XML API-k

dr. Paller Gábor

XML API-k

- Gyakorlati feladatokban az XML technológiákat hagyományos programozási technológiákkal integrálva használjuk.
- Az XML független a programozási nyelvektől (ennek köszönhető népszerűsége), ezért ezek az API-k sok esetben nyelvfüggetlen, ill. a népszerű programozási nyelvekre leképezett változatokban léteznek.
- A mi példáink Jáva-alapúak.
- Következő API-kat nézzük át:
  - XML feldolgozás
    - SAX
    - DOM (W3C)
    - StAX (JSR 173)
  - XPath kiértékelés - Jáva ipari szabvány
  - XSLT feldolgozás - Jáva ipari szabvány, TrAX

SAX

- Simple API for XML processing.
  - Az első API XML feldolgozókhoz
  - Megelőzte a DOM-ot, amelyet korábban kezdtek, de a nehézkes szabványosítási szervezet miatt később fejeztek be.
  - Igazi garázsmunka eredménye, David Megginson javasolta az API működési módját és 4 hónap alatt fejezték be az XML-DEV levelezőlista tagjainak segítségével. A SAX1 1998 májusában lett kész.
  - Jelenleg a SAX2 az aktuális változat.
  - Számos parser támogatja, pl. a standard JDK-ban van SAX parser.
  - Otthona: www.saxproject.org

SAX programozási modell

- A SAX eseményalapú.
  - Az alkalmazás közli az XML feldolgozóval, milyen csomópontok érdeklik. Ehhez szűrőket definiál és eseménykezelőket regisztrál.
  - Az XML feldolgozó sorba veszi a csomópontokat (eközben esetleg validálást végez).
  - Ha az éppen feldolgozott csomópont illeszkedik a szűrőre, az alkalmazás eseményhívást kap.
- Előnyök:
  - Egyszerű XML parser (bár a nagy, sémavalidáló XML parser-ek bonyolultak tudnak lenni).
  - Minimális memóriaigény - a teljes XML dokumentumot sose kell tárolni a memóriában.
  - Inkrementális feldolgozás egyszerű (az XML dokumentumot a beérkezése közben dolgozzuk fel).
- Hátrányok:
  - Nehezebben áttekinthető alkalmazáskód (egy állapotgépet kell implementálni az alkalmazás oldalán).
  - Eredeti, "push" változatában az alkalmazás nem befolyásolhatja az XML feldolgozás folyamatát, passzív résztvevő (ha eseményt kap, kezeli).

Egyszerű SAX alkalmazás

```
import java.io.FileReader;
import org.xml.sax.XMLReader;
import org.xml.sax.InputSource;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;

public class MySAXApp extends DefaultHandler {
  public static void main ( String args[] )
              throws Exception  {
    XMLReader xr = XMLReaderFactory.createXMLReader();
    MySAXApp handler = new MySAXApp();
    xr.setContentHandler( handler );
    xr.setErrorHandler( handler );
// az első hívási paraméter az XML fájl neve
    FileReader r = new FileReader(args[0]);
    xr.parse(new InputSource( r ));
  }
}
```

Egyszerű SAX alkalmazás, 2.

- Hívása:
  ```
  java MySAXApp sample.xml
  ```
  vagy
  ```
  java -Dorg.xml.sax.driver=org.apache.xerces.parsers.SAXParser
  MySAXApp sample.xml
  ```
  - Az org.xml.sax.driver property-t be lehet állítani, hogy az XMLReaderFactory megtalálja a használni kívánt implementációt.
  - Értéke az implementációtól függ, pl. Xerces: org.apache.xerces.parsers.SAXParser, JDK 1.6 default: com.sun.org.apache.xerces.internal.parsers.SAXParser
  - A JDK 1.6-ban van alapértelmezett beállítás, ami a JDK-val adott parser implementációra mutat, így a property-vel általában nem kell törődni.
  - Lehetséges a parser implementáció közvetlen instanciálása is, pl:
    ```
    XMLReader xr = new
    org.apache.xerces.parsers.SAXParser();
    ```
    de ez ellenjavalt.

Eseménykezelők

- Ez az alkalmazás látszólag nem csinál semmit. Oka, hogy saját magát állítja be eseménykezelőnek (setContentHandler, setErrorHandler). Ez működik, mert a szülőosztálya az org.xml.sax.helpers.DefaultHandler, amely azonban csak üres eseménykezelőket tartalmaz, ezért nem történik látszólag semmi. Valójában az XML feldolgozó elemzi a dokumentumot, bár ennek hatása nincsen.
- Eseménykezelőket kell létrehoznunk, felüldefiniálva a DefaultHandler üres eseménykezelőit.
- ```
  public void startElement (String uri, String name,  String
  qName, Attributes atts) {
          if ("".equals (uri))
           System.out.println("Start element: " + qName);
         else
           System.out.println("Start element: {" + uri + "}" +
  name +
                    " ("+qName+")" );
  }

  public void endElement (String uri, String name, String qName)
  {
      if ("".equals (uri))
           System.out.println("End element: " + qName);
      else
           System.out.println("End element:   {" + uri + "}" +
  name
                              + " ("+qName+")" )
  }
  ```
  ```

Eseménykezelők, 2.

- Jól ismert "books" példánkat felhasználva
- ```
  Start element: books
  Start element: book
  Start element: title
  End element: title
  ...
  ```
- Névterületekkel megvariálva:
  ```
  <b:books xmlns:b="http://www.books.com/ns">
    <b:book id="id1">
      <b:title>My first XML book</b:title>
  ...
  ```
- Eredmény:
  ```
  Start element: {http://www.books.com/ns}books (b:books)
  Start element: {http://www.books.com/ns}book (b:book)
  Start element: {http://www.books.com/ns}title (b:title)
  End element:   {http://www.books.com/ns}title (b:title)
  ...
  ```
  ```

Eseménykezelők, 3.

```
public void characters( char ch[], int start, int length) {
    System.out.print("Characters:    \"");
    for (int i = start; i < start + length; i++) {
      switch (ch[i]) {
        case '\\':
                System.out.print("\\\\");
                break;
        case '"':
                System.out.print("\\\"");
                break;
        case '\n':
                System.out.print("\\n");
                break;
        case '\r':
                System.out.print("\\r");
                break;
        case '\t':
                System.out.print("\\t");
                break;
        default:
                System.out.print(ch[i]);
                break;
      }
    }
    System.out.println( "\"" );
  }
```

Eseménykezelők, 4.

- Eredmény:

```
Start element: {http://www.books.com/ns}books (b:books)
Characters:    "\n  "
Start element: {http://www.books.com/ns}book (b:book)
Characters:    "\n    "
Start element: {http://www.books.com/ns}title (b:title)
Characters:    "My first XML book"
End element:   {http://www.books.com/ns}title (b:title)
Characters:    "\n    "
...
```

Eseménykezelők, 5.

- ```
  public void startPrefixMapping (String prefix, String
  uri) {
     System.out.println( "Start prefix mapping:
  "+prefix+"="+uri );
    }

    public void endPrefixMapping (String prefix) {
      System.out.println( "End prefix mapping: "+prefix
  );
    }
  ```
- Eredmény:

  ```
  Start prefix mapping: b=http://www.books.com/ns
  Start element: {http://www.books.com/ns}books (b:books)
  Characters:    "\n  "
  ```
  …
  ```

Attribútumok

```
public void startElement (String uri, String name,
                          String qName, Attributes atts) {
// Kinyomtatja az elem nevét, már láttuk, innen törölve
// Most kinyomtatja az elem attribútumait (ha vannak)
    for (int i = 0; i < atts.getLength(); i++) {
      String attruri = atts.getURI(i);
      String localName = atts.getLocalName(i);
      String value = atts.getValue(i);
      if( "".equals( attruri ) )
        System.out.print( "Attribute name: " +  localName );
      else
        System.out.print( "Attribute {"+attruri+"}"+ localName
);
      System.out.println( " value: "+value );
    }
  }
```

- Eredmény:

```
Start element: books
Start element: book
Attribute name: id value: id1
...
```

Property, feature

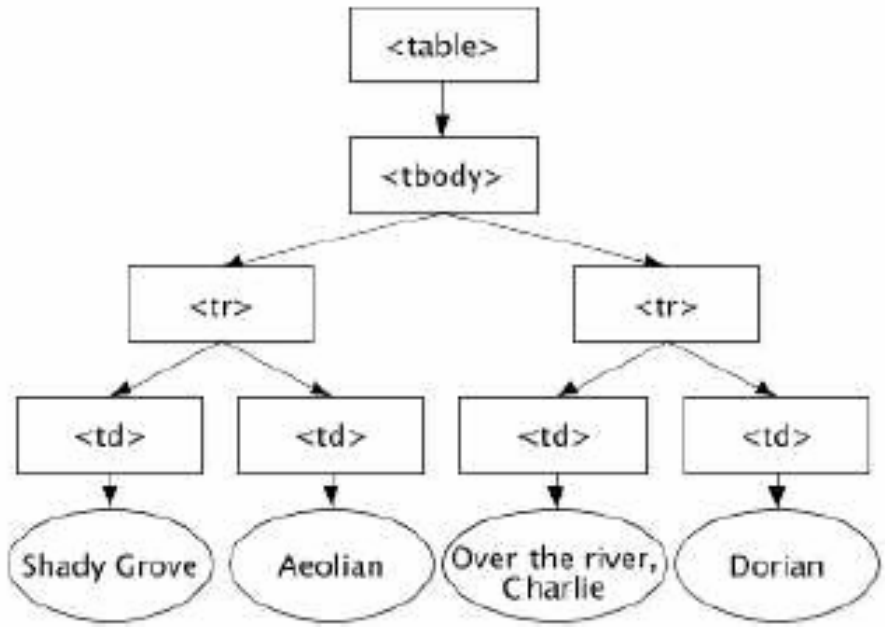
- Az XML elemző állapotát és szolgáltatásait a property ill. a feature absztrakción keresztül befolyásolhatjuk.
- Az XMLReader getProperty, setProperty, getFeature, setFeature metódusain keresztül működnek.
- Feature: az XML elemző egy opcionális szolgáltatása, amit ki vagy be lehet kapcsolni.
  - Bináris értékük van (on/off)
  - URI azonosítja őket
    - Pl. http://xml.org/sax/features/validation: validálás be vagy kikapcsolása.
  - xr.setFeature(" http://xml.org/sax/features/validation", true);
  - Az elemzőnek nem kell ismernie (SAXNotRecognizedException) vagy nem kell támogatnia (SAXNotSupportedException)
- Property: csak olvasható vagy írható/olvasható adat az elemzés állapotáról.
  - Példa: http://xml.org/sax/properties/document-xml-version - csak olvasható, 1.0 vagy 1.1, startDocument() esemény után elérhető.
  - java.lang.Object érték - a property számára definiált típus (pl. a document-xml-version esetén String)

DOM

- Document Object Model, W3C szabvány.
- Első kiadás: 1999
- Motiváció: JavaScript kód és Java appletek összekötése.
- Nyelvfüggetlen specifikáció OMG IDL-ben.
- További specifikációk képezik le Jávára, C++-ra, JavaScript-re ...
- Mi csak a Jáva leképezéssel foglalkozunk.

Fa-alapú programozási modell

```
<table>
 <tbody>
  <tr>
   <td>Shady Grove</td>
   <td>Aeolian</td>
  </tr>
  <tr>
   <td>Over the River, Charlie</td>
   <td>Dorian</td>
  </tr>
 </tbody>
</table>
```



DOM elemző létrehozása

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import java.io.IOException;
import java.util.StringTokenizer;

public class DOMParser {
  public static void main( String args[] ) {
    try {
      DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
      DocumentBuilder parser = factory.newDocumentBuilder();
      Document d = parser.parse( args[0] );
    } catch (SAXException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    } catch (ParserConfigurationException e) {
      e.printStackTrace();
    }
  }
}
```

DOM elemző létrehozása

- Amit látunk, az nem része a DOM-nak, hanem a JDK része.
- DocumentBuilderFactory.newInstance - létrehoz egy DOM parser factory-t egy meglehetősen összetett konfigurációs folyamatot használva. A DocumentBuilderFactory osztályt rendszerproperty, JRE konfigurációs fájl, a classpath-en levő JAR fájlokban elhelyezett konfigurációs fájl adhatja meg és van alapértelmezett is.
- A DocumentBuilderFactory egyeden állíthatunk be feature-öket és kérdezhetünk le property-ket.
- A DocumentBuilderFactor.newDocumentBuilder készíti a tényleges parser egyedet (DocumentBuilder class).
- Ennek parse() metódusa építi fel a DOM fát. A parse() öt különböző paraméterrel létezik a forrásdokumentum elérési módja szerint. Ez lehet File, String (ami a forrásdokumentum URI-ját tartalmazza), InputSource (SAX számára kifejlesztett forrásdokumentum absztrakció), InputStream.
- Hiba esetén SAXException dobódik (!)
- A parse() hívás eredménye egy org.w3c.dom.Document egyed, ami a DOM fa legfelső csomópontja.

Node

- A DOM fa elemeinek alapja, ő maga egy interfész (ahogy az összes többi DOM elem is az).
- Leszármazottai: Attr, CDATASection, CharacterData, Comment, Document, DocumentFragment, DocumentType, Element, Entity, EntityReference, Notation, ProcessingInstruction, Text - gyakorlatilag az összes Infoset-ből ismert elem.
- A Node képességei:
  - Neve van a szokásos képességekkel: getNodeName, getLocalName, getNamespaceURI, getPrefix.
  - Típusa van: getNodeType (Node.ATTRIBUTE_NODE , Node.CDATA_SECTION_NODE, Node.COMMENT_NODE, Node. DOCUMENT_FRAGMENT_NODE, Node.DOCUMENT_NODE, Node.ELEMENT_NODE, Node.ENTITY_NODE, Node.ENTITY_REFERENCE_NODE, Node.NOTATION_NODE, Node.PROCESSING_INSTRUCTION_NODE, Node.TEXT_NODE).
  - Értéke van, amely csomópont-típustól függően értelmeződik: getNodeValue
  - Gyerekcsomópontjai vannak, amelyeket le lehet kérdezni: getChildNodes, getFirstChild, getLastChild, hasChildNodes
  - Szülő- és testvércsomópontjai vannak: getOwnerDocument, getNextSibling.
  - Attribútumai vannak: getAttributes, hasAttributes
  - Szöveges tartalma lekérdezhető: getTextContent (XPath-hoz hasonló működés)
  - Gyerekcsomópontjai módosíthatók: appendChild, removeChild, replaceChild, insertBefore

Document

- Node leszármazott
- Megszerezhető tőle és beállítható rajta az XML dokumentum verziója, standalone státusza, karakterkódolása, fellelhetőségének címe (URI).
- Meg lehet tőle szerezni a dokumentum legfelső szintű elemét, egy elemet adott ID-vel, az összes adott nevű elemet.
- nodeName: "#document"

Element

- Egy XML elemet reprezentál.
- Többi elemtől megkülönbözteti, hogy attribútumai vannak. Minden Node-nak van attribútum-elérése, de ez csak az Element-nél működik. Attribútumokat egy hashtable-szerű konstrukcióban lehet elérni (NamedNodeMap).
- nodeName: prefix:elem név

Document és Element példa

- ```
  ...
  Document d = parser.parse( args[0] );
  printChildElement( d.getDocumentElement(),0 );
  ...
  ```
- ```
  private static void printChildElement( Element e, int depth ) {
    System.out.println( "Element: "+e.getTagName()+", depth="+depth );
    System.out.print( "  " );
    NodeList nl = e.getChildNodes();
    for( int i = 0 ; i < nl.getLength() ; ++i ) {
      Node n = nl.item( i );
      if( n.getNodeType() == Node.ELEMENT_NODE )
         System.out.print( n.getNodeName()+" " );
    }
    System.out.println();
    for( int i = 0 ; i < nl.getLength() ; ++i ) {
      Node n = nl.item( i );
      if( n.getNodeType() == Node.ELEMENT_NODE )
        printChildElement( (Element)n,depth+1 );
    }
  }
  ```
- ```
  Element: b:books, depth=0
    b:book b:book
  Element: b:book, depth=1
    b:title b:published b:publisher b:author b:author b:abstract
  ...
  ```
  ```

Attr

- Egy attribútumot reprezentál.
- Az attribútumok elem csomópontokhoz vannak kötve és a Node.getAttributes ill. Element.getAttributeNode és rokonai segítségével kérdezhetők le. Mindezeknek removeXXX és setXXX párjuk is van.
- Előző programunk kibővítése:

```
...
NamedNodeMap nnm = e.getAttributes();
for( int i = 0 ; i < nnm.getLength() ; ++i ) {
   Node n = nnm.item( i );
   System.out.print(
     n.getNodeName()+"="+n.getNodeValue()+" " );
}
...
```

-
```
Element: b:books, depth=0
  b:book b:book
xmlns:b=http://www.books.com/ns
...
```

Text

```
for( int i = 0 ; i < nl.getLength() ; ++i ) {
    Node n = nl.item( i );
    if( n.getNodeType() == Node.TEXT_NODE )
       System.out.println( "TEXT: "+n.getNodeValue() );
    else
    if( n.getNodeType() == Node.ELEMENT_NODE )
    printChildElement( (Element)n,depth+1 );;
}
```

```
Element: title, depth=2

TEXT: My first XML book
TEXT:
```

További csomópont-típusok

- CDataSection, Comment, Entity, EntityReference, Notation, ProcessingInstruction.
- DocumentFragment - a Document "könnyűsúlyú" implementációja XML részletek tárolására. Ha beszúrják egy XML fába, a DocumentFragment nem szúródik be, csak a gyerekei.

StAX

- Streaming API for XML
- Több kísérleti API után a JSR-173-ban szabványosították (2004). 1.0-ás verziónál tart.
- A standard JDK 1.6-os tartalmaz ilyen parser-t is beépítve
- Streaming
  - Nem egyben dolgozza fel a dokumentumot, mint a DOM, hanem inkrementálisan, ahogy érkezik (ilyen a SAX is).
  - A StAX pull API szemben a SAX-szal, ami push API.
    - A push API-nál az elemzés megindítása után folyamatosan érkeznek az események, amíg az elemzés meg nem szakad.
    - A pull API-nál az alkalmazásnak magának kell kérnie a következő eseményt.
  - Előnyei:
    - Az alkalmazás vezérli a végrehajtást, nem a parser.
    - Egyszerre több dokumentum is elemezhető egy végrehajtási szállal.
    - Bármelyik elemzett XML dokumentum feldolgozása felfüggeszthető és később folytatható.
    - Mobil Jávára (J2ME) is adaptálható.

StAX kurzor API

```
import javax.xml.stream.*;
import java.io.*;
import javax.xml.stream.events.*;

public class StAXParser {
    public static void main( String args[] ) {
     try {
      XMLInputFactory xmlif = XMLInputFactory.newInstance();
      XMLStreamReader xmlr = xmlif.createXMLStreamReader( args[0],
         new FileInputStream( args[0] ));
       int eventType = xmlr.getEventType();
       System.out.println( getEventTypeString( eventType ) );
       while(xmlr.hasNext()) {
         eventType = xmlr.next();
         System.out.println( getEventTypeString( eventType ) );
       }
    } catch( IOException ex ) { ex.printStackTrace(); }
      catch( XMLStreamException ex ) { ex.printStackTrace(); }
  }

  private static String getEventTypeString(int eventType) {
    switch (eventType) {
        case XMLEvent.START_ELEMENT:            return "START_ELEMENT";
        case XMLEvent.END_ELEMENT:            return "END_ELEMENT";
        case XMLEvent.PROCESSING_INSTRUCTION: return "PROCESSING_INSTRUCTION";
        case XMLEvent.CHARACTERS:           return "CHARACTERS";
        case XMLEvent.COMMENT:            return "COMMENT";
        case XMLEvent.START_DOCUMENT:           return "START_DOCUMENT";
        case XMLEvent.END_DOCUMENT:           return "END_DOCUMENT";
        case XMLEvent.ENTITY_REFERENCE:           return "ENTITY_REFERENCE";
        case XMLEvent.ATTRIBUTE:            return "ATTRIBUTE";
        case XMLEvent.DTD:            return "DTD";
        case XMLEvent.CDATA:            return "CDATA";
        case XMLEvent.SPACE:            return "SPACE";
    }   return "UNKNOWN_EVENT_TYPE " + "," + eventType;
  }
```

StAX kurzor API, 2.

- A kurzor API az alacsonyabb szintű StAX API
- Eseményeket generál.
  - Egy esemény mint egy egész szám kerül az alkalmazáshoz.
  - Az esemény részleteit az olvasó parser (XMLStreamReader) metódusainak hívásával szerezhetjük meg. Példa:

```
if( eventType == XMLEvent.START_ELEMENT )
   System.out.print( " ("+xmlr.getName()+")" );
```

  - Az elemzés megkezdésekor a START_DOCUMENT esemény már betöltődött, tehát az első eseményt nem a next()-tel, hanem a getEventType()-pal hozzuk el. int eventType = **xmlr.getEventType();**
- Előző dia programjának kimenete a START_ELEMENT részleteinek kiírásával együtt:

```
START_DOCUMENT
DTD
START_ELEMENT (books)
SPACE
...
END_ELEMENT
END_DOCUMENT
```

StAX kurzor API, attribútumok

- Ha az esemény START_DOCUMENT, az XMLStreamReader attribútumokkal kapcsolatos metódusai meghívhatók (és csak akkor).
- ```
  if( eventType == XMLEvent.START_ELEMENT ) {
     System.out.print( " ("+xmlr.getName()+")" );
     for( int i = 0 ; i < xmlr.getAttributeCount() ; ++i ) {
         System.out.print( " "+
                         xmlr.getAttributeName( i )+
                         "="+
                         xmlr.getAttributeValue( i ) );
     }
  }
  ```
- Eredmény:

  ```
  ...
  START_ELEMENT (books)
  SPACE
  START_ELEMENT (book) id=id1
  ...
  ```
  ```

StAX kurzor API, írás

```
XMLOutputFactory xof = XMLOutputFactory.newInstance();
FileOutputStream fo = new FileOutputStream( fileName );
XMLStreamWriter xtw = xof.createXMLStreamWriter(
   new OutputStreamWriter( fo,"utf-8" ) );
xtw.writeComment(
                    "all elements here are explicitly in the
HTML namespace");
xtw.writeStartDocument("utf-8", "1.0");
xtw.setPrefix("html", "http://www.w3.org/TR/REC-html40");
xtw.writeStartElement("http://www.w3.org/TR/REC-html40",
"html");
xtw.writeNamespace("html", "http://www.w3.org/TR/REC-html40");
xtw.writeStartElement("http://www.w3.org/TR/REC-html40",
"head");
...
xtw.writeEndElement();
xtw.writeEndDocument();
xtw.flush();
xtw.close();
```

StAX kurzor API, írás, 2.

- Eredmény (sorvégi soremeléseket az olvashatóság kedvéért én írtam bele!)

```
<!--all elements here are explicitly in the HTML
namespace-->
<?xml version="1.0" encoding="utf-8"?>
<html:html xmlns:html="http://www.w3.org/TR/REC-
html40">
<html:head>
...
</html:body>
</html:html>
```

StAX iterátor API

- Variációja a kurzor API-nak.
- Itt is eseményekről van szó, de ezek az események objektumok.
  - Előny: ezek az események szűrőláncokon keresztül terjeszthetők elegánsan modularizálva a programot (pl. egy modul csak az XML dokumentum egy részével foglalkozik és az előtte levő modul csak a kérdéses rész eseményeit terjeszti hozzá).
  - Hátrány: az iterátor API használata megterhelőbb memóriafogyasztás szempontjából.

StAX iterátor API

```
XMLInputFactory factory =
XMLInputFactory.newInstance();
XMLEventReader r = factory.createXMLEventReader(
 filename, new FileInputStream(filename));
while (r.hasNext()) {
    XMLEvent e = r.nextEvent();
    System.out.println(e.toString());
}
```

TrAX

- Transformations API for XML (a JAXP API része).
- Az XSLT támogatására készült.
- Kulcsfogalmai:
  - TransformerFactory (javax.xml.transform) - ő tudja a transzformáló motort legyártani.
  - Transformer (javax.xml.transform) - maga a transzformáló motor, amely egy konkrét XSLT stíluslapot jelképez.
- Egy Transformer egyed egy bemeneti dokumentumot kimeneti dokumentummá alakít át.
- Az API olyan, hogy lehetővé teszi a hatékony implementációt, pl. a Transformer egyed előfordíthatja az XSLT stíluslapot és az előfordított stíluslapot tárolhatja. Ezért érdemes a Transformer egyedeket tárolni, ha többször is alkalmazni akarjuk ugyanazt a stíluslapot.

TrAX, 2.

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

Source xmlSource = new StreamSource(xmlFile);
Source xsltSource = new StreamSource(xsltFile);
Result result = new StreamResult(outFile);
TransformerFactory transFact =
      TransformerFactory.newInstance();
Transformer trans =
   transFact.newTransformer(xsltSource);
trans.transform(xmlSource, result);
```

XPath API

- A Jáva Platform 5.0 változata óta van jelen.
- XPath kifejezések absztrakciója: XPathExpression (javax.xml.xpath). Ez lehetővé teszi az előfordítás-alapú implementációt is.
- Az API (de nem feltétlenül az alatta levő implementáció!) lehetővé teszi tetszőleges adatmodellt használó XML reprezentáció szűrését
  - XPathExpression.evaluate(InputSource source) - XPath kifejezés kiértékelése a dokumentumra, ha az streaming formában érkezik.
  - XPathExpression.evaluate(Object item) - XPath kifejezés kiértékelése egy DOM dokument egy csomópontjának kontextusában.

- Ezen felül az eredmény típusa megadható és akkor az evaluate() az XPath típusnak megfelelő Jáva objektum formájában adja vissza az eredményt (csomóponthalmaz: org.w3c.dom.Node, szám: java.lang.Double, boolean: java.lang.Boolean, string: java.lang.String).
- Ha az eredmény nem felel meg az evaluate-nek megadott eredménytípusnak, az hiba.
- Eredménytípus megadása: XPathConstants konstansok (XPathConstants.BOOLEAN, XPathConstants.NUMBER, XPathConstants.STRING, XPathConstants.NODESET).

XPath API, példa

```
// DOM bemenetre
DocumentBuilderFactory domFactory =
DocumentBuilderFactory.newInstance();
domFactory.setNamespaceAware(true);
DocumentBuilder builder =
domFactory.newDocumentBuilder();
Document doc = builder.parse( xmlFile );

XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
XPathExpression expr  =
   xpath.compile( "sum(/doc/num)" );

Object result = expr.evaluate(doc,
   XPathConstants.NUMBER );

System.out.println( ((Double)result).toString() );
```