

API pro práci s XML

T. Pitner, L. Bártek, A. Rambousek. L Grolig
FI MU Brno 2021

API pro zpracování XML

Poskytují standardizovaný přístup k XML.

Slouží k propojení aplikace a parseru.

Umožňují zpracování XML bez znalosti fyzické struktury.

Optimalizují zpracování XML.

Základní typy API:

- Stromově orientovaná API – dokument je reprezentován stromem uzlů.
- API založená na událostech – parser prochází dokument a generuje události.
- Pull API – události jsou „vytahovány“ z dokumentu.

Stromová API

- Mapují dokument do v paměti uložené stromové struktury.
- Umožňují procházení celého DOM stromu.
- Nejznámější je [W3C DOM](#).
- Existuje řada implementací pro různé programovací jazyky:
 - Java [JDOM](#)
 - Java [Dom4J](#)
 - Java [XOM](#)
 - Python [4Suite](#)
 - PHP [SimpleXML](#)
 - JavaScript DOM

Objektový Model XML

Základní rozhraní pro zpracování stromové reprezentace XML

Tři verze DOM

- DOM Level 1
- DOM Level 2
- DOM Level 3

Popsán pomocí

- IDL (Interface Description Language)
- API pro jednotlivé programovací jazyky (C++, Java, ...)

HTML DOM

HTML Core DOM je víceméně konsolidován XML DOM.

Navržen pro potřeby CSS.

Používá se pro dynamické programování HTML

–Skriptování pomocí jazyků VB Script, JavaScript, ...

Obsahuje rozhraní pro prostředí prohlížeče.

–Přidány proměnné pro přístup k oknům, historii, ...

Dokumentace DOM

JAXP, část věnovaná [DOM Part III: XML and the Document Object Model \(DOM\)](#)

[Portál věnovaný DOM](#)

[DOM 1 Interface Overview](#)

Tutoriál [„Understanding DOM \(Level 2\)“](#)

Použití DOM v Javě

Novější verze Javy nabízí API pro práci s DOM.

Aplikace pouze musí provést import potřebných rozhraní, tříd, atd. z balíku `org.w3c.dom`.

Většinou jsou zapotřebí:

-**Element**

-**Node**

-**NodeList**

Použití DOM v Javě

Element

- odpovídá elementu v logické struktuře dokumentu,
- Poskytuje přístup ke jménu elementu, ke jménům atributů, dceřiným uzlům (včetně textových).
- Užitečné metody

Node getParentNode() - vrací uzel rodiče

String getTextContent() - vrací textový obsah elementu.

NodeList getElementsByTagName(String name) - vrací dceřiné elementy a jejich potomky s odpovídajícím jménem.

Použití DOM v Javě

Node

- nadřazené rozhraní k Element
- odpovídá obecnému uzlu W3C DOM
- může obsahovat

Element

Textový uzel

Komentář

...

Použití DOM v Javě

• NodeList – seznam uzlů

– Výsledek vrácený např. metodou `getElementsByTagName`.

– Poskytuje následující metody pro práci se seznamem:

• `int length()` - počet uzlů v seznamu

• `Node item(int index)` – vrací uzel v seznamu na daném indexu.

• Document – odpovídá uzlu dokumentu

– Rodičovský uzel kořenového elementu.

Event-based API

Při parsování uzlu Document generuje posloupnost událostí.

Technická implementace

-Callback metody

Hollywoodský princip: „Nevolejte nám, my Vás zavoláme.“

Aplikace implementuje zpracování událostí.

-Zpracovávají události generované parserem.

Pracuje na nižší úrovni než stromová rozhraní.

Aplikace musí provádět další zpracování.

Šetří paměť

-Sama o sobě negeneruje žádné perzistentní objekty.

Příklady událostí

•Příklady událostí:

-start document, end document

-start element – obsahuje také atributy, end element

-processing instruction

-comment

-entity reference.

•Nejznámější událostmi řízené API - [SAX](#)

Ukázka zpracování XML dokumentu

Zdrojový dokument:

```
<?xml version="1.0"?>
```

```
<doc>
```

```
<para>Hello, world!</para>
```

```
<!-- that's all folks -->
```

```
<hr/>
```

```
</doc>
```

SAX - Zpracování dokumentu

- Generované události:
- start document start element: doc
- list of attributes: empty
- start element: para
- list of attributes: empty
- characters: Hello, world!
- end element: para
- comment: That's all folks.

SAX – Zpracování dokumentu

•Dokončení generovaných událostí:

•start element: hr

•end element: hr

•end element: doc

•end document

Kdy použít událostmi řízené API

- Snažší pro programátora parseru, náročnější pro aplikačního programátora.
- Chybí kompletní dokument.
- Programátor si musí udržovat stav analýzy sám.
- Vhodné pro úlohy řešitelné bez celého dokumentu.
- Obvykle nejrychlejší způsob zpracování XML dokumentu.
- Problémy lze řešit pomocí rozšíření jako je [Streaming Transformation for XML \(STX\)](#).

Volitelné možnosti parseru SAX

- Chování parseru SAX lze ovlivnit pomocí [voleb a vlastností](#).
- Pro další podrobnosti viz Use properties and features in SAX parsers (IBM DeveloperWorks/XML).

Filtry SAX

- SAX filtry (implementace rozhraní `org.xml.sax.XMLFilter`) mohou být naprogramovány s použitím API SAX.
- Filter přijímá vstupní události, zpracovává je a posílá na výstup.
- Více informací viz [Change the events output by a SAX stream](#) (IBM DeveloperWorks/XML).

Další zdroje o SAX

• Primární zdroj:

– <http://www.saxproject.org>

• [SAX Tutorial on JAXP](#)

Pull-based API

- Aplikace nezpracovává příchozí události, ale vytahuje si data ze zpracovávaného souboru.
- Dá se použít, pokud programátor zná strukturu vstupního dokumentu a může si “vytahovat” data ze souboru.
- Protiklad API založenému na událostech.
- Velmi komfortní pro aplikačního programátora, ale implementace jsou obvykle pomalejší než zpracování postavené na událostech.

Pull-based API pro Javu

- Java nabízí API pro XML-Pull Parser API – viz [Common API for XML Pull Parsing](#)
- a také nově vyvinuté API – [Streaming API for XML \(StAX\)](#) vyvinuté v rámci JCP

Streaming API for XML (StAX)

- API se má v budoucnosti stát součástí Java API for XML Processing (JAXP).
- Nabízí dva způsoby zpracování postavenému na Pull API:
 - “vytahování” událostí pomocí iterátoru – komfortnější
 - nízko-úrovňový přístup s použitím tzv. kurzoru – rychlejší.

StAX – ukázka použití iterátoru

• Viz také [Oracle Java Tutorials](#)

• V této ukázce klientská aplikace vytahuje události pomocí metody parseru `next()`

• Ukázková data:

• `<?xml version="1.0" encoding="UTF-8"?>`

• `<BookCatalogue xmlns="http://www.publishing.org">`

• `<Book>`

• `<Title>Yogasana Vijnana: the Science of Yoga</Title>`

• `<Author>Dharma Publishing</Author>`

StAX – zdrojový kód Java

```
try {  
    for (int i = 0 ; i < count ; i++) {  
        // pass the file name.. all relative entity  
        // references will be resolved against this  
        // as base URI.  
  
        XMLStreamReader xmlr =  
xmlif.createXMLStreamReader(filename,  
        new FileInputStream(filename));
```

```
        // ... XMLStreamReader ...
```


StAX – zdrojový kód Java

```
while(xmlr.hasNext()) {  
    eventType = xmlr.next();  
    printEventType(eventType);  
    // these functions print the information  
    // about the particular event by calling  
    // the relevant function  
    printStartElement(xmlr);  
    printEndElement(xmlr);  
}
```

Kombinace stromového a událostmi řízeného zpracování

•Events \rightarrow tree

•Tree \rightarrow events

Events → tree

- Umožňuje buď přeskočit nebo dofiltrovat nezajímavé části dokumentu pomocí monitorování událostí.
- Vytvořit stromový model z části, která nás zajímá a tuto část zpracovat.

Tree → events

- Vytvoříme strom dokumentu (a zpracováváme ho) a
- při procházení stromu generujeme události jako bysme zpracovávali XML soubor.
- Tento přístup umožňuje integraci obou typů zpracování.

Virtual Object Model

- DOM není vytvořen v paměti, ale vytváří se podle potřeby při přístupu k jednotlivým uzlům.
- Kombinuje výhody zpracování pomocí údajosti a pomocí stromu (rychlost a komfort).
- Existuje implementace [Sablotron procesor](#).

Alternativní stromové modely

.XML Object Model (XOM)

- XOM (XML Object Model) created as an one man project (author Eliote Rusty Harold).
- It is an interface that strictly respect XML data logical model.
- For motivation and specification see the XOM home page (<http://www.xom.nu>).
- You can get there the open-sourceXOM implementation and
- the API documentation, too.

Alternativní stromové modely

- DOM4J – prakticky použitelný stromový model
 - comfortable, fast and memory efficient tree-oriented interface
 - designed and optimized for Java.
 - Dostupný jako open-source.
 - Perfektní “kuchařka”.