VOLTAR

RMI, SOAP (Web Services) e REST

Apresentar os conceitos relacionados à comunicação de objetos remotos, nos aplicativos distribuídos.

NESTE TÓPICO

O primeiro software que se tem notícias, a precisar de comunicação entre localidades remotas foi o SABRE (sigla para Semi-Automated Business Research Environment). Ele foi desenvolvido nos anos 1950 pela IBM, para atender a uma demanda da empresa de transportes aéreos American Airlines. A empresa queria um sistema que lhe permitisse gerenciar de modo eficaz as reservas de lugares em seus aviões, pois ela não estava conseguindo lidar com o aumento exagerado no volume de passageiros.

A solução foi usar um servidor central numa pequena cidade chamada Briarcliff Manor, no estado de Nova Iorque, que era acessado pelos terminais de reservas on-line espalhados pelos Estados Unidos. Em sua primeira versão, esse sistema era capaz de processar algo em torno de 83 mil chamadas telefônicas por dia. As chamadas telefônicas eram realizadas pelos terminais, através dos quais os usuários digitavam os dados necessários para realizar a reserva, e recebiam a confirmação da reserva.

Esse sistema abriu o caminho para que as grandes corporações, que tinham suas filiais espalhadas pelo mundo, e precisavam enviar/receber dados para e/ou da matriz, pudessem interagir mais rapidamente, usando soluções proprietárias de empresas como IBM, Unisys, NCR e Honeywell. Essas soluções talvez continuassem proprietárias, não fosse a mudança de foco da ARPANET, de uso apenas militar para uso público, que teve seu nome alterado para Internet.

A Internet pedia por padrões abertos para solucionar o problema da comunicação, e uma das primeiras soluções foi o RPC, uma implementação de software criada em 1976 para permitir o compartilhamento remoto de recursos.

O RPC (sigla de Chamada Remota de Procedimento, tradução para Remote Procedure Call) é um procedimento para permitir que um programa de computador possa chamar e executar outro programa sendo executado em outro computador (desde que eles estejam conectados na mesma rede de comunicação), sem que o programador não precise implementar o software que estabeleça e controle a conexão entre os dois computadores, um vez que isso já é feito pelos outros protocolos de transferência de dados usados na Internet. Desse modo, a chamado do outro programa de computador é feita como se estivesse acontecendo dentro do próprio computador.

Em 1976, em termos de usuários, a Internet tinha um alcance bastante restrito, pois era usada somente no meio acadêmico. Sendo assim, o primeiro uso comercial do RPC acabou ocorrendo nas soluções que usavam a arquitetura cliente-servidor. O RPC se tornou mais comum quando os fabricantes dos sistemas operacionais baseados em Unix disponibilizaram pacotes para facilitar o uso do RPC, e os sistemas operacionais mais lançados posteriormente, seguiram esse padrão de facilitar a implementação de software usando o RPC. Seguindo essa tendência, as plataformas de desenvolvimento, como Java EE e .Net, incorporaram componentes que implementam o RPC, o RMI para o Java, e o DCOM para o .Net.

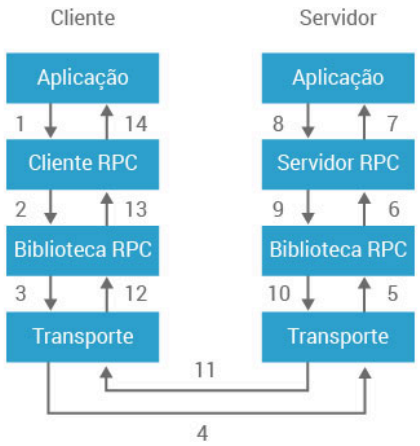

Figura 1 - Princípio da implementação da comunicação entre objetos distribuídos na arquitetura cliente-servidor, através do RPC.

RMI (Remote Method Invocation)

Nesse processo, o aplicativo executado no cliente, invoca um aplicativo sendo executado no servidor, envia uma mensagem com os dados que ele quer que sejam processados, e espera uma resposta do aplicativo sendo executado no servidor. A resposta vem na forma de uma mensagem, contendo o resultado do processamento efetuado pelo servidor. Isso significa que o aplicativo executado no servidor tem que ficar sempre em execução, ou processando os dados, ou esperando por uma nova requisição enviada por um cliente.

Como esse procedimento é realizado em dois locais diferentes, e um depende do outro, é necessário que o desenvolvedor sempre leve em consideração:

- O aplicativo sendo executado no cliente ou no servidor deve prever a ocorrência de erros causados por falhas na rede de comunicação, e tratar esses erros;
- O aplicativo cliente também deve considerar possíveis falhas no aplicativo do servidor, e tratar essas informações de modo coerente;
- As trocas de informações devem ser feitas somente por troca de mensagens, uma vez que as variáveis indicam endereços de memória, e um aplicativo sendo executado no cliente não é capaz de acessar a memória do servidor, e o mesmo ocorre com o aplicativo sendo executado no servidor, em relação ao aplicativo sendo executado no cliente;
- O aplicativo apresentará queda no desempenho, uma vez que essa comunicação entre cliente e servidor demanda um certo tempo para ser processado, e se for muito elevado, certamente comprometerá o uso do aplicativo;
- O aplicativo sendo executado no servidor tem que saber quem está solicitando o processamento, para encaminhar o resultado para esse cliente, ou seja, isso significa que a comunicação deve ocorrer sempre de modo seguro.

Para executar um serviço usando RMI, é necessário iniciar o serviço *rmiregistry*, para que a JVM possa instalar as classes que funcionarão como servidores como serviços gerenciados pelos sistemas operacionais. A forma de inicialização do rmiregistry depende da versão do Java SDK sendo usado, devido às mudanças em relação às políticas de segurança que a linguagem de programação Java vem sofrendo nos últimos anos. Assim, depois de conferir a forma correta de iniciar o rmiregistry, para implementar os serviços RMI, é necessário:

1. criar uma interface de acesso à classe que será a servidora do serviço;
2. implementar a classe que implementa o serviço;

A listagem 1 apresenta um exemplo para implementar a interface de acesso, e a listagem 2 apresenta um exemplo para implementar a classe servidora.

Listagem 1 - Exemplo de implementação de uma interface para um serviço RMI

```
package rmi_test;

import java.math.BigInteger;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ServicoPotencia extends Remote{

    // calcula o quadrado de um número
    public BigInteger quadrado(int numero) throws RemoteException;

    // calcula a potência de um número
    public BigInteger potencia(int numero, int potencia) throws RemoteException;

}
```

```
Listagem 2 - Implementação da classe servidora

package rmi_test;

import java.math.BigInteger;
import java.rmi.AlreadyBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class ServicoPotenciaServidor implements ServicoPotencia {

    // declara o construtor padrão do serviço
    public ServicoPotenciaServidor() throws RemoteException {
        super();
    }

    // calcula o quadrado de um número
    @Override
    public BigInteger quadrado(int numero) throws RemoteException {
        // converte o número recebido para um BigInteger
        BigInteger biNum = new BigInteger(String.valueOf(numero));
        // calcula o quadrado, multiplicando o número por ele mesmo
        biNum.multiply(biNum);
        // retorno o valor calculado
        return biNum;
    }

    // calcula a potência de um número
    @Override
    public BigInteger potencia(int numero, int potencia) throws RemoteException {
        // converte o número recebido para um BigInteger
        BigInteger biNum = new BigInteger(String.valueOf(numero));
        // calcula a potência;
        biNum.pow(potencia);
        // retorno o valor calculado
        return biNum;
    }

    // declaração do método main, que implementa as rotinas de segurança
    public static void main(String[] args) {
        try {
            // cria a instância da classe servidora do serviço
            ServicoPotenciaServidor obj = new ServicoPotenciaServidor();
             // cria a instância da interface de uso da classe servidora, que é a que
será usada pelos clientes
             ServicoPotencia stub = (ServicoPotencia) UnicastRemoteObject.exportObjec
t(obj, 0);

            // inicia o registry RMI
            Registry registry = LocateRegistry.getRegistry();
            // atribui o nome do serviço
            registry.bind("ServicoPotencia", stub);

            System.out.println("Serviço adicionado");
        } catch (RemoteException ex) {
            System.out.println("Erro no servidor: " + ex.getMessage());
        } catch (AlreadyBoundException ex) {
            System.out.println("Erro no servidor: " + ex.getMessage());
        }

    }
}
```

A listagem 3 apresenta um exemplo de um cliente Java, que usa a classe servidora:

Listagem 3 - Exemplo de cliente

```
import java.rmi.*;
import java.rmi.Naming;
import java.io.*;

public class ClienteServicoPotencia
{
        public static void main(String args[]) throws Exception
        {
                // Verifica se os parâmetros de execução estão corretos
                if (args.length != 1)
                {
                        System.out.println
                         ("Para executar este exemplo, digite ClienteServicoPotencia [IP da máquina servidora]");
                        System.exit(1);
                }

                // Tenta executar o serviço no servidor
                 ServicoPotencia servico = (ServicoPotencia) Naming.lookup("rmi://" + args[0] + "/ServicoPotencia");

                DataInputStream din = new DataInputStream (System.in);

                for (;;)
                {
                        System.out.println ("1 - Calcula o quadrado");
                        System.out.println ("2 - Calcula a potência");
                        System.out.println ("3 - Encerrar");
                        System.out.println();
                        System.out.print ("Escolha uma das opções: ");

                        String linha = din.readLine();
                        Integer escolha = new Integer(linha);

                        int valor = escolha.intValue();

                        switch (valor)
                        {
                        case 1:
                          System.out.print ("Número: ");
                          linha = din.readLine();System.out.println();
                          escolha = new Integer (linha);
                          valor  = escolha.intValue();

                          // Executa o método remoto
                            System.out.println("Resposta: " + servico.quadrado(valor));

                          break;
                        case 2:
                          System.out.print ("Número: ");
                          linha = din.readLine();System.out.println();
                          escolha = new Integer (linha);
                          valor  = escolha.intValue();

                          System.out.print ("Potência: ");
                          linha = din.readLine();System.out.println();
                          escolha = new Integer (linha);
                          int potencia = escolha.intValue();

                          // Executa o método remoto
                           System.out.println("Resposta: " + servico.potencia(valor, potencia));

                          break;
                        case 3:
                          System.exit(0);
                        default :
                          System.out.println ("Opção Inválida!!!");
```

```
68.                         break;
69.                         }
70.                 }
71.         }
72.
73. }
```

Para executar estes exemplos, é necessário:

1. iniciar o *rmiregistry* (consulte a documentação no site da Oracle, para verificar como se inicia o rmiregistry na versão so Java SDK que você está usando);
2. compile as classes servidoras, e use a ferramenta *rmic* para criar as interfaces de acesso rmiregistry (consulte a documentação no site da Oracle, para verificar como isso pe feito na versão so Java SDK que você está usando);
3. inicie o servidor, usando *"java NomeClasseServidora"* no diretório em que as classes foram compiladas;
4. execute o cliente, usando o comando "*java NomeClasseCliente localhost"*;

O RMI é uma excelente solução para comunicação entre objetos remotos, pois a programação dos clientes é bastante facilitada pela possibilidade de usar os objetos remotos como se fossem instâncias locais, como se estivessem no mesmo projeto, e responde muito bem aos requisitos de segurança impostos pelos modelos de arquitetura distribuída, uma vez que o serviço RMI somente permite o acesso através da interface de acesso às classes servidoras. O RMI pode ser usado por vários tipos de aplicação Java, entretanto, todo o software precisa ser desenvolvido em Java.

Outra desvantagem do RMI é a mudança nas políticas de segurança que estão sendo aplicadas à linguagem de programação Java nos SDKs mais recentes, para execução de objetos remotos. No caso de atualização da JVM, dependendo de como o software dos serviços remotos foram implementados, pode ser necessário atualizar as classes, e as chamadas aos métodos remotos.

SOAP (Simple Object Access Protocol)

O protocolo simplificado de acesso a objetos, SOAP, é uma especificação de protocolo usado na troca de dados estruturados entre aplicativos para a Internet, através de serviços web (Web Services). Apesar da especificação do SOAP ter começado a ser desenvolvida dentro da Microsoft, a coordenação dos trabalhos sobre esse protocolo foi transferida para a W3C, que disponibiliza as informações relacionadas a esse protocolo através do link https://www.w3.org/TR/soap/ (https://www.w3.org/TR/soap/).

Um exemplo conhecido do uso de SOAP é o serviço de busca de CEP, dos Correios do Brasil, a Empresa de Correios e Telégrafos. Ela disponibiliza um serviço web on-line de consulta os dados do CEP. No caso, o web service deles pede que o sistema que deseja consulta as informações sobre um

determinado CEP informem alguns parâmetros de pesquisa, e a partir desses parâmetros, ele realizado uma busca na base de dados, e envia uma resposta com os resultados que podem satisfazer o pedido do sistema. Tanto a solicitação quanto a resposta, são enviadas através de um protocolo baseado em XML, que é definido pelo web servido, e que deve ser implementado pelo aplicativo que está solicitando o processamento.

A proposta do SOAP é prover um modo para estender as funcionalidades de um aplicativo, mantendo a neutralidade e a independência das várias funcionalidades, tanto na sua forma de comunicação, quando em termos de sistemas operacionais em que elas são executadas, e em relação às linguagens de programação em que essas funcionalidades são escritas.

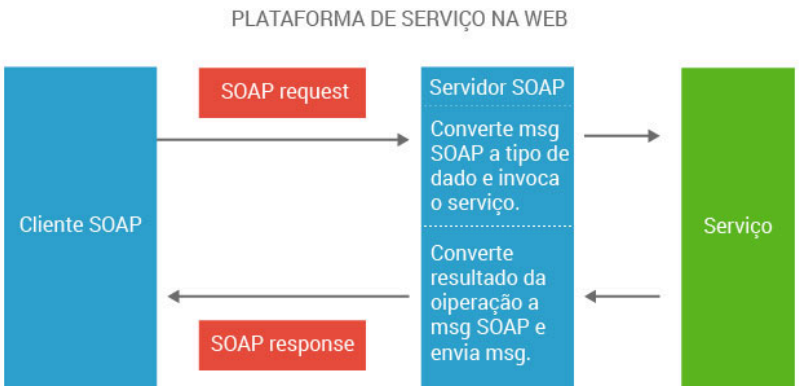

Figura 2 - Princípio de execução remota de serviços, usando SOAP

Essa flexibilidade é possível porque o SOAP se baseia na troca de dados realizada por mensagens que são formatadas de acordo com o padrão XML. Uma mensagem nesse padrão pode ser facilmente escrita por um aplicativo simples, independentemente da linguagem de programação usada. As mensagens são transmitidas de um aplicativo para outro através dos protocolos da camada de aplicação da Internet, sendo os mais usados o HTTP ou o SMTP. Assim, qualquer desenvolvedor pode usar o SOAP para implementar seu aplicativo, desde que ele formate seu arquivo XML do seguinte modo:

- Criar um envelope que identifica o arquivo XML como uma mensagem do tipo SOAP;
- Criar um elemento de cabeçalho, com as informações de cabeçalho;
- Criar um elemento de corpo, com as informações relacionadas à chamada e resposta;
- Criar um elemento de falha, contendo os erros e status da aplicação.

O formato do arquivo XML usado para configurar uma mensagem SOAP possui uma sintaxe específica, que também deve ser seguida:

- A mensagem deve ter um elemento chamado SOAP Envelope;
- A mensagem deve ter um elemento SOAP Encoding;
- A mensagem não deve conter referência a documentos DTD;

- A mensagem não deve conter instruções para o processamento de elementos XML.

A listagem 4 apresenta a forma básica de uma mensagem SOAP, de acordo com as regras definidas:

```
Listagem 4 - Exemplo de estrutura de um arquivo SOAP

<?xml version="1.0"?>


<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">


<soap:Header>
...
</soap:Header>


<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>


</soap:Envelope>
```

SOAP Envelope

Esse elemento (xmlns:soap) é a raíz da mensagem SOAP e é ele quem identifica que o arquivo XML é uma mensagem SOAP. Esse elemento é definido pelo valor "http://www.w3.org/2003/05/soap-envelope/ (http://www.w3.org/2003/05/soap-envelope/)", como pode ser observado na Listagem 4.

SOAP Encoding

Esse elemento (soap:encodingStyle) define o tipo de dado que é usado no arquivo, e pode aparecer em qualquer elemento SOAP, e ele é aplicado ao conteúdo do elemento, e de deus elementos filhos.

SOAP Header

Esse elemento é opcional e contém informações específicas à aplicação (como por exemplo: forma de autenticação, pagamento, etc...) sobre a mensagem SOAP. Se esse elemento está presente, ele deve ser o primeiro elemento filho do elemento Envelope.

```
Listagem 5 - Exemplo de estrutura de um header SOAP

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

O elemento Header da Listagem 2 possui um elemento “trans”, com um atributo “mustUnderstand” definido como tendo o valor 1, enquanto que o valor do elemento “trans” tem o valor 234. Além disso, o elemento Header, por ser um elemento filho de Envelope, herda o atributo encodingStyle definido em Envelope. Essa configuração do Header significa que o aplicativo deverá autenticar se o valor 234 é válido ou não.

Atributo mustUnderstand

Esse atributo determina se um determinado elemento do cabeçalho da mensagem é obrigatório ou opcional por quem está recebendo a mensagem. O valor 1 indica que o elemento deve ser verificado, e 0 indica que o elemento não precisa ser verificado.

SOAP Body

Este elemento contém a mensagem que deve ser tratada pelo serviço web, ou pelo cliente do serviço, e pode-se dizer que é a parte do protocolo que possui os dados que realmente interessam a quem vai fazer o processamento.

```
Listagem 6 - Estrutura e localização do elemento Body

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>


</soap:Envelope>
```

O exemplo acima, formulado pela W3Schools (www.w3schools.com (http://www.w3schools.com)), solicita a um web service que informa o preço de um determinado produto, através do elemento *?m:GetPrice?*, e informa o item que deve ter o preço informado, através do item *?m:Item?*. Esses elementos são definidos pelo desenvolvedor do aplicativo, e são eles que

definem o que o serviço web deverá fazer, e estarão sempre do elemento Body. Os outros elementos do protocolo SOAP são usados para endereçar e iniciar o serviço web solicitado.

Ao realizar o processamento, o web service deverá montar uma resposta, que terá o seguinte formato:

```
Listagem 7 - Resposta SOAP

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Body>
  <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

A forma de se implementar Web Services em Java depende da versão da plataforma Java EE que está sendo usadae desde a versão 5 do Java EE, parte da implementação de aplicativos como os serviços web usa a injeção de código-fonte, para definir uma estrutura ed funcionamento, em substituição aos arquivos XML, bastante comuns até a versão 2 do Java EE. Assim, a Listagem 8 apresenta o código-fonte que implementa um serviço web, que calcula o quadrado de um número, como o exemplo do RMI.

```
Listagem 9 - Código de um Web Service implementado na versão 7 do Java EE, a partir de uma classe Java


package br.uninove.calculadoraws;

import java.math.BigInteger;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

// declaração do web service
@WebService(serviceName = "CalculadoraWS")
public class CalculadoraWS {

    /**
     * Operação de Web service
     */
    @WebMethod(operationName = "quadrado")
    // @WebParam define o parâmetro que deve ser incluído no SOAP,
    // para realizar o cálculo
    public BigInteger quadrado(@WebParam(name = "numero") int numero) {
        String n = String.valueOf(numero);
        BigInteger quadrado = new BigInteger(n);
        quadrado = quadrado.pow(2);
        return quadrado;
    }
}
```

A forma de criar um projeto de Web Service, depende da IDE sendo usada, o que pode variar a mecânica de implementação do projeto. Além disso, como o Web Service obedece a uma certa estrutura de implementação, também é

comum usar ferramentas que construam o software, como o ANT ou o Maven, que aprimoram o uso dos compiladores Java, e a forma de distribuição de um aplicativo Java. Então, a correta forma para implementar o código do serviço web, depende do seu ambiente de desenvolvimento.

Algumas IDEs possuem interfaces de teste para os serviços web, como é mostrado na Figura 3, permitindo acelerar o teste do serviço web.

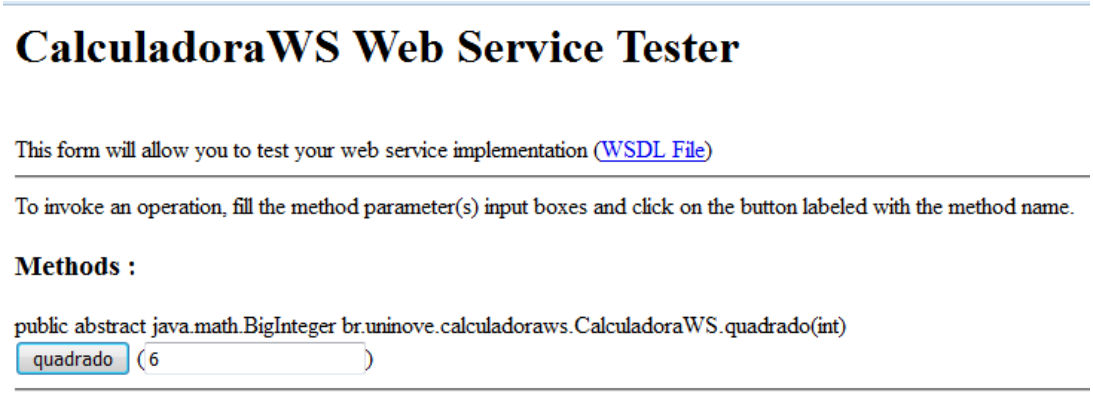

Figura 3 - Ferramenta para teste de web service

Uma das vantagens da injeção de código, é que eles já implementam as partes que tratam do protocolo SOAP, retirando essa carga do desenvolvedor. No caso dessa ferramenta de teste, ela já extrai a forma como o arquivo XML do SOAP deve ser formato, seja para a requisição quanto para a resposta, conforme mostram as listagens 10 e 11.

```
Listagem 10 - Requisição SOAP

<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:quadrado xmlns:ns2="http://calculadoraws.uninove.br/">
            <numero>6</numero>
        </ns2:quadrado>
    </S:Body>
</S:Envelope>
```

```
Listagem 11 - Resposta SOAP

<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:quadradoResponse xmlns:ns2="http://calculadoraws.uninove.br/">
            <return>36</return>
        </ns2:quadradoResponse>
    </S:Body>
</S:Envelope>
```

A implementação de Web Services na plataforma Java EE é feita com o auxílio da API Java para Web Services XML (JAX-WS). As anotações mostradas na Listagem 9 fazem parte dessa API, e são usadas para escrever o protocolo SOAP. Como pôde ser visto, a API JAX-WS JAX-WS simplifica a tarefa de desenvolvimento de Web services utilizando a tecnologia Java, e

resolve alguns problemas enfrentados pela API JAX-RPC. A JAX-WS fornece suporte a vários protocolos usados para os serviços web, e também usa a API JAXB para criar os arquivos em formato XML para vincular os dados que serão usados nas requisições e respostas, e também fornece alguns recursos para poder lidar com os protocolos usados no transporte de informações, como o HTTP.

Como os Web Services trocam dados através de arquivos XML, que podem ser escritos/lidos por qualquer linguagem de programação. Entretanto, os serviços web somente funcionam no ambiente de Internet, o que significa dizer que sempre será necessário traduzir os dados do formato texto, que é o padrão da Internet, para os outros tipos de dados, como os dados numéricos inteiros ou de pontos flutuantes, binários, etc... Isso pode causar um certo atraso no processamento das informações, além de ser necessário criar uma espécie de comunicação via Internet, mesmo que ela não esteja sendo usada.

REST (Representational State Transfer)

A transferência do estado representativo é outra solução para a integração de vários aplicativos, baseada em Web Services. Mas ao contrário dos serviços web baseados em SOAP, que trocam arquivos XML entre os clientes e aplicativos que executam os serviços, o REST foca o uso do protocolo HTTP e nas URI (Identificador Uniforme de Recursos).

Em suma, o REST disponibiliza recursos que são identificados pelos URIs, e eles podem ser manipulados por uma interface padrão, como o HTTP, e a troca de informações ocorre através das representações desses recursos.

A partir da versão 6 do Java EE, foi incorporada à plataforma, a API JAX-RS, que permite ao desenvolvedor implementar serviços web que usam todos os recursos disponibilizados pelo REST. Nessa situação, os web services costumam ser chamados de RESTful Web Services.

As classes que implementam os serviços web RESTful podem ser escritas a partir do zero, ou podem ser desenvolvidas usando alguns frameworks, como o Jersey e o RESTeasy.

Para implementar um Web Service RESTful, as várias IDEs existentes no mercado possuem vários utilitários para , mas os passos costumam ser os mesmos usados para os demais tipos de web services.

Listagem 12 - Classe principal do Web Service REST, que calcula o quadrado de um número

```
package br.uninove.calculadoraws;

import br.uninove.entidades.Potencia;
import java.math.BigInteger;
import javax.json.Json;
import javax.json.JsonObjectBuilder;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.Consumes;
import javax.ws.rs.Produces;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PUT;
import javax.ws.rs.PathParam;
import javax.ws.rs.core.MediaType;

/**
 * Declaração do REST WebService, e define o caminho que será
 * usado para instanciar o Web Service definido por esta classe
 */
@Path("calculadora")
public class CalculadoraResource {

    @Context
    private UriInfo context;

    /**
     * Cria uma instância deste recurso
     */
    public CalculadoraResource() {
    }

    /**
     * Representação da instância de br.uninove.calculadoraws.CalculadoraResource
     * @return um objeto de tipo JSON
     */
    // A anotação GET indica que o caminho do Web Service será direcionado
    // para cá
    @GET
    // Aqui, o @Path indica a seqüência com que os parâmetros serão lidos da URL
    @Path("{numero}")
    // define que a saída do web service é um arquivo JSON
    @Produces(MediaType.APPLICATION_JSON)
    public String getQuadrado(@PathParam("numero") int numero) {
        // lê o valor que foi passado como parâmetro na URL, e o converte para
        // o formato BigInteger
        BigInteger quadrado = new BigInteger(String.valueOf(numero));
        // calcula o quadrado;
        quadrado = quadrado.pow(2);

        // cria um objeto JSON de saída
        JsonObjectBuilder job = Json.createObjectBuilder();
        job.add("numero", numero);
        job.add("potencia", 2);
        job.add("resultado",quadrado);

        // informa o objeto JSON que será a resposta
        return job.build().toString();
    }

    /**
     * PUT, método para criar ou atualizar uma instância de CalculadoraResource
     * @param content representação do recurso
     */
    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    public void putJson(BigInteger content) {
    }
```

```
71. }
```

A Listagem 12 apresenta a classe principal que implementa um recurso que ficará disponível através de um serviço web REST. Para manter o exemplo, ele também realizará o cálculo do quadrado de um número informado, e devolverá um arquivo no formato JSON, com os valores usados no cálculo.

Um Web Service REST depende de uma classe que configura os recursos disponíveis, e que tem o formato da listagem 13, que deve ser modificado apenas com as informações sobre a localização das classes que implementam seu Web Service. Quando se cria um novo serviço web nas IDEs, e indica que ele é um RES, as IDEs já costumam criar essa classe.

```
1.  Listagem 13 - Classe de Configuração do Web Service REST, criado automaticamente pel
    a IDE
2.
3.  /*
4.   * To change this license header, choose License Headers in Project Properties.
5.   * To change this template file, choose Tools | Templates
6.   * and open the template in the editor.
7.   */
8.  package br.uninove.calculadoraws;
9.
10. import java.util.Set;
11. import javax.ws.rs.core.Application;
12.
13. @javax.ws.rs.ApplicationPath("webresources")
14. public class ApplicationConfig extends Application {
15.
16.     @Override
17.     public Set<Class<?>> getClasses() {
18.         Set<Class<?>> resources = new java.util.HashSet<>();
19.         addRestResourceClasses(resources);
20.         return resources;
21.     }
22.
23.     /**
24.      * Do not modify addRestResourceClasses() method.
25.      * It is automatically populated with
26.      * all resources defined in the project.
27.      * If required, comment out calling this method in getClasses().
28.      */
29.     private void addRestResourceClasses(Set<Class<?>> resources) {
30.         resources.add(br.uninove.calculadoraws.CalculadoraResource.class);
31.     }
32.
33. }
```

Para testar esse Web Service, basta compilar o projeto, e instalar no servidor de aplicações que será usado. Depois de instalado, ele pode ser acessado diretamente pela barra de endereços do browser, como mostra a Figura 4.

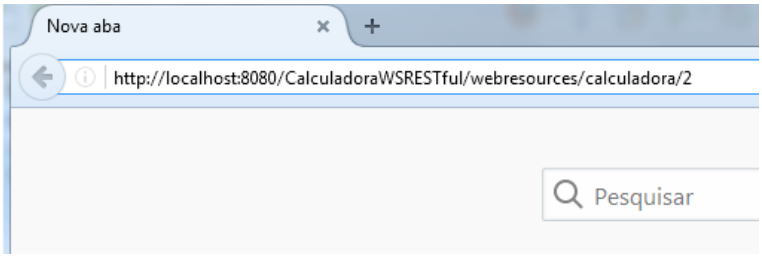

Figura 4 - Chamada de um Web Service do tipo REST na barra de endereço do Browser

A resposta vem no próprio browser, como mostra a Figura 5.

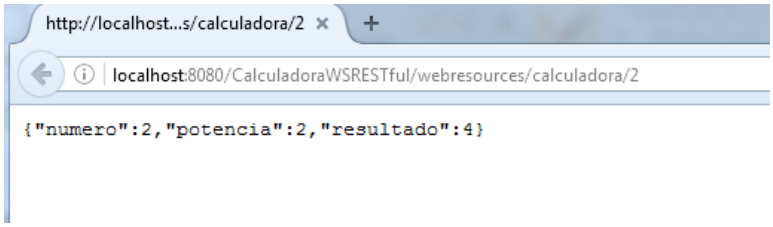

Figura 5 - Resposta enviada pelo Web Service REST

Quiz

Exercício Final

RMI, SOAP (Web Services) e REST

INICIAR

Referências

ORACLE. Remote Method Invocation Home. Disponível em http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html. Acesso em 10/11/2016

W3C (WORLD WIDE WEB CONSORTIUM). W3C SOAP Specifications. Disponível em https://www.w3.org/TR/soap/. Acesso em 10/11/2016

W3SCHOOLS. XML SOAP. Disponível em http://www.w3schools.com/xml/xml_soap.asp. Acesso em 10/11/2016

W3C (WORLD WIDE WEB CONSORTIUM). Web Services Architecture. Disponível em https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. Acesso em 10/11/2016

Avalie este tópico

ANTERIOR
Metadados para Troca de Dados (XML e JSON)

Índice

PRÓXIMO
Design Patterns

Ajuda?
(https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site