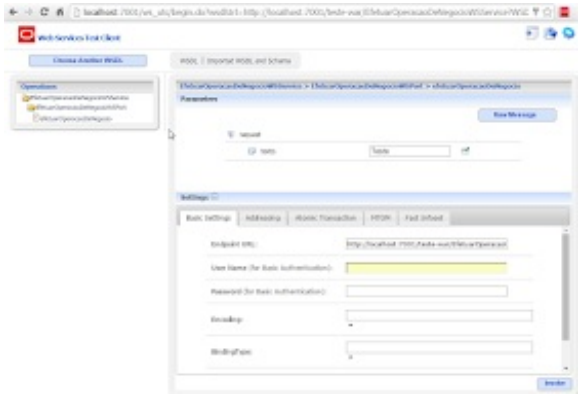


Selecione a opção Test da operação efetuarOperacaoDeNegocio. Será exibido um formulário para preenchimento dos dados da mensagem de entrada do serviço. Preencha a propriedade texto

com o valor desejado e clique no botão Invoke.



Serão apresentadas as mensagens de requisição e a mensagem de resposta retornada pelo serviço:



Pontos de atenção

- O Web Service criado está exposto como um serviço público, portanto, não valida a autorização do cliente na execução do serviço. Em um post futuro será abordado como criar um Web Service com autenticação para os casos onde este controle se faz necessário.
- Caso nossa aplicação fosse composta apenas pelo módulo EJB, seria possível expor o método de negócio como um Web Service diretamente deste módulo através do uso das mesmas anotações. Como temos um módulo dedicado aos componentes WEB, preferi alocar o Web Service neste módulo.

Espero que este tutorial tenha auxiliado na criação de seus projetos e na integração entre os sistemas corporativos do seu ambiente de trabalho. Qualquer dúvida, crítica ou sugestão utilizem os comentários.

Obrigado!