Computer Science E-259

Last Time

- Computer Science E-259
- J2EE
- XML
  - What
  - Who
  - When
  - How
  - Why
- Computer Science E-259
This Time

- XML 1.1
- SAX 2.0.2
- JAXP 1.3 and Xerces 2.7.1 (2.9.1)
- Parsing
- My First XML Parser
XML 1.1

A Representative Document

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE students SYSTEM "student.dtd">

<!-- This is an XML document that describes students -->
<?studentdb displaydesc="true"?>

<students>
  <student id="0001">
    <name>Jim Bob</name>
    <status>graduate</status>
    <dorm/>
    <major>Computer Science & Music</major>
    <description>
      <![CDATA[
        <h1>Jim Bob!</h1>
        Hi my name is jim. I look like
        <img src="jim.jpg" />
      ]]>  
    </description>
  </student>
  <student id="0002">
    ...
  </student>
</students>
```
XML 1.1

XML Declaration

- Optional
- Must appear at the very top of an XML document
- Used to indicate the version of the specification to which the document conforms (and whether the document is “standalone”)
- Used to indicate the character encoding of the document
  - UTF-8
  - UTF-16
  - iso-8859-1
  - ...

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
XML 1.1

DOCTYPE

- References a Document Type Definition (DTD)
- Can refer to an external DTD file or include some DTD information within the tag itself
- DTD is the original mechanism for specifying the schema of an XML document
  - Inherited in part from SGML
  - Arcane syntax
  - Limited expressive functionality
- More in Lecture 8...

<!DOCTYPE students SYSTEM "students.dtd">
XML 1.1

Elements

- Main structure in an XML document
- Only one root element allowed
- Start Tag
  - Allows specification of zero or more attributes
    `<student id="0001" ...>`
- End Tag
  - Must match name, case, and nesting level of start tag
    `</student>`
- Name must start with letter or underscore and can contain only letters, numbers, hyphens, periods, and underscores

`<name>Jim Bob</name>`
XML 1.1

Element

- Element Content
  
  ```xml
  <student>
    <status>...</status>
  </student>
  ```

- Parsed Character Data (aka PCDATA, aka Text)
  
  ```xml
  <name>Jim Bob</name>
  ```

- Mixed Content
  
  ```xml
  <name>Jim <initial>J</initial> Bob</name>
  ```

- No Content
  
  ```xml
  <dorm/>
  ```
Attributes

- Name
  - Must start with letter or underscore and can contain only letters, numbers, hyphens, periods, and underscores

- Value
  - Can be of several types, but is almost always a string
  - Must be quoted
    - title="Lecture 2"
    - match='item="baseball bat"'
  - Cannot contain < or & (by itself)
PCDATA

- Text that appears as the content of an element
- Can reference entities
- Cannot contain < or & (by itself)
XML 1.1

Entities

- Used to “escape” content or include content that is hard to enter or repeated frequently
  - Somewhat like macros
- Five pre-defined entities
  - &amp; &lt; &gt; &apos; &quot;
- Character entities can refer to a single character by unicode number
  - e.g., &amp#xA9; is ©
- Must be declared to be legal
  - &lt;!ENTITY nbsp "&#160;"&gt;
- Cannot refer to themselves
XML 1.1

CDATA

- Parsed in “one chunk” by the XML parser
- Data within is not checked for subelements, entities, etc.
- Allows you to include badly formed markup or character data that would cause a problem during parsing
- Examples
  - Including HTML tags in an XML document
  - Used in XSLT to write out non-XML text
XML 1.1

Comments

- Can include any text inside a comment to make it easier for human readers to understand your document
- Generally not available to applications reading the document
- Always begin with <!-- and end with -->
- Cannot contain --
XML 1.1

Processing Instructions

- “Sticky notes” to applications processing an XML document that explain how to handle content
- The target portion (e.g., studentdb) of a PI indicates the application that is to process this instruction; cannot start with “xml”
- The remainder of the PI can be any text that gives instructions to the application

Examples

- Instructions to an application to display different versions of an image
- Instructions to an application to suppress display of certain content
- ...
SAX 2.0.2

A Sample Document

<students>
    <student id="0001"/>
</students>
Event-Based Parsing

Document

```xml
<students>
    <student id="0001"/>
</students>
```

ContentHandler

```java
startDocument();
startElement("students", {});
characters("\n   ");
startElement("student", {"id", "0001"});
endElement("student");
characters("\n   ");
endElement("students");
endDocument();
```
JAXP 1.3 and Xerces 2.7.1

SAXDemo

javax.xml.parsers.SAXParserFactory
javax.xml.parsers.SAXParser
org.xml.sax.*
org.xml.sax.helpers.*

...
**Parsing**

**Definition**

- In linguistics, to divide language into small components that can be analyzed. For example, parsing this sentence would involve dividing it into words and phrases and identifying the type of each component *(e.g., verb, adjective, or noun)*
- For XML, parsing means reading an XML document, identifying the various components, and making it available to an application
Parsing

Grammars in Backus-Naur Form

- In order to parse a document, you need to be able to specify exactly what it contains
- XML specification does this for XML using a grammar in Backus-Naur Form (BNF)
- A grammar describes a language through a series of rules
  - A rule describes how to produce a something (e.g., a start tag) by assembling characters and other non-terminal symbols
- Made up of
  - non-terminal symbols
  - terminal symbols (data that is taken literally)
Parsing

Arithmetic

- A grammar for arithmetic equations
  
  Eqn ::= Term '=' Term
  Term ::= '(' Term Op Term ')' | Value
  Op ::= '+' | '-' | '/' | '*'
  Value ::= <any number>

- Produces
  - \((4 + 3) = 7\)
  - \((1 + 2) = (3 - 0)\)
  - \(((10 / 2) + 1) = (3 * 2)\)
  - \(4 = 5\)
  - ...
A (much simplified) grammar for XML

\begin{verbatim}
element ::= STag content Etag
content ::= (element | CharData)*
STag    ::= '<' Name '>
ETag    ::= '<' '/' Name '>
\end{verbatim}

where \texttt{Name} is one or more characters excluding \texttt{>} and \texttt{CharData} is zero or more characters excluding \texttt{<}.  

My First XML Parser

Tokenizing and Recognizing

- **Tokenizing**
  - Creates tokens from the character stream
  - Element name, equal sign, start tag
- **Recognizing**
  - Understands the syntax of the document and checks for correctness
  - Builds a syntax tree
- **In `mf/XMLParser`, there will be no clear distinction between tokenizing and recognizing**
My First XML Parser

Recursive Descent Parsing

- XML's grammar works well with a parsing technique known as recursive descent parsing
- Basically:
  - You write a function that is responsible for parsing every non-terminal in the grammar
  - You assume that the document matches the grammar
  - The correct alternation in a rule can be determined by examining a few tell-tale starting characters (lookahead)
  - You recursively parse the document, calling each non-terminal parsing function as dictated by the grammar
  - Use exception handling to handle errors when they occur deep in the recursive call tree
My First XML Parser

Source Code

cscie259.project1 mf.*
Next Time

- The SAX API has a number of important advantages...
  - You can write very fast SAX parsers
    - No memory to allocate, data structures to link
    - “Fire and forget”
  - It is useful for large documents
    - Loading the whole document into memory is prohibitive
  - It is easy to use
- ...but it doesn't solve every problem
  - Need to have an internal data structure for some applications
  - To follow links in information (especially backwards ones)
  - To perform operations that require having multiple pieces of the document at the same time
- Enter the Document Object Model (DOM)...
Lecture 2: XML 1.1 and SAX 2.0.2

24 September 2007

David J. Malan
malan@post.harvard.edu