XML解析

简介

JAXP介绍

Java API for XML Processing,SUN公司为java开发人员处理XML提供的统一的编程接口，其本身并不提供对XML的解析，需要调用其它解析器实现

XML编程常用包

```
javax.xml.parsers
org.xml.sax
org.w3c.dom
```

解析器

- 验证型解析器：检查xml的良构性和有效性
- 非验证型解析器：检查xml的良构型

基于事件的解析器：SAX

- 对xml文档边读边解析(不需要将整个文档都加载到内存)
- 对xml文档一次性顺序读取，不能随机读取
- 读到后面就忘了前面的内容，要想再次获取前面内容，必须重新读取
- 对xml文档内容是只读的

基于内存树的解析器：DOM

- 将整个xml文档读入内存，形成驻留在内存中的树型结构，然后在进行解析
- 对xml文档可以进行反复随机读取
- 对xml文档内容可以进行增、删、改、查

SAX编程步骤

- 创建SAX解析器

```
XMLReader parser = XMLReaderFactory.createXMLReader();
org.xml.sax.XMLReader
org.xml.sax.helpers.XMLReaderFactory
```

- 设置SAX解析器

```
String VALIDATION = "http://xml.org/sax/features/validation";
String NS = "http://xml.org/sax/features/namespaces";
parser.setFeature(VALIDATION,true);
parser.setFeature(NS,true);
```

- 创建事件处理器

  可以实现ContentHandler接口，也可以继承DefaultHandler类

```
org.xml.sax.ContentHandler
org.xml.sax.helpers.DefaultHandler
class MyHandler extends DefaultHandler{
    startDocument()
   endDocument()
    startElement(...)
    endElement(...)
    characters(...)
}
ContentHandler handler = new MyHandler();
```

- 绑定事件处理器

```
parser.setContentHandler(handler);
```

- 解析xml

```
parser.parse("student.xml");
```

DOM编程步骤：

- 创建DOM解析器工厂

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
javax.xml.parsers.DocumentBuilderFactory
```

- 设置DOM解析器工厂

```
String VALIDATION = "http://xml.org/sax/features/validation";
String NS = "http://xml.org/sax/features/namespaces";
factory.setFeature(VALIDATION,true);
factory.setFeature(NS,true);
factory.setValidating(true);
factory.setNamespaceAware(true);
```

- 创建文档构建器

```
DocumentBuilder builder = factory.newDocumentBuilder();
javax.xml.parsers.DocumentBuilder
```

- 创建文档对象
- 读： Document doc = builder.parse(”student.xml”);
- 创建： Document doc = builer.newDocument();
  org.w3c.dom.Document
- 对xml进行处理

dom4j解析开发包

- 项目中导入dom4j的包
- 使用相关API进行编程

```
public class Dom4jTest {
  @Test
  public void demo1() throws Exception {
    //1.获取Document对象
    SAXReader reader = new SAXReader();
    Document document = reader.read(new File("src/com/briup/dtd/web.xml"));
    //2.获取根元素
    Element rootElement = document.getRootElement();
    //3.获取version属性值
    String version = rootElement.attributeValue("version");
    System.out.println("version:" + version);
    //4.获取所有子元素
    List<Element> allChildElement = rootElement.elements();
    //5.遍历子元素，<servlet>、<servlet-mapping>
    for (Element element : allChildElement) {
      //6.处理<servlet>
      String name = element.getName();
      //System.out.println(name);
      if (name.equals("servlet")) {
        //7.处理<servlet-name>
        Element servletNameElement = element.element("servlet-name");
        String servletName = servletNameElement.getText();
```

```
            System.out.print(servletName + ":");
            //8.处理<servlet-class>
            Element servletClassElement = element.element("servlet-class");
            String servletClass = servletClassElement.getText();
            System.out.println(servletClass);
          }
        }
      }
}
```

- 模拟服务器端程序（Servlet）执行案例实现

```
Map<String, String> data=new HashMap<String, String>();
@Before
public void demo4Before()throws Exception{
     //1.获取Document对象
     SAXReader reader=new SAXReader();
     Document document=reader.read(new File("src/com/briup/servlet/web.xml"));
     //2.获取根元素
     Element rootElement=document.getRootElement();
     //3.获取所有子元素
     List<Element> allChildElement=rootElement.elements();
     //4.遍历子元素，<servlet>、<servlet-mapping>
     for(Element element:allChildElement){
     String name=element.getName();
     //5.处理<servlet>
     if(name.equals("servlet")){
     //6.处理<servlet-name>
     String servletName=element.element("servlet-name").getText();
     //7.处理<servlet-class>
     String servletClass=element.element("servlet-class").getText();
     //放入集合data(servlet-name,servlet-class)
     data.put(servletName,servletClass);
     }
     if(name.equals("servlet-mapping")){
     //8.处理<servlet-name>
     String servletName=element.element("servlet-name").getText();
     //9.处理<url-pattern>
     String urlPattern=element.element("url-pattern").getText();
     //根据servlet-name，从集合中获取servlet-class
     String servletClass=data.get(servletName);
     //放入新的数据（url-pattern,servlet-class）
     data.put(urlPattern,servletClass);
     //根据servlet-name，删除原有数据
     data.remove(servletName);
     }
     }
     }
@Test
public void demo4()throws Exception{
     //客户端浏览器要访问的路径
     String url="/hello";
     //找到路径对应的类
     String className=data.get(url);
     Class clazz=Class.forName(className);
     MyServlet myServlet=(MyServlet)clazz.newInstance();
     myServlet.init();
     myServlet.service();
     myServlet.destroy();
     }
```