**XML의 장점**

작성하기가 간편하다(tag구조)  
XML 사람이 읽기가 쉽다. (즉 각 정보들이 의미하는 바를 한눈에 보기가 좋다.  
DTD 등 XML자체의 기능을 확장할 여지가 많이 있다.

**XML의 단점**

문서의 양이 필요이상으로 많다.(실 데이터가 아닌 tag글자)  
배열형식이나 반복 구조의 경우 불필요한 데이터가 계속 해서 나타난다.  
결국 이로 인해 파싱이 힘들어지고 속도는 느려진다.

**JSON의 장점**

내용이 함축적으로 최소한의 정보만을 가지고있다.  
그렇기때문에 XML대비 용량이 획기적으로 줄어들고 속도는 그만큼 빨라지게 된다.  
객체 구조와 {} 배열 구조의 [] 적절한 만남으로 아주 효율적인 데이터 구성이 가능하다.   
파싱이 매우 간편하고, 때로는 일반적인 변수처럼 사용도 가능하다.  
즉, 사용하기 쉽다.

**JSON의 단점**

내용이 함축적이다 보니 내용의 의미 파악은 힘들 수 있다.   
아무래도 적은 규격의 데이터 전송에 적합한 방식이기때문에 XML보다는 빠르지만  
대용량급의 데이터 송수신엔 부적합 모습도 있다.

XML파싱 + DOM과 SAX의 차이

XML이란?

 트리구조의 데이터를 단순히 텍스트 형태로 나타낸 것이다.

<?xml version="1.0"?>  
<고양이>  
 <이름>나비</이름>  
 <품종>샴</품종>  
 <나이>6</나이>  
 <중성화>예</중성화>  
 <발톱 제거>아니요</발톱 제거>  
 <등록 번호>Izz138bod</등록 번호>  
 <소유자>이강주</소유자>  
</고양이>

이런식으로 <태그> </태그> 구조로 자유롭게 만들 수 있다.

XML은 범용성과 확장성을 가지기 때문에 환경설정이나 데이터 전달포멧으로 쓰이는 것을 많이 볼 수 있다.

장점 : 유연하게 쓰임 (이기종간 데이터전달 등) , 네임스페이스+문서검증기능?

단점 : 가독성이 쓰레기임, 파싱하기 어려움, 처리속도도 느림...

=> 웬만하면 json을 쓰자! ㅋㅋ

**XML파싱**

XML 파싱 방법으로는 **DOM** (Document Object Model) 방식과 **SAX** (Simple API for XML) 방식이 있다.

차이점은 DOM방식은 메모리에 모두 로드 후 파싱하고 SAX는 순차적으로 읽어가며 파싱하는 것이다.

경우에 따라서 적절한 방법을 선택하여 사용하면 된다.

**- DOM방식**

1. 처음 XML문서를 메모리에 모두 로드한 후 값을 읽는다.

2. XML문서가 메모리에 모두 로드되어있기 때문에 노드의 검색, 수정, 구조변경등이 빠르고 용이하다.

3. 직관적이고 SAX보다 파싱하기 단순하다.

**\*접근방법**

File file = new File("test.xml");

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

DocumentBuilder db = dbf.newDocumentBuilder();

Document doc = db.parse(file);

**\*DOM 파서 구현**

doc.getDocumentElement().normalize();

String rootName = doc.getDocumentElement().getNodeName(); //루트 엘리먼트의 이름을 리턴 합니다.

NodeList nodeLst = doc.getElementsByTagName("region\_id"); //문서에서 region\_id 노드를 전부 찾아 배열로 돌려줍니다.

Node fstNode = nodeLst.item(0); //nodeList의 첫번째 노드를 추출 합니다.

Element fstElmnt = (Element) fstNode;

NodeList fstNmElmntLst = fstElmnt.getElementsByTagName("announce\_date");

//fstElmnt의 하위에 announce\_date태그 네임을 검색하여 배열로 돌려 줍니다

Element fstNmElmnt = (Element) fstNmElmntLst.item(0); ////fstNmElmntList의 첫번째 노드를 추출 합니다.

NodeList fstNm = fstNmElmnt.getChildNodes(); //fstNmElmnt의 하위 노드들 추출

System.out.println("URL : "+ ((Node) fstNm.item(0)).getNodeValue()); //첫번째 노드의 값 출력

위와 같이 getElementsByTagName() 과 getNodeValue()같은 함수를 사용하여 노드를 검색하고 값을 뽑아냅니다.

**- SAX방식**

1. XML문서를 순차적으로 읽어들이면서 노드가 열리고 닫히는 과정에서 이벤트가 발생한다.

2. XML문서를 메모리에 전부 로딩하고 파싱하는것이 아니라서 메모리 사용량이 적고 단순히 읽기만 할 때 속도가 빠름.

3. 발생한 이벤트를 핸들링하여 변수에 저장,활용하는 것이기 때문에 복잡하고, 노드 수정이 어렵다.

4. DOM보다 어렵다.

**\*접근방법**

SAXParser parser = SAXParserFactory.newInstance().newSAXParser();

File xmlFile = newFile(“test.xml”);

parser.parse(xmlFile, new SAXSampleParser());

**\*파싱방법**

클래스 선언에서 DefaultHandler를 상속받아 아래 함수들을 오버라이드 하여 사용해야 한다.

  // 문서의 시작시 발생하는 이벤트 입니다.

  public void startDocument() throws SAXException {

    super.startDocument();

  }

  // 파서가 xml을 읽는 도중 엘리먼트 시작테그를 만날 때마다 발생하는 이벤트

  public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {

 //qName 엘리먼트의 이름

// 엘리먼트 속성

    for (int i = 0; i < attributes.getLength(); i++) {

      //System.out.println(“Attribute: ” + attributes.getQName(i) + “=” + attributes.getValue(i));

    }

  }

  // 엘리먼트 끝

  public void endElement(String uri, String localName, String qName) throws SAXException {

    //System.out.println(“End Element: ” + qName);

  }

  // 엘리먼트 이벤트 중간중간 텍스트를 만났을때 발생

  public void characters(char ch[], int start, int length) throws SAXException {

    //System.out.println(“Character: ” + new String(ch, start, length));

  }

  // 문서의 끝

  public void endDocument() throws SAXException {

    super.endDocument();

    System.out.println(“End Document”);

  }

위와 같이 SAX방식은 이벤트 발생시마다 호출 되는 함수에 제어할 구문을 구현하여 사용하여야 한다.

**파싱예**

<?xml version="1.0"?>

<company>

<staff id="1001">

<firstname>yong</firstname>

<lastname>mook kim</lastname>

<nickname>mkyong</nickname>

<salary>100000</salary>

</staff>

<staff id="2001">

<firstname>low</firstname>

<lastname>yin fong</lastname>

<nickname>fong fong</nickname>

<salary>200000</salary>

</staff>

</company>

package com.mkyong.seo;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.DocumentBuilder;

import org.w3c.dom.Document;

import org.w3c.dom.NodeList;

import org.w3c.dom.Node;

import org.w3c.dom.Element;

import java.io.File;

public class ReadXMLFile {

public static void main(String argv[]) {

try {

File fXmlFile = new File("/Users/mkyong/staff.xml");

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

//optional, but recommended

//read this - http://stackoverflow.com/questions/13786607/normalization-in-dom-parsing-with-java-how-does-it-work

doc.getDocumentElement().normalize();

System.out.println("Root element :" + doc.getDocumentElement().getNodeName());

NodeList nList = doc.getElementsByTagName("staff");

System.out.println("----------------------------");

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = nList.item(temp);

System.out.println("\nCurrent Element :" + nNode.getNodeName());

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

System.out.println("Staff id : " + eElement.getAttribute("id"));

System.out.println("First Name : " + eElement.getElementsByTagName("firstname").item(0).getTextContent());

System.out.println("Last Name : " + eElement.getElementsByTagName("lastname").item(0).getTextContent());

System.out.println("Nick Name : " + eElement.getElementsByTagName("nickname").item(0).getTextContent());

System.out.println("Salary : " + eElement.getElementsByTagName("salary").item(0).getTextContent());

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

결과

Root element :company

----------------------------

Current Element :staff

Staff id : 1001

First Name : yong

Last Name : mook kim

Nick Name : mkyong

Salary : 100000

Current Element :staff

Staff id : 2001

First Name : low

Last Name : yin fong

Nick Name : fong fong

Salary : 200000