Въведение в съвременното Web-програмиране - Java Server Pages

© Светлин Наков, 2000
http://www.nakov.com

Много от нас знаят че в последните 1-2 години, когато става дума за разработка на софтуер, JAVA е най-често споменаваната технология в силиконовата долина. Много от нас знаят също, че тази прочута силиконова долина играе определящо значение за развитието на софтуерния и въобще на компютърния бизнес в световен мащаб и затова на всички им е интересно какво се крои там. Без да се отвличам от темата, ще ви спомена само още няколко от технологиите, които там са най-популярни – XML, Enterprise Java Beans, Java Server Pages, Oracle, SQL Server. На фона на това от ORACLE (компанията която измести Microsoft от завидното челно място в бизнеса) твърдят че "или правиш e-business или си out-of-business". От това веднага става ясно, че WEB-технологиите ще са от още по-голямо значение през идващите години. Ето защо в настоящата статия смятам да ви запозная с една от най-модерните технологии за ИНТЕРНЕТ програмиране – Java Server Pages (JSP), която в момента е почти хит. Настоящата статия се явява естествено продължение на статията "JAVA сървлети – въведение в WEB-програмирането", публикувана в брой 5/2000 на сп. PC Magazine/Bulgaria, така че съветвам всички, които не са просветени в WEB-програмирането с JAVA, да прочетат първо нея.

Нека сега ви запозная как стоят нещата с WEB-програмирането с JAVA. Понеже e-business-а се гради основно на WEB-базираните информационни системи, от важно значение са технологиите за извличане на динамично генерирани HTML-страници. С риск да повторя част от вече изяснените в предишната статия основни механизми за извличане на динамичен HTML, ще отбележа, че те могат да се разделят на два типа – генериране на цял HTML документ в резултат на извикване на Server Side приложение /скрипт/ (написан на Perl, Bash, C++, Java и т.н.) и генериране на динамично съдържание в резултат на вграждане на програмен код (на PHP, Active Server Pages /ASP/, Java Server Pages /JSP/ или Cold Fusion) вътре в самия HTML документ, който се интерпретира от сървъра. За да можем да изясним JSP технологията, не можем първо да не се запознаем с JAVA сървлетите, понеже както ще разберем след малко, тези две технологии са много тясно свързани. JAVA сървлетът представлява програма на JAVA, която се изпълнява от сървъра при постъпване на заявка за достъп, изпратена от WEB-браузъра на клиента. Тази JAVA програма се състои от един или няколко класа, записани като .class файлове или .jar архив. Тя приема параметрите, изпратени от клиента и в зависимост от тях връща някакъв динамично генериран HTML документ. В помощ на новаците, ще отбележа, че сървлетите нямат нищо общо с аплетите, както JAVA няма нищо общо с JavaScript.

За да можете да използвате JAVA сървлети и JSP ви е необходим WEB-сървър или Application-сървър, който ги поддържа. Ако използвате някой от известните JAVA базирани сървъри (като JAVA Web Server на Sun или Weblogic Application Server), или някои други Application сървъри с вградена JAVA поддръжка (като ORACLE Application Server и IBM WebSphere Application Server), няма да има нужда да инсталирате нищо допълнително, но за безспорно най-популярните WEB-сървъри Apache и IIS (MS Internet Information Server) все още са необходими специални разширения. Един от най-успешните проекти за интегриране на Servlet/JSP технологията в Apache е Tomcat (http://jakarta.apache.org). За мераклиите, които искат да се занимават с него, ще отбележа, че инсталирането на Tomcat съвсем не е лесна работа за хора без опит и е малко вероятно човек, който не разбира от UNIX и JAVA да успее да го подкара. Дано в следващите версии на най-разпространения WEB-сървър в света Apache (с над 60% пазарен дял – http://www.apache.org) Tomcat да бъде вече напълно интегриран както например Perl, защото аз лично се борих с Tomcat в продължение на доста часове...

Имайки инсталиран Servlet/JSP поддържащ сървър, вече можем да започнем работа. Да разгледаме механизма на действие на сървлетите. При първо извикване сървърът прочита class файловете на сървлета и ги зарежда в паметта като обекти във вид, готов за изпълнение и разбира се след това ги изпълнява. При следващо извикване, обаче те са вече подготвени и сървърът само ги изпълнява. Така нещата стават доволно бързи. (Доволно бързи са, разбира се, само на доволно бързи компютри, но за хората от цивилизования свят това обикновено не е проблем. В България много хора смятат че "JAVA-та била бавна". Ами така си е, никой не им е виновен на тези хора, че нямат 512 MB

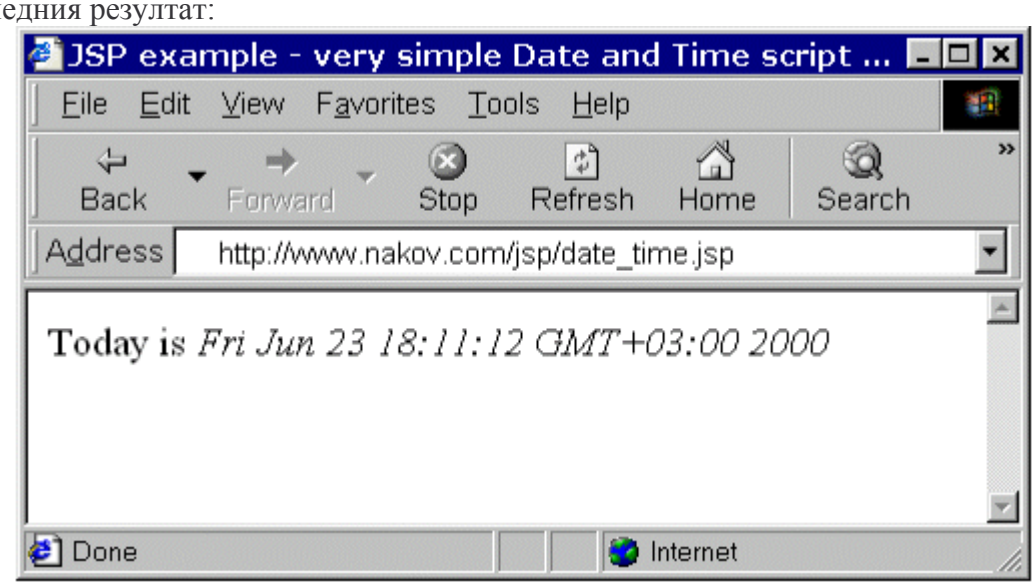
RAM и Katmai на 800 Mhz. Аз лично на такава машина нямам проблеми със скоростта. Ясно е, че сървлетите, като програми на JAVA са много по-бавни от повечето конкурентни технологии, но Sun са направили всичко възможно, за да ги пробутат на пазара, и както се забелязва, са успели. За това си има, разбира се, доста причини, но да не се отклоняваме от темата...).

Нека видим сега какво са JSP и каква е връзката им със сървлетите. JSP (Java Server Pages) е технология за вграждане на програмен код, написан на JAVA в HTML документи. Идеята е проста – да не се генерира цял HTML документ (както е при сървлетите), а да се вмъкват само отделни програмни фрагменти. Това олекотява значително задачата на JAVA програмиста, защото той вече не трябва да се грижи изцяло за оформянето на HTML документа, което си е една тежка задача, особено ако искаме страницата да изглежда добре. Нещата стоят така: Пишем си най-обикновен HTML с любимия HTML редактор и решаваме, че ни трябва някаква динамична информация. В този момент отваряме специален таг `"<%"`и в него си пишем на JAVA. За извеждане на текст в HTML документа използваме метода `out.println(String)`, а за получаване на подадените от браузера параметри използваме метода `request.getParameter(String param_name)`. Когато свършим с JAVA кода, затваряме JAVA тага с `"%>"` и продължаваме с HTML-а. При заявка за достъп до този документ сървърът го прочита, изпълнява JAVA кода в него, замества го с резултата от неговото изпълнение и връща на клиента вече преработения документ във вид на чист HTML. Е, това, разбира се не е съвсем така, но външно изглежда така. На практика за постигане на по-голяма бързина документите, съдържащи JSP скриптове (най-често файлове с разширение .jsp или .jhtml) се компилират предварително до JAVA сървлети (файлове с разширение .java), а оттам до .class файлове. За целта сървърът използва специален JSP компилатор, с който получава от HTML документа програма на JAVA (някакъв сървлет), а след това я компилира със стандартния компилатор на Sun `javac`. JSP компилаторът използва един стандартен шаблон на JAVA сървлет и просто добавя в него за всеки ред `"xxx"` от HTML кода по един ред от вида `out.println("xxx")`, а всеки ред, който е JAVA код, просто го копира 1 към 1. Така се получава сорс код на JAVA сървлет, който съответства на JSP документа. В крайна сметка всеки JSP документ, който ние пишем скрито от нас става на сървлет и генерира 100% динамичен HTML.

Изяснихме същността на JSP технологията. Нека сега я видим в действие. Ето един прост примерен JSP скрипт *(date_time.jsp)*, който печата коя дата и колко часа е в момента на изпълнението му:

```
<%@ page import="java.util.*;" %>
<HTML>
  <TITLE>JSP example - very simple Date and Time script</TITLE>
  Today is <I>  <% out.println(new Date()); %>  </I>
</HTML>
```

След като го запишем в съответната директория на сървъра и го извикаме от Internet Explorer получаваме следния резултат:



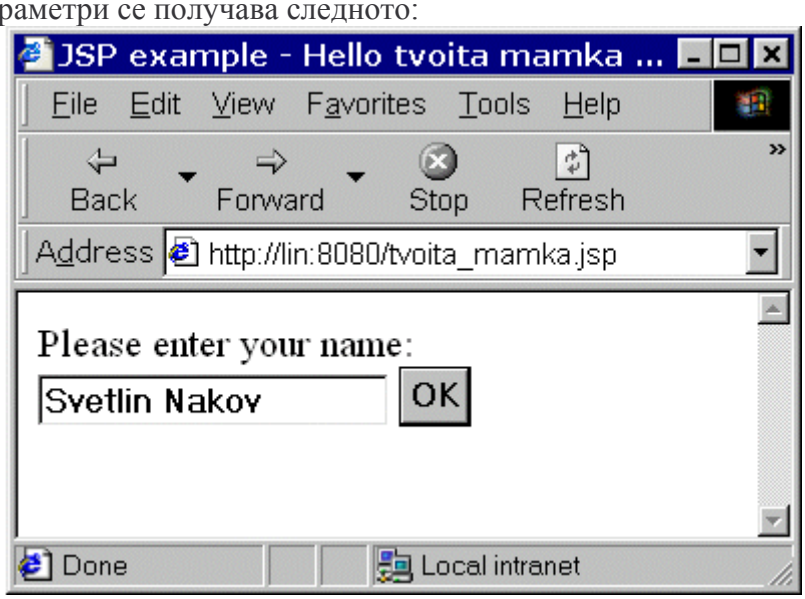
Забележете разликите от обикновения HTML документ – на първия ред чрез тага `<%@ page import="java.util.*;" %>` се включват пакетите `java.util.*`, за да може по-надолу да се ползва

`java.utils.Date()`. След това с `out.println()` се печатат датата и часа, които се взимат по стандартния за JAVA начин – чрез създаване на обект от класа `Date`.

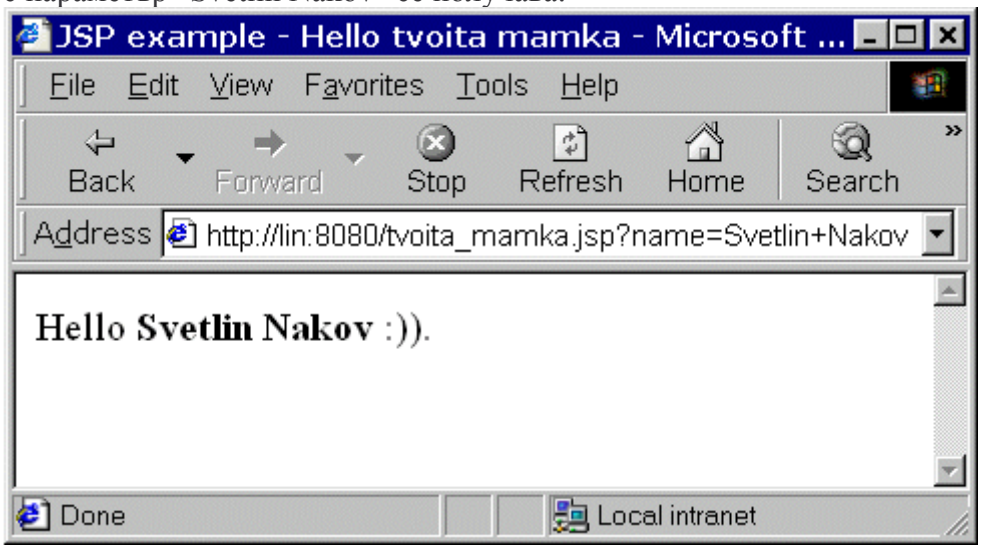
Нека сега видим нещо доста важно – как се взимат параметрите. Да си поставим една проста задачка. Да се напише JSP страничка, която при извикване без параметри да извежда форма с поле за въвеждане на име на потребител. При submit-ване на формата да се вика същата страничка с параметър `name=<име_на_потребител>`. В този случай тя да поздравява потребителя, който си е въвел името, а ако той не е въвел нищо, да му се кара. Ето как изглежда едно възможно решение *(tvoita_mamka.jsp)* :

```
<HTML>
   <TITLE> JSP example - Hello tvoita mamka </TITLE>
<% String name = request.getParameter("name");
   if (name == null) { %>
      <FORM METHOD=GET ACTION="/tvoita_mamka.jsp">
      Please enter your name: <BR>
      <INPUT TYPE=TEXT NAME="name">
      <INPUT TYPE=SUBMIT VALUE="OK">
      </FORM>
<% } else if (name.trim().length() > 0)
       out.println("Hello <B>" + name + "</B> :)).");
     else out.println("Tvoita mamka, daj si imeto be!");
%>
</HTML>
```

При изпълнение без параметри се получава следното:



При изпълнение с параметър “Svetlin Nakov” се получава:



Както виждате получаването на параметрите (всичко, което стои вдясно от символа “?” в URL-то) се прави тривиално лесно – с функцията `request.getParameter(String param_name)`, която

връща искания параметър във вид на символен низ или `null`, ако той не присъства. Също, както при сървлетите, е възможно да имате проблем с неправилното декодиране на кирилицата при получаването на параметрите. Без да влизам в подробни обяснение (това го направих в предишната статия), ви препоръчвам да използвате следния код:

```
String param = request.getParameter("parameter_name");
try { param = java.net.URLDecoder.decode(param);
} catch (Exception e) { e.printStackTrace(); }
```

Както споменах в предходната статия за сървлетите, смесването на програмен код с HTML документа не е хубаво да се прави. В съвременните софтуерни технологии е възприет така наречения многослоен модел при дизайна на приложенията. Винаги, когато се работи върху голям проект, той трябва да се раздели логически на много слоеве, които биха могли да работят независимо, а след това те трябва да се "сглобяват". Това е единственият разумен начин за работа на много хора върху един и същ проект. JSP ни дава възможност за разслояване на кода. Стандартният подход е логиката да се изнесе в отделен модул (в друг файл), а да не се смесва с кода, който се грижи за "красивия" външен вид и художественото оформяне на сайта. По този начин WEB-мастера ще има грижата да направи привлекателен сайт, като използва предварително написаните от програмиста модули за обработка на информацията и извличане на динамичното съдържание. В нашия случай тези модули са JAVA Beans, които, както ще видим след малко, се викат тривиално лесно от JSP документи. Първо трябва да изясним, че JAVA Bean-а е просто един JAVA клас, който има няколко публични метода с имена от вида `getXXX` и `setXXX` за достъп до някои полета, скрити в класа, като тези полета се наричат Property-та. Въпреки че това определение за bean е малко неточно, за нашите цели ще ни върши работа. Използването на bean-ове от JSP документ става посредством тага `<jsp:useBean id=`*`name`* `scope=`*`scope`* `class=`*`bean_class_name`*`>`, който създава нов обект от зададения клас и му дава исканото име *`name`*. Този обект е обект в смисъла на обектната технология в JAVA, все едно че е създаден чрез *`name = new bean_class_name()`* и се използва по стандартния начин, по който се използват обектите в JAVA.

Сега, за да илюстрираме мощността на JSP/Beans технологията, ще си поставим една доста по-сложна задача от практиката. Чрез нея не само ще изясним как се ползват bean-ове от JSP, но също и как се осъществява достъп до бази от данни през WEB като се използва JSP. Да си представим, че имаме една база от знания, която е записана в таблицата `emails` в ORACLE база от данни. Таблицата `emails` се състои от колоните *id, area, subject, sender, problem, solution*, където всеки запис си има уникален номер *id* и съдържа едно знание, което се описва от останалите колони. Нашата задача е да напишем JSP скрипт, който по зададен номер на запис да показва информацията (знанието) за този запис във вид на табличка. На основата на този скрипт, използвайки го за шаблон, всеки добър програмист ще може да напише цяла WEB-ориентирана информационна система за достъп и актуализация на данни. Ето защо избрахме точно тази задача – хем да илюстрираме технологията, хем да укажем помощ на сега започващите да се занимават с сериозно с JSP програмиране. Да предположим, че имаме написан клас наречен SearchBean, който има метод за извличане на запис от базата данни и методи за достъп до полетата на този запис. Тогава бихме могли да напишем скрипта, който показва записа по следния начин (*Search.jsp*) :

```
<%@ page import="SearchBean" %>
<!-- SearchBean should be in the JSP Servlet Engine's CLASSPATH -->

<HTML>
<TITLE> JSP, BEANS and ODBC example </TITLE>
<BODY>

<jsp:useBean id="record" class="SearchBean" scope="application">
</jsp:useBean>

<%  String id = request.getParameter("id");
    record.loadRecordFromDatabase(id);     %>

<center>
<table width=100% border=1>
```

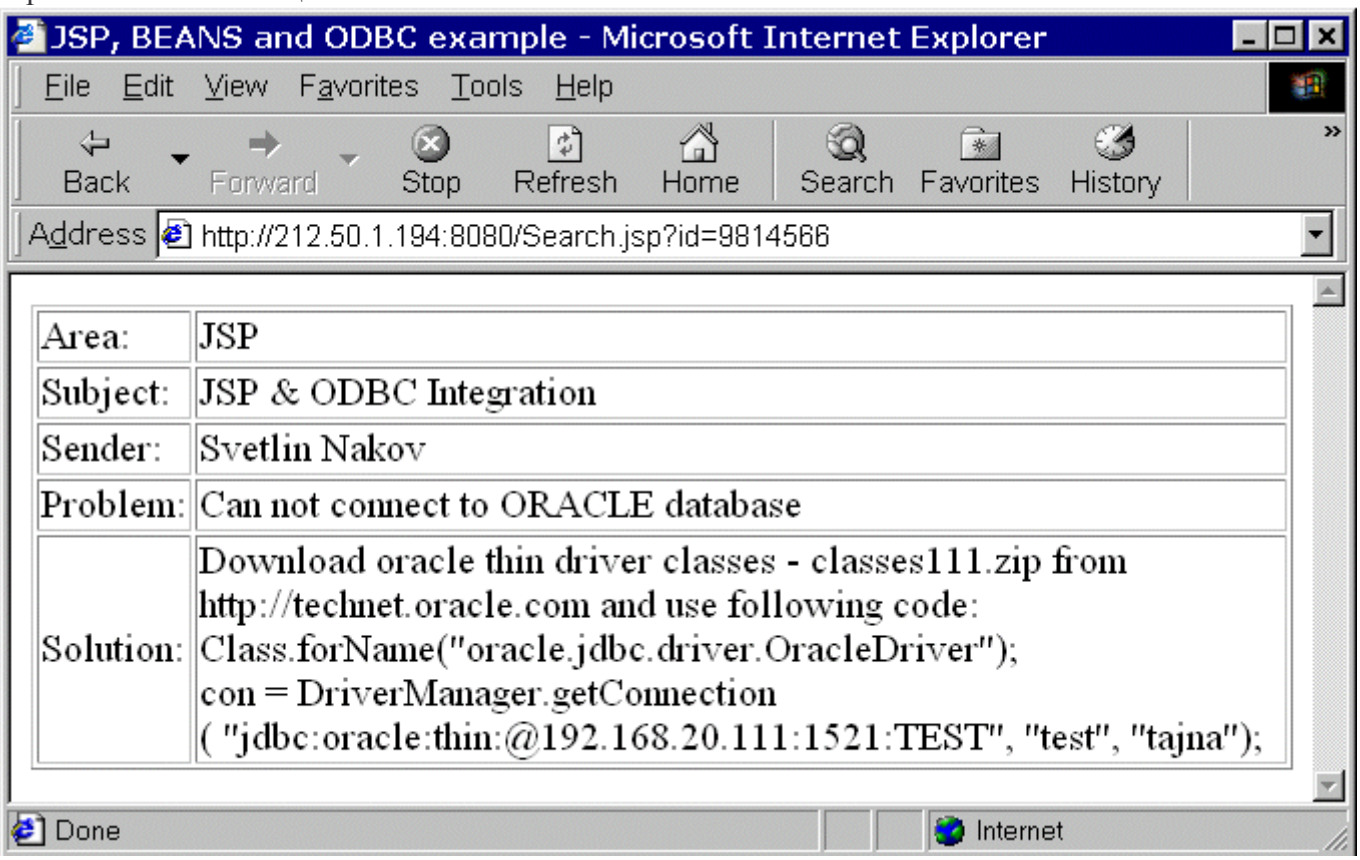
```
  <tr> <td> Area:     </td> <td> <%= record.getArea()     %> </td>
</tr>
  <tr> <td> Subject:  </td> <td> <%= record.getSubject()  %> </td>
</tr>
  <tr> <td> Sender:   </td> <td> <%= record.getSender()   %> </td>
</tr>
  <tr> <td> Problem:  </td> <td> <%= record.getProblem()  %> </td>
</tr>
  <tr> <td> Solution: </td> <td> <%= record.getSolution() %> </td>
</tr>
</table>
</center>

</BODY> </HTML>
```

Както личи от кода, в началото на скрипта чрез тага ″<jsp:useBean … >″ се създава обект от класа SearchBean наречен record. След това скриптът взима от подадените му параметри стойността на параметъра id и вика метода за зареждане на записа с номер id от базата данни. След това скриптът описва табличка по стандартния за HTML начин и вместо да слага текст в полетата й, той вика методите на обекта record за извличане на съответните полета на прочетения от базата запис. За печатане на резултата от извикването на метод, който връща String, се ползва тага ″<%=″. Както виждате в JSP документа няма сложен JAVA код и целият документ е ясно четлив. WEB-мастерът може да създаде страхотен външен вид, без да се интересува от това как се извличат данните. За него е важно само кои са Property-тата на съответния bean, които трябва да се печатат и как се инициализира bean-а. Програмистът, от своя страна само трябва да напише съответния bean *(SearchBean.java)* без изобщо да се занимава с оформянето на външния вид на данните, които ще се извеждат. Достатъчно е само да създаде лесен начин за тяхното извличане. В това се състои разслояването на приложението – имаме 2 напълно независими части – интерфейс (JSP документа, създаден от WEB-дизайнера) и бизнес логика (JAVA bean-а, написан от програмиста). Да погледнем и самия bean *(SearchBean.java)* :

```
import java.sql.*;

public class SearchBean {

  private static Connection con;
  private String area, sender, subject, problem, solution;

  // Establish database connection
  private static void connectToDatabase() {
    try {
      if (con != null)
        if (!(con.isClosed())) return;  // Already connected
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con = DriverManager.getConnection(
            "jdbc:oracle:thin:@192.168.20.111:1521:MYORACLE",
            "mente", "mente123");
    } catch (Exception e)
       { e.printStackTrace(); }  // Can't connect to database
  }

  // Retrieve the record with given id from the database
  public void loadRecordFromDatabase(String id) {
    connectToDatabase();
    String query = "SELECT area, subject, sender, problem," +
                   "solution FROM emails WHERE id=" + id;
    area = subject = problem = solution = "";
    try {
      ResultSet res = con.createStatement().executeQuery(query);
      if (res.next()) {
        area    = res.getString("area");
        subject = res.getString("subject");
        sender  = res.getString("sender");
```

```
        problem = res.getString("problem");
        solution= res.getString("solution");
      }
    } catch (Exception e) { e.printStackTrace(); }
  }

  public String getArea() { return area; }
  public String getSubject() { return subject; }
  public String getSender() { return sender; }
  public String getProblem() { return problem; }
  public String getSolution() { return solution; }
}
```

При извикването на метода `loadRecordFromDatabase(String id)` се проверява дали има отворена връзка с базата и ако има се използва тя, а ако няма се създава нова. След това се изпълнява проста команда на SQL (използването на SQL е нещо много типично за съвременните информационни системи), която извлича данните за зададения запис от базата в полетата *area*, *subject*, *sender*, *problem* и *solution*. За краткост в публикувания код грешките се игнорират, но в практически ситуации те трябва внимателно да се обработват. Методите за достъп до полетата (*getArea()*, *getSubject()* и т.н.) просто връщат съответното поле. За връзка с ORACLE базата данни се използва JDBC и стандартният клас за достъп до ORACLE - `oracle.jdbc.driver.OracleDriver` който може да се изтегли от http://technet.oracle.com. Всъщност драйверът за достъп до ORACLE се състои не само от този клас, а е цял архив от пакети с име classes111.zip, който също може да се изтегли от посочения сайт и трябва да бъде включен в класпътя на JSP engine-а. Това важи и за самия клас SearchBean, защото ако го няма в CLASSPATH-а, JAVA виртуалната машина няма да го открие и ще се получи съобщение за грешка. Параметрите, които се указват за свързването в базата са примерни и трябва да се настроят за вашия сървър. Повече информация за тях може да се намери в документацията. Ето и един примерен резултат от изпълнението на JSP скрипта Search.jsp, който използва класа SearchBean за извличане на запис номер 9814566 от таблицата `emails` от ORACLE-базата:



Ако някой е програмирал с ASP (Active Server Pages на Microsoft), може би веднага ще забележи удивителната прилика между JSP и ASP. С малки изключения разликата е само в езика, на който се пише кода – при ASP се ползва VBScript или JScript, а при JSP – JAVA. Основните идеи и на двете

технологии са едни, дори голяма част от таговете съвпадат. Пълно сравнение на JSP с ASP има на: http://java.sun.com/products/jsp/jsp-asp.html .

Мощността на езика JAVA и технологиите, които JAVA предлага позволяват създаването на големи и сложни приложения, които са достъпни и през WEB (посредством JSP), като гарантира надеждност, стабилност, съвместимост и преносимост. Комбинацията на JSP с bean-ове, Enterprise Java Beans, RMI и използването на XML правят приложението съвременно и на едно наистина професионално ниво без да изискват много сериозни програмистки умения. Ето защо JSP е една мощна съвременна технология, която обаче със сигурност няма да е последна в този динамично развиващ се сектор на софтуерните технологии, понеже все още има какво още да се подобри, за да се направи животът на INTERNET програмиста още по-лесен.

Всички примерни скриптове и bean-ове в настоящата статия са тествани и работят правилно. Използвахме Debian/GNU Linux с JDK 1.2, Apache 1.3.9, Tomcat 3.1, Oracle 8i on Windows NT 4.0 и Internet Explorer 5.0.

Повече информация за JSP можете да намерите на следните адреси:
http://www.servlets.com/resources/urls/document.html
http://java.sun.com/products/jsp
http://www.esperanto.org.nz/jsp