




软件开发环境

主讲教师 刘凡

fanliu@hhu.edu.cn



第十一章

JSP中使用XML



本章主要内容

◆11.1 XML文件的基本结构

◆11.2 XML声明

◆11.3 标记

◆11.4 DOM解析器

◆11.5 SAX解析器

◆11.6 XML与CSS

概述

- 如果Web应用没有用到数据库独有的一些特性，而仅仅是查询数据而已，并且这些数据 可能占用较大的存储空间。在这种情况下，如果选择用数据库来处理数据显然得不偿失，因为使用数据库要付出降低程序运行效率的代价。
- 当需要查询文件中的某些内容时，显然希望这种文件应当具有某种特殊的形式结构，即文件应当按照一定的标准来组织数据，这就是XML文件。

11.1 XML文件的基本结构

- XML是eXtensible Markup Language 缩写，称之为可扩展置标语言。
- 所谓可扩展是指XML允许用户按照XML的规则自定义标记。
- XML文件是由标记构成的文本文件，使得XML文件能够很好地体现数据的结构和含义。
- W3C推出XML的主要目的是使得Internet上的数据相互交流更方便，让文件的内容更加显而易懂。

11.1 XML文件的基本结构

Time.xml

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

<列车时刻表>

<T28>

<开车时间>20点58分</开车时间>

<终到时间>08点18分</终到时间>

</T28>

<T876>

<开车时间>23点12分</开车时间>

<终到时间>07点25分</终到时间>

</T876>

</列车时刻表>

11.1 XML文件的基本结构

- 规范的XML文件应当用“XML声明”开始、文件有当且仅有一个根标记，其它标记都必须封装在根标记中。
- 根标记可以有若干个子标记，称根标记的子标记。根标记的子标记还可以有子标记，以此类推。
- 如果一个标记仅仅包含文本，这样的标记称为叶标记。
- 非空标记必须由“开始标记”与“结束标记”组成、空标记没有“开始标记”和“结束标记”等等。
- XML文件的标记必须形成树型结构，即一个标记的子标记必须包含于该标记的开始标记和结束标记之间。简单地说，就是标记之间不允许出现交叉。

11.1 XML文件的基本结构

- W3C吸取了HTML发展的教训，对XML指定了严格的语法标准。为了检查XML文件是否规范，一个简单的方法就是用浏览器，打开XML文件，如果XML是规范的，浏览器将显示XML源文件，否则，将显示出错信息。

11.2 XML声明

- 一个规范的XML文件应当以XML声明作为文件的第一行，在其前面不能有空白、其他的处理指令或注释。
- XML声明以 “<?xml” 标识 开始、以 “?” 标识 结束。以下是一个最基本的XML声明：

<?xml version="1.0" ?>

- 一个简单的XML声明中可以只包含属性version（目前该属性的值只可以取1.0），指出该XML文件使用的XML版本。

11.2 XML声明

- XML声明中也可以指定encoding属性的值，该属性规定XML文件采用哪种字符集进行编码。如果在XML声明中没有指定encoding属性的值，那么该属性的默认值是UTF-8。

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

- 如果encoding属性的值为“UTF-8”，XML文件必须选择“UTF-8”编码来保存。

11.2 XML声明

- 如果在编写XML文件时只准备使用ASCII字符和汉字，也可以将encoding属性的值设置为gb2312. 这时XML文件必须使用ANSI编码保存.

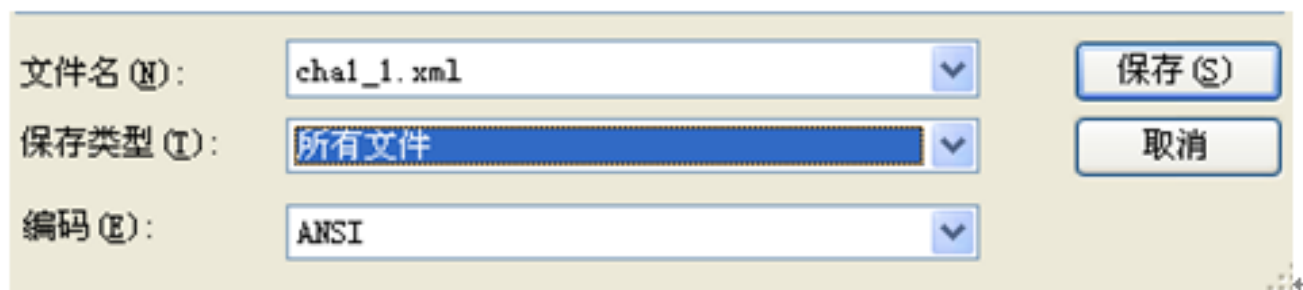


图 9.2 encoding 值是 gb2312 时 XML 文件的保存

11.2 XML声明

- 如果在编写XML文件时只准备使用ASCII字符，也可以将encoding属性的值设置为ISO-8859-1. 这时XML文件必须使用ANSI编码保存.

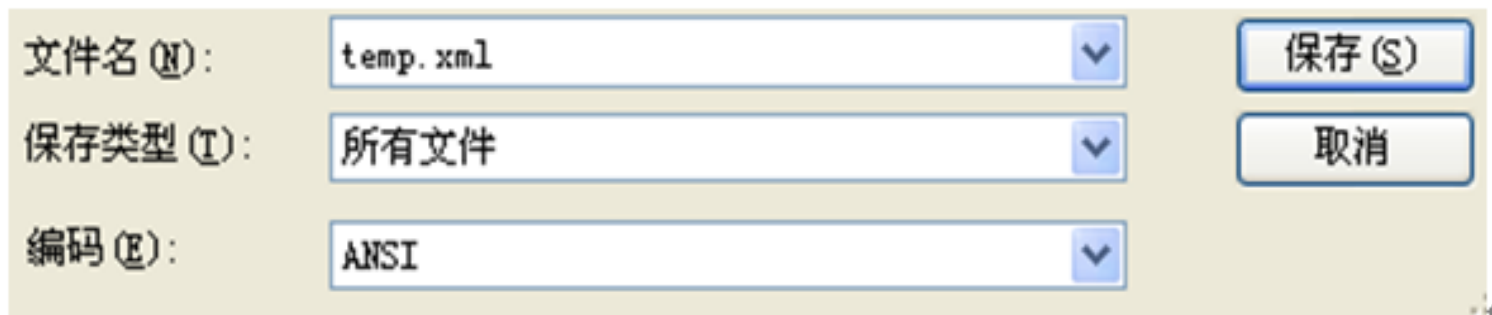


图 9.3 encoding 值是 ISO-8859-1 时 XML 文件的保存

11.3 标记

- XML文件是由标记构成的文本文件。标记的名称可以由字母、数字、下划线“_”、点号“.”或连字符“-”组成，但必须以字母或下划线开头。

标记名称区分大小写，例如：

<name>Kevin</name>

与

<Name> Kevin</Name>

是完全不同的标记。

11.3 标记

1. 空标记

- 空标记就是不含有任何内容的标记，即不含有子标记或文本内容。
- 空标记不需要开始标记和结束标记
- 空标记以 “<” 标识开始，用 “/>” 标识结束

<空标记的名称 属性列表 />

或

<空标记的名称 />

例如：

<chair width="24" height="12" />

<和标记名称之间不要含有空格

11.3 标记

2. 非空标记

- 由“开始标记”与“结束标记”组成，“开始标记”与“结束标记”之间是该标记所含有的内容。

<sex> 男 </sex>

- 开始标记以“<”标识开始，用“>”标识结束
- 结束始标记以“</”标识开始，用“>”标识结束

11.3 标记

3. CDATA段

- 标记内容中的文本数据中不可以含有左尖括号、右尖括号、与符号、单引号和双引号这些特殊字符。如果标记内容想使用这些特殊字符，办法之一是通过使用CDATA段。
- CDATA段用“<![CDATA[”作为段的开始，用“]]>”作为段的结束。段开始和段结束之间称为CDATA段的内容，CDATA段中的内容可以包含任意的字符

```
<hello>  
  <![CDATA[  
    boolean boo=true&&false  
    <你好>  
  ]]>  
</hello>
```


11.3 标记

4. 属性

- 属性是一个名值对，即属性必须由名字和值组成。属性必须在非空标记的开始标记或空标记中声明，用“=”为属性指定一个值。

<桌子 width="300" height="600" length="1000">

吃饭用的

</桌子>

<椅子 color="red" />

11.4 DOM解析器

- 使用XML解析器可以从XML文件中解析出所需要的数据。
- DOM（Document Object Model，文档对象模型）是W3C制定的一套规范标准。DOM规范的核心是按树型结构处理数据。
- 简单地说，DOM解析器必须按着DOM规范在内存中按树型结构组织数据，DOM解析器通过读入XML文件在内存中建立一个“树”，也就是说XML文件的标记、标记的文本内容都会和内存中“树”的某个节点相对应。一个应用程序可以方便地操作内存中“树”的节点来处理XML文档，获取自己所需要的数据。

11.4 DOM解析器

1.使用DOM解析器的基本步骤

- ① 使用 javax.xml.parsers 包中的 DocumentBuilderFactory 类调用其类方法 newInstance() 实例化一个 DocumentBuilderFactory 对象：

```
DocumentBuilderFactory factory= DocumentBuilderFactory.newInstance();
```

- ② factory 对象调用 newDocumentBuilder() 方法返回一个 DocumentBuilder 对象（称做DOM解析器）：

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

- ③ builder 对象调用 public Document parse(File f) 方法解析参数 f 指定的文件，并返回一个实现了 Document 接口的实例，该实例被称做 Document 对象：

```
Document document= builder.parse(new File("price.xml"));
```

11.4 DOM解析器

2.Document对象

- DOM解析器负责在内存中建立Document对象，即调用parse方法返回一个Document对象

```
Document document= builder. parse(new File("price.xml")) ;
```

- parse方法将整个被解析的XML文件封装成一个Document对象，即将XML文件和内存中的Document对象相对应。
- Document对象就是一棵“树”，文件中的标记都和Document对象中的某个节点相对应。

11.4 DOM解析器

2.Document对象-对象结构

- Element类、Text类和CDATASection类都是实现了Node接口的类，是比较重要的三个类，这些类的对象分别被称作Document对象中的Element节点、Text节点和CDATASection节点.
- 一个Element节点中还可含有Element节点、Text节点和CDATASection节点。比如Document对象的根节点就是一个Element节点.

11.4 DOM解析器

对于下面的XML文件E.xml，相对应的Document对象如图9.4所示

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

```
<root>
```

```
<A>    hello
```

```
</A>
```

```
<B>
```

```
<B1>
```

```
    welcome.
```

```
</B1>
```

```
<![CDATA[  
    x>100&&y>30;  
]]>
```

```
How are you.
```

```
</B>
```

```
</root>
```

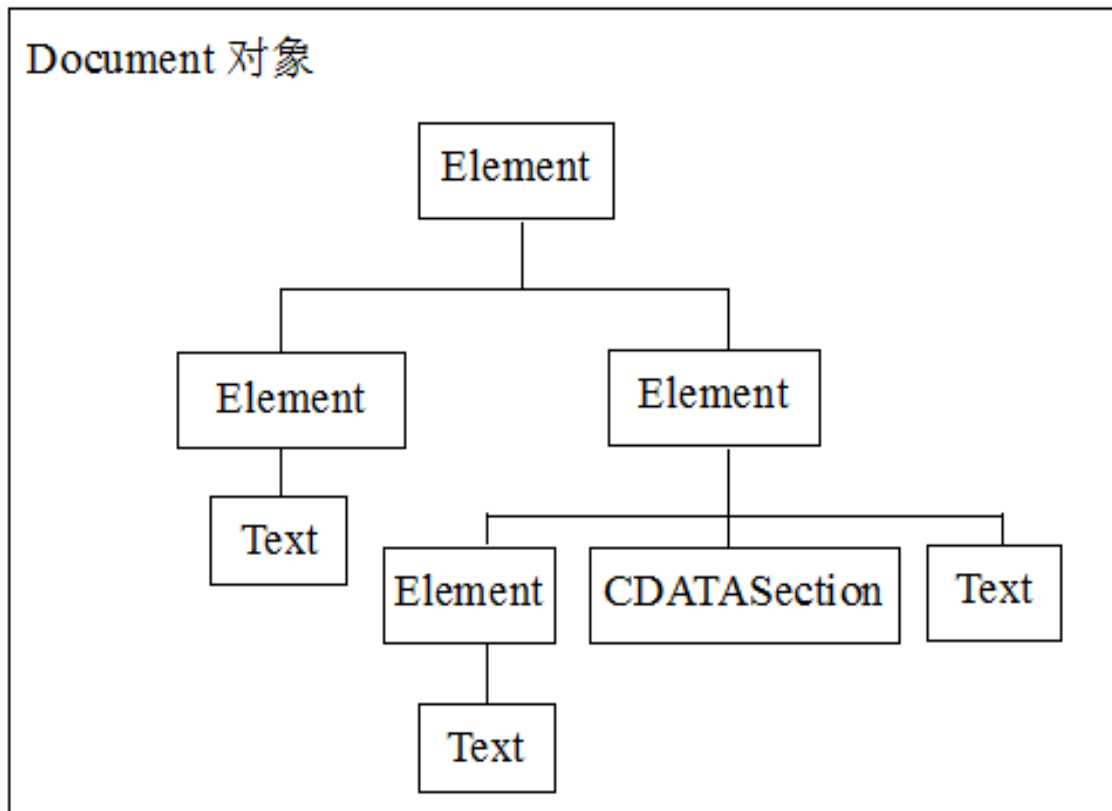


图 9.4 E.xml 文件对应的 Document 对象

11.4 DOM解析器

2.Document对象-Element节点

Element节点经常使用下列方法获取和该节点相关的信息

- `String getTagName()` 返回该节点的名称，该名称就是和此节点相对应的XML中的标记的名称。
- `String getTextContent()` 返回当前节点的所有Text子孙节点中的文本内容（也就是返回相对应的XML文件中的标记及其子孙标记中含有的文本内容）。
- `String getAttribute(String name)` 返回节点中参数name的属性值，该属性值是和此节点对应的XML中标记中的属性值。

11.4 DOM解析器

2.Document对象-Element节点

Element节点经常使用下列方法获取和该节点相关的信息

- `Boolean hasAttribute(String name)` 判断当前节点是否有参数name指定的属性。
- `NodeList getElementByTagName(String name)` 返回一个NodeList对象，该对象由当前节点的Element类型子孙节点组成，这些子孙节点的名字由参数name指定。
- `NodeList getChildNodes()` 返回一个NodeList对象，该对象由当前节点の子节点组成。

11.4 DOM解析器

2.Document对象-Text节点

Text节点使用String `getWholeText()` 方法获取节点 中的文本（包括其中的空白字符）。

2.Document对象- CDATASection节点

CDATASection节点使用String `getWholeText()` 方法获取该节点中的文本，即CDATA段中的文本（包括其中的空白字符）。

11.4 DOM解析器

3.查询成绩

本节通过具体的例子来讲述如何使用DOM解析器从XML文件中解析出所需要的数据。

Score.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<成绩>
  <学号 number="a1001">
    <姓名>赵一</姓名>
    <高等数学 课程性质="必修">89 </高等数学>
    <大学物理 课程性质="必修">88 </大学物理>
    <摄影艺术 课程性质="选修">良好 </摄影艺术>
  </学号>
  <学号 number="a1002">
    <姓名>钱二</姓名>
    <高等数学 课程性质="必修">77 </高等数学>
    <大学物理 课程性质="必修">66 </大学物理>
    <摄影艺术 课程性质="选修">良好 </摄影艺术>
  </学号>
```

11.4 DOM解析器

Score.xml

<学号 number="a1003">

<姓名>孙三</姓名>

<高等数学 课程性质="必修">75 </高等数学>

<大学物理 课程性质="必修">69 </大学物理>

<摄影艺术 课程性质="选修">良好 </摄影艺术>

</学号>

<学号 number="a1004">

<姓名>李四</姓名>

<高等数学 课程性质="必修">76 </高等数学>

<大学物理 课程性质="必修">87 </大学物理>

<摄影艺术 课程性质="选修">良好 </摄影艺术>

</学号>

</成绩>

11.4 DOM解析器

input.jsp

```
<%@ page contentType="text/html;charset=GB2312" %>
<HTML><BODY bgcolor=cyan>
  <FORM action="byNumber" method=post name=form>
    输入学号，查询成绩：
    <INPUT type="text" name="studentNmber">
    <INPUT TYPE="submit" value="提交" >
  </FORM>
</HTML>
```

11.4 DOM解析器

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
  <servlet>
    <servlet-name>number</servlet-name>
    <servlet-class>sun.yourservlet.Number
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>number</servlet-name>
    <url-pattern>/byNumber</url-pattern>
  </servlet-mapping>
</web-app>
```

11.4 DOM解析器

Number.java

```
package sun.yourservlet;
import java.io.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Number extends HttpServlet
{ public void init(ServletConfig config) throws ServletException
    {super.init(config);
    }
public void doPost(HttpServletRequest request,HttpServletResponse
    response) throws ServletException,IOException
    { boolean boo=false;
      response.setContentType("text/html;charset=GB2312");
      PrintWriter out=response.getWriter();
      out.println("<html><body>");
      String searchedNumber=request.getParameter("studentNmber");
```

11.4 DOM解析器

Number.java

```
try{ DocumentBuilderFactory factory=DocumentBuilderFactory. newInstance();
    DocumentBuilder builder= factory.newDocumentBuilder();
    Document document= builder. parse(new File("D:\\1000\\Score.xml")) ;
    Element root=document.getDocumentElement() ; //获取根节点
    //返回根节点的Element子节点，这些子节点对应着XML文件中的“学号”标记：
    NodeList elemnetNodes=root.getElementsByTagName("学号") ;
    int size=elemnetNodes.getLength();
    for(int k=0;k<size;k++)
    { Node node=elemnetNodes.item(k);
      if(node.getNodeType()==Node.ELEMENT_NODE) //判断node节点的类型
      { //再得到node节点的number属性的值：
        String stuNumber=((Element)node).getAttribute("number");//获取学号
        if(stuNumber.equals(searchedNumber))
        { boo=true;
          NodeList childNodes=node.getChildNodes();//获取node的全部子节点
          for(int i=0;i<childNodes.getLength();i++)
          { Node child=childNodes.item(i);
            if(child.getNodeType()==Node.ELEMENT_NODE)
            { String nodeName=((Element)child).getTagName().trim();
```

11.4 DOM解析器

Number.java

```
        out.println("<BR>" + nodeName);
        String courseType=((Element)child).getAttribute("课程性质");
        String contentStr=((Element)child).getTextContent();
        if(nodeName.startsWith("姓名"))
            out.println(": " + contentStr);
        else
            out.println("(" + courseType + ") : " + contentStr);
    }
}
}
}
}
}
}
if(boo==false)
    out.println("不存在您要查询的学号！");
}
catch(Exception ee)
{
}
out.println("</body></html>");
}
}
```


11.5 SAX解析器

- 如果XML文件较大，相应的Document对象就要占据较多的内存空间.
- 和DOM解析器不同的是，SAX解析器不在内存中建立和XML文件相对应的树型结构数据，SAX解析器的核心是事件处理机制。和DOM解析器相比，SAX解析器占有的内存少，对于许多应用程序，使用SAX解析器来获取XML数据具有较高的效率.

11.5 SAX解析器

1.使用SAX解析器的基本步骤

- ① 使用 javax.xml.parsers 包中的 SAXParserFactory 类调用其类方法 newInstance() 实例化一个 SAXParserFactory 对象：

```
SAXParserFactory factory=SAXParserFactory.newInstance();
```

- ② SAXParserFactory 对象调用 newSAXParser() 方法返回一个 SAXParser 对象，称之为 SAX 解析器：

```
SAXParser saxParser=factory.newSAXParser();
```

- ③ saxParser 对象调用 public void parse(File f,DefaultHandler dh) 方法解析参数 f 指定的 XML 文件。

```
saxParser.parse(new File("price.xml"), handler);
```

11.5 SAX解析器

2.SAX解析器的工作原理

- SAX解析器调用parse方法解析XML文件，parse方法的第2个参数dh是DefaultHandler类型，称为事件处理器。
- parse方法在解析XML文件的过程中，根据从文件中解析出的数据产生相应的事件，并报告这个事件给事件处理器dh，事件处理器dh就会处理所发现的数据，即处理器dh会根据相应的事件调用相应的方法来处理数据。
- parse方法必须等待事件处理器处理完毕后才能继续解析文件、报告下一个事件。

11.5 SAX解析器

3. 事件的产生与处理

- ① **文件开始事件与结束事件**：当解析器开始解析器XML文件时，就会报告“文件开始”事件给事件处理器，然后再陆续地报告其他的事件，最后报告“文件结束”事件。解析器报告“文件开始”事件，事件处理器就会调用 `startDocument()` 方法；解析器报告“文件结束”事件，事件处理器就会调用 `endDocument()` 方法。

11.5 SAX解析器

3. 事件的产生与处理

- ② 开始标记事件与结束标记事件：当解析器发现一个标记的开始标记时，报告开始事件给事件处理器，事件处理器调用**startElement**方法对发现的数据做出处理。

startElement(String uri,String localName,String qName,Attributes atts)

- 方法中atts是解析器发现的标记的全部属性，参数uri的取值就是解析器发现的标记的名称空间（如果没有名称空间uri是空字符串），localName是标记的名称，qName是带前缀的标记名称（如果名称空间的前缀）或标记名称（如果没有名称空间的前缀）。

11.5 SAX解析器

3. 事件的产生与处理

- 解析器报告完该标记的“标记开始”事件后，一定还会报告该标记的“标记结束”事件，事件处理器就会调用 `endElement` 方法进行处理

`endElement(String uri,String localName,String qName)`

- 如果一个标记是空标记，解析器也报告“标记开始”事件和“标记结束”事件。

11.5 SAX解析器

3. 事件的产生与处理

- ③ **文本数据事件**：当解析器解析报告 “文本数据” 事件给处理器，事件处理器就会然后调用**characters**方法对解析的数据做出处理

public void characters(char[] ch,int start,int length)

- 字符数组中ch存放的就是解析器解析出的文本数据，start是数组ch中存放字符的起始位置，length是存放的字符个数。
- **注意**：对于文本数据，解析器可能分成几个连续的“文本数据”报告给事件处理器。

11.5 SAX解析器

4.例子: **SAX**解析列车时刻表

trainList.xml

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

```
<trainList>
```

```
  <北京西站>
```

```
    <始发列车>
```

```
      <车次>
```

```
        <名字>T83次</名字>
```

```
        <运行区间>北京西-南京</运行区间>
```

```
        <开车时间>20:17</开车时间>
```

```
        <终到时间>09:10</终到时间>
```

```
      </车次>
```

```
      <车次>
```

```
        <名字>T199次</名字>
```

```
        <运行区间>北京西-上海</运行区间>
```

```
        <开车时间>23:25</开车时间>
```

```
        <终到时间>10:10</终到时间>
```

```
      </车次>
```

```
    </始发列车>
```


11.5 SAX解析器

trainList.xml

```
<终到列车>  
  <车次>  
    <名字>T84次</名字>  
    <运行区间>南京-北京西</运行区间>  
    <开车时间>21:17</开车时间>  
    <终到时间>10:10</终到时间>  
  </车次>  
  <车次>  
    <名字>T200次</名字>  
    <运行区间>上海-北京西</运行区间>  
    <开车时间>22:25</开车时间>  
    <终到时间>09:10</终到时间>  
  </车次>  
</终到列车>  
</北京西站>
```

11.5 SAX解析器

trainList.xml

```
<广州站>  
  <始发列车>  
    <车次>  
      <名字>T186次</名字>  
      <运行区间>广东-武汉</运行区间>  
      <开车时间>22:17</开车时间>  
      <终到时间>09:24</终到时间>  
    </车次>  
    <车次>  
      <名字>T78次</名字>  
      <运行区间>广东-长沙</运行区间>  
      <开车时间>18:25</开车时间>  
      <终到时间>11:10</终到时间>  
    </车次>  
  </始发列车>  
</终到列车>
```

11.5 SAX解析器

trainList.xml

```
<终到列车>
  <车次>
    <名字>T193次</名字>
    <运行区间>南京-广东</运行区间>
    <开车时间>21:17</开车时间>
    <终到时间>10:10</终到时间>
  </车次>
  <车次>
    <名字>T200次</名字>
    <运行区间>上海-广东</运行区间>
    <开车时间>21:15</开车时间>
    <终到时间>12:10</终到时间>
  </车次>
</终到列车>
</广州站>
</trainList>
```

11.5 SAX解析器

train.jsp

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="moon.yourbean.*"%>
<jsp:useBean id="ok" class="moon.yourbean.SAXBean" scope="page"/>
<jsp:setProperty name="ok" property="stationName" param="stationName"/>
<jsp:setProperty name="ok" property="startOrArrive" param="startOrArrive"/>
<HTML><BODY bgcolor=cyan><Font size=2>
  <FORM action="" Method="post" >
    选择站名:<Select name="stationName">
      <Option value="北京西站">北京西站
      <Option value="广东站">广东站
    </Select>
    选择始发或终到: <INPUT type="radio" name="startOrArrive" value="始
      发列车" checked="default">始发列车
    <INPUT type="radio" name="startOrArrive" value="终到列车">终到列车
    <BR> <Input type=submit value="提交">
  </FORM>
  <jsp:getProperty name="ok" property="stationName"/>,
  <jsp:getProperty name="ok" property="startOrArrive"/>:
  <jsp:getProperty name="ok" property="trainMessages"/>
</FONT></BODY></HTML>
```

11.5 SAX解析器

SAXBean.java

```
package moon.yourbean;
import javax.xml.parsers.*;
import org.xml.sax.helpers.*;
import org.xml.sax.*;
public class SAXBean
{
    String stationName="北京西站";           //站名
    String startOrArrive="";                 //始发或终到
    StringBuffer trainMessages=new StringBuffer(); //信息
    SAXParserFactory factory=null;
    SAXParser saxParser=null; //解析器
    MyHandler handler; //事件处理器
    public void setStationName(String s)
    {
        stationName=s.trim();
        try{ byte b[]=stationName.getBytes("ISO-8859-1");
            stationName=new String(b);
        }
        catch(Exception ee){}
    }
    public String getStationName()
    {
        return stationName;
    }
}
```

11.5 SAX解析器

SAXBean.java

```
public void setStartOrArrive(String s)
{
    startOrArrive=s.trim();
    try{ byte b[]=startOrArrive.getBytes("ISO-8859-1");
        startOrArrive=new String(b);
    }
    catch(Exception ee){}
}
public String getStartOrArrive()
{
    return startOrArrive;
}
public StringBuffer getTrainMessages()
{
    try{ factory=SAXParserFactory.newInstance() ;
        saxParser=factory.newSAXParser();
        handler=new MyHandler(trainMessages,stationName,startOrArrive);
        saxParser.parse("D:/1000/trainList.xml",handler);
    }
    catch(Exception e){ System.out.println(e);}
    return trainMessages;
}
}
```

11.5 SAX解析器

SAXBean.java

class MyHandler extends DefaultHandler

{ StringBuffer trainMessages;

String stationName,startOrArrive;

boolean 站名标记=false,始发或终到标记=false;

MyHandler(StringBuffer mess,String sName,String startOrArr)

{ trainMessages=mess;

stationName=sName;

startOrArrive=startOrArr;

}

public void startDocument()

{ trainMessages.append("<table border=2> ");

trainMessages.append("<tr>");

trainMessages.append("<th>车次名字</th>");

trainMessages.append("<th>运行区间</th>");

trainMessages.append("<th>始发时间</th>");

trainMessages.append("<th>终到时间</th>");

trainMessages.append("</tr>");

}

11.5 SAX解析器

SAXBean.java

```
public void endDocument()
{ trainMessages.append("</table> ");
}
public void startElement(String uri,String localName,
                        String qName,Attributes atts)
{ qName=qName.trim();
  if(qName.equals(stationName))
  { 站名标记=true;
    trainMessages.append(" "+qName);
  }
  if(qName.equals(startOrArrive))
  { 始发或终到标记=true;
  }
  if(qName.endsWith("车次"))
  { trainMessages.append("<tr>");
  }
}
```


11.5 SAX解析器

SAXBean.java

```
public void endElement(String uri,String localName,String qName)
{
    if(qName.startsWith(stationName))
    { 站名标记=false;}
    if(qName.startsWith(startOrArrive))
    { 始发或终到标记=false;}
    if(qName.endsWith("车次"))
    { trainMessages.append("</tr>");}
}
public void characters(char[] ch,int start,int length)
{
    String text=new String(ch,start,length);
    text=text.trim();
    if(站名标记==true&&始发或终到标记==true&&text.length()>0)
    { String str=text.trim();
      trainMessages.append("<td>"+str+"</td>");
    }
}
}
```

11.6 XML与CSS

- W3C为显示XML中所有标记所含有的文本数据发布了一个建议规范：CSS（层叠样式表）
- 为了让XML使用层叠样式表显示其中的文本数据，XML文件必须使用操作指令
- `<?xml-stylesheet href = "样式表的URI" type= "text/css" ?>`
- 将当前XML文件关联到某个层叠样式表，样式表的URI如果是一个文件的名字，该文件必须和XML文件在同一目录中，如果是一个URL，必须是有效可访问的
- `<?xml-stylesheet href="show.css" type="text/css" ?>`
- `<?xml-stylesheet href= "http://www.yahoo.com/show.css" type="text/css" ?>`

11.6 XML与CSS

1. 样式表

- 在CSS中，最重要的概念就是样式表。样式表是一组规则，通过这组规则告诉浏览器用什么样式来显示文本。比如，告诉浏览器使用什么样的字体、颜色和页边距来显示文本。一个样式表的格式如下：

文本代表

{ 样式规则

}

11.6 XML与CSS

1. 样式表

- 对于XML文件，样式表中的“文本代表”可以是标记的名称；样式表中的“样式规则”是若干个用分号分隔的“属性名：属性值”，例如，样式表

name

```
{ display:block;font-size:36pt;  
  font-weight:bold;  
}
```

- 用来显示标记“name”的文本内容，其中的“display:block;”告知浏览器将标记“<name>...</name>”所标记的文本内容显示在一个“块区域”。

11.6 XML与CSS

1. 样式表

- 如果有多个标记的内容需要用完全一样方式来显示，“文本代表”也可以是这些标记的名称用逗号分隔的字符串。如：

```
name,sex,birthday
```

```
{ display:block;font-size:36pt;  
  font-weight:bold;  
}
```

- 一个层叠样式表（CSS）就是由若干个样式表组成的文本文件（扩展名为.css），该文本文件可以使用ANSI或UTF-8编码来保存。
- 注意：文本代表中不要含有非**ASCII**字符，早期的**IE6.0**不支持这样的样式表。

11.6 XML与CSS

2. 文本的显示方式

1. 块方式

display:block

time

```
{ display:block;  
}
```

3. 按列表方式

display: list-item

time

```
{ display:list-item;  
}
```

2. 行方式

display:line

time

```
{ display:line;  
}
```

4. 不显示

display:none

time

```
{ display:none;  
}
```

11.6 XML与CSS

3. 字体

- 与字体有关的属性包括font-family、font-style、font-variant、font-weight、font-size

Name

```
{ display:list-item;  
  font-size:14pt;  
}
```

11.6 XML与CSS

4. 文本样式

- 与文本样式有关的属性包括text-align、text-indent、text-transform、text-decoration、vertical-align、line-height
- text-align设置文本的对齐方式
- text-indent设置文本首行的缩进量
- text-transform设置是否将文本中的字母全部大写、全部小写、首字母大写
- text-decoration设置是否将文本加划线
- vertical-align设置文本的垂直对齐方式
- Line-height设置文本间的间距

11.6 XML与CSS

5. 显示数学公式和化合物分子式

- 数学公式和化合物分子式中经常涉及字符的上标和下标。
XML文件通过与CSS样式表文件相关联，可以将某个字符的显示位置设置成另一个字符的上标或下标位置。
- 例子9_3中，JSP页面使用超链接请求一个XML文件，其中XML文件与CSS样式表文件相关联。

11.6 XML与CSS

type.jsp

<%@ page contentType="text/html;charset=GB2312" %>

<HTML><BODY bgcolor=yellow >

**
**显示几个数学公式和化合物分子式:

显示

</BODY></HTML>

11.6 XML与CSS

formula.xml

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

`<?xml-stylesheet href="show.css" type="text/css" ?>`

`<root>`

`<math>` 几个数学公式:

`<formula>`

平方和公式:

`(A+B)²=`

`A²+2AB+B²`

`</formula>`

`<formula>`

立方和公式:

`(A+B)³=`

`³+3A²B+3AB²+B³`

`</sup>`

`</formula>`

`</math>`

11.6 XML与CSS

formula.xml

<chemistry>

几个化合物分子式：

<molecular>

水的分子式：

H<low>2</low>O

</molecular>

<molecular>

二氧化硫的分子式：

SO<low>2</low>

</molecular>

<molecular>

碳酸的分子式：

H<low>2</low>CO<low>3</low>

</molecular>

</chemistry>

</root>

11.6 XML与CSS

show.css

math

```
{ display:block;
  background-color:yellow;
  color:green;
  left=100;
}
```

chemistry

```
{ display:block;
  background-color:cyan;
  color:green;
  left=100;
}
```

formula

```
{ display:list-item;
  list-style-type:lower-roman;
  margin-left:60;
  font-size:14pt;
  color:black;
}
```

11.6 XML与CSS

show.css

molecular

```
{ display:list-item;  
  list-style-type:decimal;  
  margin-left:60;  
  font-size:14pt;  
  color:black;  
}
```

sup

```
{ display:line;  
  font-size:10pt;  
  font-weight:bold;  
  font-style:italic;  
  color:blue;  
  vertical-align:super;  
}
```

low

```
{ display:line;  
  font-size:8pt;  
  font-weight:bold;  
  color:blue;  
  vertical-align:bottom;  
}
```

11.6 XML与CSS

- 通过将XML文件与CSS层叠样式表关联，可把数据的显示和数据结构相分离。XML只关心数据的结构，数据的显示外观由CSS控制。
- 比如，如果准备用**12pt**大小的文字来显示数学公式中的下标字符，只需修改下面的**CSS**层叠样式表文件show.css中**low**文本代表中的内容即可。
- 另一方面，如果在XML中再增加若干<formula>标记，也无须修改show.css文件。

小结

- XML文件是由标记构成的文本文件。XML文件有且仅有一个根标记，其他标记都必须封装在根标记中。文件的标记必须是树型结构，非空标记必须由“开始标记”与“结束标记”组成，空标记没有“开始标记”和“结束标记”。
- DOM解析器在内存中按树型结构组织数据，DOM解析器通过读入XML文件在内存中建立一棵“树”，XML文件的标记、标记的文本内容都会和内存中“树”的某个节点相对应。
- SAX解析器根据从文件中解析出的数据产生相应的事件，并报告这个事件给事件处理器，事件处理器就会处理所发现的数据。
- 通过将XML文件和一个CSS样式表文件相关联，可以方便地显示XML文件中的标记所含有的文本。