

Языковая поддержка для вебсайтов с использованием JSP

Создайте вебсайт, который использует родной для пользователя язык.

Server-Side Java

Гоувайнд Сишадри

Компании более не могут позволить себе поддерживать только англоязычные сайты электронной коммерции. Каждый пользователь, неспособный прочитать текст на англоязычном сайте- это потерянный клиент. Вопрос языковой поддержки для вебсайтов стал достаточно актуальным в последнее время. В этой статье Гоувайнд Сишадри обсуждает создание глобальных страниц JSP. Используя некоторые классы из Java 2 Software Development Kit (SDK), вы можете создавать веб-приложения, способные динамически изменять свое отображение и адаптироваться к различным языкам и странам, где они могут просматриваться. Эта статья подразумевает, что читатель знаком с основами программирования JSP. (3000 слов)

Обычно, большинство вебсайтов с которыми вы встречаетесь созданы для определенного набора символов- локали (locale), которую можно определить как географическую область или политический субъект, разделяющие общие язык, культуру и обычаи. Хотя это может быть неважно для большинства людей, это может создавать ряд интересных проблем при создании вебсайтов. По существу, электронная коммерция создана для того, чтобы развивать торговлю поверх национальных, языковых или культурных границ. В то время как Веб открывает бизнес для действительно международной аудитории, веб-компании сталкиваются с вполне вероятным сценарием непонимания неанглоязычным пользователем содержания сайта. Как уже поняло много компаний, каждый посетитель ушедший с англоязычного сайта- это потеря потенциального покупателя.

Существует ли простое, легко настраиваемое решение для этой проблемы, которая мешает вам поддерживать несколько версий сайта? Если динамическое содержимое сайта создается с помощью технологий Java, то да! Дело в том, что Java содержит внутреннюю поддержку для локаль-независимых приложений с помощью различных классов, поддерживающих *internationalization*, находящихся в пакетах `java.util`, `java.io` и `java.text`. Internationalization- это название для процесса разработки, когда информация, специфичная для страны, языка и культуры отделяется и упаковывается во внешние файлы или объекты. Соответственно, переделка приложения для совместимости с новым языком или страной не требует переписывания большей части логики программы, скорее, она требует создания дополнительной версии внешних файлов, специфичной для данной локали. Такой процесс создания локаль-специфичных вхождений (как файлов, так и объектов) вместе с переведенным текстом называется *локализацией (localization)* [часто можно встретить сокращение *l10n* по-анalogии с *i18n* для *internationalization* — Прим. переводч.].

Вебсайты, основанные на JavaServer Pages (JSPs) легче поддаются интернационализации, чем созданные с помощью сервлетов (servlets). Это связано с тем, что эта технология способствует разделению представления и дизайна от логики приложения. Теперь давайте исследуем эти возможности подробнее, создав глобальную версию онлайн-магазина "Музыка без границ" (Music Without Borders), который был показан в моей ранней статье об архитектуре JSP (см. раздел [Ресурсы](#)). Этот онлайн-магазин основан на архитектуре Модель-Вид-Контроллер (Model-View-Controller (MVC)), для того, чтобы максимально разделить дизайн и содержание.

Несмотря на то, что интернационализация веб-приложения, основанного на JSP, не является чем-то особенно сложным, лучше начать этот процес еще на стадии дизайна. К концу разработки вся локаль-специфичная текстовая информация, содержащаяся в пользовательском интерфейсе- заголовки и окончания страниц, сообщения для пользователя, надписи на кнопках, даты, названия денежных единиц, и так далее, должна быть выделена во внешние файлы. Такие вхождения, содержащие разную информацию для разных локалей, называют *ресурсами (resources)*, а ресурсы, сгруппированные во внешнем файле класса или ресурса- *связками ресурсов (resource bundle)* или просто *связками (bundle)*.

Java 2 SDK предоставляет абстрактный класс `java.util.ResourceBundle` для работы с ресурсами, также как и его подклассы `ListResourceBundle` и `PropertyResourceBundle`. Связка для поддерживаемой локали может быть определена либо как классфайл (используя `ListResourceBundle`) или как файл свойств (используя `PropertyResourceBundle`). Оба эти механизма позволяют вашему приложению получать доступ к значениям ресурсов по их именам, точно также как в объекте `Hashtable`. Несмотря на то, что вы можете сохранить объекты любого типа как значения, используя классфайлы, большинство веб-приложений используют файлы свойств, так как в большинстве случаев нужно настраивать только строки. Соответственно, я сосредоточусь на использовании файлов свойств.

1. Разработка глобальных веб-приложений

В этом примере, онлайн-магазин "Музыка без границ" был локализован для поддержки пользователей из США, Германии, Швеции и Франции. Листинги 1 и 2 показывают файлы свойств для поддержки локалей США и Франции (вы найдете исходный код для поддержки Германии и Швеции в [Ресурсах](#).) Как вы видите, файл свойств изолирует все элементы интерфейса пользователя от собственно страниц JSP. Файлы свойств должны следовать определенным соглашениям об именах. Суффикс должен всегда быть `properties`. Также, так как локаль в Java обычно задается с помощью кода страны и/или кода языка и дополнительного варианта, имя связки должно соответствовать имени локали. Например, все следующие префиксы связок верны:

```
BundleName + "_" + localeLanguage + "_" + localeCountry +
                                         "_" + localeVariant
BundleName + "_" + localeLanguage + "_" + localeCountry
BundleName + "_" + localeLanguage
BundleName
```

Заметьте, что файл свойств, обслуживающий локаль для Франции (показан в листинге 2) был назван `Message_fr_FR.properties`. Связки, предназначенные для определенной страны, должны содержать как код языка, так и код страны в префиксе. (Также, я мог-бы использовать дополнительный специфичный для приложения/ОС вариант и дополнительно уточнить файл как `Message_fr_FR_MAC.properties` — связку для всех франкоговорящих пользователей Макинтош во Франции. Но я

воздержусь!) Соответственно, связка содержащая немецкие метки называется `Message_de_DE.properties`, и так далее. [Ресурсы](#) содержат ссылки для получения корректных кодов языка и страны. Для любопытствующих, следующий код может создать классфайл, служащий для того-же, что и файл свойств из Листинга 2:

```
public class Message_fr_FR extends ListResourceBundle {
    public Object[][] getContents() {
        return contents;
    }

    static final Object[][] contents = {
        {"main.title", "Musique sans frontieres"},
        {"main.subhead", "Sons du village global"},
        {"main.addLabel", "Ajouter"},
        . . .
        {"cd.quantityLabel", "Quantitй"}
    }
}
```

Вы можете спросить, почему файл свойств, содержащий английские метки (показан в Листинге 1) не содержит кода страны или языка в префиксе. Я мог бы назвать его `Message_en_US.properties`, но я выбрал просто имя `Message.properties`. Любая связка, которая не содержит кода страны или языка, обрабатывается как связка по-умолчанию и служит особой цели: она выбирается как последняя возможность, если нет соответствующей связки для требуемой локали. Заметьте, что ключи остаются одинаковыми для всех локалей, меняются только значения. Теперь, когда мы изолировали элементы интерфейса и локализовали их в файлах свойств, я покажу вам, как использовать их в страницах JSP:

Листинг 1: Message.properties

```
main.title=Music Without Borders
main.subhead=Sounds from the Global Village
main.addLabel=Add
main.qtyLabel=Quantity
main.bgcolor=#33CCFF
cart.bgcolor=#FFFFFF
cart.delLabel=Delete
cart.checkoutLabel=Checkout
checkout.bgcolor=#33CCFF
```

Языковая поддержка для вебсайтов с использованием JSP

```
checkout.title=Music Without Borders Checkout
checkout.subhead=Thanks for your order!
checkout.totalLabel=Total
checkout.returnLabel=Shop some more!
dollar.exchRate=1.00
cd.albumLabel=Album
cd.artistLabel=Artist
cd.countryLabel=Country
cd.priceLabel=Price
cd.quantityLabel=Quantity
```

Листинг 2: Message_fr_FR.properties

```
main.title=Musique sans frontiirres
main.subhead=Sons du village global
main.addLabel=Ajouter
main.qtyLabel=Quantitй
main.bgcolor=#33CCFF
cart.bgcolor=#FFFFFF
cart.delLabel=Supprimer
cart.checkoutLabel=Passez a la caisse
checkout.bgcolor=#33CCFF
checkout.title= Caisse pour Musique sans frontiirres
checkout.subhead=Merci pour votre commande!
checkout.totalLabel=Total
checkout.returnLabel=Faites d'autres commandes!
dollar.exchRate=6.48
cd.albumLabel=Album
cd.artistLabel=Artiste
cd.countryLabel=Pays
cd.priceLabel=Prix
cd.quantityLabel=Quantitй
```

Страница JSP `i18nDemo.jsp` (показана в Листинге 3) работает как вход в онлайнный магазин. Ее главное задание- позволить пользователю выбрать соответствующий язык для просмотра сайта, как показано на Рисунок 1.



Рисунок 1. Вход в локализованное веб-приложение.

В Java, главный фокус для всей работы по интернационализации заключается в классе `java.util.Locale`, который инкапсулирует код страны, код языка и дополнительный код варианта. Класс `Locale` также содержит массу удобных констант и представляет все наиболее часто используемые локали. Рассмотрите следующий фрагмент кода, устанавливающий локаль для веб-приложения:

```
Locale locale=null;
if (lang.equals("German")) {
locale=Locale.GERMANY;
} else if (lang.equals("French")) {
locale=Locale.FRANCE;
} else if (lang.equals("Swedish")) {
locale=new Locale("sv","SE");
} else {
locale=Locale.US;
}
```

Тогда как экземпляр `Locale` обычно получают из преопределенной константы в

Языковая поддержка для вебсайтов с использованием JSP

классе `Locale`, он также может быть создан с использованием конструктора. Вы можете это сделать, задав код страны и языка (существует другая версия конструктора `Locale`, принимающая код дополнительного варианта). Код языка- это две буквы в нижнем регистре, а код страны- всегда две буквы в верхнем. (Как упоминалось выше, вы можете найти коды страны или языка в [Ресурсах](#)).

Так как я не нашел предустановленной константы для Швеции, я создал экземпляр `Locale` путем передачи кодов страны и языка:

```
locale=new Locale("sv", "SE");
```

Как только локаль создана, вы можете получить соответствующую связку:

```
ResourceBundle bundle =  
ResourceBundle.getBundle("Message", locale);
```

Вы можете хранить связки где угодно внутри `classpath`, в понимании сервера приложений и JSP. Вызов `getBundle()` начинает поиск связки после выделения кода языка (скажем, `xx`) и кода страны (`YY`) из объекта локали, переданного, как параметер. (Вы также можете вызвать `getBundle()` без передачи экземпляра `Locale`, в этом случае будет использована локаль по-умолчанию). Вы сначала проводите поиск класса `Message_xx_YY.class` внутри `classpath`. Если он отсутствует, поиск пытается найти `Message_xx.class` и потом `Message.class`. Если попытка неудачна, начинается поиск соответствующего файла свойств, начиная с `Message_xx_YY.properties`, переходя к `Message_xx.properties` и заканчивая `Message.properties`. Если и эта попытка неудачна, `getBundle()` выдает `MissingResourceException`.

Листинг 3: `i18nDemo.jsp`

```
<%@ page import="java.util.*" %>  
<%  
String lang = request.getParameter("lang");  
  
if (lang == null) {  
%>  
<html>  
<head>  
<title>  
Music Without Borders  
</title>
```

Языковая поддержка для вебсайтов с использованием JSP

```
</head>
<body bgcolor="#33CCFF">
<font face="Times New Roman,Times" size=+3>
Music Without Borders
</font>
<hr>
<p>
Please select a language:
<form action="il8nDemo.jsp" method="post">
English <input type="radio" name="lang" value="English" checked>
Deutsch <input type="radio" name="lang" value="German">
Franzais <input type="radio" name="lang" value="French">
<p>
<input type="submit" value="Continue">
</form>
</body>
</html>
<%
} else {
Locale locale=null;
if (lang.equals("German")) {
locale=Locale.GERMANY;
} else if (lang.equals("French")) {
locale=Locale.FRANCE;
} else if (lang.equals("Swedish")) {
locale=new Locale("sv","SE");
} else {
locale=Locale.US;
}
session.putValue("myLocale",locale);
ResourceBundle bundle =
ResourceBundle.getBundle("Message",locale);

for (Enumeration e = bundle.getKeys();e.hasMoreElements();) {
String key = (String)e.nextElement();
String s = bundle.getString(key);
session.putValue(key,s);
}
%>
<jsp:forward page="eshop.jsp" />
```

Языковая поддержка для вебсайтов с использованием JSP

```
<%  
}  
%>
```

После получения связки, вы получаете ресурс для локали вместе с соответствующими значениями и помещаете их в сессию как:

```
for (Enumeration e = bundle.getKeys(); e.hasMoreElements();) {  
String key = (String)e.nextElement();  
String s = bundle.getString(key);  
session.putValue(key, s);  
}
```

Как только сессия создана со всей необходимой локализованной информацией, передается запрос на `eshop.jsp` (показан в Листинге 4), который создает главное окно онлайн-магазина.

Как и любой хорошо-написанный JSP, `eshop.jsp` имеет дело главным образом с представлением интерфейса клиенту. На самом деле, единственная работа, которую он производит, это форматирование поля с ценами в соответствии с выбранной локалью. Посмотрите на метод `computePrice()`, объявленный на странице, который помогает отобразить названия валют и отформатировать их в соответствии с локалью. Вся работа выполняется объектом `java.text.NumberFormat` для локали, полученной при вызове `getCurrencyInstance()`. При вызове метода `format()` возвращается строка `String`, которая содержит правильно отформатированный номер и название валюты. Заметьте, что я также использовал ресурс `dollar.exchRate` для преобразования входного значения в долларах в соответствующий эквивалент в другой валюте.

Вы можете видеть, что большинство элементов GUI, включая сообщения в заголовках, надписи на кнопках, и тд., зависят от локали. Они устанавливаются во время выполнения при взаимодействии сессии с соответствующим ресурсом:

```
<title>  
<%=session.getValue("main.title")%>  
</title>
```

Листинг 4: `eshop.jsp`

```
<%@ page import="java.util.*, shopping.i18n.CD, java.text.*" %>  
  
<%!
```

```
public String computePrice(double price, HttpSession session) {

String exchangeRate =
    (String)session.getValue("dollar.exchRate");
price*= new Double(exchangeRate).doubleValue();
NumberFormat form =
    NumberFormat.getCurrencyInstance((Locale)
session.getValue("myLocale"));
return form.format(price);

}
%>

<html>
<head>
<title>
<%=session.getValue("main.title")%>
</title>

</head>
<body bgcolor='<%=session.getValue("main.bgcolor")%>'>
<font face="Times New Roman,Times" size=+3>
<%=session.getValue("main.title")%>
</font>
<br>
<em>
<%=session.getValue("main.subhead")%>
</em>
<hr>
<p>
<center>
<form name="shoppingForm" action="main.jsp" method="post"
CD:
<select name=CD>
<option>A Toda Cuba le Gusta| Afro-Cuban All Stars | Cuba |
<%=computePrice(16.95,session)%></option>
<option>Vujicsics | Vujicsics | Croatia |
<%=computePrice(14.95,session)%></option>
<option>Kaira | Tounami Diabate| Mali |
<%=computePrice(16.95,session)%></option>
```

Языковая поддержка для вебсайтов с использованием JSP

```
<option>The Lion is Loose | Eliades Ochoa | Cuba |
<%=computePrice(13.95,session)%></option>
<option>Dance the Devil Away | Outback | Australia |
<%=computePrice(14.95,session)%></option>
<option>Record of Changes | Samulnori | Korea |
<%=computePrice(12.95,session)%></option>
<option>Djelika | Tounami Diabate | Mali |
<%=computePrice(14.95,session)%></option>
<option>Wimme | Wimme | Finland |
<%=computePrice(12.95,session)%></option>
<option>Cesaria Evora | Cesaria Evora | Cape Verde |
<%=computePrice(16.95,session)%></option>
<option>Ibuki | Kodo | Japan |
<%=computePrice(13.95,session)%></option>
</select>
<%=session.getValue("main.qtyLabel")%>:
<input type="text" name="qty" size="3" value="1">
<input type="hidden" name="action" value="ADD">
<input type="submit" name="Submit"
value='<%=session.getValue("main.addLabel")%>'>
</form>
</center>
<p>
<jsp:include page="cart.jsp" flush="true" />
</body>
</html>
```

Также, обратите внимание, что другая страница JSP, `cart.jsp` (показана в Листинге 5), включена в `eshop.jsp` через директиву:

```
<jsp:include page="cart.jsp" flush="true" />
```

Здесь `cart.jsp` управляет представлением корзины покупок, реализованной как объект `Vector`. Посмотрите на скриплет вначале `cart.jsp`:

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

В принципе, скриплет выделяет данные корзины покупок из сессии. Если корзина пуста или еще не создана, она ничего не показывает; следовательно, если пользователь

обращается к приложению в первый раз, он получает страницу, как показано на Рисунок 2, если предположить, что пользователь предпочел французский вариант сайта.

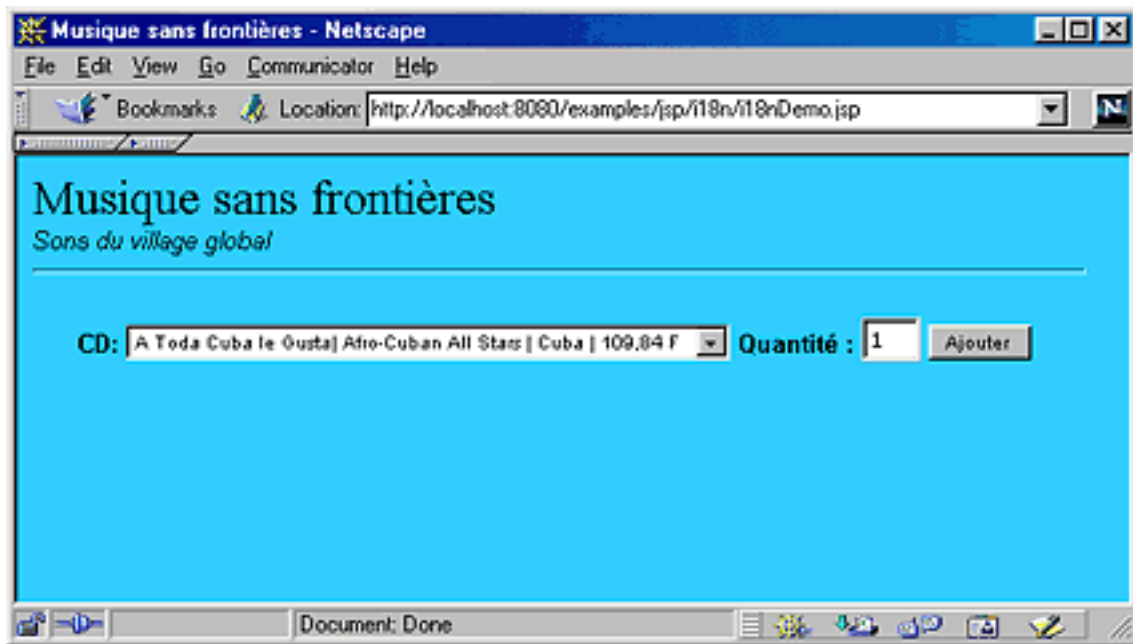


Рисунок 2. Главное окно, французская локаль.

Листинг 5: cart.jsp

```
<%@ page import="java.util.*, shopping.i18n.CD" %>

<%
Vector buylist = (Vector) session.getValue("shopping.shoppingcart");
if (buylist != null && (buylist.size() >0)) {
%>
<center>
<table border=0 cellpadding=0 width=100%
bgcolor='<%=session.getValue("cart.bgcolor")%>'>
<tr>
<td><%=session.getValue("cd.albumLabel")%></td>

<td><%=session.getValue("cd.artistLabel")%></td>
<td><%=session.getValue("cd.countryLabel")%></td>
<td><%=session.getValue("cd.priceLabel")%></td>
```

Языковая поддержка для вебсайтов с использованием JSP

```
<td><%=session.getValue("cd.quantityLabel")%></td>
<td></td>
</tr>

<%
for (int index=0; index< buylist.size();index++) {
CD anOrder = (CD) buylist.elementAt(index);
%>
<tr>
<td><%=anOrder.getAlbum()%></td>
<td><%=anOrder.getArtist()%></td>
<td><%=anOrder.getCountry()%></td>
<td><%=anOrder.getPrice()%></td>
<td><%=anOrder.getQuantity()%></td>
<td>
<form name="deleteForm" action="main.jsp" method="post">
<input type="submit"
value='<%=session.getValue("cart.delLabel")%>'>
<input type="hidden" name=delindex value='<%=index%>'>
<input type="hidden" name="action" value="DELETE">
</form>
</td>
</tr>
<% } %>
</table>
<p>
<form name="checkoutForm" action="main.jsp" method="post">
<input type="hidden" name="action" value="CHECKOUT">
<input type="submit" name="Checkout"
value='<%=session.getValue("cart.checkoutLabel")%>'>
</form>
</center>
<% } %>
```

Если корзина не пуста, выбранные CD выбираются из нее по-одному, как показано в этом скрипте:

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

%>

После создания переменной, описывающей CD, она просто вставляется в статический HTML-шаблон, используя выражения JSP. Вы можете избежать большого количества кода в скрипте, взаимодействуя с бинами (beans) с проиндексированными свойствами, реализовав цикл со своими метками. Рисунок 3 показывает шведский вариант страницы после того, как пользователь добавил несколько покупок в корзину. Сравните с английским вариантом на Рисунок 4. Заметьте, как легко проходит локализация при использовании той-же логики приложения.



Рисунок 3. Корзина пользователя, шведский вариант.

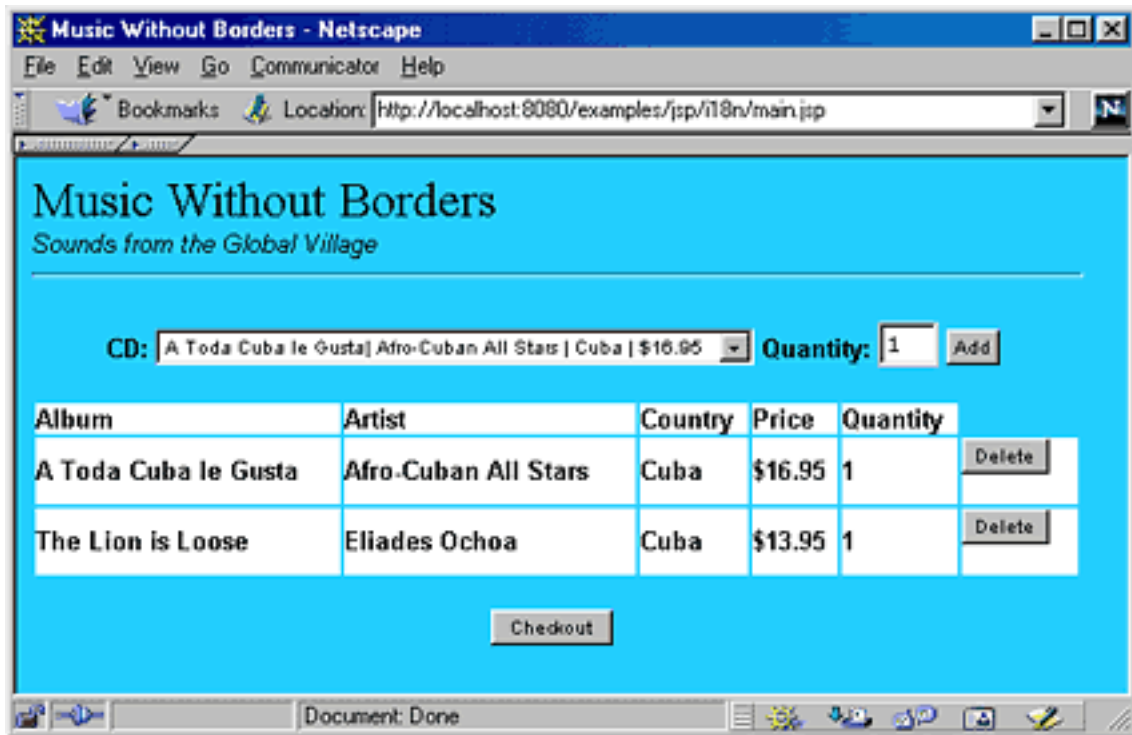


Рисунок 4. Корзина пользователя, английский вариант.

Тут важно заметить, что обработка всех действий в `eshop.jsp` или в `cart.jsp` производится в *контроллере*- JSP-странице `main.jsp`, показанной в Листинге 6.

Листинг 6: `main.jsp`

```
<%@ page import="java.util.*, java.text.*, shopping.i18n.CD" %>
<%
String action = request.getParameter("action");
if (action == null) {
%>
<jsp:forward page="i18nDemo.jsp" />
<%
} else {
Vector buylist=
(Vector)session.getValue("shopping.shoppingcart");
if (!action.equals("CHECKOUT")) {
if (action.equals("DELETE")) {
String del = request.getParameter("delindex");
int delIndex = (new Integer(del)).intValue();
```

```
buylist.removeElementAt(delIndex);
} else if (action.equals("ADD")) {
//any previous buys of same cd?
boolean match=false;
%>
<jsp:useBean id="aCD" class="shopping.i18n.CD" scope="page">
<% aCD.setCDProperties(request); %>
</jsp:useBean>
<%
if (buylist==null) {
//add first cd to the cart
buylist = new Vector(); //first order
buylist.addElement(aCD);
} else { // not first buy
for (int i=0; i< buylist.size();i++) {
CD cd = (CD) buylist.elementAt(i);
if (cd.getAlbum().equals(aCD.getAlbum())) {
int tmpQty =
(new Integer(cd.getQuantity())).intValue() +

(new Integer(aCD.getQuantity())).intValue();
cd.setQuantity(new Integer(tmpQty).toString());
buylist.setElementAt(cd,i);
match = true;
} //end of if name matches
} // end of for
if (!match) buylist.addElement(aCD);
} //end of not first buy
} //end of action==ADD
session.putValue("shopping.shoppingcart", buylist);
%>
<jsp:forward page="eshop.jsp" />

<%
} else { // if checkout
NumberFormat nFormat = NumberFormat.getCurrencyInstance(
(Locale)session.getValue("myLocale"));
float total =0;
for (int i=0; i< buylist.size();i++) {
CD anOrder = (CD) buylist.elementAt(i);
```

Языковая поддержка для вебсайтов с использованием JSP

```
Number n = nFormat.parse(anOrder.getPrice().trim());
float price= n.floatValue();
int qty = (new Integer(anOrder.getQuantity())).intValue();
total += (price * qty);
}
String amountDue=nFormat.format(total);
request.setAttribute("amountDue", amountDue);
%>
<jsp:forward page="checkout.jsp" />

<%
}
}
%>
```

Если пользователь пытается добавить или убрать элемент или выписать счет, запрос отправляется в контроллер, `main.jsp`. Эта страница контроллера управляет запросами на добавление, исходящими из `eshop.jsp`, также как запросами на удаление и выписку из `cart.jsp`. Если это запрос на добавление, например, контроллер обрабатывает параметры запроса для добавляемого элемента, а потом создает бин CD (показан в Листинге 7), представляющий выбор. Обновленный объект корзины покупок далее передается назад в сессию. Контроллер также достаточно умный, для того, чтобы понять, что если выбранный CD выбирается опять, контроллер должен просто увеличить количество бинов для этого CD в корзине. Изменения, отражающиеся на состоянии корзины, такие, как добавление или удаление, вызывают передачу запроса контроллером после обработки в `eshop.jsp`. Это, в свою очередь, перезагружает главную страницу вместе с новым содержимым корзины. Если пользователь решает выписать счет, запрос после обработки передается в `checkout.jsp`, показанный в Листинге 8.

Большое преимущество в существовании отдельной страницы контроллера заключается в том, что вы всегда управляете тем, как определенные ресурсы вызываются в ответ на соответствующие действия. На самом деле, вы можете заставить контроллер инициализировать себя с помощью специального файла свойств, содержащего ресурсы для соответствующих действий пользователя. В этом случае, вы полностью выделите настройки своего веб-сайта и добьетесь максимальной гибкости.

Листинг 7: CD.java

```
package shopping.i18n;

import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class CD {
    String album;
    String artist;
    String country;
    String price;
    String quantity;

    public CD() {
        album="";
        artist="";
        country="";
        price="";
        quantity="";
    }

    public void setCDDProperties(HttpServletRequest req) {
        String myCd = req.getParameter("CD");
        StringTokenizer t = new StringTokenizer(myCd, "|");
        this.album= t.nextToken();
        this.artist = t.nextToken();
        this.country = t.nextToken();
        this.price = t.nextToken().trim();
        this.quantity = req.getParameter("qty");
    }

    public void setAlbum(String title) {
        album=title;
    }
    public String getAlbum() {
        return album;
    }

    public void setArtist(String group) {
```

Языковая поддержка для вебсайтов с использованием JSP

```
artist=group;
}
public String getArtist() {
return artist;
}

public void setCountry(String cty) {
country=cty;
}
public String getCountry() {
return country;
}

public void setPrice(String p) {
price=p;
}

public String getPrice() {
return price;
}

public void setQuantity(String q) {
quantity=q;
}

public String getQuantity() {
return quantity;
}
}
```

Страница `checkout.jsp` просто выделяет корзину пользователя из сессии и общую сумму запроса, и потом отображает выбранные элементы и их общую сумму. Рисунок 5 показывает окно клиента во время выписки, немецкий вариант.

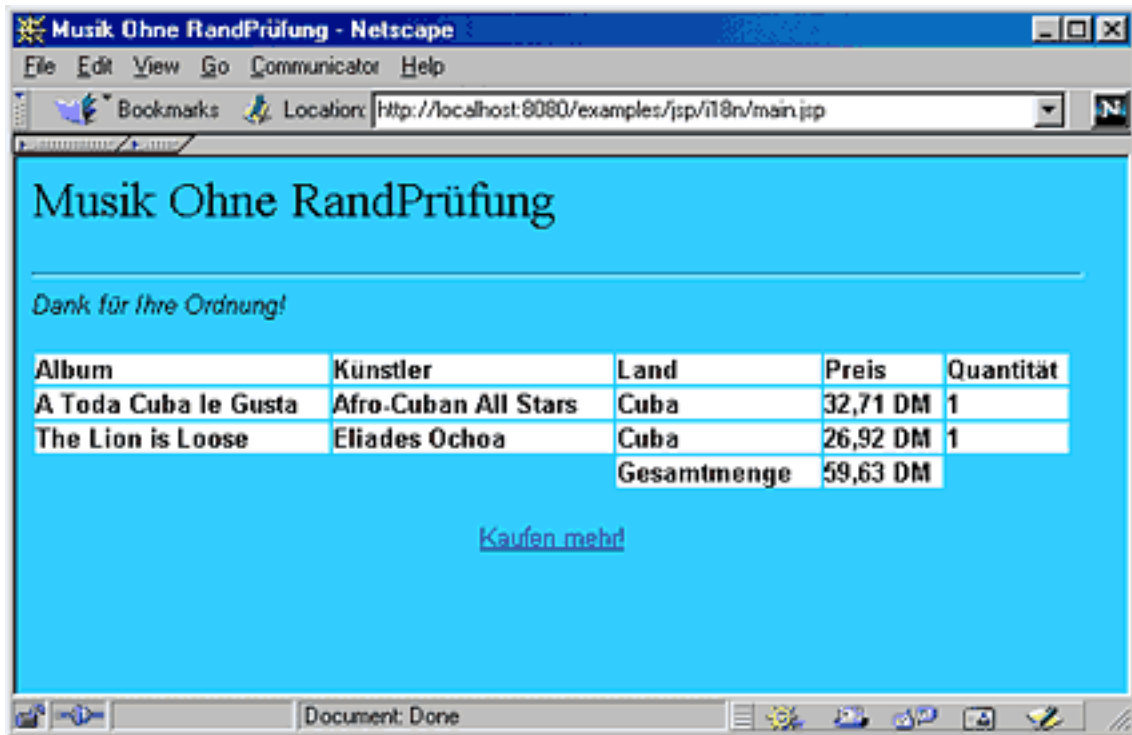


Рисунок 5. Выписка счета, немецкий вариант.

В момент выписки важно удалить объект сессии. Это делается с помощью вызова `session.invalidate()` в конце страницы. Этот процесс необходим по двум причинам. Во-первых, если сессия не ликвидирована, корзина покупок пользователя не сбрасывается; если пользователь затем пытается опять произвести покупки, его корзина будет содержать вещи, которые уже были оплачены. Вторая причина заключается в том, что если пользователь просто покинет сайт во время выписки, сессия не будет удалена сборщиком мусора и будет отнимать ресурсы у системы пока не истечет срок ее работы. Так как срок работы сессии по-умолчанию около 30 минут, для популярных сайтов это может привести к исчерпанию памяти.

Листинг 8: checkout.jsp

```
<%@ page import="java.util.*, shopping.i18n.CD" %>

<html>
<head>
```

Языковая поддержка для вебсайтов с использованием JSP

```
<title>
<%=session.getValue("checkout.title")%>
</title>
</head>
<body bgcolor='<%=session.getValue("checkout.bgcolor")%>'>
<font face="Times New Roman,Times" size=+3>
<%=session.getValue("checkout.title")%>
</font>
<p> <hr>
<em><%=session.getValue("checkout.subhead")%></em>
<p>
<center>
<table border="0" cellpadding="0" width="100%"
bgcolor='<%=session.getValue("cart.bgcolor")%>'>
<tr>
<td><%=session.getValue("cd.albumLabel")%></td>
<td><%=session.getValue("cd.artistLabel")%></td>
<td><%=session.getValue("cd.countryLabel")%></td>
<td><%=session.getValue("cd.priceLabel")%></td>
<td><%=session.getValue("cd.quantityLabel")%></td>
<td></td>
</tr>

<%
Vector buylist=(Vector)session.getValue("shopping.shoppingcart");
for (int i=0; i< buylist.size();i++) {
CD anOrder = (CD) buylist.elementAt(i);
%>
<tr>
<td><%=anOrder.getAlbum()%></td>
<td><%=anOrder.getArtist()%></td>
<td><%=anOrder.getCountry()%></td>
<td><%=anOrder.getPrice()%></td>
<td><%=anOrder.getQuantity()%></td>
</tr>
<% } %>
<tr>
<td> </td>
<td> </td>
<td><%=session.getValue("checkout.totalLabel")%></td>
```

```
<td><%=request.getAttribute("amountDue")%></td>
<td>      </td>
</tr>
</table>
<p>
<a href=i18nDemo.jsp><%=session.getValue("checkout.returnLabel")%></a>
</center>
<%
session.invalidate();
%>
</body>
</html>
```

2. Доставка локализованных веб-приложений

Я предполагаю, что вы используете последнюю версию JavaServer Web Development Kit (JSWDK) от Sun для запуска примера. Если нет, посмотрите в [Ресурсах](#), где его можно взять. Предполагая, что сервер установлен в \jswdk-1.0.1, его директории по-умолчанию в Microsoft Windows, доставка файлов приложения выглядит так:

- Создать каталог i18n в \jswdk-1.0.1\examples\jsp
- Скопировать i18nDemo.jsp в \jswdk-1.0.1\examples\jsp\i18n
- Скопировать eshop.jsp в \jswdk-1.0.1\examples\jsp\i18n
- Скопировать cart.jsp в \jswdk-1.0.1\examples\jsp\i18n
- Скопировать main.jsp в \jswdk-1.0.1\examples\jsp\i18n
- Скопировать checkout.jsp в \jswdk-1.0.1\examples\jsp\i18n
- Откомпилировать CD.java набрав javac CD.java
- Создать каталог shopping\i18n в \jswdk-1.0.1\examples\Web-Inf\jsp\beans
- Скопировать CD.class в \jswdk-1.0.1\examples\Web-Inf\jsp\beans\shopping\i18n

После запуска сервера вы должны запустит приложения используя URL <http://localhost:8080/examples/jsp/i18n/i18nDemo.jsp>.

3. Говоря о символах

В этом примере я использовал 8-bit кодировку ISO Latin-1 (aka ISO-8859-1), так как она поддерживает 200 символов, общих для большинства западноевропейских языков. Но

Языковая поддержка для вебсайтов с использованием JSP

для создания по-настоящему глобального приложения, нужно использовать кодировки для поддержки китайского, арабского, японского, и тд, языков, с диапазоном символов, превышающим набор из 256 символов в ISO Latin-1. Вы можете получить список поддерживаемых в Java кодировок в [Ресурсах](#). Если вы используете не ISO-8859-1, а что-то другое, вы должны сообщить это браузеру в атрибуте contentType тэга page:

```
<%@ page contentType="text/html; charset=charset_name" %>
```

Например, для создания динамического содержимого в кириллице (для русских или болгарских страниц), ваш JSP должен сообщить об этом браузеру:

```
<%@ page contentType="text/html; charset=ISO-8859-5" %>
```

Конечно, в этом случае вашим пользователям понадобится браузер, который поддерживает новый набор символов. Кроме того, вам нужно убедиться, что браузеры совместимы с HTML 4.0, так как HTML 4.0 поддерживает Basic Multilingual Plane, стандартный 16-bit набор символов, поддерживающий большинство мировых языков. Важно также, чтобы браузер был укомплектован необходимыми шрифтами для отображения символов соответствующего языка.

4. Выводы

В этой статье я продемонстрировал создание многолокальных страниц JSP. Вебсайты, основанные на технологии JSP легче поддаются интернационализации из-за упрощения разделения дизайна и логики приложения. Разработка по-настоящему глобального сайта, однако, трудная задача, связанная с разными наборами символов и несовместимостью их поддержки в браузерах.

5. Об авторе



[Гоувайнд Сишадри](#) Java гуру в jGuru.com и автор *Enterprise Java*

Computing — Applications and Architecture, изданной Cambridge University Press (1999).

Узнайте больше о [Гоувайнл](#) на jGuru.com.

6. Ресурсы

- "Understanding JavaServer Pages Model 2 architecture," Govind Seshadri (*JavaWorld*, December 1999):
<http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html>
- Sun's internationalization home page:
<http://web2.java.sun.com/products/jdk/1.1/docs/guide/intl/index.html>
- List of valid language codes (ISO-639):
<http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>
- List of valid country codes (ISO-3166):
http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html
- List of Java-supported encodings:
<http://java.sun.com/products/jdk/1.1/docs/guide/intl/encoding.doc.html>
- Download the latest JavaServer Web Development Kit:
<http://java.sun.com/products/jsp/download.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-03-ssj-jsp.html>

[Перевод на русский © Сергей Миссан, 2000](#)