Java编程中更新XML文档的常用方法 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022_Java_E7_BC_96_E7_A8_8B_c104_252456.htm 本文简要的讨论了Java语言编程中更新XML文档的四种常用方法,并且分析这四种方法的优劣。其次,本文还对如何控制Java程序输出的XML文档的格式做了展开论述。 JAXP是Java API for XML Processing的英文字头缩写,中文含义是:用于XML文档处理的使用Java语言编写的编程接口。JAXP支持DOM、SAX、XSLT等标准。为了增强JAXP使用上的灵活性,开发者特别为JAXP设计了一个Pluggability Layer,在Pluggability Layer的支持之下,JAXP既可以和具体实现DOM API、SAX API 的各种XML解析器(XML Parser,例如Apache Xerces)联合工作,又可以和具体执行XSLT标准的XSLT处理器(XSLT Processor,例如Apache Xalan)联合工作。应用Pluggability Layer的好处在于:我们只需要熟悉JAXP各个编程接口的定义即可,而不需要对所采用的具体的XML解析器、XSLT处理器有很深入的了解。比如在某个Java程序中,通过JAXP调用XML解析器Apache Crimson对XML文档进行处理,如果我们希望使用别的XML解析器(比如Apache Xerces),以便提高该程序的性能,那么原程序代码可能不需要任何改变,直接就可以使用(你所需要做的事情只是将包含Apache Xerces代码的jar文件加入到环境变量CLASSPATH中,而将包含Apache Crimson代码的jar文件在环境变量CLASSPATH中删除)。 目前JAXP已经应用的十分普遍了,可以说是Java语言中处理XML文档的标准API。有些初学者在学习使用JAXP的过程中,经常会提出这样的问题:我编写的程序对DOM Tree做了更新,但是

当程序退出以后,原始的XML文档并没有改变,还是老样子,如何实现对原始XML文档和DOM Tree的同步更新呢?咋一看来,在JAXP中似乎没有提供相应的接口/方法/类,这是很多初学者都感到困惑的问题。本文的主旨就在于解决这个问题,简单的介绍几种常用的同步更新原始XML文档和DOM Tree的方法。为了缩小讨论的范围,本文所涉及的XML解析器仅包括Apache Crimson和Apache Xerces,而XSLT处理器仅仅使用Apache Xalan。 方法一:直接读写XML文档 这也许是最笨最原始的办法了。当程序获取DOM Tree之后,应用DOM模型的Node接口的各个方法对DOM Tree进行更新,下一步应该对原始的XML文档进行更新了。我们可以运用递归的办法或者是应用TreeWalker类,遍历整个DOM Tree,与此同时,将DOM Tree的每一个节点/元素依次写入到预先打开的原始XML文档中,当DOM Tree被遍历完全之后,DOM Tree和原始的XML文档就实现了同步更新。实际中,这个方法极少使用,不过如果你要编程实现自己的XML解析器,这种方法还是有可能用得上的。 方法二:使用XMLDocument类 使用XmlDocument类?JAXP中分明没有这个类呀!是不是作者搞错了?没有错!就是使用XmlDocument类,确切的说,是使用XMLDocument类的write()方法。 在上文已经提到过,JAXP可以和各种各样的XML解析器联合使用,这次我们选用的XML解析器是Apache Crimson。XmlDocument(org.apache.crimson.tree.XmlDocument)是Apache Crimson的一个类,并不包含于标准的JAXP中,难怪在JAXP的文档中找不到XmlDocument类的芳踪呢。现在问题出来了,如何应用XmlDocument类来实现更新XML文档的功能?在XMLDocument类中提供了下面三个write()方法(根

据Crimson最新的版本------Apache Crimson 1.1.3): public void write (OutputStream out) throws IOExceptionpublic void write (Writer out) throws IOExceptionpublic void write (Writer out, String encoding) throws IOException 上述三个write()方法的主要作用就是输出DOM Tree中的内容到特定的输出介质中,比如文件输出流、应用程序控制台等等。那么又如何使用上述三个write()方法呢?请看下面的Java程序代码片断: String name="fancy".DocumentBuilder parser.DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance().try { parser = factory.newDocumentBuilder(). Document doc = parser.parse("user.XML"). Element newlink=doc.createElement(name). doc.getDocumentElement().appendChild(newlink).((XmlDocument)doc).write(new FileOutputStream(new File("xuser1.XML"))).}catch (Exception e) { //to log it } 在上面的代码中,首先创建了一个Document对象doc,获取完整的DOM Tree,然后应用Node接口的appendChild()方法,在DOM Tree的最后追加了一个新节点(fancy),最后调用XmlDocument类的write(OutputStream out)方法,把DOM Tree中的内容输出到xuser.xml中(其实也可以输出到user.xml,更新原始的XML文档,在这里为了便于做对比,故而输出到xuser.xml文件中)。需要注意的是不能直接对Document对象doc直接调用write()方法,因为JAXP的Document接口并没有定义任何write()方法,所以必须将doc由Document对象强制转换为XMLDocument对象,然后才能调用write()方法,在上面的代码中使用的是write(OutputStream out)方法,这个方法使用缺省的UTF-8编

码输出DOM Tree中的内容到特定的输出介质中,如果DOM Tree中包含中文字符,那么输出的结果有可能是乱码,亦即存在所谓的"汉字问题",解决的办法是使用write (Writer out, String encoding)方法,显式指定输出时的编码,例如将第二个参数设为"GB2312",这时即不存在"汉字问题",输出结果能够正常显示中文字符。 完整的例子请参考下列文件: AddRecord.java(见附件)、user.XML(见附件)。该例子的运行环境为:Windows XP Professional、JDK 1.3.1。为了能够正常编译运行AddRecord.Java这个程序,你需要到网址http://XML.apache.org/dist/crimson/去下载Apache Crimson,并将所获取的crimson.jar文件加入到环境变量CLASSPATH中。注意: Apache Crimson的前身是Sun Project X Parser,后来不知何故,由X Parser演变为Apache Crimson,至今Apache Crimson的很多代码都是从X Parser中直接移植过来的。比如上文用到的XmlDocument类,它在X Parser中是com.sun.xml.XmlDocument,到了Apache Crimson中摇身一变,就变成了org.apache.crimson.tree.XmlDocument类,其实它们的绝大部分代码是一样的,可能就package语句和import语句以及文件开头的一段lience有所不同而已。早期的JAXP是和X Parser捆绑在一起的,因此一些老的程序使用了com.sun.xml包,如果你现在重新编译它们,有可能不能通过,肯定就是因为这个原因。后来的JAXP和Apache Crimson捆绑在一起,比如JAXP 1.1,如果你使用JAXP 1.1,那么不需要额外下载Apache Crimson,也能够正常编译运行上面的例子(AddRecord.java)。最新的JAXP 1.2 EA(Early ACCESS)改弦更张,采用性能更好的Apache Xalan和Apache Xerces分别作为XSLT处理器和XML解

析器,不能直接支持Apache Crimson了,所以如果你的开发环境采用了JAXP 1.2 EA或者是Java XML Pack(内含JAXP 1.2 EA),那么将无法直接编译运行上面的例子(AddRecord.Java),你需要额外下载并安装Apache Crimson。 方法三:使用TransformerFactory和Transformer类 在JAXP中所提供的标准的更新原始XML文档的方法就是调用XSLT引擎,亦即使用TransformerFactory和Transformer类。请看下面的Java代码片断: //首先创建一个DOMSource对象,该构造函数的参数可以是一个Document对象//doc代表更改后的DOM Tree。DOMSource doms = new DOMSource (doc).//创建一个File对象,代表DOM Tree所包含的数据的输出介质,这是一个XML文件。 File f = new File ("XMLOutput.XML").//创建一个StreamResult对象,该构造函数的参数可以取为File对象。StreamResult sr = new StreamResult (f).//下面调用JAXP中的XSLT引擎来实现输出DOM Tree中的数据到XML文件中的功能。//XSLT引擎的输入为DOMSource对象,输出为StreamResut对象。try {//首先创建一个TransformerFactory对象,再由此创建Transformer对象。Transformer//类相当于一个XSLT引擎。通常我们使用它来处理XSL文件,但是在这里我们使//用它来输出XML文档。TransformerFactory tf=TransformerFactory.newInstance(). Transformer t=tf.newTransformer ().//关键的一步, 调用Transformer对象(XSLT引擎)的transform()方法,该方法的第一//个参数是DOMSource对象,第二个参数是StreamResult对象。t.transform(doms,sr). }catch (TransformerConfigurationException tce){

System.out.println("Transformer Configuration Exception\n-----").tce.printStackTrace(). }catch (TransformerException te) { System.out.println ("Transformer Exception\n---------"). te.printStackTrace (). } 在实际的应用中,我们可以应用传统的DOM API从XML文档中获取DOM Tree,然后根据实际的需求对DOM Tree执行各种操作,得到最终的Document对象,接下来可以由此Document对象创建DOMSource对象,剩下的事情就是照搬上面的代码了,程序运行完毕后, XMLOutput.xml就是你所需要的结果(当然了,你可以随意更改StreamResult类构造函数的参数,指定不同的输出介质,而不必是千篇一律的XML文档)。 这个方法最大的好处在于可以随心所欲的控制DOM Tree中的内容输出到输出介质中的格式,但是光靠TransformerFactory类和Transformer类并不能实现这个功能,还需要依赖OutputKeys类的帮助。 完整的例子请参考下列文件: AddRecord2.java(见附件)、user.xml(见附件)。该例子的运行环境为:Windows XP Professional、JDK 1.3.1。为了能够正常编译运行AddRecord2.java这个程序,你需要到网址http://java.sun.com去下载安装JAXP 1.1或者Java XML Pack(Java XML Pack已经内含JAXP了)。 OutputKeys类 javax.XML.transform.OutputKeys类和Java.util.Properties类配合使用,可以控制JAXP的XSLT引擎(Transformer类)输出XML文档的格式。请看下面的代码片断: //首先创建一个TransformerFactory对象,再由此创建Transformer对象。TransformerFactory tf=TransformerFactory.newInstance(). Transformer t=tf.newTransformer ().//获取Transformser对象的输出属性,亦即XSLT引擎的缺省输出属性,这是一

个//Java.util.Properties对象。 Properties properties = t.getOutputProperties(). //设置新的输出属性:输出字符编码为GB2312,这样可以支持中文字符,XSLT引擎所输出//的XML文档如果包含了中文字符,可以正常显示,不会出现所谓的"汉字问题"。 //请留意OutputKeys类的字符串常数OutputKeys.ENCODING 。properties.setProperty(OutputKeys.ENCODING,"GB2312"). /更新XSLT引擎的输出属性。t.setOutputProperties(properties). //调用XSLT引擎,按照输出属性中的设置,输出DOM Tree中的内容到输出介质中 。t.transform(DOMSource_Object,StreamResult_Object). 从上面的程序代码,我们不难看出,通过设置XSLT引擎(Transformer类)的输出属性,可以控制DOM Tree中的内容的输出格式,这对于我们定制输出内容是很有帮助的。那么JAXP的XSLT引擎(Transformer类)有那些输出属性可以设置呢? Javax.XML.transform.OutputKeys类定义了很多字符串常数,它们都是可以自由设置的输出属性,常用的输出属性如下所示: public static final Java.lang.String METHOD 可以设为"XML"、"html"、"text"等值。 public static final Java.lang.String VERSION 所遵循规范的版本号,如果METHOD设为"XML",那么它的值应该设为"1.0",如果METHOD设为"html",那么它的值应该设为"4.0",如果METHOD设为"text",那么这个输出属性会被忽略。 public static final Java.lang.String ENCODING 设置输出时所采用的编码方式,比如"GB2312"、"UTF-8"等等,如果将其设置为"GB2312",可以解决所谓的"汉字问题"。 public static final Java.lang.String OMIT_XML_DECLARATION 设置输出

到XML文档中时是否忽略XML声明,亦即类似于: <?XML version="1.0" standalone="yes" encoding="utf-8" ?> 这样的代码。它可选的值有"yes"、"no"。 public static final Java.lang.String INDENT IDENT设定XSLT引擎在输出XML文档时,是否自动添加额外的空格,它可选的值为"yes"、"no"。 public static final Java.lang.String MEDIA_TYPE MEDIA_TYPE设定输出文档的MIME类型。 如果设定XSLT引擎的输出属性呢?下面我们来总结一下: 首先是获取XSLT引擎(Transformer类)的缺省输出属性的集合,这需要使用Transformer类的getOutputProperties()方法,返回值是一个Java.util.Properties对象。 Properties properties = transformer.getOutputProperties(). 然后是设定新的输出属性,比如:
properties.setProperty(OutputKeys.ENCODING,"GB2312").properties.setProperty(OutputKeys.METHOD,"html").properties.setProperty(OutputKeys.VERSION,"4.0").…… 最后是更新XSLT引擎(Transformer类)的缺省输出属性的集合,这需要使用Transformer类的setOutputProperties()方法,参数是一个Java.util.Properties对象。 我们编写了一个新的程序,其中应用了OutputKeys类,用以控制XSLT引擎的输出属性,该程序的架构和前一个程序(AddRecord3.java)大致相同,不过输出结果略有不同。完整的代码请参考下列文件: AddRecord3.java(见附件)、user.xml(见附件)。该例子的运行环境为:Windows XP Professional、JDK 1.3.1。为了能够正常编译运行AddRecord3.java这个程序,你需要到网址http://java.sun.com去下载安装JAXP 1.1或者Java XML Pack(Java XML Pack内含JAXP了)。 方法四:使用Xalan XML

Serializer 方法四其实是方法三的一个变种,它需要Apache Xalan和Apache Xerces的支持才能够运行。例子代码如下所示: //首先创建一个DOMSource对象,该构造函数的参数可以是一个Document对象//doc代表更改后的DOM Tree。DOMSource domSource = new DOMSource (doc). //创建一个DOMResult对象,临时保存XSLT引擎的输出结果。DOMResult domResult = new DOMResult().//下面调用JAXP中的XSLT引擎来实现输出DOM Tree中的数据到XML文件中的功能。//XSLT引擎的输入为DOMSource对象,输出为DOMResut对象。try { //首先创建一个TransformerFactory对象,再由此创建Transformer对象。Transformer //类相当于一个XSLT引擎。通常我们使用它来处理XSL文件,但是在这里我们使 //用它来输出XML文档。TransformerFactory tf=TransformerFactory.newInstance(). Transformer t=tf.newTransformer (). //设置XSLT引擎的属性(必不可少,否则会产生"汉字问题")。 Properties properties = t.getOutputProperties(). properties.setProperty(OutputKeys.ENCODING,"GB2312"). t.setOutputProperties(properties). //关键的一步, 调用Transformer对象 (XSLT引擎)的transform()方法,该方法的第一 //个参数是DOMSource对象,第二个参数是DOMResult对象。 t.transform(domSource,domResult). //创建缺省的Xalan XML Serializer,使用它将临时存放在DOMResult对象 //(domResult)中的内容以输出流的形式输出到输出介质中。 Serializer serializer = SerializerFactory.getSerializer(OutputProperties.getDefaultMethodProperties("XML")). //设置Xalan XML Serializer的输出属性,这一

步必不可少,否则也可能产生 //所谓的"汉字问题"。 Properties prop=serializer.getOutputFormat(). prop.setProperty("encoding","GB2312"). serializer.setOutputFormat(prop). //创建一个File对象,代表DOM Tree所包含的数据的输出介质,这是一个XML文件。 File f = new File ("xuser3.XML"). //创建文件输出流对象fos,请留意构造函数的参数。 FileOutputStream fos=new FileOutputStream(f). //设置Xalan XML Serializer的输出流。 serializer.setOutputStream(fos). //串行化输出结果。 serializer.asDOMSerializer().serialize(domResult.getNode()).}catch (Exception tce){ tce.printStackTrace(). } 这个方法不太常用,而且似乎有点画蛇添足,所以我们就不展开讨论了。完整的例子请参考下列文件: AddRecord4.java(见附件)、user.XML(见附件)。该例子的运行环境为:Windows XP Professional、JDK 1.3.1。为了能够正常编译运行AddRecord4.Java这个程序,你需要到网址http://XML.apache.org/dist/去下载安装Apache Xalan和Apache Xerces。 或者是到网址http://java.sun.com/xml/download.html去下载安装Java XML Pack。因为最新的Java XML Pack(Winter 01版)包含了Apache Xalan和Apache Xerces技术在内。 结论: 本文简略的讨论了Java语言编程中更新XML文档的四种方法。第一种方法是直接读写XML文件,这种方法十分繁琐,而且比较容易出错,极少使用,除非你需要开发自己的XML Parser,否则不会使用这种方法。第二种方法是使用Apache Crimson的XmlDocument类,这种方法极为简单,使用方便,如果你选用Apache Crimson作为XML解析器,那么不妨使用这种方法,不过这种方法似乎效率不高(源于效率低下的Apache Crimson),另

外,高版本的JAXP或者是Java XML Pack、JWSDP不直接支持Apache Crimson,亦即这种方法不通用。第三种方法是使用JAXP的XSLT引擎(Transformer类)来输出XML文档,这种方法也许是标准的方法了,使用起来十分灵活,特别是可以自如控制输出格式,我们推荐采用这种方法。第四种方法是第三种方法的变种,采用了Xalan XML Serializer,引入了串行化操作,对于大量文档的修改/输出有优越性,可惜的是要重复设置XSLT引擎的属性和XML Serializer的输出属性,比较麻烦,而且依赖于Apache Xalan和Apache Xerces技术,通用性略显不足。 除了上面讨论的四种方法以外,实际上应用别的API(比如JDOM、Castor、XML4J、Oracle XML Parser V2)也有很多办法可以更新XML文档,限于篇幅,在这里就不一一讨论了。 