Tricks and Tips for Making Your XML Application Go Faster

Neil Graham
IBM Canada Ltd.
Objectives

- To give you ideas on how to make the XML parsing portions of applications go faster
- Help you choose which API to use
  - Compare and contrast various parsing APIs with respect to performance
- Some SAX tips
- DOM tips
- Using JAXP 1.3 APIs
Agenda

- Compare XML Parsing paradigms and common APIs that correspond to them
- Using JAXP 1.3 to improve performance
  - Parsing and XPath
- DOM level 3 performance tips
- SAX performance hints
Parsing API Paradigms

- **XML Object Models:**
  - Document is mapped to a tree structure
  - Elements, attributes, comments, *etc.* are nodes in the tree
  - Each node (usually) corresponds to an Object
- **Examples:** W3C DOM, JDOM, DOM4J, XOM
Parsing API Paradigms (Continued)

- Event-based APIs:
  - Application behaves like a listener to a radio sports broadcast
  - As the document goes by, the parser describes it in detail
  - Element start tags with their attributes; comments; content; end tags. ...

- e.g.: SAX, XNI (Xerces Native Interface)
Parsing API Paradigms (Continued)

- Pull parsing APIs:
  - Application has a “cursor” on the document
  - At any point, the application can explore deeper in the tree, stay at the same depth, or go towards the root to explore other paths

- e.g.: StAX, XMLPull, NekoPull
Parsing API Paradigms (Continued)

- **Object-binding APIs:**
  - XML content mapped to a language-specific structure
    - Often using a schema to define the mapping
  - An element might map to a class
  - A particular attribute could map to a field

- **Examples:** JAXB, XMLBeans, Castor, SDO (Service Data Objects)

- Beyond this talk’s scope
When to Use Which API?

- Tree-based models good when you have to:
  - Process all, or almost all of the document
  - Documents are small relative to the size of available RAM
  - Changes need to be made resulting in document restructuring
  - An XML-centric view of document required
  - W3C DOM is widely available and interoperable
When to Use Which API (Continued)

- But, tree-based APIs:
  - Tend to result in memory footprint larger than original document
    - Sometimes much larger
  - Are very expensive to build
  - Can be expensive to access portions first time (implementations often defer node creation)

- Many such APIs carry virtually all information any application could ever want
When to Use Which API (Continued)

- Event-based APIs:
  - Are comparatively lightweight
  - Good for building custom object models
  - SAX is very widely available and interoperable

- But, they’re low-level, harder to use
  - Application has to build its own structures/do its own bookkeeping
  - Still generate events for whole document
  - With SAX, lots of Strings!
When to Use Which API (Continued)

- **Pull APIs:**
  - May not need to produce events for all of document
  - May be more intuitive than event-based APIs
- Not widely available (no pull implementation part of JSE 5.0 or JEE 1.4)
  - APIs and implementations not nearly as mature as others
JAXP 1.3 and Performance

- JAXP 1.3 mandated support in JSE 5.0 of XML 1.1/XML Namespaces 1.1, XInclude, XML Schema
- Also completes XML Schema datatype → Java mapping
- But offers a means for compiling XML Schemas and XPath expressions for fast validation/processing
- We’ll cover examples of both
JAXP 1.3: Validation API

- Grammars tend to be expensive to parse
- One wants to be able to reuse them for multiple documents
- Would also be nice to validate a generic DOM level 2/3 tree or a stream of SAX events — say dynamically generated from an XSL transformation
- This API allows for all of this
The Schema Abstract Class

- Represents the result of processing a particular grammar (e.g., set of XML Schema documents)
- Immutable (and therefore thread-safe)
- Used for creating Validators (and ValidatorHandlers) to validate documents
- No facilities provided for introspecting the grammar (yet!)
SchemaFactory

- A factory for creating Schema objects
- `static newInstance(String schemaLanguage):` parameter describes the schema language for which a factory is to be created
- JAXP 1.3 defines the URI to be used for Relax NG, but only XML Schema must be supported
- To support multiple schema languages as well as multiple implementations, a complex lookup mechanism provided
Validator

- Validates some (SAX or DOM) Source, optionally producing a (SAX or DOM, respectively) Result with infoset augmentations
- *e.g.*, default attributes will be filled in
- These objects are not thread-safe
- As with most JAXP objects, implementation-specific features and properties may be set/queried
Validator (Continued)

- An LSResourceResolver may be attached (to deal with schemaLocation hints referring to schemas outside the knowledge of the Validator, for instance)

- An ErrorHandler should also be set
  - As with SchemaFactory, if no ErrorHandler is set, all errors will result in SAXParseExceptions being thrown

- In deference to the parser/transform packages, StreamSources and StreamResults are not handled
**ValidatorHandler**

- Validates a stream of SAX2 events by acting as a ContentHandler
- A ContentHandler implementation can be set, in which case it acts as a filter, augmenting events appropriately
- An LSResourceResolver may be set, an ErrorHandler should be set
JAXP Validation Example: Simple Instance

```xml
<inventory xmlns="http://www.booze.com">
  <product sku="b01" amount="4000"
    price="0.25" cost="0.10">
    bad beer
  </product>
  <product sku="b03" amount="250" price="1.00"
    cost="0.50">
    good beer
  </product>
</inventory>
```
JAXP Validation Example: Simple Instance (Continued)

```xml
<product sku="s05" amount="100"
    price="20.00" cost="12.00">
    half-decent Scotch
</product>

<assets cash="5400.00" inventory="3250.00"/>
</inventory>
```
JAXP Validation Example: Schema

```xml
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.booze.com"
    xmlns="http://www.booze.com">
  <xsd:simpleType name="moneyType">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0.00"/>
      <xsd:fractionDigits value="2"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```
JAXP Validation Example: Schema (Continued)

```xml
<xsd:element name="inventory">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="product"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="assets"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```
JAXP Validation Example: 
Schema (Continued)

```xml
<xsd:element name="product">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="cost" type="moneyType"/>
        <xsd:attribute name="price" type="moneyType"/>
        <xsd:attribute name="sku" type="xsd:ID"/>
        <xsd:attribute name="amount" type="xsd:nonNegativeInteger"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```
JAXP Validation Example: Schema (Continued)

```xml
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
<xsd:element name="assets">
  <xsd:complexType>
    <xsd:attribute name="cash" type="moneyType"/>
    <xsd:attribute name="inventory" type="moneyType"/>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```
public class ValidationTest implements ErrorHandler {
    public static void main (String [] args) throws Exception {
        ValidationTest test = new ValidationTest();
        // create a SchemaFactory
        System.out.println(
            "creating SchemaFactory instance");
        SchemaFactory sf = SchemaFactory.newInstance(
            XMLConstants.W3C_XML_SCHEMA_NS_URI);
        // in case there are any errors in the schema
        sf.setErrorHandler(test);
// and create a schema from args[1]
System.out.println("creating Schema instance from "+ args[1]);
Schema s = sf.newSchema(new File(args[1]));
// create a DOM parser factory
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
dbf.setSchema(s);
DocumentBuilder db = dbf.newDocumentBuilder();
db.setErrorHandler(test);
JAXP Validation Example: Code (Continued)

```java
System.out.println("parsing " + args[0]);
Document inventory = db.parse(args[0]);
test.printStatus(inventory);
// set ourselves up to do transactions
DOMSource docSource = new DOMSource(inventory);
Validator val = s.newValidator();
val.setErrorHandler(test);
```
JAXP Validation Example: Code (Continued)

System.out.println(
    "transaction #1: buy 1000 bad beer"); 
if(test.transact(inventory, docSource, val, 
    true, "b01", 1000)) {
    System.out.println(
        "transaction failed! Bailing...
    "); 
    test.printStatus(inventory); 
    System.exit(0); 
} 
    test.printStatus(inventory); 
    // a second transaction omitted...
System.out.println("Let's buy expensive scotch 'til we go broke...");

while(true) {
    if(test.transact(inventory, docSource, val, true, "s05", 400)) {
        System.out.println(
            "transaction failed! Bailing...\n");
        test.printStatus(inventory);
        System.exit(0);
    }
}

test.printStatus(inventory);
private boolean fError;
public ValidationTest() {
    fError = false;
};

// SAX ErrorHandler methods just print
// appropriate messages and set fError
// printStatus is not interesting; a few DOM
// l2 methods to traverse the tree
public boolean transact(Document doc,
    Source docSource,
    Validator validator,
    boolean weAreBuying,
    String sku, int amount) {
    Element target = doc.getElementById(sku);
    if(target == null )
        return false;
    Attr targetAmount =
        target.getAttributeNodeNS(null, "amount");
Attr targetPrice = target.getAttributeNodeNS(null, "price");
Attr targetCost = target.getAttributeNodeNS(null, "cost");

// use some knowledge of doc structure here...
Element assets = (Element)doc.getElementsByTagNameNS("http://www.booze.com", "assets").item(0);
Attr inventoryValue = assets.getAttributeNodeNS(null, "inventory");
Attr cash = assets.getAttributeNodeNS(null, "cash");

if(weAreBuying) {
    BigDecimal totalCost = (new BigDecimal(targetCost.getValue())).multiply(new BigDecimal(amount));

    BigDecimal totalPrice = (new BigDecimal(targetPrice.getValue())).multiply(new BigDecimal(amount));
targetAmount.setValue(Integer.toString(
    Integer.parseInt(targetAmount.getValue())
    + amount));
inventoryValue.setValue((new BigDecimal(
    inventoryValue.getValue())).add(
    totalPrice).toString());
cash.setValue((new BigDecimal(
    cash.getValue())).subtract(
    totalCost).toString());
JAXP Validation Example:

Code *(Continued)*

```java
// similar for when selling; now validate:
try {
    validator.validate(docSource);
} catch(SAXException e) {
    return false;
} catch(IOException e) {
    return false;
} return fError;
```
JAXP Validation Example:

Output

creating SchemaFactory instance
creating Schema instance from ex6.xsd
parsing ex6.xml

<table>
<thead>
<tr>
<th>Product SKU</th>
<th>Amount</th>
<th>Total Cost</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>b01</td>
<td>4000</td>
<td>400.00</td>
<td>1000.00</td>
</tr>
<tr>
<td>b03</td>
<td>250</td>
<td>125.00</td>
<td>250.00</td>
</tr>
<tr>
<td>s05</td>
<td>100</td>
<td>1200.00</td>
<td>2000.00</td>
</tr>
</tbody>
</table>

Cash on hand: 5400.00; total value of inventory: 3250.00
JAXP Validation Example: Output (Continued)

transaction #1: buy 1000 bad beer

<table>
<thead>
<tr>
<th>Product SKU</th>
<th>Amount</th>
<th>Total Cost</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>b01</td>
<td>5000</td>
<td>500.00</td>
<td>1250.00</td>
</tr>
<tr>
<td>b03</td>
<td>250</td>
<td>125.00</td>
<td>250.00</td>
</tr>
<tr>
<td>s05</td>
<td>100</td>
<td>1200.00</td>
<td>2000.00</td>
</tr>
</tbody>
</table>

Cash on hand: 5300.00; total value of inventory: 3500.00
JAXP Validation Example: Output (Continued)

Let's buy expensive scotch 'til we go broke...

<table>
<thead>
<tr>
<th>Product SKU</th>
<th>Amount</th>
<th>Total Cost</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>b01</td>
<td>5000</td>
<td>500.00</td>
<td>1250.00</td>
</tr>
<tr>
<td>b03</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>s05</td>
<td>500</td>
<td>6000.00</td>
<td>10000.00</td>
</tr>
</tbody>
</table>

Cash on hand: 750.00; total value of inventory: 11250.00
JAXP Validation Example: Output (Continued)

Error: http://www.w3.org/TR/xml-schema-1#cvc-minInclusive-valid?-4050.00&0.0&moneyType

Error: http://www.w3.org/TR/xml-schema-1#cvc-attribute.3?assets&cash-&4050.00&moneyType

transaction failed! Bailing...
JAXP Validation Example: Output (Continued)

<table>
<thead>
<tr>
<th>Product SKU</th>
<th>Amount</th>
<th>Total Cost</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>b01</td>
<td>5000</td>
<td>500.00</td>
<td>1250.00</td>
</tr>
<tr>
<td>b03</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>s05</td>
<td>900</td>
<td>10800.00</td>
<td>18000.00</td>
</tr>
</tbody>
</table>

Cash on hand: -4050.00; total value of inventory: 19250.00
JAXP XPath API

- Allows XPath 1.0 expressions to be evaluated on any data model that can be mapped to XPath 1.0
- Various other XPath API’s are either bound to a particular data model (e.g., DOM level 3) or somewhat tied to an implementation (e.g., Jaxen)
- Since both data models and implementations may vary, factory lookup mechanism is complex and works like that of validation API
**XPathFactory (Continued)**

- W3C DOM only data model that must be supported
- Can get and query features (but not properties)
- XPathFunctionResolver and XPathVariableResolver can be set, and will be used by all manufactured XPath objects
- The newXPath() method will create new XPath objects appropriate for the type of data model in use
**XPathVariableResolver**

- Implemented by the application, this is used to provide values for variables referenced in XPath expressions
- Will be called by the XPath processor when it encounters the variable
- Returns an Object; that Object must be appropriate for the data model in use
- *e.g.*, for DOM, would most often be some type of org.w3c.dom.Node
**XPathFunctionResolver**

- Like XPathVariableResolver, implemented by the application and invoked when an unknown XPath function is encountered.

- Takes a QName object — the function’s name — and an int corresponding to the number of arguments.

- Returns an XPathFunction object.

- The processor will call the evaluate method on this object.

- `evaluate` takes a List with the appropriate number of elements; returns an Object which must be appropriate to the data model.
XPath

- XPath objects can be reset; they are neither immutable nor thread-safe
- An XPathVariableResolver and XPathFunctionResolver can be attached/queried
- A NamespaceContext can also be set/queried
  - Required when compiling an XPathExpression
  - Since there is no context node
XPath (Continued)

- This kind of static compilation is very useful when particular XPath expressions will be used often and/or against various documents
  - With many processors, the XPathExpression will be compiled into Java bytecode and executed directly
- XPath objects may also be used for evaluation; then a String representing the XPath expression is interpreted directly
- In this case, the namespaces in scope given the context node are used to interpret the XPath expression
Evaluation of XPath Expressions

- Both XPath and XPathExpression have 4 evaluate methods
  - Only difference: XPath’s take a String for the XPath expression; XPathExpression embodies the expression

- May provide a Node as context, or, by providing an InputSource, use document root as context

- Either a QName parameter specifies return type, or String implied
XPath 1.0 Data Model → Java

- XPath number type: maps to Java Double; called out with the NUMBER field of XPathConstants
- XPath string type: maps to Java String; called out with the STRING field of XPathConstants
- XPath boolean type: maps to Java Boolean; called out with the BOOLEAN field of XPathConstants
- XPath node list type: depends on data model; in DOM, maps to NodeList; called out with the NODESET field of XPathConstants
XPath Example: Instance

```xml
<?xml version="1.0" encoding="ASCII"?>
<purchaseOrder xmlns="http://stockings.com">
    <order id="12365">
        <product name="ratty green socks" id="rgs09" quantity="22"/>
        <shipTo name="Pete the Pauper" address="a warm grate"/>
    </order>
    <billTo name="Pete the Pauper" address="a warm grate" totalPrice="22"/>
</purchaseOrder>
```
XPath Example: Code

private static final String[] BAD_CUSTOMERS = {
    "Shifty Seamus", "Pete the Pauper"};

// in the main method
String xpathFilter =
    "string(s:purchaseOrder/s:billTo/@name)";
String totalPriceExpr =
    "number(s:purchaseOrder/s:billTo/@totalPrice)";
String destinationsExpr =
    "s:purchaseOrder//@shipTo";
XPath Example: Code (Continued)

```java
try {
    DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    dbf.setValidating(false);
    DocumentBuilder db =
    dbf.newDocumentBuilder();
    /* get an XPathFactory for the DOM */
    XPathFactory xf =
    XPathFactory.newInstance();
    XPath xpath = xf newXPath();
}```
XPath Example: Code (Continued)

// we have to make a NamespaceContext since we're going to precompile our filter:
NamespaceContext nsc = new SimpleNamespaceContext();
xpath.setNamespaceContext(nsc);
XPathExpression filterExpression = xpath.compile(xpathFilter);
InputSource fileSource = new InputSource();
for(int i=0; i<args.length; i++) {
    fileSource.setSystemId(args[i]);
    String customerName = filterExpression.evaluate(fileSource);
    if(customerName == null ||
        customerName.length() == 0) {
        System.out.println(
            "not processing file " + args[i] + "; no customer.");
        continue;
    }
    boolean processCustomer = true;
for(int j=0; j<BAD_CUSTOMERS.length; j++) {
    if(customerName.equalsIgnoreCase(
            BAD_CUSTOMERS[j])) {
        System.out.println(
            "Considering order from bad customer...");
        // lost the bet; have to process this
        
        db.reset();
        Document doc = db.parse(fileSource);
XPath Example: Code (Continued)

Double totalPrice =
    (Double)xpath.evaluate(
        totalPriceExpr,
        doc, XPathConstants.NUMBER);

NodeList orders =
    (NodeList)xpath.evaluate(
        destinationsExpr,
        doc, XPathConstants.NODESET);
if (totalPrice.doubleValue() > 25.0) {
    processCustomer = false;
    System.out.println("
            Declining to process order for bad customer"
                        + customerName + " because they ordered $" + totalPrice + ", which is more than $25");
}
if (orders.getLength() > 1) {
    processCustomer = false;
    System.out.println("Declining to process order for bad customer" + customerName + " because their order involved " + orders.getLength() + " destinations.");
}
XPath Example: Code (Continued)

```java
}
if(processCustomer) {
    System.out.println("order from customer"
    + customerName
    + " will be processed.");
}
}
} catch (Exception e) {
    System.out.println(
        "Oh-oh; something went badly wrong...");
}
private static class SimpleNamespaceContext
    implements NamespaceContext {
private HashMap<String, String> fPrefix2URIMap;
private HashMap<String, String> fURI2PrefixMap;
SimpleNamespaceContext () {
    fPrefix2URIMap = new HashMap<String, String>();
    fURI2PrefixMap = new HashMap<String, String>();
// set up our namespaces
fURI2PrefixMap.put("http://stockings.com", "s");

fPrefix2URIMap.put("s", "http://stockings.com");

public String getNamespaceURI(String prefix) {
    String retVal = fPrefix2URIMap.get(prefix);
    return (retVal == null) ? XMLConstants.NULL_NS_URI : retVal;
}
XPath Example: Code (Continued)

```java
public String getPrefix(String uri) {
    String retVal = fURI2PrefixMap.get(uri);
    return (retVal == null) ? XMLConstants.NULL_NS_URI : retVal;
}

public Iterator getPrefixes(String uri) {
    String retStr = fPrefix2URIMap.get(uri);
    Vector<String> retVal = new Vector<String>();
    if(retStr != null) {
        retVal.addElement(retStr);
    }
    return retVal.iterator();
}
```
XPath Example: Output

Considering order from bad customer...
Declining to process order for bad customer
  Shifty Seamus because they ordered $839.0, which is more than $25
Declining to process order for bad customer
  Shifty Seamus because their order involved 2 destinations.
order from customer Reliable Rick will be processed.
Considering order from bad customer...
order from customer Pete the Pauper will be processed.
DOM Level 3

- DOM level 3 not for improving performance
- It adds function:
  - A new infrastructure for configuring features
  - Methods for handling text content
  - Documents can be “normalized” (namespaces fixed-up, text nodes merged, maybe revalidated)
  - Some PSVI information (name of type responsible for validating Element/Attr)
  - Provides its own error reporting interface
DOM l3: Load/Save Module

- Finally specifies a means for parsing / serializing documents
- And for filtering out certain nodes
  - So, if want a DOM but don’t require all of document...
- LSPARSERFILTER can help!
- LSSerializ filter similarly lets you avoid writing entire DOM to a file
- LSResourceResolver provides entity resolution capabilities
LSParseFilter

- Application implements the LSParseFilter interface
  - It must call setFilter method of LSParse to attach the filter to the LSParse instance
- LSParse will call getWhatToShow(): int on the LSParseFilter
  - That’s how it learns what Node types to inform the LSParseFilter of
LSParserFilter (Continued)

- `startElement(Element):` short: filter method called by LSParser after `startTag` has been parsed
  - If rejected by filter, permits efficient skipping of entire subtree!
  - Will never be built!

- `acceptNode(Node):` short: called by LSParser after `Node` has been processed
  - Application may accept, reject, or even modify it
  - But the node’s subtree was already constructed
DOM 3 LS: Example

- The following parses a document, preventing all PI’s or elements or attribute with localName “silly” from becoming part of the DOM, then serializes it

```java
// this is bogus; JDK doesn't seem to contain default DOMImplementationSources
System.setProperty(
    DOMImplementationRegistry.PROPERTY,
    "com.sun.org.apache.xerces.internal.dom.DOMXSImplementationSourceImpl");
// get DOM Implementation using DOM Registry
DOMImplementationRegistry registry =
    DOMImplementationRegistry.newInstance();
```
DOM 3 LS: Example (Continued)

// note the explicit need to cast
DOMImplementationLS impl =
(DOMImplementationLS)
registry.getDOMImplementation("LS");
// create DOMBuilder
// not validating so schemaType is null
builder = impl.createLSParser(
   DOMImplementationLS.MODE_SYNCHRONOUS,
   null);

DOMConfiguration config =
    builder.getDomConfig();
// create filter
LSParserFilter filter = new DOMLSTest();
builder.setFilter(filter);
// This relies on autoboxing!
config.setParameter("validate", false);
config.setParameter("namespaces", true);
// parse document
System.out.println("Parsing " + argv[0] + ", removing silly elements and PI's...");
Document doc = builder.parseURI(argv[0]);
DOM 3 LS: Example (Continued)

```
// create LSSerializer
LSSerializer domWriter = impl.createLSSerializer();
System.out.println("Serializing document");
config = domWriter.getDomConfig();
config.setParameter("xml-declaration", true);
// serialize document to argv[1]
LSOutput dOut = impl.createLSOutput();
FileOutputStream fos = new FileOutputStream(new File(argv[1]));
    dOut.setByteStream(fos);
domWriter.write(doc, dOut);
```
DOM 3 LS Example: acceptNode Implementation

public short acceptNode(Node node) {
    if(node.nodeType == Node.ELEMENT_NODE) {
        // loop through attrs removing silly ones
        Element elt = (Element)node;
        NamedNodeMap attrMap = node.getAttributes();
        for(int i=0; i<attrMap.getLength(); i++) {
            if(attrMap.item(i).getLocalName().equalsIgnoreCase("silly")){
                elt.removeAttributeNode((Attr) attrMap.item(i));
            }
        }
    }
}
DOM 3 LS Example: `acceptNode` *(Continued)*

```java
} }

if (node.getNodeType() == Node.PROCESSING_INSTRUCTION_NODE) {
    return NodeFilter.FILTER_REJECT;
}

else if (node.getNodeType() ==
          Node.PROCESSING_INSTRUCTION_NODE) {
    return NodeFilter.FILTER_REJECT;
}

return NodeFilter.FILTER_ACCEPT;
```

Neil Graham — Tricks and Tips for Making Your XML Applications Go Faster
public int getWhatToShow() {
    return NodeFilter.SHOW_PROCESSING_INSTRUCTION
           | NodeFilter.SHOW_ELEMENT;
}
DOM 3 LS Example:

```java
public short startElement(Element elt) {
  if(elt.getLocalName().equalsIgnoreCase("silly")) {
    return NodeFilter.FILTER_REJECT;
  }
  return NodeFilter.FILTER_ACCEPT;
}
```
DOM 3 LS Example:
Example Instance

```xml
<root xmlns="http://www.tests.org/ex5"
     xmlns:ns="http://silly.com">
  <ns:serious>A serious element</ns:serious>
  <!-- and a comment -->
  <silly>
    <ns:serious>another serious element</ns:serious>
  </silly>
  <?processing instruction?>
  <serious silly="a silly attribute" attr="another attribute"/>
</root>
```
DOM 3 LS Example: Example Output

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://www.tests.org/ex5"
     xmlns:ns="http://silly.com">
  <ns:serious>A serious element</ns:serious>
  <!-- and a comment -->
  <serious attr="another attribute"/>
</root>
```
Other DOM Tips

- Xerces-J (in both Sun and IBM JDK’s) uses a “deferred” DOM implementation by default
- Great for large documents, or if much of the document will never be traversed
- But if your application visits most/all nodes in the DOM, can be costly...
- So turn it off:
  - `DOMConfiguration.setParameter(http://apache.org/xml/features/dom/defer-node-expansion, false)`
Some Tips for SAX

- SAX tends to produce a lot of Strings
- But XML documents are very repetitive
  - e.g., elements repeat with their attributes
- SAX provides a feature,
  http://xml.org/sax/features/string-interning
  - When enabled, parser will intern all element and attribute names in JDK
    - Saves objects!
  - Also, you can compare using ==, much cheaper than String.equals!
Some Tips for SAX (Continued)

- What if your application is only interested in the initial part of a document?
- Throw a SAXException when done!
  - This will abort the parse
  - The parser will clean up all resources (streams, etc.)
- Will have to catch it in your own code of course
Some Tips for SAX (Continued)

- Common pattern with SAX:
  - Lots of logic in startElement call
  - *e.g.*, if elemName.equals(x) {...} else if descendFrom(y) {...}

- Can gain by separating code into different ContentHandlers
  - *e.g.*, if see y, attach new ContentHandler for y’s descendants; it reattaches old one when done
  - Current ContentHandler just focuses on processing x and identifying y
References — APIs

- Document Object Model (DOM) Level 3 Core Specification:
  http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

- Document Object Model (DOM) Level 3 Load and Save Specification:
  http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407
References — APIs (Continued)

- SAX home page: http://www.saxproject.org/
- JAXP 1.1/1.2 (Final): http://www.jcp.org/en/jsr/detail?id=63
- JAXP 1.3 (Final, included in JSE 5.0): http://www.jcp.org/en/jsr/detail?id=206