文章编号: 1001－9081(2004)03－0103－03

运用 XML 和 XSLT 技术实现 Web 页面的重用

李 伟, 郑 宁
(杭州电子工业学院 计算机学院, 浙江 杭州 310037)
(waylee@163.com)

摘 要: 从分析页面结构特征入手, 运用 XML 和 XSLT 技术实现 Web 页面的重用。该方法用动态 XML 文件定义页面的组成, 用 XSLT 文件定义页面布局, 通过 XSLT 转化把二者统一起来得到最终的页面。页面布局与页面组成的分离实现了页面布局和页面组成块的重用。

关键词: 动态 XML; XSLT; 页面布局
中图分类号: TP311.11 文献标识码: A

Reuse of Web Pages by XML and XSLT Technology

LI Wei, ZHENG Ning
(*School of Computer Science, Hangzhou Institute of Electronics Engineering, Hangzhou Zhejiang* 310037, *China*)

Abstract: After analyzing the features of Web page structure, a method of reusing Web pages by XML and XSLT was presented. It defined the page composition in dynamic XML files and defined the page layout in XSLT files. Then it created the pages by integrating XML files with XSLT files. BY seperation of page composition and page layout, simple blocks of Web pages can be reused.

Key words: dynamic XML; XSLT; layout

保持页面风格的一致性很容易带来页面代码的重复, 这就会大大增加开发和维护的难度。解决重复性的一个很好的方法就是重用, 但是现在人们对 Web 重用的讨论主要集中在后台的业务组件方面, 讨论的多是 Java Bean、COM 等组件的重用, 而对 Web 页面的重用重视不够。本文运用 XML 和 XSLT 技术实现了 Web 页面的重用, 既可减少页面代码的重复, 又可保证页面风格的一致性。

1 页面结构分析

虽然 Web 页面风格变化多端、内容各不相同, 但是它们的组成结构却具有一定的相似性。比如 Web 页面通常由标题、导航菜单、正文、页脚等部分组成, 每一部分是页面中的一个矩形区域, 称为“块”, 这些块可以由更小的子块组成, 如“正文”可能由“内容 1”、“内容 2”、…、“内容 n”等部分组成。组成页面的这些块之间的关系可以用一棵树来表示, 如图 1 所示。

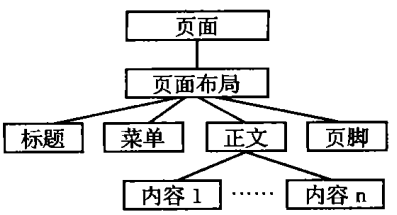

图 1 页面结构关系图

叶子节点是组成页面的基本单元, 称为“简单块”。中间节点描述的是简单块在较大块或最终页面中的位置, 称为“页面布局”。页面布局定义了页面的结构特征, 如页面有哪几类简单块组成, 这些简单块之间的位置关系如何等等。而一个页面具体有哪几个简单块组成, 称为“页面组成”。根结点即页面本身, 它包含页面的布局和组成信息。在 Web 应用中, 不同页面间的相似性正是由页面布局和简单块的相似性决定的, 页面代码的重复也正是页面布局和简单块的重复引起的。因此只要实现了页面布局和简单块的重用也就实现了页面的重用。但是, 现在页面布局和页面组成往往混杂在一起, 所以实现页面重用的关键是把页面布局和页面组成分离开来。

2 基于 XML 和 XSLT 技术的 Web 页面重用

2.1 基本思想

本文把简单块从页面中分离出来用独立的文件表示, 这些文件称为“块文件”。在原来的页面中, 用块名来指代分离出来的块就得到页面布局, 它是通过一个 XSLT 文件实现的, 称为“页面布局定义文件”。最后, 通过一个 XML 文件把页面布局定义文件中的块名和具体的块文件对应起来——即实现页面的组成定义, 该 XML 文件成为“页面组成定义文件”。页面组成定义文件是通过动态 XML 技术来实现的, 所谓动态 XML 就是通过活动页面(JSP/ASP)定义的 XML 文件, 它是符合 XML 规范的 JSP 或 ASP 文件。动态 XML 文件首先要经过活动页面解析引擎解析才能得到转换所需要的 XML 文件, 对于 JSP 解析引擎可以用 Tomcat 等, 而对于 ASP 一般采用微软的 IIS。通过 XSLT 转换就把页面布局定义文件和页面组成定义文件结合起来得到最终的页面。该模式的核心就是 XML 的 XSLT 转换。

2.2 工作过程

当请求某个页面时, 活动页面的解析引擎首先把表示页

收稿日期: 2003－08－12; 修订日期: 2003－10－15

作者简介: 李伟(1979－), 男, 山东高青人, 硕士研究生, 主要研究方向: 电子商务、企业信息系统; 郑宁(1961－), 男, 教授, 博士生导师, 主要研究方向: ICCAD、电信网管、电子商务.

面组成定义的动态 XML 转换为具体的 XML 文件，然后用指定的 XSLT 文件对该 XML 文件进行转换就得到所要求的 HTML 页面。这一过程如图 2 所示：

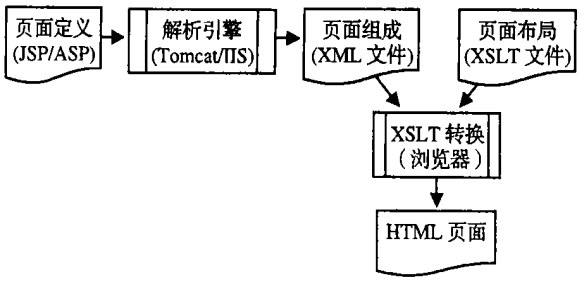

图 2 工作过程

通过同一个页面布局定义 XSLT 文件转换得到的 HTML 页面具有相同的页面风格，这样就实现了页面布局的重用，进而也达到了页面风格的一致性。同样，组成页面的简单块文件也实现了重用。这里的转换可以在服务器端进行，也可以在浏览器端进行。

2.3 简单实例

存在两个页面 A、B，它们的标题和内容不同，而页脚、菜单和页面布局等都是相同的。下面具体说明如何实现页面布局定义文件和页面组成定义文件以及具体的转化过程。

1）页面布局定义文件(layout.xsl)

页面布局定义文件是一个 XSL 转换文件，它定义了页面的布局，下面给出主要代码片段：

```
... ...
<xsl:template match="page">
  <html>
    <head><title><xsl:value-of select="header"/>
    </title></head>
    <body><table width="100%" border="1">
      <tr><td width="25%">
        <xsl:value-of select="menu"/></td>
        <td width="75%"><xsl:value-of select="body"/>
        </td>
      </tr>
      <tr>
        <td width="100%" colspan="2">
        <xsl:value-of select="footer"/></td>
      </tr>
    </table></body>
  </html>
</xsl:template>
... ...
```

这里通过一个 XSLT 模板定义了页面 A 和页面 B 是由 header、menu、body 和 footer 四类简单块组成的，并结合 HTML 标签给出了页面的布局定义。

2）页面组成定义文件——页面 A(pageA.jsp)

页面组成定义文件定义了页面布局中块名所对应的具体块文件，下面给出页面 A 的主要代码片段：

```
... ...
<?xml-stylesheet type="text/xsl" href="layout/layout.xsl"?>
<page>
  <header> This is Page A !</header>
  <menu><jsp:include page = "layout/menu.jsp"/></menu>
  <body><jsp:include page="/layout/abody.jsp"/></body>
  <footer><jsp:include page="/layout/footer.jsp"/></footer>
</page>
... ...
```

pageA.jsp 定义了简单块 header、menu、body 和 footer 所对应的具体块文件分别为 menu.jsp、abody.jsp 和 footer.jsp，这些都是预先定义好的简单块。只要把<head>和<body>两个标签的内容修改为 pageB 所对应的内容就可以得到页面 B 的组成定义文件。

3）解析、转换

当请求页面 A 时，解析引擎 Tomcat 会把各个块的内容填到页面组成定义文件(pageA.jsp)中形成最终的页面内容。这个文件被传到浏览器端，接下来浏览器会调用相应的 XSLT 文件(layout/layout.xsl)对该 XML 文件进行转换。最后，浏览器会把转换得到的 HTML 页面呈现给用户。

只需要把解析引擎改为 IIS，再把页面组成定义文件和相应的块文件改为 ASP 形式的定义就可以实现 ASP 页面的重用，而 XSLT 转换文件无需更改。这样就实现了页面布局文件在不同开发平台间的重用。

3 页面集中定义方式

在上面的应用中，每个页面都要有独立的页面进行组成定义，这虽然实现了页面的重用，但是没有减少页面数，也就没有降低应用的复杂度。为此，把页面组成定义都集中到同一个动态 XML 文件中。这样不但减少了页面数，降低系统的复杂度，而且实现了集中管理。为了说明方便，把这里的扩充方法命名为“集中定义方式”，而把最初的方法命名为“独立定义方式”。

3.1 基本思想

在独立定义方式中，每个页面实际上是由很多块组成的一棵树，把这些树看作一个分支组成一棵新的树就得到了集中定义方式的动态 XML 文件。在处理过程中，集中定义方式首先在页面集中定义 XML 文件中找到指定的节点，然后再进行转换。转换过程与独立定义方式是相同的。集中定义方式的核心是页面定义处理器，它负责页面组成定义节点的查找和转换。页面定义器可以定义在很多地方，比如 JSP/ASP 文件、Servlet、Struts 的 Action 中等。

3.2 工作过程

当请求页面 A 时，页面定义处理器就会在页面集中定义文件中查找页面 A 的定义，然后调入页面 A 的各个组成块，再把形成的页面返回给用户，如图 3 所示。

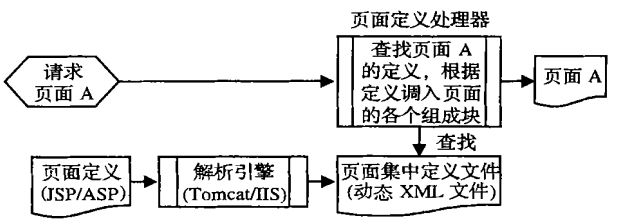

图 3 页面集中定义方式的工作过程

3.3 简单实例

还是以上面的例子进行说明。JSP 的解析引擎还是选用 Tomcat，页面布局文件保持不变，还是文件 layout.xsl。下面主要说明集中定义文件和页面定义处理器的定义：

1）页面组成集中定义文件(tiles-defs.jsp)

```
... ...
<pages-definitions>
  <page name="pageA">
    <header> This is Page A !</header>
    <menu><jsp:include page = "layout/menu.jsp" />
    </menu>
```

```
        <body><jsp:include page="/layout/abody.jsp"/>
        </body>
        <footer><jsp:include page="/layout/footer.jsp"/>
        </footer>
    </page>
    <page name="pageB" extends="pageA">
        <header>This is Page B!</header>
        <body><jsp:include page="/layout/bbody.jsp"/>
        </body>
    </page>
</pages-definitions>
... ...
```

页面 A 和页面 B 都是在 tiles-defs.jsp 中定义的,这就减少了页面数。注意页面 B 的定义,在 page 节点中增加了“extends”属性,它表明页面 B 的定义“继承”了页面 A 的定义。这样,页面 B 中与页面 A 相同的块就不用重新定义,从而简化了页面组成的定义。对于页面 A 中不同于页面 B 的块,只要进行重新定义就可以覆盖掉页面 A 中的定义,这里页面 B 覆盖了页面 A 的 header 和 body 块。

2)页面定义处理器的定义

在本文中页面定义处理器是在 Servlet 中定义的,页面集中定义文件和页面布局文件是作为该 Servlet 的参数在 web.xml 文件中设置的。

首先从 Web 请求中获取所请求页面的名称,根据页面名称在页面集中定义文件中找到该页面的组成定义节点(page 节点)。接下来看该 page 节点是否有 extends 属性:如果有,把其父节点的内容加进来,同时覆盖掉父节点中子节点已有的内容,得到最终的页面组成定义;如果没有,则可以直接进行下一步。最后,根据页面布局文件对页面组成定义节点进行转换得到最终的页面,把该页面返回给发出该请求的浏览器。这一过程如图 4 所示。

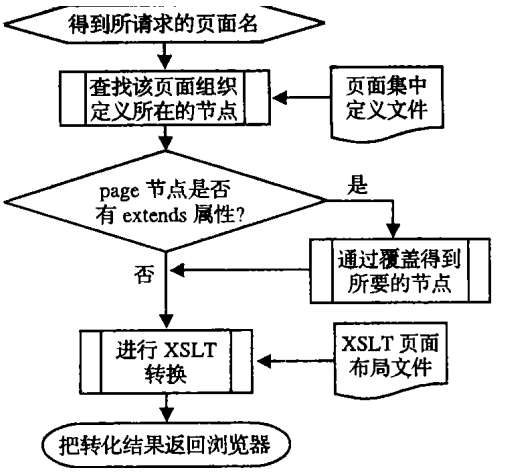

图 4　页面定义处理器的工作流程

3)使用

用户通过在请求该页面处理器 Serlvet 的 url 中加入参数 pageName,并把所请求页面的名字赋给它就可以实现对页面的访问。比如请求页面 A 可以通过以下形式进行:

http://localhost/servletName?pageName=pageA

这里的 servletName 就是实现页面定义处理器功能的 servlet 的名字。

参考文献

[1] Powell TA. Web 设计大全[M]. 北京:机械工业出版社, 2001.

[2] Malani P. UI design with Tiles and Struts[EB/OL]. http://www.javaworld.com/javaworld/jw-01-2002/jw-0104-tilestrut p.html, 2002-01-04.

[3] Cagle K, 等. XSL 高级编程[M]. 北京:机械工业出版社, 2002.

[4] Cavaness C. Programming Jakarta Struts[M]. O'Reilly, 2002.

(上接第 102 页)

题,组装应用和协调互作用;可以处理程序任务、监控活动、建立合约、执行应用过程和寻找最佳服务结果。另外,Agent 采用最新的基于 Web 技术,如 Java、XML 和 HTTP,易于使用,较易处理普适性、异构性、与平台无关性方面的问题。多 Agent 系统(MAS)能够在多个 Agent 之间互换谈判资料或共享信息,特别适合处理复杂问题。因此,采用多 Agent 系统解决基于 Internet 计算平台的构件组装是未来的一个发展趋势。

参考文献

[1] Nierstrasz O, Meijler TD. Research Directions in Software Composition[J]. ACM Computing Surveys, 1995, 27(2):262-264.

[2] Cerqueira R, Cassino C, Ierusalimschy R. Dynamic Component Gluing Across Different Componentware Systems[A]. DOA'99 - International Symposium on Distributed Objects and Applications[C]. IEEE Computer Society, Edinburgh, Scotland, 1999. 362-371.

[3] Mili H, Mili A, Yacoub S, *et al*. Reuse-Based Software Engineering: Techniques, Organization, and Control[M]. John Wiley & Sons, Inc, 2002.

[4] Yakimovich D, Travassos GH, Basili VR. A Classification of Software Components Incompatibilities for COTS Integration[A]. Proceedings of the 24th Software Engineering Workshop[C]. NASA/Goddard Space Flight Center, 1999.

[5] Zaremski AM, Wing JM. Specification Matching of Software Components[J]. ACM Transactions on Software Engineering and Methodology, 1997, 6(4):333-369.

[6] Ye Y. Supporting Component-Based Software Development with Active Component Repository Systems[D]. University of Colorado, 2001.

[7] Eskelin P. Component Interaction Patterns[A]. Proceedings of Pattern Languages of Program[C]. 1999.

[8] Garlan D, Allen R, Ockerbloom J. Architectural Mismatch: Why Reuse Is So Hard[J]. IEEE Software, 1995, 12(6):17-26.

[9] Gacek C. Detecting Architectural Mismatches During Systems Composition[D]. University of Southern California, 1998.

[10] Deline R. A Catalog of Techniques for Resolving Packaging Mismatch[A]. Symposium on Software Reusability[C]. Los Angeles, CA, 1999.

[11] Vanderperren W, Wydaeghe B. Towards a New Component Composition Process[A]. Proceedings of the Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems[C]. 2001.

[12] McInnis K. Component-based Development—The Concepts, Technology and Methodology[EB/OL]. http://www.cbd-hq.com, 2003-05-08.

[13] Mili H, Mili A, Yacoub S, *et al*. Reuse-Based Software Engineering: Techniques, Organization, and Control[M]. John Wiley & Sons, Inc, 2002.