Технология JavaServer Pages

Stephanie Bodoff

Технология JavaServer Pages (JSP) позволяет вам легко создавать web содержимое, которое имеет как статические, так и динамические компоненты.

JSP технология воплощает все динамические возможности технологии Java Servlet, но обеспечивает более естественный способ создания статического содержимого. Главные особенности JSP технологии:

- Язык разработки JSP страниц, которые являются текстовыми документами и описывают, как обработать запрос и построить ответ.
- Структурные компоненты для доступа к серверным объектам.
- Механизмы для определения расширений языка JSP.

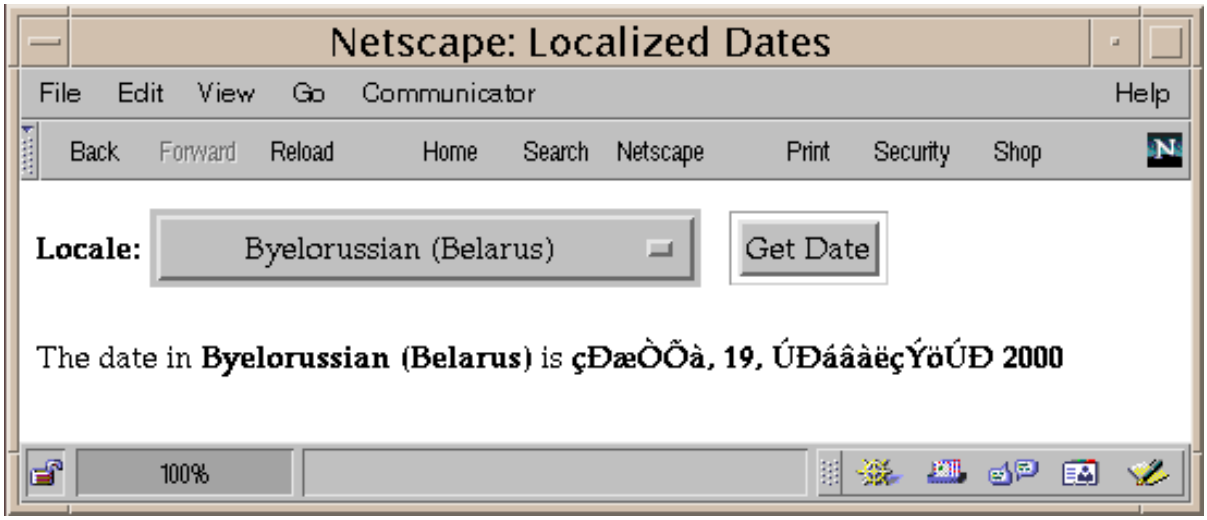
Технология JSP также содержит API, который используется разработчиками web контейнеров, но этот API не рассматривается в данной главе.

1. Что такое JSP Page?

Страница JSP является текстовым документом, которая содержит текст двух типов: статические исходные данные, которые могут быть оформлены в одном из текстовых форматов HTML, SVG, WML, или XML, и JSP элементы, которые конструируют динамическое содержимое.

Следующая web страница является формой, которая позволяет вам выбрать локализацию и отобразить дату в соответствующем стиле.

Copyright © 1999-2004 Javable.com. All rights reserved.



Исходный текст для этого примера расположен в `../examples/`. Страница JSP index.jsp использованная для создания этой формы, показана ниже. Это – типичная смесь статической HTML разметки и элементов JSP. Если вы разрабатывали web страницы, вы хорошо знакомы со структурными выражениями HTML документа (`<head>`, `<body>`, и так далее) и с HTML выражениями, которые создают форму `<form>` и меню `<select>`. Выделенные строки в примере содержат следующие типы JSP конструкций:

- Директивы (**<@page ... %>**) импорта классов из пакета `java.util` и установки типа содержимого, возвращаемого страницей.
- Элемент **jsp:useBean** создает объект, содержащий коллекцию локализаций, и инициализирует переменную, которая указывает на такой объект.
- Скриплеты (**<% ... %>**) извлекают значение параметра, перебирают коллекцию имен локализаций и условно вставляют HTML текст в вывод.
- Выражения (**<%= ... %>**) вставляют значение имени локализации в ответ.
- Элемент **jsp:include** посылает запрос к другой странице (`date.jsp`) и включает ее ответ в ответ вызвавшей страницы.

```
 <%@ page import="java.util.*" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
 <html>
 <head><title>Localized Dates</title></head>
 <body bgcolor="white">
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
<jsp:useBean id="locales" scope="application"
   class="MyLocales"/>
 <form name="localeForm" action="index.jsp" method="post">
 <b>Locale:</b>
 <select name=locale>
<%
   String selectedLocale = request.getParameter("locale");
   Iterator i = locales.getLocaleNames().iterator();
   while (i.hasNext()) {
      String locale = (String)i.next();
      if (selectedLocale != null &&
         selectedLocale.equals(locale)) {
%>
         <option selected><%=locale%></option>
<%
      } else {
%>
         <option><%=locale%></option>
<%
      }
   }
%>
 </select>
 <input type="submit" name="Submit" value="Get Date">
 </form>
<jsp:include page="date.jsp"/>
 </body>
 </html>
```

Чтобы построить, развернуть и выполнить эту страницу:

1. Перейдите в `examples/src` и постройте пример для выполнения `ant date` (смотрите How to Build and Run the Examples).
2. Создайте J2EE приложение, названное `date`.
   1. Выберите File->New Application или нажмите кнопку New Application.
   2. Введите `date.ear` в поле Application File Name.
   3. Нажмите OK.
3. Добавьте web компонент `date` к приложению `date`.
   1. Выберите File->New Web Component или нажмите кнопку New Web Component.
   2. Выберите приложение `date` из Create new WAR File в выпадающем списке

Copyright © 1999-2004 Javable.com. All rights reserved.

Application.
3. Введите `date` в поле WAR Display Name.
4. Нажмите Add.
5. Перейдите в `examples/build/web/date`. Выделите index.jsp, date.jsp, MyDate.class и MyLocales.class и нажмите Add, затем нажмите Finish.
6. Нажмите Next.
7. Выделите JSP в переключателе Web Component, затем нажмите Next.
8. Выберите index.jsp из выпадающего списка JSP Filename. Введите `date` в поле Web Component Display Name. Нажмите Next и Finish.

4. Разверните приложение. Выберите Tools->Deploy Application или нажмите кнопку Deploy Application. В мастере развертывания (deploy), установите context root для `date`.
5. Активизируйте URL `http://<host>:8000/date` в браузере.

Вы увидите выпадающий список, который содержит локализации. Выберите локализацию и нажмите Get Date. Вы увидите дату, представленную в стиле, соответствующем вашей локализации.

2. Пример страниц JSP

Для иллюстрации JSP технологии в этой главе переписывается каждый сервлет приложения `Duke's Bookstore`, представленный в The Example Servlets , как страница JSP:

Функция	JSP страница
Вход в книжный магазин	`bookstore.jsp`
Создание баннера (заголовка) магазина	`banner.jsp`
Просмотр книг, предлагаемых для продажи	`catalog.jsp`
Отбор книг в карту покупок («корзинку»)	`catalog.jsp` and `bookdetails.jsp`
Получить подробную информацию о книге	`bookdetails.jsp`
Показать карту покупок	`showcart.jsp`
Удалить книгу(и) из карты покупок	`showcart.jsp`
Купить книги из карты покупок	`cashier.jsp`

Copyright © 1999-2004 Javable.com. All rights reserved.

Послать подтверждение покупки	`receipt.jsp`

Таблица 1. Пример Duke's Bookstore JSP Pages.

Исходные тексты для этого приложения расположены в `examples/src/web/bookstore2`. Данные для приложения bookstore по прежнему поддерживаются в базе данных Cloudscape. Однако, два изменения сделаны для объекта helper базы данных (database helper object) database.BookDB:

- Объект helper переписан таким образом, чтобы соответствовать моделям проектирования компонентов JavaBeans в JavaBeans Components in JSP Pages. Это изменение делается так, чтобы JSP страницы могли получить доступ к helper объекту, используя элементы языка JSP, специфичные для компонентов JavaBeans.
- Вместо непосредственного доступа к базе данных книжного магазина, helper объект использует enterprise bean. Преимущество использования enterprise bean состоит в том, что объект helper больше не отвечает за соединение с базой данных; эта работа поручена enterprise bean. Кроме того, контейнер EJB поддерживает пул соединений с базой, поэтому enterprise bean может быстрее получить это соединение, чем helper объект. К enterprise bean имеют отношение следующие интерфейсы и классы: `database.BookDBEJBHome` home интерфейс, `database.BookDBEJB` remote интерфейс, и класс реализации `database.BookDBEJB`, который содержит все JDBC вызовы к базе данных.

Наконец, эта версия примера использует апплет для генерации динамических цифровых часов в баннере. Смотрите Including an Applet, где описывается JSP элемент, который генерирует HTML для загрузки апплета.

Чтобы построить, развернуть и исполнить пример:

1. Перейдите в `examples/src` и постройте пример, выполнив `ant bookstore2`.
2. Запустите базу данных Cloudscape, выполнив `cloudscape -start`.
3. Если у вас еще нет базы данных bookstore, выполните `ant create-web-db`.
4. Запустите `j2ee` сервер.
5. Запустите `deploytool`.
6. Создайте J2EE приложение, названное `bookstore2`.
    1. Выберите File->New Application или нажмите кнопку New Application.
    2. Введите `bookstore2.ear` в поле Application File Name.
    3. Нажмите OK.

Copyright © 1999-2004 Javable.com. All rights reserved.

7. Добавьте `bookstore2` WAR к приложению `bookstore2`.
   1. Выберите File->Add to Application->Web WAR или нажмите кнопку Web WAR.
   2. В диалоге Add Web WAR перейдите в `examples/build/web/bookstore2`. Выделите bookstore2.war. Нажмите Add Web WAR.
8. Добавьте к приложению `BookDBEJB` enterprise bean.
   1. Выберите File->New Enterprise Bean или нажмите кнопку New Enterprise Bean.
   2. Выберите из выпадающего списка Enterprise Bean bookstore2.
   3. Введите `BookDBEJB` в поле JAR Display Name.
   4. Кликните Add Чтобы добавить файлы содержимого. Перейдите в папку `examples/build/web/ejb` и добавьте базу данных и пакеты исключений. Нажмите Next.
   5. Выберите тип Bean: Session и Stateless.
   6. Выберите BookDBEJBImpl для класса Enterprise Bean.
   7. Выберите BookDBEJBHome для Home интерфейса.
   8. Выберите BookDBEJB для Remote интерфейса.
   9. Введите BookDBEJB для имени Enterprise Bean.
   10. Введите `BookDBEJB` в поле JNDI Name около компонента `BookDB`.
9. Добавьте ресурсную ссылку на базу данных Cloudscape.
   1. Выберите BookDBEJB enterprise bean.
   2. Выберите вкладку Resource Ref's.
   3. Нажмите Add.
   4. Выберите javax.sql.DataSource из списка Type.
   5. Введите `jdbc/BookDB` в поле Coded Name.
   6. Введите `jdbc/Cloudscape` в поле JNDI Name.
10. Добавьте ссылку на enterprise bean `BookDBEJB`.
    1. Выберите bookstore2 WAR.
    2. Выберите вкладку EJB Ref's.
    3. Нажмите Add.
    4. Введите `ejb/BookDBEJB` в поле Coded Name.
    5. Введите Session в поле Type.
    6. Введите `BookDBEJBHome` в поле Home.
    7. Введите `BookDBEJB` в поле Remote.
    8. Введите `BookDBEJB` в поле JNDI Name.
11. Разверните приложение. Выберите Tools->Deploy Application или нажмите кнопку

Copyright © 1999-2004 Javable.com. All rights reserved.

Deploy Application. Нажмите Next дважды. Введите `BookDBEJB` в поле Application->JNDI Name и `bookstore2` для context root. Нажмите Next и Finish.

12. Откройте bookstore URL `http://<host>:8000/bookstore2/enter`.

Смотрите Troubleshooting для помощи с диагностикой общих проблем.

3. Цикл жизни JSP Page

Страница JSP обслуживает запросы подобно сервлету. Таким образом, цикл жизни и многие особенности JSP pages (в частности динамические аспекты) определены технологией Java Servlet technology, и многое из обсуждаемого в данной главе ссылается на функции, описанные в Java Servlet Technology.

Когда запрос обращается к JSP странице, он обрабатывается специальным сервлетом, который сначала проверяет, старше ли сервлет этой JSP страницы, чем сама JSP page. Если старше, он транслирует эту JSP страницу в класс сервлета и компилирует этот класс. Одно из преимуществ JSP page над сервлетами состоит в том, что процесс «построения» выполняется автоматически.

3.1. Трансляция и компиляция

На этапе трансляции каждый тип данных в JSP page интерпретируется по разному. Исходные данные трансформируются в код, который будет выделять данные в поток, возвращающий данные клиенту. JSP элементы интерпретируются следующим образом:

- Директивы используются, чтобы контролировать, как web контейнер транслирует и выполняет JSP страницу.
- Скриптовые элементы (Scripting elements) вставляются в класс сервлета JSP страницы. Подробности смотрите в JSP Scripting Elements.
- Элементы форм `<jsp:XXX ... />` конвертируются в метод, вызывающий компоненты JavaBeans или вызовы Java Servlet API.

Для JSP страницы, названной *pageName*, исходный текст сервлета JSP страницы servlet хранится в файле:

```
J2EE_HOME/repository/host/web/context root/pageName_jsp.java
```


Copyright © 1999-2004 Javable.com. All rights reserved.

Например, исходный текст для страницы с именем index (`index.jsp`) для примера локализации даты, обсуждаемого в начале главы, был бы назван:

```
J2EE_HOME/repository/host/web/date/index_jsp.java
```

Этапы трансляции и компиляции могут выдавать ошибки, которые наблюдаются, только когда страница запрашивается в первый раз. Если ошибка встречается во время трансляции страницы (например, если транслятор встречает плохо сформированный JSP элемент) сервер выбросит `ParseException`, и исходный файл класса сервлета будет пустым или неполным. Последняя некомплектная строка укажет на некорректный JSP элемент.

Если ошибка встречается во время компиляции (например, ошибка синтаксиса в скриплете), сервер вернет `JasperException` и сообщение, которое содержит имя сервлета JSP страницы и строку, в которой встретилась ошибка.

Если страница была оттранслирована и откомпилирована, полученный сервлет в основном следует циклу жизни обычного сервлета, описанному в

Servlet Life Cycle:

1. Если экземпляр сервлета JSP страницы еще не создан, контейнер:
    1. Загружает класс сервлета JSP страницы
    2. Устанавливает экземпляр класса сервлета
    3. Инициализирует экземпляр сервлета, вызывая метод `jspInit`
2. Вызывает метод `_jspService`, передавая объекты запроса и ответа.

Если контейнеру необходимо удалить этот сервлет JSP страницы, он вызывает метод `jspDestroy`.

3.2. Выполнение

Вы можете контролировать различные параметры выполнения JSP страницы. Директивы, которые относятся к буферизации выхода и ошибкам обработки, обсуждаются здесь. Другие директивы рассматриваются в контексте специфических задач на протяжении всей главы.

3.2.1. Буферизация

Copyright © 1999-2004 Javable.com. All rights reserved.

Когда выполняется JSP страница, выход, записанный в объект `response`, автоматически буферизуется. Вы можете настраивать размер буфера следующей директивой:

```
<%@ page buffer="none|xxxkb" %>
```

Буфер большего размера позволяет записать больше содержимого прежде, чем что-то будет реально отправлено обратно клиенту. Таким образом JSP странице выделяется больше времени для установки соответствующих кодов состояния и заголовков или переадресации к другим web ресурсам. Маленький буфер сокращает загрузку памяти сервера и позволяет клиенту начать получение данных быстрее.

3.2.2. Ошибки обработки

Любое количество исключений может возникать в процессе выполнения JSP страницы. Для определения того, что web контейнер должен передать управление странице ошибок (error page), если встречается исключение, поместите следующую директиву в начало вашей JSP page:

```
<%@ page errorPage="file_name" %>
```

Страница приложения `Duke's Bookstore` initdestroy.jsp содержит директиву

```
<%@ page errorPage="errorpage.jsp"%>
```

В начале страницы errorpage.jsp указано, что она используется как страница ошибок. Для этого служит директива:

```
<%@ page isErrorPage="true|false" %>
```

Эта директива делает объекты исключений (типа javax.servlet.jsp.JspException) доступными странице ошибок, так что вы можете извлечь, интерпретировать и отобразить информацию о причине исключения на странице ошибок.

Замечание:

Вы также можете определить страницу ошибок для WAR, который содержит JSP страницу. Если страница ошибок определена и для WAR и для JSP page, ошибка JSP страницы принимает приоритет.

Copyright © 1999-2004 Javable.com. All rights reserved.

4. Инициализация и завершение JSP Page

Вы можете переделать процесс инициализации, чтобы позволить JSP странице читать постоянные данные конфигурации, инициализировать ресурсы и выполнить любые другие разовые действия путем перегрузки метода `jspInit` интерфейса `JspPage`. Вы освобождаете ресурсы, используя метод `jspDestroy`. Методы, определяемые с использованием JSP деклараций, обсуждаются в разделе Declarations.

Страница примера bookstore initdestroy.jsp определяет метод `jspInit`, чтобы найти или создать компонент JavaBeans database.BookDB, который представляет базу данных bookstore:

```
private BookDB bookDB;
public void jspInit() {
   bookDB =
   (BookDB)getServletContext(). getAttribute("bookDB");

   if (bookDB == null) {
      try {
         bookDB = new BookDB();
         getServletContext().setAttribute(
            "bookDB", bookDB);
      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }
}
```

И метод `jspDestroy`, чтобы удалить компонент JavaBeans и освободить переменную `BookDB`.

```
public void jspDestroy() {
   try {
      bookDB.remove();
   } catch (Exception ex) {
      System.out.println(ex.getMessage());
   }
   getServletContext().removeAttribute("bookDB");
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
   bookDB = null;
}
```

Дальше следует ввод в действие бина database. Этот бин имеет две переменных: текущую книгу и ссылку на *enterprise bean*database. Ссылка создана с применением технологии, описанной в Getting Started.

```
public class BookDB {
   String bookId = "0";
   private BookDBEJB database = null;

   public BookDB () throws Exception {
      try {
         InitialContext ic = new InitialContext();
         Object objRef = ic.lookup(
            "java:comp/env/ejb/BookDBEJB");
         BookDBEJBHome home =
            (BookDBEJBHome)PortableRemoteObject.
               narrow(objRef, database.BookDBEJBHome.class);
         database = home.create();
      } catch (RemoteException ex) {
         throw new Exception(
            "Couldn't create database bean.");
      } catch (CreateException ex) {
         throw new Exception(
            "Couldn't create database bean.");
      } catch (NamingException ex) {
         throw new Exception("Unable to lookup home: "+
         "java:comp/env/ejb/BookDBEJB.");
   }
   public void remove () throws Exception {
      try {
         database.remove();
         database = null;
      } catch (RemoteException ex) {
         throw new Exception(
            "Error while removing database bean.");
      } catch (EJBException ex) {
         throw new Exception(
            "Error while removing database bean.");
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
      } catch (RemoveException ex) {
         throw new Exception(
            "Error while removing database bean.");
      }
   }
   public void setBookId(String bookId) {
      this.bookId = bookId;
   }
   ...
}
```

Так как database bean разделяется между всеми JSP страницами (совместно используется ими), он должен быть инициализирован, когда стартует приложение, вместо того чтобы инициализироваться в каждой JSP странице. Java Servlet технология обеспечивает события цикла жизни приложения и классы listener для этих целей. В качестве упражнения вы можете перенести код, который управляет бином, в context listener class. Смотрите Handling Servlet Life Cycle Events о context listener, который инициализирует версию Java Servlet приложения bookstore.

5. Создание статического содержимого

Вы создаете статическое содержимое JSP страницы, просто написав ее так, как будто она состоит только из этого содержимого. Статическое содержимое может быть представлено в любом текстовом формате, таком как HTML, WML или XML. Форматом по умолчанию является HTML. Если вы хотите использовать другой формат, вы включаете в начало вашей JSP страницы `page` директиву с атрибутом `contentType`, устанавливающим тип формата. Например, если вы хотите, чтобы страница содержала данные, представленные в wireless markup language (WML), вы должны включить следующую директиву:

```
<%@ page contentType="text/vnd.wap.wml"%>
```

Реестр имен типа содержимого хранится в IANA (Агентство по выделению имен и уникальных параметров протоколов интернет) на:

```
ftp://ftp.isi.edu/in-notes/iana/assignments/media-types
```


Copyright © 1999-2004 Javable.com. All rights reserved.

6. Создание динамического содержимого

Вы создаете динамическое содержимое подключением Java объектов из скриптовых элементов.

6.1. Использование объектов внутри JSP Pages

Вы можете из JSP page получить доступ к разнообразным объектам, включая enterprise beans и JavaBeans компоненты. JSP технология автоматически делает такие объекты доступными, и вы можете также создать и получить доступ к объектам, специфическим для приложения.

6.1.1. Неявные объекты

Неявные объекты создаются web контейнером и содержат информацию, связанную с отдельными запросами, страницами или приложениями. Многие из этих объектов определяются Java Servlet технологией, на которой основана JSP технология, и которая подробно обсуждается в Java Servlet Technology. Таблица 19 перечисляет эти неявные объекты.

Переменная	Класс	Пояснение
`application`	javax.servlet.ServletContext	Контекст для сервлета JSP страницы и любого web компонента, содержащегося в том же самом приложении. Смотрите Accessing the Web Context.
`config`	javax.servlet.ServletConfig	Информация об инициализации сервлета JSP страницы.
`exception`	java.lang.Throwable	Доступно только из страницы ошибок (error page). Смотрите Handling Errors.
`out`	javax.servlet.jsp.JspWriter	Выходной поток.
`page`	java.lang.	Экземпляр сервлета JSP

Copyright © 1999-2004 Javable.com. All rights reserved.

	Object	страницы, обрабатывающий текущий запрос. Обычно не используется авторами JSP страниц.
`pageContext`	javax.servlet.jsp.PageContext	Контекст JSP страницы. Содержит единственный API для управления различными контекстными атрибутами, описанными в Sharing Information. Этот API широко используется, когда реализуют tag handlers (программы обработки тегов) (смотрите Tag Handlers).
`request`	подтип от javax.servlet.ServletRequest	Запрос, запускающий выполнение JSP страницы. Смотрите Getting Information From Requests.
`response`	подтип от javax.servlet.ServletResponse	Ответ, возвращаемый клиенту. Обычно не используется авторами JSP страниц.
`session`	javax.servlet.http.HttpSession	Объект – сессия клиента. Смотрите Accessing the Web Context.

Таблица 2. Неявные объекты.

6.1.2. Объекты, специфические для приложения

Если возможно, поведение приложения должно быть инкапсулировано в объекты, чтобы разработчики JSP страниц могли сосредоточиться на представлении результатов. Объекты могут быть созданы разработчиками, которые являются специалистами по Java, организации доступа к базам данных и другим сервисам. Имеется четыре пути создания и использования объектов внутри страницы JSP:

- Экземпляры и переменные класса сервлета JSP страницы создаются в декларациях (*declarations)* и становятся доступными в скриплетах (*scriptlets)* и выражениях (*expressions)*.

Copyright © 1999-2004 Javable.com. All rights reserved.

- Локальные переменные класса сервлета JSP страницы создаются в декларациях и становятся доступными в скриплетах и выражениях.
- Атрибуты scope объектов (смотрите Scope Objects) создаются и используются в скриплетах и выражениях.
- JavaBeans компоненты могут быть созданы и стать доступными с использованием модернизированных (streamlined) JSP элементов. Эти элементы обсуждаются в главе JavaBeans Components in JSP Pages. Вы можете создать компонент JavaBeans в декларации или скриплете, и вызвать его методы в скриплетах и выражениях.

Декларации, скриплеты и выражения описаны ниже в JSP Scripting Elements.

6.1.3. Разделяемые объекты

Условия эффективного конкурентного доступа к разделяемым объектам описаны в Sharing Information и относятся к объектам, доступным из JSP страниц, которые выполняются как многопоточные сервлеты. Вы можете указать, как web контейнер должен координировать множественные клиентские запросы с помощью следующей `page` директивы:

```
<%@ page isThreadSafe="true|false" %>
```

Если `isThreadSafe` установлен в `true`, web контейнер может выбрать отправку множественных конкурентных клиентских запросов странице JSP. Это – настройка по умолчанию. Если используется `true`, вы должны должным образом синхронизировать доступ ко всем разделяемым объектам, определенным на уровне этой страницы. Сюда включаются объекты, созданные в декларациях, компоненты JavaBean, доступные на странице, и атрибуты scope объектов страницы.

Если `isThreadSafe` установлен в `false`, запросы направляются по одному в каждый момент времени в порядке их получения, и доступ к объектам страницы не нужно контролировать. Однако, вы все еще должны обеспечить, чтобы доступ к атрибутам приложений или `session` scope объектов и JavaBean компонентам был должным образом синхронизирован.

6.2. JSP скриптовые элементы

JSP скриптовые элементы используются, чтобы создать и получить доступ к объектам,

Copyright © 1999-2004 Javable.com. All rights reserved.

определить методы и управлять потоком контроля. Так как одной из целей JSP технологии является отделение статических исходных данных от кода, необходимого для генерации динамического содержимого, рекомендуется очень умеренно использовать JSP скрипты. Много работы, которая требует использования скриптов, может быть устранена при использовании заказных тегов, описанных ниже в разделе Extending the JSP Language (расширения языка JSP).

JSP технология позволяет контейнеру поддерживать любые языки скриптов, которые могут вызывать Java объекты. Если вы хотите использовать другой язык скриптов вместо `java`, который задан по умолчанию, вы должны определить его в `page` директиве в начале JSP страницы:

```
<%@ page language="scripting language" %>
```

Так как скриптовые элементы конвертируются в выражения языка программирования в классе сервлета JSP страницы, вы должны декларировать все классы и пакеты, используемые в JSP странице.

Если языком страницы является `java`, вы декларируете, что JSP страница будет использовать класс или пакет, с помощью `page` директивы:

```
<%@ page import="packagename.*, fully_qualified_classname" %>
```

Страничка примера bookstore showcart.jsp импортирует классы, необходимые, чтобы реализовать карту покупок (shoppingcart), с помощью следующей директивы:

```
<%@ page import="java.util.*, cart.*" %>
```

6.2.1. Декларации

Декларация используется, чтобы объявить переменные и методы языка скриптов страницы. Синтаксис декларации имеет вид:

```
<%! декларация языка скриптов %>
```

Если в качестве языка скриптов используется Java, переменные и методы в JSP декларациях становятся декларациями класса сервлета JSP страницы.

Copyright © 1999-2004 Javable.com. All rights reserved.

Страничка примера bookstore initdestroy.jsp определяет экземпляр переменной, названной `bookDB`, а также методы инициализации и завершения `jspInit` и `jspDestroy`, обсужденные раньше:

```
<%!
   private BookDB bookDB;

   public void jspInit() {
      ...
   }
   public void jspDestroy() {
      ...
   }
%>
```

6.2.2. Скриплеты

Скриплет может содержать любой фрагмент кода, который адекватен языку скриптов, используемому на странице. Синтаксис скриплета имеет вид:

```
<%
       выражения на языке скриптов
%>
```

Если в качестве языка скриптов выбран `java`, скриплет трансформируется во фрагмент Java оператора и вставляется в метод service сервлета JSP страницы. Переменные языка программирования, созданные скриплетом, доступны везде на JSP странице.

JSP страница showcart.jsp содержит скриплет, который извлекает итератор для коллекции элементов, содержащихся в карте покупок (корзинке), и настраивает логическую структуру для перебора всех элементов карты. Внутри цикла JSP страница извлекает свойства объектов (книг) и форматирует их, используя разметку HTML. Так как цикл `while` открывает блок, разметка HTML формируется скриплетом, который закрывает этот блок.

```
<%
   Iterator i = cart.getItems().iterator();
```

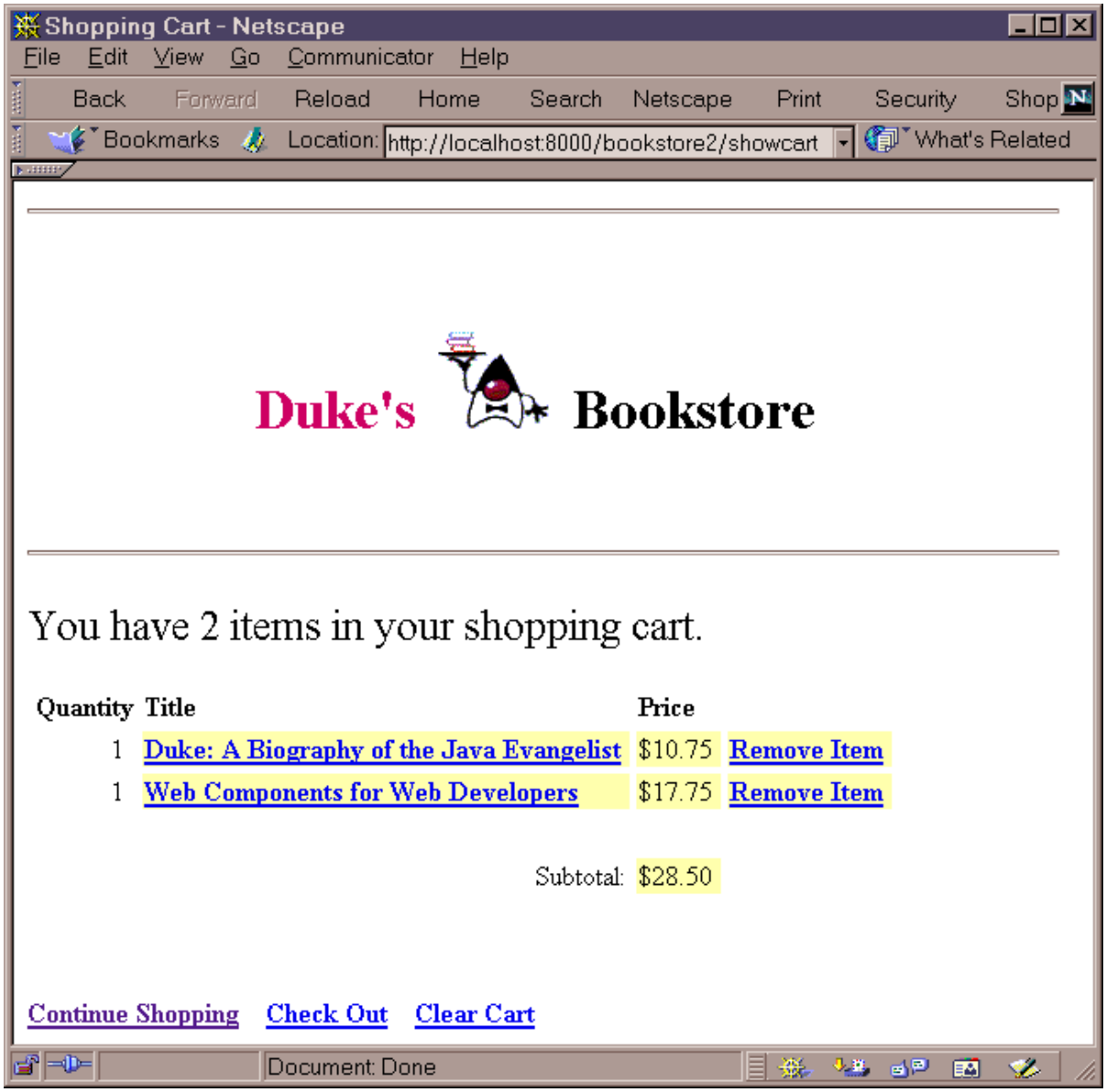

Copyright © 1999-2004 Javable.com. All rights reserved.

```
   while (i.hasNext()) {
      ShoppingCartItem item =
         (ShoppingCartItem)i.next();
      BookDetails bd = (BookDetails)item.getItem();
%>

      <tr>
      <td align="right" bgcolor="#ffffff">
      <%=item.getQuantity()%>
      </td>
      <td bgcolor="#ffffaa">
      <strong><link href="
      <%=request.getContextPath()%>/bookdetails?bookId=
      <%=bd.getBookId()%>"><%=bd.getTitle()%></a></strong>
      </td>
      ...
<%
   // Конец цикла while
   }
%>
```

Результат показан ниже:

Copyright © 1999-2004 Javable.com. All rights reserved.



6.2.3. Выражения

JSP выражения используются, чтобы вставлять значение выражения языка скриптов, преобразованное в строку, в поток данных, возвращаемых клиенту. Если языком скриптов является Java, выражение трансформируется в оператор, который

Copyright © 1999-2004 Javable.com. All rights reserved.

преобразует значение выражения в объект `String` и вставляет его в неявный `out` объект.

Синтаксис выражения имеет вид:

```
<%=выражение языка скриптов %>
```

Отметим, что точка с запятой не допускается внутри JSP выражений, даже если такое выражение содержит точку с запятой, когда вы используете его в скриплете.

Следующий скриплет извлекает количество элементов в карте покупок:

```
<%
   // Печатаем сводку для карты покупок (корзинки)
   int num = cart.getNumberOfItems();
   if (num > 0) {
%>
```

Выражения затем используются, чтобы вставить значение из `num` в выходной поток и определить соответствующую строку, чтобы вставить количество:

```
<font size="+2">You have <%= num %>
   <%= (num==1 ? " item" : " items") %> in your shopping cart.
</font>
```

7. Включение содержимого в JSP страницу

Имеется два механизма для включения содержимого из другого источника в страницу JSP: директива `include` и элемент `jsp:include`.

Директива `include` обрабатывается, когда JSP страница транслируется в класс сервлета. Результатом директивы является вставка в JSP страницу текста, содержащегося в другом файле: или статического содержимого или другой JSP страницы. Вы вероятно должны использовать директиву `include`, чтобы включить содержимое баннера, информацию об авторском праве, или любую порцию содержимого, которую вы хотели бы многократно использовать в другой JSP странице. Синтаксис директивы `include` имеет вид:

Copyright © 1999-2004 Javable.com. All rights reserved.

```
<%@ include file="filename" %>
```

Например, все страницы приложения bookstore включают файл banner.jsp, содержащий текст баннера, с помощью следующей директивы:

```
<%@ include file="banner.jsp" %>
```

Вдобавок, страницы bookstore.jsp, bookdetails.jsp, catalog.jsp, ишowcart.jsp содержат элементы JSP, которые создают и уничтожают бин `database` с помощью элемента:

```
<%@ include file="initdestroy.jsp" %>
```

Поскольку вы должны статически вставлять директиву `include` в каждый файл, чтобы многократно использовать ее ресурс, этот подход имеет ограничения. Более гибкий подход построения страницы из кусков содержимого смотрите в A Template Tag Library.

Элемент `include` обрабатывается, когда страница JSP исполняется. Операция `include` позволяет вам включить или статический или динамический файл в JSP файл. Результат включения статических или динамических файлов получается совершенно различным. Если файл является статическим, его содержимое вставляется в вызывающий JSP файл. Если файл является динамическим, включаемой JSP странице посылается запрос, она исполняется, и затем результат включается в ответ вызвавшей ее JSP страницы. Синтаксис для элемента `jsp:include` имеет вид:

```
<jsp:include page="includedPage" />
```

Приложение `date`, представленное в начале главы, включает страницу, которая генерирует изображение локализованной даты, с помощью следующего элемента:

```
<jsp:include page="date.jsp"/>
```

8. Контроль пересылки данных другому Web компоненту

Механизм контроля пересылки данных другому web компоненту из JSP страницы использует функциональность, обеспечиваемую Java Servlet API как описано в Transferring a Control to Another Web Component. Вы получаете доступ к этой

Copyright © 1999-2004 Javable.com. All rights reserved.

функциональности из JSP страницы с помощью элемента jsp:forward:

```
<jsp:forward page="/main.jsp" />
```

Отметим, что если какие-либо данные уже возвращены клиенту, элемент jsp:forward откажется работать и выбросит IllegalStateException.

8.1. Элемент Param

Когда вызывается элемент include или forward, целевой странице передается исходный объект request. Если вы хотите передать этой странице дополнительные данные, можете добавить параметры к объекту request с помощью элемента param:

```
<jsp:include page="..." >
   <jsp:param name="param1" value="value1"/>
</jsp:include>
```

9. Включение апплета

Вы можете включить апплет или компонент JavaBeans в JSP страницу с помощью элемента jsp:plugin. Этот элемент генерирует HTML, который содержит структурные компоненты, зависящие от клиентского браузера (<object> или <embed>). Это обеспечит загрузку Java Plugin (сменных программных модулей Java), если требуется, и клиентского компонента, и последующее выполнение клиентского компонента. Синтаксис для элемента jsp:plugin имеет вид:

```
<jsp:plugin
   type="bean|applet"
   code="objectCode"
   codebase="objectCodebase"
   { align="alignment" }
   { archive="archiveList" }
   { height="height" }
   { hspace="hspace" }
   { jreversion="jreversion" }
   { name="componentName" }
   { vspace="vspace" }
   { width="width" }
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
   { nspluginurl="url" }
   { iepluginurl="url" } >
   { <jsp:params>
      { <jsp:param name="paramName" value= paramValue" /> }+
   </jsp:params> }
   { <jsp:fallback> arbitrary_text </jsp:fallback> }
</jsp:plugin>
```

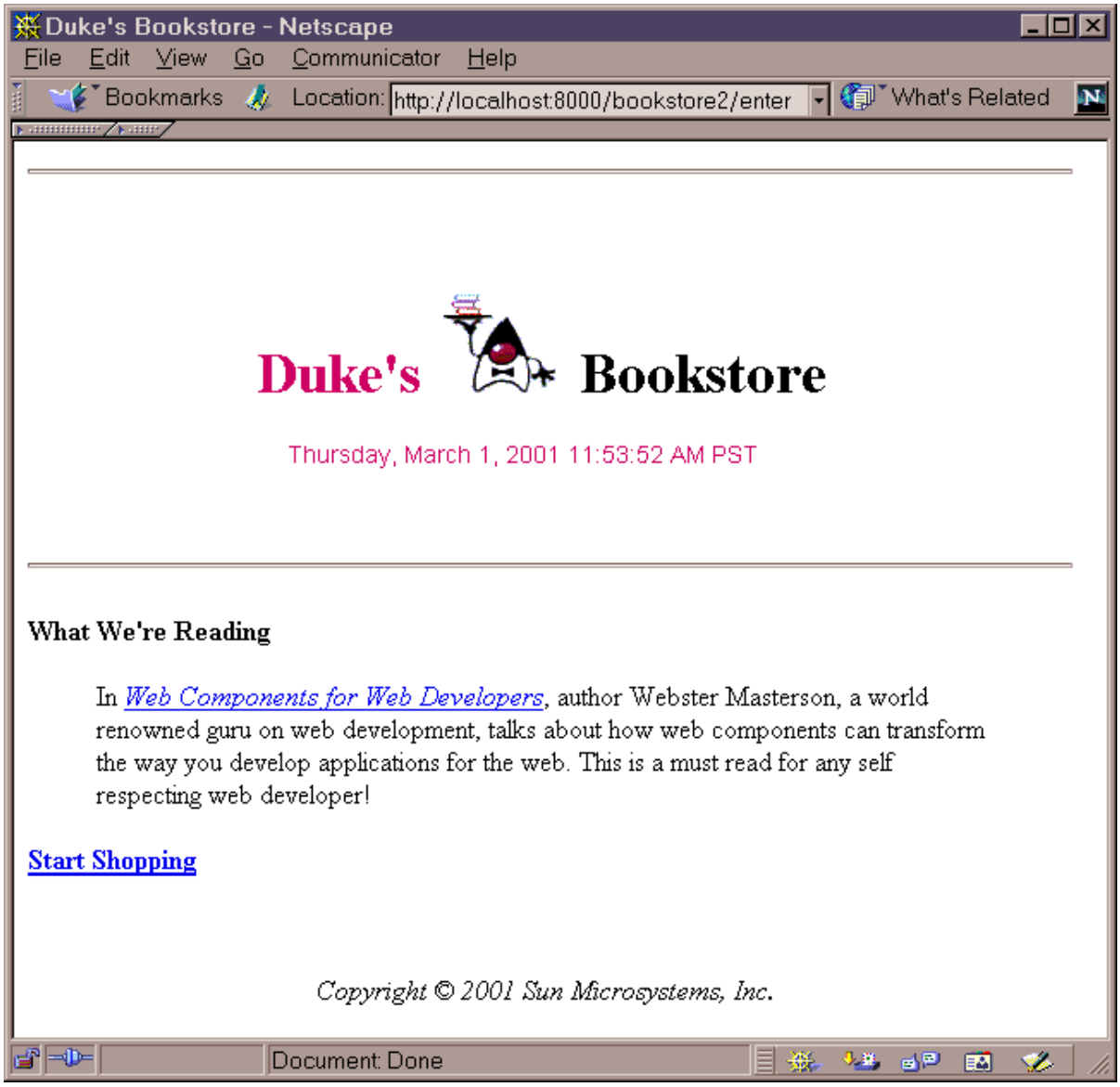
Тег `jsp:plugin` заменяется тегом `<object>` или `<embed>`, который соответствует запросу клиента. Атрибуты тега jsp:plugin обеспечивают конфигурационные данные для представления элемента, а также версию нужного plugin. Атрибуты `nspluginurl` и `iepluginurl` определяют URL, откуда этот plugin может быть загружен. Элементы `jsp:param` указывают параметры апплета или компонента JavaBeans.

Элемент `jsp:fallback` указывает на содержимое, которое должно быть использовано клиентским браузером, если этот plugin не сможет стартовать (поскольку `<object>` или `<embed>` не поддерживается клиентом, или имеется другая проблема).

Если этот plugin может стартовать, но апплет или компонент JavaBeans не может быть найден или запущен, пользователь получит специфическое для plugin сообщение (окно с сообщением о `ClassNotFoundException`).

Страничка banner.jsp примера `Duke'sBookstore`, которая создает баннер, показывающий динамические цифровые часы, генерируемые `DigitalClock`:

Copyright © 1999-2004 Javable.com. All rights reserved.



Элемент `jsp:plugin` использован для загрузки апплета следующим образом:

```
<jsp:plugin
   type="applet"
   code="DigitalClock.class"
   codebase="/bookstore2"
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
   jreversion="1.3"
   align="center" height="25" width="300"
   nspluginurl="http://java.sun.com/products/
                        plugin/1.3.0_01/plugin-install.html"
   iepluginurl="http://java.sun.com/products/
                        plugin/1.3.0_01/jinstall-130_01-win32.cab#Version=1,3,0,1" >
   <jsp:params>
      <jsp:param name="language"
         value="<%=request.getLocale().getLanguage()%>" />
      <jsp:param name="country"
         value="<%=request.getLocale().getCountry()%>" />
      <jsp:param name="bgcolor"
        value="FFFFFF" />
      <jsp:param name="fgcolor"
        value="CC0066" />
   </jsp:params>
      <jsp:fallback>
      <p>Unable to start plugin.</p>
   </jsp:fallback>
</jsp:plugin>
```

10. Расширения языка JSP

Вы можете выполнять широкое множество задач динамической обработки с помощью компонентов JavaBeans в соединении со скриплетами, в том числе: организацию доступа к базам данных, использование enterprise services (служб), таких как электронная почта (email) или каталоги (directories), и управление потоком. Один из недостатков скриплетов однако состоит в том, что они могут сделать JSP страницу более трудной для поддержки. В качестве альтернативы JSP технология предусматривает механизм, называемый *заказными тегами*, которые позволяют вам инкапсулировать динамическую функциональность в объекты, которые доступны благодаря расширениям языка JSP. Заказные теги приносят выгоды от другого типа компоновки JSP страницы.

Например, повторный вызов скриплета, используемого в цикле, и отображающего содержимое карты покупок примера `Duke'sBookstore`:

```
<%
```


Copyright © 1999-2004 Javable.com. All rights reserved.

```
   Iterator i = cart.getItems().iterator();
   while (i.hasNext()) {
      ShoppingCartItem item =
         (ShoppingCartItem)i.next();
      ...
%>
      <tr>
      <td align="right" bgcolor="#ffffff">
      <%=item.getQuantity()%>
      </td>
      ...
<%
   }
%>
```

Заказной тег `iterate` устраняет логическую часть кода и управляет скриптовой переменной `item`, которая ссылается на переменные в карте покупок:

```
<logic:iterate id="item"
   collection="<%=cart.getItems()%>"
   <tr>
   <td align="right" bgcolor="#ffffff">
   <%=item.getQuantity()%>
   </td>
   ...
</logic:iterate>
```

Заказные теги упаковывают и распространяют как модуль, называемый библиотекой тегов. Синтаксис заказных тегов тот же самый, что и для JSP элементов, а именно `<prefix:tag>`, но для заказных тегов этот prefix определяется пользователем библиотеки тегов и `tag` определяется его разработчиком. Раздел CustomTagsinJSPPages объясняет, как использовать и разрабатывать заказные теги.

Перевод на русский © Сергей Киреев, 2001

Copyright © 1999-2004 Javable.com. All rights reserved.