XPath Basics

Mikael Fernandus Simalango

Agenda

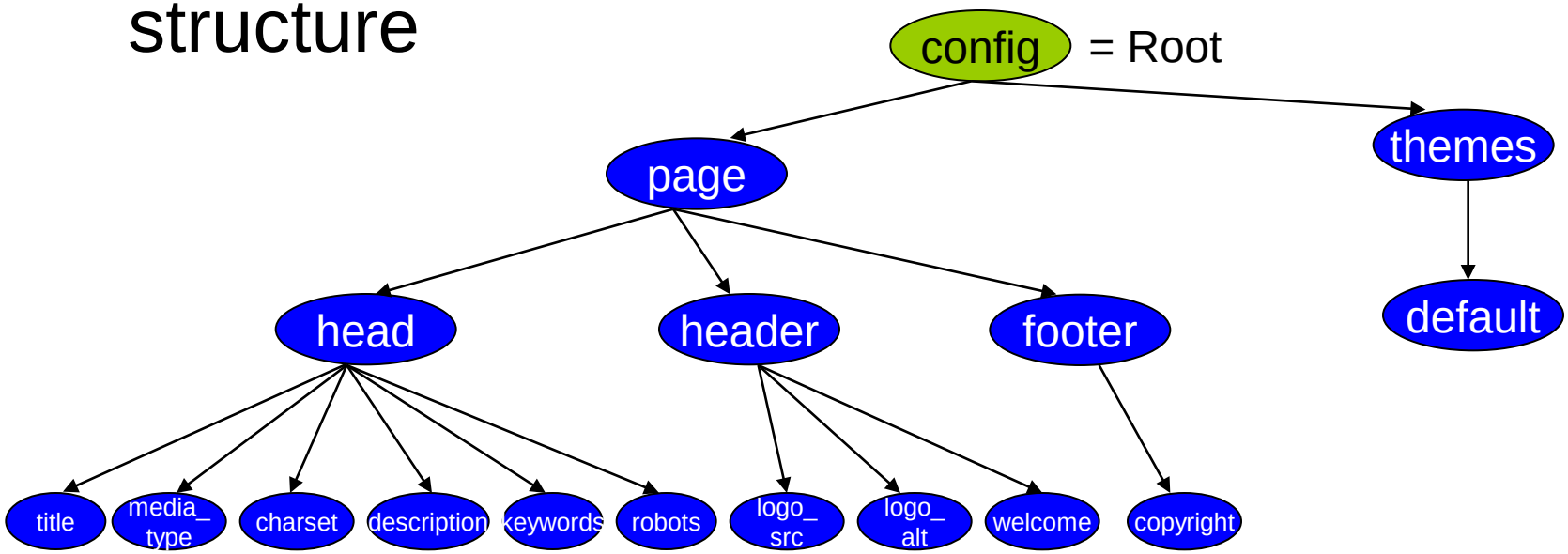
- XML Overview
- XPath Basics
- XPath Sample Project

XML Overview

- eXtensible Markup Language
- Constituted by elements identified by tags and attributes within
- Elements are arranged in nested form
- May or may not contain DTD or schema

XML Overview

- Standard XML file can be represented in tree structure



- A tree is composed of nodes

XML Overview

- Types of nodes:
  - Root node -> root of the tree
  - Element node -> a representation of an element in the document
  - Text node -> formed by character data (<![CDATA[<]]>)
  - Attribute node -> formed by attributes in an element
  - Namespace node ->formed by xmlns attribute
  - Processing instruction node -> formed by processing attribute (eg: src, href, etc)
  - Comment node -> formed by comment in the document (<!- - … -->)

Agenda

- XML Overview
- XPath Basics
- XPath Sample Project

XPath

- A W3C recommendation used to address/refer to/point to/match parts in an XML document
- Current standards: http://www.w3.org/TR/xpath
- What XPath contains:
  - Location path
  - Expressions
  - Functions

XPath

- Location Path:
  - Absolute location path: path measured from root node (location starts from /, and separated by /)
  - Relative location path: path measured from certain location/context node (location separated by /)
- Location step:
  - Axis -> specifies the tree relationship between the nodes selected by the location step and the context node
  - Node test -> specifies node type
  - [Predicate] -> refines the selection of nodes

XPath

- Some important axes:
  - **child** -> child of context node (default axis)
  - **parent** -> parent of context node
  - **self** -> context node itself
  - **attribute** -> attribute of context node
  - **following-sibling** -> following sibling of context node
  - **preceding-sibling** -> preceding sibling of context node

XPath

- Node tests:
  - * -> selects any node of the principal node
  - text() -> selects text node only
  - x::y -> selects y in principal node type x
- Examples:
  - attribute::src -> selects **src** attribute of context node
  - child::text() -> selects text node children of context node
  - attribute::* -> selects all attributes of context node

XPath

- Predicates:
  - Use notation: [predicate_expression]
  - predicate_expression = expressions

XPath

- Some abbreviated syntaxes:
  - * : select all element children of context node
  - @foo : select **foo** attribute of context node
  - foo: select **foo** element children of context node
  - foo[1]: select the first **foo** child of context node
  - foo[@bar="now"]: select **foo** children of context node which have **bar** attribute with value **now**
  - /doc/chapter[7]/section[1]: select the first section of seventh chapter of the doc
  - //foo: select all **foo** descendants of document root

XPath

- Some of the expressions:
  - Or expression (… or …)
  - And expression (… and …)
  - Equality expression (.. = .., .. != ..)
  - Relational expression (.. < .., .. > .., .. <= .., .. >= ..)
  - Numerical expression (.. + .., .. - .., .. * .., .. div .., .. mod ..)

XPath

- Some of the functions
  - Node set functions:
    - *number* last() -> returns a number equal to the context size of the evaluation
    - *number* position() -> returns a number equal to the context position of the evaluation
    - *number* count(node-set) -> returns a number of nodes in argument node-set
    - *node-set* id(object) -> selects elements by their unique ID

XPath

- Some of the functions
  - String functions:
    - *string* string(object) -> converts an object int a string
    - *string* concat(string,string,..) -> returns concatenated strings in arguments
    - *boolean* contains(string,string) -> checks if the first argument string contains the second argument string
    - *string* substring(string, number1, number2) -> returns substring of first argument starting from **number1** with length **number2**

XPath

- Some of the functions
  - Boolean functions:
    - *boolean* boolean(object): converts the argument to a boolean
    - *boolean* not(boolean): returns true if argument is false, and false otherwise
    - *boolean* true(): returns true
    - *boolean* false(): returns false

XPath

- Some of the functions
  - Number functions:
    - *number* number(object) : converts argument into a number
    - *number* floor(number): returns the largest integer that is not greater than the argument
    - *number* ceiling(number): returns the smallest integer that is not less than the argument
    - *number* round(number): returns the closest integer to the argument

XPath

- XPath 1.0 vs XPath 2.0

Item	XPath 1.0	XPath 2.0
Return type	Node-sets (unordered)	Sequences (ordered)
Data types	Node-sets, string, booleans, numbers	XML Schema built-in data types + user-defined
Function set	Data manipulation	Functions in XPath 1.0 + extras (e.g. math, sequence comparison)
Data sources	Single source	Multiple sources in single query

XPath

- More example:

```
<bank>
      <account>
            <account-number>A-101</account-number>
            <branch-name>Downtown </branch-name>
            <balance>500</balance>
      </account>
      <depositor>
          <account-number>A-101</account-number>
          <customer-name>Johnson</customer-name>
      </depositor>
 </bank>
```

- XPath expressions:
  - /bank/account[balance>300] ->get account elements with a balance value greater than 300
  - /bank/depositor[account-number=“A-101”]/customer-name -> get customer name of depositor with account number A-101

Implementations

- XPath is implemented as libraries:
  - In Java 5+: javax.xml.xpath
  - Another library for java:
    - Jaxen (http://jaxen.codehaus.org)
    - Xalan (http://xml.apache.org/xalan-j/)
    - XmlBeans (http://xmlbeans.apache.org/)
  - Libraries also available in other programming languages

Agenda

- XML Overview
- XPath Basics
- XPath Sample Project

XPath Implementation in Java

- Using javax.xml.xpath.*

```
package experiment.xpath;

import org.w3c.dom.*;
import javax.xml.xpath.*;
import javax.xml.parsers.*;
import java.io.IOException;
import org.xml.sax.SAXException;
```

XPath Implementation in Java

- Using javax.xml.xpath.* (cont'd)

```
/**
 * Parse an XML document into DOM representation
 */
protected static Document createDomObject(String filename) {
   Document domRep = null;
   DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
   domFactory.setNamespaceAware(true);
   try {
      DocumentBuilder builder = domFactory.newDocumentBuilder();
      domRep = builder.parse(filename);
   } catch (ParserConfigurationException pe) {
      pe.printStackTrace();
   } catch (IOException ie) {
      ie.printStackTrace();
   } catch (SAXException se) {
      se.printStackTrace();
   }
   return domRep;
}
```

XPath Implementation in Java

- Using javax.xml.xpath.* (cont'd)

```
/**
 * Evaluate an XPath Expression
 * @return nodes
 */
public static NodeList getNodeListByXpath (String expression,String filename) {
   Object res = null;
   XPath xpath = XPathFactory.newInstance().newXPath();
   try {
      XPathExpression expr = xpath.compile(expression);
      res = expr.evaluate(createDomObject(filename), XPathConstants.NODESET);
   } catch(Exception e) {
      e.printStackTrace();
   }
   NodeList nodes = (NodeList) res;
   return nodes;
}
```

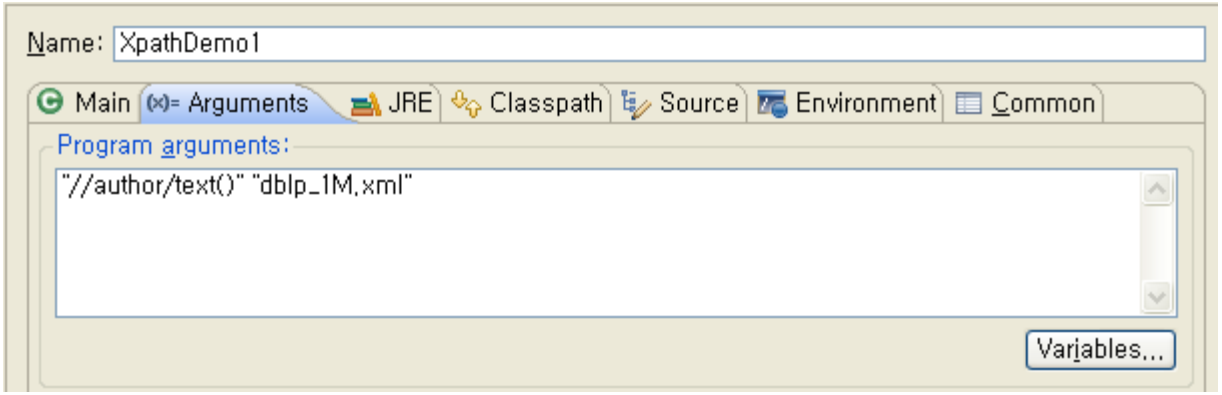
XPath Implementation in Java

- Using javax.xml.xpath.* (cont'd)

```
public static void main(String[] args) throws IOException{
   long startTime;
   if(args.length != 2) {
      System.out.println("Usage: java -c XPathDemo1 xpath_expression source_file");
      System.exit(0);
   }
   String ex = args[0];
   String srcFile = args[1];
   if(args[0].contains("/text()") == false) {
      System.out.println("This demo will only evaluate */text() pattern");
      System.exit(0);
   }
   startTime = System.currentTimeMillis();
   nodes = getNodeListByXpath(ex,srcFile);
   if(nodes instanceof NodeList) {
      for (int i =0; i< nodes.getLength(); i++) {
         System.out.println("Value: "  + nodes.item(i).getNodeValue());
      }
   }
   else {
      System.out.println("Expression yields no result");
   }
   System.out.println("Elapsed time:" +
     (System.currentTimeMillis() - startTime) + " ms");
}
```

XPath Implementation in Java

- Using javax.xml.xpath.* (cont'd)



XPath Implementation in Java

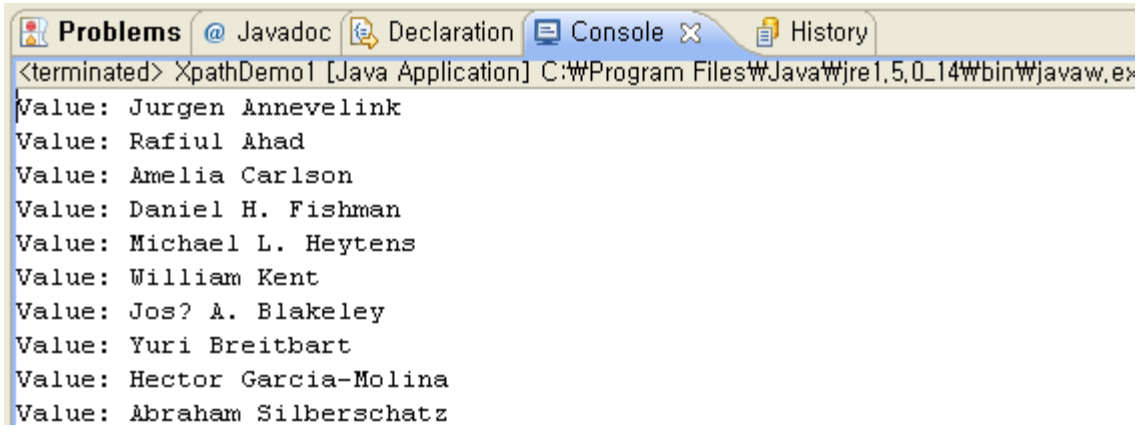
- Using javax.xml.xpath.* (cont'd)

XpathDemo1.java | XQueryDemo1.java | dblp_1M.xml

```
<?xml version="1.0" ?><dblp>
<incollection mdate="2002-01-03" key="books/acm/kim95/AnnevelinkACFHK95">
<author>Jurgen Annevelink</author>
<author>Rafiul Ahad</author>
<author>Amelia Carlson</author>
<author>Daniel H. Fishman</author>
<author>Michael L. Heytens</author>
<author>William Kent</author>
<title>Object SQL - A Language for the Design and Implementation of Object Datab
<pages>42-68</pages>
<year>1995</year>
<booktitle>Modern Database Systems</booktitle>
<url>db/books/collections/kim95.html#AnnevelinkACFHK95</url>
</incollection>
<incollection mdate="2002-01-03" key="books/acm/kim95/Blakeley95">
<author>José A. Blakeley</author>
<title>OQL[C++]: Extending C++ with an Object Query Capability.</title>
<pages>69-88</pages>
<booktitle>Modern Database Systems</booktitle>
<url>db/books/collections/kim95.html#Blakeley95</url>
<year>1995</year>
</incollection>
<incollection mdate="2004-03-08" key="books/acm/kim95/BreitbartGS95">
<author>Yuri Breitbart</author>
<author>Hector Garcia-Molina</author>
<author>Abraham Silberschatz</author>
<title>Transaction Management in Multidatabase Systems.</title>
<pages>573-591</pages>
<booktitle>Modern Database Systems</booktitle>
<crossref>books/acm/Kim95</crossref>
<url>db/books/collections/kim95.html#BreitbartGS95</url>
<year>1995</year>
</incollection>
```

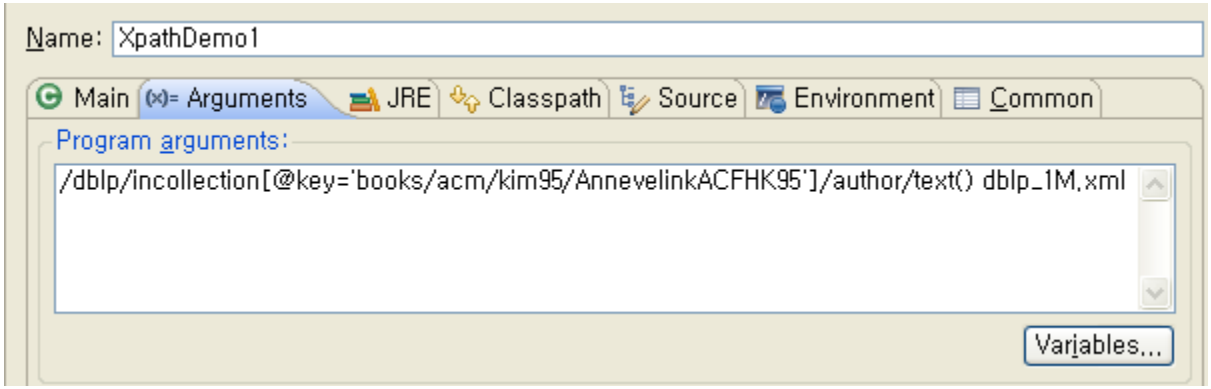
XPath Implementation in Java

- Using javax.xml.xpath.* (cont'd)



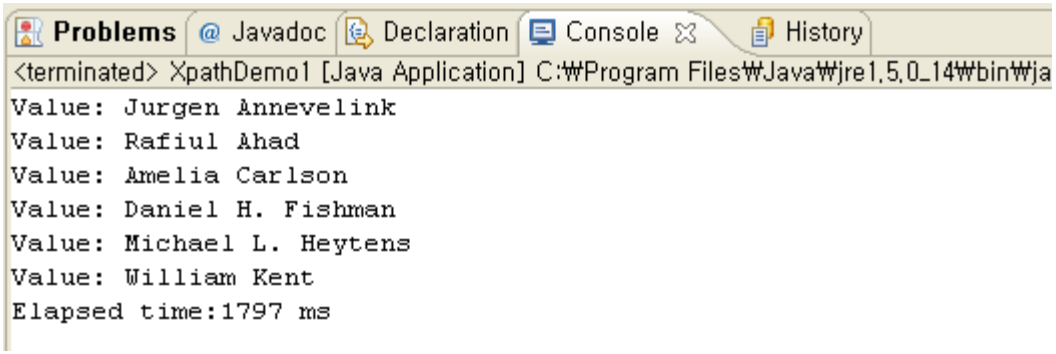
XPath Implementation in Java

- Using javax.xml.xpath.* (cont'd)



XPath Implementation in Java

- Using javax.xml.xpath.* (cont'd)



THANK YOU