01

Acessando um WebService

Transcrição

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/laboratorio-java/stages/07/capitulo7.zip) do projeto completo do capítulo anterior e continuar seus estudos a partir deste capítulo.

Integração entre sistemas

Já resolvemos o problema de como interpretar os dados oriundos de um arquivo XML, mas apesar disso, no mundo real, dados são gerados dinamicamente a todo instante das mais diversas fontes.

No mercado de bolsa de valores é comum o uso de aplicações que permitem aos seus usuários analisar o mercado, e até mesmo comprar e vender ações em tempo real. Mas como é possível analisar o mercado se não temos acesso aos dados da Bovespa?

A integração e a comunicação com o sistema da Bovespa se faz necessária para que possamos receber dados sempre atualizados. Geralmente essa comunicação se dá pelo próprio protocolo da web, o HTTP, com o formato difundido e já estudado XML. Essa integração e comunicação entre aplicações possui o nome de **Web Service**.

O que são Web Services

Por definição, um *Web Service* é alguma lógica de negócio acessível usando padrões da Internet. O mais comum é usar HTTP como protocolo de comunicação e XML para o formato que apresenta os dados - justamente o que praticaremos aqui. Mas nada impede o uso de outros formatos.

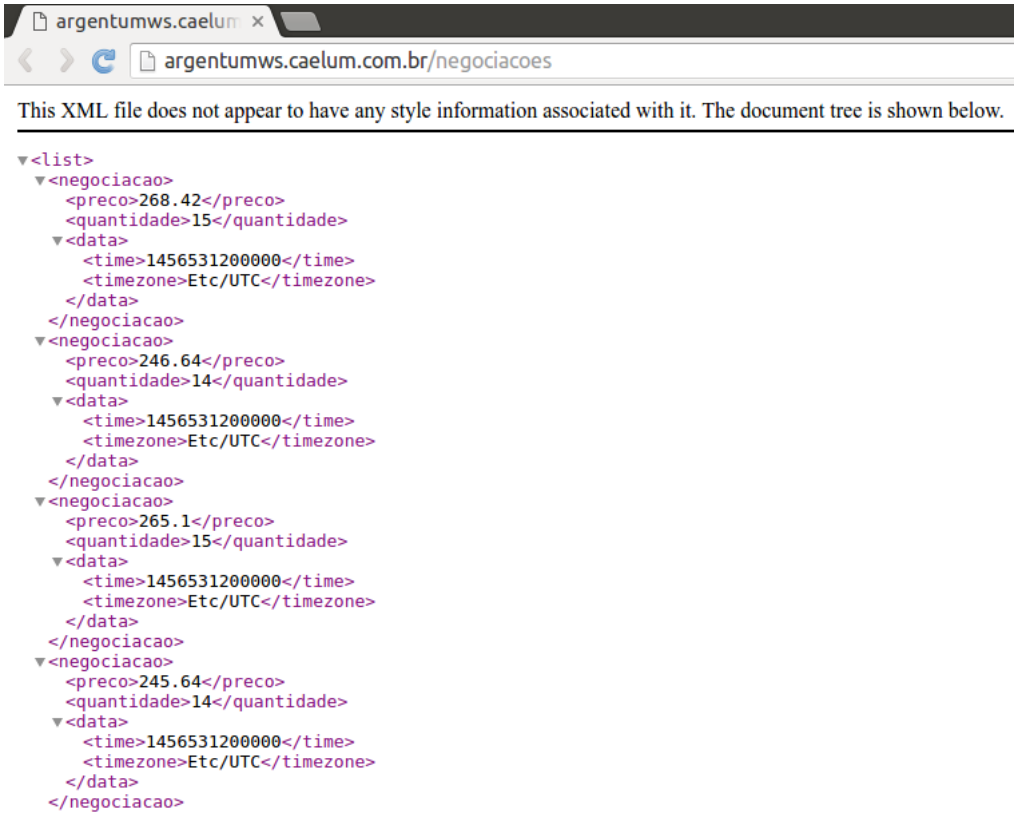
Uma tentativa de especificar mais ainda o XML dos *Web Services* são os padrões **SOAP** e **WSDL**. Junto com o protocolo HTTP, eles definem a base para comunicação de vários serviços no mundo de aplicações *enterprise*. O SOAP e WSDL tentam esconder toda comunicação e geração do XML, facilitando assim o uso para quem não conhece os padrões Web. Neste curso do Alura (https://cursos.alura.com.br/course/web-services-soap) vemos também os detalhes sobre publicação e a criação de clientes *Web Services enterprise* com SOAP e WSDL, além de outros formatos mais simples e modernos.

Outro formato bastante popular nos *Web Services* é o **JSON**. JSON é parecido com XML, mas um pouco menos verboso e fácil de usar com JavaScript. JSON ganhou popularidade através das requisições AJAX e conquistou o seu espaço nos *Web Services* também. Além de ser bastante difundido no desenvolvimento mobile por ser mais leve no tráfego via rede.

Consumindo dados de um Web Service

Para consumir dados vindos de outra aplicação, a primeira coisa importante é saber onde essa aplicação se encontra. Em termos técnicos, qual a URL desse *Web Service*. Em nosso projeto, a URL específica da aplicação será **https://argentumws-spring.herokuapp.com/negociacoes (https://argentumws-spring.herokuapp.com/negociacoes)**.

Podemos testar essa URL facilmente dentro do navegador, basta copiar e colar na barra de endereço. Ao executar, o navegador recebe como resposta o XML de negócios que já conhecemos,por exemplo:



Criando nosso cliente Java

Já sabemos de onde consumir, resta saber como consumir esses dados pela web. No mundo web trabalhamos com o conceito de requisição e resposta. Se queremos os dados, precisamos realizar uma requisição para aquela URL, mas como?

Na própria **API** do Java temos classes que tornam possível essa tarefa. Como é o caso da classe URL, que nos permite referenciar um recurso na Web, seja ele um arquivo ou até mesmo um diretório:

```
URL url = new URL("https://argentumws-spring.herokuapp.com/negociacoes");
```

Conhecendo a URL, falta agora que uma requisição HTTP seja feita para ela. Faremos isso através do método `openConnection`, que nos devolve um `URLConnection`. Entretanto, como uma requisição HTTP se faz necessária, usaremos uma subclasse de `URLConnection`, que é a `HttpURLConnection`:

```
URL url = new URL("https://argentumws-spring.herokuapp.com/negociacoes");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
```

Dessa conexão, pediremos um `InputStream` que será usado pelo nosso `LeitorXML`:

```
URL url = new URL("https://argentumws-spring.herokuapp.com/negociacoes");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
InputStream content = connection.getInputStream();
List<Negociacao> negociacoes = new LeitorXML().carrega(content);
```

Dessa forma já temos em mãos a lista de negociações, que era o nosso objetivo principal. Resta agora encapsularmos este código em alguma classe, para que não seja necessário repeti-lo toda vez que precisarmos receber os dados da

negociação. Vamos implementar a classe `ClienteWebService`, dentro do pacote `br.com.alura.argentum.ws`, e nela deixar explícito qual o caminho da aplicação que a conexão será feita:

```
public class ClienteWebService {

    private static final String URL_WEBSERVICE  =
            "https://argentumws-spring.herokuapp.com/negociacoes";
}
```

Por último, mas não menos importante, declararemos um método que retorna uma lista de negociações, justamente o que usaremos no projeto.

```
public class ClienteWebService {

    private static final String URL_WEBSERVICE  =
            "https://argentumws-spring.herokuapp.com/negociacoes";

    public List<Negociacao> getNegociacoes() {

        URL url = new URL(URL_WEBSERVICE);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();
        InputStream content = connection.getInputStream();
        return new LeitorXML().carrega(content);
    }
}
```

Nossa classe ainda não compila, pois tanto o construtor de `URL` quanto os métodos de `HttpURLConnection` lançam exceções que são do tipo `IOException`. Vamos tratar o erro e fechar a conexão que foi aberta pelo método `getInputStream`, em um bloco `finally`:

```
public List<Negociacao> getNegociacoes() {

    HttpURLConnection connection = null;

    try {
        URL url = new URL(URL_WEBSERVICE);
        connection = (HttpURLConnection)url.openConnection();
        InputStream content = connection.getInputStream();

        return new LeitorXML().carrega(content);
    } catch (IOException e) {
        throw new RuntimeException(e);
    } finally {
        connection.disconnect();
    }
}
```

Dessa forma conseguimos nos comunicar com um **Web Service** e consumir os dados disponibilizados por ele através de um XML. Esta ainda é uma prática bastante utilizada pelo mercado.

O que aprendemos:

- Criar um Client para o Webservice.
- A extrair constantes para atributos da classe.
- Lembrar sempre de fechar as conexões.