XML Manipulation Service

Summary

The XML manipulation service provides the functions to create, read, write and manipulate XML. **The XML(Extensible Markup Language)**is the multipurpose mark-up language recommended for creating markup languages with another special purposes at**W3C**. The XML is the simplified subsets of**SGML** but can be applied in describing many kinds of data. The XML is designed to overcome the limit of HTML by enabling to send and receive data easily between different systems, in particular, the systems connected to Internet. **The XML(Extensible Markup Language)**is the multipurpose mark-up language recommended for creating markup languages with another special purposes at**W3C**. The**XML** is the simplified subset of **SGML** but can be applied to describe many kinds of data.

메모 [bj1]: 앞부분과 중복. 원본체크하기.

Description

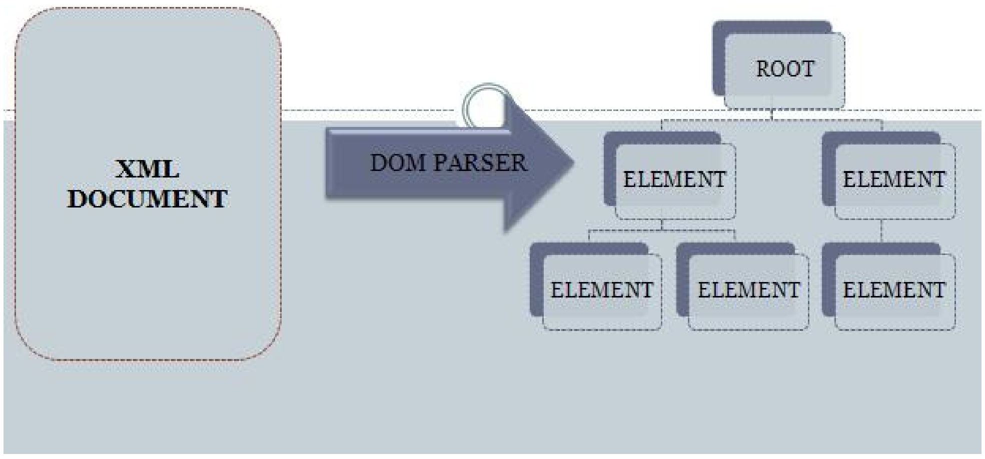
XML Parser

There are two types of parsers that play a role of reading the XML document:**DOM(Document Object Model)**creates and processes the objects by getting the contents of XML file once in the tree structure, and **SAX(Simple API for XML)** reads the XML document whenever each tag and content are recognized.

DOM(Document Object Model)

Summary

The XML document is the hierarchical information of the tree structure formed in **element**, **attribute and Text**. ⇒Read the object in the tree structure for each element of XML document using DOM. DOM is the standard interface for the respective objects representing XML documents. DOM parser plays a role of creating DOM structure from XML document.



Sample Source

1) Creating a document object for the XML documents

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder parser = facory.newDocumentBuilder();
Document dc = parser.parse("XML file name");
Element root = xmldoc.getDocumentElemnt();
```

```
logger.debug(root);
```

1. Create the DocumentBuilderFactoryinstance.
2. Create parses from created DocumentBuilderFactory.
3. Load the XML documents by calling parse() methods of DocumentBuilder.
4. Obtain the root elements of document.

2) Extracting child elements directly under it: extract the first child elements of getFirstChild(),getNextSibling() inherited by Node interface, and the sister elements of this element.

```
Element root = doc.getDocumentElement();
for(Node ch = root.getFirstChild(); ch != null; ch = ch.getNextSibling()
{
logger.debug(ch.getNodeName());
}
```

1. Obtain the root elements of documents.
2. Extract the first element among the child elements of getFirstChild(),getNextSibling() and the sister elements of this element.

3) Extracting the element name only: extract the element only except the text node indicating the blank, examine the type of extracted node and display the node name only in case of Node.ELEMENT_NODE.

```
Element root = doc.getDocumentElement();
for(Node ch = root.getFirstChild(); ch != null; ch = ch.getNextSibiling())
{
if(ch.getNodeType() == Node.ELEMENT_NODE)
logger.debug(ch.getNodeName());
}
```

1. Obtaining root element of document
2. Examination of the type of node and display the node name only in case of Node.ELEMENT_NODE

4) Extracting all child elements: call getNode() methods for each child node and extract the child nodes of child. Utilize the recursive function method.

```
Element root = xmldoc.getDocumentElement();
getNode(root)

public static void getNode(Node n)
{
for(Node ch = n.getFirstChild(); ch != null; ch = ch.getNextSibling())
   {
if(ch.getNodeType() == Node.ELEMENT_NODE)
       {
logger.debug(ch.getNodeName());
           getNode(ch);
       }
   }
}
```

1. Locate the child node of node.
2. Extract the child node of child by calling getNode() method.

5) Extracting text only except the blank: use the control sentence in the following conditions in order to extract the text nodes for the contents of elements excluding text node indicating blank.

```
public static void getNode(Node n)
{
```

```
for(Node ch = n.getFirstChild(); ch != null; ch = ch.getNextSibling())
    {
if(ch.getNodeType() == Node.ELEMENT_NODE)
        {
logger.debug(ch.getNodeName());
            getNode(ch);
        }
        // process the text.
else if(ch.getNodeType() == Node.TEXT_NODE&&ch.getNodeValue().trim().length() !=0)
        {
logger.debug(ch.getNodeValue());
        }
    }
}
```

1. Ordered List Item

6-1) Parsing per type of node: the method of processing per type of node and returning the contents of XML documents extracted to the character string.

```
privateStringxmlString = "";
publicStringprintString(Node node)
{
int type = node.getNodeType();
switch(type)
    {
caseNode.DOCUMENT_NODE:
printString(((Document)node).getDocumentElement());
break;
caseNode.ELEMENT_NODE:
xmlString += "<"+node.getNodeName();
            NamedNodeMapattrs = node.getAttributes();
            for(inti = 0; i<attrs.getLength(); i++)
            {
                Node attr = attrs.item(i);
            xmlString += " "+attr.getNodeName()+"="+attr.getNodeValue()+"'";
            xmlString += ">";
            NodeList children = node.getChildNodes();
            if(children != null)
                {
            for(inti=0; i<children.getLength(); i++)
                    {
            logger.debug(children.item(i));
                    }
                }
            break;
            }
caseNode.CDATA_SECTION_NODE:
xmlString += "<![CDATA["+node.getNodeValue()+"]]>";
            break;
caseNode.TEXT_NODE:
xmlString += node.getNodeValue().trim();
            break;
caseNode.PROCESSING_INSTRUCTION_NODE:
xmlString += "<?"+node.getNodeName()+" "+ node.getNodeValue()+"?>";
            break;
    }
if(type == Node.ELEMENT_NODE)
    {
xmlString += "</"+node.getNodeName()+">";
    }
returnxmlString;
```

}

6-2) Parsing per type of nodes: use the method of getNodeType() supported at the Node interface and process depending on which type of the recognized child node is.

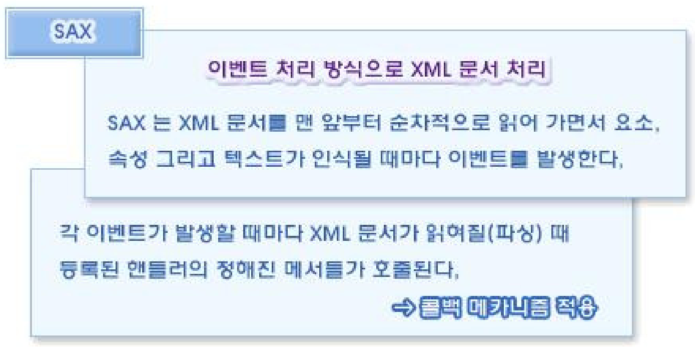
NODE TYPE	Description
Node.DOCUMENT_NODE	Call printString() with DocumentElementobject information
Node.ELEMENT_NODE	Extract element name, property information(NAME,VALUE) and save in xmlString, and call pringString() with child element information.
Node.CDATA_SECTION_NODE	Add <!CDATA[and]]> to the extracted value
Node.TEXT_NODE	Extract VALUE only
Node.PROCESSING_INSTRUCTION_NODE	Add <? And ?> to the extracted value

SAX(Simple API for XML)

Summary

SAX is the application program API that reads the XML document, and this regards the XML document in one long character string. SAX generates the Event whenever element or property is recognized while reading the character strings from the first, one by one. Implement the function to perform whenever EVENToccurs using **event handler technology.**

Course of SAX Program Implementation


SAX

이벤트 처리 방식으로 XML 문서 처리

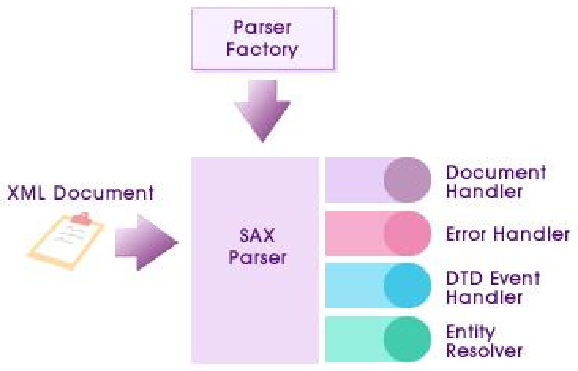
SAX 는 XML 문서를 맨 앞부터 순차적으로 읽어 가면서 요소, 속성 그리고 텍스트가 인식될 때마다 이벤트를 발생한다.

각 이벤트가 발생할 때마다 XML 문서가 읽혀질(파싱) 때 등록된 핸들러의 정해진 메서들가 호출된다.

➩ 콜백 메카니즘 적용

1. Develop a handler(event handling object) for the events to be generated.
2. Create the SAX object.
3. Read the XML document while registering handler already implemented.

Sample Source



* Inherit the following interfaces depending on which type of the handler will be implemented if required.

org.xml.sax.ContentHandlder
org.xml.sax.DTDHandler
org.xml.sax.EntityResolver
org.xml.sax.ErrorHandler

* org.xml.sax.ContentHandler: this handler is the core of SAX and processes general document events.

void characters(char[] ch,intstart,int length) // called when the document data is recognized.
voidendDocument() // called when the end of document is recognized.
voidendElement(StringnamespaceURI,StringlocalName,StringqName)
// called when the end of element is recognized.
voidendPrefixMapping(String prefix) // called when the termination of prefix-URI name space is recognized.
voidignorableWhitspace(char[] ch,int start, int length)
// called when the ignorable blank is recognized among element contents.
voidprocessingInstruction(Stringtarget,String data) // Called when PI is recognized.
voidsetDocumentLocator(Locator locator) // called to deliver objects that informs the location information where event occurs.
voidskippedEntity(Strig name) // called when skipped entity is recognized.
voidstartDocument() // called when the starting of document is recognized.
void startElement(StringnamespaceURI,StringlocalName,StringqName,Attributesatts)
// called when the starting of element is recognized.
voidstartPrefixMapping(Stringprefix,Stringuri)
// "called when the starting of prefix-URI name space is recognized."

* org.xml.sax.DTDHandler: called to handle the DTD events required for basic parsing. That is, it is called when the notation method and unparsed entity declaration are met.

voidnotationDecl(Stringname,StringpublicId,StringsystemId) // called when DTD declaration event is recognized.
void unparsedEntityDecl(Stringname,StringpublicId,StringsystemId,StringnotationName)
// called when the unparsed entity declaration event is recognized.

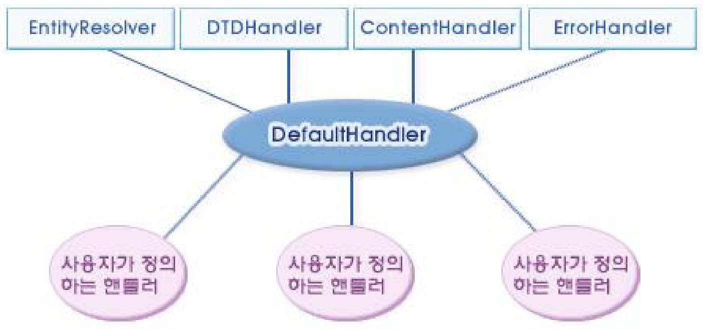
* org.xml.sax.EntityResolver: called to refer to external entity. If there is no external entity reference in the document, this interface is not required.

InputSourceresolveEntity(StringpublicId,StringsystemId) // process the function related to expansion entity processing.

* org.xml.sax.ErrorHandler: used to process error. For the parser, this handler is called to process all warnings and errors.

```
void error(SAXParseException exception) // called if recoverable error occurs.
voidfatalError(SAXParseException exception) // called if irrecoverable error occurs.
void warning(SAXParseException exception) // called if warning error occurs.
```

* DefaultHandlerclass: to override and implement the necessary methods only, inherit this interface and use DefaulatHandlerclass overriding the abstract methods defined in each interface.


EntityResolver
DTDHandler
ContentHandler
ErrorHandler
DefaultHandler
사용자가 정의 하는 핸들러
사용자가 정의 하는 핸들러
사용자가 정의 하는 핸들러

```
//Handler class implemented by inheriting DefaultHandler
classSampleHandler extends DefaultHandler
{
public void startDocument()
   {
logger.debug("XML started.");
   }
public void endDocument()
   {
logger.debug("XML is terminated.");
   }
}
```

* Create SAX object: create SAXParser object after creating SAXParseFactory objects similar to how to create DOM objects.

```
SAXParserFactoryspf = SAXParserFactory.newInstance();
SAXParsersp = spf.newSAXParser();
```

* Register handler and read XML document: after creating the object of SAXParser and object of handler class, select one of several overloaded methods as shown below and read XML as SAX object.

```
void parse(Filef,DefaultHandler dh)
void parse(InputStream is, DefaultHandler dh)
void parse(InputStream is, DefaultHandler dh)
void parse(Stringuri, DefaultHandler dh)
// Example of registering handler and reading XML document
SampleHandlersh = new SampleHandler();
// Read the document.
sp.parse(new FileInputStream("text.xml"),sh);
```

XML Manipulation

Configuration

```
<bean id="xmlCofig" class="egovframework.rte.fdl.xml.EgovXmlset">
<property name="domconcrete" ref="domconcreteCont" />
```

```
<property name="saxconcrete" ref="saxconcreteCont" />
</bean>
<bean id="domconcreteCont" class="egovframework.rte.fdl.xml.EgovConcreteDOMFactory"/>
<bean id="saxconcreteCont" class="egovframework.rte.fdl.xml.EgovConcreteSAXFactory"/>
```

PROPERTIES	Description
domconcrete	Concrete Class creating EgovDOMValidatorService
saxconcrete	Concrete Class creating EgovSAXValidatorService

```
<context:property-placeholder location="classpath*:spring/egovxml.properties" />
<bean id="xmlconfig" class="egovframework.rte.fdl.xml.XmlConfig">
	<property name="xmlpath" value="${egovxmlsaved.path}" />
</bean>
```

PROPERTIES	Description
xmlpath	Designate the location of directory for creating XML document

```
// XML default saving directory
egovxmlsaved.path=C:\\Temp\\
```

▶Example of egovxml.propertiescontents

Sample Source

```
/** abstractXMLFactoryServiceinherited Class **/
@Resource(name = "domconcreteCont")
EgovConcreteDOMFactorydomconcrete = null;

/** abstractXMLFactoryServiceinherited Class **/
@Resource(name = "saxconcreteCont")
EgovConcreteSAXFactorysaxconcrete = null;

/** AbstractXMLUtilityinheritedDOMValidator **/
EgovDOMValidatorServicedomValidator = null;
/** AbstractXMLUtilityinheritedSAXValidator **/
EgovSAXValidatorServicesaxValidator = null;
```

Create DOM Service

```
@Test
public void ModuleTest() throws UnsupportedException
{
	domValidator = domconcrete.CreateDOMValidator();
	logger.debug("fileName:"+fileName);
	domValidator.setXMLFile(fileName);
}
```

Create SAX Service

```
@Test
public void ModuleTest() throws UnsupportedException
{
	saxValidator = saxconcrete.CreateSAXValidator();
	logger.debug("fileName:"+fileName);
	saxValidator.setXMLFile(fileName);
}
```

well-formed, Validation Test

◎ Execute Validation test while performing well-formed test of XMLdocument(optional)

```
public void WellformedValidate(booleanused,booleanisvalid,AbstractXMLUtility service) throws
ValidatorException
```

```
{
          if(used)
          {
                    if(service.parse(isvalid))
                     {
                              if(isvalid)
                              logger.debug("Validation document.");
                              else
                              logger.debug("well-formed document.");
                     }
          }
}
```

PARAMETER	**Description**
isvalid	Validation test or not

View XPATH

◎ View the search expression(expression) such as Element or Attribute to search

```
public   voidXPathResult(booleanused,AbstractXMLUtilityservice,Document doc) throws JDOMException
{
if(used)
   {
Listlist = service.getResult(doc,"//*[@*]");
viewEelement(list);
   }
}
```

PARAMETER	Description
expr	Validation search formula
doc	Document object

Create XML

◎ Create XML document using the element entered.

```
public void
createNewXML(booleanused,AbstractXMLUtilityservice,Documentdoc,StringEleName,Listlist,String path)
throwsJDOMException,TransformerException,FileNotFoundException
{
if(used)
service.createNewXML(doc,EleName, list,path);
}
```

PARAMETER	Description
doc	Document object
EleName	Root name
list	Created Element List
path	Path of XML document to be created

Add Element

◎ Add entered element to XML document

```
public void
addElement(booleanused,AbstractXMLUtilityservice,Documentdoc,StringEleName,Listlist,String path)
throwsJDOMException,TransformerException,FileNotFoundException
{
if(used)
service.addElement(doc,EleName,list,path);
```

}

PARAMETER	Description
doc	Document object
EleName	Root name
list	Created Element List
path	Path of XML document to be created

Add TextNode Element

◎ Add the entered Text Element to XML document

```
public void addTextElement(booleanused,AbstractXMLUtilityservice,Document doc,
StringelemName,Listlist,String path)
throwsJDOMException,TransformerException,FileNotFoundException
{
if(used)
service.addTextElement(doc,elemName,list,path);
}
```

PARAMETER	Description
doc	Document object
EleName	Root name
list	Created Element List
path	Path of XML document to be created

UpdateTextNode Element

◎Update to entered Text Element

```
public void updTextElement(booleanused,AbstractXMLUtilityservice,Documentdoc,Listlist,String path)
throwsJDOMException,TransformerException,FileNotFoundException
{
        if(used)
        service.updTextElement(doc,list,path);
}
```

PARAMETER	Description
doc	Document object
list	Created Element List
path	Path of XML document to be created

Delete Element

◎ Delete entered Element

```
public void delElement(booleanused,AbstractXMLUtilityservice,Documentdoc,StringEleName,String path)
throwsJDOMException,TransformerException,FileNotFoundException
 {
if(used)
        service.delElement(doc, EleName,path);
 }
```

PARAMETER	Description
doc	Document object
EleName	Element name
path	Path of XML document to be created

Update Element

```
public void updElement(booleanused,AbstractXMLUtilityservice,Document doc,
StringoldElement, StringnewElement,String path)
throwsJDOMException,TransformerException,FileNotFoundException
{
if(used)
service.updElement(doc, oldElement, newElement,path);
}
```

PARAMETER	Description
doc	Document object
oldElement	Element name to be updated
newElement	Updated Element name
path	Path of XML document to be created

Execution Sequence of Guide Program

1. Set Classpath of egovframework.rte.fdl.xml-1.0.0-SNAPSHOT.jar in an execution management repository.
2. Create a spring folder and then set Classpath.
3. Make sure to copy egovxmlCfg.xml and egovxml.properties files to a spring folder in the execution management Repository.
4. Set contents of egovxmlCfg.xml by changing the location of egovxml.properties file.
5. Change and set egovxmlsaved.path of egovxml.properties.
6. Copy context-test.xml,context-xmltest.xml required for test to the spring folder or the location set earlier.
7. After setting is completed, execute ControlXMLTest.java.

Reference