XML Pull Parser를 위한

JFlex 와 BYacc/J의 적용방안

장주현° 노희영
강 원 대 학 교 컴 퓨 터 과 학 과
hc98038@kwnu.kangwon.ac.kr, young@cc.kangwon.ac.kr

Adaptation method of JFlex and BYacc/J for XML Pull Parser

Ju-Hyun Jang° Hi-Young Roh
Dept. of Computer Science, Kangwon National University

요 약

Xml 파서들의 벤치 마킹 결과에서 기존 파서와는 달리 JFlex와 BYacc/J를 사용한 Piccolo는 다른 파서들에 비해 0.5 ~ 1.5배 향상된 빠른 속도의 파싱 속도를 나타내었다. 하지만 XML Pull 모델의 표준 Interface인 XPP(XML Pull Parser)가 제정되고 안정화 됨에 따라서 MXP(XPP Version 3.0)[1]는 기존 XML 파서들과 달리 Parser Generator tool을 사용하여 가장 빠른 파싱 속도를 보였던 Piccolo 파서[2] 보다도 빠른 파싱 속도를 나타내었고 이는 현 XML파서 중 가장 빠른 파싱 속도로 Pull 모델이 기존의 Object, Push 보다 빠른 모델이라는 점을 입증하였다. 본 논문에서는 Pull Parser를 구현하는데 있어 Piccolo에서 사용한 JFlex와 BYacc/J를 사용함으로 Pull 파서의 속도를 한층 더 높여 Piccolo, MXP 보다 빠른 파싱 속도를 내는 파서를 구현 하기 위한 JFlex와 BYacc/J의 적용방안에 대해 연구 하였다.

1. 서 론

XML은 SGML과 같이 데이터의 구조와 속성을 정의할 수 있고 HTML의 특징처럼 웹 상에서 문서표현을 쉽게 할 수 있는 장점을 가진 Mark-up언어이다. 또한 문서의 표현 위주가 아닌 자료 위주의 Mark-up언어이기 때문에 자료의 효율적 표현을 위해 많이 사용되어 지고 있으며 통신망의 발달과 Internet의 보급으로 인하여 PDA, Celluar-phone등의 장치에서 정보 교환의 수단으로 널리 사용되고 있다.

현재 XML(eXtensible Markup Language)문서의 표현을 위한 파싱 Model은 Object, Push, Pull의 세 모델이 사용되고 있다. Object 모델은 문서의 모든 데이터를 메모리 내에 구성하는 모델로써 문서의 구조에 강하다는 강점이 있지만 모든 데이터를 메모리 구조에 사용하기 때문에 많은 메모리 사용과 느린 속도를 가진다는 단점이 있다. 이에 반해 Push Model은 XML문서상에 모든 데이터를 Event로 처리하는 모델로써 Object 모델의 단점을 극복하기 위하여 나온 파싱 모델이다. 마지막으로 가장 최근에 나온 파싱 모델은 Pull Model이며 위의 두 모델보다는 문서의 구조에는 취약하지만 빠른 파싱을 위해 제안된 모델이다.[3] 하지만 최근까지 Object 모델의 DOM(Document Object Model)이나 Push 모델의 SAX(Simple Api for XML)와 같은 표준 Interface가 제공되지 않았다.

Pull 모델이 표준화 되기 전까지 나온 파서 중 가장 빠른 파서는 Piccolo XML for Java Parser이며 Push Model 파서로 문서 내의 데이터의 Token 추출과 상태 전이를 빨리 하여 속도 향상을 꾀한 파서이고 이를 위하여 Parser Generator tool인 JFlex 와 BYacc/J를 사용하였다. 기존 파서가 전부 Hand-write 파서인 것에 비하면 Piccolo는 다른 구현 방식을 가지고 있고 이로 인해 기존 파서들보다도 0.5~1.5배 정도 빠른 파싱 속도를 보이고 있다. 이와 동일한 시기에 Pull Model이 표준화 되었다. Pull Model은 응용프로그램에서 파싱의 제어를 하기에 빠른 파싱속도와 적은 메모리 사용, 간단한 Interface로 인해 기존 파싱 모델보다 속도와 메모리 사용면에서 뛰어나다.[7] Pull 모델의 표준 인터페이스인 XMLPull 인터페이스는 기존 Pull Parser인 Kxml 파서와 XPP 파서의 개발자들이 Pull 모델 파서를 보급하기 위하여 표준 Interface를 제정하였고 Pull 모델 파서의 급속한 보급에 영향을 끼치고 있다. 또한 XMLPull의 안정화로 인하여 MXP(XPP Version 3)는 기존 벤치마킹 결과와는 달리 벤치마킹 결과의 거의 모든 분야에서 Piccolo 파서보다 5~20% 빠른 파싱 속도를 보이고 있다. 본 논문에서는 기존 파싱 모델중에 가장 빠른 모델인 Pull Model의 표준 Interface인 XMLPull을 구현함에 있어 Piccolo 파서에서 사용한 JFlex 와 BYacc/J를 어휘분석과 구문분석에 적용하는 빠른 파싱 속도를 내는 파서의 구현방안에 대해 연구하였다.

2. 관련 연구

2.1 XMLPull API

XMLPull API의 구성을 살펴 보면 XmlPullParser의 Instance를 제공하는 XmlPullParserFactory Class가 있다. 파서를 사용하기 위한 Instance는 이 Class를 객체화 하여 사용한다. XmlPullParser Interface는 Pull 파서와 관련된 추상 메소드, 이벤트에 관련된 상수와 그 밖의 Pull Parser를 위한 변수가 정의 된 Interface이다. XmlSerializer Interface는 XML 문서의 Serializer에 관련된 Interface이지만 아직 안정화 되어 있지 않은 Interface로 현재 사용되고 있지는 않다.

XmlPullParserException Class는 입출력과 같은 일반적인 오류

를 제외한 파싱과 관련된 예외 처리에 관한 Class 이다.

본 논문에서 사용한 XmlPullParser Interface에는 XML spec 1.0 에 정의된 XML 문서에 관한 feature, properties의 처리에 대해 명시가 되어 있으며 데이터의 Event 처리를 위하여 Event Type에 관련된 상수들이 정의되어 있다. 그리고 문서의 파싱시에 응용프로그램에게 Event Type에 관한 정보를 넘겨주기 위한 메소드 들이 존재 한다. 이러한 메소드중에 두 개의 key 메소드가 존재를 하며 첫 번째 next() 메소드는 상위 레밸의 파싱을 위하여, nextToken() 메소드는 하위 레밸의 파싱을 위하여 존재 한다. 그 밖에 DTD, version, 속성 등에 관련된 추상 메소드 들이 정의되어 있다.

2.2 Piccolo

Piccolo 파서는 기존보다 빠른 Push 모델 파서를 개발하고자 하여 나온 파서로 Interface SAX1.0, SAX 2.0.1, JAXP(Java Api for XML Processing)1.1 을 구현한 파서이다. 파서의 구현시에 hand-write 방식의 파서보다 Parser Generator tool을 사용한 파서가 문서의 Token 추출에 빠르다는 것을 이용하여 JFlex 와 BYacc/J를 사용하여 파서를 구현하였다. 벤치마킹 결과에서 33K의 적은 양의 문서부터 36MB의 비교적 큰 Data의 문서를 파싱했을 때의 결과에서 Piccolo 파서가 기존 파서들에 비하여 확연히 빠른 속도를 나타내고 있는 것을 알 수가 있다.[2]

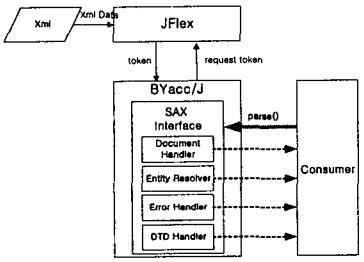

그림 1. Piccolo에서 사용한 JFlex와 BYacc/J 구조

3. 파서의 설계

파서의 기본적인 동작은 문서를 순차적으로 읽으면서 문서에 대한 정보를 인식하는 것이다. XML 파서의 기본동작은 XML 문서의 시작을 인식하고 그 다음 version, encoding등의 선언에 대한 정보 추출이 필요하다. 그 다음에는 Tag 분석과정과 Tag Value 분석과정을 반복하면서 문서에 대한 정보를 인식하게 된다.[7]

3.1 JFlex 와 BYacc/J를 이용한 Pull 파서 구조

본 논문은 JFlex 와 BYacc/J를 이용하여 Pull 파서를 구현하기 위하여 그림2와 같이 XML Pull 파서를 설계하였다.

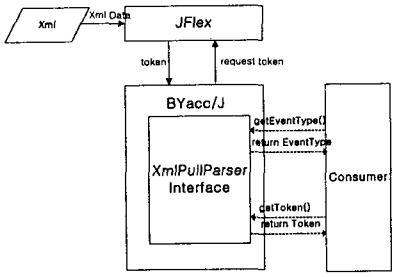

그림 2. Pull 파서 구현을 위해 설계한 JFlex와 BYacc/J 구조

파서는 처리하는 역할에 따라 세 부분으로 구성되어 있다. 문서의 Token 추출를 위한 어휘분석 부분(JFlex)과 Token들에 대한 의미를 부여 하는 구문 분석 부분(BYacc/J의 Code부분 제외), 응용프로그램과의 Parser Interface 부분(XmlPullParser Interface 구현)으로 나뉜다. 그림 1의 Piccolo의 구조에서 BYacc/J는 EventHandler, Document Handler등의 Event 처리 Hander를 구현하고 이를 이용하여 응용프로그램에게 결과를 반환해 준다. 하지만 Pull 파서에서는 파서의 역할이 XML 데이터의 EventType과 관련 정보를 인식하고 응용프로그램에서 EventType, 관련정보를 요청하면 이를 반환해 주는 역할을 한다. XML Pull 파서의 구현을 위한 JFlex와 BYacc/J에서는 XML 데이터의 Token 추출, Token의 EventType, 기타 XML문서와 관련된 정보를 인식하고 그 정보를 응용프로그램에게 전달해 주는 구조를 이루고 있다.

4. JFlex와 BYacc/J의 적용방안

XML Pull 파서에서 사용되는 문자 집합, properties, features는 2000년 10월에 제정된 XML spec 1.0(Second Edition)[4]의 정의에 따랐으며 어휘분석과 구문분석시에 결정할 모든 Token, 문법 정의 역시 XML spec 1.0을 기준으로 작성되었다. 파서의 구현 내용을 살펴보면 파서는 직접 구현하는 두 개의 파일과 자동 생성되는 하나의 파일로 구성 되어 있다. 하나는 어휘분석을 위한 파일로 JXPullParserLexer.flex에서 생성된 파일 *JXPullParserLexer.java*이고 다른 하나는 파일 JXPullParser.y에서 생성된 파일 JXPullParser.java이다.[5][6] JXPullParserVal.java 파일은 자동으로 생성되고 이는 Token의 data type을 위해 사용된다. 파서를 위해 사용할 패키지로는 com.JXP.xml로 정하였으며 XmlPullParser Interface와XmlPullParserException class를 사용하기 위하여 org.xmlpull.v1 패키지를 import 하였고 Pull Model의 표준 XPP의 버전은 XPP1.1.1.4를 이용하여 구현하였다.

4.1 JXPullParserLexer.flex

어휘분석단계인 JFlex 응용프로그램인 JXPullParserLexer.flex에서는 com.JXP.xml 패키지를 선언, class의 선언, unicode 지원을 위해 unicode 사용을 선언한다. 또한 Xml Token들을 정의하고 해당 Token을 추출한 후의 BYacc/J에게 해당 Token을 넘겨주는 역할을 위한 Lexical rules 부분을 작성하였다. 정의된 모든 Token들은 정규 식으로 표현되며 표1 과 같이 선언된다.

표 1. Token 정의

```
EncName = [A-Za-z][A-Za-z0-9._₩-]*
VersionNum = ([a-zA-Z0-9_.:₩] | '-')+
```

표1 은 Token을 정의하는 예를 나타낸 것으로 EncName은 인코딩이름에 관한 Token을 위해 정의하고 VersionNum은 버전에 관련된 정보를 파싱하기 위하여 정의한 것이다. 그리고 문서의 흐름에 맞는 모든 상태들을 정의하였다. 본 논문에서 사용된 주요 상태들을 보면 다음과 같다.

표 2. 주요 상태

초기 상태	XML_TAG
초기상태에서의 처리	YYINITIAL_DIRECT, ENTITYREF, PI, PI_WS
TAG 상태	TAG, TAG_START, TAG_NS, TAG_START_NS
TAG VALUE 상태	TAG_VALUE, TAG_VALUE_ENTITY,TAG_VALUE_ENTITYREF
닫음 TAG	CLOSE_TAG, CLOSE_TAG_NS

표2 에서 나온 상태들은 DTD가 없는 문서나 DTD와 관련되지 않은 문서들에 관한 상태들이다. XML 문서는 DTD와 밀접한 관계를 가지기 때문에 DTD에 관련된 상태 들은 따로 정의하였다. 표3은 DTD와 관련된 주요 상태들을 정의해 놓은 표이다.

표 3. DTD 처리에 관련된 주요 상태들

DTD의 시작	DTD
DTD에서 속성과 관련된 상태	DTD_ATTLIST_ELEMENT, DTD_ATT_NAME, DTD_ATT_TYPE
DTD TAG 와 관련된 상태	DTD_TAG, DTD_TAG_START DTD_TAG_VALUE_ENTITY DTD_TAG_VALUE_EXTERNAL_ENTITY
DTD Entity 정의	DTD_ENTITY_DECL

표 2,3 에서 나온 상태를 제외하고도 주석, Namespace 등과 관련된 몇 상태들이 있지만 파싱의 주요 상태는 아니므로 표에서는 제외 하였다. 또한 DTD에서 정의된 Tag, 속성, Entity의 이름 등을 hashtable에 저장하여 정의되지 않은 값들을 사용하는지에 대한 판단을 한다. 그리고 Tag의 시작, 마침, 속성의 개수등을 판단하기 위해서 stack을 사용해서 시작 Tag 선언시에 push, 마침 Tag시에 pop을 하여 Well-formed를 판단하고 처리한다.

얻어진 Token들과 각각의 상태들은 Token들의 Event Type과 기타 정보의 반환을 위한 구문분석과정인 BYacc/J에서 사용한다.

4.2 JXPullParser.y

구문분석 과정에서 사용한 파일 JXPullParser.y는 실질적 파서 부분이고 패키지 선언과 입출력을 위해 사용될 java.io 패키지와 Entity들을 위해 사용될 java.util 패키지를 import 하였고 XmlPullParser Interface를 위한 implement문을 선언 하였다. 패키지와 예외, 구현등에 대한 선언이 끝난 후 사용할 Token을 선언하고 얻어진 Token들을 XML 문법에 적용하여 Event Type 선언과 Encoding 정보 등을 할당하였다.

표 4. XML 문서의 정의

```
document: xml_decl dtd body epilog
        | xml_decl body epilog
```

표4는 XML 문서의 정의를 한 예이다. 문서는 XML 선언인 xml_decl, DTD를 위미하는 dtd, 문서의 몸체를 의미하는 body, 마지막으로 epilog를 순서대로 나타내었고 또는 dtd를 제외한 형태를 나타낼 수 있다고 정의하였다. 각각은 다시 세분화 되어 정의 된다. 표 5는 위의 body 부분을 세분화 하여 정의한 것이다.

표 5. 문서의 몸체 정의

```
body:   EMPTY_TAG
      | OPEN_TAG content CLOSE_TAG
      | misc body
```

표5에서는 표4 에서 정의한 body의 구성이 세가지 형태로 세분화 되어 나올 수 있는 것을 정의하였고 body의 구성요소인 content, misc는 계속 세분화 되게 정의하였다. 이렇게 정의된 값들을 Event Type에 맞게 값을 할당해 주었다. 표 6은 XML 선언의 처리에 대한 예를 보여 주고 있다.

표 6. Event 값의 할당

```
xml_decl: XML_TAG VERSION STANDALONE
 {        this.version = $2;
          this.standalone = $4;
 }
```

표6은 Token의 Event에 해당하는 값의 할당에 대해 나타낸 것이다. version, standalone, peatures, propertys등에는 값을 바로 할당하지만 Tag, 속성등에 관련된 값에는 Event Type값만을 할당하였고 응용프로그램에서 요청시에 이 값들을 반환해 줄수 있도록 Interface 부분을 작성하였다.

4.3 Parser Interface

본 논문에서 설계한 Pull Parser에서는 Interface의 역할은 어휘분석 과정과 구문분석을 통해서 얻어진 결과 값을 단순히 제공해 주는 역할만을 하면 된다. 이 Interface에서 가장 중요한 key method는 next() 메소드와 getEventType() 메소드이다. next()는 다음 Event로 가는 역할을 하는 메소드로 본 논문에서는 JFlex에게 다음 Token을 요청하고 BYacc/J에서 Token의 Event Type을 정의하고 다시 getEventType() 메소드나 그 밖의 다른 메소드 호출이 일어나면 이벤트 타입을 응용프로그램에게 전달해 주는 형태로 구현하였다. 그 다음 getEventType()의 메소드는 BYacc/J에서 Event Type을 위한 변수를 정하고 그 변수를 반환 해 줌으로써 구현을 하였다.

5. 결론 및 향후 연구과제

본 논문은 기존 XML 파서보다 빠른 파서를 구현하기 위하여 Pull 모델을 채택하였고 Piccolo에서 속도 향상을 위해 사용한 JFlex 와 BYacc/J를 Pull 모델에 어휘분석과 구문분석처리에 대 적용하는 방안에 대하여 논하였다. 이 방법을 가지고 파서를 구현한 후 다른 파서와의 속도비교를 통해서 JFlex 와 BYacc/J를 적용하는 것이 빠른 속도를 낸다는 것을 검증해야 한다. 본 논문에서 사용한 Pull 모델은 속도가 빠르다는 장점을 가지고 있지만 파서의 역할이 Event Type과 Event 값을 위주로 제공한다는 점에서 문서의 재구성, 수정에 있어서는 기존의 Object, Push 파서에 비하여 취약한 것을 알 수 있다. 그러므로 파싱 속도는 크게 저하시키지 않으면서 문서의 구조에 관해 접근을 할 수 있는 구현 방법을 향후에 연구해야 할 것이다.

참고문헌

[1] Aleksander Slominski, http://www.extreme.indiana.edu/xgws/xsoap/xpp/

[2] Yuval Oren, http://piccolo.sourceforge.net/

[3] Dennis M. Sosnoski, XML documents on the run, http://www.javaworld.com/javaworld/jw-04-2002/jw-0426-xmljava3_p.html

[4] http://www.w3.org/TR/REC-xml

[5] Gerwin Klein, http://www.jflex.de/

[6] Bob Jamison, http://troi.lincom-asg.com/~rjamison/byacc/index2.html

[7] 장주현,노희영 JFlex와 BYacc/J를 이용한 Xml Pull Parser 설계, 봄 학술발표논문집, 제 30권 제1호(B), 31 ~ 33