<JSTORM>

JDOM을 이용한 즐거운 JAVA/XML 프로그래밍<2부>

*Revision <**1.0**>*

중앙대학교 컴퓨터공학과
자바 동호회
JSTORM
http://www.jstorm.pe.kr

Document Information

Document title:	JDOM을 이용한 즐거운 Java/XML프로그래밍 2부
Document file name:	jdom2_jinho.pdf
Revision number:	<1.0>
Issued by:	<최진호>(csecau@orgio.net)
Issued Date:	<2001/02/11>
Status:	Final

Content Information

Audience:	중급, 고급
Abstract:	Java로 XML프로그래밍을 해봤다면 그 불필요하게 복잡한 코딩이 짜증나지 않았는지? 아무리 SAX와 DOM이 표준이지만 자바는 자바만의 특성을 가진 API가 있어야 하지 않을까? 여기 JDOM이라는 매력적인 도구가 당신을 앞으로 즐거운 XML의 세계로 인도할 것이다. DOM보다 빠르고 경량이며 SAX보다 편리한 JDOM의 세계로 당신을 초대한다.
Reference:	1)http://www.javaworld.com/javaworld/jw-07-2000/jw-0728-jdom2_p.html 2)http://jdom.org/ 3)http://www.w3.org/DOM/
참고 사항	본 문서는 자바월드의 상기 레퍼런스 1번의 문서를 번역한 문서임을 밝힌다.

Document Approvals

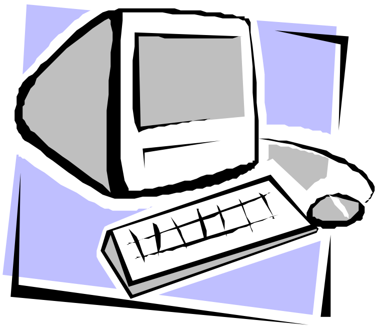
박지훈

고문	Signature	date
지위	Signature	date

Revision History

Revision	Date	Author	Description of change

Table of Contents



2부

JDOM을 이용해서 즐겁게 java/xml 프로그래밍하기

JDOM은 자바로 XML을 읽고 쓰고 제어하는 새로운 API입니다. 2000년 4월에 Jason Hunter와 Brett McLaughlin이 만들었구요. 보다 직관적이고 간단하며 생산적으로 XML을 요리해보자는 취지에서 만들어진 것이죠. 본 문서에서는 1,2부로 나눠서 어떻게 JDOM을 사용해서 XML을 읽을 것인가하는 주제를 1부에 담을 것이고 2부에서는 JDOM을 이용해서 XML데이타를 만들고 수정하는 방법을 담을 것입니다.

저자: Jason Hunter ,Brett McLaughlin
번역자: 최진호 (csecau@orgio.net)

1부에서는 JDOM을 간단히(?)소개하고 XML데이타소스로부터 정보를 가져오는 방법에 대해 논했습니다. 그에 더해서 JDOM클래스들에 대해 언급하고 각 클래스들이 XML문서를 어떻게 표현해내는지 말씀드렸죠.

이번 2부에서는 XML문서를 수정하거나 여러 정보를 이용해서 XML데이터를 만들어내는 법을 설명하겠습니다. 본 글을 시작하기 전에 말씀드릴 주의사항은, DJOM API는 현재 베타버젼이고 1.0버젼이 나올 때까지는 조금씩의 변화가 있을 수 있다는 것입니다. 공개 버젼은 몇달 안에 나오겠지만 확실한 시기는 JDOM커뮤니티가 얼마나 디자인과 구현을 열심히 하느냐에 달려있겠죠. JDOM 홈페이지를 지켜봐주세요.

문서 만들기

1부에서는 외부문서를 이용해서 JDOM Document를 만드는 방법을 배웠습니다.

```
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(url);
```

물론 외부문서가 아닌 새로운 JDOM Document를 만드는 것이 더 쉽습니다.

```
Element root = new Element("myRootElement");
Document doc = new Document(root);
```

간단히 root로 Element를 생성해서 Document와 Element를 만들 수 있지요. 여기서 중요한 점은 보통 다음과 같이 문서에 대해서

```
root.setText("This is a root element");
```

위와 같은 코드는 주어진 문자열로 root Element의 텍스트 내용을 설정한다는 뜻입니다.
지금 Document를 XMLOutputter를 이용해서 출력해본다면 결과는 다음과 같을겁니다.

```
<?xml version="1.0"?>

<root>This is a root element</root>
```

JDOM의 주요 장점중 하나는 Document클래스를 이용해서 문서를 만들거나 외부 문서를 가져오는 그런 모든 절차가 팩토리를 이용한다든가 하는 복잡한 절차를 거쳐야 할 필요가 없다는 것입니다.
실제로 DOM에서 위와 같은 작업을 하는 코드와 비교를 해볼까요?

```
// DOM code:
// 파서 제작자만의 고유한 방식의 Document Creating
Document doc = new org.apache.xerces.dom.DocumentImpl();

// text노드인 root 노드를 만들고 팩토리로서 Document를 사용합니다.
Element root = myDocument.createElement("myRootElement");
Text text = myDocument.createText("This is a root element");

// document tree에 노드들을 집어넣습니다.
root.appendChild(text);
myDocument.appendChild(root);
```

위의 예제에서 Document를 생성하는 어떤 표준적인 방법이 없음을 눈치채셨을 겁니다. 즉, 특정 명령을 사용하거나 썬 마이크로시스템즈사의 새로운 JAXP API를 이용해야 함을 아셔야합니다. (JAXP가 모든 파서에서 동작하는 것은 아닙니다. 더구나 아직 DOM2와 SAX

2.0을 아직 지원하지 못합니다.)

다음 예제는 서로다른 Document들 간에 JDOM Element를 이동하는 방법을 보여줍니다. parent1과 parent2는 서로 다른 JDOM Element들의 Element입니다.

```
// 첫번째 document의 element를 더합니다
Element movable = new Element("movableRootElement");
parent1.addContent(movable);

// 또다른 document의 element에 더한 후 원래의 Element를 제거
parent1.removeContent(movable);
parent2.addContent(movable);
```

JDOM Element는 특정 Document에 의해 만들어진 것이 아니기 때문에 이러한 이동이 가능하게 된거죠. DOM에서는 이러한 작업을 가능케 하기 위해서 첫번째 DOM Document에서의 Element를 import한 후 두번째 Element에 더하는 작업을 수행해야 했죠.

```
// DOM code:
Element movable = doc1.createElement("movable");
parent1.appendNode(movable);

// 또다른 document의 element에 더합니다
parent2.appendNode(movable);
// error! Incorrect document!
```

여기에서 보는 바와 같이 JDOM의 가장 중요한 특징은 Document의 생성과 수정이 간단하는 것이죠.

Element 생성하기

모든 JDOM Element는 최소한 하나의 root Element를 포함해야합니다. 또한 각 Element는 XML Document로 표현된 데이타에 필요한 만큼의 Element들을 담을 수 있습니다. root Element는 아래 보는 바와 같이 그것이 생성되는 시점에 Document에 넘겨 질 수 있습니다.

```
Element root = new Element("myRootElement");
Document doc = new Document(root);
```

반면에 Document의 root Element를 바꾸기 위해서는 Document의 메소드인 setRootElement (Element element)를 호출합니다.

```
Element newRoot = new Element("myNewRootElement");
doc.setRootElement(newRoot);
```

org.jdom.Element의 생성자는 그 Element의 이름이 담긴 String이 필요하고 Element의 namespace에 대한 정보를 선택적으로 필요로 하지요. 여기서 Element의 이름은 XML1.0스펙에서 요구하는 것에 따릅니다(만약 이름이 스펙에 따르지 않는다면 IllegalNameException이 발생합니다). 따라서 매번 Element를 만들때 마다 예외처리를 해야하는 불편함을 해소하기 위해서 IllegalNameException은 런타임 예외인 IllegalArgumentException으로 확장됩니다. 그러므로 XML 스펙에 준수한 이름을 사용한다면 새로운 Element를 생성할 때 어떤 특별한 예외처리를 해주어야할 필요는 없습니다.

```
// 아래 코드는 런타임시에 IllegalNameException을 발생할겁니다.
Element badElement = new Element("*(foo)");
```

유효성에 대한 것은 org.jdom.Verifier클래스가 담당하는데, XML문법의 규칙을 담고 있지요. XML에디터같은 애플리케이션들은 이 클래스를 사용해서 사용자가 사용하는 이름들이 유효한지 체크할 수도 있습니다.
자식이 없는 Element들은 대부분 아래와 같이 텍스트형식의 내용을 담고 있는 경우가 많죠.

```
<elementName>Here is some textual content</elementName>
```

1부에서 getText()메소드를 이용해서 텍스트 내용을 가져오는 것을 배웠죠?

```
String textualContent = element.getText();
```

물론 같은 방법으로 setText(String text)와 같은 방법으로 내용을 설정할 수 있습니다.

```
element.setText("My textual content");
```

JDOM에서는 &,<,>,',"과 같은 특수 문자들에 대한 처리도 매우 쉽게 할 수 있습니다. 그냥 단순히 일반 스트링에 특수문자와 같이 섞어서 쓰더라도 XMLOutputter클래스는 자동적으로 그것들을 XML문법에 맞게 표현해내죠. 물론 모든 builder들은 자동적으로 그들의 전통적인 스트링 값으로 변환시키고요. 아래 예제는 JDOM에서 완벽하게 작동합니다.

```
element.setText("Save cocoon.properties in <TOMCAT_HOME>/conf");
```

실제로 위의 것들이 XMLOutputter를 통해 출력될 때 어떻게 변환되는지 볼까요?

```
&lt;elementName&gt;Save cocoon.properties in
&lt;TOMCAT_HOME&gt;/conf&lt;/elementName&gt;
```

애들을 낳자

Element들은 종종 자식 Element를 가지죠. Element에 자식을 더하기 위해서 addContent(Element element)메소드를 사용하면 됩니다.

```
// 부모Element를 하나 만들고 두 자식Element을 만듭니다
Element parent = new Element("parent");
Element child1 = new Element("firstChild").setText("I'm number one");
Element child2 = new Element("secondChild").setText("I'm number two");

// 자식들을 더합니다
parent.addContent(child1);
parent.addContent(child2);
```

결과는 다음과 같습니다.

```
<parent>
  <firstChild>I'm number one</firstChild>
  <secondChild>I'm number two</secondChild>
</parent>
```

JDOM에서는 BEA의 웹로직의 htmlKona패키지나 아파치 프로젝트의 Element Construction Set(ECS)와 유사하게 쉽게 Document를 만들 수 있는 방법을 지원합니다.

```
Document doc = new Document(
  new Element("family")
  .addContent(new Element("mom"))
  .addContent(new Element("dad")
       .addContent("kidOfDad")));
```

주의할 점은 위와 같은 것은 간단한 문서의 생성일 때만 쓰시라는 점이죠. 복잡한 구조의 JDOM Document의 경우 혼란과 에러를 유발합니다.
Element들은 팩토리없이 만들어지기 때문에 쉽게 Element클래스에서 서브클래스를 만들 수 있습니다. 예를 들면, 모든 XML Document가 공통적으로 꼬리말을 가지고 있다면 FooterElement와 같은 footer클래스를 만들 수가 있다는 말이죠.

```
root.addContent(new FooterElement());
```

위의 FooterElement가 아래와 같이 구성되었다면...

```
<footer>
  <copyright>
    Jstorm 2001
  </copyright>
</footer>
```

FooterElement 클래스는 다음과 같이 만듭니다.

```
public class FooterElement extends Element {
  public FooterElement() {
    super("footer");
    addContent("copyright").setText("Jstorm 2001");
  }
}
```

Element가 만들어지면 자동적으로 그것의 자식 Element들이 더해지죠. 이러한 개념을 이용해서 저작권에 관련된 범용 템플릿을 만들 수도 있겠죠?

```
  public FooterElement(int copyrightYear) {
    super("footer");
    addContent("copyright").setText("Jstorm " + copyrightYear);
  }
```

그러면 다음해에는 아래와 같이 사용하는거죠.

```
root.addContent(new FooterElement(2002));
```

집단 관리

1부에서는 자바의 List객체처럼 Element의 자식들을 가져오는 방법에 대해 간단히 살펴보았습니다.

```
List children = element.getChildren();
```

정말 자식 리스트를 가져오고 관리하는 너무나 쉬운 방법아닌가요? 자바의 Collection API를 이용해서 JDOM은 프래그래머에게 Document를 요리하게 하기 때문에 이해하고 배우는 데 많은 시간이 필요하지 않습니다. 다음 예제에서 보는 바와 같이 쉽게 List를 가져오고 제어합니다.

```
List children = element.getChildren();

// 세번째 자식을 제거합니다.( -_- )
children.remove(3);

// "진호"라는 이름의 자식을 모두 제거합니다 (T_T)
children.removeAll(element.getChildren("진호"));

// 새로운 자식을 더합니다.
children.add(new Element("준호"));

// 두번째 자리에다 새로운 Element를 더합니다
children.add(1, new Element("second"));
```

물론 List 없이 범용적인 작업은 가능합니다.

```
// "진호" 라는 이름의 자식을 지웁니다
element.removeChildren("진호");

// 새로운 Element를 추가합니다.
element.addContent(new Element("준호"));
```

Element의 속성 설정

Element속성을 제어하는 것은 자식관리보다 더 쉽습니다. 왜냐면 속성들은 항상 하나의 텍스트값이거든요. JDOM은 JDOM Element로부터 속성을 제어하는 간단한 메소드를 제공합니다.

```
table.addAttribute("vspace", "0");
table.removeAttribute("border");
```

물론 Attribute클래스 객체를 만들어서 Element객체의 addAttribute()메소드에 더할 수 있습니다.

```
element.addAttribute(new Attribute("align", "right"));
```

자식 Element를 제어하는 것과 유사하게 Element의 속성 역시 getAttributes()메소드를 이용해서 List로 가져올 수 있습니다.

```
// 모든 attribute들을 가져옵니다.
List attributes = element.getAttributes();

// 모든 attribute들을 지웁니다.
table.getAttributes().clear();
```

또한 Element와 유사하게 Attribute 생성자 역시 잘못된 이름을 지정했을 경우 IllegalNameException이 발생하죠.

```
// IllegalNameException 가 런타임시에 발생할겁니다.
Attribute illegalAttribute = new Attribute("@lutris.com");
```

Namespace가지고 놀기

table이라는 Element 이름이 있다고 할 때, 이게 먹는 테이블의 table인지 html문서의 table인지 구별할 수 있게 해주는 것이 바로 XML Namespace입니다. 따라서 다음과 같이 콜론으로 Element의 소속 namespace를 지정해주면 문제가 없겠죠.

```
<furniture:table />
```

위의 furniture같은 접두사는 URI로 매핑됩니다. 즉 실제 특별한 지시어죠. xmlns라는 속성으로 xml Document의 root Element안에 위치하게 됩니다.

```
<?xml version="1.0"?>

<root xmlns:furniture="http://www.havertys.com"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <furniture:table>
    <furniture:numChairs>4</furniture:numChairs>
    <furniture:cushion available="yes" xlink:href="available_yes.jpg"/>
  </furniture:table>
</root>
```

JDOM에서는 org.jdom.Namespace클래스에서 XML Namespace를 표현하고 있습니다. 다음과 같이 namespace URI를 가져옵니다.

```
Namespace furniture =
  Namespace.getNamespace("furniture", "http://www.havertys.com");
Namespace xlink =
  Namespace.getNamespace("xlink", "http://www.w3.org/1999/xlink");
```

Namespace가 동일한지 체크하는 방법을 쉽게 사용하기 위해서 같은 Namespace 객체를 반환하는 동일한 접두사/URI를 위해 Namespace.getNamespace()메소드가 사용되죠. Namespace 객체는 Element나 Attribute의 생성자에서 전달합니다.(선택사항입니다)

```
Element table = new Element("table", furniture);
Attribute href = new Attribute("href", "available_yes.jpg", xlink);
table.addAttribute(href);
```

JDOM의 출력 클래스들은 자동적으로 필요로 하는 Namespace 선언을 첨가하고 Element에 접두사를 더하죠.
Namespace를 제어하기 위해서 다음과 같이 removeChildren()과 removeAttribute()메소드가 제공됩니다.

```
element.removeChildren("img", xhtml);
element.removeAttribute("width", xlink);
```

각각의 Element와 Attribute들은 자신만의 Namespace를 가지고 있기 때문에 쉽게 Namespace에 대한 걱정없이 서로 다른 Document간에 Element와 attribute를 이동할 수가

있습니다.

처리지시(Processing Instruction) 제공

XML 처리지시(Processing Instruction:PI)는 XML Document에 존재하는 명령같은 역할을 하죠. 주로 XML Document를 처리하는 애플리케이션에서 어떤 작업을 위해 사용됩니다.

```
<?cocoon-process type="xslt"?>
<?format-names %s//%g\n?>
```

처리지시는 target과 data 두개의 정보를 담고있습니다. target키워드는 위의 예제에서 보는 바와 같이 cocoon-process와 format-names를 지칭하며, data는 나머지부분 즉 type="xslt"와 is %s//%g\n를 뜻합니다. JDOM에서는 org.jdom.ProcessingInstruction클래스가 처리지시를 담당합니다.

```
ProcessingInstruction pi =
  new ProcessingInstruction("cocoon-process", "type=\"xslt\"");
```

위의 첫번째 예제에서 처럼 name과 value로 data가 이루어진 경우 JDOM에서 상당히 편리하게 작업할 수 있는데 name과 value쌍을 Map 객체로 담을 수 있기 때문이죠.

```
Map pairs = new HashMap();
pairs.put("type", "xslt");
ProcessingInstruction pi =
  new ProcessingInstruction("cocoon-process", pairs);
```

이러한 일련의 기능은 특별히 외부소스로부터 값들이 올 경우 여러 개의 name/value 쌍들을 처리하기 용이하죠. 아래 예제를 볼까요?

```
<?message-format type="XML-RPC" format="XML" encoding="UTF8" length="256"?>
```

다음과 같은 코드로 처리지시를 만들죠.

```
Map pairs = new HashMap();
pairs.put("type", "XML-RPC");
pairs.put("format", "XML");
pairs.put("encoding", encoding);
pairs.put("length", length);
ProcessingInstruction message =
  new ProcessingInstruction("message-format", pairs);
```

처리지시는 주로 XML Document의 level을 더하는데 사용됩니다만 그것을 Document 어디에 위치시켜도 상관없습니다.

```
ProcessingInstruction pi =
  new ProcessingInstruction("cocoon-process", "type=\"xslt\"");

Element root = new Element("root");
Document doc = new Document(root);
// document에 더합니다
doc.addProcessingInstruction(pi);
// element아래에 더합니다
root.addContent(new ProcessingInstruction("myPI", "myData");)
```

자식 Element를 더하는 메소드 이름과 동일하게 addContent()메소드를 사용해서 Element에 처리지시를 더합니다. 또한 같은 패턴으로 처리지시를 지울 수 있습니다.

```
// PI 지우기
doc.removeProcessingInstruction(pi);

// PI를 주어진 이름으로 지우기
doc.removeProcessingInstruction("cocoon-process")
```

더붙여서, 여러 개의 처리지시를 지울 수도 있습니다. 매우 유용하겠죠.

```
// cocoon-process라는 이름을 가진 모든 처리지시를 지웁니다.
doc.removeProcessingInstructions("cocoon-process")
```

다른 JDOM 생성자와 같이 이 처리지시역시 잘못된 이름을 사용할 경우 IllegalNameException을 런타임시에 발생합니다.

```
// runtime exception이 일어나겠죠?
ProcessingInstruction badPI =
  new ProcessingInstruction("$바보", "잘못된 이름!");
```

DocTypes

XML의 DOCTYPE은 DTD를 참조해서 문서의 유효성검사를 하는 데 사용됩니다. DOCTYPE은 여러 부분으로 나뉘어지는데, root Element, ID, 참조할 DTD의 특정 이름(W3C에 의해 정의된 표준 이름), 그리고 시스템 ID입니다. 대부분 DOCTYPE의 선언은 다음과 같습니다.(XHTML DTD를 사용하는 DTD입니다)

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

위의 예제에서 html은 root Element이고 public ID는 "W3C//DTD XHTML 1.0 Transitional//EN", 그리고 시스템 ID는 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"입니다. 이 시스템ID는 로컬 파일시스템의 DTD를 참조하여도 무방합니다.

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "/usr/local/xml/DTDs/xhtml1-transitional.dtd">
```

JDOM에서 DOCTYPE은 org.jdom.DocType에서 사용합니다. 선언을 위한 다양한 옵션사항이 있기 때문에 생성자 역시 다양합니다. 먼저 볼 것은 세부분으로 나눠어진 정보들을 이용한 생성입니다.

```
DocType xhtml = new DocType("html",
                    "-//W3C//DTD XHTML 1.0 Transitional//EN",
                    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd");
```

아래의 경우 public ID가 제공되지 않죠.

```
xhtml = new DocType("html",
                    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd");
```

이렇게 만들었으면 JDOM의 Document에 setDocType(DocType doctype)메소드를 이용해서 추가합니다.

```
doc.setDocType(xhtml);
```

JDOM은 아직 메모리에서 Document의 유효성검사를 지원하지 않습니다. 하지만 곧 추가될 겁니다.

주석

대부분의 프로그래밍 언어와 마찬가지로 XML역시 문서를 만드는 이가 자신만의 주석을 문서에 첨가할 수 있도록 배려하고 있습니다. XML 주석은 HTML과 같이 <!--와 -->로 구성됩니다.

```
<?xml version="1.0"?>

<!-- Comments can be at the document level -->

<root>
  <!-- Comments can be within elements as well -->
  <placeholder />
</root>
```

JDOM에서 주석은 org.jdom.Comment클래스가 담당합니다. 다음과 같이 간단하게 주석을 만들 수 있습니다.

```
Comment comment = new Comment("I would have lots of content here");
```

주석안에는 "--"문자열이 들어갈 수 없으므로 주석이 만들어지면 Verifier클래스가 체크해서 IllegalDataException을 발생시킵니다.

```
// runtime exception 이 발생할 겁니다
Comment badComment = new Comment("This --> will never work!");
```

Element클래스는 주석을 제어하는 메소드를 가지고 있습니다.

```
// 주석을 더합니다
root.addContent(new Comment("Anything but double dashes"));
```

Document역시 다음과 같은 메소드를 가지고 있죠

```
doc.addComment(new Comment("This is a document level comment"));
```

짬뽕 Content들

1부에서 잠깐 혼합된 Element의 content에 대해 언급했었죠. Element가 담고 있는 정보중에 text이외의 정보들에 대해서 getMixedContent()메소드가 제공됩니다. 이것은 type에 전혀 상관없이 Element로부터 모든 내용을 가져오는 메소드입니다. 여 경우 혼합되어있는 내용들의 리스트를 설정하는 작업이 필요하게 됩니다.
예를들어 어떤 Element의 모든 자식들을 다른 Element로 이동해야한다고 가정해보죠.

```
// 혼합된 content를 가져옵니다.
List mixed = oldElement.getMixedContent();

// 예전 element를 지웁니다.
for (Iterator i = mixed.iterator(); i.hasNext(); ) {
  oldElement.removeContent(i.next());
}

// 새로운 Element에 그것을 더합니다. iterator를 쓰는 것보다 훨씬 간편하죠.
newElement.setMixedContent(mixed);
```

피보나치: 새것과 낡은 것을 섞기

이제 좀 복잡한 문서를 만들어볼까 합니다. 하지만 아래 예제는 그렇게 복잡하지는 않습니다. XML을 이용해서 만든 전통적인 피보나치 생성기입니다. 이것은 아래와 같은 xml데이타를 출력해냅니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<Fibonacci_Numbers>
  <fibonacci index="0">0</fibonacci>
  <fibonacci index="1">1</fibonacci>
  <fibonacci index="2">1</fibonacci>
  <fibonacci index="3">2</fibonacci>
  <fibonacci index="4">3</fibonacci>
  <fibonacci index="5">5</fibonacci>
  <fibonacci index="6">8</fibonacci>
  <fibonacci index="7">13</fibonacci>
  <fibonacci index="8">21</fibonacci>
  <fibonacci index="9">34</fibonacci>
  <fibonacci index="10">55</fibonacci>
  <fibonacci index="11">89</fibonacci>
  <fibonacci index="12">144</fibonacci>
  <fibonacci index="13">233</fibonacci>
  <fibonacci index="14">377</fibonacci>
  <fibonacci index="15">610</fibonacci>
  <fibonacci index="16">987</fibonacci>
  <fibonacci index="17">1597</fibonacci>
  <fibonacci index="18">2584</fibonacci>
  <fibonacci index="19">4181</fibonacci>
  <fibonacci index="20">6765</fibonacci>
  <fibonacci index="21">10946</fibonacci>
  <fibonacci index="22">17711</fibonacci>
  <fibonacci index="23">28657</fibonacci>
  <fibonacci index="24">46368</fibonacci>
  <fibonacci index="25">75025</fibonacci>
</Fibonacci_Numbers>
```

참고로 이 예제는 XML Bible(IDG BOOK)의 저자인 Elliotte Rusty Harold씨의 XML DevCom에서 가져왔습니다. root Element와 Fibonacci_Numbers라 이름지어진 Element로 이루어져있지요.

```
import org.jdom.Element;
import org.jdom.Document;
import org.jdom.output.XMLOutputter;
import java.math.BigInteger;
import java.io.*;

public class FibonacciJDOM {

  public static void main(String[] args) {

    Element root = new Element("Fibonacci_Numbers");
    BigInteger low = BigInteger.ZERO;
    BigInteger high = BigInteger.ONE;

    for (int i = 0; i <= 25; i++) {
      Element fibonacci = new Element("fibonacci");
      fibonacci.addAttribute("index", String.valueOf(i));
      fibonacci.setText(low.toString());
      BigInteger temp = high;
      high = high.add(low);
      low = temp;
      root.addContent(fibonacci);
    }

    Document doc = new Document(root);
    // serialize it into a file
    try {
      FileOutputStream out = new FileOutputStream("fibonacci.xml");
      XMLOutputter serializer = new XMLOutputter();
      serializer.output(doc, out);
      out.flush();
      out.close();
    }
    catch (IOException e) {
      System.err.println(e);
    }
  }
}
```

역자 후기

XML과 가장 잘 어울리는 언어는 자바라는 데 누구도 이견을 달 사람은 없을 것입니다. 하지만 자바로 XML데이타를 처리하기에는 역시 필요이상의 중복작업들이 많았지요. 그런 중에 JDOM을 우연히 알게되었고 '아~ 내가 찾던 바로 그것이구나'라는 생각이 들더군요. 개인적인 프로젝트를 하려고 공부하다가 좀 더 많은 사람들이 JDOM을 알았으면 하는 생각에 번역을 했습니다. 군데군데 미숙한 번역상의 표현이 있었다면.. 죄송합니다. ^ ^

jstorm.pe.kr의 최진호(csecau@orgio.net)

About the author

Jason Hunter is a senior technologist with Collab.net, a company that provides tools and services for open source collaboration. In addition to being the cocreator of JDOM, he is the author of Java Servlet Programming (O'Reilly) and the publisher of http://Servlets.com. He has worked on projects from the largest (setting up an intranet application for a Fortune 100 company) to the smallest (helping develop a commercial product for a small startup). He contributes to Apache's Jakarta project and belongs to the working group responsible for Servlet API development.

Brett McLaughlin works as an Enterprise Java consultant at Metro Information Services and specializes in distributed systems architecture. In addition to cocreating JDOM, he has written Java and XML (O'Reilly) and Enterprise Applications in Java (O'Reilly). Brett is involved in technologies such as Java servlets, Enterprise JavaBeans, XML, and business-to-business applications. He is an active developer on the Apache Cocoon project and EJBoss EJB server, and he is a cofounder of the Apache Turbine project.

http://www.javaworld.com/javaworld/jw-07-2000/jw-0728-jdom2_p.html

End of Document