XML Programming - DOM

Andy Clark

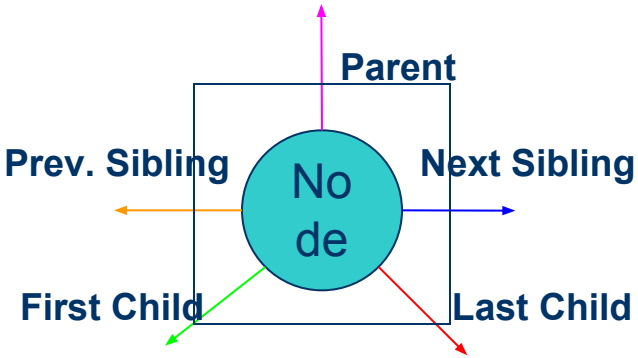
17 Apr 2002

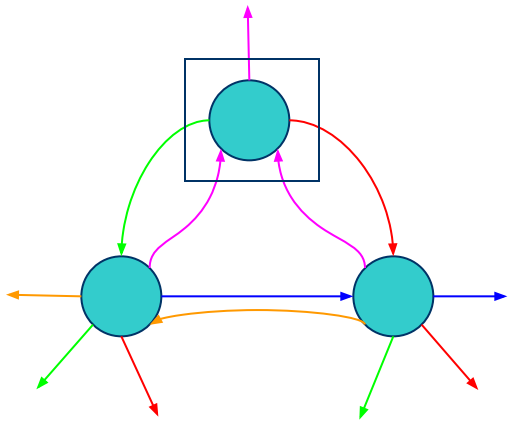
DOM Design Premise

- Derived from browser document model
- Defined in IDL
  - Lowest common denominator programming lang.
    - Most methods packed into base Node object
    - Re-invents basic collection interfaces
- Factory model
  - Allows custom classes to be used for nodes
- Separation between required and optional modules

Document Object Model

- Generic tree model
  - Node
    - Type, name, value
    - Attributes
    - Parent node
    - Previous, next sibling nodes
    - First, last child nodes
  - “Live” Collections
    - Lists
    - Maps





Node Interfaces

- Package org.w3c.dom
  - Document
  - Element, Attr, Text
  - ProcessingInstruction, Comment
  - EntityReference, CDATASection
  - DocumentType, Notation, Entity
  - DocumentFragment

  - *Note:* All nodes descend from org.w3c.dom.Node.

org.w3c.dom.Node (1 of 4)

- Node information
  - short getNodeType();
    - e.g. Node.ELEMENT_NODE
  - String getNodeName();
  - String getNodeValue();
- Namespace information
  - String getPrefix();
  - String getLocalName();
  - String getNamespaceURI();

org.w3c.dom.Node (2 of 4)

- Attributes
  - boolean hasAttribute(String name);
  - NamedNodeMap getAttributes();
- Children
  - Node getFirstChild();
  - Node getLastChild();
  - boolean hasChildNodes();
  - NodeList getChildNodes();

org.w3c.dom.Node (3 of 4)

- Links to other nodes
  - Document getOwnerDocument();
  - Node getParentNode();
  - Node getPreviousSibling();
  - Node getNextSibling();
- Other methods
  - Node cloneNode(boolean deep);

org.w3c.dom.Node (4 of 4)

- Editing node values
  - void setNodeValue(String value);
  - void setPrefix(String prefix);
- Editing tree structure
  - Node appendChild(Node child);
  - Node insertBefore(Node newChild, Node refChild);
  - Node removeChild(Node child);
  - Node replaceChild(Node newChild, Node oldChild);

org.w3c.dom.NodeList

- Methods
  - int getLength();
  - Node item(int index);

org.w3c.dom.NamedNodeMap (1 of 2)

- Query methods
  - int getLength();
  - Node item(int index);
  - Node getNamedItem(String name);
  - Node getNamedItemNS(String namespaceURI, String localName);

org.w3c.dom.NamedNodeMap (2 of 2)

- Editing methods
  - Node setNamedItem(Node node);
  - Node setNamedItemNS(Node node);
  - Node removeNamedItem(String name);
  - Node removeNamedItemNS(String namespaceURI, String localName);

Parsing a Document

- Instantiate parser
  - org.apache.xerces.parsers.DOMParser parser = new org.apache.xerces.parsers.DOMParser();
- Parse file and query document
  - parser.parse("document.xml");
  - org.w3c.dom.Document document = parser.getDocument();
  - *Note:* Should use JAXP to instantiate parser.

Traversing a Document (1 of 8)

- Recursively: method #1

```
01    public void traverse(Node node) {
02       System.out.println("node: "+node.getNodeName());
03       if (node.hasChildNodes()) {
04          NodeList children = node.getChildNodes();
05          for (int i = 0; i < children.getLength(); i++) {
06             Node child = children.item(i);
07             traverse(child);
08          }
09       }
10    }
```

Traversing a Document (2 of 8)

- Recursively: method #1

```
01    public void traverse(Node node) {
02       System.out.println("node: "+node.getNodeName());
03       if (node.hasChildNodes()) {
04          NodeList children = node.getChildNodes();
05          for (int i = 0; i < children.getLength(); i++) {
06             Node child = children.item(i);
07             traverse(child);
08          }
09       }
10    }
```

Traversing a Document (3 of 8)

- Recursively: method #1

```
01    public void traverse(Node node) {
02       System.out.println("node: "+node.getNodeName());
03       if (node.hasChildNodes()) {
04          NodeList children = node.getChildNodes();
05          for (int i = 0; i < children.getLength(); i++) {
06             Node child = children.item(i);
07             traverse(child);
08          }
09       }
10    }
```

Traversing a Document (4 of 8)

- Recursively: method #1

```
01    public void traverse(Node node) {
02       System.out.println("node: "+node.getNodeName());
03       if (node.hasChildNodes()) {
04          NodeList children = node.getChildNodes();
05          for (int i = 0; i < children.getLength(); i++) {
06             Node child = children.item(i);
07             traverse(child);
08          }
09       }
10    }
```

  - *Note:* Avoid calling geLength() in loop *but* remember that DOM collections are live!

Traversing a Document (5 of 8)

- Recursively: method #2

```
01    public void traverse(Node node) {
02       System.out.println("node: "+node.getNodeName());
03       Node child = node.getFirstChild();
04       while (child != null) {
05          traverse(child);
06          child = child.getNextSibling();
07       }
08    }
```

Traversing a Document (6 of 8)

- Recursively: method #2

```
01    public void traverse(Node node) {
02       System.out.println("node: "+node.getNodeName());
03       Node child = node.getFirstChild();
04       while (child != null) {
05          traverse(child);
06          child = child.getNextSibling();
07       }
08    }
```

Traversing a Document (7 of 8)

- Recursively: method #2

```
01    public void traverse(Node node) {
02       System.out.println("node: "+node.getNodeName());
03       Node child = node.getFirstChild();
04       while (child != null) {
05          traverse(child);
06          child = child.getNextSibling();
07       }
08    }
```

Traversing a Document (8 of 8)

- Recursively: method #2

```
01    public void traverse(Node node) {
02       System.out.println("node: "+node.getNodeName());
03       Node child = node.getFirstChild();
04       while (child != null) {
05          traverse(child);
06          child = child.getNextSibling();
07       }
08    }
```

  - *Note:* Be sure to call nextSibling() on the *child* object, **not** the *node* object.

org.w3c.dom.Document (1 of 3)

- Extends org.w3c.dom.Node
- Query methods
  - DocumentType getDoctype();
  - Element getDocumentElement();
  - Element getElementById(String id);
  - NodeList getElementsByTagName(String name);
  - NodeList getElementsByTagNameNS(String nsURI, String local);

org.w3c.dom.Document (2 of 3)

- Important factory methods
  - Element createElement(String name);
  - Element createElementNS(String namespaceURI, String qualifiedName);
  - Attr createAttribute(String name);
  - Attr createAttributeNS(String namespaceURI, String qualifiedName);
  - Text createTextNode(String data);

org.w3c.dom.Document (3 of 3)

- Other factory methods
  - ProcessingInstruction createProcessingInstruction(String target, String data);
  - Comment createComment(String data);
  - CDATASection createCDATASection(String data);
  - EntityReference createEntityReference(String name);
  - DocumentFragment createDocumentFragment();
- Missing factory methods
  - DocumentType, Notation, Entity

Building a Tree Programmatically

- Instantiate Document object
  - org.w3c.dom.Document document = new org.apache.xerces.dom.DocumentImpl();
- Create content
  - Element root = document.createElement("root");
  - document.appendChild(root);
  - Text text = document.createTextNode("text");
  - root.appendChild(text);

  - *Note:* Should use JAXP to instantiate document.

Searching for Elements (1 of 4)

- Document#getElementsByTagName

```
01      public void iteratePeople(Document document) {
02          NodeList people = document.getElementsByTagName("person");
03          int length = people.getLength();
04          for (int i = 0; i < length; i++) {
05              Node person = people.item(i);
06              // do something with <person>
07          }
08      }
```

Searching for Elements (2 of 4)

- Document#getElementsByTagName

```
01  public void iteratePeople(Document document) {
02      NodeList people = document.getElementsByTagName("person");
03      int length = people.getLength();
04      for (int i = 0; i < length; i++) {
05          Node person = people.item(i);
06          // do something with <person>
07      }
08  }
```

Searching for Elements (3 of 4)

- Document#getElementsByTagName

```
01      public void iteratePeople(Document document) {
02          NodeList people = document.getElementsByTagName("person");
03          int length = people.getLength();
04          for (int i = 0; i < length; i++) {
05              Node person = people.item(i);
06              // do something with <person>
07          }
08      }
```

Searching for Elements (4 of 4)

- Document#getElementsByTagName

```
01      public void iteratePeople(Document document) {
02          NodeList people = document.getElementsByTagName("person");
03          int length = people.getLength();
04          for (int i = 0; i < length; i++) {
05              Node person = people.item(i);
06              // do something with <person>
07          }
08      }
```

Importing Nodes (1 of 2)

- The problem
  - Moving nodes from different documents, *even if same implementation,* throws a DOM exception
  - Why?
    - Factory design pattern allows different implementations and custom node implementations
    - Therefore, cannot simply move nodes to another tree
- The solution
  - Document#importNode
  - Document#adoptNode – DOM Level 3 working draft

Importing Nodes (2 of 2)

- Document#importNode
  - WRONG!

```
01    Document doc1, doc2;
02    Node node1, parent2;
03    parent2.appendChild(node1);
```

  - RIGHT

```
01    Document doc1, doc2;
02    Node node1, parent2;
03    Node node2 = doc2.importNode(node1);
04    parent2.appendChild(node2);
```

Common Reactions (1 of 2)

- Why use interfaces? Classes would be better!
  - Factory design pattern
    - Removes dependency on specific implementation
    - Allows multiple implementations
    - Allows factory to return custom data objects
- Why are all these methods on Node interface?
  - Other languages don't have RTTI
  - Avoids type casting (no, the cast is *not* free)

Common Reactions (2 of 2)

- DOM sucks! “xyz” is so much better!
  - Cons, actual and perceived
    - Designed by committee
    - Steeper learning curve than “xyz”
    - Not as easy to use as “xyz”
  - Pros
    - Allows custom implementations
    - Standard tree model for XML documents

Advanced Topics – for another time

- DOM Level 2
  - Events
  - Traversal
    - Tree walkers and iterators
  - Ranges
    - Selection
- DOM Level 3 – working draft
  - Load and Save / Abstract Schemas
  - XPath

Useful Links

- DOM Level 1 Specification
  - http://www.w3.org/TR/REC-DOM-Level-1
- DOM Level 2 Specifications
  - http://www.w3.org/TR/DOM-Level-2-Core
  - http://www.w3.org/TR/DOM-Level-2-Views
  - http://www.w3.org/TR/DOM-Level-2-Events
  - http://www.w3.org/TR/DOM-Level-2-Style
  - http://www.w3.org/TR/DOM-Level-2-HTML

XML Programming: DOM

Andy Clark