

Совместное использование XML и JSP

Два замечательных вкусовых оттенка вместе дают замечательный букет.

Server-Side Java

Алекс Чаффи

XML и JSP – два уже навязших в зубах слова. В статье показывается как использовать две эти технологии совместно, создавая динамический Web-сайт. Вы также увидите примеры программного кода использующего DOM, Xpath, XSL и другие Java-технологии. *(3000 слов)*

Я буду считать, что как большинство Java-программистов, вы знаете что такое JavaServer Pages (JSP) и Extensible Markup Language (XML), но вам не совсем ясно как их использовать. Объяснить необходимость использования JSP довольно легко. JSP позволяет вам конструировать Web-сайт из файлов, которые выглядят и исполняются весьма схоже HTML-файлам. Единственным отличием является то, что JSP-файлы могут исполняться динамически — например, они могут обрабатывать формы или читать базы данных — используя Java в качестве скриптового языка на стороне сервера. Оправдать необходимость использования XML несколько труднее. Хотя, как кажется, каждый новый программный продукт поддерживает XML, каждый из них, как кажется, использует XML для разных целей.

В статье вы узнаете как конструировать систему, используя XML достаточно умеренным образом. Многие Web- сайты хранят большое количество данных, которые визуализируются более или менее стандартным образом. Я сконструирую систему, которая использует XML-файлы, чтобы хранить данные на Web-сервере, а JSP-файлы, чтобы визуализировать эти данные.

1. XML против реляционных баз данных

"Погодите," вы спросите, "вы используете XML, чтобы хранить данные? Почему не базу данных?" Хороший вопрос. Ответ: для многих случаев база данных слишком громоздка. Чтобы использовать базу данных вам необходимо установить и поддерживать отдельный процесс на сервере, что часто требует также и установки и поддержания администратора базы данных. Вы должны изучить SQL, и писать SQL-запросы, которые конвертируют данные из реляционной в объектную структуру, и обратно. Если же вы храните XML-файлы, вам нет необходимости использовать подобную надстройку. Вы также получаете простую возможность редактировать данные, используя любой текстовый редактор вместо сложной оболочки базы данных. XML- файлы также легко восстанавливать, делиться ими с друзьями или загружать клиентам. Вы также можете легко обновлять их на вашем сайте, используя FTP.

Более абстрактным преимуществом XML является то, что будучи иерархичнее реляционного формата, XML может быть использован для конструирования структуры данных значительно более очевидным образом. Вам не нужно применять редактор отношений элементов или нормализовывать вашу схему. Если у вас один элемент содержит другой, вы можете отразить это непосредственно в формате, а не использовать совместную таблицу.

Отметим, однако, что для многих приложений использование лишь файловой системы будет недостаточно. Если у вас большая интенсивность обновлений данных, файловая система может сбоить и разрушаться при множестве одновременных попыток записи; базы данных обычно поддерживают транзакции, чтобы осуществлять параллельную запись данных. Кроме того, базы данных – это отличное средство, если вы выполняете сложные запросы, особенно, если эти запросы могут меняться время от времени. Базы данных строят индексы и изменяют их в соответствии с меняющимся набором данных. Реляционные базы данных имеют также множество других преимуществ, включая богатый язык запросов, обширные средства конструирования схем, проверенную масштабируемость, проработанный контроль доступа.

Замечание:

Вы можете использовать простую блокировку файлов, чтобы создать свой сервер транзакций для бедных. Вы можете также реализовать на Java средство XML индексации и поиска, но это тема другой статьи.

В нашем случае, как и в большинстве мелких и средних Web-сайтов предназначенных для публикаций, мы можем полагать следующее: большинство операций доступа к данным является чтением, а не записью; данные, хотя и обширны, меняются редко; нет необходимости в сложном поиске, а если такая необходимость и появляется, поиск выполняется отдельным поисковым приложением. В таком случае преимущества использования реляционной базы данных блекнут, а преимущества использования объектно-ориентированной модели данных расцветают.

Наконец, весьма возможно использовать преобразователь результатов SQL-запросов к вашей базе данных в XML-потоки, так что вы можете иметь результаты в обоих видах. Тогда XML становится более живучим, дружественным к программисту передним слоем "настоящей" базы данных, используемой для хранения и поиска. (Oracle's XSQL сервлет является примером такой техники.)

2. Приложение: онлайн-фотоальбом

Все любят фотографии! Люди любят показывать фотографии свои, своих друзей, своих домашних животных, мест, где они отдыхали. Web – идеальная среда для незадачливых трепачей. Так как полномасштабный сайт фотоальбомов потребует сложной объектной модели, я сосредоточусь на определении единственного объекта Picture. (Исходный код этого приложения см. в [Ресурсы](#).) В объекте описывающем фотографию должны быть поля с заголовком, датой съемки, изменяемой подписью и, конечно, указателем на изображение.

Изображение, в свою очередь, должно иметь свои поля: местоположение файла (GIF или JPEG), высота и ширина в отчетах (чтобы помочь вам при описании тэгов ``). Вот вам еще одно преимущество использования файловой системы в качестве базы данных: вы можете хранить файлы изображений в той же директории, что и файлы данных.

Наконец, дополним объект Picture элементом, описывающим набор иконок для использования, например, таблице содержания. Здесь я использую ту же концепцию image, что я определил ранее. XML-представление объекта Picture может выглядеть подобно следующему

```
<picture>
```

```
<title>Alex On The Beach</title>
<date>1999-08-08</date>
<caption>Trying in vain to get a tan</caption>
<image>
  <src>alex-beach.jpg</src>
  <width>340</width>
  <height>200</height>
</image>
<thumbnails>
  <image>
    <src>alex-beach-sm.jpg</src>
    <width>72</width>
    <height>72</height>
  </image>
  <image>
    <src>alex-beach-med.jpg</src>
    <width>150</width>
    <height>99</height>
  </image>
</thumbnails>
</picture>
```

Отметим, что используя XML, вы помещаете всю информацию о фотографии в один файл, а не разбрасываете ее по 3-4 отдельным таблицам. Назовем этот файл .pix – т.е. ваша файловая система может выглядеть подобно следующему:

```
summer99/alex-beach.pix
  summer99/alex-beach.jpg
  summer99/alex-beach-sm.jpg
  summer99/alex-beach-med.jpg
  summer99/alex-snorkeling.pix
  etc.
```

3. Техника программирования

Как существует несколько способов ободрать кошку, так имеется несколько способов донести XML-данные до вашей JSP-страницы. Вот перечень некоторых из этих способов. (Список не является исчерпывающим; множество других продуктов и спецификаций может быть использовано с одинаковым успехом).

- **DOM:** Вы можете использовать классы реализующие DOM-интерфейс, чтобы

Совместное использование XML и JSP

- просматривать и проверять XML-файл
- **XMLEntryList**: Вы можете использовать мой код, чтобы загрузить XML в список пар имя-значение `java.util.List`
- **XPath**: Вы можете использовать Xpath-процессор (подобно Resin), чтобы определить местоположение элементов в XML-файле по пути
- **XSL**: Вы можете использовать XSL-процессор, чтобы трансформировать XML в HTML
- **Cocoon**: Вы можете использовать открытый проект Cocoon framework
- **Roll your own bean**: Вы можете написать внешний класс, который использует один из других способов, чтобы загрузить данные в специальный `JavaBean`

Отметим, что эти способы в равной степени могут быть применены к любому XML-поток, который вы получаете из другого источника, например, от клиента или сервера приложений.

4. JavaServer Pages

Спецификация JSP имеет много воплощений, и различные JSP-продукты реализуют различные, несовместимые версии данной спецификации. Я буду использовать Tomcat по следующим причинам:

- поддерживает наиболее свежие версии спецификации JSP и сервлетов
- одобрен Sun и Apache
- может быть запущен без Web-сервера
- является открытым проектом

(Для получения подробной информации о Tomcat, см. [Ресурсы](#).)

Вы вольны использовать любой JSP-движок, но конфигурируйте его сами! Убедитесь, что он поддерживает хотя бы спецификацию JSP 1.0; имеется много отличий между 0.91 и 1.0. JSWDK (Java Server Web Development Kit) работает просто отлично.

5. Структура JSP

Создавая JSP-ведомый Web-сайт (известный также как Webapp), я предпочитаю помещать общие функции, объявления подключаемых пакетов, констант в отдельный файл [init.jsp](#). Затем я грузу этот файл в каждый JSP-файл, используя

`<%@include file="init.jsp"%>`. Директива `<%@include%>` работает подобно директиве `#include` языка C, помещая текст включаемого файла (здесь, `init.jsp`) и компилируя его, как если бы он был частью включающего файла (здесь, `picture.jsp`). Напротив, тэг `<jsp:include>` компилирует файл как отдельный JSP-файл и помещает его вызов в откомпилированный JSP-файл.

6. Нахождение файла

Когда JSP стартует, первое, что ему необходимо после инициализации – это найти требуемый вам XML-файл. Откуда он узнает какой из множества файлов вам требуется? Ответ: из параметра CGI. Пользователь запускает JSP-файл посредством задания URL `picture.jsp?file=summer99/alex-beach.pix` (или передавая параметр `file` через HTML-форму).

Однако, передать JSP параметр – значит сделать только полдела. Вам необходимо еще знать в каком месте файловой системы находится корневой каталог. Например, в системе Unix, файл может находиться в каталоге `/home/alex/public_html/pictures/summer99/alex-beach.pix`. JSP не имеет понятия текущего каталога исполнения, поэтому вам необходимо передать полный путь пакету `java.io`.

Servlet API предоставляет метод для превращения относительного URL для текущего JSP или сервлета, в полный путь файловой системы. Это делает метод `ServletContext.getRealPath(String)`. Каждый JSP имеет объект `ServletContext`, названный `application`, и код может выглядеть так: `String picturefile = application.getRealPath("/") + request.getParameter("file");`

или

```
String picturefile = getServletContext().getRealPath("/") +
request.getParameter("file");
```

Этот код также работает внутри сервлета. (Вы должны добавить `/`, Так как метод ожидает передачи результатов `request.getPathInfo()`.)

Предупреждение:

Так как вы предоставляете локальные ресурсы, очень внимательно контролируйте допустимость входных

данных. Хакер или невнимательный пользователь может послать вредоносные данные и взломать ваш сайт. Подумайте, к примеру, что произойдет, если будет задано значение `file=../../../../etc/passwd`. В этом случае пользователь сможет прочитать файл пароля вашего сервера.

7. Document Object Model

DOM является аббревиатурой *Document Object Model*. Это стандартный API для просмотра документов XML, разработанный World Wide Web Consortium (W3C). Интерфейсы находятся в пакете `org.w3c.dom`, а документация на сайте W3C (см. [Ресурсы](#)).

Существует много реализаций DOM-парсера. Я выбрал XML4J от IBM, но вы можете использовать любой DOM-парсер. Это возможно из-за того, что DOM является набором интерфейсов, а не классов — и все DOM-парсеры возвращают объекты, которые должным образом реализуют эти интерфейсы. К сожалению DOM имеет два основных недостатка:

1. API, хотя и объектно ориентирован, является довольно запутанным.
2. Не существует стандартного API для DOM-парсера, так что, хотя каждый парсер возвращает объект `org.w3c.dom.Document`, средства инициализации парсера и загрузки файла являются специфическими для каждого парсера.

Простой файл для фотографии, описанный выше представляется в DOM несколькими объектами в древовидной структуре.

```
Document Node
--> Element Node "picture"
    --> Text Node "\n " (whitespace)
    --> Element Node "title"
        --> Text Node "Alex On The Beach"
    --> Element Node "date"
        --> ... etc.
```

Чтобы получить значение Alex On The Beach вам необходимо выполнить несколько вызовов методов, гуляя по дереву DOM. Более того, парсер может разбросать в дереве некоторое количество текстовых узлов, состоящих из одних пробелов. Вы должны будете прокрутить эти узлы и, или не учитывать их, или добавить их в путь (вы можете поправить это вызвав метод `normalize()`). Парсер

может также включить отдельные узлы для объектов XML (например, `&`), узлы CDATA или другие элементы (к примеру, the `big` bear может иметь три узла, один из которых — элемент `b`, содержащий текстовый узел с текстом `big`). В DOM не существует метода с помощью которого можно было бы сказать "дай мне текстовое значение элемента с названием таким-то названием" В общем, прогулка по дереву DOM несколько утомительна. (См. Раздел XPath этой статьи посвященный альтернативе DOM.)

В общем, проблема с DOM заключается в том, что XML-объекты недоступны непосредственно в качестве Java- объектов. Доступ к ним обеспечивается "кусками" посредством DOM API. См. заключение к этой статье о технологии Java-XML Data Binding, которая обеспечивает Java-подход для доступа к данным XML. Я написал небольшой вспомогательный класс `DOMUtils`, который содержит `static`-методы для выполнения типовых задач DOM. Например, чтобы получить текстовое содержимое подэлемента `title` корневого элемента `picture`, вы должны написать следующий код:

```
Document doc = DOMUtils.xml4jParse(picturefile);
Element nodeRoot = doc.getDocumentElement();
Node nodeTitle = DOMUtils.getChild(nodeRoot, "title");
String title = (nodeTitle == null) ? null :
    DOMUtils.getTextValue(nodeTitle);
```

Получение значение подэлементов изображений также незамысловато:

```
Node nodeImage = DOMUtils.getChild(nodeRoot, "image");
Node nodeSrc = DOMUtils.getChild(nodeImage, "src");
String src = DOMUtils.getTextValue(nodeSrc);
```

И т.д.. Так как каждому значимому элементу соответствует Java-переменная, все, что вам нужно – это поместить эти переменные внутрь вашей HTML-разметки, используя стандартные тэги JSP.

```
<table bgcolor="#FFFFFF" border="0"
    cellspacing="0" cellpadding="5">
<tr>
<td align="center" valign="center">
"
    height="<%=height%>" border="0"
    alt="<%=src%>"></td>
</tr>
```

```
</table>
```

Для детального изучения см. полный код примера. Выходной HTML, формируемый JSP-файлом — если угодно HTML-стоп-кадр — лежит в файле [picture-dom.html](#).

8. Использование JSP beans для разделения model/view

Код в начале файла [picture-dom.jsp](#) выглядит непривлекательно. Хотя вы можете помещать сотни строк Java-кода внутрь JSP, существует более чистый подход: вы используете JSP JavaBean-ы, чтобы хранить там большие порции Java-кода, в то время, как тэги скриптата JSP (<% и %>) используются для контроля ветвей исполнения и минимальных манипуляций с переменными внутри страницы JSP.

При разработке прототипа проекта чаще бывает легче сначала поместить весь Java-код внутрь JSP. Когда же вы получаете более четкое представление о задаче, вы можете вернуться к написанному, извлечь из JSP фрагменты Java-кода и написать JavaBean-ы. В этом случае начальные трудозатраты больше, но затраты на сопровождение продукта, соответственно, меньше. Так как приложение становится более модульным. Вы можете использовать одни и те же JavaBean-ы на различных JSP-страницах, не прибегая к кошмару copy/paste кода.

В нашем случае явный кандидат для написания под него JSP JavaBean-а является код, который извлекает значения String из XML-файла. Вы можете определить классы Picture, Image и Thumbnails, описывающие основные элементы XML-файла. Эти bean-ы будут иметь конструкторы или методы setXXX, которые получают узел DOM или имя файла из которых нужно извлечь их значения. Вы можете посмотреть пакет picturebeans в исходном коде (см. [Ресурсы](#)) или файл picture-beans.jsp.

Просматривая исходный текст picture-beans.jsp, обратите внимание на следующее:

- Я объявил интерфейсы отдельно от реализации классами, так, что в будущем вы сможете поменять реализацию. Вы можете захотеть хранить значения в List, непосредственно в DOM или даже в базе данных.
- Bean-ы определены в пакете picturebeans. Все JSP bean-ы должны быть в каком-либо определенном разработчиком пакете, Так как большинство JSP-движков не способны найти классы в default пакете.

- Я описал не только `getXXX`, но и `setXXX` методы. Хотя сейчас вы только считываете информацию, в дальнейшем вы можете захотеть позволить пользователям редактировать фотографии, так, что вам необходимо запланировать возможность изменения свойств `bean`-ов.
- Я должен теперь использовать `<%=picture.getCaption() %>` вместо простого `<%=caption%>`, Так как значения хранятся в `bean`-е, а не в локальных переменных. Однако, если вы хотите, вы можете объявить локальные переменные подобно `String caption = picture.getCaption();`. Это допустимо, Так как делает код несколько более читабельным.

9. Увеличение с использованием thumbnail-изображений

Вы, наверное, заметили, что выход моего первого JSP, `picture-dom.html`, использует файл изображения в реальном масштабе. Изменим немного код так, чтобы вместо изображения в реальном масштабе, демонстрировалась уменьшенная (`thumbnail`) версия. Я буду использовать список `thumbnail`-изображений, сохраняемых в XML-файле.

Определим параметр `zoom`, величина которого определяет какое из `thumbnail`-изображений визуализировать. Щелчок мыши по `thumbnail`-изображению выведет изображение в реальном масштабе; щелчок по кнопкам `Zoom In` или `Zoom Out` выведет следующее или предыдущее `thumbnail`-изображение из списка.

Так как объект `Thumbnails` возвращает список `java.util.List` объектов `Image`, нахождение нужного `thumbnail`-изображения весьма несложно: `(Image)picture.getThumbnails().get(i)`.

Чтобы создать связи `Zoom In` и `Zoom Out`, вам необходимо создать рекурсивную ссылку на ту же страницу, но с различными параметрами. Для этого вы используете метод `request.getRequestURI()`. Это дает вам только путь к сервлету без параметров так, что необходимые параметры вы должны задавать сами.

```
<%
if (zoom < (thumbnails.size() -1)) {
    out.print("<a href='" +
                request.getRequestURI() +
```

```
        "?file=" + request.getParameter("file") +  
        "&zoom=" + (zoom+1) +  
        "'>");  
    out.print("Zoom In</a>");  
}  
%>
```

Далее показан [кадр](#) HTML работающей страницы JSP.

10. Использование тэгов JSP bean

Спецификация JSP определяет тэг `<jsp:useBean>` для автоматической инициализации и использования JavaBean-ов из страницы JSP. Тэг `useBean` всегда может быть заменен Java-кодом, что я и сделал здесь. По этой причине многие люди сомневаются в необходимости тэгов `useBean` и `setProperty`. Аргументы в пользу этих тэгов таковы:

- Синтаксис, основанный на тэгах выглядит менее устрашающим для HTML-дизайнеров.
- Тэг `useBean` имеет параметр `scope`, который автоматически определяет должен bean храниться в локальной переменной, переменной сессии или атрибуте приложения.
- Если переменная является `persistent` (переменной сессии или приложения), тэг `useBean` инициализирует ее, если необходимо, или, если она существует, просто использует ее.
- Тэги потенциально более совместимы с будущими версиями спецификации JSP, или альтернативными реализациями (например, гипотетический JSP-движок, который запоминает переменные в базе данных и организует общий доступ к ним для всех серверных процессов).

Эквивалент тэга `useBean` для нашего приложения выглядит так:

```
<jsp:useBean id="picture"  
            scope="request"  
            class="picturebeans.DOMPicture">  
<%  
    Document doc = DOMUtils.xml4jParse(picturefile);  
    Element nodeRoot = doc.getDocumentElement();  
    nodeRoot.normalize();  
    picture.setNode(nodeRoot);
```

```
%>
</jsp:useBean>
```

или, если вы определили метод `setFile(String)` внутри `DOMBean`:

```
<jsp:useBean id="picture"
  scope="request"
  class="picturebeans.DOMPicture">
  <jsp:setProperty
    name="picture" property="file"
    value="<%=picturefile%>" />
</jsp:useBean>
```

11. Использование XMLEntryList

Чтобы преодолеть некоторые трудности использоваия DOM APIs, я создал класс `XMLEntryList`. Этот класс реализует Java Collections interface `java.util.List`, а также методы `get` и `put` из `java.util.Мap`, обеспечивая более интуитивно уместный набор методов для сканирования простой древовидной структуры XML. Вы можете использовать стандартную абстракцию Collections API, для получения `iterator`-ов и `subviews`. Каждый элемент в `EntryList` имеет ключ и значение, подобно `Map`; ключи являются именами подузлов, а значения — или `Strings` или подструктуры `XMLEntryLists`.

Введение `XMLEntryList` не означает полной замены DOM, Так как данный класс не способен выполнять некоторые функции DOM. Однако это удобное средство выполнения `getXXX`, `setXXX` и списко-ориентированных функций над XML-данными. Например, чтобы получить элемент `caption` узла `picture`, вы можете записать:

```
String caption =
  (String)picturelist.get("caption");
```

Значение поля `caption` уже найдено и сохранено как `String`.

12. Кеширование

Несмотря на преимущества, поиск в XML-занимает время. Чтобы ускорить XML-приложения, вам необходимо использовать кэш. Этот кэш должен хранить XML-объекты в памяти, основываясь на имени файла из которого они были загружены. Если данный файл был модифицирован после загрузки объекта в память,

Совместное использование XML и JSP

тогда объект должен быть перезагружен. Я выполнил простую реализацию такой структуры данных — `CachedFS.java`. Вы можете задействовать `CachedFS` для получения обратного вызова (callback function), используя встроенные (inner) классы, что и выполняет XML-сканирование, преобразуя файл в объект. Затем кэш помещает этот объект в память.

Далее следует код для создания кэша. Этот объект имеет рабочей областью приложение так, что последующие запросы будут использовать тот же самый объект кэша. Я поместил этот код в файл `init.jsp`, так, что вам не нужно `copy/paste` этот инициализирующий код в другие JSP-страницы Web-приложения. Вообще, вы должны определять объекты области приложения в одном месте, чтобы не завершать различные процедуры инициализации в различных местах.

```
<jsp:useBean id="cache"
    class="com.purpletech.io.CachedFS"
    scope="application">
  <% cache.setRoot(application.getRealPath("/"));
    cache.setLoader( new CachedFS.Loader() {
        // load in a single Picture file
        public Object process(
            String path, InputStream in) throws IOException
        {
            try {
                Document doc = DOMUtils.xml4jParse
                    (new BufferedReader(new InputStreamReader(in)));
                Element nodeRoot = doc.getDocumentElement();
                nodeRoot.normalize();
                Picture picture = new DOMPicture(nodeRoot);
                return picture;
            }
            catch (XMLException e) {
                e.printStackTrace();
                throw new IOException(e.getMessage());
            }
        }
    });
  %>
</jsp:useBean>
```

13. XPath

XPath является простым синтаксисом для нахождения узлов на XML-дереве. Его легче использовать, чем DOM, так как вместо того, чтобы каждый раз вызывать метод для перехода к другому узлу, вы помещаете весь путь к узлу в строку — например, `/picture/thumbnails/image[2]`. Resin от Caucho (см. [Ресурсы](#)), включает XPath-процессор, который вы можете использовать в своих приложениях. Вы можете использовать объект Caucho XPath сам по себе, не покупая остальные средства Resin.

```
Node verse = XPath.find("chapter/verse", node);
```

Resin также включает скриптовый язык, совместимый с JavaScript, который обеспечивает простой доступ к средствам XPath и XSL из вашей JSP.

14. XSL

Статья рассматривает возможность помещения Java-кода внутрь JSP, чтобы извлекать данные из XML-узлов. Существует другая популярная модель для выполнения этой задачи: Extensible Stylesheet Language (XSL). Эта модель кардинально отличается от JSP-модели, которую я рассмотрел. В JSP основным документом является HTML, содержащая фрагменты Java-кода; в XSL основным документом является XSL-документ, содержащий фрагменты HTML. Можно было бы многое рассказать о взаимосвязи XSL и Java/JSP, но объемы статьи ограничены. Будущая статья в *JavaWorld* исследует совместное использование XSL и JSP.

15. Заключение и пути дальнейших улучшений

После прочтения этой статьи вы должны иметь ясное представление о структуре JSP-XML-приложения и ее мощи. Вы должны также представлять ограничения данной структуры.

Наиболее скучной частью создания JSP-XML-приложения является написание JavaBeanов для каждого элемента вашей XML-схемы. XML Data Binding развивает технологию, которая автоматически генерирует Java-классы на основе данной схемы. Я также разработал открытый проект-прототип технологии Java-XML data binding.

Совместное использование XML и JSP

IBM alphaWorks недавно выпустил XML Master или XMas еще одну систему XML-Java data binding.

Другой возможностью является расширение функциональности файловой системы новыми более мощными свойствами такими, как запросы и транзакции. Естественно, я рассматриваю возможность реализации подобной файловой системы как открытого проекта. Кто-нибудь хочет написать XML поисковый движок?

16. Об авторе



[Алекс Чаффи](#) работает гуру по программному обеспечению в [jGuru](#). Он

продвигает, обучает и программирует на Java с 1995. В качестве управляющего разработкой программного обеспечения для Earthweb, Алекс был одним из создателей Gamelan, каталога ресурсов для сообщества Java. Он выступал на множестве конференций, опубликовал статьи в нескольких Java журналах, и принес свой вклад в написание книги *The Official Gamelan Java Directory*. Вы можете увидеть его исходный код на <http://www.purpletech.com/code>. Узнайте больше об Алексе на [jGuru](#).

17. Ресурсы

- For the source code for the application built in this article:
<http://www.javaworld.com/jw-03-2000/jspxml/jspxml-webapp.zip>
- For future updates to that source code:
<http://www.purpletech.com/jspxml/>

XML:

- <http://www.xml.org>
- <http://www.xml.com>
- <http://www.jguru.com/faq/XML>

SML, a controversial simplification of XML syntax:

- <http://www.xml.com/pub/1999/11/sml/index.html>

- <http://www.xml.com/pub/1999/12/sml/responses.html>

JSP:

- <http://java.sun.com/products/jsp>
- <http://www.jguru.com/faq/JSP>

DOM specification:

- <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/java-language-binding.html>
- <http://www.w3.org/TR/REC-DOM-Level-1>

Java-XML data binding:

- JSR 31: http://java.sun.com/aboutJava/communityprocess/jsr/jsr_031_xmld.html
- IBM XML Master: <http://alphaworks.ibm.com/tech/xmas>
- Purple Technology XDB: <http://www.purpletech.com/xdb>
- XBeans: <http://www.xbeans.org>
- Tomcat open source servlet/JSP engine:
<http://jakarta.apache.org>
- Java Server Web Development Kit (JSWDK):
<http://java.sun.com/products/servlets>
- XML4J, IBM's DOM parser:
<http://alphaworks.ibm.com>
- For XMLEntryList:
<http://www.purpletech.com/code/src/com/purpletech/xml/XMLEntryList.java>
- For Resin, by Caucho (XPath processor):
<http://www.caucho.com>
- "Understanding JSP Model 2 Architecture," Govind Seshadri (*JavaWorld*, December 1999):
<http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html>

Reprinted with permission from the March 2000 edition of JavaWorld magazine. Copyright © ITworld.com, Inc., an IDG Communications company.

View the original article at: <http://www.javaworld.com/jw-03-2000/jw-0331-ssj-jspxml-2.html>

[Перевод на русский © Павел Серафимович, 2000](#)