Základní standardy a rozhraní rodiny XML

Obsah

Specifikace a validita XML

Aktuální specifikace XML

- Původní specifikace (W3C Recommendation) XML 1.0 na W3C: http://www.w3.org/XML/
- 4th Edition (aktualizace, opravy, ne změny) na Extensible Markup Language (XML) 1.0 (Fourth Edition) [http://www.w3.org/TR/2006/REC-xml-20060816/]
- výborná komentovaná verze téhož na XML.COM (Annotated XML): http://www.xml.com/pub/a/axml/axmlintro.html
- XML 1.1 (Second Edition) [http://www.w3.org/TR/2006/REC-xml11-20060816/] - změny indukované zavedením *UNICODE 3*, lepší možnosti *normalizace*, upřesnění postupu manipulace se znaky *ukončení řádku*. XML 1.1 není už vázaný na konkrétní verzi UNICODE, ale vždy na verzi poslední.

Jakou verzi použít?

Jakou verzi specifikace bychom měli v nových aplikacích používat?

Odpověď dává W3C XML Core Working Group [http://www.w3.org/XML/Core/#Publications]:

- nepíšeme-li parser, ale aplikaci, která generuje nebo vytváří XML (editor), používejme XML 1.0 (zpětná kompatibilita)

- nové parsery by měly umět XML 1.1

Validita XML dokumentů

- Opakování: každý XML dokument MUSÍ být správně utvořený (*well formed*)
- Nové: XML dokument můřže být platný (*valid*) dokument:

  Platný podle specifikace znamená *přísnější* omezení než správně utvořený.

  Obvykle se validitou myslí soulad s *DTD* (Document Type Definition) dokumentu nebo

  (moderněji) - soulad s XML Schematem, případně jinými schématy (RelaxNG, Schematron).

Document Type Definition (DTD)

Document Type Definition (DTD)

- Definice typu dokumentu (použití této definice je pak **deklarace typu dokumentu**).
- Specifikována přímo (základním) standardem XML 1.0.
- Popisuje přípustný **obsah elementů**, **atributů**, jejich implicitní (default) hodnoty, definuje použité **entity**.
- Může být uvedena jako **interní** nebo **externí** DTD (*internal and external subset*) nebo "napůl" - tam i tam.
- Dokument vyhovující DTD je označován jako *valid* (platný).
- DTD a podobným jazykům se také říká *modelovací* -- modelují/definují konkrétní značkování.
- Syntaxe DTD *není* XML (na rozdíl od většiny dalších modelovaních jazyků).

Motivace pro DTD, srovnání, výhody a nevýhody

Jaká jsou úskalí DTD?

- Zásadním úskalím DTD je "nekompatibilita" se jmennými prostory (XML Namespaces) a
- slabá modelovací schopnost - řadu omezení v modelu nelze pomocí DTD popsat.
- "Přímým" a mocnějším, ale také podstatně složitějším, nástupcem DTD je W3C XML Schema [http://www.w3.org/XML/Schema].
- Existují rovněž zdařilé, jednoduché a mocné alternativy ke XML Schematu - jako je RelaxNG [http://relaxng.org]. (on Wikipedia:RELAX_NG [http://en.wikipedia.org/wiki/RELAX_NG])

Proč vůbec (ještě) používat DTD?

- Jednoduché, všechny validující parsery ho umí.
- Pro mnoho jednoduchých značkování nepoužívajících jmenné prostory DTD dostačuje.

DTD - tutoriály

- Webreview: http://www.webreview.com/2000/08_11/developers/08_11_00_2.shtml
- ZVON: http://www.zvon.org/xxl/DTDTutorial/General/contents.html
- XML DTD Tutorial (101): http://www.xml101.com/dtd/
- W3Schools DTD Tutorial: http://www.w3schools.com [http://www.w3school.com]

DTD - deklarace typu dokumentu podrobněji

Uvádí se těsně před kořenový elementem konstrukcí

- `<!DOCTYPE jméno-kořenového-elt Externí-ID [ interní část DTD ]>`

Interní nebo **externí** část (*internal or external subset*) nemusí být uvedena nebo mohou být uvedeny obě.

Externí identifikátor může být buď

- `PUBLIC "PUBLIC ID" "URI"` (hodí se pro "veřejná", obecně uznané DTD) nebo
- `SYSTEM "URI"` - pro soukromá nebo jiná "ne zcela standardizovaná" DTD ("URI" nemusí být jen URL na síti, může být i jméno souboru, vyhodnocení se děje podle systému, na němž se vyhodnocuje)

Význam interní a externí části je rovnocenný (a nesmí si odporovat - např. dvě definice téhož elementu).

Obsahem DTD je seznam deklarací jednotlivých prvků - *elementů, seznamů atributů, entit, notací*

DTD - podmíněné sekce

Slouží k "zakomentářování" úseků DTD např. při experimentování.

- `<![IGNORE[ toto se bude ignorovat ]]>`
- `<![INCLUDE[ toto se zahrne do DTD (tj. nebude se ignorovat)]]>`

DTD - definice typu elementu

Popisuje možný obsah elementu, má formu `<!ELEMENT jméno-elementu ... >`, kde `...` může být

- `EMPTY` - prázdný element, může být zobrazen jako `<element/>` nebo `<element></element>` - totéž
- `ANY` - povolen je libovolný obsah elementu, tj. text, dceřinné elementy, ...
- může obsahovat **dceřinné elementy** - `<!ELEMENT jméno-elementu (specifikace dceřinných elementů)>`
- může být **smíšený** (MIXED) - obsahující text i dceřinné elementy dané výčtem `<!ELEMENT jméno-elementu (#PCDATA | přípustné dceřinných elementy)*>`. Nelze specifikovat pořadí nebo počet výskytů konkrétních dceřinných elementů. Hvězdička za závorkou je *povinná* - vždy je možný libovolný počet výskytů.

Pro specifikaci dceřinných elementů používáme:

- operátor **sekvence** (*sequence*, *follow with*) `,`
- operátor **volby** (výběru, *select*, *choice*) `|`
- závorky `()` mají obvyklý význam
- nelze kombinovat v jedné skupině různé operátory `, |`
- počet výskytů dceřinného elementu omezujeme specifikátory "hvězdička", "otazník", "plus" s obvyklými významy. Bez specifikátoru znamená, že je povolen právě jeden výskyt.

DTD - definice atributu

Popisuje (datový) typ, případně implicitní hodnoty atributu u daného elementu.

Má tvar `<!ATTLIST jméno-elementu jméno-atributu typ-hodnoty implicitní-hodnota>`

DTD - definice typu hodnoty atributu

Přípustné *typy hodnot* jsou:

- `CDATA`
- `NMTOKEN`
- `NMTOKENS`
- `ID`
- `IDREF`
- `IDREFS`
- `ENTITY`
- `ENTITIES`
- výčet hodnot - např. `(hodnota1|hodnota2|hodnota3)`
- výčet notací - např. `NOTATION (notace1|notace2|notace3)`

Atribut (i nepovinný) může mít implicitní hodnotu:

- `"implicitní hodnota"` - atribut je nepovinný, ale není-li uveden, chápe se to, jako by měl hodnotu `implicitní hodnota`

DTD - předpis kardinality (počtu výskytů) atributu

Atributy mohou mít předepsán (povinný) výskyt:

- `#REQUIRED` - atribut je povinný

- `#IMPLIED` - atribut je nepovinný
- `#FIXED "pevná-hodnota"` - atribut je povinný a musí mít právě hodnotu `pevná-hodnota`

DTD - implicitní hodnota atributu

Atribut (i nepovinný) může mít implicitní hodnotu:

- `"implicitní hodnota"` - atribut je nepovinný, ale není-li uveden, chápe se to, jako by měl hodnotu `implicitní hodnota`

Fyzická struktura (entity)

Entita - deklarace a použití

Rozlišuje se:

- deklarace
- reference (tj. použití) dané (již deklarované) entity.

Entity obecné (general) - mohou být

- *parsované* - soubory se (správně utvořeným) značkováním,
- *neparsované* - např. binární soubory,
- *znakové* - znaky, např. `>` je referencí na znakovou entitu

Entity parametrické (parametric)

- mohou být použity *jen v rámci DTD*
- hodí se při např. deklaracích *seznamu atributů* (pokud je dlouhý a vícekrát použitý, nahradíme ho referencí na parametrickou entitu)
- viz např. DTD pro HTML 4.01 - http://www.w3.org/TR/html4/sgml/dtd.html
- definicí parametrické entity je např. `<!ENTITY % heading "H1|H2|H3|H4|H5|H6">`

XML Base

XML Base

- XML Base, W3C Recommendation 27 June 2001: http://www.w3.org/TR/xmlbase/
- Standard pro vyhodnocování relativních URL v odkazech z/na XML dokumenty.
- Definuje použití vyhrazeného atributu `xml:base` označujícího základ pro vyhodnocování relativních URL.

- Doplňuje se se standardem *XLink*.
- Respektuje princip "překrývání" bázové adresy nastavené v nadřazeném elementu.

XML Base - příklad

Příklad 1. xml:base určuje základ pro relativní URL

```
  <!--  Slides RelaxNG locations   -->
- <group xml:base="schema/relaxng/" id="slides-relaxng"
    prefer="public">
    <uri name="slides.rng" uri="slides.rng" />
    <uri name="slides-full.rng" uri="slides-full.rng" />
  </group>
```

Poznámka

Všimněte si použití vyhrazeného prefixu `xml`:

Jmenné prostory

Jmenné prostory (XML Namespaces)

- XML Namespaces (W3C Recommendation, aktuálně *Namespaces in XML 1.0 (Second Edition) W3C Recommendation 16 August 2006*): http://www.w3.org/TR/REC-xml-names
- Existuje také nové *Namespaces in XML 1.1 W3C Recommendation [http://www.w3.org/TR/xml-names11/]* 4th February 2004. Andrew Layman, Richard Tobin, Tim Bray, Dave Hollander
- Definují "logické prostory" jmen (elementů, atributů) v XML dokumentu.
- Dávají uzlům ve stromu XML dokumentu "třetí dimenzi".
- Logickému prostoru jmen odpovídá jeden globálně ("celosvětově") jednoznačný identifikátor, daný URI (URI tvoří nadmnožinu URL).
- NS odpovídající danému URI nemá nic společného s obsahem nacházejícím se případně na tomto URL ("nic se odnikud automaticky nestahuje" - nedochází k tzv. dereferenci daného URI).

Prefixy jmenných prostorů, shoda...

- V rámci dokumentů se místo těchto URL používají zkratky, *prefixy* těchto NS namapované na příslušné URI atributem `xmlns:prefix="URI"`.

  Jméno elementu či atributu obsahující dvojtečku se označuje jako *kvalifikované jméno, QName*.
- Dva NS jsou stejné, jestliže se jejich URI shodují po znacích přesně (v kódování UNICODE).
- NS neovlivňují význam textových uzlů.

- Element/atribut nemusí patřit do žádného NS.
- Deklarace prefixu NS nebo implicitního NS má platnost na všechny dceřinné uzly rekurentně, dokud není uvedena jiná deklarace "přemapující" daný prefix.
- Jeden NS je tzv. *implicitní* (*default NS*), deklarovaný atributem `xmlns=`
- Na atributy se *implicitní NS nevztahuje*!!!, čili atributy bez explicitního uvedení prefixu nejsou *v žádném NS*.

Příklad implicitního jmenného prostoru

V následující ukázce je pro celý úryvek platný deklarovaný implicitní jmenný prostor charakterizovaný URI (URL) `http://www.w3.org/1999/xhtml`

Příklad 2. Implicitní jmenný prostor

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
   <body>
      <h1>Huráááá</h1>
   </body>
</html>
```

Příklad explicitního jmenného prostoru

V následující ukázce je deklarován a přiřazen prefixu `xhtml` jmenný prostor charakterizovaný URI (URL) `http://www.w3.org/1999/xhtml`

Příklad 3. Jmenný prostor mapovaný na prefix

```
<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
   <xhtml:body>
      <xhtml:h1>Huráááá</xhtml:h1>
   </xhtml:body>
</xhtml:html>
```

Obtíže se jmennými prostory

Dosud ne všechny parsery dokážou rozpoznávat NS. ...*i když problémy jsou s tím dnes výjimečné*...

NS jsou nekompatibilní s DTD (DTD přísně rozlišuje např. jméno `xi:include` a `include` i v případě, že patří do stejného NS a mají tedy z hlediska aplikace obvykle stejnou interpretaci/význam).

XML Information Set

XML Information Set (XML Infoset) - cíle

- *XML Infoset 2nd Edition W3C Recommendation* First published.og 24 October 2001, revised 4 February 2004, John Cowan, Richard Tobin, http://www.w3.org/TR/xml-infoset/
- Infoset popisuje "jaké všechny informace lze o uzlu (elementu, dokumentu, atributu...) získat"

- Jinými slovy: aplikace by neměla spoléhat na informace z XML dokumentu, které se po analýze (parsingu) neobjeví v Infosetu.
- Každý správně utvořený XML dokument vyhovující standardu pro jmenné prostory má Infoset.

XML Infoset - struktura

- Infoset se skládá z *Information items*
- Infoset se týká dokumentu s již expandovanými entitami
- Rozlišuje se infoset *dokumentu*, *elementu*, *atributu*, *znaku*, *instrukci pro zpracování*, *neexpandované entitě*, *neanalyzované entitě*, *notaci*
- Podrobněji viz specifikace.

Kanonický tvar XML

Kanonický tvar XML dokumentu

- Canonical XML Version 1.0, W3C Recommendation 15 March 2001, http://www.w3.org/TR/xml-c14n
- Smyslem je popsat kritéria (a algoritmy), které pomohou rozhodnout, zda jsou dva XML dokumenty ekvivalentní, lišící se pouze fyzickou reprezentací (entity, pořadí atributů, kódování znaků)
- Kanonizace "setře" rozdíly mezi takovými dokumenty, k nimž se analyzátor "bude jistě chovat stejně", tj. z pohledu aplikace jsou totožné.
- Použití kanonického tvaru je nutné např. u *elektronického podpisu* XML dat (při výpočtu hodnoty *digest*).
- Bylo by možné nad XML dokumenty definovat i jiné relace ekvivalence než je *Canonical XML*.

Kanonický tvar - zásady konstrukce

Hlavní zásady konstrukce kanonického tvaru XML dokumentu:

- kódování v UTF-8
- zlomy řádků (CR, LF) jsou normalizovány podle algoritmu uvedeného v std. XML 1.0
- hodnoty atributů jsou normalizovány
- reference na znakové a parsované entity jsou nahrazeny jejich obsahem
- CDATA sekce jsou nahrazeny jejich obsahem
- hlavička "xml" a deklarace typu dokumentu jsou odstraněny
- bílé znaky mimo kořenový element jsou normalizovány
- jiné bílé znaky (vyjma normalizace zlomu řádků) jsou zachovány

- hodnoty atributů jsou uvozeny "
- speciální znaky v hodnotách atributů a textovém obsahu elementů jsou nahrazeny referencemi na entity
- nadbytečné deklarace jmenných prostorů jsou z každého elementu odstraněny
- implicitní hodnoty atributů jsou dodány do každého elementu (kde je to relevantní)
- na pořadí atributů a deklarací jmenných prostorů se uplatní lexikografické řazení

Potíže při definici kanonického tvaru

Ztráta řady informací (typicky pocházejících z DTD):

- neparsované entity (např. binární entity) jsou po kanonizaci nepřístupné
- notace
- typy atributů (vč. implic. hodnot)

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 [http://www.google.com/search?q=dom4j] WIKIPEDIA [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 4. 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

```
        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 reprezentaci 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` [http://www.google.com/search?q=window] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=window], `history` [http://www.google.com/search?q=history] [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` [http://www.google.com/search?q=dom4j] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=dom4j] - http://dom4j.org
- `EXML` [http://www.google.com/search?q=EXML] [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)
- implementován např. u procesoru `Sablotron` [`http://www.google.com/search?q=Sablotron`] [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)