

CAPÍTULO 6

Acessando um Web Service

"Nenhum homem é uma ilha isolada; cada homem é uma partícula do continente, uma parte da terra"
— John Donne

6.1 – INTEGRAÇÃO ENTRE SISTEMAS

No capítulo anterior resolvemos o problema de como interpretar os dados oriundos de um arquivo XML, 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?

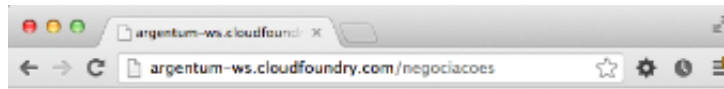
A integração e a comunicação com o sistema da Bovespa se faz necessária para que possamos receber dados sempre atualizados. Geralmente essa comunicação se dá pelo próprio protocolo da web, o HTTP, com o formato difundido e já estudado XML. Essa integração e comunicação entre aplicações possui o nome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

6.2 – 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 <http://argementws.caelum.com.br/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:



```
<list>
  <negociacao>
    <preco>250.64</preco>
    <quantidade>14</quantidade>
    <data>
      <time>1357516800000</time>
      <timezone>Btc/UTC</timezone>
    </data>
  </negociacao>
  <negociacao>
    <preco>415.37</preco>
    <quantidade>24</quantidade>
    <data>
      <time>1357516800000</time>
      <timezone>Btc/UTC</timezone>
    </data>
  </negociacao>
  <negociacao>
    <preco>336.48</preco>
    <quantidade>19</quantidade>
    <data>
      <time>1357516800000</time>
      <timezone>Btc/UTC</timezone>
    </data>
  </negociacao>
  <negociacao>
    <preco>250.64</preco>
    <quantidade>14</quantidade>
    <data>
      <time>1357516800000</time>
      <timezone>Btc/UTC</timezone>
    </data>
  </negociacao>
</list>
```

Seus livros de tecnologia parecem do século passado?



Conheça a **Casa do Código**, uma **nova** editora, com autores de destaque no mercado, foco em **ebooks** (PDF, epub, mobi), preços **imbatíveis** e assuntos **atuais**.

Com a curadoria da **Caelum** e excelentes autores, é uma abordagem **diferente** para livros de tecnologia no Brasil.

Conheça os títulos e a nova proposta, você vai gostar.

[Casa do Código, livros para o programador.](http://www.casadocodigo.com.br)

6.3 – CRIANDO O 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("http://argentumws.caelum.com.br/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("http://argentumws.caelum.com.br/negociacoes");  
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
```

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

```
URL url = new URL("http://argentumws.caelum.com.br/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 precisamos receber os dados da negociação. Vamos implementar a classe `ClienteWebService` e nela deixar explícito qual o caminho da aplicação que a conexão será feita.

```
public class ClienteWebService {  
  
    private static final String URL_WEBSERVICE =  
        "http://argentumws.caelum.com.br/negociacoes";  
}
```

Por último, e 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 =  
        "http://argentumws.caelum.com.br/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 `URLConnection` 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 = (URLConnection)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 é uma prática bastante utilizada pelo mercado e estudada com mais aprofundamento no curso **FJ-31 | Curso Java EE avançado e Web Services**.

HttpClient

Existe uma biblioteca capaz de lidar com dados de uma forma simples e mais específica do que utilizar diretamente a API do Java. Para trabalhar com o protocolo HTTP há a biblioteca **HttpClient** que faz parte do Apache Software Foundation:

<http://hc.apache.org/httpcomponents-client-ga/index.html>

Com ela ganhamos uma API que fornece toda funcionalidade do protocolo HTTP e poderíamos usá-la para chamar o Web Service. Segue um pequeno exemplo usando o `HttpClient` para executar uma requisição do tipo *GET*:

```
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet(URL_DO_WEBSERVICE);
HttpResponse response = client.execute(request);
InputStream content = response.getEntity().getContent();
```

Web Service – SOAP, JSON e outros

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. No treinamento FJ-31 veremos os detalhes sobre publicação e a criação de clientes baseados no Web Services SOAP/WSDL.

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.

6.4 – EXERCÍCIOS: NOSSO CLIENTE WEB SERVICE

1. Vamos agora implementar o cliente do Web Service, primeiramente criaremos a classe `ClienteWebService`, dentro do pacote `br.com.caelum.argentum.ws` na pasta `src/main/java`. Vamos criar também uma constante com a URL para onde será feita a requisição.

```
public class ClienteWebService {  
  
    private static final String URL_WEBSERVICE =  
        "http://argentumws.caelum.com.br/negociacoes";  
}
```

2. Agora vamos criar o método `getNegociacoes()`, que retorna a nossa lista de negociações:

```
public class ClienteWebService {  
  
    private static final String URL_WEBSERVICE =  
        "http://argentumws.caelum.com.br/negociacoes";  
}
```

```
public List<Negociacao> getNegociacoes() {  
  
    HttpURLConnection connection = null;  
  
    URL url = new URL(URL_WEBSERVICE);  
  
    connection = (HttpURLConnection)url.openConnection();  
  
    InputStream content = connection.getInputStream();  
  
    return new LeitorXML().carrega(content);  
}  
}
```

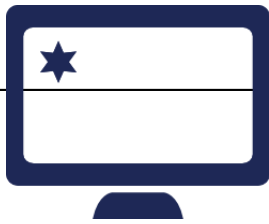
3. Não podemos esquecer de colocar o try/catch para tratar possíveis erros e logo em seguida fechar a conexão:

```
public class ClienteWebService {  
  
    private static final String URL_WEBSERVICE =  
        "http://argentumws.caelum.com.br/negociacoes";  
  
    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();  
        }  
    }  
}
```

6.5 – DISCUSSÃO EM AULA: COMO TESTAR O CLIENTE DO WEB SERVICE?

Agora é a melhor hora de aprender algo novo

Se você gosta de estudar essa apostila aberta da Caelum, certamente vai gostar dos novos **cursos online** que lançamos na plataforma **Alura**. Você estuda a qualquer momento com a **qualidade** Caelum.



[Conheça a Alura.](#)

CAPÍTULO ANTERIOR:

[Test Driven Design – TDD](#)

PRÓXIMO CAPÍTULO:

[Introdução ao JSF e Primefaces](#)

Você encontra a Caelum também em:

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter