Интернет програмиране с Java – част 8

Автор: Светлин Наков,
СУ "Св. Климент Охридски"
Web-site: http://www.nakov.com

Последна промяна: 25.11.2002

В предходната част от курса разгледахме как Java-сървлетите могат да извличат изпратените към тях параметри, изяснихме етапите от жизнения им цикъл и обяснихме как много потребители могат да работят едновременно и независимо един от друг с един сървлет като използват HTTP сесии. В тази последна част от курса ще изясним технологията Java Server Pages (JSP). Ще обясним основните тагове в JSP, ще изясним каква е връзката между JSP и сървлетите и ще покажем техники, които намаляват усилията за създаване на Web-приложения. Ще да дадем и цялостен пример за Web-приложение, което показва Java технологиите за Web-програмиране в действие.

Java Server Pages (JSP)

Java Server Pages (JSP) е технология, която позволява в статичен HTML документ да се вгражда програмен код на Java, който се изпълнява при заявка за достъп този документ. Фрагментите програмен код, вградени в HTML документа се ограждат със специални тагове и се наричат скриплети. HTML документи, съдържащи скриплети, се наричат JSP-страници или за по-кратко JSP-та. Когато Web-контейнерът изпълнява един JSP файл при клиентска заявка за достъп до него, се връща HTML документът, който се съдържа в този файл, като всички скриплети, вградени в него, се изпълняват се заместват с резултата получен от тях. Така един JSP документ, който е смесица между Java и HTML, след изпълнението си се превръща в динамично генериран чист HTML документ. Идеята е статичният HTML в един документ да си остане статичен, а програмен код на Java да се пише само за генериране на динамичните части от този документ, а не за целия документ. За разлика от сървлетите, в JSP-тата не е нужно статичният HTML текст да се отпечатва в изходния поток чрез извиквания от вида `out.println(…)`, защото вместо това може да се използва чист HTML текст. Това силно улеснява работата на разработчика, подобрява четимостта на кода и опростява поддръжката му.

JSP скриплети

Скриплетите в JSP-страниците се ограждат с таговете `<%` и `%>` съответно за начало и край. Ето един пример за JSP, което отпечатва текущата дата и час, използвайки скриплет:

```
<html>
    <head><title> Date JSP demo </title></head>
    <body>
        The date is:
        <% out.println(new java.util.Date()); %>
    </body>
</html>
```

Както се вижда от текста на това примерно JSP, то много прилича на обикновен HTML документ, само че съдържа скриплет, който се изпълнява динамично и отпечатва текущата дата и час чрез стандартните средства на Java.

Според стандарта за Java Server Pages във всички JSP-страници автоматично се създават следните обекти:

request – за достъп до заявката и параметрите изпратени чрез нея
response – за управление на отговора на заявката

out – изходен текстов поток за отговора на заявката
session – за управление на потребителските сесии
application – за достъп до данните, съхранявани във Web-приложението

За удобство на програмиста тези обекти са достъпни от всички скриплети в JSP-страницата. В нашия пример използвахме обекта out, чрез който отпечатахме текущата дата в изходния поток на JSP-страницата.

Технологията Java Server Pages предоставя на Web-разработчика освен скриплети и други тагове. Да разгледаме най-важните от тях.

JSP изрази

JSP изразите са съкратен начин за отпечатване на стойността на Java израз в изходния поток на отговора на сървлета. Това става чрез тага `<%= израз %>`, който е еквивалентен на скриплета `<% out.write(израз) %>`. За пример можем да дадем JSP-страница, която отпечатва числата от 1 до 100 и техните квадрати:

```
<html>
    <head><title> Squares JSP demo </title></head>
    <body>
        <% for (int i=1; i<=100; i++) { %>
            Square of <%= i %> is equal to <%= i*i %>.
            <br>
        <% } %>
    </body>
</html>
```

Забележете, че когато се използват конструкции за управление в Java като условни конструкции и конструкции за цикъл, които изискват тялото им да е в отделен блок, трябва винаги този блок да е ограден с отваряща и затваряща фигурна скоба, т.е. да започва с "`{`" и да завършва с "`}`". Дори ако се използва само 1 ред HTML за тяло на блок, е необходимо той да е ограден с фигурни скоби. Примерът по-горе демонстрира и още една възможност на JSP-тата – да се използва чист HTML в тялото на цикли и if-конструкции. Както се вижда, възможно е един цикъл да започва в един скриплет и да завършва в друг, а тялото му да е в чист HTML, разположен между двата скриплета. Това се обяснява с начина, по който JSP документите се трансформират в сървлети и по-точно в Java сорс-код на сървлети, който след това се компилира до класове.

Как JSP документите се преобразуват в сървлети

JSP-тата са файлове, които се трансформират в сървлети от Web-контейнера, който ги изпълнява и след това се изпълняват като най-обикновен сървлет. Поради тази причина всичко, което знаем за сървлетите от предходните части на курса, важи и за JSP-тата. С малки изключения, целият JSP документ, включващ статичен HTML текст, скриплети и други JSP тагове, се трансформира в програмен код на Java и се записва в тялото на метод, който се извиква от метода `service(…)` на класа `HttpServlet`. Заради това JSP страниците отговарят както на GET така и на POST заявки по протокола HTTP. За простота можем да приемем, че Web-контейнерът, когато трансформира един JSP документ в сървлет, замества всички редове, представляващи чист HTML текст с програмен код, който отпечатва този текст в изходния поток на отговора на сървлета. Например първият ред от нашето JSP, съдържащ текста "`<html>`", се заменя при трансформацията с програмен код подобен на `out.write ("<html>")`. Можем да си представим JSP-тата и по друг начин – като най-обикновени сървлети, в които има кратък вариант за отпечатване на чист HTML текст, без да се използва `out.println(…)` или `out.write(…)`. Разбира се, освен краткия начин за печатане на статичен HTML текст, JSP-тата ни спестяват и доста допълнителни усилия, щяха да са необходими, ако използвахме сървлети. С JSP не е нужно да се дефинираниpа клас, който наследява `HttpServlet`, припокрива методите `doGet(…)`, `doPost(…)`, взема изходния поток от response-обекта и пише в него. Всички това става автоматично. Можем да считаме, че JSP е естествена крачка от развитието на сървлетите като технология, защото разширява техните

възможности и същевременно значително намалява усилията за създаването на динамични HTML документи.

При първо извикване на едно JSP, Web-контейнерът го трансформира в Java сорс код на сървлет, компилира го и получава клас файл. След това зарежда този клас в паметта и го изпълнява. При всяко следващо извикване, ако JSP-то не е променено, то не се компилира повторно, а директно се изпълнява получения по първото извикване компилиран код. Ако JSP-то е променено, то автоматично се прекомпилирва. При JSP-тата не е необходимо да се растартира сървърът при всяка промяна, за да се види резултата от нея, както трябва да се прави при промяната на сървлет при повечето Web-контейнери. Понеже JSP-тата са сървлети, те имат същия жизнен цикъл, като сървлетите и могат да използват всичко, което знаем за тях, като например механизма за управление на потребителската сесия. При JSP-тата автоматично се създава обект с име session, който представя потребителската сесия и както знаем, е различен за всеки различен клиент на нашето Web-приложение. Можем да използваме този обект за съхранение на данни, свързани с текущия потребител и неговото взаимодействие с Web-приложението.

JSP декларации

JSP декларациите представляват фрагменти програмен код на Java, които се вмъкват в кода на генерирания от JSP-то сървлет, директно в класа на сървлета, който се получава. За разлика от скриплетите, които се вмъкват в тялото на метод, който се вика от метода `jspService(…)` на класа `HttpServlet`, JSP декларациите се вмъкват не в някакъв метод, а директно в тялото на класа. Синтактично JSP декларациите се отделят от статичния HTML текст чрез тага `<%! … %>`. Използват се най-често за дефиниране на методи, които след това могат да се извикват от скриплетите, а също и за деклариране на член-променливи в класа на сървлета, който се получава от даденото JSP. Ето един пример за JSP, което генерира и отпечатва 10 случайни числа, всяко от които е между 0 и 999:

```
<%!
private java.util.Random mRandomGenerator =
    new java.util.Random();

private int getRandomNumber(int range) {
    return mRandomGenerator.nextInt(range);
}
%>
<html>
    <head><title>Random numbers demo</title></head>
    <body>
        <% for (int i=1; i<=10; i++) { %>
            Random number #<%= i %> is
            <%= ""+getRandomNumber(1000) %>.<br>
        <% } %>
    </body>
</html>
```

Както се вижда от кода, в този пример с тага `<!% … %>` се декларира и инициализира обект от класа `java.util.Random` и метод в класа на сървлета, който връща случайно цяло число в зададен диапазон. След това този метод се използва от JSP израз, който отпечатва случайно число между 0 и 999. Може да се забележи, че използването на класа `java.util.Random` става чрез пълното име на класа, предшествано от името на пакета, в който стои този клас. При нормалното програмиране на Java в програмата могат да се включват пакети чрез ключовата дума `import`, следвана от име на пакет. След това могат да се използват класовете от включените пакети като се изписват само имената им без пакетите, на които те принадлежат. В JSP също има начин за import-ване на пакети. Това става с атрибутът `<%@ page import="име_на_пакет" %>`, който се слага обикновено в началото на JSP-страницата. Например следният ред в JSP документ:

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

е еквивалентен на реда

```
import java.util.*;
```

написан в началото на сървлета преди декларацията на класа, който се получава при трансформацията на JSP-то в сървлет. Чрез подобен атрибут на JSP документа може да се зададе и

content-type-а на върнатия документ. Например ако искаме да върнем документ, който да се интерпретира от Web-браузъра на клиента като чист текст, а не като HTML, можем да напишем следното на един от началните редове на JSP документа:

```
<%@ page contentType="text/plain" %>
```

С подобни атрибути могат да се задават и други настройки на JSP-страницата. Например атрибутът

```
<%@ page session="false" %>
```

указва, че JSP страницата няма да използва сесия, с което се ускорява достъпът до нея и същевременно сървърът се натоварва по-малко. Тази настройка трябва да се слага във всички JSP страници, които не използват потребителската сесия (обекта session). Друг полезен атрибут на JSP страниците, който може да се задава по подобен начин, е страницата за обработка на грешки (error page). Ако в началото на една JSP страница се сложи ред, който съдържа

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

то всяко изключение (exception), възникнало по време на изпълнение на JSP-то, което не е обработено от това JSP, се предава на зададената страница за обработка на грешки. Задачата на тази страница за обработка на грешки е да покаже грешката във формат, разбираем за потребителя и евентуално да се погрижи да уведоми администратора за възникналия проблем. Всяка error страница трябва да съдържа тага `<%@ page isErrorPage="true" %>`, който указва, че това е страница за обработка на грешки. В такива страници е достъпен още един допълнителен обект exception, който съдържа последното възникнало изключение, което описва грешката.

JSP и JavaBeans

Според JavaBeans спецификацията bean-овете представляват обикновени Java класове, които отговарят на следните допълнителни условия: имат конструктор без параметри; нямат публични член-променливи; могат да имат свойства (properties), които са достъпни чрез публични методи с имена `getXXX()` и `setXXX(…)`, където `XXX` е името на съответното property. В JSP страниците могат да се създават и използват Java bean-ове чрез тага `<jsp:useBean … />`. Например тага:

```
<jsp:useBean id="userInfo" class="com.nakov.example.UserInfo" />
```

декларира екземпляр на Java bean с име userInfo от класа com.nakov.example.UserInfo, който е достъпен от всички скриплети на JSP-то. Тази декларация е почти еквивалентна на обикновеното инстанциране на клас, което в скриплет може да стане чрез следния код:

```
<% com.nakov.example.UserInfo userInfo = new com.nakov.example.UserInfo(); %>
```

За разлика от директното инстанциране на класове, тагът `<jsp:useBean … />` дава доста допълнителни възможности. Една от тези възможности е задаването на обхват на действие за bean-овете чрез атрибута "`scope`". Този обхват може да бъде текущата страница (page scope), текущата заявка (request scope), текущата потребителска сесия (session scope) или цялото Web-приложение (application scope). Един bean се създава винаги при първото му използване, а след това не се унищожава, докато не излезе от обхвата, с който е дефиниран. Например ако се използва bean с обхват текущото приложение, той ще се създаде при първото му използване и ще е достъпен от всички JSP-та и сървлети в приложението. Класът на bean-а ще се инстанцира само веднъж в рамките на приложението и ще се унищожи при спиране на това Web-приложение или при спиране на сървъра. Ако се използва bean с обхват текущата сесия, той ще се създава при всяко първо извикване в рамките на всяка нова сесия и ще се унищожава при унищожаване на сесията, т.е. този bean ще има по една инстанция за всяка потребителска сесия на Web-приложението. Обхватът request и обхватът page много си приличат по това че са краткотрайни – важат само в рамките на едно извикване. Bean-овете с обхват page съществуват само през времето, в което се изпълнява JSP-страницата и се унищожават при приключване на нейното изпълнение. Bean-овете с обхват request съществуват през цялото време на подготвянето на отговора на клиентската HTTP заявка, дори ако този отговор се генерира в резултат от последователното изпълнение на няколко JSP-та.

За достъп до полетата на един bean (неговите properties), има два JSP тага: `<jsp:getProperty … />` и `<jsp:setProperty … />`. Те са еквивалентни на директния достъп до полетата на bean-а. Например изразът

```
<jsp:getProperty name="userInfo" property="name" />
```

е еквивалентен на израза

```
<%= userInfo.getName() %>
```
И двата израза отпечатват името на потребителя, който се описва от bean-а userInfo. Ползата от таговете `<jsp:getProperty …/>` и `<jsp:setProperty … />` е това, че са в XML формат, което ги прави по-лесни за използване от човек, който не е програмист. Освен това те имат и допълнителни възможности. Ето и пример за задаване на стойност на поле в bean:
```
<jsp:setProperty name="userInfo" property="password"
value='<%= request.getParameter("userPassword") %>' />
```
Забележете, че в стойността на атрибута value на тага `<jsp:setProperty … />` може да се използват JSP изрази, а не само константен текст.

Едно от полезните неща, от които може да се възползва програмистът, който използва JavaBeans съвместно с JSP при разработваното Web-приложение, е зареждането на полетата на bean-ове от параметри, изпратени към дадена JSP страница. Това може да стане чрез атрибута "`param`" на тага `<jsp:setProperty … />`. Например следният код:
```
<jsp:setProperty name="userInfo" property="password" param="userPassword" />
```
зарежда в полето `password` на bean-а `userInfo` стойността, записана в параметъра с име `userPassword` на заявката към страницата. Ако типът на полето в bean-а е числов, се прави автоматично конвертиране в число на текстовата стойност, съдържаща се в параметъра.

Друго предимство при използването на JavaBeans съвместно с JSP е, че може да се зададе автоматично зареждане на всички полета на даден bean от параметри на заявката със същите имена. Например ако имаме bean-а `userInfo`, който съдържа полетата `name`, `password` и `age`, можем да ги заредим от изпратените към страницата параметри чрез следния код:
```
<jsp:setProperty name="userInfo" property="*" />
```
За да е успешно зареждането, е необходимо към страницата да са изпратени параметри с имена `name`, `password` и `age`. Ако имената на полетата и имената на изпратените параметри не съвпаднат, при зареждането някои полета ще останат без стойност. При много на брой параметри и полета на bean-а, в който тя трябва да се запишат, този таг е много полезен, защото спестява голямо количество код и механичния труд, необходим за написването на този код.

От всичко, изтъкнато до момента, можем да си направим извода, че използването на Java bean-ове е много полезно за отделяне на логиката от визуализацията в Web-приложенията. Чрез съвместното използване на JavaBeans и JSP-та се дава възможност на Web-дизайнерът да създаде перфектния дизайн за нашето Web-приложение, без да знае Java и без да познава детайлите на JSP програмирането, а на програмиста се дава възможност да пише части от кода в отделни класове (bean-ове) извън JSP-то, като така концентрира вниманието си върху тях, а не върху HTML таговете.

Друг полезен таг в JSP страниците е тагът `<%@ include file="relative_url" %>`. Той позволява включването на съдържанието файл на текущата позиция в дадена JSP страница. Включването става по време на трансформирането на JSP страницата в сървлет. Този таг е особено подходящ когато Web-приложението съдържа много JSP страници, съдържащи общи фрагменти. Например ако трябва в началото на всяка страница от нашето Web-приложение да има меню, бихме могли да отделим кода, който създава това меню в отделен файл и да го включваме във всеки JSP файл с тага `<%@ include … %>`. Тази възможност позволява повторното използване на вече написани фрагменти код (code reuse), което при големи проекти е много често използвана техника. Например може в началото на JSP документа да се включи следния ред:
```
<%@ include file="menu.jsp" %>
```
Той включва съдържанието на файла menu.jsp в текущата JSP страница по време на компилация. Освен този таг, за включване на фрагмент код в текущата JSP страница има и още един подобен таг: `<jsp:include page="relative_url" />`. При включване чрез `<%@ include … %>` включеният файл се прочита веднъж при първото изпълнение на JSP-то и след това дори да бъде променен, промените не се отразяват на JSP-то (static include). При включване на файл чрез `<jsp:include … />` включеният файл се изпълнява при всяка заявка към JSP страницата и резултатът от него се вмъква в страницата (dynamic include). Така, ако включеният файл бъде променен, промяната се отразява и на всички JSP-та, които го включват. Освен това чрез `<jsp:include … />` могат да се включват сървлети, CGI скриптове и други ресурси, достъпни чрез зададеното URL, а не само фрагменти от JSP документи. Ето и пример за включване на заглавен фрагмент в началото на JSP страница:
```
<jsp:include page="header.jsp" flush="true"/>
```

Атрибутът `flush="true"` е задължителен и трябва винаги да се включва при използване на `<jsp:include … />` тага. Стойност false не е допустима.

Още един полезен таг в JSP стандарта е тагът за пренасочване към друго JSP `<jsp:forward page="relative_URL"/>`. При изпълнение на този таг, като резултат от заявката на клиента се връща резултатът от изпълнението на посоченото URL. Има голяма разлика между пренасочване чрез `request.sendRedirect(…)` (browser redirection) и `<jsp:forward … />` (server redirection). Методът `request.sendRedirect(…)` просто казва на браузъра да зареди посоченото URL вместо това URL, което е поискал. Това става като сървърът върне отговор с код 302 на HTTP заявката (document temporary moved). Такова пренасочване е еквивалентно на това потребителят да напише посоченото URL в address bar-а на браузъра и да го зареди. Пренасочването с `<jsp:forward … />` работи по друг начин. При него браузърът не разбира, че на сървъра се е извършило пренасочване, а просто получава резултата от изпълнението на URL-то, към което е направено пренасочване с `<jsp:forward … />`. В такъв случай в address bar-а на браузъра URL-то не се променя.

Цялостен пример за Web-приложение

За да илюстрираме всички техники, с които се запознахме, ще напишем едно цялостно многопотребителско Web-приложение и ще го изпълним с Web-контейнера Tomcat. Да си поставим за задача разработката на много прост дискусионен форум. Приложението трябва да има две страници – едната за влизане във форума, а другата за четене на съобщенията и добавяне на нови съобщения. За влизане във форума се изисква потребителско име и парола. Ако въведеното име съвпада с въведената парола, системата трябва да пуска потребителя на страницата за четене и добавяне на съобщения. В тази страница трябва да се извеждат в таблица всички съобщения и да има форма за добавяне на ново съобщение. Съобщенията се състоят от тема и съдържание. За леснота съобщенията могат да се пазят само в паметта на приложението, т.е. се губят при рестартиране на сървъра. Системата не трябва да позволява достъп до форума на неоторизирани лица, който не са влезли през началната страница. Ето как изглежда едно възможно решение на задачата:

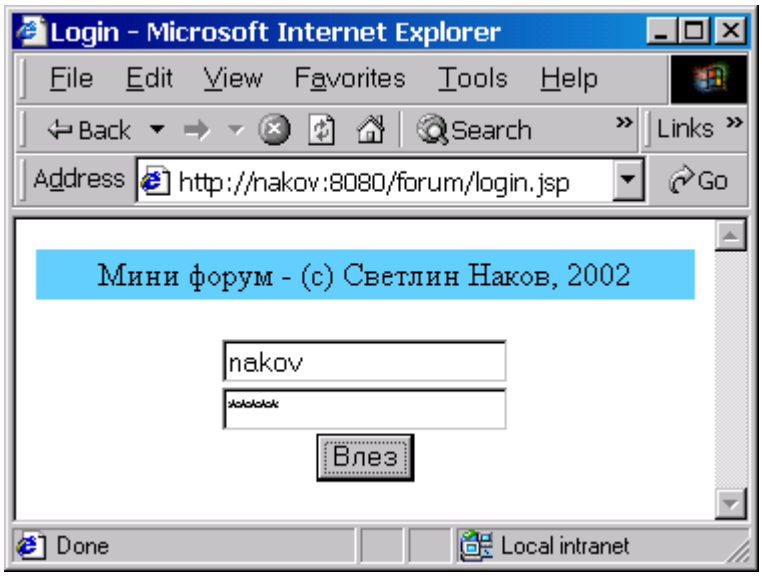
```
<%-- File name: login.jsp --%>
<%@ page contentType="text/html; charset=windows-1251" %>
<html>
<head><title>Login</title></head>
<body>
    <%@ include file="header.jsp" %>
    <div align="center">
    <%
    String userName =
        request.getParameter("user");
    String password =
        request.getParameter("pass");
    if ((userName!=null) && (password!=null)
            && (userName.length()>0)
            && (userName.equals(password))) {
        session.setAttribute("USER", userName);
        response.sendRedirect("main.jsp");
    }
    if (userName!=null) {
    %>
        Невалиден логин. Опитайте отново.<br>
    <%
    }
    %>
    <form action="login.jsp">
        <input type="text" name="user"><br>
        <input type="password" name="pass"><br>
        <input type="submit" value="Влез">
    </form>
    </div>
</body>
</html>

<%-- File name: main.jsp --%>
<%@ page contentType="text/html; charset=windows-1251" %>
```

```
<%@ page import="java.util.*" %>
<%!
    private String getMsgsHtml(ArrayList msgs)
    {
        if (msgs.size() == 0) {
            return "Няма съобщения.";
        }
        String resultHtml =
            "<table border=1 width=100% " +
            "cellspacing=0><tr><td>Тема</td>" +
            "<td>Съобщение</td></tr>\n";
        for (int i=0; i<msgs.size(); i++) {
            Message msg = (Message) msgs.get(i);
            resultHtml = resultHtml +
                "<tr><td>" + Utils.htmlEscape(
                msg.getSubject()) +
                "</td><td>" + Utils.htmlEscape(
                msg.getContents()) +
                "</td></tr>\n";
        }
        resultHtml = resultHtml + "</table>\n";
        return resultHtml;
    }
%>
<html>
<head><title>View Forum</title></head>
<body>
    <%@ include file="header.jsp" %>
    <div align="center">
    <%
    String userName = (String)
        session.getAttribute("USER");
    if (userName==null) {
    %>
        Нямате достъп до тази страница.<br>
        Моля <a href="login.jsp">влезте</a>
        в системата.
    <%
    } else {
        ArrayList msgList = (ArrayList)
            application.getAttribute("MESSAGES");
        if (msgList == null)
            msgList = new ArrayList();
        %>

        <%-- Add the new message if any --%>
        <jsp:useBean id="msg" class="Message" />
        <jsp:setProperty name="msg" property="*" />
        <%
            if (msg.getContents() != null) {
                msgList.add(msg);
                application.setAttribute(
                    "MESSAGES", msgList);
            }
        %>

        <br>
        <%-- Print all the messages --%>
        <%= getMsgsHtml(msgList) %>
        <br>

        <%-- Add new message form --%>
        <form action="main.jsp">
        <table border="0">
        <tr><td>Тема:</td><td>
        <input type="text" name="subject">
        </td></tr>
        <tr><td>Съобщение:</td><td>
        <input type="text" name="contents">
        </td></tr>
        <tr><td> </td><td>
        <input type="submit" value="Добави">
        </td></tr>
        </table>
```

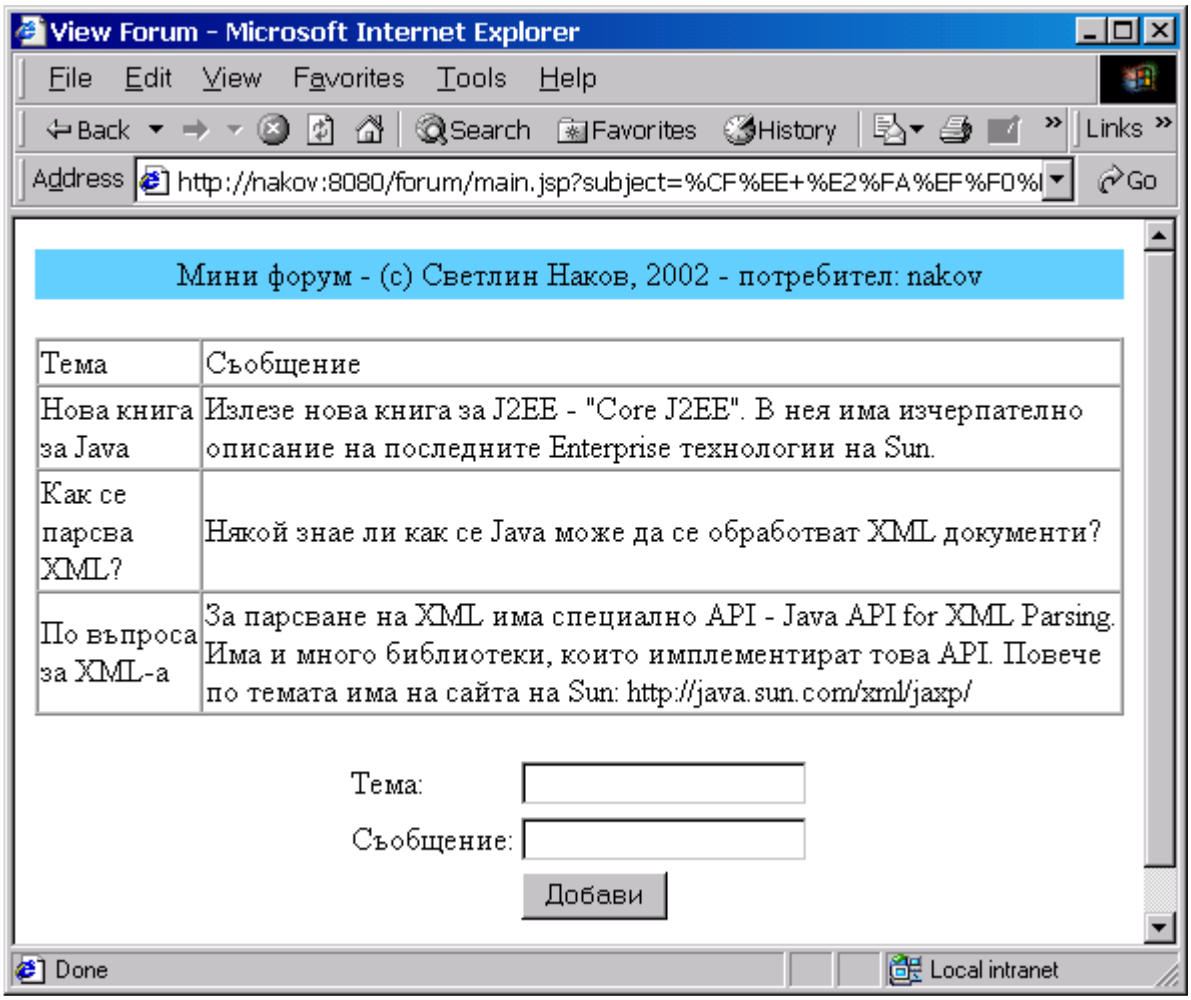
```
        </form>
    <%
    }
    %>
    </div>
</body>
</html>

<%-- File name: header.jsp --%>
<table border="0" bgcolor="#66CCFF" width="100%">
<tr><td align="center">
Мини форум - (c) Светлин Наков, 2002
<%
    String currentUser = (String)
        session.getAttribute("USER");
    if (currentUser != null)
        out.write(" - потребител: " +
        Utils.htmlEscape(currentUser));
%>
</td></tr>
</table>

// File name: Message.java
public class Message {
    private String mSubject;
    private String mContents;

    public String getSubject() {
        return mSubject;
    }
    public void setSubject(String subject) {
        mSubject = subject;
    }
    public String getContents() {
        return mContents;
    }
    public void setContents(String contents) {
        mContents = contents;
    }
}

// File name: Utils.java
public class Utils {
    public static String htmlEscape(String s) {
        StringBuffer out = new StringBuffer();
        for (int i=0; i<s.length(); i++) {
            char ch = s.charAt(i);
            if (ch == '\'')
                out.append("&#39;");
            else if (ch == '\"')
                out.append("&#34;");
            else if (ch == '<')
                out.append("&lt;");
            else if (ch == '>')
                out.append("&gt;");
            else if (ch == '&')
                out.append("&amp;");
            else
                out.append(ch);
        }
        return out.toString();
    }
}
```

За да се накара приложението да заработи под Tomcat, трябва да се направи следното: Да се създаде поддиректория с име `forum` в `%TOMCAT_HOME%/webapps` и в нея да се запишат всички JSP файлове. В същата тази директория `forum` трябва да се създаде поддиректория с име `WEB-INF`, а в нея поддиректория `classes`. Забележете, че има значение между малки и главни букви, особено ако искаме нашето Web-приложение да работи и под Linux. В поддиректорията `classes` трябва да се копират всички Java файлове и да се компилират до `class` файлове. Това е всичко, което е необходимо за стартиране на форума. Структурата на директориите не е подбрана случайно, а се

описва от спецификацията за Web-приложения на стандарта за J2EE. Таи спецификация гарантира, че ако директорията `forum` се копира заедно със всичките ѝ поддиректории на друг Web-контейнер, приложението ще заработи по същия начин. След като се стартира сървърът Tomcat, приложението е достъпно от адрес `http://localhost:8080/forum/login.jsp`. Ето и изглед от началния екран (`login.jsp`):



След успешно влизане във форума, се преминава на екрана за разглеждане и добавяне на съобщения (`main.jsp`):



Да разгледаме по-подробно кода на нашето примерно Web-приложение. Началната страница `login.jsp` започва със задаване на content-type-а и набора от символи, които да се използват при показване на страницата. Избрана е стандартната кодова таблица windows-1251, която е

единствената, използвана в България за работа с кирилица под Windows. Тази кодова таблица се поддържа от всички стандартни Web-браузъри и затова кирилицата в страницата би трябвало да се покаже правилно без да е нужно потребителят да задава никакви допълнителни настройки. Следва стандартно начало на HTML документ, след което статично се включва файлът `header.jsp`. Този файл се включва в началото на всяка страница на приложението и служи за изобразяване на стандартна заглавна част, в която пише името на приложението, името на текущия потребител и още някаква друга допълнителна информация. Следва извличане на параметрите потребителско име и парола, ако са изпратени такива. Понеже `login.jsp` работи в два варианта – без параметри и с параметри, се разглеждат съответно няколко случая. Ако са изпратени потребителско име и парола, те се валидират (за простота просто се проверява дали съвпадат и не са празни низове) и ако валидацията е успешна, в сесията се записва името на валидирания потребител под ключ "`USER`" и браузърът на клиента се препраща към основната страница на приложението (`main.jsp`). Ако валидацията е неуспешна, се отпечатва съобщение за грешка и се извежда формата за въвеждане на потребител и парола. Ако `main.jsp` се извика без параметри, се извежда същата форма за въвеждане на потребител и парола, но без никакво съобщение. Тази форма е настроена да изпраща данните, въведени от потребителя, към същата страница, в която се намира самата тя (`index.jsp`). Така потребителят може да прави последователно много опити за влизане в системата, но достига до главната страница на форума само при успешна валидация. Ако той се направи на хитър и въведе директно URL-то на основната страница `http://localhost:8080/forum/main.jsp`, системата няма да го пусне, защото няма да намери име на потребител, записано в сесията под ключа "`USER`".

Нека сега разгледаме същината на приложението – главната страница `main.jsp`. Тя също може да се вика с параметри или без параметри. Ако се извика без параметри, тя извежда всички съобщения от списъка. Ако се извика с параметри тема и съобщение, тя добавя това съобщение в списъка със съобщенията и извежда този списък. Подобно на `login.jsp` първоначално се задава кодовата таблица, което осигурява правилното извеждане на кирилицата. След това се показва заглавната част на страницата чрез статичното включване на файла `header.jsp`. Следва вземане на името на валидирания потребител от текущата сесия. Това се прави като се извлича стойността записана под ключ с име "`USER`" в сесията. Ако потребителят е преминал успешно през `login.jsp`, би трябвало такава стойност да има. Ако потребителят не е валидиран, такава стойност няма да има, т.е. ще се получи стойност `null`. В този случай потребителят се уведомява че няма право на достъп и се подканва да влезе в системата през началната страница. Ако потребителят е валидиран, има два случая – или към страницата е изпратена заявка за добавяне на ново съобщение, или страницата е поискана без параметри. За извличане на параметрите се използва JavaBean класа `Message`, който се инстанцира с тага `<jsp:UseBean … />`. След това параметрите се записват в bean-а чрез техниката за автоматично прехвърляне на параметри с тага `<jsp:setProperty … />`. Забележете, че bean-а `Message` има полета, които точно съответстват на имената на очакваните параметри. Ако е имало изпратен параметър за съдържание на съобщение, в полето `contents` на bean-а ще има стойност, различна от `null`. Ако стойността е `null`, се счита, че страницата е извикана с цел показване на всички съобщения, а не с цел добавяне на ново съобщение. За съхранение на съобщенията в приложението се използва списък (`java.util.ArrayList`), който се съхранява в приложението под ключ "`MESSAGES`". Забележете, че списъкът със съобщенията се съхранява в приложението, което го прави достъпен за всички потребители, а не се съхранява в сесията, както името на активния потребител. Възможно е в приложението да няма ключ под име "`MESSAGES`". Това означава, че все още никой потребител не е добавял никакви съобщения. Ако списък със съобщения няма, той се създава и първоначално е празен. След това ако е изпратено като параметър ново съобщение, то се добавя в списъка и стойността под име "`MESSAGES`" в приложението се актуализира. Независимо дали е добавено ново съобщение или не, всички съобщения се извеждат. За целта се използва процедура, дефинирана чрез JSP декларация с тага `<%! … %>`. Тя поема като параметър списък от съобщения и връща като резултат HTML текст, който представлява визуализация на тези съобщения във вид на таблица. След кода за визуализацията на съобщенията в JSP страницата е разположена HTML формата за добавяне на нови съобщения. Тази форма съдържа само две полета и бутон за добавяне на съобщение, който изпраща данните, попълнени от потребителя към същата JSP страница

(`main.jsp`). Както вече обяснихме, тези параметри се записват в bean и се обработват по подходящ начин след това. Остана да обясним как се визуализират съобщенията във вид на HTML. Методът `getMsgsHtml(…)`, който се дефиниран като частен за класа на JSP страницата `main.jsp`, първо проверява дали списъкът, който му е подаден като параметър не е празен. Ако е празен, връща подходящо съобщение, а в противен случай създава таблица и извежда в нея елементите на списъка със съобщенията използвайки for-цикъл. Едно много важно нещо, което се прави при отпечатването на съобщенията в таблицата, която се генерира, е да се заменят непозволените за HTML документите символи с техния еквивалент в HTML (HTML escaping). Става въпрос за символи като `<`, `>`, `&` и някои други, които могат да повредят HTML документа, ако се извеждат в него без да се escape-ват. Например ако някой добави във форума съобщение със съдържание "`</td></tr></table>грешка!`", може да повреди таблицата и да предизвика писане на текст извън нея. В HTML има специален начин за избягване на опасните символи, наречен HTML escaping, който трябва да се използва винаги, когато се извеждат низове, които биха могли да съдържат опасни символи. Въпреки, че този проблем възниква почти в 100% от Web-приложенията, разработвани с Java, в JSP/Servlet API-то няма стандартен метод, който извършва HTML escaping. Колкото и да е странно, стандартен еквивалент на PHP функцията `html_escape(…)` няма и ние трябва сами да си го напишем. Съществува един стандартен клас `java.net.URLEncoder`, но той служи за друг вид escaping – URL escaping, който се използва за кодиране на информацията при изпращане на HTML форми. За да може да се използва от всички JSP-та на приложението, този метод е дефиниран като статичен в отделен клас с име Utils. Ако някога ви се налага HTML escaping, което е доста вероятно, ако пишете Web-приложение, използвайте наготово метода `htmlEscape(…)`, който е даден в края на сорс-листинга, за да не губите време да си го пишете сами или да го търсите в Интернет.

По листинга по-горе следва реализацията на файла `header.jsp`, съдържащ фрагмент от код, който е предназначен да стои в началото на всяка страница от приложението. В този код се проверява дали в сесията има верифициран потребител и ако има, освен стандартният заглавен текст, се отпечатва и името на активния потребител.

Примерното приложение има много ограничена функционалност, защото изискванията към него са да бъде доста кратко и същевременно да илюстрира възможно по-голяма част от JSP таговете и техниките описани в настоящата статия. Оставяме на читателя да се опита да го подобри и разшири, като вярваме, че със знанията придобити от настоящата поредица статии "Интернет програмиране с Java", това няма да го затрудни.

В заключение трябва да отбележим, че JSP технологията за разработка на Web-приложения е една от водещите в световен мащаб и е популярна не по-малко от ASP, PHP, Perl и ColdFusion. Наблюдава се тенденция за по-големите и по-сложните приложения да използва именно JSP вместо Perl и PHP, защото J2EE платформата отговаря на всички нужди, които имат тези приложения – надеждност, сигурност, скалируемост, разпределеност, разширяемост и т.н.