웹 기술 및 응용

XML DOM and SAX

2018년 2학기

Instructor: Prof. Young-guk Ha
Dept. of Computer Science & Engineering

Part I.
DOM (Document Object Model)

Contents

- What is DOM?
- DOM Core Interface
  - Tree-Structured Model
  - Memory Management
  - Fundamental Interfaces
- Xerces DOM API
- DOM API Code Example

What is DOM?

❑ W3C definition

- "The ***Document Object Model*** (***DOM***) is a *platform-neutral and language-neutral standard interface* that will allow programs and scripts to dynamically access and update the content, structure and style of documents*"

* *Valid HTML and well-formed XML documents*

❑ Why DOM?

- If there is no standard API accessing the XML document, each parser would implement the same set of features in its own way

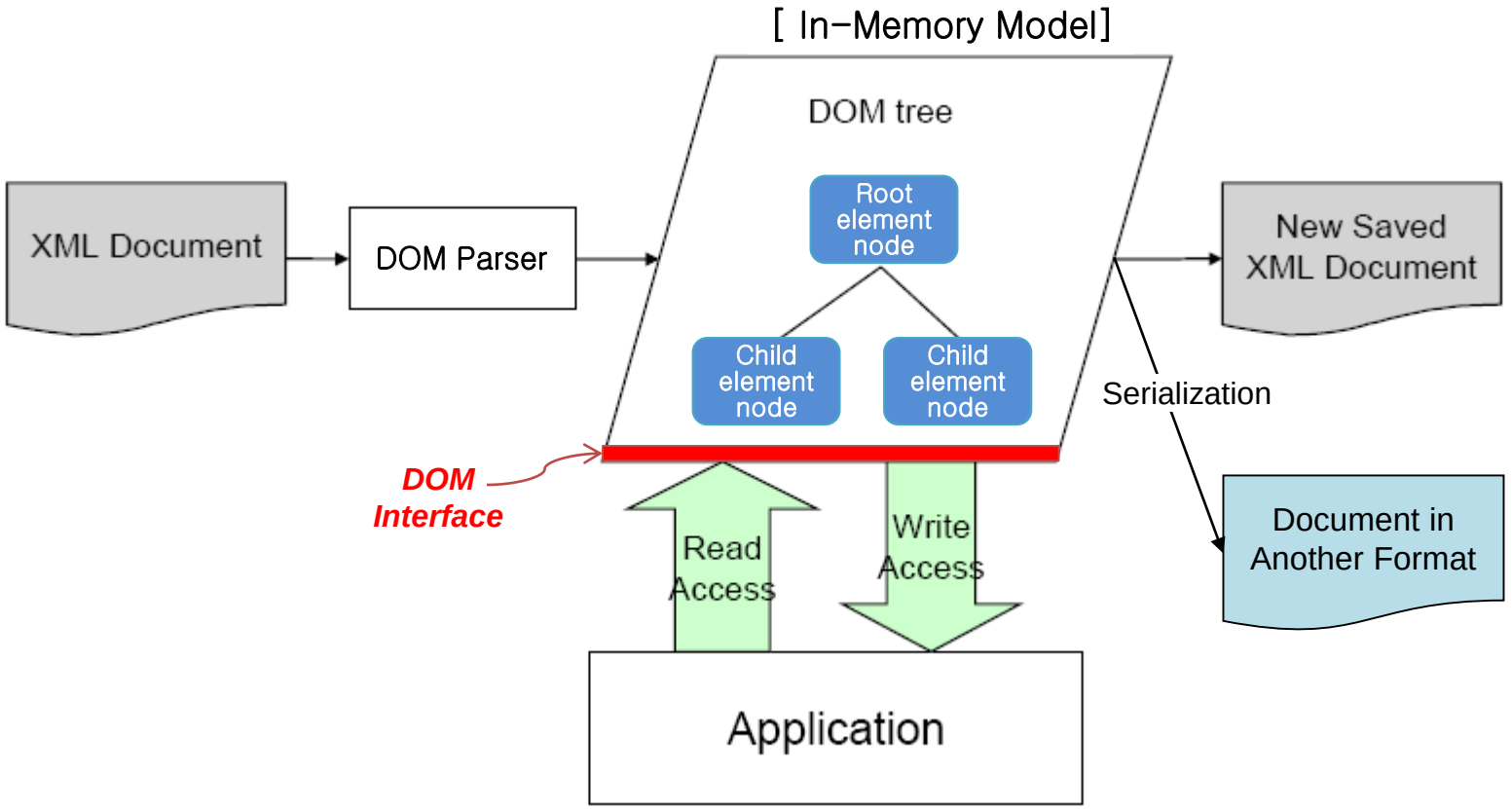
What the DOM is/does not

- DOM is NOT a product, but an idea
- DOM does NOT specify any sort of binary formats NOR any concrete data structures
  - It defines *abstract interfaces* for handling XML
- DOM is NOT a middleware technology such as COM or CORBA
  - Though standard DOM structure can be used in interoperating environments

History of DOM

- Started by W3C recommendation in 1997
- DOM specification is being built level-by-level
  - Refer to http://www.w3.org/TR/DOM-Requirements for the specific requirements of each DOM level
- Current DOM version is Level 3
  - Working on DOM level 4 (working draft)

How the DOM processes XML Documents



When to use or not to use the DOM (1/2)

- The two major consideration
  - The size of the document with which we are working
  - What type of processing we want to do with the document

When to use or not to use the DOM (2/2)

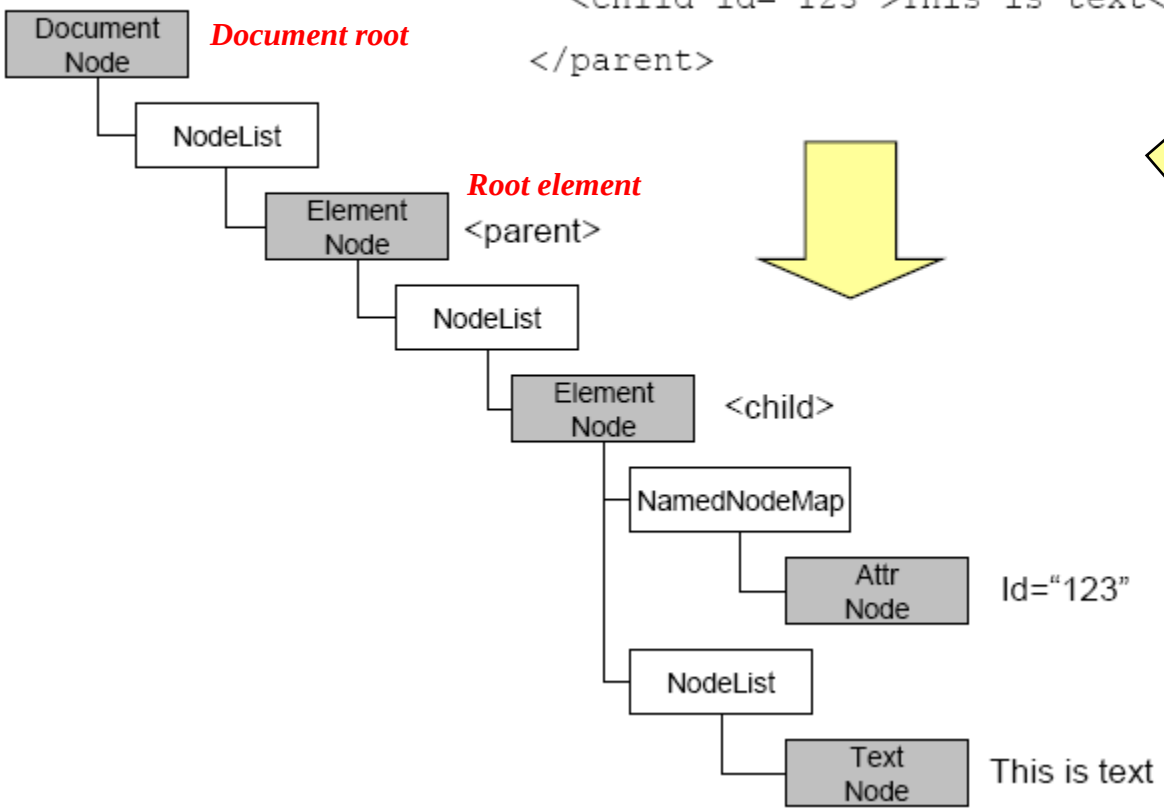
- DOM creates an in-memory tree representation of the XML document
  - DOM representation can need over 5-10 times as much memory as the corresponding XML file itself
- When the size of the document becomes larger, system load becomes more significant!
- An alternative to DOM approach
  - SAX (Simple API for XML)
    - Does not load the entire document at once
    - Uses *event-driven processing*: processes XML documents as stream of events

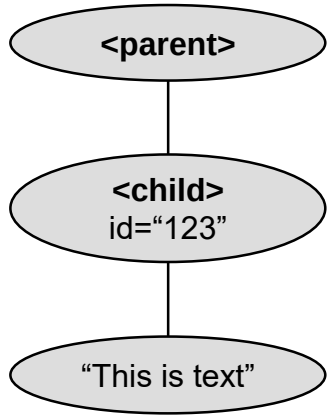
DOM Interfaces

- DOM implementation creates an in-memory tree object model (***DOM tree***) that represents the document
  - DOM tree is an internal representation of the document, which consists of ***Nodes***
- Node exposes a set of interfaces
  - DOM specification tells us what these interfaces are
  - Defined in OMG's IDL (Interface Definition Language)
- DOM interfaces are broken down into
  - Fundamental interfaces
    - Fully implemented by all conforming implementations of the DOM
  - Extended interfaces
    - Only for extended features of XML (never be encountered in a DOM implementation that deals only with HTML)

DOM Tree Example

```
<parent>
  <child id="123">This is text</child>
</parent>
```





< DOM Tree >

< Conceptual Tree >

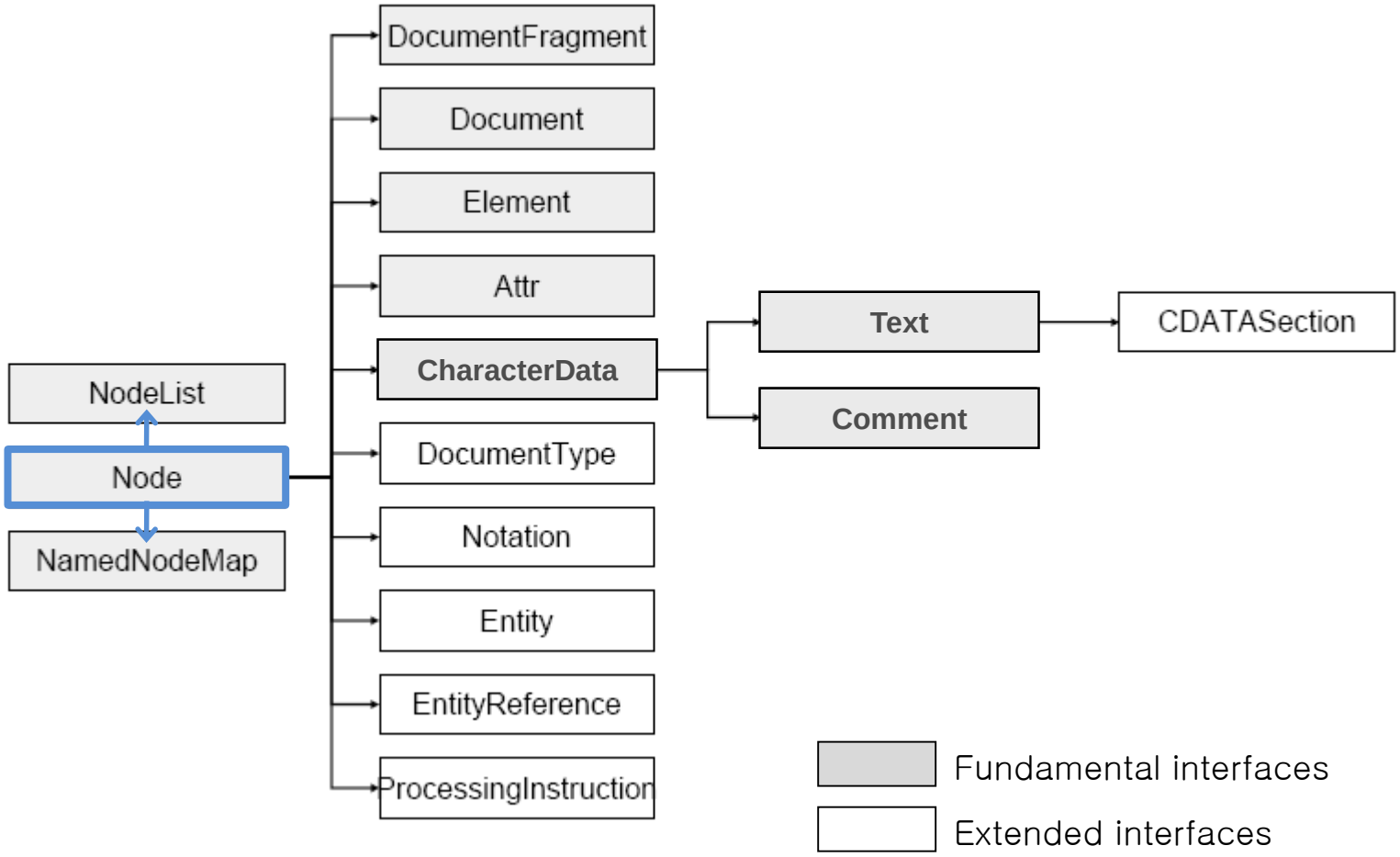
Memory Management

- DOM API is designed to be compatible with a wide range of programming languages
  - Language bindings
    - Java, JavaScript
    - C, C++
    - Python, …
  - Thus, DOM API needs to be operated across a variety of memory management philosophy
- To ensure a consistent API across platforms, DOM does NOT address specific memory management issues at all
  - Leaves these to individual implementations

Basic Types

- **`DOMString`**
  - UTF-16 encoded strings
  - E.g., bound to **`String`** type in Java
- **`DOMTimestamp`**
  - Used to store an absolute or relative time in millisecond
  - E.g., bound to **`long`** type in Java
- **`DOMUserData`**
  - Used to store application data
  - E.g., bound to **`Object`** type in Java
- **`DOMObject`**
  - Represents an object reference
  - E.g., bound to **`Object`** type in Java

Core Interfaces



Node Interface

- Most fundamental interface in the DOM
  - All node objects extend this interface
- Allows us to
  - Traverse the DOM tree
  - Get information about the Node
    - Node itself and attributes
    - Parent node and child nodes
    - Sibling nodes
  - Modify tree structure: add, remove and update nodes
    - Node itself
    - Child nodes

`Node` Interface Properties

Property	Description
`nodeName`	The name of the node
`nodeValue`	The valud of the node
`nodeType`	The type of the node
`parentNode`	The node that is this node's parent
`childNodes`	A `NodeList` containing all of this node's children
`firstChild`	The first child of this node
`lastChild`	The last child of this node
`previousSibling`	The **node** immediately preceding this node
`nextSibling`	The node immediately following this node
`attributes`	A `NamedNodeMap` containing the attributes of this node
`ownerDocument`	The document to which this node belongs

* Also has properties for namespaces: **`namespaceURI`**, **`prefix`**, ...

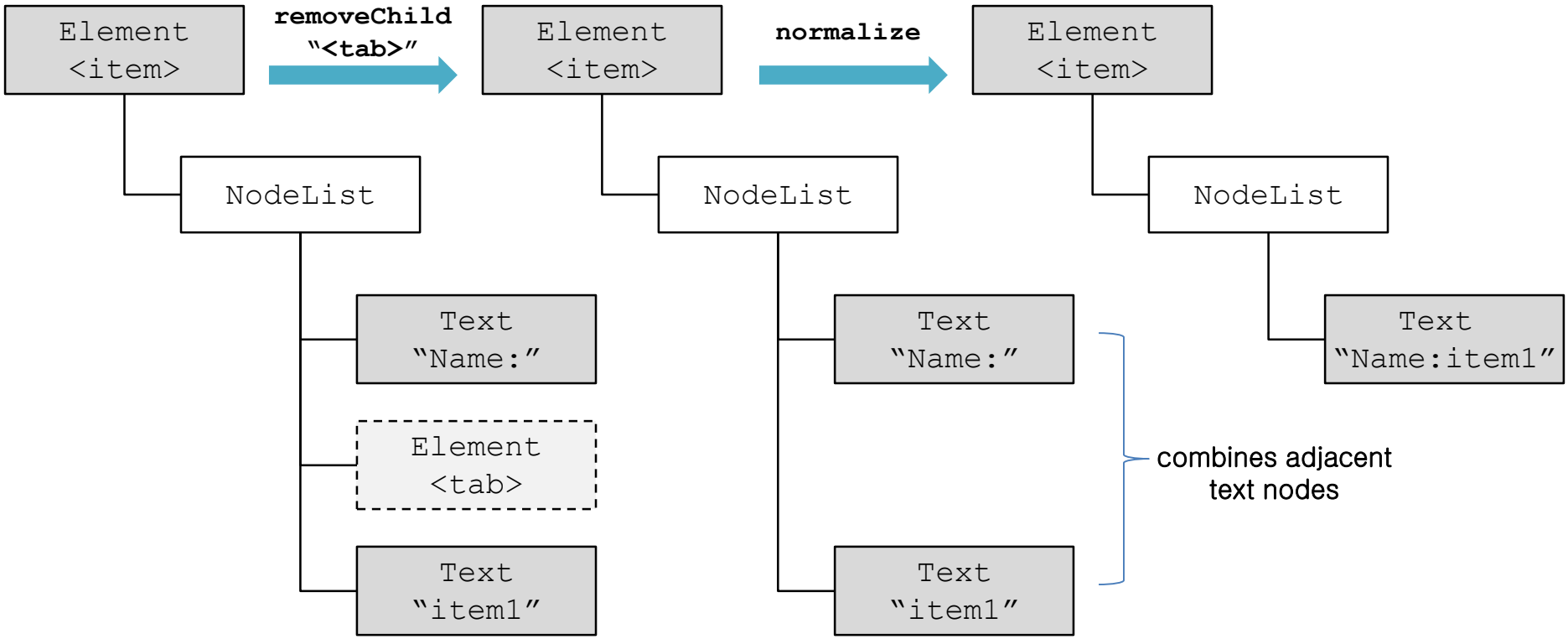
Node Interface Methods

Method	Description
`insertBefore(newChild, refChild)`	Inserts the `newChild` node before the existing `refChild`. If `refChild` is `NULL`, it inserts the **node** at the end of the list
`replaceChild(newChild, oldChild)`	Replaces `oldChild` with `newChild`. Used to update existing records. Return `oldChild`
`removeChild(oldChild)`	Removes `oldChild` from the list. and return it.
`appendChild(newChild)`	Adds `newChild` to the end of the list, and return it.
`hasChildNodes()`	Returns a boolean; `true` if the node has any children, `false` otherwise.
`cloneNode(deep)`	Return a duplicate of this node. If the boolean `deep` parameter is true, this will recursively clone the subtree under the node, otherwise it will only clone the node itself
`Normalize()`	If there are multiple adjacent Text child nodes, this method will combine them again. It doesn't return a value
`isSupported(feature, version)`	Indicates whether this implementation of the DOM supports the feature passed. Returns a boolean, `true` if it supports the feature, `false` otherwise

Node Interface Method normalize

```
<item>
     Name:<tab/>item1
</item>
```

```
<item>
     Name:item1
</item>
```



`Document` Interface

- Represents an entire XML document (***Document Root***)
  - Document root is a conceptual node which owns everything in the document including the ***Root Element***
  - The document root (**`Document`**) node extended from the **`Node`** interface

- We cannot create a DOM tree without first creating the **`Document`** node object
  - **`Document`** provides useful factory methods to create other DOM node objects
  - How to build a DOM tree
    1. Create nodes using factory methods of the `Document` interface
       - E.g., `createElement`, `createAttribute`, `createCDATASection`, ...

       or copy from other documents
       - E.g., `importNode` (cf., `adoptNode` to move)
    2. Add the nodes to appropriate spots using the `Node` interface
       - E.g., `insertBefore`, `appendChild`, `replaceChild`, ...

`DocumentFragment` Interface

- "Lightweight" or "minimal" Document object
  - `Document` object can potentially be a heavy-weight object
  - `DocumentFragment` object represents a fragment of DOM tree
- Has zero or more child nodes
  - Usually contains element nodes, but even just a text node
  - Do NOT even need to be well-formed
- Handy for subtree operations
  - For example, cut and pasting or rearranging a part of document by moving fragments around
  - When a `DocumentFragment` is inserted into a document, the children of the `DocumentFragment` are inserted, not the `DocumentFragment` itself

`NodeList` Interface

- Manage an *ordered* collection of **`Nodes`**
  - Stores child element and text nodes
- There is only one property and one method
  - The **`length`** property stores the number of items
  - The method **`item(unsigned long index)`** returns the ***`index`***th item from the list
- Always "live"
  - If we add some nodes to, and remove nodes from the document, a node list will always reflect those changes immediately

`NamedNodeMap` Interface

- Represent an *unordered* collection of **`Nodes`**
  - Stores attribute nodes
  - Like **`NodeList`**, always live
- Add or remove nodes using their node names
  - **`getNamedItem(DOMString name)`**
  - **`setNamedItem(Node arg)`**
  - **`removeNamedItem(DOMString name)`**
- Sometimes, we also might want to iterate through nodes one by one just like a **`NodeList`**
  - For this reason, **`NamedNodeMap`** provides **`length`** property and an **`item(index)`** method
  - Note that the index does NOT mean the actual order of the item nodes

`Attr` Interface

- Represents an attribute in elements
- Extends the **`Node`** interface
  - Supplies **`name`** and **`value`** properties
    - Have the same values as **`nodeName`** and **`nodeValue`** properties in **`Node`**
    - E.g., in Java, **`attr.getName()`** == **`attr.getNodeName()`**
- However, attributes are NOT direct parts of the tree structure in a document
  - Attributes are NOT children of elements, just properties of the elements
  - Thus, can NOT be traversed with tree interfaces → i.e., accessed through **`getAttributes`** method, not **`getChildNodes`**
  - Can NOT have a parent, sibling nor child → i.e., **`parentNode`**, **`childNodes`**, **`previousSibling`**, **`nextSibling`** properties of **`Attr`** nodes are always **`NULL`**

`Element` Interface

- Represents an element in XML documents
- Inherited from `Node` interface
  - Defines **`tagName`** property and related methods
    - `getElementsByTagName(DOMString name)`
    - `getElementsByTagNameNS(DOMString namespaceURI, DOMString localName)`
- Provide operations for attributes
  - **`getAttribute(DOMString name)`** `// returns value`
  - **`getAttributeNode(DOMString name)`** `// returns` **`Attr`** `node`
    `(getAttributeNS, getAttributeNodeNS)`
  - **`setAttribute(DOMString name, DOMString value)`**
  - **`setAttributeNode(Attr newAttr)`**
    `(setAttributeNS, setAttributeNodeNS)`
  - **`removeAttribute(DOMString name)`**
  - **`removeAttributeNode(Attr oldAttr)`**
    `(removeAttributeNS,` ***`no removeAttributeNodeNS`*** **`)`**

`CharacterData` and `Text` interface

- XML documents involves a lot of work with text
- `CharacterData` interface
  - Has properties and useful methods to work with general text
- `Text` interface
  - Extends `CharacterData`
  - Represents textual content of `Element` or `Attr`
- They cannot have children like `Attr`

`CharacterData` interface methods

Property

- `DOMString data`

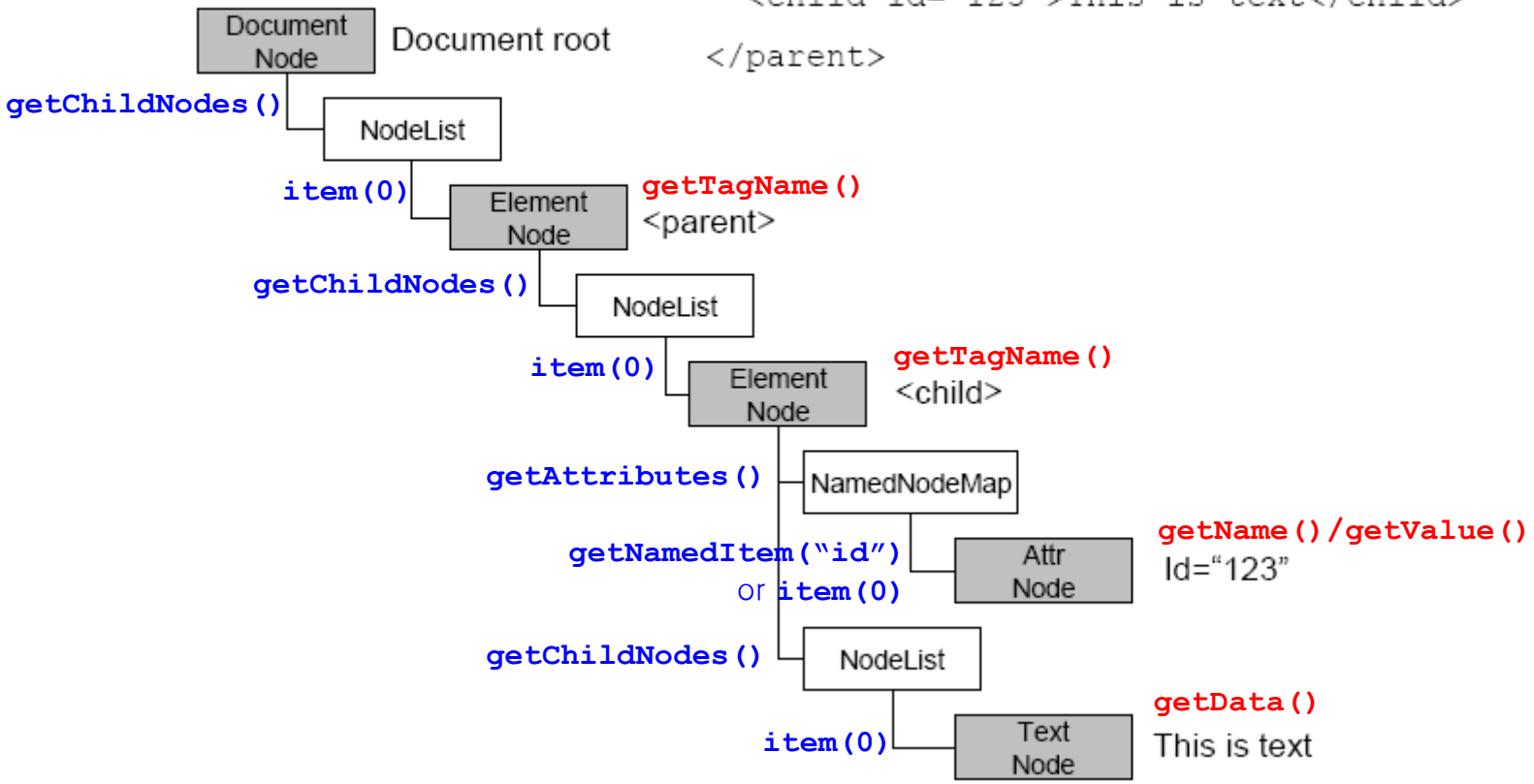
Methods

- `substringData(unsigned long offset,`
  `unsigned long count)`
- `appendData(DOMString arg)`
- `insertData(unsigned long offset,`
  `DOMString arg)`
- `deleteData(unsigned long offset,`
  `unsigned long count)`
- `replaceData(unsigned long offset,`
  `unsigned long count,`
  `DOMString arg)`

DOM Tree Example Revisited (1/3)

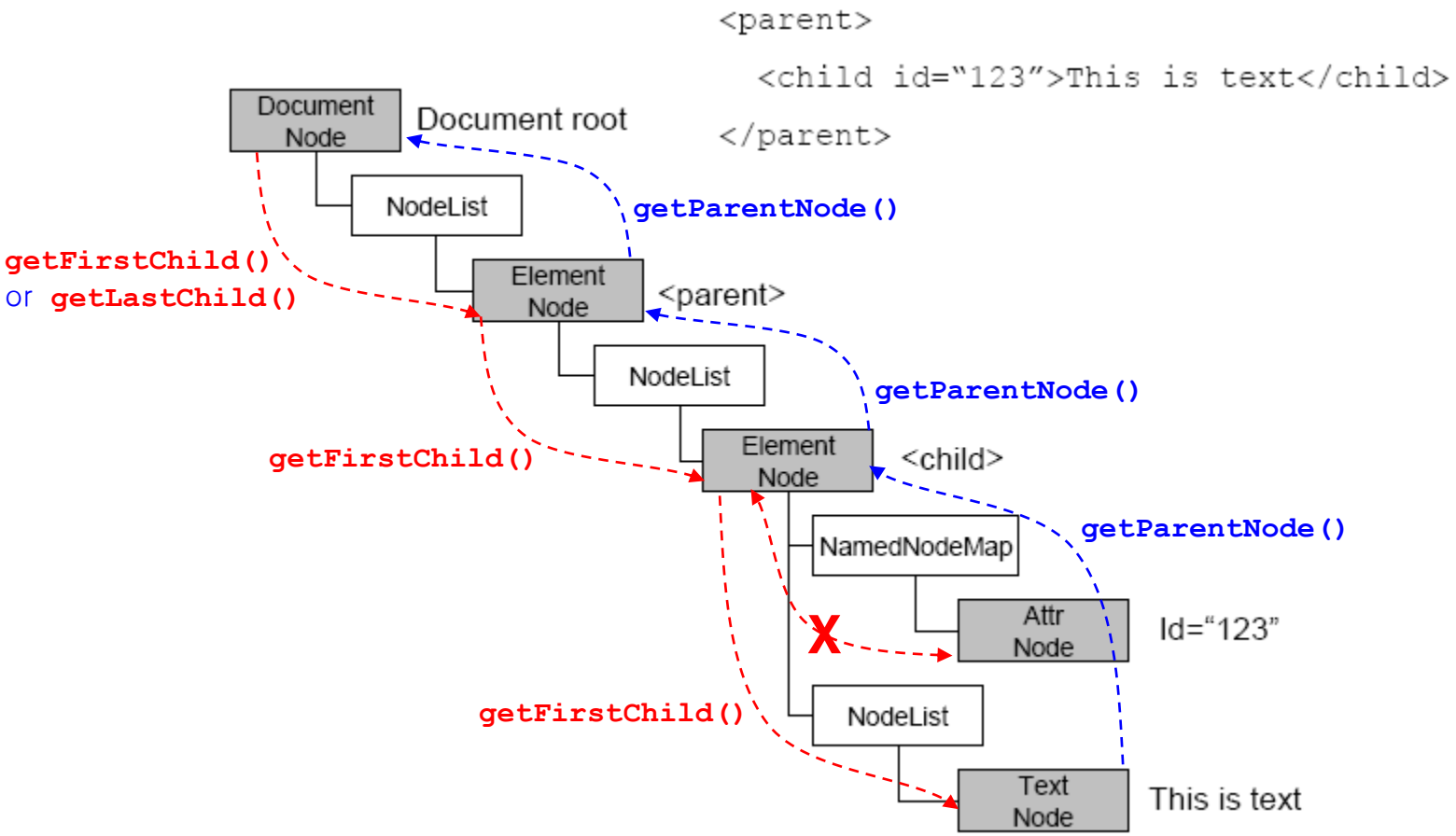
❑ Traversing DOM by item index

```
<parent>
  <child id="123">This is text</child>
</parent>
```



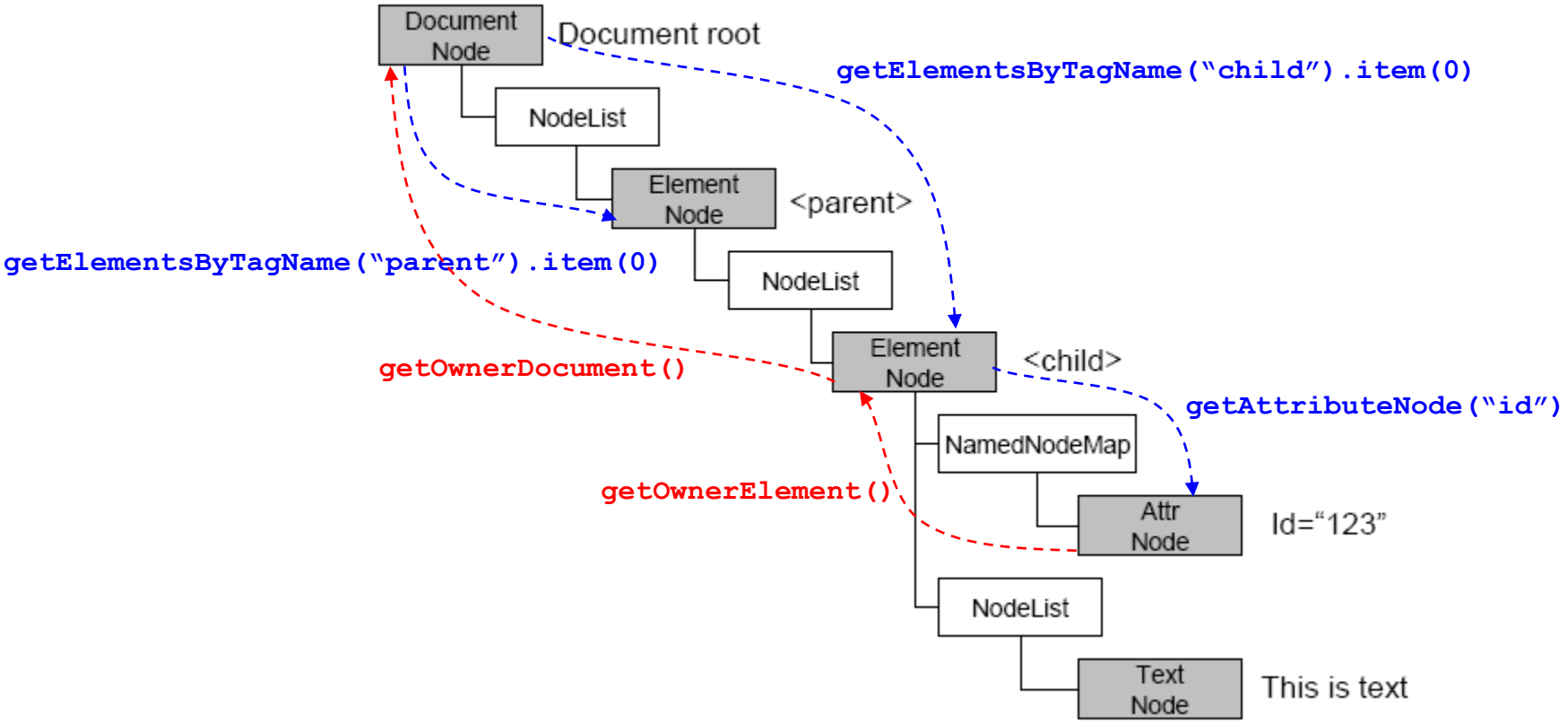
DOM Tree Example Revisited (2/3)

❑ Traversing DOM by parent-child relationship



DOM Tree Example Revisited (3/3)

- Traversing DOM by node name



Xerces

- A part of the Apache XML project
  - Fully-validating XML parsers are available for both Java, C++ and Python (PyXerces)
- Xecers C++
  - Supporting XML 1.0 3rd ED and XML 1.1 1st ED
  - W3C DOM (Level 1, 2, 3) and SAX (version 2) standard
  - XML Schema 1.0 Structures and Datatypes
- Xerces2 Java
  - Supporting XML 1.0 4rd ED and XML 1.1 2nd ED
  - And more support than C++ version
  - Download from http://xerces.apache.org/xerces2-j/download.cgi

❖ Note JavaScript also implements DOM APIs for browsers

Node Count Example (1/2)

```
import org.w3c.dom.*;
import org.apache.xerces.parsers.DOMParser;

    ...

public String countDom(String srcFile) {
    DOMParser dom = new DOMParser();
    dom.parse(srcFile);

    Document document = dom.getDocument();
    count(document);

    StringBuffer msg = new StringBuffer("Statistics for");
    msg.append(srcFile);
    msg.append("\n==================\n");
    msg.append("\nNumber of Element Nodes " + elementNd);
    msg.append("\nNumber of Attributes " + numAtts);
    msg.append("\nNumber of Text Nodes " + textNd);
    return msg.toString();
}

    ...
```

Node Count Example (2/2)

```
private void count(Node nd) {
    NodeList offspring;
    If (nd == null) return;

    int type = nd.getNodeType();
    Switch (type) {
          case Node.DOCUMENT_NODE:
                    offspring = nd.getChildNodes();
                    for (int i =0; i < offspring.getLength(); i++)
                              count(offspring.item(i));
          break;
          case Node.ELEMENT_NODE:
                    elementNd++;                              // count Element Nodes

                    NamedNodeMap atts = nd.getAttributes();
                    if (atts.getLength() > 0) numAtts += atts.getLength();
                                                              // count Attr Nodes
                    offspring = nd.getChildNodes();
                    for (int i = 0; i < offspring.getLength(); i++)
                              count(offspring.item(i));
          break;
          case Node.TEXT_NODE:
                    textNd++;                                 // count Text Nodes
    }
}
```

Part II.
SAX (Simple API for XML)

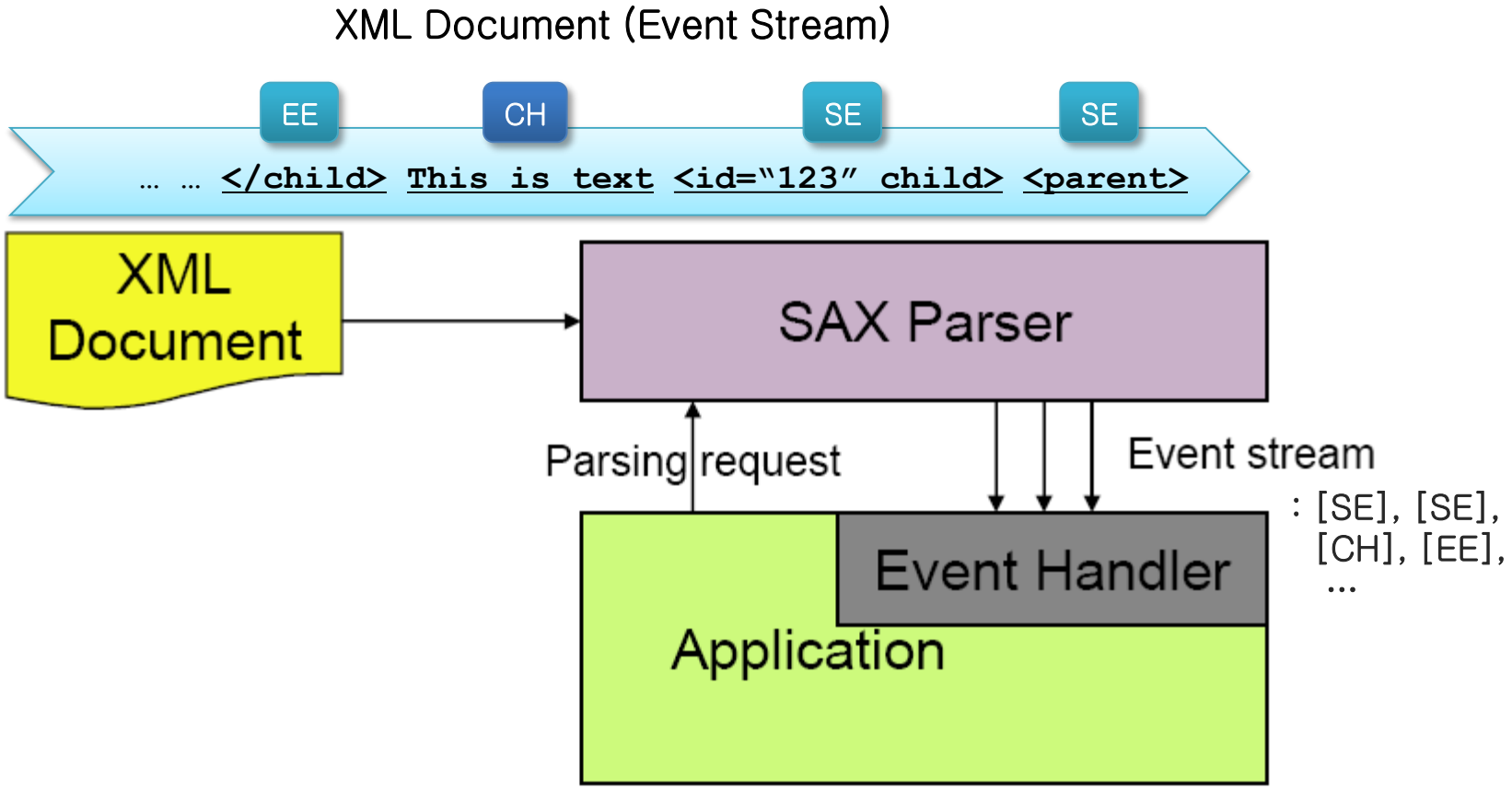
Contents

- What is SAX?
  - How the SAX Works
  - Event-Driven Processing
  - Pros and Cons of SAX
- SAX Code Example Using Xerces
- `XMLFilter` Interfaces

What is SAX?

- Emerged as the alternative to DOM
  - Created by Dave Megginson and his fellow subscribers to an email list called XML-DEV (originally a Java-only API)
  - ***Event-driven*** processing
    - Fast and requires relatively little memory
    - Can be used to process any size of document
- SAX is neither an official standard nor a recommendation of the W3C
  - Nevertheless, SAX became a de facto standard since many XML products support it
- Current version is 2.0.2 (SAX2 R3)

How the SAX works



Event-driven Processing (1/2)

- New paradigm for programming
  - Only processes data in response to events
- If an event is raised, an ***event handler*** for the event is invoked
  - If a handler hasn't been written for the event, that event is ignored
- In a SAX-based program, event handlers can be written for each node type and others
  - Documents, elements, chars, processing instructions
  - Prefix mappings, whitespaces, skipped entities

Event-driven Processing (2/2)

- Each time the parser finds a node, it searches for an appropriate handler
  - If a handler is found → the node is passed to the handler for processing
  - If no handler is found → the node is ignored and the parser moves along to the next node
- Especially useful when only some nodes are interesting, not the entire XML document

Disadvantages of SAX

- It does NOT allow us to randomly traverse the entire document structure (e.g., DOM tree)
- It is NOT adept at repeatedly moving backwards and forwards through the document
  - We only get one shot of the document as the event stream goes forward through the parser

Choosing Between SAX and DOM

❑ When the SAX is more helpful than the DOM

- ➢ Handling large XML documents
- ➢ Document stream processing
  - o We can escape from SAX processing during any event handler
  - o Supports `XMLFilter` interface

❑ The areas where the DOM is more adept

- ➢ Frequently changing document structure
  - o SAX is considered to be read-only
- ➢ Supports random document access through a tree structure
  - o SAX supports only serial access through a document stream

SAX Parsers

Parser Name	Creator	SAX version Supported	Language(s) supported
Aelfred	David Megginson	2.0	Java
Saxon	Michael Kay	2.0	Java
MSXML3	Microsoft	2.0	C++, VB and any COM-compliant language
Xerces C++ Parser	Apache XML Project	2.0	C++
Xerces Java	Apache	2.0	Java
JAXP	Sun	2.0	Java
XP	James Clark	1.0	Java

Simple SAX application using Xerces2 Java

❑ Basic interface and helper class

- ➢ Interface **`XMLReader`**
  - o The interface for reading an XML document using callbacks
- ➢ Interface **`ContentHandler`**
  - o The interface to receive notification of the logical content of a document
- ➢ Interface **`ErrorHandler`**
  - o The basic interface for SAX error handlers
- ➢ Class **`DefaultHandler`**
  - o Default implementation of `ContentHandler` and `ErrorHandler`

❑ How to develop a SAX application

1) Define a main class extending **`DefaultHandler`**
2) Create a SAX parser and access the parser with **`XMLReader`** interface
   - ① Register the class (`this`) as the content and error handler of the parser
3) Override methods of **`DefaultHandler`**
   - ① Override methods from `ContentHandler` to handle events
   - ② Override methods from `ErrorHandler` to handle errors

`XMLReader` Interface

- **`XMLReader`** is the interface that every SAX parser must implement
  - **`XMLReader`** implementation in Xerces2 Java
    - `org.apache.xerces.parsers.SAXParser`
- Allows applications to
  - Set and query feature flags and properties in the parser
    - To change parser behavior
  - Register event handlers for document processing
    - Content handlers
    - Error handlers
  - Initiate document parsing

`XMLReader` Interface Methods

- `getFeature(String name)` and `setFeature(String name, boolean value)`
  - E.g., `setFeature("http://xml.org/sax/features/validation", true)`
- `getProperty(String name)` and `setProperty(String name, Object value)`
  - E.g., `setProperty("http://xml.org/sax/properties/document-xml-version", "1.0")`
- `getContentHandler()` and `setContentHandler( … )`
- `getErrorHandler()` and `setErrorHandler( … )`
- `parse(String systemid)`
- And more …

`ContentHandler` Interface

- Main interface in the `org.xml.sax` package
- Used to receive events which are triggered by the structure of the XML document
  - Then an appropriate method of the content handler is invoked (callback)
  - E.g., when start tag of an element is read → `startElement` method is invoked
- Must be registered to the parser so that the parser knows where to pass events
  - By the `XMLReader.setContentHandler` method

`ContentHandler` Methods (1/2)

- `void startDocument()`
  - Handles the start of a document
- `void endDocument()`
  - Handles the end of the document
- `void startElement(String namespaceURI, String localName, String qName, Attributes atts)`
  - Handles the start of an element
- `void endElement(String namespaceURI, String localName, String qName)`
  - Handles tags that close elements
- `void characters(char[] ch, int start, int length)`
  - Handles character nodes
- `void processingInstruction(String target, String data)`
  - Handles processing instruction nodes (except XML declaration)
- And more ...

ContentHandler Methods (2/2)

void	characters(char[] ch, int start, int length) Receive notification of character data.
void	endDocument() Receive notification of the end of a document.
void	endElement(String uri, String localName, String qName) Receive notification of the end of an element.
void	endPrefixMapping(String prefix) End the scope of a prefix-URI mapping.
void	ignorableWhitespace(char[] ch, int start, int length) Receive notification of ignorable whitespace in element content.
void	processingInstruction(String target, String data) Receive notification of a processing instruction.
void	skippedEntity(String name) Receive notification of a skipped entity.
void	startDocument() Receive notification of the beginning of a document.
void	startElement(String uri, String localName, String qName, Attributes atts) Receive notification of the beginning of an element.
void	startPrefixMapping(String prefix, String uri) Begin the scope of a prefix-URI Namespace mapping.

Attributes Interface

- This interface allows access to a list of attributes
  - Passed to the **`startElement`** method when the method is invoked
- Supported Methods
  - **`getIndex(String qName)`**
  - **`getLength()`**
    - Returns the number of attributes in the list
  - **`getQName(int index)`**
    - Looks up an attribute's qualified (prefixed) name by index
  - **`getType(int index), getType(String qName)`**
    - Returns the attribute type as a string
    - E.g., `"CDATA"`, `"ID"`, `"IDREF"`, `"IDREFS"`, `"NMTOKEN"`, ...
  - **`getValue(int index), getValue(String qName)`**
  - And more **...**

`ErrorHandler` Interface

- Used for customizing the way applications handle errors
  - Although this interface is not required, the behavior of the application may be unpredictable if it is not implemented
- Must be registered with the parser
  - By the **`XMLReader.setErrorHandler`** method
- The following methods are available
  - **`void warning(SAXParseException exception)`**
    - Receive notification of a warning (default: no action)
  - **`void error(SAXParserException exception)`**
    - Receive notification of a recoverable error (default: no action)
    - E.g., violation of a validity constraint
  - **`void fatalError(SAXParseException exception)`**
    - Receive notification of a non-recoverable error (stop parsing)
    - E.g., violation of a well-formedness constraint

SAX Example using Xerces

```
import org.xml.sax.XMLReader;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;

public class MyApp extends DefaultHandler {
   public static void main(String args[]) throws SAXException {
      try {
                ...
         XMLReader parser = XMLReaderFactory.createXMLReader(
                                  "org.apache.xerces.parsers.SAXParser");
         parser.setContentHandler(this);
         parser.setErrorHandler(this);
                ...
         parser.parse(srcFile);
      } catch (SAXParseException spe) {
                ...
      } catch (SAXException se) {
                ...
      } catch (Exception e) {
                ...
      }
   }
}
```

SAX Example using Xerces: Handling Events

```
public void startDocument()
{
   system.out.println("start document");
}

public void endDocument()
{
   system.out.println("end document");
}

public void startElement(String uri, String name, String qName,
                         Attributes atts)
{
   system.out.println("start element: {" + uri + "}" + name);
}

public void endElement(String uri, String name, String qName)
{
   system.out.println("end element: " + qName);
}
        ...
```

SAX Example using Xerces: Handling Errors

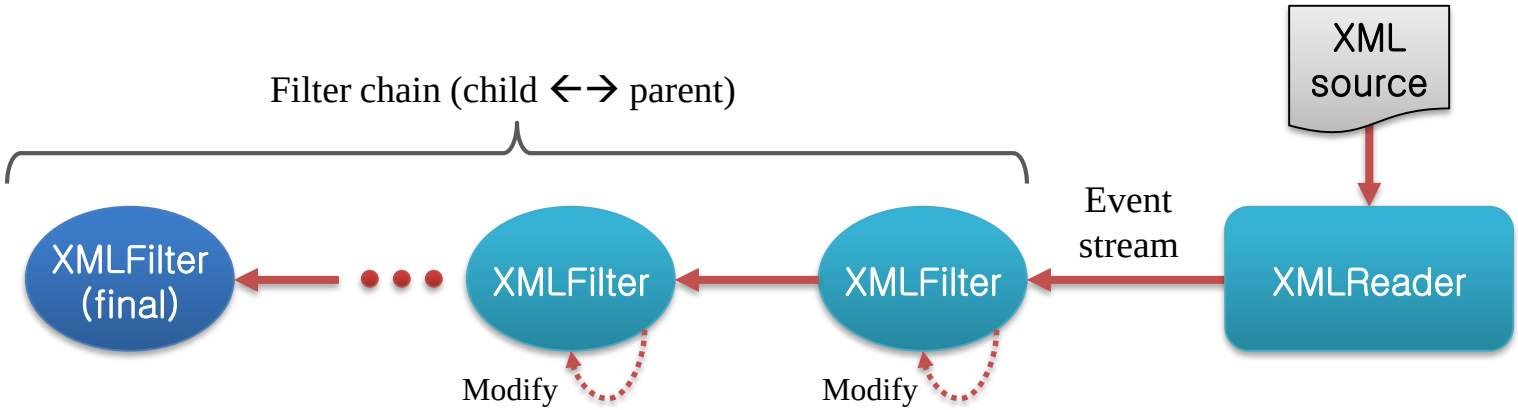
```
public void fatalError(SAXParseException spe)
{
   system.err.println("Fatal error has occurred: " + spe);
}

public void error(SAXParseException spe)
{
   system.err.println("a parse error has occurred: " + spe);
}

public void warning(SAXParseException spe)
{
   system.err.println("Parse warning were issued: " + spe);
}
```

`XMLFilter` Interface

- Inherited from **`XMLReader`** (SAX 2)
  - Only two methods are added
    - `getParent()`
    - `setParent(XMLReader parent)`
- **`XMLFilter`** is similar to **`XMLReader`**, except
  - It obtains events from another **`XMLReader`** (parent) rather than an XML document
- **`XMLFilter`** may modify an event stream as they are passed to the final application (called "filter chain")



References

- DOM
  - http://www.w3.org/DOM/
- SAX
  - http://www.saxproject.org/
- Xecers
  - http://xerces.apache.org/