02

Refatorando a lógica STAX

Transcrição

O objetivo agora é melhorarmos um pouco o código da nossa mais recente abordagem para a leitura de um XML. No caso, temos um `while` bastante complexo, com diversos `if` em seu interior. A lógica desse `while` é pegar o próximo evento e, se tal evento for da tag `<produto>`, criar um `produto`.

Uma das técnicas mais famosas de refatoração é extrair um código muito complexo para um método separado. Por exemplo, dentro do `while`, depois que identificamos que um elemento `produto` foi iniciado, queremos ter um método `criaUmProduto()` que utiliza como base os eventos que conhecemos. Sendo assim, criaremos o método estático `criaUmProduto()` que, baseado na nossa listagem `eventos`, devolve um `Produto`.

```
private static Produto criaUmProduto(XMLEventReader eventos) {
    // TODO Auto-generated method stub
    return null;
}
```

No corpo desse método incluiremos a lógica de criação que fizemos no vídeo anterior, incluindo a criação de um objeto `Produto`, o que nos permitirá removê-la do método `main()`. Para percorrermos a lista de eventos, manteremos o `while(eventos.hasNext())`. Criaremos as variáveis locais necessárias (`XMLEvent evento` e `Produto produto`) e adicionaremos um `throws Exception` ao nosso método `criaUmProduto()`. Ao final da execução, ao invés de adicionarmos `produto` a uma lista, simplesmente sairemos do loop com um `break` e retornaremos o `produto`. Por fim, no método `main()`, adicionaremos esse `produto` na lista `produtos` com um `add()`.

```
public class LeArquivoXmlTerceiraForma {
    public static void main(String[] args) throws Exception {
        InputStream is = new FileInputStream("src/vendas.xml");
        XMLInputFactory factory = XMLInputFactory.newInstance();
        XMLEventReader eventos = factory.createXMLEventReader(is);
        List<Produto> produtos = new ArrayList<>();

        while(eventos.hasNext()) {
            XMLEvent evento = eventos.nextEvent();

            if(evento.isStartElement() && evento.asStartElement().getName().getLocalPart().equa
                Produto produto = criaUmProduto(eventos);
                produtos.add(produto);
            }

        }

        System.out.println(produtos);
    }

    private static Produto criaUmProduto(XMLEventReader eventos) throws Exception {
        Produto produto = new Produto();

        while(eventos.hasNext()) {
            XMLEvent evento = eventos.nextEvent();
```

```
                if(evento.isStartElement() && evento.asStartElement().getName().getLocalPart().e
                    evento = eventos.nextEvent();
                    String nome = evento.asCharacters().getData();
                    produto.setNome(nome);
                }else if(evento.isStartElement() && evento.asStartElement().getName().getLocalPa
                    evento = eventos.nextEvent();
                    String conteudo = evento.asCharacters().getData();
                    double preco = Double.parseDouble(conteudo);
                    produto.setPreco(preco);
                }else if(evento.isEndElement() && evento.asEndElement().getName().getLocalPart(
                    break;
            }

        } return produto;
    }
}
```

Dessa forma a lógica do nosso código fica um pouco mais legível: percorremos os eventos, pegamos o próximo e, se for a abertura de uma tag `<produto>`, criamos um novo `produto` e adicionamos as suas respectivas informações. Finalizando, adicionamos esse produto na lista. Essa técnica, que consiste em percorrer uma lista de eventos, é chamada de STAX. Pensando nisso, renomearemos a nossa classe para `LeXmlStax.java`.

Mas e aí, qual desses métodos devemos utilizar para ler XML no dia-a-dia? Isso vai variar muito a depender da situação. O SAX e o STAX são bastante parecidos: enquanto no primeiro nós notificados sobre quais eventos ocorreram, no segundo nós ativamente buscamos essa informação. A principal diferente entre esses dois e o DOM é que o último carrega a árvore de elementos na memória.