Język XML

Czym jest XML (Extensible Markup Language)?

- Język znaczników służący do wymiany danych w formacie strukturalnym
- Reprezentuje dane w formacie tekstowym
- Niezależny od platformy

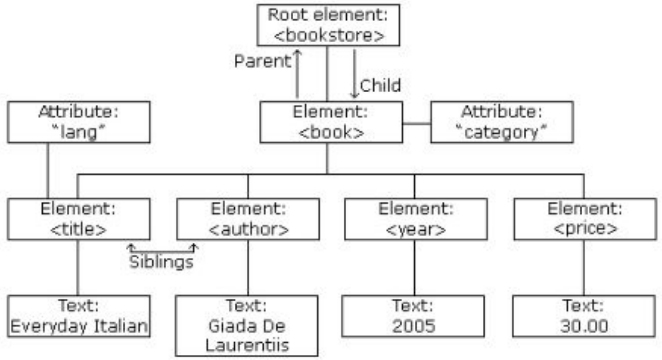
Struktura dokumentu XML

Dokument składa się z zagnieżdżonych elementów, zapisywanych jako znaczniki.

- Każdy dokument musi posiadać wyłącznie jeden element główny,
- elementy mogą zawierać treść bądź kolejne elementy
- elementy mogą posiadać atrybuty

Dokument XML w postaci drzewa - przykład

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```



Porównanie XML i JSON cz. 1

XML

- Extensible Markup Language
- Język znaczników (tagów)
- Mało czytelny
- Zorientowany na wymianę dokumentów
- Nie obsługuje tablic
- Bezpieczniejszy
- Wspiera różne kodowania

JSON

- JavaScript Object Notation
- Meta-język
- Łatwy do odczytania
- Zorientowany na wymianę danych
- Obsługuje tablice
- Mniej bezpieczny
- Wspiera tylko UTF-8

Porównanie XML i JSON cz. 2

XML

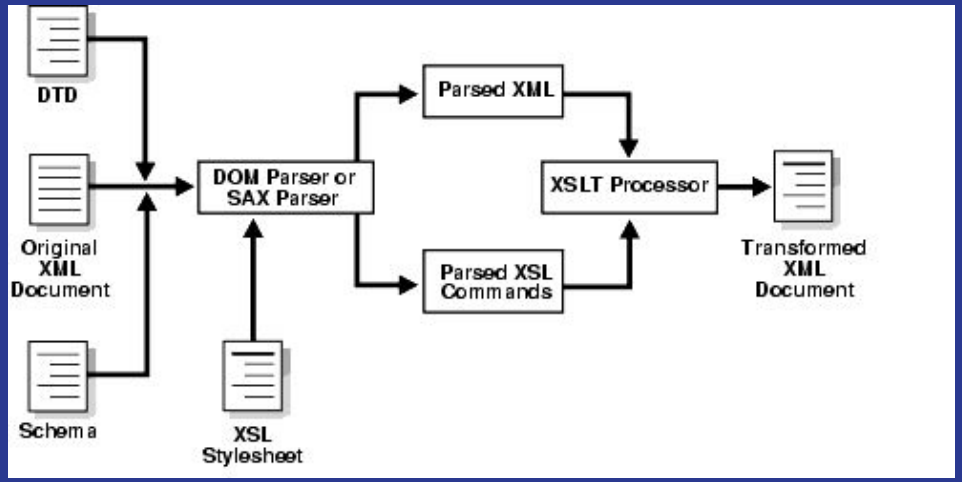
```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
            "employees" :  [
            {
                  "name" : "James Kirk",
                  "age" : 40,
            },
            {
                  "name" : "Jean-Luc Picard",
                  "age" : 45,
            },
            {
                  "name" : "Wesley Crusher",
                  "age" : 27,
            }
                              ]
      }
}
```

Porównanie XML, JSON i YAML

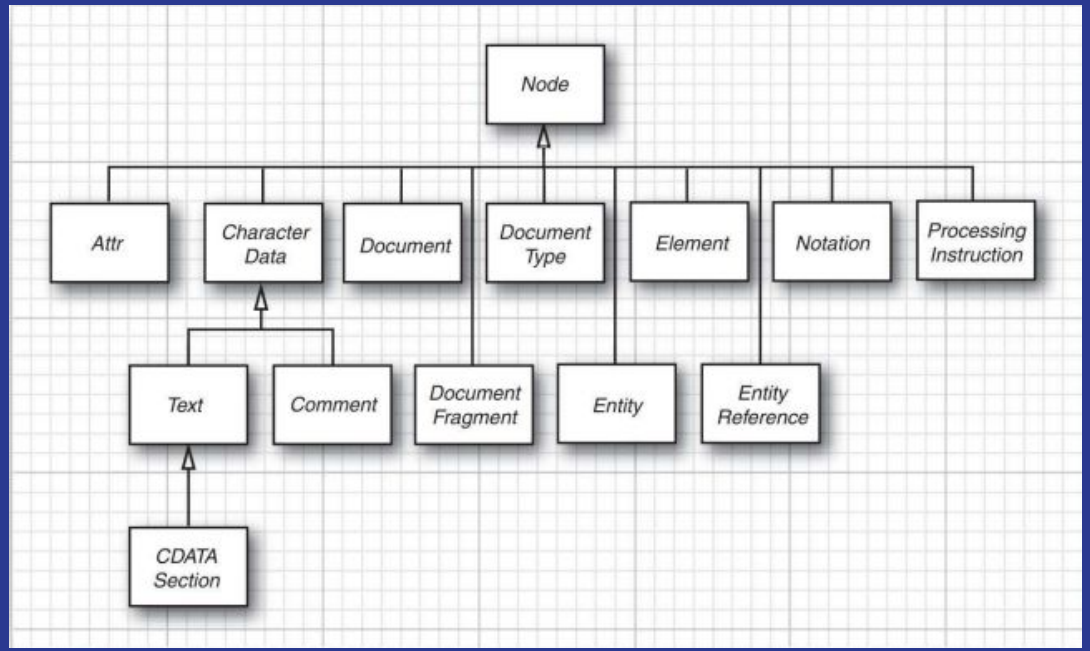
Parsery XML



Podział parserów:

- Tree parsers - parsery które wczytują dokument XML w strukturę drzewa, np. DOM (Document Object Model)
- Streaming parsers - parsery, które generują zdarzenia podczas czytania dokumentu XML, np. Simple API for XML (SAX)

Parsowanie XML w Javie



Struktury danych dokumentu XML

- Node - interfejs głównego typu danych dla całego DOM
- Document - reprezentacja struktury drzewa dokumentu XML
- Element - interfejs reprezentujący konkretny element w drzewie DOM

Walidacja DTD

Walidacja DTD (Document Type Definitions) zawiera reguły, które wyjaśniają, w jaki sposób należy utworzyć dokument poprzez określenie elementów podrzędnych oraz atrybutów dla każdego elementu.

```
<!ELEMENT font (name,size)>
```

Element *font* musi posiadać dwa podrzędne elementy o nazwie *name* oraz *size*

Reguła DTD w dokumencie XML

```
<?xml version="1.0"?>
<!DOCTYPE config [
   <!ELEMENT config . . .>
   more rules
   . . .
]>
<config>
   . . .
</config>
```

Walidacja XML

Schemat XML jest trochę bardziej złożony niż DTD.
Aby odwołać się do pliku schematu należy dodać odpowiednią konfigurację.

config.xsd jako walidator

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="config.xsd">
   . . .
</config>
```

Definiowanie własnych prostych typów

```
<xsd:simpleType name="StyleType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="PLAIN" />
      <xsd:enumeration value="BOLD" />
   </xsd:restriction>
</xsd:simpleType>
```

Specyfikacja typów elementów

```
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="size" type="xsd:int"/>
<xsd:element name="style" type="StyleType"/>
```

Porównanie DTD i XML

Porównanie definicji elementów podrzędnych za pomocą DTD oraz XML dla *Element*

DTD

```
<!ELEMENT font (name,size)>
```

XML

```
<xsd:element name="font">
   <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="size" type="xsd:int"/>
   </xsd:sequence>
</xsd:element>
```

XPath

XPath to w tłumaczeniu XML Path Language. Jest to język służący do opisu ścieżek XML
- wskazywaniu elementów, atrybutów, lub całych fragmentów dokumentu XML.

```
<?xml version="1.0" encoding="utf-8"?>
<rysunek>
<kwadrat id=”1”>
 <kolor value=”red”/>
 </kwadrat>
 <grupa id=”2”>
 <kolo id=”4”>
 <kolor value=”green”/>
 </kolo>
 <kwadrat id=”5”>
 <kolor><red>5</red><green>100</green><blue>50</blue></kolor>
 </kwadrat>
 <grupa id=”3”/>
 </grupa>
</rysunek>
```

/rysunek/kwadrat – odnosi się do wszystkich elementów „kwadrat” zagnieżdżonych bezpośrednio pod elementem „rysunek”

/rysunek//kwadrat - odnosi się do wszystkich elementów „kwadrat” zagnieżdżonych pod elementem „rysunek”

//grupa/kwadrat – odnosi się do wszystkich kwadratów, które są dziećmi jakiejś grupy, która nie musi być korzeniem dokumentu

//kwadrat[@id='1'] – odnosi się do kwadratu o wartości atrybutu id równej 1

//kolor/red/text() - tekst elementu red, który jest bezpośrednim potomkiem dowolnego elementu kolor

//grupa | //kwadrat – dowolna grupa, bądź dowolny kwadrat

XPath - przykład Java

XML

```
<widgets>
<widget>
<manufacturer/>
<dimensions/>
</widget>
</widgets>
```

Pobranie elementu <widget>

```
// parse the XML as a W3C Document
DocumentBuilder builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document document = builder.parse(new File("/widgets.xml"));

XPath xpath = XPathFactory.newInstance().newXPath();
String expression = "/widgets/widget";
Node widgetNode = (Node) xpath.evaluate(expression, document, XPathConstants.NODE);
```

Pobranie elementu <manufacturer> na podstawie elementu <widget>

```
XPath xpath = XPathFactory.newInstance().newXPath();
String expression = "manufacturer";
Node manufacturerNode = (Node) xpath.evaluate(expression, widgetNode, XPathConstants.NODE);
```

XML Namespaces

Przestrzeń nazw XML służy do uniknięcia konfliktu nazw elementów w dokumencie XML. Pozwalają na stosowanie wielu języków opartych na XML-u w jednym pliku XML.

```
<parent xmlns:prefix1="http://Namespace-name-URI">
   <child xmlns:prefix2="http://Namespace-name-URI">
      ...
   </child>
 </parent>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cont:contact xmlns:cont="http://sssit.org/contact-us">
   <cont:name>Vimal Jaiswal</cont:name>
   <cont:company>SSSIT.org</cont:company>
   <cont:phone>(0120) 425-6464</cont:phone>
</cont:contact>
```

Streaming parsers

Służą do parsowania dużych plików XML, które gdyby były sparsowane w całości mogłyby znacząco spowolnić aplikację. Dwoma podstawowymi parserami tego typu są SAX i StAX.

SAX - parser ten podczas przetwarzania XML emituje eventy o rozpoczęciu i zakończeniu tagów/dokumentu oraz zawartości tagów. Dla tego parsera musimy zdefiniować *Handler* który eventy te będzie otrzymywał.

StAX - jest to parser typu *pull*. Oznacza to, iż dokument XML parsowany jest krok po kroku, ale tylko na żądanie. Możemy dzięki temu iterować się poprzez dokument XML używając pętli poprzez wywołanie na parserze funkcji *next*.

Generowanie XML

```
Document doc = builder.newDocument();

Element rootElement =
doc.createElement(rootName);

Element childElement =
doc.createElement(childName);

Text textNode =
doc.createTextNode(textContents);


doc.appendChild(rootElement);

rootElement.appendChild(childElement);

childElement.appendChild(textNode);




rootElement.setAttribute(name, value);
```

Utworzenie pustego dokumentu i głównych elementów

Przypisanie głównego elementu do dokumentu a następnie węzłów potomnych do rodziców

Określenie atrybutów do elementu

Generowanie XML z namespace

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
builder = factory.newDocumentBuilder();

String namespace = "http://www.w3.org/2000/svg";

Element rootElement =
doc.createElementNS(namespace, "svg");

rootElement.setAttributeNS(namespace,
qualifiedName, value);
```

Zasadnicze różnice:

- Ustawienie opcji setNameSpaceAware(true)
- createElement -> createElementNS
- setAttribute -> setAttributeNS

Zapis XML

Transformacje XSLT API
(Extensible Stylesheet Language Transformations)

```
// construct the do-nothing transformation
Transformer t = TransformerFactory.newInstance().newTransformer();
// set output properties to get a DOCTYPE node
t.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM, systemIdentifier);
t.setOutputProperty(OutputKeys.DOCTYPE_PUBLIC, publicIdentifier);
// set indentation
t.setOutputProperty(OutputKeys.INDENT, "yes");
t.setOutputProperty(OutputKeys.METHOD, "xml");
t.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");
// apply the do-nothing transformation and send the output to a file
t.transform(new DOMSource(doc), new StreamResult(new FileOutputStream(file)));
```

Interfejs LSSerializer

```
DOMImplementation impl = doc.getImplementation();

var implLS = (DOMImplementationLS)
impl.getFeature("LS", "3.0");

LSSerializer ser = implLS.createLSSerializer();

//formatowanie (spacje, łamanie linii)

ser.getDomConfig().setParameter("format-pretty-print
", true);

//zapis do String

String str = ser.writeToString(doc);

//Zapis do pliku

LSOutput out = implLS.createLSOutput();

out.setEncoding("UTF-8");

out.setByteStream(Files.newOutputStream(path));

ser.write(doc, out);
```

Zapis XML - StAX API

```
//Inicializacja

XMLOutputFactory factory = XMLOutputFactory.newInstance();
XMLStreamWriter writer = factory.createXMLStreamWriter(out);


//Inicializacja elementów

writer.writeStartDocument() //Nagłówek
writer.writeStartElement(name);//Rozpocznij element
writer.writeAttribute(name, value); //Dodawanie atrybutów
writer.writeCharacters(text); //Tekst
writer.writeEndElement(); //Zakończ element


//Element bez węzłów potomnych (np. <img . . ./>)

writer.writeEmptyElement(name);


//Zamknięcie dokumentu (Zamyka wszystkie otwarte elementy)

writer.writeEndDocument();

Następnie należy manualnie zamknąć XMLStreamWriter (nie
rozszerza interfejsu AutoCloseable)
```

Jak w przypadku używania podejścia DOM/XSLT nie musimy martwić się o znaki końcowe w wartościach atrybutów.

Należy jednak wziąć pod uwagę możliwość stworzenia zniekształconego pliku XML, np. dokumentu z wieloma węzłami głównymi.

Kolejną z rzeczy do rozważenia jest fakt iż StAX API w obecnej wersji nie obsługuje tworzenia danych wyjściowych z wcięciami czy łamaniem linii.

Przykład

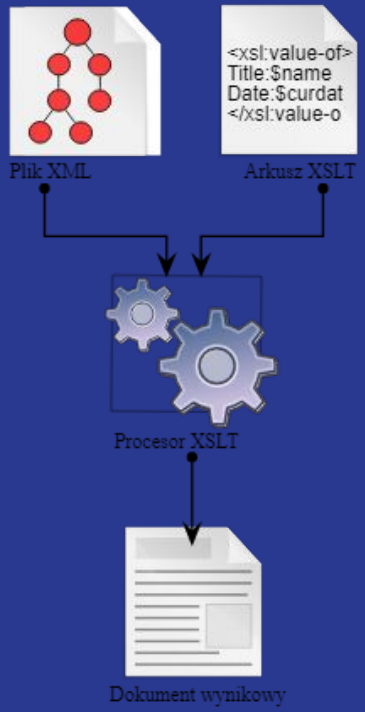
```
//Rozpoczęcie dokumentu
public static void main(String[] args) throws Exception
{
Document doc = newDrawing(600, 400);
writeDocument(doc, "drawing1.svg");
writeNewDrawing(600, 400, "drawing2.svg");
}

//Utworzenie elementów
public static Document newDrawing
(int drawingWidth, int drawingHeight)
throws ParserConfigurationException
{
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
DocumentBuilder builder = factory.newDocumentBuilder();;
var namespace = " http://www.w3.org/2000/svg ";
Document doc = builder.newDocument();
Element svgElement = doc.createElementNS(namespace, "svg");
doc.appendChild(svgElement);
svgElement.setAttribute("width", "" + drawingWidth);
svgElement.setAttribute("height", "" + drawingHeight);
int width = generator.nextInt(drawingWidth);
int height = generator.nextInt(drawingHeight);
Element rectElement = doc.createElementNS(namespace, "rect");
rectElement.setAttribute("width", "" + width);
rectElement.setAttribute("height", "" + height);
rectElement.setAttribute("fill",
String.format("#%02x%02x%02x", r, g, b));
svgElement.appendChild(rectElement);
return doc;
}
```

```
//Zapis za pomocą transformacji
public static void writeDocument(Document doc, String filename)
throws TransformerException, IOException {
Transformer t = TransformerFactory.newInstance().newTransformer();
t.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM,URI);
t.setOutputProperty(OutputKeys.DOCTYPE_PUBLIC, "-//W3C//DTD SVG
20000802//EN");
t.setOutputProperty(OutputKeys.INDENT, "yes");
t.setOutputProperty(OutputKeys.METHOD, "xml");
t.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");
t.transform(new DOMSource(doc), new StreamResult(
Files.newOutputStream(Paths.get(filename)))); }

//Zapis za pomocą StAX
public static void writeNewDrawing(int drawingWidth, int
drawingHeight,
String filename) throws XMLStreamException, IOException {
XMLOutputFactory factory = XMLOutputFactory.newInstance();
XMLStreamWriter writer = factory.createXMLStreamWriter(
Files.newOutputStream(Paths.get(filename)));
writer.writeStartDocument();
writer.writeDTD("ENTER_DOCTYPE”);
writer.writeStartElement("svg");
writer.writeDefaultNamespace(" http://www.w3.org/2000/svg ");
writer.writeAttribute("width", "" + drawingWidth);
writer.writeAttribute("height", "" + drawingHeight);
int width = generator.nextInt(drawingWidth - x);
int height = generator.nextInt(drawingHeight - y);
writer.writeEmptyElement("rect");
writer.writeAttribute("x", "" + x);
writer.writeAttribute("y", "" + y);
writer.writeAttribute("width", "" + width);
writer.writeAttribute("height", "" + height);
writer.writeAttribute("fill", String.format("#%02x%02x%02x", r, g,
b));
writer.writeEndDocument(); // closes svg element }
```

XSL Transformations



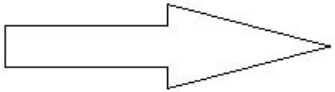
Transformacje XSL to mechanizm pozwalający na zdefiniowanie zasad transformacji między dokumentami XML a innymi formatami np. XHTML czy txt.

By dokonać takiej translacji musimy podać do procesora XSL arkusz stylów.

```
<staff>
   <employee>
      <name>Carl Cracker</name>
      <salary>75000.0</salary>
      <hiredate year="1987" month="12" day="15"/>
   </employee>
   <employee>
      <name>Harry Hacker</name>
      <salary>50000.0</salary>
      <hiredate year="1989" month="10" day="1"/>
   </employee>
   <employee>
      <name>Tony Tester</name>
      <salary>40000.0</salary>
      <hiredate year="1990" month="3" day="15"/>
   </employee>
</staff>
```

Przykładowa reguła arkusza stylów

```
<xsl:template match="/staff/employee/name">
   <td><xsl:apply-templates/></td>
</xsl:template>
```



```
<table border="1">
<tr>
<td>Carl Cracker</td><td>$75000.0</td><td>1987-12-15</td>
</tr>
<tr>
<td>Harry Hacker</td><td>$50000.0</td><td>1989-10-1</td>
</tr>
```

Dziękujemy za uwagę

Wykonali:

Grzegorz Podwika
Dominik Pepaś
Michał Mamla
Sebastian Smulski
Mariusz Morawski
Patryk Paluch
Dawid Aleksandrowicz