Developing XML Solutions with JavaServer Pages Technology

XML (eXtensible Markup Language) is a set of syntax rules and guidelines for defining text-based markup languages. XML languages have a number of uses including:

- Exchanging information
- Defining document types
- Specifying messages

Information that is expressed in a structured, text-based format can easily be transmitted between, transformed, and interpreted by entities that understand the structure. In this way XML brings the same cross-platform benefits to information exchange as the Java^{TM} programming language has for processing.

JavaServer Pages^{TM} (JSP^{TM}) technology provides specification and serving of documents that combine static markup language elements and elements created dynamically by Java programming language objects. JSP pages execute as Java servlets, typically in the execution context of a Web server where they generate content-dependent response documents using data stored in databases and in other server-based application objects. JSP technology is a Java technology with specifications developed with broad industry participation through the Java Community Process (JCP). Application servers and development tools that support JSP technology are available from multiple vendors (including open source groups) for a broad range of hardware-OS platforms.

JSP technology provides a number of capabilities that are ideally suited for working with XML. JSP pages can contain any type of text-based data, so it is straightforward to generate documents that contain XML markup. Also, JSP pages can use the full power of the Java platform to access programming language objects to parse and transform XML messages and documents. In particular, as part of the Java software environment, JSP pages can use objects that leverage the new Java APIs for processing XML data. Finally JSP technology provides an abstraction mechanism to encapsulate functionality for ease of use within a JSP page.

This paper highlights how JSP pages can

- Consume XML data
- Generate XML-based markup languages for various types of Web-based clients

The paper assumes that you are familiar with JSP technology, including JavaBeans components and custom tag libraries. For information on these topics, consult the resource and technical resource areas on the JSP technology Web sites.

Using XML Data Sources in JSP Pages

It is easy to use multiple data sources, including XML sources, in a JSP page; Figure 1 illustrates the standard way to do so. A page connects to a data source through a server-side object, transforms the information into data abstractions, and then renders the data using JSP elements.

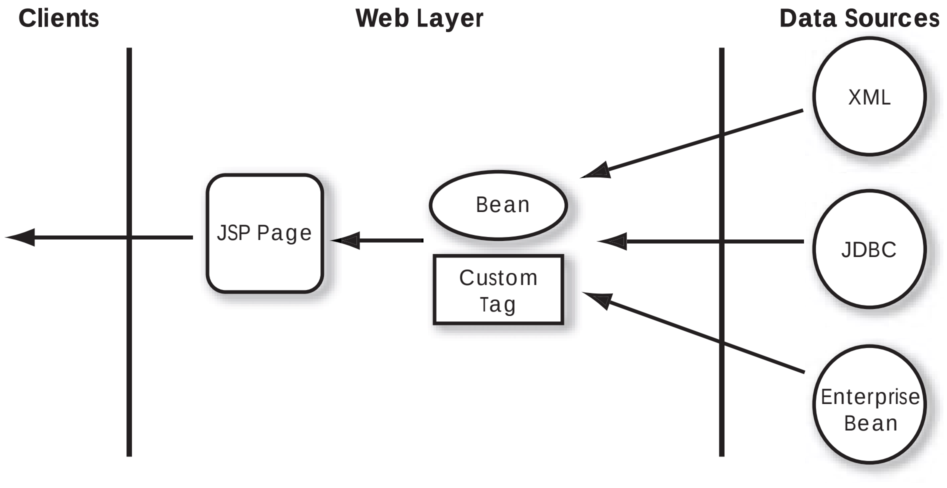


Figure 1 Accessing Heterogeneous Data Sources From a JSP Page

Consider the following XML representation of a book inventory:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<books>
   <book isbn="123">
       <title>Web Servers for Fun and Profit</title>
       <quantity>10</quantity>
       <price>$17.95</price>
   </book>
   <book isbn="456">
       <title>Web Programming for Mobile Devices</title>
       <quantity>2</quantity>
       <price>$38.95</price>
   </book>
   <book isbn="789">
       <title>Duke: A Biography of the Java Evangelist</title>
       <quantity>25</quantity>
       <price>$15.00</price>
   </book>
</books>
```

Example 1 XML Document Representing a Book Inventory

There are two ways you could use this XML data in a JSP page:

- Convert the XML elements into server-side objects and then extract the object properties.
- Invoke a transformation on the XML data.

Convert XML to Server-Side Objects and Extract Object Properties

In the first approach, objects are created to represent the information carried by the XML document. Currently you have to write code, using DOM or SAX and encapsulated into a custom tag or a JavaBeans component, to create these objects. In the future, XML/Java Binding technology (JSR 31 in the Java Community Process) will automate this process because it will enable you to compile an XML schema into Java classes.

In the following example, a JSP page retrieves XML data from two URLs and uses the data to generate an HTML page. Note that the URL can point to a source that dynamically generates XML data (See Generating XML From a JSP Page (page 9)) as well as a static XML document, and the same technique can be used to generate XML.

The JSP page uses the `parse` custom tag to extract and store XML data in two objects: a customer and a collection of books. The page then extracts properties of the customer object and uses a custom tag to iterate through the collection and render a table that lists properties of the book objects.

```
<%@ taglib uri="..." prefix="tl" %>
<html>
<tl:parse id="customer" type="Customer"
   xml="XML_Customer_URL"/>
<tl:parse id="saleBooks" type="BookInventory"
   xml="XML_Book_Inventory_URL"/>
<head>
<title>Welcome</title>
</head>
<body>
Welcome&nbsp;
<jsp:getProperty name="customer" property="lastName"/>
&nbsp;
<jsp:getProperty name="customer" property="firstName"/>
<table border="0" width="50%">
<tl:iterate id ="book" type=”Book”
   collection=”<%= saleBooks.getBooks() %>” >
<tr>
   <td>
       <jsp:getProperty name="book"
           property="title"/>
   </td>
   <td>
       <jsp:getProperty name="book"
           property="price"/>
   </td>
```

```
</tr>
</tl:iterate>
</table>
</body>
</html>
```

Example 2 Converting XML Data to Server-Side Objects

One of the XML documents in this example (the book inventory data) is composed of XML fragments (`book` elements). It is not really necessary to parse the whole document into memory in order to extract the data—you can parse it, using SAX or JDOM, for example, and extract data from one fragment at a time. The following version of the example uses the `iterateOnXMLStream` custom tag to implement this alternative:

```
<%@ taglib uri="..." prefix="tl" %>
<html>
<tl:parse id="customer" type="Customer"
   xml="XML_Customer_URL"/>
<head>
<title>Welcome</title>
<body>
   as above ...
<table border="0" width="50%">
<tl:iterateOnXMLStream id="book" type="Book"
   xml="XML_Book_Inventory_URL">
<tr>
   as above ...
</tr>
</tl:iterateOnXMLStream>
</table>
</body>
</html>
```

Example 3 Iteratively Converting XML Elements to Server-Side Objects

Convert XML Using An XSLT Transformation

Another way to use XML data in a JSP page is to apply a transformation on the XML data source, either to extract the data, or to create a new format. This transformation can be done using a number of different mechanisms and accessed through custom tags.

XSLT is a transformational language standardized in W3C that can be used to transform XML data to HTML, PDF, or another XML format. For example, you can use XSLT to convert an XML document in a format used by one company to the format used by another company.

To generate the HTML page discussed in the previous section using this approach you need an XSL stylesheet and a way to apply the stylesheet.

The XSL stylesheet in Example 4 performs the required transformation for `books` and `customer` elements. The stylesheet generates HTML markup and extracts data from the elements.

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" omit-xml-declaration="yes"/>
<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>
<xsl:template match="books">
<table border="0" width="50%">
   <xsl:for-each select="book">
   <tr>
       <td>
           <i><xsl:value-of select="title"/></i>
       </td>
       <td>
           <xsl:value-of select="price"/>
       </td>
   </tr>
   </xsl:for-each>
</table>
</xsl:template>
```

```
<xsl:template match="customer">
<b><xsl:value-of select="first_name"/>&#160;
   <xsl:value-of select="last_name"/></b>
</xsl:template>
</xsl:stylesheet>
```

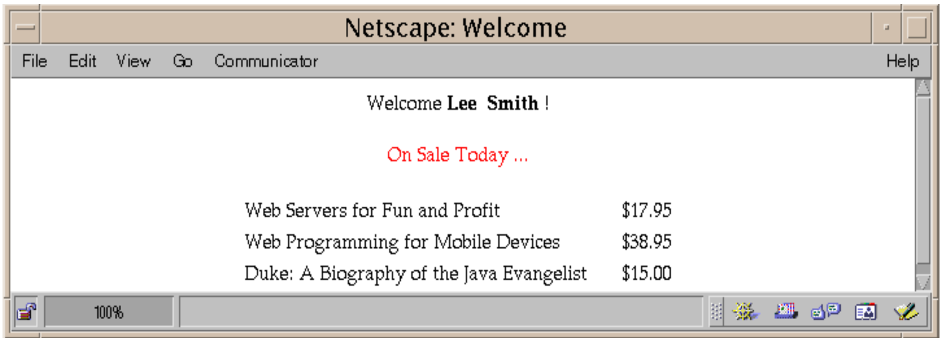
Example 4 `store.xsl`

To apply the stylesheet, you programmatically invoke an XSLT processor from a JSP scriptlet or custom tag. The `jakarta-taglibs` project at the Apache Software Foundation hosts a tag library that contains a number of tags for processing XML input and applying XSLT transformations. The JSP page in Example 5 invokes the `apply` tag from this library with `store.xsl` to transform customer data and the book inventory.

```
<%@taglib uri="http://jakarta.apache.org/taglibs/xsl-1.0"
   prefix="xsltlib" %>
<html>
<head>
<title>Book Inventory</title>
</head>
<body bgcolor="white">
<center>
Welcome&nbsp;
<xsltlib:apply xml="XML_Customer_URL" xsl="store.xsl"/>!
<p></p>
<font color="red">
On Sale Today ...
</font>
<p></p>
<xsltlib:apply xml="XML_Book_Inventory_URL" xsl="store.xsl"/>
</center>
</body>
</html>
```

Example 5 Applying an XSL Stylesheet in a JSP Page

The result of executing this JSP page appears below.



Welcome **Lee Smith** !

On Sale Today ...

Web Servers for Fun and Profit	$17.95
Web Programming for Mobile Devices	$38.95
Duke: A Biography of the Java Evangelist	$15.00

Generating Markup Languages Using JSP Pages

In order for Web-based services to have the broadest use it has become increasingly important that they be accessible from the widest variety of clients. Recent years have witnessed a proliferation of different types of user-oriented clients for Web-based applications: PDAs, WAP-enabled mobile phones, and landline voice clients. With the rise of business-to-business transactions, servers that consume XML can also be clients. The scenario we envision supporting looks like the following:

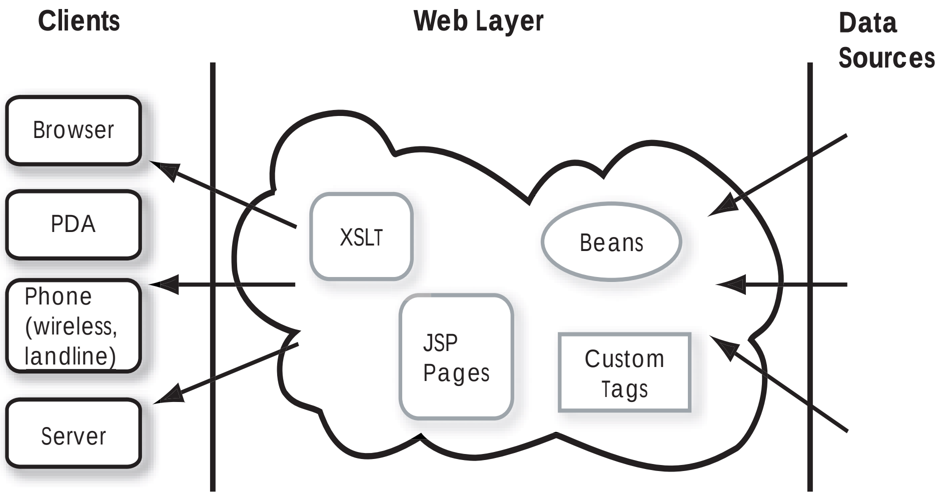


Figure 2 Web Application With Heterogeneous Clients

These clients speak the following languages:

Client	**Markup Language**
PC-based browser	HTML, DHTML, XHTML
PDA	WML, XHTML
mobile phone	WML, XHTML
landline phone	VoiceXML
server	application-specific XML languages

The effort required to support all of these clients will dwarf that expended on HTML applications during the 1990s. Fortunately better tools are available now: JSP technology and XML. Next we describe how to use these tools to develop multilingual Web applications.

Generating XML From a JSP Page

We have already seen how JSP pages can consume XML data to generate dynamic content. JSP pages can also generate XML data. Recall the XML document introduced at the beginning of this paper:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<books>
   <book isbn="123">
       <title>Web Servers for Fun and Profit</title>
       <quantity>10</quantity>
       <price>$17.95</price>
   </book>
   ...
</books>
```

A JSP page could generate a response containing this document to satisfy a business-to-business request for the contents of a book warehouse. The consumer examples described earlier could also use this page as a source of XML data.

The main requirement for generating XML is that the JSP page set the content type of the page appropriately:

```
<%@ page contentType="text/xml"%>
... XML document
```

With this change, the techniques described earlier to generate HTML content can be used to generate XML.

Generating Multiple Markup Languages

There are several approaches to generating multiple markup languages from a Web application:

- Single pipeline
- Multiple pipeline
- Combination pipeline

Single Pipeline

In the single pipeline approach, the Web application's JSP pages generate client-specific markup by applying transformations to incoming XML data. Each type of client requires a different stylesheet and the bulk of the development costs are associated with creating and maintaining these stylesheets.

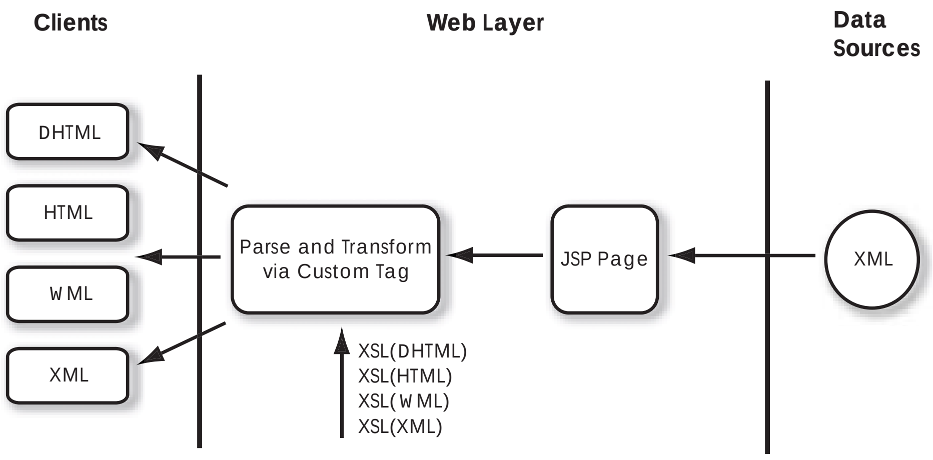


Figure 3 Single Pipeline Generating Multiple Markup Languages

This approach defers generation of both the static and dynamic portions of a response to runtime. The runtime costs are associated with:

- Parsing the XML data
- Parsing the stylesheet
- Applying the transformation

If you are using XSLT transformations, you can improve the performance of transformations by creating a binary representation (using an XSLT compiler, for example, XSLTC) of a stylesheet. However, this also makes the maintenance process more complex: each time the presentation is changed, the stylesheet must be recompiled.

Sometimes the differences between clients are minor and may not merit a full-fledged transformation. For example, there are a number of slightly different browser-based desktop clients. In some cases, one may want to take full advantage of the differences instead of generating content for the minimum common denominator. Often the differences between these clients can be encapsulated into a custom tag that generates different content depending on the properties of the client.

Generating the presentation for clients with different interaction models and flow of control (for example, PC-based browsers versus WAP phones) will require very different transformations. For example, a mobile phone cannot display a table containing book inventory data. Instead the data would have to be displayed as a set of nested lists. Supporting such transformations increases both the development and runtime costs.

Multiple Pipeline

The multiple pipeline approach uses a set of client-specific JSP pages to generate output.

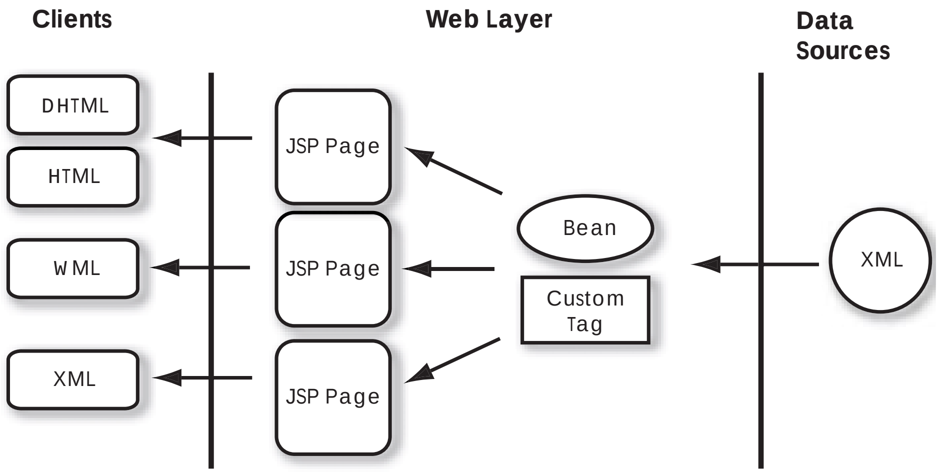


Figure 4 Multiple Pipelines Generating Multiple Markup Languages

As compared with using XSLT transformations, this approach keeps the work of creating the static content in the development phase with the dynamic content generation occurring at runtime.

Aside from creating the client-specific JSP pages, development costs are incurred in creating and maintaining server-side objects that represent the application's data abstractions. This step is not required in the single pipeline approach. Nevertheless the multiple pipeline approach can be more cost effective than the single pipeline for the following reasons:

- Data abstractions can be reused by different kinds of JSP pages.
- Data abstractions typically change at a much lower rate than presentation.
- Executing a JSP page to generate markup is much more efficient than performing an XSLT transformation to generate the same markup.

The following table summarizes the costs of each approach:

Pipeline	**Development**	**Runtime**
Single	Client-specific stylesheets	Parse XML data Parse stylesheet Apply transformation
Multiple	Data abstractions Client-specific JSP pages	Parse XML data Instantiate and initialize data abstraction components Execute JSP page

Combination

You can combine the single and multiple pipeline approaches. If your clients speak different languages you probably should use one pipeline for each language. To generate dialects of a language, you can apply XSLT transformations to that language's pipeline.

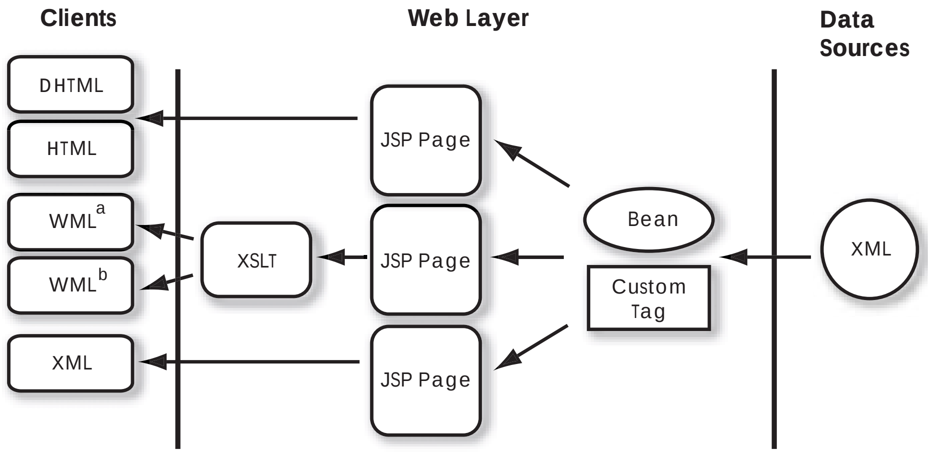


Figure 5 Combination Pipeline Generating Multiple Markup Languages

Conclusion

JavaServer Pages technology and XML are natural partners for developing Web applications that use heterogeneous data sources and support multilingual clients. This paper has described several approaches to addressing these requirements that trade off development and maintenance versus runtime costs. As part of the general-purpose Java software platform, JSP technology provides developers with the capability to evaluate these trade-offs and to adopt an architecture for XML processing that best satisfies their particular application requirements.

Resources

For further information about the technologies described in this paper, see the following resources:

- `http://java.sun.com/products/jsp` - JSP technology
- `http://java.sun.com/products/servlet` - Java Servlet technology
- `http://java.sun.com/xml` - Java technology and XML
- `http://www.sun.com/xml` - XML at SUN
- `http://jakarta.apache.org/taglibs` - Jakarta Taglibs