01

Fazendo consultas com XPath

Transcrição

Até o momento aprendemos três formas de ler um documento XML: a forma STAX, na qual percorremos uma lista de eventos; a SAX, noa qual criamos uma classe que definirá como parsear esse documento; e a DOM, na qual montamos o documento na memória e o consultamos por meio de alguns métodos.

Mas e se tivéssemos um documento XML com 50 produtos e precisássemos realizar uma consulta, pegando apenas o segundo produto da lista? Para fazermos isso com `Sistema.java`, que utiliza o método DOM, teríamos que passar um intervalo que começasse com `i` igual a `1` e acabasse quando `i` fosse menor que `2`.

```
for(int i = 1;i < 2;i++) {
    Element produto = (Element) produtos.item(i);
    String nome = produto.getElementsByTagName("nome").item(0).getTextContent();
    double preco = Double.parseDouble(produto.getElementsByTagName("preco").item(0).getTextCont
    Produto prod = new Produto(nome, preco);

    System.out.println(prod);
}
```

Com isso, conseguiríamos retornar o segundo produto com sucesso. Entretanto, poderíamos ter um filtro bem mais complicado que somente pegar o segundo item de uma lista. Por exemplo, e se quiséssemos todos os produtos que contêm o nome "livro"? Uma alternativa seria adicionarmos uma verificação consultando de `nome` possui (`contains()`) essa palavra, adicionando o produto na lista em caso positivo.

```
for(int i = 0;i < produtos.getLength();i++) {
    Element produto = (Element) produtos.item(i);
    String nome = produto.getElementsByTagName("nome").item(0).getTextContent();

    if(nome.contains("Livro")) {
        double preco = Double.parseDouble(produto.getElementsByTagName("preco").item(0).getText
        Produto prod = new Produto(nome, preco);
        System.out.println(prod);
    }

}
```

Perceba que, para fazermos essa filtragem, precisamos incluir um `if` no nosso `for`. Mas e se o filtro fosse ainda mais complicado, incluindo categorias, faixas de preço e outras características? Nesse caso teríamos um encadeamento muito extenso de operadores `if` apenas para realizarmos uma consulta.

Pensando justamente em cenários como esse, nos quais consultas em XML podem ser muito complicadas, foi criado o XPath, uma espécie de linguagem que nos permite realizar consultas em uma árvore de elementos - no nosso caso, o DOM. O XPath é bem parecido com o SQL em bancos de dados: nós escrevemos uma string representando uma consulta e, com a execução, recebemos os resultados.

Para testarmos o funcionamento do XPath, voltaremos o código de `Sistema` para sua versão original. Então, no método `main()`, criaremos uma string `exp` (de "expressão"). O XPath opera por meio de expressões que lembram bastante um diretório do sistema operacional, como `/Desktop/cursos-xml`, e cada tag funciona como um diretório específico. Sendo assim, se queremos pegar todos os produtos que estão dentro da tag `<produtos>` e que por sua vez está dentro da tag `<venda>`, teríamos algo como `/venda/produtos/produto`.

```
String exp = "/venda/produtos/produto";
```

Para transformarmos essa string em um XPath, criaremos uma nova instância a partir de `XPathFactory`.

```
String exp = "/venda/produtos/produto";
XPath path = XPathFactory.newInstance().newXPath();
```

Feitas as importações necessárias, chamaremos o método `path.compile()` passando como argumento a `exp` que criamos, e armazenaremos o retorno em uma variável `expression`.

```
String exp = "/venda/produtos/produto";
XPath path = XPathFactory.newInstance().newXPath();
XPathExpression expression = path.compile(exp);
```

Agora precisamos executar essa expressão no nosso documento. Para isso, chamaremos `expression.evalute()`, um método que precisa de dois argumentos: a árvore de elementos, que é o nosso `document`; e o tipo de retorno, que usarmeos `XPathConstants.NODSET`, já que essa consulta pode retornar mais de um nó. Por fim, faremos um cast com `(NodeList)` para obtermos o objeto correto dessa chamada.

```
String exp = "/venda/produtos/produto";
XPath path = XPathFactory.newInstance().newXPath();
XPathExpression expression = path.compile(exp);

NodeList produtos = (NodeList) expression.evaluate(document, XPathConstants.NODESET);
```

Se executarmos nosso código, a consulta funcionará corretamente, trazendo todos os produtos. Agora, se quisermos fazer outras consultas, precisaremos alterar somente a expressão `exp`. Se quisermos somente o segundo produto, por exemplo, passaremos a expressão `/venda/produtos/produto[2]`, evidenciando o índice desejado entre colchetes.

Também podemos fazer consultas um pouco mais complicadas. Por exemplo, para conseguirmos todos os produtos com a palavra "Livro":

```
String exp = "/venda/produtos/produto[nome='Livro']";
```

Isso não nos retornará nenhum produto, pois `nome='Livro'` faz uma busca exata pelo nome "Livro". Já se pesquisarmos por `Livro de xml`, teremos o produto desejado como retorno. Também é possível utilizar algumas expressões regulares ou funções de apoio. Se não quisermos o nome exato, por exemplo, podemos utilizar a função `contains()`:

```
String exp = "/venda/produtos/produto[contains(nome, 'Livro')]";
```