XML w Jakarta

Michał Sitko Katarzyna Surzyn Jakub Piątek Patryk Świerkowski Krzysztof Rokosz

XML (Extensible Markup Language)

- Tekstowy format reprezentacji danych

- Podobnie jak JSON służy do wymiany danych między programami

- Zaprojektowany pod przyjazność dla technologii Web

```
<?xml version="1.0" encoding="UTF-8"?>
<websites>
  <website id="133">
    <title lang="en">File Extension Database</title>
    <url>https://www.file-extension.info</url>
    <category>Data Formats</category>
  </website>
</websites>
```

Wady

- Wysoka złożoność formatu
- Czasochłonna walidacja i przetwarzanie
- Nieczytelność dla człowieka w “surowym” stanie

Zalety

- Walidacja na poziomie formatu
- Duża gęstość danych
- W większości wspierane na technologiach “legacy”

Protokół Soap i JAX-WS

SOAP (Simple Object Acces Protocol)

- Pozwala komunikować się aplikacjom napisanym w różnych technologiach za pośrednictwem internetu

- Korzysta z formatu XML do komunikacji

- Endpointy opisane są w specjalnym języku WSDL

- Wsparcie w środowisku java zakończone z pojawieniem się java 9

JAX-WS (Java API for XML web services)

- Służy jako pośrednik w serwisie SOAP

- Tłumaczy dane XML na obiekty Java

- Odpowiada za generacje artefaktów w technologii SOAP

Przykładowa Aplikacja SOAP

Aplikacja do obliczania równania kwadratowego

Jako programiści musimy tylko zaimplementować Web Service. Klasa musi być oznaczona adnotacją.

@WebService

I wewnątrz musi się znajdować co najmniej jedna metoda oznaczona

@WebMethod

Resztą zajmie się JAX-WS

```
/**
 *
 * @author micha
 */
@WebService(serviceName = "kwadratowe")
@Stateless()
public class kwadratowe {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "zerospot")
    public List zerospot(@WebParam(name = "a") double a
            , @WebParam(name = "b") double b
            , @WebParam(name = "c") double c) {
        double square = Math.sqrt((b*b) - (4.0*a*c));
        double zero1 = (-b-square)/(2*a);
        double zero2 = (-b+square)/(2*a);
        List<Double> zeros = new ArrayList();
        zeros.add(zero1);
        zeros.add(zero2);
        return zeros;
    }
```

Wywołano następnie tę metodę z pewnymi parametrami.

Bindowaniem elementów XML na elementy java i wice-wersa zajmuję się za nas JAX-WS

Pomimo że metoda web zwraca obiekt typowy dla Java czyli ArrayList, nadal jest ona zwracana w sposób który może zostać zrozumiany przez inny program

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body xmlns:ns2="http://lab8.mycompany.com/">
        <ns2:zerospot>
            <a>0.8</a>
            <b>3.0</b>
            <c>1.0</c>
        </ns2:zerospot>
    </S:Body>
</S:Envelope>
```

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body xmlns:ns2="http://lab8.mycompany.com/">
        <ns2:zerospotResponse xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <return xsi:type="xs:double">-3.3801993223490365</return>
            <return xsi:type="xs:double">-0.369800677650963</return>
        </ns2:zerospotResponse>
    </S:Body>
</S:Envelope>
```

Przetwarzanie XML

XML w Jakarta

Może być parsowany przez:

- JAXP
- StAX
- JAXB

Reprezentacja DOM

Jeżeli XML nie jest za duży można go odczytać i zapisać w pamięci za pomocą reprezentacji DOM - document object model. W tej reprezentacje każde poddrzewo nazywane jest jako “node” i jest instancją interfejsu org.w3c.dom.Node.

Do zbudowania drzewa możemy użyć klasę pomocniczą, która jest zapisana po prawej stronie.

```
class DOMIterator {
    private int i = 0;
    private Node n;
    public DOMIterator(Node n)
    {this.n = n;}
    public Stream<Node> stream()
    {
        NodeList nl = n.getChildNodes();
        int len = nl.getLength();
        return len == 0 ?Stream.empty():
Stream.iterate(nl.item(0), n2 -> {i++; return
nl.item(i); }).limit(len);
    }
    public static Stream<Node> stream(Node n)
    {return new DOMIterator(n).stream();}
}
```

Prosty przykład odczytania dokumentu DOM

```
String xml = ...;
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
InputSource is = new InputSource(new ByteArrayInputStream(xml.getBytes()));
Document doc = db.parse(is);
```

Reprezentacja StAX

StAX w odróżnieniu od DOM nie zapisuje XML w pamięci, a używa iteratora w celu zeskanowania XMLa. Dzięki czemu dużo łatwiej możemy poruszać się po dużych dokumentach, ponieważ nie musimy pobierać całego dokumentu, a tylko poszczególne rekordy.

Najlepszym sposobem na wykorzystanie możliwości StAX jest korzystanie z przesyłania strumieniowego Java 8 i klasy pomocniczej do pośredniczenia między StAX a strumieniami.

StAX processing

- Dodanie do projektu biblioteki ProtonPack
- Zaaktualizowanie projektu
- Stworzenie klasy pomocniczej do komunikacji pomiędzy StAX i strumieniami
- Wykorzystanie klasy pomocniczej do stworzenia strumieni

StAX processing

Wykorzystanie klasy pomocniczej do stworzenia strumieni

- przykładowy kod :

```
1   // count records
2   bis.reset();
3   XMLStreamReader parser3 =
4    factory.createXMLStreamReader(bis);
5   long recNum = StaxIterator.stream(parser3).
6    filter(elem -> elem.isStartElement()).
7    filter(elem -> elem.getLocalName().equals("recording")).
8    count();
9   System.err.println(recNum);
10
11  // show all elements
12  StaxIterator.stream(parser).
13   filter(elem -> elem.isStartElement()).
14   forEach(sr -> {
15   System.err.println(sr.getLocalName());
16  });
```

Reprezentacja SAX

Reprezentacja SAX w odróżnieniu od reprezentacji DOM lub StAX używa klasy, która umożliwia odczytanie części dokumentu XML podczas parsowania. Nazywana jest inaczej jako “push”, ponieważ wysyła ona wydarzenia do aplikacji.

Do odczytu danych potrzebujemy klasy nasłuchującej, która zazwyczaj filtruje i zamienia dane. Dane odczytujemy tworząc handler, który uruchomi klasę i zacznie parsować dane.

Klasa nasłuchująca

```
class UserHandler extends DefaultHandler {
    @Override
    public void startElement(String uri, String localName, String qName,Attributes attributes)throws SAXException {
        String attrs = "[";
        for(int i=0; i < attributes.getLength();i++) {
            attrs += attributes.getLocalName(i) + "=" +attributes.getValue(i)+ ",";
        }
        attrs = attrs.length() > 1 ?attrs.substring(0, attrs.length()-1) : attrs;attrs += "]";
        System.err.println("-> " + localName + " - " +uri + " - " + qName + " - " + attrs);
        }
        @Override
        public void endElement(String uri,String localName, String qName)throws SAXException {
            System.err.println("<- " + localName + " - " +uri + " - " + qName);
        }
        @Override
        public void characters(char ch[], int start,int length) throws SAXException {
            String chars = new String(ch, start, length);
            if(!chars.trim().isEmpty())System.err.println("CHARS: " + chars);
        }
    }
```

Handler

```
ByteArrayInputStream bis = new ByteArrayInputStream(xml.getBytes());
InputSource is = new InputSource(bis);
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
UserHandler userhandler = new UserHandler();saxParser.parse(is, userhandler);
```

DZIĘKUJEMY ZA UWAGE