03

Tratando eventos

Transcrição

Agora começaremos a escrever a lógica que usaremos para ler o arquivo XML. Primeiro, precisamos entender como a classe `XMLReader` funciona. Ao longo da leitura do XML, essa classe notificará ao nosso `LeitorXml` alguns eventos, como a abertura da tag `<venda>`. Poderemos, então, programar o que deverá ser feito quando esse evento acontecer.

Da mesma forma, quando a `XMLReader` encontrar o elemento `<formaDePagamento>`, ela passará não só a abertura da tag, como também a leitura do conteúdo e o fechamento dessa tag. Com base nesses três eventos, conseguiremos ler qualquer arquivo XML.

Na ocorrência de um evento, o `XMLReader` chamará o método `startElement()` do nosso objeto `logica`. Porém, repare que nosso `LeitorXml` não possui esse método, e ele precisa vir de algum lugar - no caso, a classe `DefaultHandler`. Podemos utilizar o `@Override` para sobrescrevermos esse método da forma que desejarmos.

O `startElement()` recebe alguns parâmetros, como `uri`, `localName`, `qName` e `attributes`, e lança uma exceção do tipo `SAXException`. Ao invés da implementação padrão sugerida pelo Eclipse, pediremos um `System.out.println()` da mensagem `"abriu a tag:"` somado de `qName`, que consiste nos nomes das tags.

```
public class LeitorXml extends DefaultHandler{
    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes)
        System.out.println("abriu a tag: " + qName);

        if(qName.equals("produto")) {

        }
    }
}
```

Assim, teremos como retorno:

```
abriu a tag: venda

abriu a tag: formaDePagamento

abriu a tag: produtos

abriu a tag: produto

abriu a tag: nome

abriu a tag: preco

abriu a tag: produto

abriu a tag: nome
```

abriu a tag: preco

Ou seja, uma lista de todas as tags que foram abertas, exatamente na ordem em que elas aparecem no XML. Queremos carregar a lista de produtos na memória, portanto é interessante sabermos quando a tag `produto` é aberta. Podemos utilizar o operador `if` e o método `equals()` para encontrarmos esse momento e, posteriormente, executar alguma ação.

```
if(qName.equals("produto")) {
    //ação a ser executada
}
```

Queremos carregar um produto na memória. Para isso, instanciaremos um objeto `Produto` e o armazenaremos em uma variável local `produto`.

```
public class LeitorXml extends DefaultHandler{
    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes)
        System.out.println("abriu a tag:" + qName);

        if(qName.equals("produto")) {
            Produto produto = new Produto();
        }
    }
}
```

Para que nosso código compile, precisaremos criar um construtor vazio na classe `Produto`:

```
public Produto() {
    // TODO Auto-generated constructor stub
}
```

Depois de conseguirmos o produto, o próximo passo será lermos o conteúdo da tag `nome`. Podemos conseguir esse evento com o método `characters()`, que sobrescreveremos assim como o `startElement()`. Tal método recebe uma sequência de caracteres e um intervalo (`start` e `lenght`), que armazenaremos em uma nova string e atribuiremos a uma variável `conteudo`.

```
@Override
public void characters(char[] ch, int start, int lenght) throws SAXException {
    conteudo = new String(ch, start, lenght);
}
```

Como o método `characters()` nos passa um intervalo, pode ser que ele seja executado várias vezes para ler o conteúdo de cada tag, sendo necessário acumularmos esse conteúdo. Para isso, podemos utilizar a classe `StringBuilder`, que chamaremos de `conteudo`. A partir dela, chamaremos o método `append()`.

```
public class LeitorXml extends DefaultHandler{
```

```
    StringBuilder conteudo;

    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes)
        System.out.println("abriu a tag:" + qName);

        if(qName.equals("produto")) {
            Produto produto = new Produto();
        }

    }

    @Override
    public void characters(char[] ch, int start, int lenght) throws SAXException {
        conteudo.append(new String(ch, start, lenght));
    }
}
```

Após salvarmos o conteúdo, precisamos fechar a tag `produto`. Para isso, sobrescreveremos o método `endElement()` e executaremos o operador `if` e o método `equals()` para verificarmos quando `qName` corresponder à tal tag. Nosso objetivo é salvarmos o produto que criamos em uma lista. Sendo assim, teremos que criar uma `List<Produto>`, que chamaremos de `produtos`, e que será uma instância de `ArrayList<>`. Com ambas as classes importadas, adicionaremos o `produto` à lista `produtos` com o método `add()`.

Para acessarmos a variável `produto`, vamos transformá-la em um novo atributo. Com isso, o método `startElement()` somente manipulará esse atributo, que também poderemos utilizar no método `endElement()`.

```
public class LeitorXml extends DefaultHandler{

    List<Produto> produtos = new ArrayList<>();
    StringBuilder conteudo;
    Produto produto;

    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes)
        System.out.println("abriu a tag:" + qName);

        if(qName.equals("produto")) {
            produto = new Produto();
        }

    }

    @Override
    public void characters(char[] ch, int start, int lenght) throws SAXException {
        conteudo.append(new String(ch, start, lenght));
    }

    @Override
    public void endElement(String uri, String localName, String qName) throws SAXException {
        if(qName.equals("produto")) {
            produtos.add(produto);
        }
```

```
    }
}
```

Terminada a execução do código, queremos que todos os produtos estejam na memória. Para tanto, no nosso método `main()`, faremos um `System.out.println()` de `logica.produtos`.

```
public static void main(String[] args) throws Exception {
    XMLReader leitor = XMLReaderFactory.createXMLReader();
    LeitorXml logica = new LeitorXml();
    leitor.setContentHandler(logica);
    InputStream ips = new FileInputStream("src/vendas.xml");
    InputSource is = new InputSource(ips);
    leitor.parse(is);

    System.out.println(logica.produtos);
}
```

Se executarmos o código dessa forma, receberemos uma `NullPointerException`, ocorrido porque a variável `conteudo` não possui nenhuma informação. Corrigiremos isso iniciando um novo `StringBuilder()` toda vez que um elemento for aberto. Aproveitaremos para remover o log que avisa quando cada tag é aberta, facilitando nossa leitura do retorno.

```
@Override
public void startElement(String uri, String localName, String qName, Attributes attributes) thro

    if(qName.equals("produto")) {
        produto = new Produto();
    }

    conteudo = new StringBuilder();

}
```

Executando o código, teremos:

```
[Nome:null
Preço:0.0
, Nome:null
Preço:0.0
]
```

Os dois produtos foram carregamos na memória, mas o nome e o preço vieram nulos! Para corrigirmos isso, precisaremos trabalhar novamente com os eventos. Por exemplo, quando a tag `nome` for fechada, chamaremos `produto.setNome()` passando a string armazenada no nosso conteúdo (`conteudo.toString()`). Repetiremos esse processo quando a tag `preco` for fechada, dessa vez criando uma variável `double` que receberá a chamada de `Double.parseDouble()` passando novamente a string do nosso conteúdo. Por fim, armazenaremos o `preco` com `produto.setPreco()`.

```
@Override
public void endElement(String uri, String localName, String qName) throws SAXException {
```

```
        if(qName.equals("produto")) {
            produtos.add(produto);
        }

        else if(qName.equals("nome")) {
            produto.setNome(conteudo.toString());
        }
        else if(qName.equals("preco")) {
            Double preco = Double.parseDouble(conteudo.toString());
            produto.setPreco(preco);
        }
    }
```

Executando novamente, teremos:

```
[Nome:Livro de xml
Preço:29.9
, Nome:Livro de O.O. java
Preço:29.9
]
```

Assim, conseguimos ler o nome e o preço de cada um dos produtos cadastrados no nosso XML, dessa vez por meio do tratamento de eventos. Para isso, precisamos criar uma classe nova e, em comparação com a primeira abordagem, nosso código ficou um pouco mais complexo, o que pode ser visto como uma desvantagem. No próximo vídeo tentaremos melhorar esse código.