

Архитектура "Модели 2" для JavaServer Pages

Исследование паттерна "Модель-Вид-Контроллер" (Model-View-Controller — MVC).

Server-Side Java

Говинд Сешадри

Рассматривая приведенную в статье реализацию корзины покупателя, вы изучите, как использовать паттерн проектирования "Модель-Вид-Контроллер" (MVC) и по-настоящему отделять представление от содержания при использовании JavaServer Pages. Govind Seshadri показывает насколько просто это можно сделать. *(2000 слов)*

Несмотря на относительно недавнее появление, технология JavaServer Pages (JSP) идет к тому, чтобы стать лидирующей Java технологией создания приложений для обработки динамического Web-содержания. Java-разработчики любят JSP по многим причинам. Некоторые из-за того, что она реализует парадигму "write once, run anywhere" по отношению к интерактивным Web-страницам; других привлекает простота изучения и возможность использования Java в качестве языка сценариев на стороне сервера. Но большинство сходится на одном мнении — самое большое преимущество использования JSP состоит в том, что эта технология помогает эффективно отделить представление от содержания. В этой статье я рассматриваю более подробно возможность оптимального отделения представления от содержания с использованием архитектуры JSP, известной как "Модель 2". Эта модель может также рассматриваться в качестве серверной реализации популярного образца (паттерна) "Модель-Вид-Контроллер" (MVC) . Вы должны быть знакомы с основами программирования JSP и сервлетов для того, чтобы лучше понять данный материал,

поскольку я не касаюсь вопросов синтаксиса в этой статье.

1. Так что же не так с сервлетами?

Поскольку технология JSP очень удобна для обработки динамического Web-содержания и отделения содержания от представления, некоторые могут все еще задаваться вопросом, почему нужно бросать сервлеты ради JSP. Полезность сервлетов не является темой для дискуссии. Они превосходны для обработки данных на стороне сервера, и, остаются ключевой технологией, благодаря широкой поддержке. Фактически, с точки зрения архитектуры JSP можно рассматривать как абстракцию более высокого уровня по отношению к сервлетам, которая реализована как расширение Servlet 2.1 API. Однако, Вы не должны использовать сервлеты бездумно; они не всегда соответствуют предъявляемым требованиям. Например, если дизайнеры могут легко создавать JSP-страницы, используя обычные инструментальные средства для верстки HTML или XML, сервлеты более удобны для разработчиков серверной логики (а это — процесс, который вообще требует более высокого опыта программирования), потому что они часто используют IDE. При размещении сервлетов, разработчики должны быть внимательными и убедиться, что не существует тесной связи между представлением и содержанием. Возможно Вы обычно добиваетесь этого, используя дополнительный пакет "HTML-обертки" (wrapper), например `htmlKona`. Но даже этот подход, обеспечивающий некоторую гибкость в вопросах изменения простых элементов экранного представления, все же не ограждает Вас от изменений непосредственно в формате представления. Например, если ваше представление, изменилось от HTML к DHTML, Вам бы пришлось убедиться, что пакеты "обертки" совместимы с новым форматом. В самом худшем сценарии, если пакет "обертки" не доступен, Вы можете жестко закодировать представление вместе с динамическим содержанием. Так, что же является решением? Как Вы скоро увидите, один из подходов состоит в том, чтобы использовать и JSP, и сервлеты для создания приложений.

2. Различные подходы

В ранних спецификациях JSP были выделены два подхода к формированию приложений, использующих JSP. Эти подходы, названные Моделью 1 и Моделью 2

Архитектура "Модели 2" для JavaServer Pages

(JSP Model 1/2 architectures), отличаются, по существу, тем, какой компонент выполняет большую часть обработки запроса. В Модели 1 (рис. 1), JSP страница сама обрабатывает входящий запрос и генерирует ответ для клиента. Разделение представления и содержания присутствует, так как весь доступ к данным выполняется через компоненты Java-beans. Хотя, вероятно, Модель 1 вполне подходит для простых приложений, ее использование в сложных системах нежелательно. Неразборчивое использование этой архитектуры обычно ведет к большому количеству скриплетов или Java-кода, внедренного в JSP-страницу, особенно, если необходимо выполнить объемную обработку запроса. Хотя это и не проблема для Java-разработчиков, это становится проблемой, если ваши JSP страницы создаются и поддерживаются дизайнерами, что является обычной практикой при разработке крупных проектов. В конечном счете, это может вести к неясному распределению ролей и обязанностей между членами команды, вызывая головные боли у руководителей проекта, которых можно было бы легко избежать.

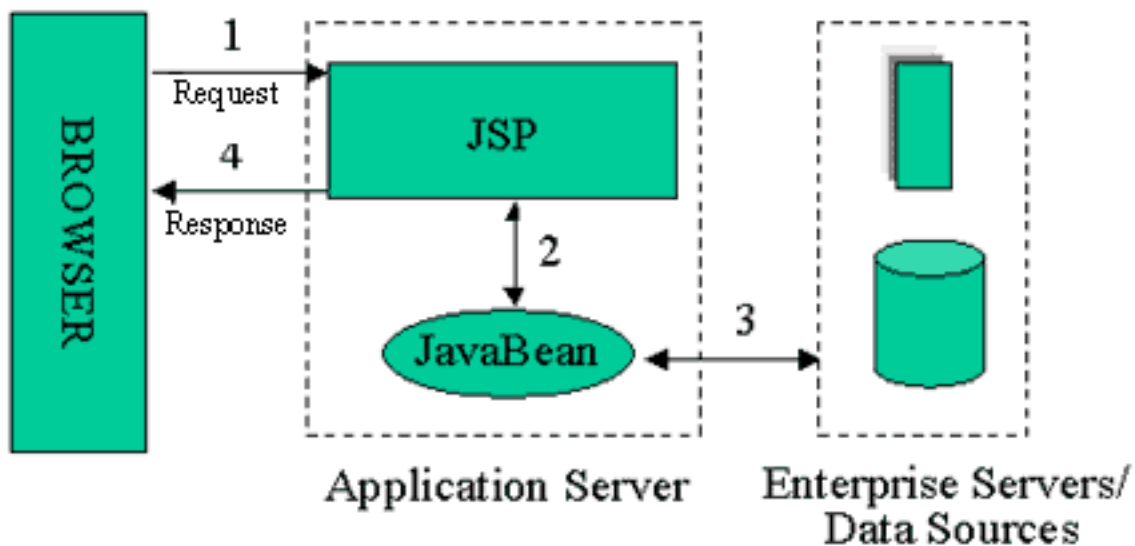


Рисунок 1. Архитектура JSP Модель 1.

Архитектура Модели 2 (рис.2), является гибридной технологией обработки динамического содержания, так как она комбинирует использование сервлетов и JSP. Это позволяет воспользоваться преимуществами обеих технологий, применяя JSP для реализации уровня представления и сервлеты для объемной обработки данных. В этой схеме сервлет действует как *контроллер* и отвечает за обработку запроса и создание

компонентов Java-beans, используемых JSP, а также в зависимости от действий пользователя принимает решение, какой JSP-странице перенаправить запрос. Обратите внимание, что непосредственно в JSP-странице нет никакой логики обработки информации; она просто отвечает за извлечение любых объектов или beans, которые, возможно, были предварительно созданы сервлетом, и получение динамического содержания от этого сервлета для включения в статические шаблоны. По-моему, этот подход в общем случае приводит к наиболее полному отделению представления от содержания, и позволяет упорядочить роли и обязанности программистов и дизайнеров страниц в вашей команде разработчиков. Фактически, чем более сложным является ваше приложение, тем больше будет выгода от использования Модели 2.

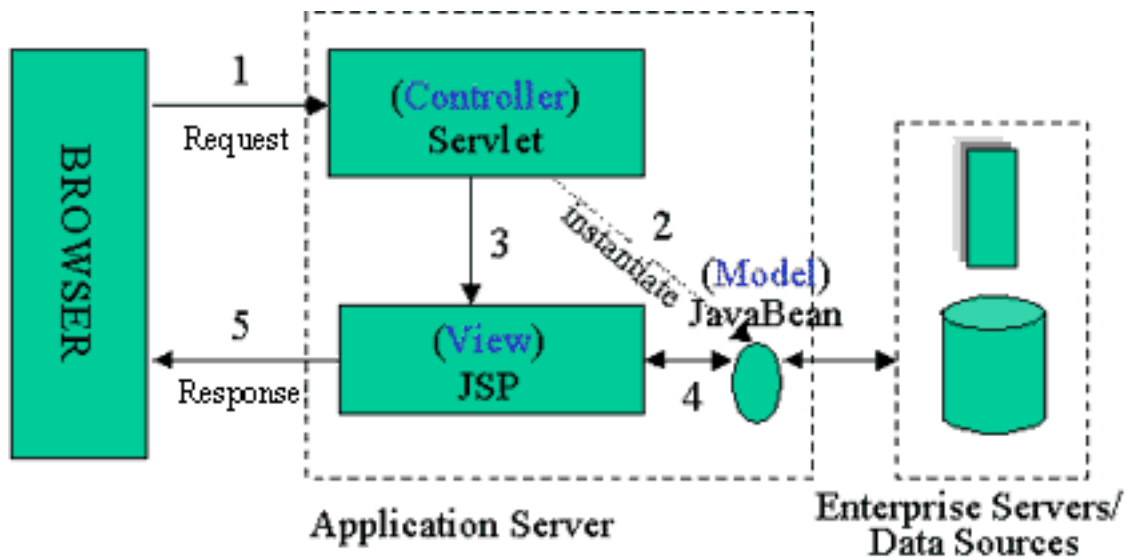


Рисунок 2. Архитектура JSP Модели 2.

Чтобы уяснить концепции, лежащие в основе архитектуры Модели 2, давайте рассмотрим ее во всех подробностях на примере типового электронного музыкального магазина "Music Without Borders".

3. Реализация "Music Without Borders"

Основное представление (*view*, — в модели Model-View-Controller), для нашего электронного магазина, обеспечивается JSP-страницей `EShop.jsp` (листинг 1). Вы можете заметить, что эта страница занимается исключительно отображением

Архитектура "Модели 2" для JavaServer Pages

основного пользовательского интерфейса приложения, и не выполняет вообще никакой обработки данных — это оптимальный сценарий использования JSP . Также, обратите внимание, что другая JSP страница, Cart.jsp (листинг 2), включена в EShop.jsp посредством директивы `<jsp:include page="Cart.jsp" flush="true" />`.

Листинг 1: EShop.jsp

```
<%@ page session="true" %>
<html>
<head>
  <title>Music Without Borders</title>
</head>
<body bgcolor="#33CCFF">
  <font face="Times New Roman,Times" size="+3">
    Music Without Borders
  </font>
  <hr><p>
  <center>
    <form name="shoppingForm"
      action="/examples/servlet/ShoppingServlet"
      method="POST">
    <b>CD:</b>
    <select name=CD>
    <option>Yuan | The Guo Brothers | China | $14.95</option>
    <option>Drums of Passion | Babatunde Olatunji | Nigeria | $16.95</option>
    <option>Kaira | Tounami Diabate| Mali | $16.95</option>
    <option>The Lion is Loose | Eliades Ochoa | Cuba | $13.95</option>
    <option>Dance the Devil Away | Outback | Australia | $14.95</option>
    <option>Record of Changes | Samulnori | Korea | $12.95</option>
    <option>Djelika | Tounami Diabate | Mali | $14.95</option>
    <option>Rapture | Nusrat Fateh Ali Khan | Pakistan | $12.95</option>
    <option>Cesaria Evora | Cesaria Evora | Cape Verde | $16.95</option>
    <option>Ibuki | Kodo | Japan | $13.95</option>
    </select>
    <b>Quantity: </b><input type="text" name="qty" SIZE="3" value=1>
    <input type="hidden" name="action" value="ADD">
    <input type="submit" name="Submit" value="Add to Cart">
  </form>
  </center>
```

```
<p>
<jsp:include page="Cart.jsp" flush="true" />
</body>
</html>
```

Листинг 2: Cart.jsp

```
<%@ page session="true" import="java.util.*, shopping.CD" %>
<%
    Vector buylist = (Vector) session.getValue("shopping.shoppingcart");
    if (buylist != null && (buylist.size() > 0)) {
%>
<center>
<table border="0" cellpadding="0" width="100%" bgcolor="#FFFFFF">
    <tr>
        <td><b>ALBUM</b></td>
        <td><b>ARTIST</b></td>
        <td><b>COUNTRY</b></td>
        <td><b>PRICE</b></td>
        <td><b>QUANTITY</b></td>
        <td></td>
    </tr>
    <%
        for (int index=0; index < buylist.size();index++) {
            CD anOrder = (CD) buylist.elementAt(index);
        %>
    <tr>
        <td><b><%= anOrder.getAlbum() %></b></td>
        <td><b><%= anOrder.getArtist() %></b></td>
        <td><b><%= anOrder.getCountry() %></b></td>
        <td><b><%= anOrder.getPrice() %></b></td>
        <td><b><%= anOrder.getQuantity() %></b></td>
        <td>
            <form name="deleteForm"
                action="/examples/servlet/ShoppingServlet"
                method="POST">
                <input type="submit" value="Delete">
                <input type="hidden" name="delindex" value='<%= index %>'>
                <input type="hidden" name="action" value="DELETE">
            </form>
        </td>
```

Архитектура "Модели 2" для JavaServer Pages

```
</tr>
  <% } %>
</table>
<p>
<form name="checkoutForm"
  action="/examples/servlet/ShoppingServlet"
  method="POST">
  <input type="hidden" name="action" value="CHECKOUT">
  <input type="submit" name="Checkout" value="Checkout">
</form>
</center>
<% } %>
```

В вышеприведенном примере, `Cart.jsp` представляет содержимое покупательской корзины, сохраняемой в HTTP-сессии пользователя (`Http-session`). Эта корзина является *моделью* в нашей архитектуре MVC . Взгляните на скриптлет в начале `Cart.jsp`:

```
<%
  Vector buylist = (Vector) session.getValue("shopping.shoppingcart");
  if (buylist != null && (buylist.size() > 0)) {
%>
```

В основном, этот скриптлет извлекает корзину из сессии. Если корзина пуста или еще не создана, он ничего не отображает; таким образом, когда пользователь в первые обращается к приложению, корзина выглядит так, как показано на рис. 3.

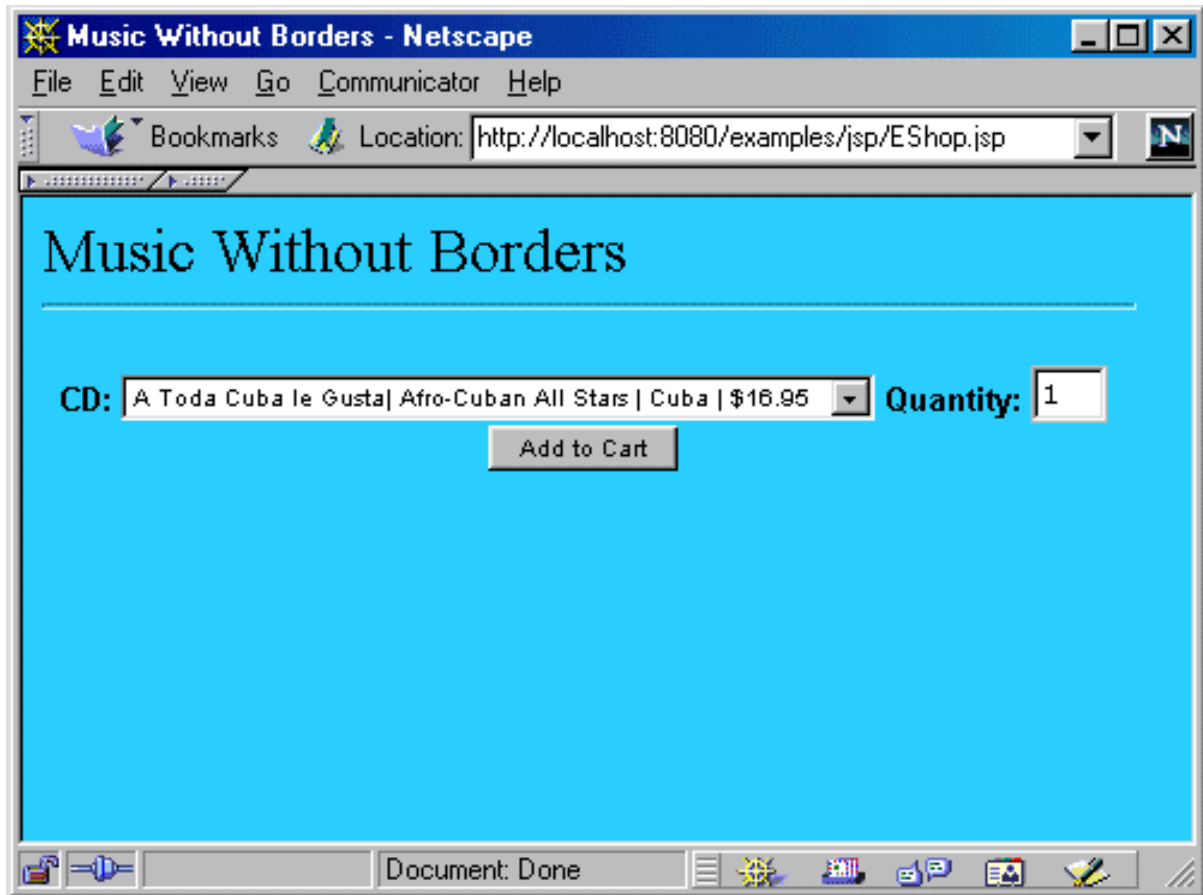


Рисунок 3. Music Without Borders, основной вид.

Если корзина не пуста, то выбранные элементы извлекаются из корзины по одному, что демонстрируется следующим скриптом:

```
<%  
    for (int index=0; index < buylist.size(); index++) {  
        CD anOrder = (CD) buylist.elementAt(index);  
    }  
%>
```

После того, как переменные, описывающие содержимое корзины, были созданы, они просто вставляются в статический HTML-шаблон в виде выражений JSP. Рис. 4 показывает внешний вид приложения после того, как пользователь поместил некоторые предметы в корзину.



Рисунок 4. Music Without Borders, корзина покупателя.

Важная деталь, которую следует отметить, состоит в том, что обработка всех действий, выполненных или в пределах `Eshop.jsp` или `Cart.jsp` производится сервлетом `ShoppingServlet.java`, играющим роль *контроллера* (листинг 3).

Листинг 3: `ShoppingServlet.java`

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import shopping.CD;
public class ShoppingServlet extends HttpServlet {
    public void init(ServletConfig conf) throws ServletException {
        super.init(conf);
    }
}
```

```
}
public void doPost (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    HttpSession session = req.getSession(false);
    if (session == null) {
        res.sendRedirect ("http://localhost:8080/error.html");
    }
    Vector buylist=
        (Vector) session.getValue("shopping.shoppingcart");
    String action = req.getParameter("action");
    if (!action.equals("CHECKOUT")) {
        if (action.equals("DELETE")) {
            String del = req.getParameter("delindex");
            int d = (new Integer(del)).intValue();
            buylist.removeElementAt(d);
        } else if (action.equals("ADD")) {
            //покупался ли ранее этот cd?
            boolean match=false;
            CD aCD = getCD(req);
            if (buylist==null) {
                //добавить первый cd в корзину
                buylist = new Vector(); //первый заказ
                buylist.addElement(aCD);
            } else { // не первая покупка
                for (int i=0; i< buylist.size(); i++) {
                    CD cd = (CD) buylist.elementAt(i);
                    if (cd.getAlbum().equals(aCD.getAlbum())) {
                        cd.setQuantity(cd.getQuantity() +aCD.getQuantity());
                        buylist.setElementAt(cd,i);
                        match = true;
                    } //конец блока If
                } // конец блока For
            }
            if (!match)
                buylist.addElement(aCD);
        }
    }
    session.putValue("shopping.shoppingcart", buylist);
    String url="/jsp/shopping/EShop.jsp";
    ServletContext sc = getServletContext();
    RequestDispatcher rd = sc.getRequestDispatcher(url);
}
```

```
rd.forward(req, res);
} else if (action.equals("CHECKOUT")) {
    float total =0;
    for (int i=0; i< buylist.size();i++) {
        CD anOrder = (CD) buylist.elementAt(i);
        float price= anOrder.getPrice();
        int qty = anOrder.getQuantity();
        total += (price * qty);
    }
    total += 0.005;
    String amount = new Float(total).toString();
    int n = amount.indexOf('.');
    amount = amount.substring(0,n+3);
    req.setAttribute("amount", amount);
    String url="/jsp/shopping/Checkout.jsp";
    ServletContext sc = getServletContext();
    RequestDispatcher rd = sc.getRequestDispatcher(url);
    rd.forward(req, res);
}
}
private CD getCD(HttpServletRequest req) {
    //представьте, что это скриптлет... ужасно, не правда ли?
    String myCd = req.getParameter("CD");
    String qty = req.getParameter("qty");
    StringTokenizer t = new StringTokenizer(myCd,"|");
    String album= t.nextToken();
    String artist = t.nextToken();
    String country = t.nextToken();
    String price = t.nextToken();
    price = price.replace('$',' ').trim();
    CD cd = new CD();
    cd.setAlbum(album);
    cd.setArtist(artist);
    cd.setCountry(country);
    cd.setPrice((new Float(price)).floatValue());
    cd.setQuantity((new Integer(qty)).intValue());
    return cd;
}
}
```

Каждый раз, когда пользователь добавляет элемент внутри EShop.jsp, запрос

посылается сервлету-контроллеру. Он, в свою очередь, определяет какое действие необходимо выполнить, и затем обрабатывает параметры запроса для добавляемого элемента. Затем он инстанцирует новый CD bean (листинг 4), описывающий выбор пользователя, и обновляет объект, представляющий корзину, перед помещением этого объекта обратно в сессию.

Листинг 4: CD.java

```
package shopping;
public class CD {
    String album;
    String artist;
    String country;
    float price;
    int quantity;
    public CD() {
        album="";
        artist="";
        country="";
        price=0;
        quantity=0;
    }
    public void setAlbum(String title) {
        album=title;
    }
    public String getAlbum() {
        return album;
    }
    public void setArtist(String group) {
        artist=group;
    }
    public String getArtist() {
        return artist;
    }
    public void setCountry(String cty) {
        country=cty;
    }
    public String getCountry() {
        return country;
    }
}
```

```
public void setPrice(float p) {
    price=p;
}
public float getPrice() {
    return price;
}
public void setQuantity(int q) {
    quantity=q;
}
public int getQuantity() {
    return quantity;
}
}
```

Обратите внимание, что мы также включили дополнительную обработку в сервлет, для того, чтобы в случае повторного выбора одного и того же CD, он просто увеличивал счетчик для данного CD bean в корзине. Сервлет-контроллер также обрабатывает другие действия, запущенные в `Cart.jsp`, такие как удаление пользователем предметов из корзины, или окончательное оформление покупки — "кассовый контроль" (checkout). Заметьте, что контроллер всегда имеет полный контроль над тем, какие ресурсы должны быть задействованы при выполнении определенных действий. Например, изменения состояния корзины, такие как добавление или удаление элементов, заставляют сервлет-контроллер после обработки перенаправлять запрос к странице `Eshop.jsp`. Это, в свою очередь, заставляет страницу заново отобразить основное представление и модифицированное содержание корзины. Если пользователь решает оплатить покупки, запрос перенаправляется странице `Checkout.jsp` (листинг 5), как показано ниже:

```
String url="/jsp/shopping/Checkout.jsp";
ServletContext sc = getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher(url);
rd.forward(req, res);
```

Листинг 5: Checkout.jsp

```
<%@ page session="true"
import="java.util.*, shopping.CD" %>
<html>
<head>
<title>Music Without Borders Checkout</title>
```

```
</head>
<body bgcolor="#33CCFF">
  <font face="Times New Roman,Times" size=+3>
    Music Without Borders Checkout
  </font>
  <hr><p>
  <center>
  <table border="0"
  cellpadding="0"
  width="100%" bgcolor="#FFFFFF">
  <tr>
  <td><b>ALBUM</b></td>
  <td><b>ARTIST</b></td>
  <td><b>COUNTRY</b></td>
  <td><b>PRICE</b></td>
  <td><b>QUANTITY</b></td>
  <td></td>
  </tr>
  <%
    Vector buylist = (Vector) session.getValue("shopping.shoppingcart");
    String amount = (String) request.getAttribute("amount");
    for (int i=0; i < buylist.size();i++) {
      CD anOrder = (CD) buylist.elementAt(i);
    %>
  <tr>
  <td><b><%= anOrder.getAlbum() %></b></td>
  <td><b><%= anOrder.getArtist() %></b></td>
  <td><b><%= anOrder.getCountry() %></b></td>
  <td><b><%= anOrder.getPrice() %></b></td>
  <td><b><%= anOrder.getQuantity() %></b></td>
  </tr>
  <%
    }
    session.invalidate();
  %>
  <tr>
  <td> </td>
  <td> </td>
  <td><b>TOTAL</b></td>
  <td><b>${<%= amount %></b></td>
```

```
<td> </td>
</tr>
</table>
<p>
<a href="/examples/jsp/shopping/EShop.jsp">Shop some more!</a>
</center>
</body>
</html>
```

Checkout.jsp просто получает корзину из сессии и затем отображает выбранные элементы и их общую стоимость. Рис. 5 показывает как это выглядит на стороне клиента. Когда пользователь оформляет покупку, важно избавиться от объектов в сессии. Для этого в конце страницы вызывается `session.invalidate()`. Это необходимо по двум причинам. Во-первых, если не закрыть сессию, корзина пользователя не будет проинициализирована повторно ; если пользователь в этом случае попытается посетить магазин вновь после "кассового контроля", его корзина будет содержать предметы, которые он уже купил. Во-вторых, если пользователь просто уйдет с сайта после контроля, объект, сохраненный в сессии, не будет собран сборщиком мусора и продолжит использовать ценные ресурсы системы, пока сервер не уничтожит сессию по тайм-ауту (по умолчанию значение тайм-аута для пользовательских сессий — приблизительно 30 минут). Это может быстро вести к исчерпанию доступной памяти в крупномасштабной системе. Конечно, все мы знаем, что бывает с приложением, которому не хватает системных ресурсов!

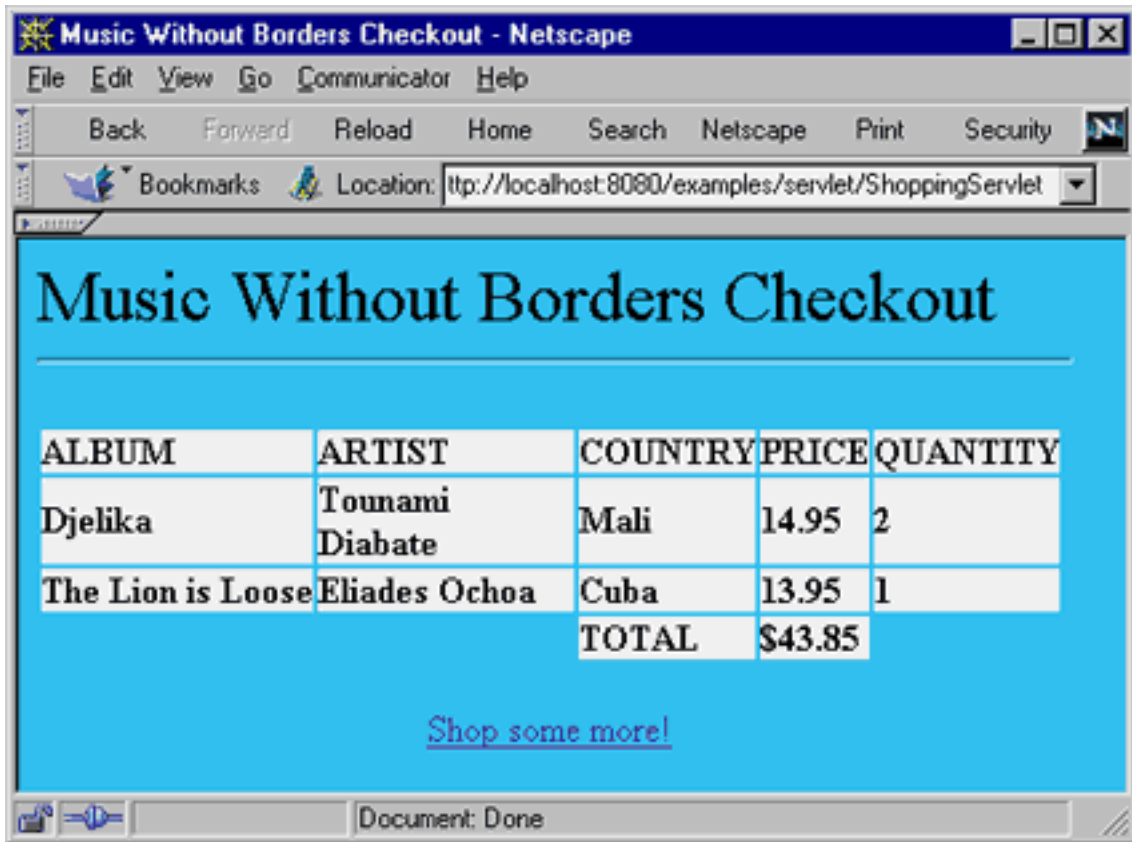


Рисунок 5. Music Without Borders, кассовый контроль.

Обратите внимание, что все ресурсы для этого приложения связаны с сессией, так как в ней сохраняется модель. Следовательно, необходимо запретить пользователю обращаться к контроллеру непосредственно, даже по ошибке. Это можно сделать, осуществляя перенаправление клиента на страницу ошибки (листинг 6), в случае если контроллер обнаруживает отсутствие открытой сессии.

Листинг 6: error.html

```
<html>
<body>
<h1>
  Sorry, there was an unrecoverable error! <br>
  Please try <a href="/examples/jsp/shopping/EShop.jsp">again</a>.
</h1>
</body>
```

</html>

4. Установка и настройка "Music Without Borders"

Я буду предполагать, что Вы используете самую последнюю версию JavaServer Web Development Kit (JSWDK) от Sun для выполнения этого примера. Если это не так, обратитесь к разделу [Ресурсы](#), чтобы узнать, где его можно получить. Пусть, например, сервер установлен в директории `\jswdk-1.0.1` (местоположение по умолчанию в среде Microsoft Windows), разместите файлы приложения "Music Without Borders" следующим образом:

- Создайте каталог `shopping` внутри `\jswdk-1.0.1\examples\jsp`
- Скопируйте `EShop.jsp` в директорию `\jswdk-1.0.1\examples\jsp\shopping`
- Скопируйте `Cart.jsp` в директорию `\jswdk-1.0.1\examples\jsp\shopping`
- Скопируйте `Checkout.jsp` в директорию `\jswdk-1.0.1\examples\jsp\shopping`
- Скомпилируйте файлы с расширением `.java`, набрав в командной строке `javac *.java`
- Скопируйте `ShoppingServlet.class` в директорию `\jswdk-1.0.1\webpages\Web-Inf\servlets`
- Создайте каталог `shopping` внутри `\jswdk-1.0.1\examples\Web-Inf\jsp\beans`
- Скопируйте `CD.class` в директорию `\jswdk-1.0.1\examples\Web-Inf\jsp\beans\shopping`
- Скопируйте `error.html` в директорию `\jswdk-1.0.1\webpages`
- Как только ваш сервер запустится, Вы сможете обратиться к приложению, по следующему URL: <http://localhost:8080/examples/jsp/shopping/EShop.jsp>

5. Заключение

В этом примере, мы подробно рассмотрели преимущества архитектуры Модели 2, позволяющие обеспечить управляемость и гибкость системы. В частности мы увидели, как лучшие особенности сервлетов и JSP-страниц могут использоваться для разделения представления и содержания. В случае правильной реализации, Модель 2

должна привести к сосредоточению всей логики обработки у сервлета-контроллера, оставляя JSP страницы отвечать только за представление. Однако, обратная сторона использования Модели 2 — ее сложность. Следовательно, использование Модели 1 для более простых приложений также возможно.

6. Об авторе



[Говинд Сешадри \(Govind Seshadri\)](#) Enterprise Java Guru на сайте

jGuru.com и автор книги *Enterprise Java Computing — Applications and Architecture* (Cambridge University Press — 1999). Вы можете узнать о [Говинде](#) посетив jGuru.com.

7. Ресурсы

- Последняя версия JavaServer Web Development Kit:
<http://java.sun.com/products/jsp/download.html>
- Исходный код этой статьи в виде JAR-файла:
<http://www.javaworld.com/jw-12-1999/ssj/ShoppingCart.jar>

Reprinted with permission from the December 1999 edition of JavaWorld magazine. Copyright © ITworld.com, Inc., an IDG Communications company.

View the original article at: <http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html>

[Перевод на русский © Антон Никитин, 2000](#)