Xml作业：

1. 创建一个xpath和xlst之间的文档转换。

package com.zte.util;

import java.io.File;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.dom4j.Document;

import org.dom4j.DocumentException;

import org.dom4j.Element;

import org.dom4j.XPath;

import org.dom4j.io.SAXReader;

/\*\*

\*

\* @author xueguosong 2014-09-09

\*/

public class Transform {

//将xml文件转换成xsl文件

@SuppressWarnings("unchecked")

public static void xmlToXsl(File xml,File xsl)throws Exception{

try {

SAXReader saxReaderXml=new SAXReader();

Document docXml = saxReaderXml.read(xml);

Element rootElement = docXml.getRootElement();

String uri = rootElement.getNamespaceURI();

Map<String, String> map=new HashMap<String, String>();

map.put("ns", uri);

//查找需要转换的根节点

XPath mappingXPath = rootElement.createXPath("//ns:mapping");

mappingXPath.setNamespaceURIs(map);

Element element = (Element)mappingXPath.selectSingleNode(docXml);

if(element!=null)

{

//消息节点

String messageValue = element.attributeValue("message");

List<Element> eleMap = element.elements("map");

StringBuilder xpaths=new StringBuilder();

List lists=new ArrayList();

if(eleMap!=null){

for(Element e:eleMap){

String data = e.attributeValue("data");

String to = e.attributeValue("to");

if(data!=null){

lists.add(new String[]{data,to});

}

}

}

if(xpaths.toString().endsWith(",")){

xpaths =new StringBuilder(xpaths.substring(0, xpaths.length()-1));

}

if(messageValue!=null&&!"".equals(messageValue)){

XSLCreator xslCre=new XSLCreator();

try {

xslCre.createXSL(lists, messageValue,xsl);

} catch (Exception e) {

e.printStackTrace();

}

}

}else{

throw new Exception("mapping node is not found!");

}

} catch (DocumentException e) {

throw new DocumentException(e);

}

}

/\*\*

\* @param args

\*/

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

xmlToXsl(new File("D:\\test\\source.xml"), new File("D:\\test\\result.xml"));

}

}

package com.zte.util;

/\*\*

\*

\* @author xueguosong 2014-09-04

\*/

public class XMLConstants {

// 默认数据节点

public static final String MESSAGE = "MESSAGE";

// xml编码

public static final String ENCODING = "UTF-8";

// xsd默认命名空间

public static final String XSD\_DEFAULT\_NAMESPACE = "xs";

// xsd定义的默认数据节点

public static final String XSD\_DEFAULT\_DATANODE = "MESSAGE";

// xsd复合类型节点

public static final String XSD\_COMPLEX\_TYPE = "complexType";

// xsd序列节点

public static final String XSD\_SEQUENCE = "sequence";

// xsd元素节点

public static final String XSD\_ELEMENT = "element";

// xsd注解节点

public static final String XSD\_ANNOTATION = "annotation";

// xsd注解文档节点

public static final String XSD\_DOCUMENTATION = "documentation";

// xsd简单类型节点

public static final String XSD\_SIMPLE\_TYPE = "simpleType";

// xsd限制节点

public static final String XSD\_RESTRICTION = "restriction";

// xsd name属性

public static final String XSD\_ATTRIBUTE\_NAME = "name";

// xsd type属性

public static final String XSD\_ATTRIBUTE\_TYPE = "type";

// xsd base属性

public static final String XSD\_ATTRIBUTE\_base = "base";

// 用来描述xsd中的unbounded节点信息 for-eache节点

public static final String XSD\_UNBOUNDED = "[unbounded]";

public static final String XSD\_UNBOUNDED\_REPLATE = "\\[unbounded\\]";

public static final String XSL\_ELEMENT\_FOREACH = "for-each";

public static final String XSL\_ELEMENT\_SELECT = "select";

/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\* 创建xslt基础变量配置 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

public static final String STYLESHEET = "stylesheet";

public static final String OUTPUT = "output";

public static final String OUTPUTENCODING = "encoding";

public static final String INDENT = "indent";

public static final String VERSION = "version";

public static final String VERSIONNUM = "1.0";

public static final String NAMESPACE = "xsl";

public static final String NAMESPACEADDRESS = "http://www.w3.org/1999/XSL/Transform";

public static final String TEMPLATE = "template";

public static final String MATCH = "match";

public static final String APPLYTEMPLATES = "apply-templates";

public static final String VALUEOF = "value-of";

public static final String SELECT = "select";

public static final String XMLENCODING = "UTF-8";

public static final String ROOTSPER = "/";

public static final String DOUBELROOTSPER = "//";

public static final String SPER = ":";

/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*XSD DID相关定义\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

public static final String XS\_ELEMENT = "xs:element";

public static final String XS\_SEQUENCE = "xs:sequence";

public static final String XS\_CHOICE = "xs:choice";

public static final String XS\_COMPLEXTYPE = "xs:complexType";

public static final String XS\_SIMPLETYPE = "xs:simpleType";

public static final String XS\_RESTRICTION = "xs:restriction";

public static final String XS\_ANNOTATION = "xs:annotation";

public static final String XS\_DOCUMENTATION = "xs:documentation";

}

package com.zte.util;

/\*\*

\*

\* @author xueguosong 2014-09-04

\*/

public class XSDNode {

// 节点名称

private String name;

// 节点XPath

private String xPath;

// 节点描述

private String annotation;

// 节点类型

private String type;

// 业务用路径,描述路径中的unbound节点

private String unboundedXpath;

private String isUnbounded;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getXPath() {

return xPath;

}

public void setXPath(String path) {

xPath = path;

}

public String getAnnotation() {

return annotation;

}

public void setAnnotation(String annotation) {

this.annotation = annotation;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public String getUnboundedXpath() {

return unboundedXpath;

}

public void setUnboundedXpath(String unboundedXpath) {

this.unboundedXpath = unboundedXpath;

}

public String getIsUnbounded() {

return isUnbounded;

}

public void setIsUnbounded(String isUnbounded) {

this.isUnbounded = isUnbounded;

}

}

package com.zte.util;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import org.dom4j.Attribute;

import org.dom4j.Document;

import org.dom4j.Element;

import org.dom4j.Node;

import org.dom4j.io.SAXReader;

/\*\*

\*

\* @author xueguosong 2014-09-11

\*/

public class XSDReader {

private List<XSDNode> list = new ArrayList<XSDNode>();

/\*\*

\* 解析XSD，返回数据节点对象列表

\*

\* @param xsd

\* @return

\* @throws Exception

\*/

public List<XSDNode> paserXSD(String xsd) throws Exception {

SAXReader saxReader = new SAXReader();

Document doc = saxReader.read(xsd);

Element element = doc.getRootElement();

String basePath = "//" + XMLConstants.XSD\_DEFAULT\_NAMESPACE+ ":element[@name=\"" + XMLConstants.MESSAGE + "\"]";

Element dataElement = (Element) element.selectSingleNode(basePath);

String elementPath = "//" + XMLConstants.XSD\_DEFAULT\_NAMESPACE+ ":element";

paseData(dataElement, "//", elementPath, "//");

return list;

}

/\*\*

\* 转换XSD的数据节点，生成XSDNode对象

\*

\* @param element

\* @param xPath

\* @param xsdPath

\* @param unboundedXpath

\*/

public void paseData(Element element, String xPath, String xsdPath,

String unboundedXpath) {

// 获取节点name属性

String nodeName = element.attributeValue("name");

// 组装xml文档中节点的XPath

xPath += nodeName;

unboundedXpath += nodeName;

// 并列多节点限制属性

String maxOccurs = element.attributeValue("maxOccurs");

if (maxOccurs != null && !"1".equals(maxOccurs)&& !("//" + XMLConstants.MESSAGE + "").equals(xPath)) {// 节点可以有多个

unboundedXpath += XMLConstants.XSD\_UNBOUNDED;

}

// 下一个element元素的xsd XPath:sequence

String currentXsdPath\_sequence = xsdPath + "[@name=\"" + nodeName+ "\"]" + "/" + XMLConstants.XS\_COMPLEXTYPE + "/"+ XMLConstants.XS\_SEQUENCE + "/" + XMLConstants.XS\_ELEMENT;

// 下一个element元素的xsd XPath:choice

String currentXsdPath\_choise = xsdPath + "[@name=\"" + nodeName + "\"]"+ "/" + XMLConstants.XS\_COMPLEXTYPE + "/"+ XMLConstants.XS\_CHOICE + "/" + XMLConstants.XS\_ELEMENT;

String currentXsdPath = "";

// 查找该节点下所有的element元素

List<Node> elementNodes\_sequence = element.selectNodes(currentXsdPath\_sequence);

List<Node> elementNodes\_choice = element.selectNodes(currentXsdPath\_choise);

List<Node> elementNodes = null;

if (elementNodes\_sequence != null && elementNodes\_sequence.size() > 0) {

elementNodes = elementNodes\_sequence;

currentXsdPath = currentXsdPath\_sequence;

} else {

elementNodes = elementNodes\_choice;

currentXsdPath = currentXsdPath\_choise;

}

if (elementNodes != null && elementNodes.size() > 0) {// 如果下面还有element,说明不是叶子

Iterator<Node> nodes = elementNodes.iterator();

while (nodes.hasNext()) {

if (!xPath.endsWith("/")) {

xPath += "/";

unboundedXpath += "/";

}

Element ele = (Element) nodes.next();

paseData(ele, xPath, currentXsdPath, unboundedXpath);

}

} else { // 该element为叶子

XSDNode xsdNode = new XSDNode();

// 获取注释节点

String annotation = "";

Node annotationNode = element.selectSingleNode(xsdPath

+ "[@name=\"" + nodeName + "\"]/"

+ XMLConstants.XS\_ANNOTATION + "/"

+ XMLConstants.XS\_DOCUMENTATION);

if (annotationNode != null)

annotation = annotationNode.getText();

// 获取节点类型属性

String nodeType = "";

Attribute type = element.attribute("type");

if (type != null)

nodeType = type.getText();

else {

String spath = xsdPath + "[@name=\"" + nodeName + "\"]/"+ XMLConstants.XS\_SIMPLETYPE + "/"+ XMLConstants.XS\_RESTRICTION;

Element typeNode = (Element) element.selectSingleNode(spath);

if (typeNode != null) {

Attribute base = typeNode.attribute("base");

if (base != null)

nodeType = base.getText();

}

}

xsdNode.setName(nodeName);

xsdNode.setXPath(xPath);

xsdNode.setAnnotation(annotation);

xsdNode.setType(nodeType);

xsdNode.setUnboundedXpath(unboundedXpath);

if (xPath.equals(unboundedXpath))// 说明不是unbounded节点

xsdNode.setIsUnbounded("0");

else

xsdNode.setIsUnbounded("1");

list.add(xsdNode);

}

}

public static void main(String[] args) {

try {

XSDReader reader = new XSDReader();

List<XSDNode> nodes = reader.paserXSD("D:\\test\\ExpReport.xsd");

for (XSDNode node : nodes) {

System.out.println(node.getUnboundedXpath());

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

package com.zte.util;

import java.io.File;

import java.io.FileWriter;

import java.io.Writer;

import java.util.List;

import org.dom4j.Document;

import org.dom4j.DocumentHelper;

import org.dom4j.Element;

import org.dom4j.Namespace;

import org.dom4j.io.OutputFormat;

import org.dom4j.io.XMLWriter;

/\*\*

\*

\* @author xueguosong 2014-09-04

\* 创建XSLT文件

\*/

public class XSLCreator {

/\*\*

\* <p>

\* 根据传入的 XPath集合生成XSLT文件

\* Xpath路径类似//MESSAGE/BASEINFO/INFO\_SOURCE/infoRegionLimitList

\* [unbounded]/Region[unbounded]/china/jobProvince[unbounded]

\* [unbounded]标记路径中的unbounded节点

\* 列:

\* list.add(new String()[]{"/a/b/c","/d/e/f"});数组中第0个元素表示数据源

\* list.add(new String()[]{"/a/b/c","/d/e/f"});数组中第1个元素表示数据源将要被转换的目标源

\* </p>

\*

\* @param lists xpath路径集合

\* @param message 存放消息节点

\* @param outFile 存放xls内容文件

\* @throws Exception

\*/

public void createXSL(List lists,String message,File outFile) throws Exception {

// 解析传入的Xpath

// String[] xPathsArr = paths.split(",");

// 生成根节点,并创建XSLT基本信息

String rootElement=null;

if(message==null){

rootElement=XMLConstants.MESSAGE;

}else{

rootElement=message;

}

Document documents = DocumentHelper.createDocument();

Element documentsRoot = documents.addElement(XMLConstants.NAMESPACE + XMLConstants.SPER+ XMLConstants.STYLESHEET)

.addAttribute(XMLConstants.VERSION, XMLConstants.VERSIONNUM);

documentsRoot.add(new Namespace(XMLConstants.NAMESPACE,XMLConstants.NAMESPACEADDRESS));

//创建格式节点

documentsRoot.addElement(XMLConstants.NAMESPACE + XMLConstants.SPER+ XMLConstants.OUTPUT)

.addAttribute(XMLConstants.OUTPUTENCODING, XMLConstants.XMLENCODING)

.addAttribute(XMLConstants.INDENT, "yes");

Element elementTemplate = documentsRoot.addElement(XMLConstants.NAMESPACE + XMLConstants.SPER+ XMLConstants.TEMPLATE);

elementTemplate.addAttribute(XMLConstants.MATCH, XMLConstants.ROOTSPER);

// Element root = elementTemplate.addElement(rootElement);//apply-templates节点需要添加根节点

Element root = elementTemplate;//apply-templates节点不需要添加根节点

root.addElement(XMLConstants.NAMESPACE + XMLConstants.SPER+ XMLConstants.APPLYTEMPLATES).addAttribute(XMLConstants.SELECT, rootElement);

// 创建节点模板

Element elementTemplateChild = documentsRoot.addElement(XMLConstants.NAMESPACE + XMLConstants.SPER+ XMLConstants.TEMPLATE);

elementTemplateChild.addAttribute(XMLConstants.MATCH, rootElement);

// 生成各个节点转换模板

for (Object xPathArs : lists)

{

String[] entry = (String[])xPathArs;

String xPathData=entry[0];

String xPathTo=entry[1];

// 记录同一路径下unbound节点的个数

int unbounds = 0;

String[] unboundPathSplit = xPathData.split(XMLConstants.XSD\_UNBOUNDED\_REPLATE);

Element ele = elementTemplateChild;

// 获取Message后各级节点名称

String[] nodes = xPathTo.split("/");

// 标记节点是否为unbound节点

boolean isUnbound = false;

// 标记改路径下的循环节点个数

for (int i = 0; i < nodes.length; i++) {

String node = nodes[i];

// 如果含有unbound标识,且topath为最后一个节点时该节点为unbound节点

if(unboundPathSplit.length>1&&i==nodes.length-2){

isUnbound = true;

// 将同一路径下的unbound节点数加1

unbounds++;

}

else{

isUnbound = false;

}

if (node != null && !node.equals("")) {

// 先判断节点下有没有循环,如果有循环,跳过循环节点

Element each = ele.element(XMLConstants.XSL\_ELEMENT\_FOREACH);

if (each != null){

ele = each;

}

// 如果没有该节点,创建一个节点作为当前节点

if (ele.element(node) == null) {

// 判断当前节点是否为for-each,如果是,一直找到不为for-each的父亲节点

while (ele.getName().equals(XMLConstants.XSL\_ELEMENT\_FOREACH)){

ele = ele.getParent();

}

// 如果该节点为unbound节点则创建xsl:for-each节点

if (isUnbound) {

// 生成一个for-each节点

ele = ele.addElement(XMLConstants.NAMESPACE+ XMLConstants.SPER+ XMLConstants.XSL\_ELEMENT\_FOREACH);

// 获取对应的selectPath

String selectPath = unboundPathSplit[unbounds - 1];

// 去除中间路径中开始的"/"标记,否则XSL会不合法,判断unbounds不等于1是因为第一个路径是由"//"开始,故需要排除

if (unbounds != 1 && selectPath.startsWith("/")){

selectPath = selectPath.substring(1, selectPath.length());

}

// 为for-each节点添加select属性

ele.addAttribute(XMLConstants.XSL\_ELEMENT\_SELECT,selectPath);

}

ele = ele.addElement(node);

} else{

// 如果已经有该节点,获取节点作为当前节点

ele = ele.element(node);

}

}

// 如果当前节点为叶子节点,则为叶子节点选择数据

if (i == nodes.length - 1) {

// 如果当前路径中存储unbound节点情况,取split后数组最后一位

if (unbounds != 0) {

// 如果叶子节点也是unbound节点,for-each里面的select路径用"."表示

if (unbounds == unboundPathSplit.length)

// 选择当前节点数据

xPathTo = ".";

else {

xPathTo = unboundPathSplit[unbounds];

if (!"".equals(xPathTo) && xPathTo.startsWith("/"))

xPathTo = xPathTo.substring(1, xPathTo.length());

}

}else{

//没有for-eache节点

xPathTo=xPathData;

}

ele.addElement(XMLConstants.NAMESPACE + XMLConstants.SPER+ XMLConstants.VALUEOF).addAttribute(XMLConstants.SELECT, xPathTo);

}

}

}

documents.setXMLEncoding(XMLConstants.XMLENCODING);

Writer fileWriter=new FileWriter(outFile);

XMLWriter xmlWriter=new XMLWriter(fileWriter,OutputFormat.createPrettyPrint());

xmlWriter.write(documents);

xmlWriter.close();

}

}

source.xml文件内容

<mapping message="studentlist">

<map data="/studentlist/student[unbounded]/name" to="/studentlist/student/name"/>

<map data="/studentlist/student[unbounded]/sex" to="/studentlist/student/sex"/>

<map data="/studentlist/student[unbounded]/score" to="/studentlist/student/score"/>

</mapping>

student.xml文件内容

<?xml version="1.0" encoding="gb2312"?>

<studentlist>

<student id="A101">

<name>李华</name>

<sex>男</sex>

<birthday>1978.9.12</birthday>

<score>92</score>

<skill>Java</skill>

<skill>Oracle</skill>

<skill>C Sharp</skill>

<skill>SQL Server</skill>

</student>

<student id="A101">

<name>张辽</name>

<sex>女</sex>

<birthday>078.9.12</birthday>

<score>192</score>

<skill>大刀</skill>

<skill>长矛</skill>

<skill>剑</skill>

<skill>SQL Server</skill>

</student>

</studentlist>

根据source.xml转换后生成xslt文件结果

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:output encoding="UTF-8" indent="yes"/>

<xsl:template match="/">

<xsl:apply-templates select="studentlist"/>

</xsl:template>

<xsl:template match="studentlist">

<studentlist>

<xsl:for-each select="/studentlist/student">

<student>

<name>

<xsl:value-of select="name"/>

</name>

<sex>

<xsl:value-of select="sex"/>

</sex>

<score>

<xsl:value-of select="score"/>

</score>

</student>

</xsl:for-each>

</studentlist>

</xsl:template>

</xsl:stylesheet>

1. 做一个xlst和html之间的文档转换

<!--

- XSLT is a template based language to transform Xml documents

It uses XPath to select specific nodes

for processing.

- A XSLT file is a well formed Xml document

-->

<!-- every StyleSheet starts with this tag -->

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

version="1.0">

<!-- indicates what our output type is going to be -->

<xsl:output method="html" />

<!--

Main template to kick off processing our Sample.xml

From here on we use a simple XPath selection query to

get to our data.

-->

<xsl:template match="/">

<html>

<head>

<title>Welcome to <xsl:value-of select="/company/name"/></title>

<style>

body,td {font-family:Tahoma,Arial; font-size:9pt;}

</style>

</head>

<body>

<h2>Welcome to <xsl:value-of select="/company/name"/></h2>

<p/>

<b>Our contact details:</b>

<br/>

<br/>

<xsl:value-of select="/company/name"/>

<br/>

<xsl:value-of select="/company/address1"/>

<br/>

<xsl:value-of select="/company/address2"/>

<br/>

<xsl:value-of select="/company/city"/>

<br/>

<xsl:value-of select="/company/country"/>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

xml文件如下:

<company>

<name>XYZ Inc.</name>

<address1>One Abc Way</address1>

<address2>Some avenue</address2>

<city>Tech city</city>

<country>Neverland</country>

</company>

public static void Transform(string sXmlPath, string sXslPath)

{

try

{

//load the Xml doc

XPathDocument myXPathDoc = new XPathDocument(sXmlPath);

XslTransform myXslTrans = new XslTransform();

//load the Xsl

myXslTrans.Load(sXslPath);

//create the output stream

XmlTextWriter myWriter = new XmlTextWriter

("result.html", null);

//do the actual transform of Xml

myXslTrans.Transform(myXPathDoc, null, myWriter);

myWriter.Close();

}

catch (Exception e)

{

Console.WriteLine("Exception: {0}", e.ToString());

}

}

生成result.html如下:

<html>

<head>

<title>Welcome to XYZ Inc.</title>

<style>

body,td {font-family:Tahoma,Arial; font-size:9pt;}

</style>

</head>

<body>

<h2>Welcome to XYZ Inc.</h2>

<p/>

<b>Our contact details:</b>

<br />

<br />

XYZ Inc.

<br />

One Abc Way

<br />

Some avenue

<br />

Tech city

<br />

Neverland

</body>

</html>