# XML 문서 처리

## (eXtensible Markup Language)

김덕태

(deogtae@yahoo.co.kr)

Bi Consulting. Co., Ltd.

2000년 4월

# 목차

- **XML(eXtensible Markup Language)** 소개
- **XML** 문법의 개요
- **XML** 이름 공간: 서로 다른 규격의 **XML** 문서들을 통합할 수 있는 표준
- **XML** 링킹 표준: **XML** 문서들을 서로 연결시킬 수 있는 확장된 링킹 표준
- 자바 기반 **XML** 구문 분석기 **(SAX, DOM)**
- **DOM** 객체 모델
- **XHTML: HTML 4.0** 문서 규격을 **XML** 문서 규격을 만족하도록 수정한 문서 규격
- **XSL (Extensible Stylesheet Language): XML** 문서를 다른 규격의 **XML** 문서로 변환하고 표시하기 위한 변환 정의 문서

# 소개

**XML 문서**

|

**XML 처리기 + XML 처리기 응용**

- 확장가능한 문법을 제공하는 문서 마크업 언어의 일종
- 마크업**(Markup):** 문서내용의 논리적 구조 및 표현 정보
- **XML** 처리기
  - XML 문서를 읽어들여 구문 분석한 후, 그 내용과 구조에 대한 접근을 제공하는 S/W 모듈
- **XML** 처리기 응용
  - XML 처리기를 내장(혹은 호출)하여 처리(편집, 검증, 저장, 전송, 검색, 구문 분석, 변환, 포매팅 등등) 기능을 제공하는 XML 처리기 응용 S/W

# XML의 특징 (1)

- 제품, 플랫폼 독립성
  - HTML과 유사한 텍스트 기반 문서 마크업 언어이므로, 텍스트 기반 S/W 도구를 비롯한 다양한 S/W 도구를 사용하여 처리할 수 있다.
- 단순성
  - SGML (Standard Generalized Markup Language)을 단순화한 문서 마크업 언어로서, SGML의 일종이다.
- 확장성
  - 문서 타입 정의 기능을 통하여 다양한 XML 기반 문서 형식(새로운 태그의 정의등)을 정의하여 사용할 수 있다.
  - HTML 문서 타입도 문법에 약간의 제약을 가하면 XML 문서 타입중 하나가 된다.
- 정확성
  - 주어진 응용에 가장 적합한 문서 타입 정의를 통하여 정보를 표현함으로써 문서 정보를 가장 적절히 표현할 수 있다.
  - 이러한 특징은 문서 처리의 유연성과 효율성을 제공한다.

# XML의 특징 (2)

- 통일성
  - 다양한 XML 문서 타입을 정의하여 사용하더라도 기본 골격은 XML 문서 형식을 따른다.
  - 다양한 XML 응용 S/W에서 XML 처리기를 재사용할 수 있다.
- 출력 매체 독립성
  - 문서의 내용이 특정 출력 형식과는 독립적이다.
  - HTML과는 달리 XML 문서가 최종적으로 출력되는 형태는 해당 XML 문서 타입을 지원하는 XML 처리기 응용에 달려있다.
  - 문서의 내용을 다양한 형태(테이블, 그래프 등등)로 출력할 수 있다.
  - 문서의 내용을 다양한 형식의 다른 문서(HTML, 다른 XML 문서, RTF, MS Word, Postscript, PDF, TeX, LaTeX)로의 변환이 가능하다.
  - XML 문서의 출력은 클라이언트측(웹 브라우저, 애플릿등) 및 서버측(서블릿등)에서 이루어질 수 있다. 이때, 경우에 따라 HTML 문서등 다른 문서 형식으로의 변환을 통하여 출력하는 방법이 종종 사용된다.

# XML 문서 타입의 종류 (1)

- **XML 문서 타입은 다음과 같은 부류로 나뉘어질 수 있다.**
  - 개별적인 용도를 위해 정의한 새로운 문서 형식
  - 특정 분야에서 받아들여지는 표준 형식
  - 일반적으로 폭넓게 받아들여지는 표준 형식
- **XML 문서 타입 정의 모음**
  - http://www.schema.net/
  - http://wdvl.com/Authoring/Languages/XML/Specifications.html
  - http://www.oasis-open.org/cover/xml.html
- 웹 분야 문서 타입
  - XHTML (HTML 문서 타입의 XML 타입 정의)
    - : http://www.w3.org/TR/WD-html-in-xml/
  - Resource Description Framework (RDF)
    - 다양한 웹 기반 메타데이타의 통합
    - http://www.w3.org/RDF/
  - Platform for Internet Content Selection (PICS)
    - 레이블 (메타데이타)와 인터넷 콘텐트와의 연동
    - http://www.w3.org/PICS/

# XML 문서 타입의 종류 (2)

- Platform for Privacy Preferences (P3P)
  - http://www.w3.org/P3P/
- Digital Signature (DSig)
  - http://www.w3.org/DSig/Overview.html
- Web Interface Definition Language (WIDL)
  - http://www.w3.org/TR/NOTE-widl

□ 정보 검색 분야 문서 타입

- A Query Language for XML (XML-QL)
  - SQL 유사 질의어
  - http://www.w3.org/TR/NOTE-xml-ql/

□ 사용자 인터페이스 분야 문서 타입

- Extensible Forms Description Language (XFDL)
  - 폼 기반 사용자 인터페이스 제공
  - http://www.w3.org/TR/NOTE-XFDL

# XML 문서 타입의 종류 (3)

- 그래픽스, 멀티미디아
  - Vector Markup Language (VML)
    - 벡터 그래픽
    - http://www.w3.org/TR/NOTE-VML
  - Scalable Vector Graphics (SVG)
    - 2차원 그래픽스 표현 (벡터 그래픽 도형, 이미지, 텍스트)
    - http://www.w3.org/Graphics/SVG/
  - Precision Graphics Markup Language (PGML)
    - 고정밀 2차원 그래픽스 표현 (폰트, 색상, 배치, 합성등)
    - PostScript, PDF (Portable Ducument Format)에 공통된 이미징 모델 채용
    - http://www.w3.org/TR/1998/NOTE-PGML
  - X3D
    - 3차원 그래픽 표현
    - VRML의 확장: http://www.web3d.org/fs_specifications.htm
  - Synchronized Multimedia Integration Language (SMIL)
    - 서로 독립적인 멀티미디어 자료의 동기화된 프리젠테이션
    - http://www.w3.org/AudioVideo/

# XML 문서 타입의 종류 (4)

- JSML (Java Speech Markup Language)
  - http://java.sun.com/products/java-media/speech/forDevelopers/JSML/

- 일반 문서, 프리젠테이션 타입
  - TeXML
    - TeX과 유사
    - http://www.alphaWorks.ibm.com/formula/texml
  - XslSliderMaker
    - 슬라이드 프리젠테이션
    - http://www.inria.fr/koala/XML/xslProcessor/slide.0.1.html

- 수학, 과학, 예술
  - Mathematical Markup Language (MathML)
    - 수학, 과학 기호 및 수식 표현
    - http://www.w3.org/Math/
  - Chemical Markup Language
    - 화학 분자식
    - http://ala.vsms.nottingham.ac.uk/vsms/java/jumbo/cml12/cml/
  - Music Markup Language (MusicML)
    - http://tcf.nl/3.0/musicml/index.html

# XML 문법의 기초 (1)

- **XML 1.0** 표준 문서
  - http://www.w3.org/TR/1998/REC-xml-19980210
- **XML** 문서는 다음과 같은 순서의 기본 형식을 갖는다.
  - XML 선언 (생략 가능)
  - 0개 이상의 문서 타입 선언
  - 1개의 루트 요소
- 유효한**(valid)** 문서
  - XML 기본 문법에 맞는(well-formed) 문서이어야 한다.
  - 문서내에 문서 타입 선언이 있어야 한다.
  - 문서가 선언된 문서 타입 정의를 따라야 한다.
  - 검증용 XML 처리기는 구문 분석 과정에서 문서의 유효성을 검증한다.
- **XML** 선언
  - 문법: <?xml version=1.0 encoding="인코딩 이름" standalone="yes|no" ?>
  - 인코딩 선언을 통하여 문서내에 사용된 문자의 인코딩을 지정할 수 있다.
  - 인코딩 이름은 IANA 표준 인코딩 이름을 사용하여야 한다.
    - 예) "UTF-8", "UTF-16", "ISO-8859-1", "EUC-KR", 등등

# XML 문법의 기초 (2)

- IANA 표준 인코딩 이름 문서
  - ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets
- 인코딩 선언이 생략된 경우,
  - 문서의 문자 인코딩은 "UTF-8" 혹은 "UTF-16" (유니코드)으로 가정되며,
  - 문서의 처음에 오는 바이트 순서 표식 (#xFEFF)에 의해 이들 인코딩을 구분한다.
- 모든 XML 처리기는 "UTF-8", "UTF-16" 인코딩을 지원해야 하나, 기타 인코딩의 지원은 선택 사항이다.
- standalone 선언이 생략되면 "standalone=no"를 선언한 것과 동일하며, yes인 경우에는 외부 DTD를 참조하지 않음을 의미한다.

□ 이름

- 유니코드 레터 (letter), 유니코드 숫자, `_', `:', `.', `-'등으로 이루어진 식별자이다.
- 요소 타입 이름 (태그 이름), 속성 이름, ID 이름, 개체 이름, 표기 이름등을 나타내는 데 사용된다.

□ 문자 ``""의 쌍이 구분자로서 사용되는 모든 경우에 대하여, 문자 ``""의 쌍으로 대신할 수 있다.

# XML 문법의 기초 (3)

## 요소 (element)

- 요소는 시작 태그와 끝 태그로 묶여진 논리적 문서 단위이다.
- 태그 이름과 요소 타입 이름은 동의어이다.
- 요소의 문법: 시작 태그 요소 내용 끝 태그
- 시작 태그 문법: <태그 이름 속성 이름="값" 속성 이름="값" ...>
  - 예) <termdef id="dt-dog" term="dog">
- 끝 태그 문법: </태그 이름>
  - 예) </termdef>
- 빈 태그 문법: <태그 이름 속성 이름="값" 속성 이름="값" .../>
  - : 내용이 없는 요소를 나타내며, <태그 이름 ...></태그 이름>과 동일한 의미를 갖는다.
  - 예) <termdef id="dt-dog" term="dog"/>
- HTML과는 달리 대소문자를 구분한다.
- 속성 값은 `<'를 제외한 문자로 이루어진다.
- 요소 내용은 자식 요소, 문자 데이타, CDATA 섹션들로 이루어진다.
- 유효한 문서는 선언된 문서 타입 정의내의 태그만을 사용해야 하며 그 문법을 따라야한다.

# XML 문법의 기초 (4)

- 문자 데이타
  - `<', `&' 문자는 마크업에 사용되므로 직접 표현될 수 없다.
  - 이들 문자는 문자 그 자체를 나타내기 위해서는 문자 참조나 개체 참조 ``<'', ``&''를 사용해야 한다.
- 문자 참조
  - 문법: &#10진수 문자 코드 값;
  - 문법: &#x16진수 문자 코드 값;
  - 문자 데이타로 직접 표현하기 곤란한 어떠한 문자도 ISO 10646 (유니코드의 32 비트 확장판) 문자 코드 값으로 표현할 수 있다.
  - 문자 참조는 요소 내용, 속성 값, 문서 타입 정의등 문서의 대부분의 곳에서 사용될 수 있다.
- **CDATA 섹션**
  - 문법: <![CDATA[ 문자열 ]]>
    - 예) <![CDATA[<greeting>Hello, world!</greeting>]]>
  - 마크업 표식을 포함하는 문자열을 해석하지 않고 일반 문자 데이타로서 표현하기 위해 사용한다.
  - 요소 내용에서만 나타날 수 있다.

# XML 문법의 기초 (5)

- 주석
  - 문법: **<!--"** 문자열 ``**-->**
  - `문자열'에는 ``**--"**가 올 수 없다.
  - 마크업 표식이외의 어느 곳에서도 나타날 수 있다.
- 예) **intro.xml**

```
<?xml version="1.0" encoding="euc-kr"?>
<!DOCTYPE seminar SYSTEM "seminar.dtd">
<!--
    XML 예제 화일 주석 예
-->

<seminar>
<title>XML 문서 처리</title>
<author>김덕태</author>
```

# XML 문법의 기초 (6)

```
<section>
<head>XML 문법의 기초</head>
<itemize>
   <item> XML 문서내에 다음과 같은 XML 특수 기호를 사용하기 위해서는
        개체 참조를 이용하거나,
<example>
    System.out.println(1 &lt; 2);
    System.out.println(true &amp;&amp; false)
</example>
   </item>
   <item> CDATA 섹션을 이용하거나,
<example><![CDATA[
    System.out.println(1 < 2);
    System.out.println(true && false)
]]>
</example>
    </item>
    <item> 처리기 응용의 처리에 맡긴다.
<examplefile file="xml\Test.java"/>
    </item>
   <item> `가' 대신 문자 참조 `&#xac00;'라고 표현할 수 있다.
   </item>
</itemize>
</section>
</seminar>
```

# 문서 타입 선언 (1)

- 문서 타입 선언
  - 목적: XML 문서에 문서 타입 정의 (DTD)를 포함시키기 위한 방법 제공
  - 문법: <!DOCTYPE 문서 타입 이름 문서 타입 정의 식별자 [ 문서 타입 정의 ]>
    - 유효한 문서는 문서 타입 이름이 XML 문서의 루트 요소 타입 이름(태그 이름)과 같아야 한다.
    - 문서 타입 정의 식별자와 문서 타입 정의부 모두 생략될 수 있다.
    - `문서 타입 정의 식별자'를 지정함으로써 분리되어 저장되어 있는 외부 문서 타입 정의를 포함시킬 수 있다.
  - 내부 문서 타입 정의 예

```
<?xml version="1.0" encoding="euc-kr"?>
<!DOCTYPE seminar [
<!ELEMENT seminar (title, author?, section*)>
...
<!ATTLIST examplefile
          file CDATA #REQUIRED>
...
]>

<seminar>
...
</seminar>
```

# 문서 타입 선언 (2)

- 문서 타입 정의 식별자
  - 문서 타입 선언에서 URI에 의해 식별되는 문서 타입 정의를 XML 문서 내에 논리적으로 포함시키는 기능을 제공한다.
  - 문법: SYSTEM "시스템 URI"
    - `시스템 URI'는 특정 조직 내부에서만 사용되는 문서 타입 정의가 저장되어 있는 URI이다.
  - 문법: PUBLIC "공개 URI" "시스템 URI"
    - `공개 URI'는 폭넓게 받아들여지는 문서 타입 정의가 저장되어 있는 URI이며, 이 겨우 XML 처리기는 표준 문서 타입 정의 라이브러리를 검색하여 해당 문서 타입 정의를 검색할 수 있다.
  - XML 처리기는 URI내의 비 아스키 문자는 ``%HH" 형태의 문자열로 변환하여 처리하여야 한다. (여기서 HH는 해당 문자의 UTF-8 인코딩 코드 값을 16진수로 나타낸 값이다.)
  - 이 URI에 저장된 문서 타입 정의의 첫 행에 다음 선언이 올 수 있다. (각 선언부가 생략가능하다.)
    - <?xml version="1.0" encoding="인코딩 이름" ?>

# 문서 타입 정의 (1)

- 문서 타입 정의
  - `문서 타입 정의'는 0개 이상의 마크업 선언 (요소 선언, 속성 목록 선언, 개체 선언, 매개변수 개체 선언, 표기 선언) 및 매개변수 개체 참조로 이루어진다.
  - 매개변수 개체 참조는 마크업 선언내에도 올 수 있다.
  - 문서 타입 정의는 내포될 수 있으며, 누적되고, 내부 문서 타입 정의는 외부 문서 타입 정의보다 우선 순위가 높다.
- 요소 선언
  - 문법: <!ELEMENT 태그 이름 요소 내용 패턴>
  - 다음 각 요소 내용 패턴은 이 요소가 가질 수 있는 요소 내용의 형식을 정의한다.
    - ANY: 어떠한 형식도 올 수 있다.
      - 예) <!ELEMENT container ANY>
    - EMPTY: 요소 내용을 가질 수 없다.
      - 예) <!ELEMENT br EMPTY>
    - (#PCDATA#): 요소 내용으로 문자 데이타, CDATA 섹션만이 올 수 있다.
      - 예) <!ELEMENT p (#PCDATA) >

# 문서 타입 정의 (2)

- (#PCDATA# | 태그 이름 | 태그 이름 | ...)*: 요소 내용으로 문자 데이타, 문자 참조, 개체 참조, CDATA 섹션 및 주어진 태그의 요소들을 가질 수 있다.
  - 예) <!ELEMENT p (#PCDATA|emph)* >
- 정규 수식: 주어진 패턴의 자식 요소들만을 가질 수 있다. 정규 수식은 태그 이름들과 연산자 `|' (선택), `,' (연결), 괄호 (그룹화), `?' (0번 혹은 1번), `*' (0번 이상 반복), `+' (1번 이상 반복)를 사용하여 나타낸다.
  - 예 1) <!ELEMENT spec (front, body, back?)>
  - 예 2) <!ELEMENT div1 (head, (p | list | note)*, div2*)>

## 속성 (attribute) 목록 선언

- 요소 (태그)에 사용될 수 있는 속성 이름, 속성 자료형, 디폴트 값들을 정의한다.
- 문법: <!ATTLIST 태그 이름 속성 이름 속성 자료형 디폴트 값 ...>
- `속성 자료형'은 다음과 같은 값으로 지정한다.
  - CDATA: 일반 스트링.
  - NMTOKEN: 임의의 이름
  - NMTOKENS: 공백류 문자로 구분된 임의의 이름 목록
  - (이름 | 이름 | ...): 주어진 이름중 1개

# 문서 타입 정의 (3)

- ID: ID 이름. 하나의 XML 문서내에서는 이 태그의 모든 요소에서 속성 값으로 지정된 이 이름이 모두 서로 달라야 한다. 문서내에서 요소를 링크시키기 위한 주소로 사용하기 위한 것이 주 목적이다. 속성 이름으로 "id"를 많이 사용한다.
- IDREF: ID 이름. 하나의 XML 문서내에서 속성 자료형 ID의 해당 속성 값을 갖는 요소로 링크시키기 위한 것이 주 목적이다.
- IDREFS: 공백류 문자로 구분된 ID 이름 목록. .

◆ `디폴트 값'은 해당 요소의 디폴트 속성 값에 대한 사항을 정의한다.

◆ "속성 값": 요소에서 이 속성이 지정되지 않은 경우 사용될 디폴트 속성 값

- #FIXED "속성 값": 이 속성 디폴트 값만이 이 속성이 가질 수 있는 유일한 값이다.
- #IMPLIED: 디폴트 값을 정의하지 않고 처리기 응용이 디폴트 값들 제공.
- #REQUIRED: 요소에서 이 속성이 반드시 지정되어야 하며, 디폴트 값은 없다.

◆ 예약된 속성 xml:space를 다음과 같이 속성 목록에 선언해주면 XML 처리기가 요소내의 공백류 문자를 정규화하지 않고 있는 그대로 처리기 응용에 넘겨준다.

- <!ATTLIST code xml:space (default | preserver) ‘preserve’>

# 문서 타입 정의 (4)

- 예

```
<!ATTLIST termdef
          id      ID      #REQUIRED
          name    CDATA   #IMPLIED>
<!ATTLIST list
          type    (bullets|ordered|glossary)  "ordered">
<!ATTLIST form
          method  CDATA   #FIXED "POST">
```

- 예제: **seminar.dtd**

```
<?xml encoding="euc-kr"?>

<!ELEMENT seminar (title, author?, section*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT section (head, (itemize | example | examplefile)*)>
<!ELEMENT head (#PCDATA)>
<!ELEMENT itemize (item)+>
<!ELEMENT item (#PCDATA | example | examplefile| itemize)*>
<!ELEMENT example (#PCDATA)>
<!ELEMENT examplefile EMPTY>
<!ATTLIST examplefile
          file CDATA #REQUIRED>
```

# 개체 (entity)와 개체 참조

## 개체

- 개체는 문서 타입 정의내의 개체 선언에 의해 이름과 내용을 부여받는다.
- 개체의 내용은 요소, 문자 데이타, CDATA 섹션들로 이루어질 수 있다. 즉, XML 문서의 일부로서 정의된다.
- 내부 개체는 문서 타입 정의내에 직접 나타나고, 외부 개체는 별도의 화일(URI)에 저장된다.
- 개체의 내용은 XML 문서내의 개체 참조에 의해 참조(포함)된다.
- 한번 정의된 개체를 여러곳에서 참조함으로써 반복적인 XML 문서 작업을 줄이고, 손쉽게 관리할 수 있다.
- 일종의 매크로 기능이다.

## 개체 참조

- 문법: &개체 이름;
- 개체 참조는 요소 내용 및 속성 값에서만 나타날 수 있으며, XML 처리기에 의해 해당 개체의 내용으로 대체된 후, 구문 분석된다.
- ``<" (``<"), ``&"(``&"), ``gt;"(``>"), ``quot;'(``""), ``apos;"(``'")은 해당 문자 참조로의 정의가 내장되어 있다.

# 개체 선언

- 개체 참조가 개체 정의에 나타나는 경우, 그 참조가 해석되지 않고, 있는 그대로 다루어진다.
- 다음 페이지에는 다양한 문자 개체가 정의되어 있으므로, 문서 타입 정의내에 포함시켜 사용할 수 있다.
  - http://www.schema.net/entities/
- 내부 개체 선언
  - 문법: <!ENTITY 개체 이름 "내부 개체 정의">
  - `내부 개체 정의'내에서 문자 `%'는 매개변수 개체 참조로서 해석된다.
- 외부 개체 선언
  - 문법: <!ENTITY 개체 이름 외부 개체 식별자>
  - 개체 내용은 `외부 개체 식별자'가 나타내는 URI의 내용이다.
  - 외부 개체 식별자는 문서 타입 정의 식별자와 동일한 문법과 의미를 가지며, 단지 해당 URI에 저장된 내용이 개체 정의라는 차이가 있을 뿐이다.
- 속성 목록 선언의 `속성 자료형'으로 다음을 사용할 수 있다.
  - ENTITY: 개체 이름. 이진 데이타등 #PCDATA로 전달하기 곤란한 경우 유용.
  - ENTITIES: 공백류 문자로 구분된 개체 이름 목록

## 예: entity_test.xml

```
<?xml version="1.0" encoding="euc-kr"?>
<!DOCTYPE seminar SYSTEM "seminar.dtd" [
<!ENTITY author "김덕태">
<!ENTITY head1 "<head>개체의 의미</head>">
<!ENTITY section2 SYSTEM "section2.xml">
]>

<seminar>
<title>XML 개체 처리</title>

<author>&author;</author>

<section>
&head1;
<itemize>
   <item> 개체의 내용은 요소, 문자 데이타, CDATA 섹션들로 이루어진
   내용으로 정의될 수 있다.
   </item>
</itemize>
</section>

&section2;
</seminar>
```

## 예: section2.xml

```
<?xml encoding="euc-kr"?>

<section>
<head>외부 개체 정의</head>
<itemize>
   <item> 이와 같이 외부에 정의된 개체를 개체 참조를 통하여 포함시킬
   수 있다.
   </item>
</itemize>
</section>
```

# 매개변수 개체 (entity)와 매개변수 개체 참조

## 매개변수 개체

- 매개변수 개체는 문서 타입 정의내의 매개변수 개체 선언에 의해 이름과 내용을 부여받는다.
- 매개변수 개체의 내용은 문서 타입 정의이 일부로서 정의된다.
- 내부 매개변수 개체 정의는 문서 타입 정의내에 직접 나타나고, 외부 매개변수 개체 정의는 별도의 화일(URI)에 저장된다.
- 개체의 내용은 문서 타입 정의내의 매개변수 개체 참조에 의해 참조(포함)된다.
- 한번 정의된 매개변수 개체를 여러곳에서 참조함으로써 반복적인 문서 타입 정의 작업을 줄이고, 손쉽게 관리할 수 있다.
- 일종의 매크로 기능이다.

## 매개변수 개체 참조

- 문법: %매개변수 개체 이름;
- 요소 내용과 속성 값내에서는 해석되지 않고, 문서 타입 정의내에서만 해석된다.
- 문서 타입 정의내의 거의 모든 곳에서 사용될 수 있다.

# 매개변수 개체 선언 (1)

- 매개변수 개체 선언 및 문서 타입 선언의 외부 개체 식별자가 나타내는 **URI**에 저장되어 있는 개체 내용은 문서 타입 정의의 일부로서 마크업 선언과 조건부 섹션들로 이루어진다.
- 내부 매개변수 개체 선언 문법
  - : <!ENTITY % 매개변수 개체 이름 "내부 매개변수 개체 정의">
- 외부 매개변수 개체 선언 문법
  - : <!ENTITY % 매개변수 개체 이름 외부 매개변수 개체 식별자>
- 정의된 내용이 문서 타입 정의의 일부라는 차이점만 있을 뿐, 개체 식별자와 문법 및 의미가 동일하다.
- 포함 조건부 섹션 문법: **<![INCLUDE[** 매개변수 개체 정의 **]]>**
- 배제 조건부 섹션 문법: **<![IGNORE[** 매개변수 개체 정의 **]]>**

# 매개변수 개체 선언 (2)

- 예)

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

# 표기 (notation)

- 표기 이름은 처리기 명령의 대상 응용 혹은 구문 분석되지 않는 개체의 형식을 식별하기 위한 식별자이다.
- 응용 식별자는 **XML** 처리기가 호출할 응용을 식별하는 **URI**이다.
- 선언 문법: **<!NOTATION** 표기 이름 응용 식별자**>**
- `응용 식별자'의 문법
  - SYSTEM "시스템 URI"
  - PUBLIC "공개 URI"
  - PUBLIC "공개 URI" "시스템 URI"
- 속성 목록 선언의 `속성 자료형'으로 다음을 지정함으로써 주어진 표기 이름중 **1**개를 속성 값으로 사용할 수 있도록 선언할 수 있다.
  - NOTATION (표기 이름 | 표기 이름 | ...)

# 처리기 명령 (Processing Instruction)

- 문법: **<?**표기 이름  명령**?>**
- **XML** 처리기가 주어진 문자열의 `명령'을 `표기 이름'의 선언에 의해 정의된 외부 개체 식별자에 의해 식별된 식별된 응용에 전달한다.
- 마크업 표식이외의 어느 곳에서도 나타날 수 있다.
- `표기 이름'은 표기 선언에 의해 선언된 식별자 **xml, XML** 등으로 시작할 수 없다.

# 구문 분석되지 않는 **(unparsed)** 개체

- 이미지 화일등 **XML** 문서 형식이 아닌 데이타를 나타내기 위해서 사용된다.
- 선언 문법: **<!ENTITY** 개체 이름 외부 개체 식별자 **NDATA** 표기 이름**>**
- 외부 개체 식별자가 나타내는 **URI**는 해당 데이타의 위치를 나타내는 **URI**이다.
- 구문 분석되지 않는 개체는 속성 목록 선언에서 `속성 자료형'이 다음과 같은 값으로 선언된 경우의 속성 값으로서만 사용될 수 있다.
  - ENTITY: 구문 분석되지 않는 개체의 이름.
  - ENTITIES: 구문 분석되지 않는 개체의 이름 목록.
- **XML** 처리기는 구문 분석되지 않는 개체 참조가 나타나면, 해당 `표기 이름'이 나타내는 **URI**가 식별하는 응용에 `외부 개체 식별자'가 나타내는 **URI**를 전달한다.

# XML 이름 공간 (1)

- 목적: **XML** 요소, 속성 이름을 **URI**로 식별되는 이름 공간과 연관시켜 구분
- 용도
  - 전세계적으로 고유한 요소, 속성 이름 생성: 이름 충돌 해결 및 의미 구분
  - 스키마(DTD 포함)의 재사용
  - 서로 다른 규격의 XML 문서들을 1개의 문서내에 통합.
- **XML Namespaces 1.0 Recommendation**
  - http://www.w3.org/TR/1999/REC-xml-names-19990114/
- **XML** 이름 공간은 **URI**로 식별한다.
- 이름 공간의 참조
  - 이름에 `:'이 사용된 경우, 앞부분이 이름공간을 나타내는 접두어로 해석된다.
- 이름 공간의 선언
  - 요소의 속성으로 다음과 같이 이름공간을 선언한다.
  - 하나의 요소에 여러 개의 이름 공간 선언을 할 수 있다.
  - 자손 요소에서 선언된 동일 이름 공간 선언은 조상 요소에 선언된 이름 공간 선언을 재정의(override)할 수 있다.

# XML 이름 공간 (2)

- xmlns:이름공간접두어="이름공간의 URI"
  - 이 요소 및 모든 자손 요소의 요소, 속성 이름에 이름 공간 접두어를 붙여 해당 이름 공간을 지정할 수 있다.
- xmlns="이름공간의 URI"
  - 이 요소 및 모든 자손 요소의 요소, 속성 이름에 대하여, 이름 공간 접두어가 붙지 않은 요소, 속성 이름은 디폴트로 이 이름공간에 속하게 된다. (디폴트 이름 공간)
- xmlns=""
  - 조상 요소에 선언된 디폴트 이름 공간 선언을 이 요소 및 모든 자손 요소에 대하여 취소시킨다.

## 문제점

- 이름 공간을 사용하는 경우 유효한 문서가 되기 어렵다.
- 유효한 문서가 되려면 다양한 이름 공간의 이름들이 하나의 DTD내에 정의되어있어야 한다.

# XML 이름 공간 (3)

- 예제: **namespace.xml**

```
<?xml version="1.0" encoding="euc-kr"?>
<seminar>
  <section xmlns="http://www.example.com/book"
           xmlns:html="http://www.w3.org/TR/REC-html40">
    <title>XML 소개</title>
    <html:ul>
      <html:li>XML의 유래</html:li>
      <html:li>XML의 용도</html:li>
      <li xmlns="http://www.w3.org/TR/REC-html40">XML 처리기</li>
    </html:ul>
  </section>
</seminar>
```

# XML 링킹(linking) 표준 (1)

- 링크 **(link): 2** 개 이상의 데이타 객체 **(**혹은 그 일부분**)**사이의 명시적 관계
- **XML Linking Language (XLink)**
  - 명세 문서: http://www.w3.org/TR/xlink/
  - 목적: HTML 링크인 단 방향 링크 이외에 더 강력한 링크 기능 (여러 방향 링크, 확장 링크, 아웃오브라인 링크) 제공
  - 이름 공간 식별자: http://www.w3.org/1999/xlink
- **XML Pointer Language (XPointer)**
  - 명세 문서: http://www.w3.org/TR/xptr
  - 목적: 링크의 "id" 속성이 정의되어 있지 않아도 수식으로 표현되는 문서 조각 식별자로 임의의 문서 조각을 나타낼 수 있다.
  - 예) `<xlink:simple href='doc.xml#xpointer(book/chapter[position() &lt;= 5])' />`

# XML 링킹(linking) 표준 (2)

- **위치 식별자 (Locator)**
  - 목적: URI의 확장
  - 다음중 1가지 형태를 갖는다.
    - URI: 자원의 주소. 보통, 1개의 문서 전체를 나타낸다.
    - #이름
    - #XPointer
    - URI#이름
    - URI#XPointer
  - 이들 형태중 `#' 대신에 `|'가 사용될 수 있다.
  - `이름'은 XPointer의 `id(이름)'과 동일하다.
- **XPointer**
  - 목적: XML 문서내의 요소 혹은 요소의 일부에 대한 주소
  - 일반적인 자원의 식별자는 URI를 사용한다.
  - XML 문서내의 일부를 나타내는 식별자는 XPointer를 사용해야 한다.

# 자바 XML 구문 분석기 (1)

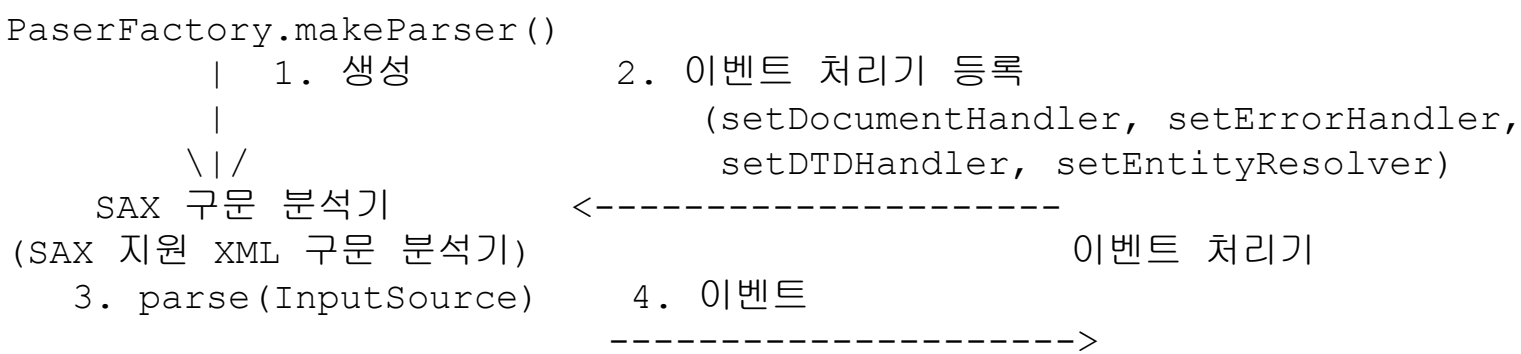
- **XML 구문분석기의 종류**
  - SAX 구문분석기
    - 이벤트 기반 XML 구문 분석기
    - 구문 분석중에 구문 분석기와 상호 작용하며 동작하는 프로그램을 작성할 수 있게 해준다.
  - DOM 구문분석기
    - XML 문서를 구문 분석하여 DOM 객체 모델을 생성하는 구문 분석기
- **JAXP (Java API for XML Parsing) 1.0**
  - XML 1.0, SAX 1.0, DOM (Core) Level 1, JAXP 1.0 지원
  - http://java.sun.com/xml/
  - javax.xml.parsers, org.xml.sax, org.w3c.dom 패키지 포함
- **IBM XML Parser for Java 3.0.1**
  - XML 1.0, SAX 1.0, SAX 2.0 베타, DOM Level 1, DOM Level 2 베타 지원
  - http://www.alphaworks.ibm.com/formula/xml/
- **Docuverse DOM SDK 1.0A5**
  - XML 1.0, SAX 1.0, DOM (Core) Level 1, DOM (HTML) Level 1, JAXP 1.0 지원
  - http://www.docuverse.com/domsdk/

# 자바 **XML** 구문 분석기 **(2)**

- **Microsoft XML Parser in Java**
  - http://www.microsoft.com/XML
  - 문서: http://www.microsoft.com/java/sdk/32/default.htm

# SAX API 1.0 (Simple API for XML)

- 목적: **SAX** 드라이버**(SAX** 지원 **XML** 구문 분석기**)**에 대한 인터페이스 정의
- **org.xml.sax** 패키지
- 홈 페이지: **http://www.megginson.com/SAX/**
- 전체 구조



# SAX API: ParserFactory 클래스

- 목적: **SAX** 지원 구문 분석기의 생성
- 메쏘드
  - **`ParserFactory.makeParser`**`() throws ClassNotFoundException, IllegalAccessException, InstantiationException, NullPointerException, ClassCastException`
    - : "org.xml.sax.parser" 시스템 프로퍼티로 지정된 클래스 이름의 SAX 지원 구문 분석기 생성
  - **`ParserFactory.makeParser`**`(String parser) throws ClassNotFoundException, IllegalAccessException, InstantiationException, ClassCastException`
    - : 주어진 클래스 이름의 SAX 지원 구문 분석기 생성. 시스템 프로퍼티를 설정하기 곤란한 애플릿에서 유용하다.

# SAX API: Parser 인터페이스

- 목적: 구문 분석의 시동, 이벤트 처리기 (콜백) 등록
- 구문 분석 도중에도 다른 이벤트 처리기로 재등록할 수 있다.
- 메쏘드
  - **parse**(InputSource source) throws SAXException, IOException
    - SAX 구문 분석기가 XML 문서 source의 구문 분석을 시작하게 한다. 구문 분석이 진행중에는 호출할 수 없다.
  - **parse**(String systemId) throws SAXException, IOException
    - = parse(new InputSource(systemId))
  - **setDocumentHandler**(DocumentHandler handler)
    - 문서 이벤트 처리기를 등록한다. 등록되어 있지 않으면 모든 문서 이벤트가 무시된다.
  - **setErrorHandler**(ErrorHandler handler)
    - 오류 이벤트 처리기를 등록한다. 등록되어 있지 않으면 치명적 오류를 제외한 모든 오류 이벤트가 무시된다.
  - **setDTDHandler**(DTDHandler handler)
    - DTD 이벤트 처리기를 등록한다. 등록되어 있지 않으면 모든 DTD 이벤트가 무시된다.

# SAX API: Parser 인터페이스

- **`setEntityResolver`**`(EntityResolver resolver)`
  - EntityResolver를 등록한다. 등록되어 있지 않으면 SAX 구문 분석기에 의해 처리된다.
- **`setLocale`**`(Locale locale) throws SAXException`
  - 경고, 오류 메시지의 로캘. 해당 로캘을 지원하지 않는 경우에는 예외 발생.

# SAX API: InputSource 클래스

- 목적: 문자 스트림, 바이트 스트림, 시스템 **URI**등으로 검색할 수 있는 **XML** 문서 내용
- 생성자
  - `new InputSource()`
  - `new InputSource(Reader characterStream)`
  - `new InputSource(InputStream byteStream)`
  - `new InputSource(String systemId)`
- 메소드
  - `Reader` **`characterStream`** ⇔
    - 입력 원천을 문자 스트림으로 설정.
  - `InputStream` **`byteStream`** ⇔
    - 입력 원천을 바이트 스트림으로 설정. 문자 스트림으로 설정되지 않은 경우, 문서 내용을 가져오기 위해 사용된다.
  - `String` **`systemId`** ⇔
    - 입력 원천을 시스템 URI로 설정. 문자 스트림 혹은 바이트 스트림으로 설정되지 않은 경우, 문서 내용을 가져오기 위해 사용된다. 설정된 URI는 문서내의 상대 URI를 처리하는 경우에도 사용된다.

# SAX API: InputSource 클래스

- `String` **`publicId`** ⇔
  - 입력 원천의 위치 정보를 제공하기위한 공개 URI.
- `String` **`encoding`** ⇔
  - 입력 원천이 문자 스트림이 아닌 경우, XML 문서의 문자 인코딩

# SAX API: DocumentHandler 인터페이스

- 목적: **SAX** 구문 분석기가 구문 분석중에 발생하는 문서 이벤트의 처리기에 대한 인터페이스
- 메소드
  - **`setDocumentLocator`**`(Locator locator)`
    - : 각 문서 이벤트 처리 메쏘드내에서 문서의 현재 위치를 알아내기 위한 locator를 알려준다. 이 인터페이스의 모든 다른 메쏘드가 호출되기 전에 호출된다.
  - **`startDocument`**`() throws SAXException`
    - : XML 문서 시작시에 1번 호출된다.
  - **`endDocument`**`() throws SAXException`
    - : XML 문서 마지막에 1번 호출된다.
  - **`startElement`**`(String name, AttributeList atts) throws SAXException`
    - : XML 문서의 구성 요소 시작시마다 호출되어, 요소 타입 이름과 애트리뷰트를 알린다. 지정된 애트리뷰트 및 디폴트 애트리뷰트를 알려주나 함축된 애트리뷰트 (#IMPLIED)는 알려주지 않는다.
  - **`endElement`**`(String name) throws SAXException`
    - : XML 문서 요소 마지막마다 호출된다.

# SAX API: DocumentHandler 인터페이스

- **characters**(char ch[], int start, int length) throws SAXException
  - : 일련의 문자 데이타를 알린다. 여러번으로 나뉘어 호출될 수 있다.
- **ignorableWhitespace**(char ch[], int start, int length) throws SAXException
  - : 무시될 수 있는 일련의 공백류 문자 (whitespace)를 알린다. 여러번으로 나뉘어 호출될 수 있다. 검증 구문 분석기는 반드시 호출하나, 비검증 구문 분석기는 호출하지 않을 수도 있다.
- **processingInstruction**(String target, String data) throws SAXException
  - : XML 선언 혹은 XML 텍스트 선언을 알린다.

# SAX API: Locator 인터페이스

- 목적: **XML** 문서의 현재 위치를 알아내기 위한 인터페이스
- 이들 메쏘드는 **DocumentHandler** 인터페이스의 메쏘드내에서만 호출되어야 한다.
- 메소드
  - `String` **`getPublicId`**`()`
    - : 현재 문서 이벤트의 공개 URI.
  - `String` **`getSystemId`**`()`
    - : 현재 문서 이벤트의 시스템 URI.
  - `int` **`getLineNumber`**`()`
    - : 현재 문서 이벤트가 종료하는 행 번호. 알려지지 않은 경우에는 -1
  - `int` **`getColumnNumber`**`()`
    - : 현재 문서 이벤트가 종료하는 열 번호. 알려지지 않은 경우에는 -1

# SAX API: org.xml.sax.helpers.LocatorImpl 클래스

- **implements Locator**
- 목적: **Locator** 객체의 저장
- 생성자
  - **`new LocatorImpl`**`()`
  - **`new LocatorImpl`**`(Locator locator)`
    - : locator와 동일한 내용을 갖는 객체 생성.
- 메소드
  - **`setPublicId`**`(String publicId)`
  - **`setSystemId`**`(String systemId)`
  - **`setLineNumber`**`(int lineNumber)`
  - **`setColumnNumber`**`(int columnNumber)`

# SAX API: AttributeList 인터페이스

- 목적: 문서 요소의 애트리뷰트들
- 메소드
  - `int` **`getLength`**`()`
    - 애트리뷰트 갯수
  - `String` **`getName`**`(int i)`
    - i번째 애트리뷰트의 이름
  - `String` **`getType`**`(int i)`
    - i번째 애트리뷰트의 자료형. "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES", "NOTATION"중 하나.
  - `String` **`getValue`**`(int i)`
    - : i번째 애트리뷰트의 값.
  - `String` **`getType`**`(String name)`
  - `String` **`getValue`**`(String name)`

# SAX API: org.xml.sax.helpers.AttributeListImpl 클래스

- **implements AttributeList**
- 목적: **AttributeList**의 저장
- 메소드
  - **`new AttributeListImpl`**`()`
  - **`new AttributeListImpl`**`(AttributeList atts)`
    - : atts와 동일한 내용을 갖는 객체 생성.
  - **`setAttributeList`**`(AttributeList atts)`
  - **`addAttribute`**`(String name, String type, String value)`
  - **`removeAttribute`**`(String name)`
  - **`clear`**`()`

# SAX API: ErrorHandler 인터페이스

- 목적: **SAX** 구문 분석기가 구문 분석중에 발생하는 오류 이벤트 처리기에 대한 인터페이스
- 메소드
  - **warning**(SAXParseException exception) throws SAXException
  - **error**(SAXParseException exception) throws SAXException
    - 구문 분석이 계속하여 진행될 수 있는 회복 가능한 오류의 알림.
  - **fatalError**(SAXParseException exception) throws SAXException
    - 구문 분석을 정상적으로 완료할 수 없는 치명적 오류의 알림.

# SAX API: DTDHandler 인터페이스

- 목적: **SAX** 구문 분석기가 구문 분석중에 발생하는 **DTD** 이벤트 처리기에 대한 인터페이스
- 표기 선언 및 구문분석되지 않는 개체 선언을 선언 순서와 관계없이 호출하여 알려준다.
- 모든 이벤트가 **DocumentHandler**의 **startDocument, startElement** 사이에 발생한다.
- 메소드
  - **`notationDecl`**`(String name, String publicId, String systemId) throws SAXException`
    - 표기 선언의 알림.
  - **`unparsedEntityDecl`**`(String name, String publicId, String systemId, String notationName) throws SAXException`
    - 구문분석되지 않는 개체 선언의 알림

# SAX API: EntityResolver 인터페이스

- 목적: **SAX** 구문 분석기가 구문 분석중에 외부 개체 처리의 커스터마이즈화
- 외부 개체의 **InputSource**를 다른 **InputSource**로 대체한다.
- 시스템 **URI**가 처리되기 곤란한 **URI (**데이타베이스등**)**인 경우 데이타를 가져와서 **InputSource**로 제공하거나**, URI** 리다이렉트**(redirect)**등에 유용하다.
- 메소드
  - `InputSource` **`resolveEntity`**`(String publicId, String systemId) throws SAXException, IOException`
    - 모든 종류의 외부 개체 (외부 문서 타입 정의, 외부 개체, 매개변수 외부 개체)를 포함시킬때마다 호출된다.
    - null을 반환하면 구문 분석기는 본래의 InputSource를 사용한다.

# SAX API: HandlerBase 클래스

- 목적: **SAX** 이벤트 처리기의 편리한 작성
- 이 클래스는 다음 각 인터페이스를 다음과 같이 동작하도록 구현한다.
  - `DocumentHandler` 인터페이스
  - `ErrorHandler` 인터페이스
    - : 경고, 회복 가능한 오류는 무시하고, 치명적 오류의 경우에는 해당 SAXParseException 예외 발생.
  - `DTDHandler` 인터페이스
    - : DTD 이벤트를 무시한다.
  - `EntityResolver` 인터페이스
    - : InputSource로 null을 반환한다.

# JAXP API (1)

- **javax.xml.parsers 패키지**
  - SAX, DOM 구문 분석기 API 사용을 위한 표준화된 API 제공
- **SAXParserFactory 추상 클래스**
  - 목적: SAX 구문 분석기 객체의 환경 설정 및 생성
  - SAXParserFactory **SAXParserFactory.newInstance**()
    - 시스템 프로퍼티 "javax.xml.parsers.SAXParserFactory"로 지정된 팩토리 클래스 객체 생성.
    - 시스템 프로퍼티가 지정되지 않은 경우 디폴트로 내장된 팩토리 클래스 객체 생성.
  - SAXParser **newSAXParser**() throws ParserConfigurationException, SAXException
  - boolean **validating** ⇔
    - 생성될 SAX 구문분석기의 검증용 구문분석기 여부
  - boolean **namespaceAware** ⇔
    - 생성될 SAX 구문분석기의 XML 이름공간 지원 여부

# JAXP API (2)

- **SAXParser** 추상 클래스
  - 목적: org.xml.sax.Parser 클래스의 포장(wrapping) 클래스
  - `boolean` **`isValidating`**`()`
  - `boolean` **`isNamespaceAware`**`()`
  - `org.xml.sax.Parser` **`getParser`**`() throws SAXException`
    - 본래의 SAX 구문분석기 구현 객체
  - **`parse`**`(java.io.InputStream is, HandlerBase hb) throws SAXException, java.io.IOException`
  - **`parse`**`(java.lang.String uri, HandlerBase hb) throws SAXException, java.io.IOException`
  - **`parse`**`(java.io.File f, HandlerBase hb) throws SAXException, java.io.IOException`
  - **`parse`**`(InputSource is, HandlerBase hb) throws SAXException, java.io.IOException`

# JAXP API (3)

- **DocumentBuilderFactory** 추상 클래스
  - 목적: SAX 구문 분석기 객체의 환경 설정 및 생성
  - DocumentBuilderFactory **DocumentBuilderFactory.newInstance**()
    - 시스템 프로퍼티 "javax.xml.parsers.DocumentBuilderFactory"로 지정된 팩토리 클래스 객체 생성.
    - 시스템 프로퍼티가 지정되지 않은 경우 디폴트로 내장된 팩토리 클래스 객체 생성.
  - DocumentBulder **newDocumentBuilder**() throws ParserConfigurationException
  - boolean **validating** ⇔
    - 생성될 DOM 구문분석기의 검증용 구문분석기 여부
  - boolean **namespaceAware** ⇔
    - 생성될 DOM 구문분석기의 XML 이름공간 지원 여부

# JAXP API (4)

- **DocumentBuilder** 추상 클래스
  - 목적: XML 문서로부터 DOM 트리 객체 생성
  - `boolean` **`isValidating`**`()`
  - `boolean` **`isNamespaceAware`**`()`
  - **`setEntityResolver`**`(EntityResolver resolver)`
  - **`setErrorHandler`**`(ErrorHandler handler)`
  - `Document` **`newDocument`**`()`
  - `Document` **`parse`**`(java.io.InputStream is) throws SAXException, java.io.IOException`
  - `Document` **`parse`**`(java.lang.String uri) throws SAXException, java.io.IOException`
  - `Document` **`parse`**`(java.io.File f) throws SAXException, java.io.IOException`
  - `Document` **`parse`**`(InputSource is) throws SAXException, java.io.IOException`

# JAXP API (5)

## JAXP 1.0 SAX 구문분석기 실행 예

```
C:\example\xml> set classpath=.;c:\jaxp1.0\jaxp.jar;C:\jaxp1.0\parser.jar
C:\example\xml> java SAXTest -v intro.xml
...
```

(문서 위치 설정)<?xml version="1.0" encoding="euc-kr"?>
(문서 시작)(외부 개체: null file:/home/dtkim/research/java/notes/example/xml/seminar.dtd)<seminar>(시작)
(공백류)<title>(시작)XML 문서 처리(문자)</title>(끝)
(공백류)<author>(시작)김덕태(문자)</author>(끝)

(공백류)<section>(시작)
(공백류)<head>(시작)XML 문법의 기초(문자)</head>(끝)
(공백류)<itemize>(시작)
  (공백류)<item>(시작) XML 문서내에 다음과 같은 XML 특수 기호를 사용하기 위해서는
    개체 참조를 이용하거나,
(문자)<example>(시작)
  System.out.println(1 (문자)<(문자) 2);
  System.out.println(true (문자)&(문자)&(문자) false)
(문자)</example>(끝)
  (문자)</item>(끝)
  (공백류)<item>(시작) CDATA 섹션을 이용하거나,
(문자)<example>(시작)
  System.out.println(1 < 2);
  System.out.println(true && false)
(문자)
(문자)</example>(끝)

(문자)</item>(끝)

(공백류)<item>(시작) 처리기 응용의 처리에 맡긴다.

(문자)<examplefile file="xml\Test.java">(시작)</examplefile>(끝)

(문자)</item>(끝)

(공백류)<item>(시작) `가' 대신 문자 참조 `(문자)가(문자)'라고 표현할 수 있다.

(문자)</item>(끝)

(공백류)</itemize>(끝)

(공백류)</section>(끝)

(공백류)</seminar>(끝)(문서 끝)

```
C:\example\xml> java SAXTest -v entity_test.xml
(문서 위치 설정)<?xml version="1.0" encoding="euc-kr"?>
(문서 시작)(외부 개체: null file:/home/dtkim/research/java/notes/example/xml/seminar.dtd)<seminar>
(공백류)<title>(시작)XML 개체 처리(문자)</title>(끝)

(공백류)<author>(시작)김덕태(문자)</author>(끝)

(공백류)<section>(시작)
(공백류)<head>(시작)개체의 의미(문자)</head>(끝)
(공백류)<itemize>(시작)
   (공백류)<item>(시작) 개체의 내용은 요소, 문자 데이타, CDATA 섹션들로 이루어진
   내용으로 정의될 수 있다.
   (문자)</item>(끝)
(공백류)</itemize>(끝)
(공백류)</section>(끝)

(공백류)(외부 개체: null file:/home/dtkim/research/java/notes/example/xml/section2.xml)

(공백류)<section>(시작)
(공백류)<head>(시작)외부 개체 정의(문자)</head>(끝)
(공백류)<itemize>(시작)
   (공백류)<item>(시작) 이와 같이 외부에 정의된 개체를 개체 참조를 통하여 포함시킬
   수 있다.
   (문자)</item>(끝)
(공백류)</itemize>(끝)
(공백류)</section>(끝)
(공백류)</seminar>(끝)(문서 끝)
```

## XML for Java 3.0.1의 SAX 구문분석기 실행 예

```
C:\example\xml> set classpath=.;C:\XML4J_3_0_1\xerces.jar

C:\example\xml> java SAXTest2 org.apache.xerces.parsers.SAXParser intro.xml
...
```

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import java.io.*;

// Usage: java SAXTest [-v|-n] <xml file> [output encoding] [-nodebug]
public class SAXTest implements DocumentHandler, ErrorHandler,
    DTDHandler, EntityResolver {

    public static void main(String[] args) throws Exception {

        try {
             SAXParserFactory factory = SAXParserFactory.newInstance();
            factory.setValidating(false);
            factory.setNamespaceAware(false);
            int i = 0;
            for (; args[i].startsWith("-"); i++) {
                if (args[i].equals("-v"))
                    factory.setValidating(true);
                else if (args[i].equals("-n"))
                    factory.setNamespaceAware(true);
            }

            String xmlFile = args[i++];
            String outEncoding = "euc-kr";
            if (i < args.length)
                outEncoding = args[i++];
            SAXTest app = new SAXTest(outEncoding);
            if (i < args.length)
                app.debug = false;
```

```
            // XML 문서 내용 설정
            File file = new File(xmlFile);
            // InputSource input = new InputSource(file.toURL().toString());
            // toURL()은 JDK 1.2 메쏘드이므로, 다음과 같이 한다.
            String path = file.getAbsolutePath();
            path.replace(File.separatorChar, '/');
            if (path.length() > 0 && path.charAt(0) != '/')
                path = '/' + path;
            input = new InputSource("file:" + path);

            // XML 구문 분석기 생성
            SAXParser jaxParser = factory.newSAXParser();
            Parser parser = ParserFactory.makeParser(parserClass);
            parser.setDocumentHandler(app);
            parser.setErrorHandler(app);
            parser.setDTDHandler(app);
            parser.setEntityResolver(app);
            parser.parse(input);
        } catch (SAXParseException ex) {
            System.err.println("==> 오류: " + ex.getSystemId() + ":"
                               + ex.getLineNumber() + ":"
                               + ex.getMessage());
            ex.printStackTrace(System.err);
        } catch (SAXException ex) {
            System.err.println("==> 오류: " + ex.getMessage());
            ex.printStackTrace(System.err);
        }
    }
```

```
String outEncoding;
PrintWriter writer;
Locator locator;
boolean debug = true;

public SAXTest(String outEncoding)
    throws UnsupportedEncodingException {
    this.outEncoding = outEncoding;
    writer = new PrintWriter(
        new OutputStreamWriter(System.out, outEncoding));
}

void print(String msg) {
    writer.print(msg);
}

void print(char[] buf, int offset, int len) {
    writer.write(buf, offset, len);
}

void println(String msg) {
    writer.println(msg);
}

void printStatus(String msg) {
    if (debug) {
    writer.print("(" + msg + ")");
    writer.flush();
    }
}
```

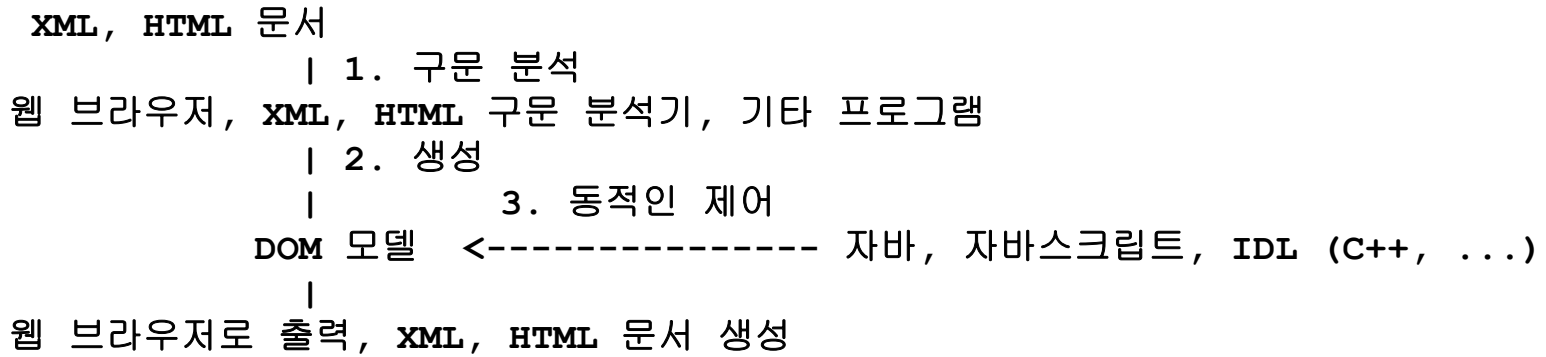
```
// DocumentHandler 인터페이스 메쏘드

   public void setDocumentLocator(Locator loc) {
       this.locator = loc;
       printStatus("문서 위치 설정");
   }

   public void startDocument() throws SAXException {
       println("<?xml version=\"1.0\" encoding=\"" + outEncoding + "\"?>");
       printStatus("문서 시작");
   }

   public void endDocument() throws SAXException {
       printStatus("문서 끝");
       println("");
       writer.flush();
   }

   public void startElement(String name, AttributeList attrs)
       throws SAXException {
       print("<" + name);

       if (attrs != null)
           for (int i = 0; i < attrs.getLength(); i++)
               print(" " + attrs.getName(i) + "=\""
                     + attrs.getValue(i) + "\"");
       print(">");
       printStatus("시작");
   }
```

```
public void endElement(String name) throws SAXException {
        print("</" + name + ">");
        printStatus("끝");
    }

    public void characters(char[] buf, int offset, int len)
        throws SAXException {
        print(buf, offset, len);
        printStatus("문자");
    }

    public void ignorableWhitespace(char buf [], int offset, int len)
        throws SAXException {
        print(buf, offset, len);
        printStatus("공백류");
    }

    public void processingInstruction(String target, String data)
        throws SAXException {
        print("<?" + target + " " + data + "?>");
        printStatus("처리기 명령");
    }
```

```
// ErrorHandler 인터페이스 메쏘드

   public void warning(SAXParseException ex)
       throws SAXParseException {
       if (writer != null)
           writer.flush();
       System.err.println("==> 경고: " + ex.getSystemId() + ":"
                          + ex.getLineNumber()
                          + ":" + ex.getMessage());
   }

   public void error(SAXParseException ex)
       throws SAXParseException {
       if (writer != null)
           writer.flush();
       System.err.println("==> 오류: " + ex.getSystemId() + ":"
                          + ex.getLineNumber()
                          + ":" + ex.getMessage());
   }

   public void fatalError(SAXParseException ex)
       throws SAXParseException {
       if (writer != null)
           writer.flush();
       System.err.println("==> 치명적 오류: " + ex.getSystemId() + ":"
                          + ex.getLineNumber()
                          + ":" + ex.getMessage());
   }
```

```
 // DTDHandler 인터페이스 메쏘드

    public void notationDecl(String name, String publicId, String systemId)
        throws SAXException {
        printStatus("표기 선언: " + publicId + " " + systemId);
    }

    public void unparsedEntityDecl(
        String name, String publicId, String systemId,
        String notationName) throws SAXException {
        println("<!ENTITY  " + name + " " + publicId + " " + systemId
              + " NDATA " + notationName);
        printStatus("구문분석되지 않는 개체");
    }

    // EntityResolver 인터페이스 메쏘드

    public InputSource resolveEntity(String publicId, String systemId)
        throws SAXException, IOException {
        printStatus("외부 개체: " + publicId + " " + systemId);
        return null;
    }
}
```

# 문서 객체 모델 (DOM)




- 목적: **XML** 및 **HTML** 문서의 동적인 변경 및 제어를 위한 플랫폼 및 언어 독립적인 표준
- **XML** 문서의 구조를 메모리내에서 나무 구조로 표현하기 위한 문서 객체 모델 표준의 일종
- 용도: 브라우저등에서 출력되는 문서의 동적인 변경 및 상호 작용, **XML, HTML** 문서 편집기
- 홈 페이지: **http://www.w3.org/DOM/**
- **XML DOM** 구문 분석기는 **XML** 문서를 구문 분석하여, **DOM** 모델로 변환한다.

# 문서 객체 모델 (DOM) (2)

- 종류
  - Document Object Model (Core) Level 1
    - XML 문서의 DOM 모델
    - 필수적으로 구현되어야 하는 기본 API와 선택적으로 구현할 수 있는 확장 API로 나뉘어진다.
  - Document Object Model (HTML) Level 1
    - HTML 문서의 DOM 모델
    - DOM (Core) Level 1의 기본 API를 포함하고 HTML의 표현에 편리한 API가 추가되어 있다.
- 자바, 자바스크립트 **(ECMA Script)**, 코바 **IDL**에 대한 표준 **API**가 정의되어 있다. **(**코바 **IDL**은 다양한 언어에서의 표준 **API**를 간단히 정의하기 위해 그 문법을 차용한 것 뿐임**)**
- **DOM** 객체의 생성
  - XML, HTML 문서로부터 DOM 구문 분석기를 통한 DOM 객체 생성
  - XML 및 HTML 문서로부터 웹 브라우저에 의한 생성 기타 프로그램의 필요에 따른 생성

# 문서 객체 모델 (DOM) (3)

- **DOM 모델의 일반적인 구조**
  - 문서 내용을 개념상 여러개의 나무 구조로 표현한다.
  - 나무 구조의 각 노드는 Node 인터페이스의 객체로 구성된다.
  - 나무 구조의 루트 노드는 Document 인터페이스의 객체이다.

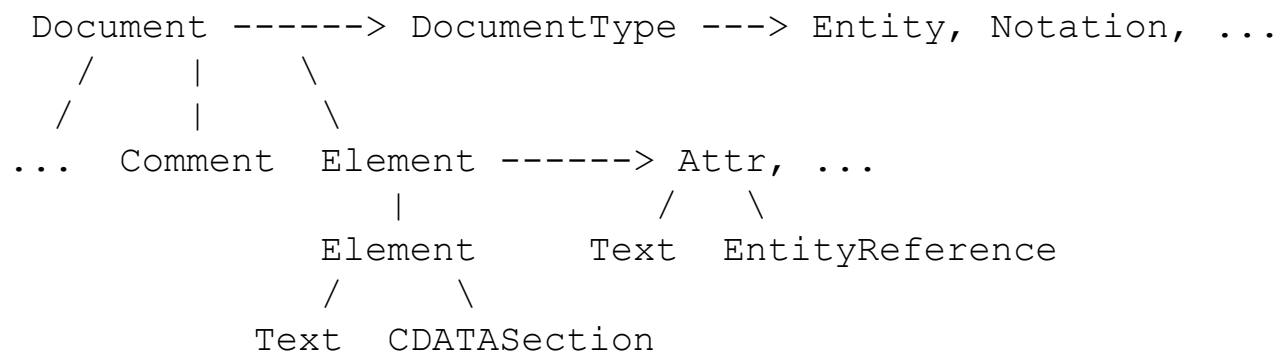
# 자바 DOM (Core) Level 1 API (1)

- **org.w3c.dom** 패키지.
- 예외 클래스를 제외하고 인터페이스만이 정의되어 있다.
- 인터페이스 계층도

```
Node
  +-- Document
  +-- Element
  +-- Attr
  +-- CharacterData
  |         +-- Text
  |         |     +-- CDATASection
  |         +-- Comment
  |
  +-- EntityReference
  +-- ProcessingInstruction
  +-- DocumentFragment
  |
  +-- DocumentType
  +-- Entity
  +-- Notation

RuntimeException
DOMException
```

# 자바 DOM (Core) Level 1 API (2)

## DOM 모델의 구조




## DOMImplementation 인터페이스

- 목적: 특정 DOM 구현 패키지의 정보 검색
- boolean hasFeature(String feature, String version)
  - DOM 구현 패키지가 주어진 기능을 제공하면 true.
  - feature의 값으로 "XML" 혹은 "HTML"을 사용할 수 있고, version의 값으로 Level 1에 대해서는 "1.0"으로 표현한다.

# 자바 DOM (Core) Level 1 API (3)

- **Node 인터페이스**
  - 목적: 모든 노드의 최상위 인터페이스
  - `short` **`getNodeType`**`()`
    - 노드의 타입. 각 하위 인터페이스 노드에 대하여 다음과 같은 상수가 정의되어 있다.
    - DOCUMENT_NODE, DOCUMENT_FRAGMENT_NODE, ELEMENT_NODE, ATTRIBUTE_NODE, TEXT_NODE, CDATA_SECTION_NODE, COMMENT_NODE, ENTITY_NODE, ENTITY_REFERENCE_NODE, NOTATION_NODE, PROCESSING_INSTRUCTION_NODE, DOCUMENT_TYPE_NODE
  - `String` **`getNodeName`**`()`
    - : 노드의 이름 (태그 이름등)
  - `String` **`nodeValue`** <-> throws DOMException
  - `NamedNodeMap` **`getAttributes`**`()`
  - `Node` **`getParentNode`**`()`
  - `NodeList` **`getChildNodes`**`()`
  - `boolean` **`hasChildNodes`**`()`

# 자바 DOM (Core) Level 1 API (4)

- Node **getFirstChild**()
- Node **getLastChild**()
- Node **getPreviousSibling**()
- Node **getNextSibling**()
- Document **getOwnerDocument**()
- Node **insertBefore**(Node newChild, Node refChild) throws DOMException
- Node **appendChild**(Node newChild) throws DOMException
- Node **removeChild**(Node oldChild) throws DOMException
- Node **replaceChild**(Node newChild, Node oldChild) throws DOMException
- Node **cloneNode**(boolean deep)

## NamedNodeMap 인터페이스

- 목적: 이름으로 검색할 수 있는 노드 목록
- int **getLength**()
- Node **getNamedItem**(String name)
- Node **setNamedItem**(Node arg) throws DOMException
- Node **removeNamedItem**(String name) throws DOMException
- Node **item**(int index)

# 자바 DOM (Core) Level 1 API (5)

- **NodeList 인터페이스**
  - 목적: 자식 노드들을 순서대로 나타내기 위한 노드 목록
  - `int` **`getLength`**`()`
  - `Node` **`item`**`(int index)`
- **Document 인터페이스**
  - 목적: 단일 문서 모델내에 포함될 노드 객체 생성 및 검색
  - 이 객체의 생성을 위한 표준 API가 정해져 있지 않으므로, 각 구현에 따라 정해진 방법을 사용하여 생성하여야 한다.
  - `DOMImplementation` **`getImplementation`**`()`
  - `DocumentType` **`getDoctype`**`()`
  - `Element` **`getDocumentElement`**`()`
  - `NodeList` **`getElementsByTagName`**`(String tagname)`
    - 문서내의 주어진 이름을 갖는 모든 노드 목록
  - `Element` **`createElement`**`(String tagName) throws DOMException`
  - `Attr` **`createAttribute`**`(String name) throws DOMException`
  - `DocumentFragment` **`createDocumentFragment`**`()`
  - `Text` **`createTextNode`**`(String data)`

# 자바 DOM (Core) Level 1 API (6)

- `Comment createComment(String data)`
- `CDATASection createCDATASection(String data) throws DOMException`
- `ProcessingInstruction createProcessingInstruction(String target, String data) throws DOMException`
- `EntityReference createEntityReference(String name) throws DOMException`

## Element 인터페이스

- 목적: 문서의 요소(태그) 노드
- `String getTagName()`
- `String getAttribute(String name)`
- `setAttribute(String name, String value)`
- `Attr getAttributeNode(String name)`
- `Attr setAttributeNode(Attr newAttr)`
- `removeAttribute(String name)`
- `Attr removeAttributeNode(Attr oldAttr)`
- `NodeList getElementsByTagName(String name)`
  - 주어진 태그 이름을 갖는 모든 후손 Element 노드 목록
- `normalize()`
  - 인접한 모든 Text 노드를 하나의 Text 노드로 병합한다.

# 자바 DOM (Core) Level 1 API (7)

- **Attr 인터페이스**
  - 목적: Element 노드의 애트리뷰트
  - 부모, 형제 노드를 갖지 않는다.
  - 자식 노드로 Text, EntityReference 노드를 가질 수 있다.
  - `String` **`getName`**`()`
  - `boolean` **`getSpecified`**`()`
    - 이 애트리뷰트가 Element 노드와 함께 명시적으로 값이 지정된 경우에 true
  - `String` **`getValue`**`()`
  - **`setValue`**`(String value)`
- **CharacterData 인터페이스**
  - 목적: 문자 데이타 노드(Text, Comment)의 공통 상위 인터페이스
  - `int` **`getLength`**`()`
  - `String` **`getData`**`() throws DOMException`
  - **`setData`**`(String data) throws DOMException`
  - **`insertData`**`(int offset, String arg) throws DOMException`

# 자바 DOM (Core) Level 1 API (8)

- `String substringData(int offset, int count) throws DOMException`
- **`appendData`**`(String arg) throws DOMException`
- **`deleteData`**`(int offset, int count) throws DOMException`
- **`replaceData`**`(int offset, int count, String arg) throws DOMException`

## Text 인터페이스

- 목적: 일반 문자 데이타 노드
- `Text splitText(int offset) throws DOMException`
  - 2개의 Text 노드로 분리시켜 이 노드의 다음 형제 노드로 추가.

## CDATASection 인터페이스

- 목적: Text 노드의 data 속성을 나타내는 CDATA 섹션
- CDATA 섹션은 마크업 명령을 무시하며, CDATA의 마지막에 오는 "]]>" 문자열만이 유일한 구분자로 해석된다.

## Comment 인터페이스

- 목적: 문서 주석을 나타내는 노드

## EntityReference 인터페이스

- 목적: Entitty 노드로의 참조 노드

# 자바 DOM (Core) Level 1 API (9)

- **ProcessingInstruction** 인터페이스
  - `String` **`getTarget`**`()`
  - **`setData`**`(String data) throws DOMException`
  - `String` **`getData`**`()`
- **DocumentFragment** 인터페이스
  - 목적: 문서의 부분
- **DocumentType** 인터페이스
  - 목적: 문서 타입 정의 (Document Type Definition)를 나타내는 노드의 인터페이스
  - `String` **`getName`**`()`
  - `NamedNodeMap` **`getEntities`**`()`
  - `NamedNodeMap` **`getNotations`**`()`
- **Entity** 인터페이스
  - 목적: 문서 타입 정의의 (구문 분석된 혹은 안된) 개체 부모 노드를 갖지 않는다.
  - `String` **`getNotationName`**`()`
  - `String` **`getPublicId`**`()`
  - `String` **`getSystemId`**`()`

# 자바 DOM (Core) Level 1 API (10)

- **Notation 인터페이스**
  - 목적: 구문 분석되지 않은 개체에 대한 형식 및 ProcessingInstruction 노드의 타겟 선언
  - 부모 노드를 갖지 않는다.
  - `String` **`getPublicId`**`()`
  - `String` **`getSystemId`**`()`
- **DOMException 클래스 extends RuntimeException**

# XML DOM 자바 구문 분석기 실행 예

- **JAXP 1.0 DOM 구문분석기 실행 예**

```
C:\example\xml> set classpath=.;c:\jaxp1.0\jaxp.jar;C:\jaxp1.0\parser.jar
C:\example\xml> java DOMTest -v intro.xml
{Document:{Element:<seminar>{Text:
{Element:<title>{Text:XML 문서 처리{Text:
{Element:<author>{Text:김덕태{Text:

{Element:<section>{Text:
{Element:<head>{Text:XML 문법의 기초{Text:
{Element:<itemize>{Text:
   {Element:<item>{Text: XML 문서내에 다음과 같은 XML 특수 기호를 사용하기 위해서는
        개체 참조를 이용하거나,
{Element:<example>{Text:
    System.out.println(1 < 2);
    System.out.println(true && false)
{Text:
   {Text:
   {Element:<item>{Text: CDATA 섹션을 이용하거나,
{Element:<example>{CDATASection:
    System.out.println(1 < 2);
    System.out.println(true && false)
```

```
{Text:
{Text:
    {Text:
    {Element:<item>{Text: 처리기 응용의 처리에 맡긴다.
{Element:<examplefile>{Text:
    {Text:
   {Element:<item>{Text: `가' 대신 문자 참조 `가'라고 표현할 수 있다.
   {Text:
{Text:
{Text:
{Comment:
    XML 예제 화일 주석 예
{Element:<seminar>{Text:
{Element:<title>{Text:XML 문서 처리{Text:
{Element:<author>{Text:김덕태{Text:

{Element:<section>{Text:
{Element:<head>{Text:XML 문법의 기초{Text:
{Element:<itemize>{Text:
   {Element:<item>{Text: XML 문서내에 다음과 같은 XML 특수 기호를 사용하기 위해서는
        개체 참조를 이용하거나,
{Element:<example>{Text:
    System.out.println(1 < 2);
    System.out.println(true && false)
{Text:
   {Text:
   {Element:<item>{Text: CDATA 섹션을 이용하거나,
```

```
{Element:<example>{CDATASection:
    System.out.println(1 < 2);
    System.out.println(true && false)

{Text:
{Text:
    {Text:
    {Element:<item>{Text: 처리기 응용의 처리에 맡긴다.
{Element:<examplefile>{Text:
    {Text:
   {Element:<item>{Text: `가' 대신 문자 참조 `가'라고 표현할 수 있다.
   {Text:
{Text:
{Text:
```

## Docuverse DOM SDK 1.0A5 DOM 구문분석기 실행 예

```
C:\example\xml> set classpath=.;c:\domsdk_a5\domsdk.jar
;c:\domsdk_a5\w3cdom1.jar;c:\jaxp1.0\jaxp.jar;c:\jaxp1.0\parser.jar
C:\example\xml> java DOMTest -Djavax.xml.parsers.DocumentBuilderFactory
=com.docuverse.dom.DocumentBuilderFactoryImpl -v intro.xml
...
```

## IBM XML Parser for Java 3.0.1 DOM 구문분석기 실행 예

```
C:\example\xml> set
classpath=.;c:\XML4J_3_0_1\xerces.jar;c:\jaxp1.0\jaxp.jar
C:\example\xml> java -Djavax.xml.parsers.DocumentBuilderFactory
=IBMDocumentBuilderFactoryImpl DOMTest -v intro.xml

...
```

```
import org.w3c.dom.*;
import org.xml.sax.*;

import java.io.*;

public class DOMTest {
    public static void main(String[] args) throws Exception {

        // XML 문서 내용 설정
        File file = new File(args[2]);
        // input = new InputSource(file.toURL().toString());
        // toURL()은 JDK 1.2 메쏘드이므로, 다음과 같이 한다.
        String path = file.getAbsolutePath();
        path.replace(File.separatorChar, '/');
        if (path.length() > 0 && path.charAt(0) != '/')
            path = '/' + path;
        InputSource input = new InputSource("file:" + path);

        Document doc = DOMFactory.openDocument(args[0], args[1], input);

        print(doc);
    }

    public static void print(Node node) {
        if (node == null)
            return;
        int type = node.getNodeType();
```

```
switch(type) {
case Node.DOCUMENT_NODE:
    print((Document) node);
    break;
case Node.DOCUMENT_FRAGMENT_NODE:
    print((DocumentFragment) node);
    break;
case Node.ELEMENT_NODE:
    print((Element) node);
    break;
case Node.ATTRIBUTE_NODE:
    //print((Attribute) node);
    break;
case Node.TEXT_NODE:
    print((Text) node);
    break;
case Node.CDATA_SECTION_NODE:
    print((CDATASection) node);
    break;
case Node.COMMENT_NODE:
    print((Comment) node);
    break;
case Node.ENTITY_NODE:
    //print((Entity) node);
    break;
case Node.ENTITY_REFERENCE_NODE:
    //print((EntityReference) node);
    break;
```

```
        case Node.NOTATION_NODE:
            //print((Notation) node);
            break;
        case Node.PROCESSING_INSTRUCTION_NODE:
            //print((ProcessingInstruction) node);
            break;
        case Node.DOCUMENT_TYPE_NODE:
            //print((DocumentType) node);
        }
    }

    public static void print(NodeList nodeList) {
        int num = nodeList.getLength();
        for (int i = 0; i < num; i++) {
            Node node = nodeList.item(i);
            print(node);
        }
    }

    public static void print(Document doc) {
        print("{Document:");
        DocumentType docType = doc.getDoctype();
        Element root = doc.getDocumentElement();
        print(docType);
        print(root);
        print(doc.getChildNodes());
        print("}");
    }
```

```
public static void print(DocumentType docType) {
     if (docType == null)
         return;
     print("{DocumentType:");
     print(docType.getNodeName());
     //print(docType.getName());
     //NamedNodeMap entities = docType.getEntities();
     //NamedNodeMap notations = docType.getNotations();
     //print(entities);
     //print(notations);
     print(docType.getChildNodes());
     print("}");
 }

 public static void print(Element elem) {
     print("{Element:");
     print("<" + elem.getTagName() + ">");
     print(elem.getChildNodes());
     print("}");
 }

 public static void print(Text text) {
     print("{Text:");
     print(text.getData());
     print(text.getChildNodes());
     print("}");
 }
```

```
 public static void print(CDATASection cdata) {
        print("{CDATASection:");
        print(cdata.getData());
        print(cdata.getChildNodes());
        print("}");
    }

    public static void print(Comment comment) {
        print("{Comment:");
        print(comment.getData());
        print(comment.getChildNodes());
        print("}");
    }

    public static void print(String val) {
        System.out.print(val);
    }
}
```

# 자바 DOM (HTML) Level 1 API

- **org.w3c.dom.html** 패키지.
- **HTML** 문서의 각 요소(태그) 종류에 대하여 **org.w3c.dom** 패키지의 하위 인터페이스가 정의되어 있다.
- **DOM (HTML)**을 지원하는 **XML** 자바 **DOM** 구문분석기로 **HTML** 문서를 구문분석하여 **DOM (HTML)** 객체 모델을 생성할 수 있다.

# XHTML (1)

- 명세 문서
  - http://www.w3.org/TR/xhtml1/
- MIME 타입: text/html, text/xml, application/xml
- XHTML 1.0 규격
  - HTML 4.0 문서 규격을 XML 문서 규격을 만족하도록 수정한 문서 규격
  - Strict, Transitional, Frameset등 3가지 문서 규격중 한가지를 만족시켜야 한다.
  - 루트 요소는<html>이어야 한다.
  - 루트 요소의 xmlns 속성으로 XHTML 1.0 이름 공간인 http://www.w3.org/1999/xhtml을 지정한다.
  - 루트 요소 이전에 다음중 하나의 DTD를 포함시켜야 한다.

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN” "DTD/xhtml1-strict.dtd">
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN‟ "DTD/xhtml1-transitional.dtd">
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN‟ "DTD/xhtml1-frameset.dtd">
```

# XHTML (2)

- <a>, <applet>, <form>, <frame>, <iframe>, <img>, <map> 요소의 요소 식별자로 “name” 속성 대신에 “id” 속성 사용
- HTML과 XML 문서 규격을 동시에 만족하는 문서를 만들 수 있다.
- 요소 이름과 속성 이름은 모두 소문자이어야 한다.

## 예: xhtml.html

```
<?xml version="1.0" encoding="euc-kr"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ko" lang="ko">
  <head>
    <meta http-equiv="Content-type" content='text/html; charset="euc-kr"' />
    <title>XHTML 소개</title>
  </head>
  <body>
    XHTML 1.0은 HTML 4.0 문서 규격을 XML 문서 규격을 만족하도록 수정한 문서 규격이다.
  </body>
</html>
```

# XHTML (3)

- **XHTML** 문서에 다른 규격의 **XML** 문서를 포함시킬 수 있다. **(**단, 현재로서는 **XHTML** 규격을 엄격히 따르는 문서가 아님**)**
  - 예: xhtml-xmlns.html

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>A Math Example</title>
  </head>
  <body>
    <p>The following is MathML markup:</p>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply> <log/>
        <logbase>
          <cn> 3 </cn>
        </logbase>
        <ci> x </ci>
      </apply>
    </math>
  </body>
</html>
```

# XHTML(4)

- 다른 규격의 **XML** 문서에 **XHTML** 문서 조각을 포함시킬 수 있다.
  - 예: xhtml-xmlns2.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6' xml:lang="en" lang="en">
  <title>Cheaper by the Dozen</title>
  <isbn:number>1568491379</isbn:number>
  <notes>
    <!-- make HTML the default namespace for a hypertext commentary -->
    <p xmlns='http://www.w3.org/1999/xhtml'>
       This is also available <a href="http://www.w3.org/">online</a>.
    </p>
  </notes>
</book>
```

# XHTML(5)

## Tidy

- HTML 문서를 XHTML 문서로 변환해주는 도구
- http://www.w3.org/People/Raggett/tidy/
- 예제: untidy.html

```
<html> <head>
<title>test</title>
</head>

<body>
<H1>Heading 1</H1>
<p>this is a paragraph.
<ul>
  <li> item 1
  <li> item 2
</ul>
</body> </html>
```

# XHTML(6)

- 실행 예

```
C:\example\xml> set CLASSPATH=c:\tools\xml\JTidy\lib\Tidy.jar
C:\example\xml> java org.w3c.tidy.Tidy -asxml untidy.html
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta name="generator" content="HTML Tidy, see www.w3.org" />
<title>test</title>
</head>
<body>
<h1>Heading 1</h1>

<p>this is a paragraph.</p>

<ul>
<li>item 1</li>

<li>item 2</li>
</ul>
</body>
</html>
```

# XML 문서의 HTML 변환 표시 방법

- 정적인 변환
  - MSXSL
    - XML과 XSL을 결합해서 HTML 문서를 생성하는 툴
    - http://www.microsoft.com/xml/xsl/downloads/msxsl.asp
- 동적인 변환
  - 서버측 혹은 클라이언트측에서 수행될 수 있다.
  - MSXSL 명령행 유틸리티
  - 액티브X 컨트롤
  - 자바 애플릿
  - 자바 스크립트

# 스타일 시트

- 정의
  - 문서가 어떤 방식으로 표시될 것인지를 규정하는규칙들의 집합
- 종류
  - XSL (eXtensible Markup Language)
  - CSS (Cascading Style Sheet)
  - DSSSL
- 장점
  - 문서의 내용과 표시 방법의 분리를 통한 문서의 명료화
  - 다운로드 시간, 네트워크 체증, 서버 부담 감소
  - 동일 문서를 목적에 따라 다양한 방식으로 표현 가능
  - 하나의 스타일 시트를 변경시킴으로써 이 스타일 시트를 따른 모든 문서의 일괄적인 변경
- 특징
  - XSL, CSS는 XML에 적용될 수 있다.
  - CSS는 HTML에 적용 가능하나, XSL은 그렇지 않다.
  - XSL은 문서 변환 기술 언어로도 사용될 수 있으나 CSS는 그렇지 않다.
  - CSS 문서는 XSL 문서로 변환될 수 있다.

# XML 문서에 스타일 시트 명시 (1)

- **XML** 문서에 스타일 시트 명시 명세 문서
  - http://www.w3.org/TR/xml-stylesheet/
- **XML** 문서에 **xml-stylesheet** 처리기 명령으로 **XSL, CSS**등의 스타일 시트를 명시할 수 있다.
  - 예) <?xml-stylesheet href="mystyle.css" type="text/css"?>
- **xml-stylesheet** 처리기 명령의 애트리뷰트
  - href CDATA #REQUIRED
  - type CDATA #REQUIRED
  - title CDATA #IMPLIED
  - media CDATA #IMPLIED
  - charset CDATA #IMPLIED
  - alternate (yes|no) "no“

# XML 문서에 스타일 시트 명시 (2)

- **XML 문서에 CSS 명시 예: intro-css.xml**

```
<?xml version="1.0" encoding="euc-kr"?>
<?xml-stylesheet href="seminar.css" type="text/css"?>
<!DOCTYPE seminar SYSTEM "seminar.dtd">
...
```

- **예제 CSS 스타일 시트: seminar.css**

```
seminar {display: block; font-size: 14pt; background-color: white;
         color: black; }
title {display: block; font-size: 32pt; font-weight: bold;
      text-align: center}
author {display: block; text-align: left; font-size}
section {display: block;}
head {display: block; font-size: 24pt; font-weight: bold;
      text-align: center}
itemize {display: block;}
item {display: list-item; list-style-type: circle;}
example {display: block; white-space: pre; border: 1pt solid;}
examplefile {display: block}
```

# XSL

- **XSL: Extensible Stylesheet Language**
- 목적: **XML** 문서를 표시 정보를 갖는 **XML** 문서 타입으로 변환하기 위한 스타일 시트 언어
- **XSL**은 다음 **2**가지의 독립적인 언어로 구성된 **XML** 문서이다.
  - 변환 언어(XSLT): 하나의 XML 문서를 다른 XML 문서로 변환
  - 포매팅 언어
- **XSL** 문서 자체는 **XML** 문서의 일종이다.
- **XSL** 명세 문서: **http://www.w3.org/Style/XSL/**
- 단점
  - XSL은 HTML 등과 같이 XML 문법을 갖는 문서로만 변환할 수 있다.
  - 정교한 포매팅은 곤란하다.
  - XSL에 스크립트 언어를 사용하거나, XSL을 사용하는 대신 XML 처리기 응용을 구현하여 표시하거나, XSL로 변환된 문서를 별도의 XML 처리기 응용으로 표시하여 해결될 수 있다.

# XSL Transformations (XSLT)

- **XSL** 규격의 일부
- **XML** 문서(소스 트리)를 다른 **XML** 문서(결과 트리)로의 변환을 기술하기 위한 **XML** 문서의 일종
- **MIME** 타입: **text/xml, application/xml**
- **XSL Transformations (XSLT) Version 1.0** 명세 문서
  - http://www.w3.org/TR/xslt
- **XML Path Language (XPath)**
  - http://www.w3.org/TR/xpath
- **XSLT** 이름 공간
  - URI: http://www.w3.org/1999/XSL/Transform
  - 이름 공간 프리픽스로 `xml:'을 사용하는 것이 관례.
- 루트 요소는 **<xsl:stylesheet>** 혹은 **<xsl:transform>** 사용

# XSL 처리기

- 목적: **XML** 문서와 **XSL** 문서를 구문분석하여 변환된 **XML** 문서 생성
- **XSL** 처리기
  - LotusXSL
    - http://www.alphaworks.ibm.com/formula/LotusXSL
  - XSLP
    - http://www.clc-marketing.com/xslp/
  - James Clark의 XT
    - http://www.jclark.com/xml/xt.html
- **LotusXSL**의 실행 예
  - 실행 후의 출력 파일 out.html을 웹 브라우저로 방문한다.

```
c:\example\xml> set
CLASSPATH=c:\lotusxsl_1_0_0\xalan.jar;c:\lotusxsl_1_0_0\xerces.jar
c:\example\xml> java org.apache.xalan.xslt.Process -in intro.xml -xsl
seminar2html.xml > out.html
```

## 예제: seminar2html.xml

```
<?xml version="1.0" encoding="euc-kr"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/seminar">
   <html>
      <head>
         <meta http-equiv="Content-type" content="text/html; charset=UTF-8" />
         <title><xsl:value-of select="title"/></title>
      </head>

      <body>
         <h1 style="text-align: center;"><xsl:value-of select="title"/></h1>
         <p>저자: <xsl:value-of select="author"/></p>
         <hr />
         <xsl:apply-templates select="section"/>
      </body>
   </html>
</xsl:template>
```

```
<xsl:template match="section">
   <xsl:apply-templates/>
   <hr />
</xsl:template>

<xsl:template match="head">
   <h2 style="text-align: center;"><xsl:value-of select="."/></h2>
</xsl:template>

<xsl:template match="itemize">
   <ul>
      <xsl:apply-templates/>
   </ul>
</xsl:template>

<xsl:template match="item">
   <li><xsl:apply-templates/></li>
</xsl:template>
```

```
<xsl:template match="example">
   <!-- <p style="border: 1pt solid; padding: 1ex; white-space:
pre; font-family: monospace;"> -->
   <pre style="white-space: pre; border: 1pt solid;"><xsl:value-of
select="."/></pre>
</xsl:template>


<xsl:template match="examplefile">
   <p> 예제)
   <a> <xsl:attribute name="href"><xsl:value-of
select="translate(@file, '\', '/')"/>
   </xsl:attribute><xsl:value-of select="@file"/></a></p>
</xsl:template>
</xsl:stylesheet>
```

# XSL 포매팅 언어

- **XSL** 포매팅 객체 변환기
  - FOP: A Formatting Object to PDF Translator
    - http://www.jtauber.com/fop/
- **XSL** 문서는 특정 문서 타입의 **XML** 문서를 요소 나무 구조를 결과 나무 구조로 변환한 후, 결과 나무 구조를 목적 **XML** 문서로 변환하기 위한 변환 규칙을 정의한다.

# 리소스

- **XML** 홈 페이지: **http://www.w3.org/XML/**
- **XML** 사이트: **http://www.xml.com/xml/pub**
- **XML** 사이트: **http://www.oasis-open.org/cover/sgml-xml.html**
- **XML S/W** 모음: **http://www.xmlsoftware.com/**
- **XML** 관련 정보: **http://www.datachannel.com/**
- **SGML/XML** 홈 페이지: **http://www.sil.org/sgml/**
- **XML FAQ: http://xml.t2000.co.kr/faq/index.html**
- **XML** 뉴스 그룹: **news:comp.text.xml**
- **XML** 사이트: **http://developerlife.com/**
- **``XML Bible,'' Elliotte Rusty Harold, IDG Books Worldwide, 1999**
- **``Professional XML Applications,'' Frank Boumphrey** 외 **11**인, 류광역, 정보문화사, **1999**
- **XML** 메일링 리스트 아카이브
  - http://www.lists.ic.ac.uk/hypermail/xml-dev
- **XML FAQ: http://www.ucc.ie/xml**