Rozhraní pro práci s XML (SAX, DOM, pull)

Obsah

Základní pojmy

Cílem rozhraní je

- poskytnout jednoduchý standardizovaný přístup ke XML datům
- "napojit" analyzátor (parser) na aplikaci a aplikace navzájem
- odstínit aplikaci od fyzické struktury dokumentu (entity)
- zefektivnit zpracování XML dat

Hlavní typy rozhraní pro zpracování XML dat:

- Stromově orientovaná rozhraní (Tree-based API)
- Rozhraní založená na událostech (Event-based API)
- Rozhraní založená na "vytahování" událostí/prvků z dokumentu (Pull API)

Stromově orientovaná rozhraní (Tree-based API)

Mapují XML dokument na stromovou strukturu v paměti

- dovolují libovolně procházet ("traverse") vzniklý strom;
- nejznámější je *Document Object Model* (DOM) konsorcia W3C, viz http://www.w3.org/DOM [http://www.w3.org/DOM/]

Modely specifické pro konkrétní prostředí

- pro Javu: JDOM - http://jdom.org
- pro Javu: dom4j WIKIPEDIA The Free Encyclopedia [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=dom4j] - http://dom4j.org
- pro Javu: XOM - http://www.xom.nu
- pro Python: 4Suite - http://4suite.org

- pro PHP: SimpleXML - http://www.php.net/simplexml

Rozhraní založená na událostech (Event-based API)

Při analýze ("parsing") dokumentu "vysílají" zpracovávající aplikaci *sled událostí*.

- technicky realizováno jako *volání metod* ("callback")
- aplikace poskytuje *handlery*, které volání zachytávají a zpracovávají
- událostmi řízená rozhraní jsou "nižší úrovně" než stromová, protože
- pro aplikaci zůstává "více práce"
- jsou však úspornější na paměť (většinou i čas), samotná analýza totiž nevytváří žádné „trvalé“ objekty

Událostmi je např.:

- začátek a konec dokumentu (start document, end document)
- začátek a konec elementu (start element, end element) - předá současně i atributy
- instrukce pro zpracování (processing instruction)
- komentář (comment)
- odkaz na entitu (entity reference)
- Nejznámějším takovým rozhraním je SAX http://www.saxproject.org

SAX - příklad analýzy dokumentu

```
<?xml version="1.0"?>
<doc>
   <para>Hello, world!</para>
   <!-- that's all folks -->
   <hr/>
</doc>
```

vyprodukuje při analýze (parsingu) sled událostí:

```
start document
start element: doc {seznam atributů: prázdný}
start element: para {seznam atributů: prázdný}
characters: Hello, world!
end element: para
comment:  that's all folks
start element: hr
end element: hr
end element: doc
end document
```

Kdy zvolit událostmi řízené rozhraní?

- O co snazší pro autora parseru, o to náročnější pro aplikačního programátora...
- Aplikace si musí (někdy složitě) pamatovat stav analýzy, nemá nikdy "celý dokument pohromadě".
- Na úlohy, které lze řešit "lokálně", bez kontextu celého dokumentu, je to vhodné rozhraní.
- Obvykle poskytuje nejrychlejší možné zpracování.
- Aplikační nepříjemnosti lze obejít použitím nadstaveb, např. Streaming Transformations for XML (STX) [http://stx.sourceforge.net]

Vlastnosti (features) nastavitelné pro analýzu - parsing

Chování parseru produkujícího SAX události je možné ovlivnit nastavením tzv. *features* a *properties*.

- *Vlastnosti* (*features*) nastavitelné pro analýzu (parsing) http://www.saxproject.org/?selected=get-set
- Blíže k jednotlivým properties a features v článku Use properties and features in SAX parsers [???] (IBM DeveloperWorks/XML).

SAX filtry

SAX rozhraní nabízí možnost napsat třídu jako tzv. SAX filtr (přesněji implementaci rozhraní org.xml.sax.XMLFilter).

Objekt takové třídy na jedné straně události přijímá, zpracuje je a posílá dále.

Další informace k filtrování událostí naleznete např. v článku Change the events output by a SAX stream [http://www.ibm.com/developerworks/xml/library/x-tipsaxfilter/] (IBM DeveloperWorks/XML).

Další odkazy k SAX

- "Přímo od zdroje" http://www.saxproject.org
- SAX Tutorial k JAXP - http://java.sun.com/webservices/docs/ea1/tutorial/doc/JAXPSAX.html

Rozhraní založená na technice "pull"

Rozhraní založená na technice "pull"

- Aplikace "nečeká na události", ale "vytahuje si" příslušná data ze vstupního parsovaného souboru.
- Využíváme tam, kde "víme, co ve zdroji očekávat" a "postupně si to bereme"
- ... vlastně opak API řízeného událostmi.
- Z hlediska aplikačního programátora velmi pohodlné, ale implementace bývají o něco pomalejší než klasická "push" událostmi řízená rozhraní.
- Pro Javu existuje *XML-PULL parser API* - viz Common API for XML Pull Parsing [http://www.xmlpull.org/] a také
- nově vyvíjené rozhraní Streaming API for XML (StAX) [http://www.jcp.org/en/jsr/detail?id=173] vznikající "shora i zdola" jako produkt JCP (Java Community Process).

Streaming API for XML (StAX)

Toto API se později může stát standardní součástí javového prostředí pro práci s XML, tzv. JAXP.

Nabízí dva přístupy k "pull" zpracování:

- přístup k "vytahovaným" událostem prostřednictvím iterátoru - pohodlnější
- nízkoúrovňový přístup přes tzv. kurzor - rychlejší

StAX - příklad s iterátorem

Příklad 1. StAX - přístup iterátorem

```
import java.io.*;
import java.util.Iterator;
import javax.xml.namespace.QName;
import javax.xml.stream.*;
import javax.xml.stream.events.*;
public class ParseByEvent {
```

```
   public static void main(String[] args)
                        throws FileNotFoundException, XMLStreamException {
      // Use the reference implementation for the XML input factory
      System.setProperty("javax.xml.stream.XMLInputFactory",
                                 "com.bea.xml.stream.MXParserFactory");
      // Create the XML input factory
      XMLInputFactory factory = XMLInputFactory.newInstance();
      // Create the XML event reader
      FileReader reader = new FileReader("somefile.xml");
      XMLEventReader r =
         factory.createXMLEventReader(reader);
      // Loop over XML input stream and process events
      while(r.hasNext()) {
         XMLEvent e = r.next();
         processEvent(e);
      }
   }
/**
 * Process a single XML event
 * @param e - the event to be processed
 */
  private static void processEvent(XMLEvent e) {
     if (e.isStartElement()) {
        QName qname = ((StartElement) e).getName();
        String namespaceURI = qname.getNamespaceURI();
        String localName = qname.getLocalPart();
        Iterator iter = ((StartElement) e).getAttributes();
        while (iter.hasNext()) {
           Attribute attr = (Attribute) iter.next();
           QName attributeName = attr.getName();
           String attributeValue = attr.getValue();
        }
     }
     if (e.isEndElement()) {
        QName qname = ((EndElement) e).getName();
     }
     if (e.isCharacters()) {
        String text = ((Characters) e).getData();
     }
     if (e.isStartDocument()) {
        String version = ((StartDocument) e).getVersion();
        String encoding = ((StartDocument) e).getCharacterEncodingScheme();
        boolean isStandAlone = ((StartDocument) e).isStandalone();
     }
  }
}
```

Poznámka

příklad převzat z Tip: Use XML streaming parsers [http://www.ibm.com/developerworks/xml/library/x-tipstx] (IBM DeveloperWorks, sekce XML).

StAX - příklad s kurzorem

Příklad 2. StAX - přístup kurzorem

```
import java.io.*;
import javax.xml.stream.*;
public class ParseByIterator {
   public static void main(String[] args)
                        throws FileNotFoundException, XMLStreamException {
      // Use reference implementation
      System.setProperty(
         "javax.xml.stream.XMLInputFactory",
         "com.bea.xml.stream.MXParserFactory");
      // Create an input factory
      XMLInputFactory xmlif = XMLInputFactory.newInstance();
      // Create an XML stream reader
      XMLStreamReader xmlr =
         xmlif.createXMLStreamReader(new FileReader("somefile.xml"));
      // Loop over XML input stream and process events
      while (xmlr.hasNext()) {
         processEvent(xmlr);
         xmlr.next();
      }
   }
   /**
    * Process a single event
    * @param xmlr - the XML stream reader
    */
   private static void processEvent(XMLStreamReader xmlr) {
      switch (xmlr.getEventType()) {
         case XMLStreamConstants.START_ELEMENT :
            processName(xmlr);
            processAttributes(xmlr);
            break;
         case XMLStreamConstants.END_ELEMENT :
            processName(xmlr);
            break;
         case XMLStreamConstants.SPACE :
         case XMLStreamConstants.CHARACTERS :
```

```
                int start = xmlr.getTextStart();
                int length = xmlr.getTextLength();
                String text =
                   new String(xmlr.getTextCharacters(), start, length);
                break;
             case XMLStreamConstants.COMMENT :
             case XMLStreamConstants.PROCESSING_INSTRUCTION :
                if (xmlr.hasText()) {
                   String piOrComment = xmlr.getText();
                }
                break;
          }
       }
       private static void processName(XMLStreamReader xmlr) {
          if (xmlr.hasName()) {
             String prefix = xmlr.getPrefix();
             String uri = xmlr.getNamespaceURI();
             String localName = xmlr.getLocalName();
          }
       }
       private static void processAttributes(XMLStreamReader xmlr) {
          for (int i = 0; i < xmlr.getAttributeCount(); i++)
             processAttribute(xmlr, i);
       }
       private static void processAttribute(XMLStreamReader xmlr, int index) {
          String prefix = xmlr.getAttributePrefix(index);
          String namespace = xmlr.getAttributeNamespace(index);
          String localName = xmlr.getAttributeName(index);
          String value = xmlr.getAttributeValue(index);
       }
    }
```

Poznámka

příklad převzat z Tip: Use XML streaming parsers [http://www.ibm.com/developerworks/xml/library/x-tipstx] (IBM DeveloperWorks, sekce XML).

Document Object Model (DOM)

Základní rozhraní pro tvorbu a přístup ke stromové re-prezentaci XML dat.

- existují verze *DOM Level 1, 2, 3*
- DOM je obecně *nezávislý* na způsobu analýzy (parsingu) vstupního XML
- Je popsán IDL definicemi+popisy rozhraní v jednotlivých jazycích (zejm. C++ a Java)

Specifický DOM pro HTML dokumenty

- Core (základ) DOM pro HTML je nyní "víceméně" sloučen s DOM pro XML
- určen pro styly CSS
- určen pro programování dynamického HTML (skriptování - VB Script, JavaScript)
- kromě samotného dokumentu model zahrnuje i prostředí prohlížeče (např. `window`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=window], `history` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=history]...)

Odkazy k DOM

- Tutoriál k JAXP, část věnovaná DOMPart III: XML and the Document Object Model (DOM) [http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/dom/index.html]
- Portál věnovaný DOM http://www.oasis-open.org/cover/dom.html
- Vizuální přehled DOM 1 rozhraní http://www.xml.com/pub/a/1999/07/dom/index.html
- Tutoriál "Understanding DOM (Level 2)" na http://ibm.com/developer/xmlhttp://ibm.com/developer/xml [http://ibm.com/developer/xml]

Implementace DOM

- v mnoha parserech, např. Xerces [http://xml.apache.org]
- jako součást JAXP (Java API for XML Processing) - http://java.sun.com/xml/jaxp/index.html
- i jako samostatné, nezávislé na parserech:
  - např. `dom4j`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=dom4j] - http://dom4j.org
  - `EXML`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=EXML] (Electric XML) - http://www.themindelectric.net

Práce s DOM v Javě

Co potřebujeme?

Ve všech novějších verzích Javy (JDK i JRE) je podpora DOM zabudována ve standardních knihovnách, nemusíme tedy nic doinstalovávat.

V programech musíme nicméně importovat potřebné symboly (rozhraní, příp. třídy), většinou z balíku `org.w3c.dom`.

Co nejčastěji použijeme?

Nejčastěji používanými rozhraními jsou:

Element — odpovídá pojmu "element" v logické struktuře dokumentu; zpřístupňuje název elementu, atributy, dceřinné uzly (vč. textových). Zajímavé metody:

- `Node getParentNode()` - vrátí rodičovský uzel
- `String getTextContent()` - vrátí textový obsah elementu
- `NodeList getElementsByTagName(String name)` - vrátí seznam následníků (dceřinných uzlů a jejich následníků) majících dané jméno

Node — nadrozhraní `Element`u, odpovídá obecnému uzlu v logické struktuře, tzn. může to být jak element, textový uzel, komentář, atd.

NodeList — jakýsi seznam uzlů (např. získatelný voláním `getElementsByTagName`), lze s ním pracovat metodami:

- `int getLength()` - vrátí počet uzlů v seznamu
- `Node item(int index)` - vrátí uzel na dané pozici `index`

Document — odpovídá uzlu dokumentu (je to rodič kořenového elementu)

Příklad 1 - vytvoření DOM stromu ze souboru

Příklad metody, která načte DOM strom z XML souboru (viz Úloha 1):

```
/**
 * Konstruktor, ktery vytvori novou instanci tridy Uloha1
 * nactenim obsahu xml dokumentu se zadanym URL.
```

```
 */
private Uloha1(URL url) throws SAXException, ParserConfigurationException,
        IOException {

    // Vytvorime instanci tovarni tridy
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

    // Pomoci tovarni tridy ziskame instanci DocumentBuilderu
    DocumentBuilder builder = factory.newDocumentBuilder();

    // DocumentBuilder pouzijeme pro zpracovani XML dokumentu
    // a ziskame model dokumentu ve formatu W3C DOM
    doc = builder.parse(url.toString());
}
```

Příklad 2 - modifikace DOM stromu

Příklad metody manipulující (modifikující) DOM strom dokumentu (viz Úloha 1):

```
/**************************************************************************
 * Metoda na upravu platu.
 * Ma-li osoba mensi plat nez <code>minimum</code>,
 * bude ji plat zvysen na <code>minimum</code>.
 * S ostatnimi osobami se nic nedeje.
 */
public void adjustSalary(double minimum) {

    // získej seznam elementů s platy
    NodeList salaries = doc.getElementsByTagName("salary");

    for (int i = 0; i < salaries.getLength(); i++) {

        // získej element s platem
        Element salaryElement = (Element) salaries.item(i);

        // získej plat
        double salary = Double.parseDouble(salaryElement.getTextContent());
        if (salary < minimum) {
            // modifikuj textový uzel/obsah elementu
            salaryElement.setTextContent(String.valueOf(minimum));
        }
    }
}
```

Příklad 3 - uložení XML z DOM do souboru

Příklad metody ukládající DOM strom do souboru (viz Úloha 1).

Postup používá *transformace*, které zatím neumíme. Berme to tedy jako "černou skříňku" :-)

```
public void serializetoXML(File output)
       throws IOException, TransformerConfigurationException, TransformerException

    // Vytvorime instanci tovarni tridy
    TransformerFactory factory = TransformerFactory.newInstance();

    // Pomoci tovarni tridy ziskame instanci tzv. kopirovaciho transformeru
    Transformer transformer = factory.newTransformer();

    // Vstupem transformace bude dokument v pameti
    DOMSource source = new DOMSource(doc);

    // Vystupem transformace bude vystupni soubor
    StreamResult result = new StreamResult(output);

    // Provedeme transformaci
    transformer.transform(source, result);
}
```

Alternativní stromové modely k DOM

XML Object Model (XOM)

- XOM (*XML Object Model*) vznikl jako one-man-projekt (autor Elliote Rusty Harold).
- Jde o rozhraní, které je "papežštější než papež" a striktně respektuje logický model XML dat.
- Motivaci a specifikaci najdete na domovské stránce XOM [http://cafeconleche.org/XOM/].
- Tam je též k získání open-source implementace XOM [http://cafeconleche.org/XOM/xom-1.0d24.zip] a
- dokumentace API [http://cafeconleche.org/XOM/apidocs/].

Alternativní parsery a stromové modely - NanoXML

- velmi malé (co do velikosti kódu) stromové rozhraní a parser v jednom
- dostupné jako open-source na http://nanoxml.n3.net
- adaptované též pro mobilní zařízení

- z hlediska rychlosti a paměťové efektivity za běhu ale nejlepší *není*

Prakticky dobře použitelný stromový model: dom4j

- pohodlné, rychlé a pamětově efektivní stromově-orientované rozhraní
- psané pro Javu, optimalizované pro Javu...
- dostupné jako open-source na http://dom4j.org
- nabízí perfektní přehled díky "kuchařce" [http://dom4j.org/cookbook/cookbook.html]
- dom4j je výkonný, viz srovnání efektivity jednotlivých stromových modelů [http://www.ibm.com/developerworks/xml/library/x-injava/]

Kombinace stromových a událostmi řízených přístupů

Události -> strom

- Je např. možné "nezajímavou" část dokumentu *přeskočit* nebo odfiltrovat pomocí sledování událostí a pak
- za "zajímavé" části vytvořit strom v paměti a ten zpracovávat.

Strom -> události

- Vytvoříme strom dokumentu (a zpracujeme ho) a
- strom následně procházíme a generujeme události jako bychom četli výchozí soubor.
- Toto umožňuje snadnou integraci obou typů zpracování v jedné aplikaci

Virtuální objektové modely

- DOM model dokumentu není přítomen v paměti, je zprostředkováván "on demand" při přístupu k jednotlivým uzlům
- spojuje výhody událostmi řízeného a stromového modelu zpracování (rychlost + komfort)
-

WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Sablotron] (např. viz http://www.xml.com/pub/a/2002/03/13/sablotron.html nebo http://www.gingerall.org/charlie/ga/xml/p_sab.xml)