JSP

مقدمه

JSP[1] یک محصول نیست بلکه مشابه سایر محصولات Java API، یک استاندارد است که توسط شرکت سان میکروسیستم برای شرکت‌های پیاده‌کننده سرویس‌دهنده‌های جاوا ارائه شده‌است.

هدف JSP ساده‌کردن نحوه ایجاد و مدیریت صفحات پویای وب است. JSP اجازه ترکیب کد HTML با کدهای جاوا در یک متن را می‌دهد. کد جاوا در یک تگ خاص قرار می‌گیرد که به نگهدارنده JSP می‌گوید از این کد باید برای تولید سرولت استفاده کند، به همین دلیل استاندارد JSP براساس مشخصات سرولت نوشته شده‌است ولی این دو تفاوت‌های زیر را با یکدیگر دارند:

- سرولت راه حل (جایگزین) جاوا برای CGI است. سرولت روی ماشین سرویس‌دهنده اجرا می شود و به عنوان یک لایه میانی بین درخواست سرویس‌گیرنده و سایر برنامه‌ها قرار می‌گیرد. سرولت محتویات پویا را با قسمت‌های ثابت ترکیب و خروجی HTML را تولید می‌نماید.
- JSP پیرو مدل J2EE است. جداسازی قسمت‌های پویا و ثابت از یکدیگر و همچنین لایه نمایش از لایه منطق یک برنامه وب، هدف اصلی JSP است.
- در سرولت برنامه‌نویس وادار به ترکیب کد و قسمت‌های ثابت می شود، ولی در JSP می‌تواند با استفاده از bean منطق برنامه را در یک کلاس پیاده‌سازی کرده و با به کار گرفتن تگ‌های JSP خروجی کلاس را نیز در بین تگ‌های HTML قرار دهد .

---

[1] Java Server Page

ایده اصلی JSP جداسازی دو بخش محتوا و نمایش برای سهولت در ایجاد و مدیریت صفحات پویا وب است. JSP ترکیبی از HTML (یا XML) و تگ‌های اسکریپت[1] است. هر فایل JSP در اولین مرتبه فراخوانی توسط سرویس‌گیرنده، به سرولت جاوا تبدیل می شود. این سرولت ترکیبی از HTML فایل JSP و محتویات پویا تعریف شده توسط تگ‌های JSP است، بنابراین هر فایل JSP باید حاوی HTML باشد، اما یک صفحه JSP می‌تواند فقط حاوی کد جاوا باشد. در ادامه به شرح هریک از موارد زیر می پردازیم.

- ساختن بلاک‌های صفحات JSP شامل Directiveها، اجزای Scripting و Actionهای استاندارد.
- اشیاء انتزاعی JSP
- طراحی برنامه های مبتنی بر JSP

یک JSP ساده می‌تواند مشابه کد زیر باشد :

```
<%@page import="java.util.Date"%>
<html>
<body>
The current time is <%= new Date().toString() %>
</body>
</html>
```

کد 18-1 JSP ساده

به فرض کد بالا در فایلی به نام SimpleJsp.jsp در دایرکتوری C:\JavaPro\Chapter18\JSPExample ذخیره شده است. خطوط زیر باید در فایل servlet.xml در دایرکتوری conf سرویس‌دهنده Tomcat وارد شود.

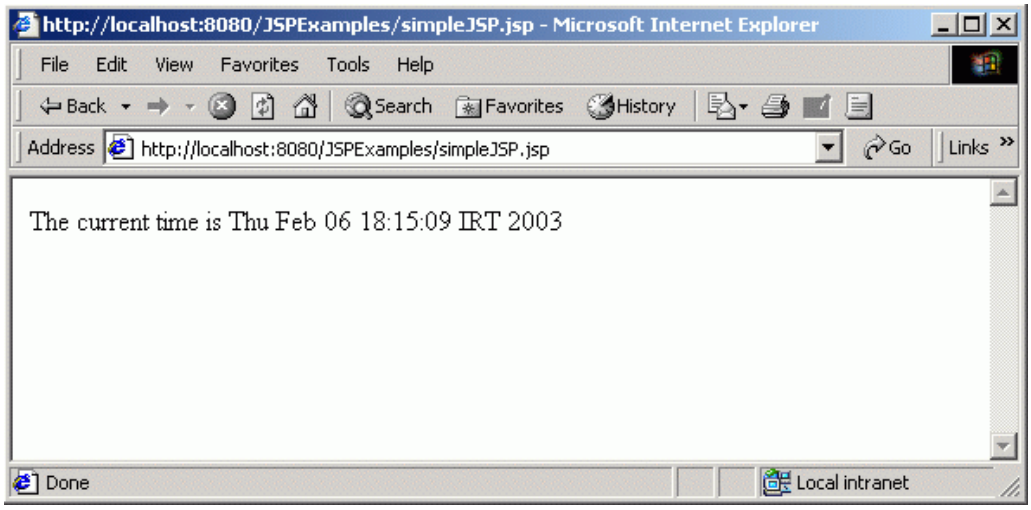
```
<context path="/JSPExamples"
      docBase="C:\JavaPro/Chapter18/JSPExamples">
</context>
```

[1] Script Tags

کد بالا باعث می شود که سرویس‌دهنده وب آدرس‌هایی که به عبارت /JSPExample ختم می‌شوند را در دایرکتوری C:\JavaPro\Chapter18\JSPExample جستجو کند. پس از راه‌اندازی Tomcat، آدرس زیر را در مروگر وارد کنید :

http://localhost:8080/JSPExample/SimpleJSP.jsp

خروجی زیر مشاهده می شود.



شکل 18-1 خروجی JSP

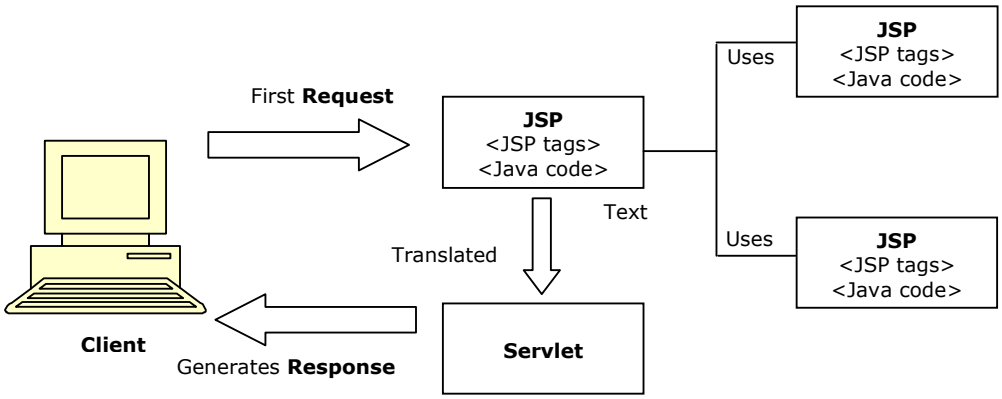
اولین باری که موتور JSP درخواست یک JSP را دریافت می‌کند، آن JSP و فایل‌های وابسته به آن را کامپایل کرده و به سرولت تبدیل می‌کند. سپس سرولت تولید شده را کامپایل کرده و کلاس آن را ایجاد می‌کند. کلاس ایجاد شده از مثال قبلی در سرویس گیرنده Tomcat به صورت زیر است.

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;
import java.util.Date;

public class simpleJSP_jsp extends HttpJspBase {
   private static java.util.Vector _jspx_includes;
   public java.util.List getIncludes() {
       return _jspx_includes;
   }

public void _jspService(HttpServletRequest request, HttpServletResponse response)
            throws java.io.IOException, ServletException {

      JspFactory _jspxFactory = null;
      javax.servlet.jsp.PageContext pageContext = null;
      HttpSession session = null;
      ServletContext application = null;
```

```
        try {
            _JspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html;charset=ISO-8859-1");
            pageContext = _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;

            out.write("\r\n");
            out.write("<html>\r\n");
            out.write("<body>\r\nThe current time is ");
            out.print( new Date().toString());
            out.write("\r\n");
            out.write("</body>\r\n");
            out.write("</html>");
        } catch (Throwable t) {
            out = _jspx_out;
            if (out != null && out.getBufferSize() != 0)
                out.clearBuffer();
            if (pageContext != null) pageContext.handlePageException(t);
        } finally {
            if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
        }
    }
}
```

کد 2-18 سرولت تولیدشده از مرحله کامپایل JSP

به عبارت دیگر اولین باری که JSP توسط حامل JSP (موتور JSP) فراخوانی می شود، کد سرولت معادل تگ‌های JSP تولید، کامپایل و در حامل سرولت بارگذاری می شود. از این به بعد تا زمانی که کد JSP تغییر نکند، کلاس سرولت کامپایل شده، درخواست های سرویس‌گیرنده را پردازش می‌کند. اگر کد JSP تغییر کند، به طور اتوماتیک دوباره کامپایل و با فراخوانی بعدی JSP به روز می شود (به همین دلیل اولین اجرای JSP زمان بیشتری به طول می‌انجامد).شکل زیر این مراحل را نشان می‌دهد.



شکل 2-18 مراحل فراخوانی تا اجرای یک JSP

همچنین شکل بالا نشان‌دهنده آن است که یک JSP می‌تواند به صورت اتوماتیک JSP دیگر را در بدنه خود فراخوانی کند. کلاس سرولت در انتهای مرحله کامپایل رابط مناسب بین حامل و JSP را نشان می‌دهد. برطبق استاندارد، JSP باید دارای یکی از سوپرکلاس های زیر باشد.

- سوپرکلاس در خود بدنه JSP با دستور page تعریف شود.
- کلاس پیاده‌کننده حامل JSP کد رابط javax.Servlet.jsp.JspPage را پیاده‌سازی می‌کند (از آنجایی که اکثر صفحات JSP از پروتکل HTTP استفاده می‌کنند، کلاس های پیاده‌کننده آنها باید رابط javax.Servlet.jsp.HttpJspPage را که از رابط javax.Servlet.JSP.JspPage ارث برده‌است، به طور کامل پیاده سازی کنند).

رابط javax.Servlet.jsp.JspPage دارای دو متد زیر است:

- public void jspInit()
  این متد هنگامی که JSP مقداردهی اولیه می شود فراخوانی شده و مشابه متد init() در سرولت است. برنامه‌نویس می‌تواند این متد را در JSP به صورت صریح پیاده‌سازی نماید.

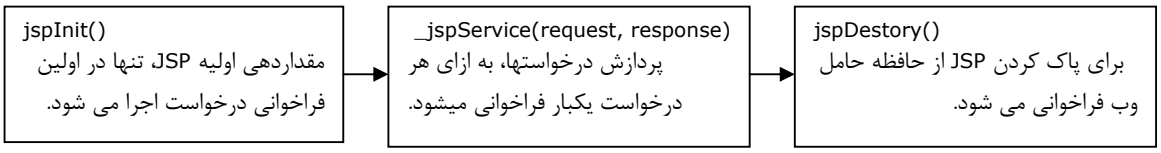
- public void jspDestroy()
  این متد هنگامی که JSP از حامل آن حذف می شود فراخوانی شده و مشابه متد destroy() در سرولت است. برنامه‌نویس برای آزادسازی منابع می‌تواند این متد را به طور صریح پیاده‌سازی نماید.

رابط فقط دارای متد زیر است.

```
public void _jspService(HttpServletRequest request,
        HttpServletResponse response) throws
                        ServletException,IOException
```

این متد متناظر با بدنه صفحه JSP و مشابه متد service() در سرولت است.
پیاده‌سازی این متد توسط حامل JSP صورت می‌گیرد و برنامه‌نویس نمی‌تواند به طور صریح آن را پیاده‌سازی کند.

این سه متد اصلی در چرخه حیات JSP با یکدیگر کار می‌کنند.



شکل 3-18 چرخه حیات JSP

- ابتدا صفحه با فراخوانی jspInit() مقداردهی اولیه می شود، این متد می‌تواند توسط برنامه‌نویس نیز تعریف شود. نحوه مقداردهی اولیه بسیار مشابه مقداردهی اولیه سرولت است، یعنی این عمل در اولین فراخوانی JSP، صورت می‌گیرد.
- به ازای هر درخواست JSP، متد _jspService() فراخوانی شده، درخواست سرویس‌گیرنده را پردازش کرده و خروجی مناسب را تولید می‌کند. این پاسخ از طریق حامل JSP به سرویس‌گیرنده فرستاده می شود.
- زمانی که JSP از سرویس‌دهنده حذف شود (برای مثال هنگام خاموش کردن یا stop کردن سرویس‌دهنده وب)، متد jspDestroy() که می‌تواند توسط برنامه‌نویس نیز پیاده‌سازی شود جهت آزادسازی منابع فراخوانی می شود.

در بسیاری از موارد نیازی به نوشتن متدهای jspInit و jspDestroy() در بدنه JSP نیست.

مبانی JSP

ساختار یک فایل JSP مابین یک سرولت و یک صفحه HTML است . کدهای JSP بین تگ‌های <% و %> و سایر تگ‌های XML قرار می‌گیرد.

تگ‌های JSP به سه دسته زیر تقسیم می‌شوند:

- **Directive**: این نوع تگ‌ها بر تمام ساختار صفحه سرولت به دست آمده از JSP تاثیر می‌گذارند.

- **Scripting Element**: این تگ‌ها امکان نوشتن کد جاوا را در داخل صفحه JSP فراهم می‌کنند.
- **Action**: تگ‌های خاصی هستند که در زمان اجرا به روی JSP تاثیر می‌گذارند.
- **Custom Tag**: تگ‌های سفارشی هستند که توسط برنامه‌نویس نوشته می‌شوند.

برخی از قواعد حاکم بر صفحات JSP بدین شرح می‌باشند:

- تگ‌های JSP نسبت به کوچک و بزرگ بودن حروف حساس هستند.
- Directive و Scripting Element دارای دو نحوه نگارش معمولی و مبتنی به XML هستند.
- تگ‌های مبتنی بر XML شامل تگ ، مشخصه‌های اختیاری و تگ پایان و یا تگ خالی هستند. مثال :

```
<somejsptag atrributename="attribute value">
body
</somejsptag>
```

یـا

```
<somejsptag atrributename="attribute value" />
```

- مقادیر مشخصه‌ها با استفاده از کاراکترهای '' یا "" همیشه به صورت نقل قول ذکر می‌شوند. اگر مقدار مشخصه دارای یکی از کاراکترهای ' یا " باشد، باید از عبارت‌های ' و " استفاده کرد.
- وجود فضای خالی دربدنه صفحه JSP اهمیت ندارد، اما در زمان ترجمه به سرولت در بدنه آن باقی می‌ماند.
- کاراکتر '\' به عنوان کاراکتر فرار[1] در تگ‌ها استفاده می شود (به عنوان مثال برای استفاده از '%' باید '%\' نوشت).
- آدرس‌های استفاده‌شده در JSP مطابق با نحوه استفاده از آنها در سرولت است. آدرسی که با یک '/' شروع شود مسیر نسبی نامیده می شود و نسبت به آدرس برنامه وب که JSP به آن تعلق دارد تفسیر می شود.

[1] Escape

دستورات JSP

دستورات JSP برای ارسال پیغام از صفحه JSP به حامل آن درنظر گرفته شده‌است. برای تعیین متغیرهای عمومی مانند معرفی کلاس، نوع خروجی و codepage خروجی استفاده می شود و منجر به تولید خروجی برای سرویس‌گیرنده نمی‌شوند. محدوده تعریف[1] یک دستور، کل صفحه JSP است و با کاراکتر '<%@' شروع و با '%>' پایان می‌یابد .

نحوه نگارش کلی یک دستور بدین شکل است:

```
<%@ directivename attribute="value" attribute="value" %>
```

از سه نوع دستور در JSP می‌توان استفاده کرد:

- دستور page
- دستور include
- دستور taglib

دستور page

این دستور برای تعریف و تغییر تعدادی از مشخصه‌های وابسته به صفحه و تاثیرگذار در تمام JSP مورد استفاده قرار می‌گیرد. یک صفحه JSP می‌تواند شامل چندین دستور page باشد و این دستورات در هر جای صفحه می‌توانند قرار گیرند. اگرچه هریک از مشخصات و مقادیر آنها فقط یک بار می‌تواند در هر دستور ذکر شود. فقط مشخصه import از این امر مستثنی است و می‌تواند در یک دستور چندین بار تکرار شود.
نحوه کلی نگارش دســتور page بدین شکل است:

```
<%@ page ATTRIBUTES %>
```

به طوریکه ATTRIBUTE های معتبر از یک نام و یک مقدار تشکیل شده‌اند و عبارتند از:

- **language** مشخص کننده زبان اسکریپت مورد استفاده در JSP است. این مشخصه امکان استفاده از چند زبان را در یک JSP فراهم می‌کند. مقدار پیش‌فرض آن "Java" است.

[1] Scope

- **extends** نام کامل سوپرکلاس سرولت تولید شده‌است (هنگام ترجمه JSP به سرولت). فاقد مقدار پیش‌فرض است.

- **import** مشابه import در جاوا، لیستی از کلاس ها و بسته‌ها (با کاما (،) از یکدیگر جدا می‌شوند) را مشخص می‌کند. فاقد مقدار پیش‌فرض است.

- **session** تعیین می‌کند که صفحه در HTTP session قرار می‌گیرد. اگر مقدار آن "true" باشد، شیی session (به javax.Servlet.http.HttpSession اشاره می‌کند) قابل دسترسی است و اگر مقدار آن "false" باشد، صفحه در http session قرار نمی‌گیرد و شیی session قابل دسترسی نخواهد بود. مقدار پیش‌فرض آن "true" است.

- **buffer** نوع بافرکردن صفحه برای نمایش خروجی به سرویس‌گیرنده را مشخص می‌کند. اگر مقدار "none" ذکر شود، عمل بافرکردن صورت نمی‌گیرد و تمام خروجی با استفاده از PrintWriter به طور مستقیم در ServletResponse نوشته می شود. اگر مقدار بافر مشخص شود (برای مثال "24kb") خروجی در بافرهایی کمتر از مقدار ذکر شده قرار می‌گیرد. مقدار پیش‌فرض آن بستگی به نوع پیاده‌سازی دارد.

- **autoFalsh** اگر مقدار آن "true" باشد، بافر خروجی هنگام پرشدن به صورت اتوماتیک خالی می شود و اگر "false" باشد، استثناء سرریزی (overflow) در طی اجرا رخ می‌دهد. مقدار پیش‌فرض آن "true" است.

- **isThreadSafe** سطح امنیت thread را مشخص می‌کند . اگر مقدار آن "true" باشد موتور JSP می‌تواند درخواست های چندین سرویس‌گیرنده را به صورت همزمان پاسخ دهد و اگر "false" باشد، موتور JSP درخواست ها را در یک صف قرار می‌دهد و به نوبت ورود آنها را پردازش می‌کند. این روش مشابه پیاده‌سازی رابط javax.servlet.SingleThreadModel بوده و مقدار پیش‌فرض آن "true" است.

- **info** مشابه متد Servlet.getServletInfo()، حاوی اطلاعات مفیدی راجع به صفحه می‌باشد. فاقد مقدار پیش‌فرض است.
- **errorPage** آدرس فایل JSP کنترل‌کننده خطاهای صفحه جاری است. صفحه اصلی یک نمونه از شیی java.lang.throwable را به صفحه خطا می‌فرستد. فاقد مقدار پیش‌فرض است.

- **isErrorPage** مشخص‌کننده آن است که صفحه جاری در صفحات دیگر به عنوان errorpage تعریف شده‌است. اگر مقدار آن "true" باشد متغییر exception که به java.lang.throwable اشاره می‌کند قابل دسترس خواهدبود. مقدار پیش‌فرض آن "false" است.

- **contentType** نوع کدبندی کاراکتر[1] را برای صفحه JSP و نوع MIME را برای نمایش صفحه مشخص می‌کند. مقدار این مشخصه می‌تواند به صورت "MIMETYPE" یا "MIMETYPE;charset=CHARSET" ذکر شود. مقدار پیش‌فرض MIMETYPE عبارت "text/html" و CHARSET مقدار ISO-8859-1 است.

مثال:

```
<%@ page language="Java" import="java.rmi.*,java.util.*"
   session="true" buffer="12kb" autoFlush="true"
   info="my page directive jsp" errorPage="error.page"
   isErrorPage="false" isThreadSafe="false"%>
<html>
<head>
   <title>Page Directive test page</title>
</head>
<body>
   <h1> Page directive test page </h1>
   This is a JSP to test the page directive.
</body>
</html>
```

کد 3-18 نحوه استفاده از دستور Page

دستورinclude

[1] Character Encoding

این دستور به حامل JSP می‌گوید که محتوای یک فایل را در محل ذکر شده این دستور در صفحه JSP جاری قرار دهد. فایل مشخص شده باید برای حامل JSP قابل دسترس باشد.

نکته مهم این است که JSP محتوای فایل داخل شده را یک بار فقط در زمان ترجمه (تبدیلjsp به سرولت) مورد تجزیه و تحلیل قرار می‌دهد. بیشتر حامل‌های JSP فایل داخل شده را تحت نظرگرفته و درصورت تغییر آن را دوباره کامپایل می‌کنند.

نحوه نگارش دستور include چنین است:

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

مشخصه file نام فایل داخل‌شده را مشخص می‌کند. فایل داخل‌شده می‌تواند یک فایل ثابت (مثل html) یا یک JSP دیگر باشد. در مثال زیر برنامه includeDirective1.jsp به هنگام کامپایل تقاضا کرده‌است که محتویات فایل ثابت copyright.html داخل صفحه شود.

```
<html>
<head>
<title>Include directive test page 1</title>
</head>
<body>
<h1>Include directive test page 1</h1>
<%@ include file="copyright.html" %>
</body>
</html>
```

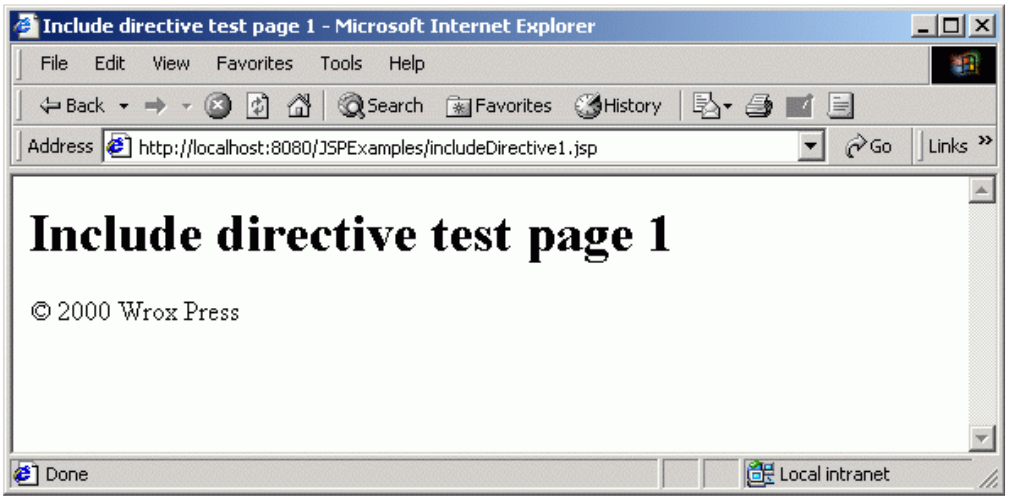
کد 4-18 نحوه استفاده از دستور include

محتویات فایل copyright.html چنین است.

```
<p>&copy; 2000 Wrox Press</p>
```

کد 5-18 محتویات فایل HTML

پس از ذخیره کردن فایل‌ها در مسیر مورد نظر کافی است آدرس
http://localhost:8080/jspExample/includeDirective1.jsp
درمرورگر واردشود تا خروجی زیر حاصل شود.



شکل 4-18 خروجی دستور include

مثال بعدی چگونگی داخل‌شدن یک فایل JSP را نشان می‌دهد. به فرض فایل includeDirective2.jsp حاوی کد زیر باشد.

```
<html>
<head>
   <title>Include directive test page 2</title>
</head>
<body>
   <h1>Include directive test page 2</h1>
   <%@ include file="included.jsp"%>
</body>
</html>
```

کد 6-18 نحوه داخل شدن یک فایل JSP

محتویات فایل داخل شده included.jsp چنین است:

```
<%@page import="java.util.Date"%>
<%= "Current date is " + new Date().toString()%>
```

کد 7-18 محتویات فایل JSP

در اینجا نیز پس از ذخیره کردن فایل‌ها در مسیر مورد نظر کافی است آدرس
`http://localhost:8080/JSPExample/includeDirective2.jsp` در
مرورگر وارد شود تا خروجی زیر حاصل شود.

شکل 5-18 نمایش محتویات فایل JSP

پس از تبدیل `includeDirective2.jsp` به سرولت نتیجه زیر به دست می‌آید.

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.util.Date;
public class includeAction_jsp {
// ...
   public void _jspService(HttpServletRequest request,
         HttpServletResponse response) throws
            java.io.IOException, ServletException {
      // ...
      out.write("<html>\r\n");
      out.write("<head>\r\n");
      out.write("<title>Include directive test page</title>\r\n");
      out.write("</head>\r\n");
      out.write("<body>\r\n");
      out.write("<h1>Include directive test page </h1>\r\n\r\n");
      out.write("<h2>Using the include directive </h2>\r\n\r\n");
      out.write("<h3>This is some static text in the html file </h3>\r\n\r\n");
      out.print( "Current date is " + new Date() );
      out.write("\r\n\r\n <h2>Using the include action</h2>\r\n\r\n");
      {
         String _jspx_qStr = "";
         out.flush();
         pageContext.include("include2.html" + _jspx_qStr);
       }
      out.write("\r\n");
      {
         String _jspx_qStr = "";
         out.flush();
         pageContext.include("include2.jsp" + _jspx_qStr);
       }
      out.write("\r\n\r\n </body>\r\rn</html>\r\n");
```

کد 8-18 بخشی از سرولت حاصل از دستور include

دستور taglib

این دستور به صفحه jsp امکان استفاده از تگ‌های سفارشی را می‌دهد. تگ سفارشی شامل کدهای جاوا است که توسط برنامه‌نویس تعریف شده‌است. به عبارت دیگر jsp به برنامه‌نویس اجازه ساخت تگ‌های مختص به خود را می‌دهد. نحوه نگارش کلی دستور taglib به صورت زیر است.

```
<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>
```

مشخصه‌های قابل دسترس این دستور عبارتند از:

- uri : یک URI[1] آدرس توصیف‌گر تگ سفارشی است. این توصیف‌گر برای منحصر بفردسازی نام تگ سفارشی و شرح وظایف آن به حامل JSP مناسب است. فاقد مقدار پیش‌فرض بوده و عدم ذکر آن باعث بروز خطای زمان کامپایل می شود.

- tagPrefix : برای تعریف تگ‌های سفارشی از عبارت prefix:tagname استفاده می شود. پیشوندهای sun، jspx، jsp، java، javax، servlet و sunw رزرو شده‌اند و نمی‌توان از آنها استفاده کرد. برای مثال اگر این مقدار mytag باشد حامل با برخورد به هر عنصری که با <mytag:tagname ... /> شروع شود به تگ سفارشی مشخص شده در آدرس uri مراجعه می‌کند. فاقد مقدار پیش‌فرض بوده و عدم ذکر آن باعث بروز خطای زمان کامپایل می شود.

اجزای Scripting

این اجزاء امکان استفاده از کد جاوا جهت تعریف متغیر، متد و عبارات مورد نیاز را در بدنه صفحه JSP فراهم می‌کنند و به شرح زیر می‌باشند.

[1] Uniform Resource Identifier

تعاریف

Declaration بلاکی از کدهای جاوا است که برای تعریف متدها و متغیرها به کار گرفته می شود. هر بلاک تعریف بین تگ‌های <% و %> قرار گرفته و هیچ خروجی روی صفحه تولید نمی‌کند. نحوه نگارش بلاک تعریف به صورت زیر است.

```
<%! Java variable and method declaration(s) %>
```

مثال :

```
<%!
   int numTimes = 3;
   public String sayHello(String name) {
      return "Hello, " + name + "!";
   }
%>
<html>
<head>
   <title>Declaration test page</title>
</head>
<body>
   <h1>Declaration test page</h1>
   <p>The value of numTimes is <%= numTimes %>.</p>
   <p>Saying hello to reader : "<%= sayHello("reader")%>".</p>
</body>
```

کد 9-18 نحوه استفاده از Declaration

در مثال بالا یک متغیر به نام numTimes از نوع int و یک متد به نام sayHello() تعریف شده که پیغام خوشامدگویی را چاپ می‌کند. اگر کد بالا در فایل declaration.jsp در مسیر مورد نظر ذخیره شود با وارد کردن آدرس http://localhost:8080/JSPExamples/declaration.jsp در مرورگر خروجی زیر حاصل می شود.

شکل 6-18 خروجی declaration.jsp

سرولت تولید شده پس از کامپایل jsp بالا چنین است.

```
import javax.servlet.*;
// ... more import statements
public class declaration_jsp extends HttpJspBase {
    int numTimes = 3;
    public String sayHello(String name) {
        return "Hello, " + name + "!";
    }
    // ... more generated code
}
```

کد 10-18 سرولت حاصل از کامپایل declaration.jsp

Scriptlet

Scriptlet بلاکی از کدهای جاوا است که در زمان پردازش درخواست، اجرا می شود. هر بلاک scriptlet بین تگ‌های <% و %> قرار می‌گیرد. در حقیقت آنچه که scriptlet انجام می‌دهد بستگی به کد خودش دارد و می‌تواند شامل کد تولیدکننده خروجی برای سرویس‌گیرنده باشد. scriptletها به همان ترتیبی که در JSP ظاهر شده‌اند در کلاس Servlet مربوطه قرار می‌گیرند.

در سرویس دهنده Tomcat کدهایی که در تگ‌های <% و %> ظاهر می‌شوند، در متد service() کلاس Servlet قرار می‌گیرند. نحوه نگارش scriptlet به صورت زیر است .

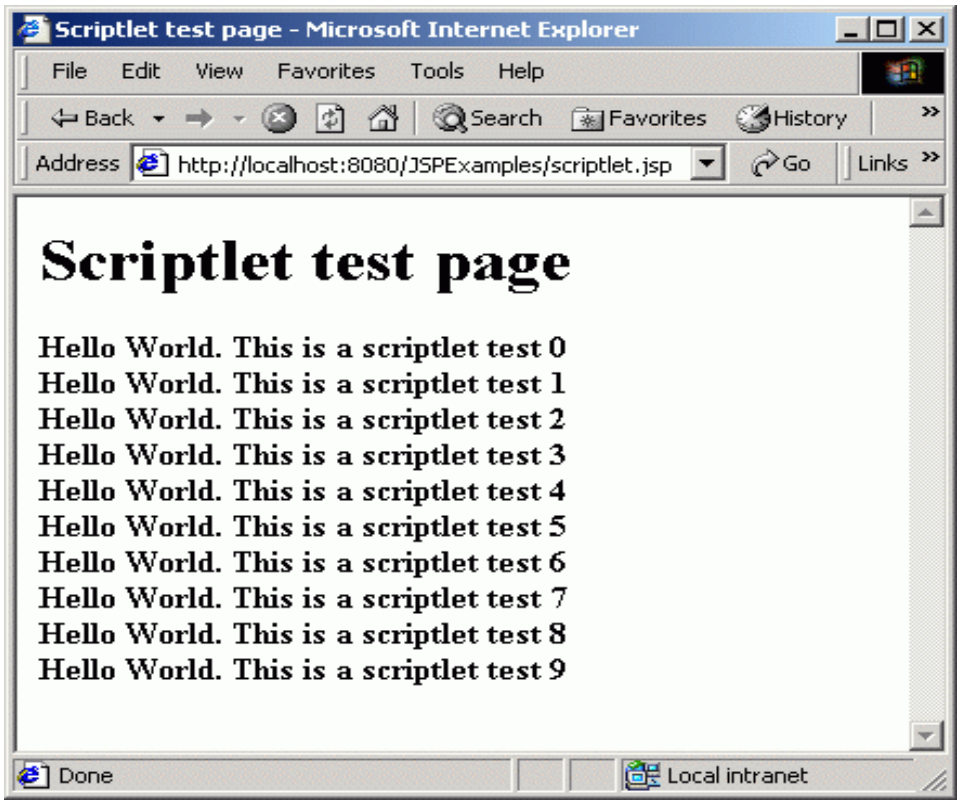
```
<% Valid Java code statement %>
```

در مثال زیر یک scriptlet ده بار اجرا شده و هر بار یک پیغام را روی پنجره مرورگر (شیی out) و کنسول Tomcat (جریان System.out) چاپ می‌کند.

```
<html>
<head>
   <title>Scriptlet test page</title>
</head>
<body>
   <h1>Scriptlet test page</h1>
   <%
   for(int i=0; i < 10; i++){
      out.println("<b> Hello World. This is a scriptlet test " + i + "</b><br>");
      System.out.println("This goes to the System.out stream" + i);
   }
   %>
</body>
</html>
```

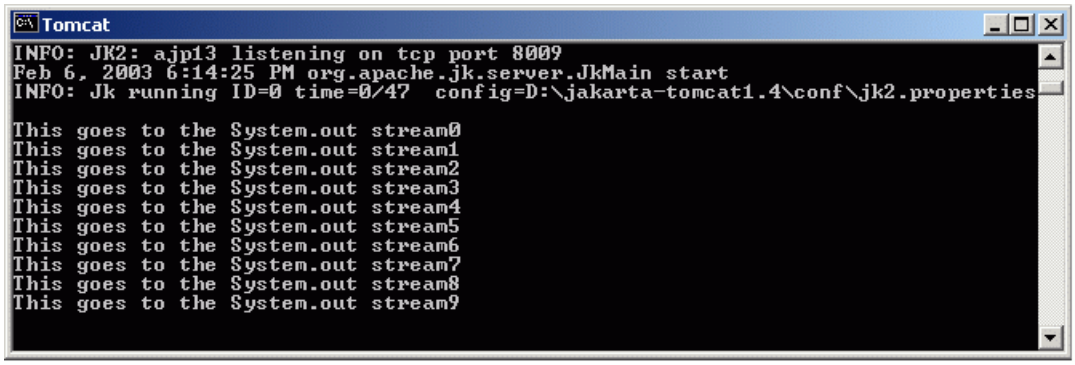
کد 11-18 نحوه استفاده از Scriptlet

اگر کد بالا در فایل scriptlet.jsp و در مسیر مورد نظر ذخیره شود با واردکردن آدرس http://localhost:8080/JSPExamples/scriptlet.jsp در مرورگر خروجی زیر حاصل می شود.



شکل 7-18 خروجی `scriptlet.jsp` در مرورگر

و خروجی آن روی کنسول Tomcat چنین است.



شکل 8-18 خروجی `scriptlet.jsp` روی کنسول Tomcat

Expression

Expression نماد خلاصه شده یک scriptlet است که مقدار یک عبارت جاوا را به سرویس‌گیرنده برمی‌گرداند. هر Expression در زمان پردازش درخواست http ارزیابی شده و نتیجه آن پس از تبدیل شدن به String نمایش داده می شود. Expression بین تگ‌های <% و =%> قرار می‌گیرد، اگر نتیجه اجرای آن یک شیی باشد، با استفاده از متد toString() عمل تبدیل روی آن صورت می‌گیرد.

نحوه نگارش آن بدین صورت است.

```
<%= Java expression to be evaluated %>
```

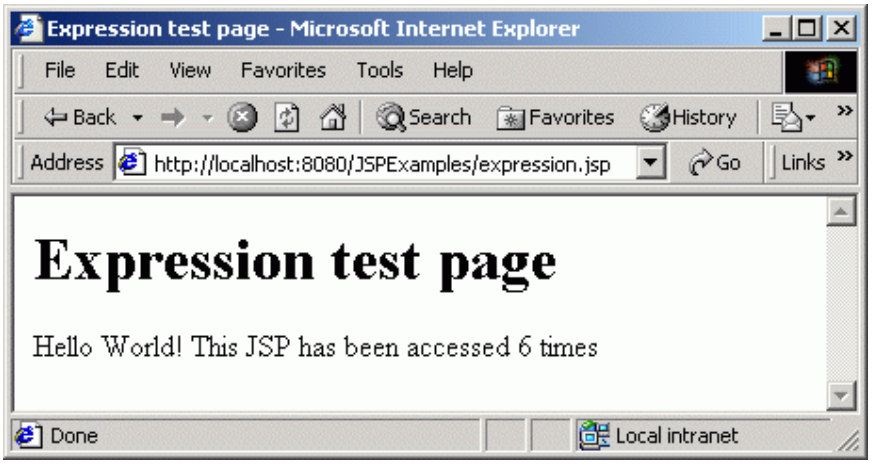
مثال زیر یک شمارنده ساده ایجاد کرده و نحوه کار declaraion، scriptlet و expression را با یکدیگر نمایش می‌دهد.

```
<html>
<head>
   <title>Expression test page</title>
</head>
<body>
   <h1>Expression test page</h1>
   <%! int i =0;%>
   <%
   i++;
   %>
   Hello World!
   <%= "This JSP has been accessed " + i + " times" %>
</body>
</html>
```

کد 12-18 ایجاد یک شـمارنده ساده

xpression.jsp در مسیر مورد نظر ذخیره می شود. در مثال بالا یک متغیر int به نام i تعریف و با عدد صفر مقداردهی اولیه شده‌است. هر زمان که نمونه سرولت ایجاد شده این کد فراخوانی شود (درخواست آدرس http://localhost:8080/JSPExamples/expression.jsp در مرورگر)، scriptlet نوشته شده یک واحد به متغیر i اضافه می‌کند. در نهایت یک expression

برای چاپ پیغام مناسب و مقدار i مورد استفاده قرار گرفته‌است. پس از چند بار فراخوانی مثال بالا آنچه که در مروگر دیده می شود، بدین شکل خواهد بود.



شکل 9-18 خروجی شمـارنده

Actionهای استاندارد

این Actionها روی رفتار JSP در زمان اجرا تاثیر گذاشته و درنهایت پاسخ مناسب برای سرویس‌گیرنده ارسال می شود. صرف‌نظر از نحوه پیاده‌سازی، کلیه حامل‌ها باید Actionهای استاندارد را تهیه کنند. در حقیقت یک action استاندارد، تگی است که می‌تواند در صفحه JSP قرار گیرد. در طی تبدیل‌شدن JSP به سرولت، حامل این تگ‌ها را شناسایی کرده و بجای آنها کد جاوا را طوری قرار می‌دهد که همان وظایف را انجام دهند.

کد زیر بیانگر یک action استاندارد است.

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

این action فایل myjsp.jsp را گرفته، درصورت لزوم کامپایل کرده و خروجی تولید شده توسط آن را بجای تگ action در پاسخ قرار می‌دهد. جزییات بیشتر آن در ادامه مطالب ذکر شده‌است. انواع Actionهای استاندارد عبارتند از:

- <jsp:useBean>
- <jsp:setProperty>
- <jsp:getProperty>
- <jsp:param>

- <jsp:include>
- <jsp:forward>
- <jsp:plugin>

<jsp:useBean>

برای جداسازی کد از نمایش، بهترین راه حل کپسوله کردن کدهای جاوا در یک Bean و نمونه‌سازی از آن در صفحات jsp است. <jsp:useBean>، <jsp:setProperty> و <jsp:getProperty> برای استفاده از javaBean در صفحات وب طراحی شده‌اند. از <jsp:useBean> برای نمونه‌سازی یک JavaBean یا تعیین یک نمونه موجود و انتساب یک نام متغیر (یا id) به آن مورد استفاده قرار می‌گیرد. همچنین می‌توان با دادن یک scope خاص به آن شیی، چرخه حیات آن را مشخص‌کرد (در این رابطه بعدها بیشتر توضیح داده شده‌است). <jsp:useBean> اطمینان دارد که شیی با id خاص در محدوده مناسب و مشخص‌شده در این تگ قابل دسترس خواهد بود. سپس این شیی می‌تواند توسط id مربوطه و براساس scope مشخص‌شده در داخل آن JSP یا دیگری مورد رجوع قرارگیرد.

نحوه نگارش <jsp:useBean> بدین شکل است.

```
<jsp:useBean id="name" scope="scopeName" beandetails />
```

که beandetails یکی از موارد زیر می‌تواند باشد.

- class="className"
- class="classname" type="typeName"
- beanName="beanName" type="typeName"
- type="typeName"

مشخصه‌های تعریف شده در <jsp:useBean> عبارتند از:

- id نامی است حساس نسبت به کوچک و بزرگ بودن حروف و جهت معرفی نمونه شیی مورد استفاده قرار می‌گیرد.این مشخصه فاقد مقدار پیش‌فرض است.
- scope معرف محدوده‌ای است که رجوع به نمونه در آن معتبر است. مقادیر "request"، "page"، "session" و "application" را می‌توان مشخص کرد (در ادامه هر یک

از این مقادیر به طور خلاصه توضیح داده شده‌اند). مقدار پیش‌فرض این مشخصه، "page " است.

- class معرف نام کامل و اصلی کلاس Bean بوده و فاقد مقدار پیش‌فرض است.
- beanName معرف نامی است که در متد instantiate() موجود در کلاس java.beans.Bean آن bean ذکر شده‌است. درصورت ذکر شدن مشخصات beanName و type می‌توان از ذکر مشخصه class اجتناب کرد. مشخصه beanName از قواعد bean استاندارد پیروی می‌کند و می‌تواند به شکل "a.b.c" باشد به طوریکه "a.b.c" یا یک کلاس است یا نام یک منبع سریال شده‌است که به عنوان "a/b/c.ser" شناسایی خواهد شد. این مشخصه فاقد مقدار پیش‌فرض است.
- Type این مشخصه اختیاری نوع متغیر script ایجاد شده را مشخص می‌کند و از قواعد تبدیل نوع استاندارد جاوا پیروی می‌کند. نوع ذکر شده باید سوپرکلاس bean، رابط پیاده‌سازی شده توسط bean یا خود کلاس bean باشد. شبیه به سایر عملیات تبدیل نوع، اگر شیی از نوع مشخص شده نباشد، استثناء java.lang.Class.CastException به هنگام درخواست رخ خواهد داد. مقدار ذکر شده در مشخصه Class به عنوان پیش‌فرض برای این مشخصه درنظر گرفته می شود.

مراحلی که حامل JSP طی می‌کند تا کلاس bean را پیدا کند بدین شرح است:

- حامل سعی می‌کند یک شیی با شناسه id در محدوده scope پیدا کند.
- اگر شیی پیدا شود و type آن نیز مشخص شده باشد، حامل سعی می‌کند نوع شیی یافت‌شده را به نوع مشخص شده در type تبدیل کند. اگر عمل تبدیل انجام نشود، استثناء ClassCastException رخ می‌دهد.
- اگر شیی در محدوده scope پیدا نشود و مشخصات class یا beanName مشخص نشده باشند، استثناء InstantiationException رخ می‌دهد.
- اگر شیی در محدوده scope پیدا نشود و کلاس قابل نمونه‌سازی باشد، این عمل انجام گرفته، یک مرجع به شیی ساخته شده با id داده شده در scope معین شده، ایجاد می‌کند. اگر این عمل با موفقیت انجام نشود، استثناء InstantiationException رخ می‌دهد.
- اگر شیی در محدوده scope پیدا نشود و مشخصه beanName مشخص شده‌باشد، متد instantiate() از java.beans.Beans با آرگومان beanName فراخوانی می شود. اگر این مرحله با موفقیت انجام شود، یک مرجع با id داده شده در scope معین شده، ایجاد می شود.

- اگر instance جدید از bean ایجاد شود و تگ <jsp:useBean> فاقد بدنه خالی باشد، بدنه آن پردازش می شود. در طول این پردازش متغیر جدید مقداردهی اولیه شده و قابل دسترس می شود. درصورت لزوم از scriptlet یا action استاندارد <jsp:setProperty> نیز برای مقداردهی اولیه نمونه bean می‌توان استفاده کرد.

مقادیر ممکن جهت مشخصه scope بدین شرح می‌باشند:

- page بدین مفهوم است که شیی موجود در request در کل صفحه معتبر است.
- request بدین مفهوم است که شیی با request خاص سرویس‌گیرنده همراه شده‌است. اگر درخواست با استفاده از <jsp:forward> برای JSP دیگری ارسال شود یا اینکه JSP دیگری این صفحه را با استفاده از تگ <jsp:include> شامل شود، شیی توسط سایر درخواست های ارسال شده در طول session جاری توسط سرویس‌گیرنده معتبر خواهد بود.
- Application بدین مفهوم است که شیی در سایر صفحات JSP این برنامه وب قابل دسترس است.

حامل JSP در طول اجرای برنامه وب انتظار دارد کلاس های جاوا را در مسیرهای معرفی شده در CLASSPATH پیدا کند. اگر کلاس bean توسط برنامه‌نویس تهیه شده‌باشد، فایل class آن باید یا در دایرکتوری WEB_INT\classes موجود در برنامه وب یا در فایل JAR موجود در دایرکتوری WEB_INF\lib قرارگیرد.

<jsp:setProperty>

از این تگ استاندارد جهت تعیین مقدار خاصیت‌های[1] یک bean همراه با تگ <jsp:useBean> استفاده می شود. خاصیت‌های یک bean می‌تواند ساده یا آرایه باشد و در یکی از سه حالت زیر مقداردهی می‌شوند:

- در زمان درخواست از پارامترهای شیی request
- در زمان درخواست از عبارت‌های جاوا
- از یک مقدار ثابت

[1] Properties

به هنگام تعیین مقدار توسط پارامترهای شیی request، می‌توان کلیه خاصیت‌های یک bean را به یک باره و توسط یک action استاندارد مقداردهی کرد.

```
<jsp:setProperty name="help" property="*" />
```

یا یک خاصیت خاص به طور واضح توسط یک action بدین صورت مقداردهی شود.

```
<jsp:setProperty name="help" property="word" />
```

نحوه نگارش action استاندارد <jsp:setProperty> بدین صورت است.

```
<jsp:setProperty name="beanName" propertydetail />
```

که propertydetails می‌تواند یکی از موارد زیر باشد.

```
property="*"
property="propertyName"
property="propertyName" param="parameterName"
property="propertyName" value="parameterValue"
```

درحالی که propertyValue یک رشته یا یک scriptlet خواهدبود. مشخصه‌های این تگ عبارتند از:

- name نام نمونه bean است که اخیراً باید با استفاده از تگ <jsp:useBean> تعریف شده‌باشد. باید دقت داشت که نام ذکر شده در <jsp:setProperty> و <jsp:useBean> یکسان باشد.
- property نام خاصیتی از bean است که مقدار آن تعیین می شود. اگر مقدار مشخصه property برابر با "*" باشد، سعی می شود مقادیر تمام پارامترهای موجود در شیی request در فیلدهای متناظر (همنام) آن در bean قرارگیرد. برای انجام این کار نام و نوع داده‌ایی خواص bean و پارامترهای درخواست باید یکسان باشد. اگر پارامتر request دارای مقدار "" (خالی) باشد، مقدار خاصیت متناظر آن در bean بدون تغییر می‌ماند.
- param به هنگام تنظیم خاصیت‌های bean از طریق پارامترهای request هیچ لزومی ندارد که این دو همنام باشند. از این مشخصه برای تعیین نام پارامتری از request که قرار است مقدار آن به یک خاصیت از bean نسبت داده‌شود، استفاده می شود. اگر این مشخصه ذکر نشود، فرض بر آن می شود که پارامتر درخواست و خاصیت bean همنام می‌باشند. اگر

هیچ پارامتری با این نام پیدا نشود یا اینکه مقدار آن "" باشد، action هیچ تاثیری روی bean نخواهد گذاشت.

- value مشخص‌کننده مقداری است که به خاصیت bean نسبت داده خواهد شد. این مقدار می‌تواند جزء مشخصه درخواست باشد یا اینکه نتیجه اجرای یک عبارت باشد. یک تگ setProperty نمی‌تواند مشخصه‌های param و value را با هم داشته باشد.

هنگام مقداردهی خاصیت‌های bean با ثابت‌های رشته‌ایی یا پارامترهای request، عمل تبدیل با استفاده از متدهای استاندارد جاوا انجام می شود. به عنوان مثال اگر خاصیت bean از نوع double باشد از متد java.lang.Double.valueOf(String) استفاده می شود.

<jsp:getProperty>

این تگ تکمیل‌کننده تگ <jsp:setProperty> بوده و برای دسترسی به مقادیر خاصیت‌های یک bean مورد استفاده قرار می‌گیرد. این تگ مقدار یک خاصیت را به String تبدیل کرده و آن را در جریان خروجی جهت ارسال به سرویس‌گیرنده، چاپ می‌کند. این عملیات مشابه رفتار متد System.out.println() می‌باشد.

برای تبدیل مقدار خاصیت به String بدین ترتیب عمل می شود:

- اگر مقدار موردنظر یک شیی باشد، از متد toString() استفاده می‌کند.
- اگر مقدار موردنظر از نوع primitive (نوع داده اولیه جاوا) باشد، به طور مستقیم مقدار آن را با استفاده از متد valueOf() کلاس متناظر آن نوع (Integer برای int و ...) به String تبدیل می‌کند.

نحوه نگارش آن چنین است.

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

مشخصه‌های قابل دسترس عبارتند از:

- name نام نمونه bean است که توسط آن می‌توان به خاصیت‌های bean دسترس پیداکرد.
- Property متغیر نمونه تعریف‌شده در bean است که به آن خاصیت می‌گویند.

البته قبل از استفاده <jsp:getProperty> باید با استفاده از <jsp:useBean> یا عملگر new یک نمونه از bean ایجاد کرد.

پس از آشنایی با actionهای <jsp:setProperty>، <jsp:useBean> و <jsp:getProperty> با یک مثال ساده چگونگی استفاده از آنها بررسی می شود. این مثال نام کاربر و زبان برنامه‌نویسی مورد علاقه او را گرفته و براساس انتخاب‌ها رای خود را صادر می‌کند. صفحه HTML آن در فایل beans.html بسیار ساده تعریف و طراحی شده‌است.

```
<html>
<head>
   <title>useBean action test page</title>
</head>
<body>
   <h1>useBean action test page</h1>
   <form method="post" action="beans.jsp">
      <p>Please enter your username:
      <input type="text" name="name">
      <br>What is your favorite programming language?
      <select>
         <option value="Java">Java
         <option value="C++">C++
         <option value="Perl">Perl
      </select>
      </p>
      <p><input type="submit" value="Submit information"></p>
   </form>
</body>
</html>
```

کد 13-18 فایل beans.html

فایل html دو پارامتر به نام‌های name و language به beans.jsp ارسال می‌کند. برنامه beans.jsp نیز ساده است.

```
<jsp:useBean id="languageBean" scope="page" class="LanguageBean" >
<jsp:setProperty name="languageBean" property="*"  />
</jsp:useBean>

<html>
  <head>
    <title>useBean action test result</title>
  </head>
  <body>
    <h1>useBean action test result</h1>
    <p>Hello,<jsp:getProperty name="languageBean"
                              property="name" />.</p>
    <p>Your favorite language is
       <jsp:getProperty name="languageBean"
                  property="language" /></p>

  </body>
</html>
```

کد 14-18 برنامه beans.jsp

تمام کدهای جاوا از بدنه jsp جدا و در کلاس languageBean پیاده‌سازی شده‌است. در این شرایط beans.jsp تنها یک instance از langugeBean با محدوده page ایجاد کرده و با استفاده از مشخصه property="*" در <jsp:setProperty> خاصیت‌های name و language را مقداردهی می‌کند. سپس از تگ <jsp:getProperty> برای بازیابی مقدار خاصیت‌های bean و همچنین خاصیت languageComments استفاده کرده‌است.

کد languageBean.java چنین است.

```
public class LanguageBean {
   private String name;
   private String language;

   public LanguageBean() { }

   public String getLanguage() {
      return language; }

   public void setLanguage(String language) {
      this.language = language; }

   public String getName() {
      return name; }

   public void setName(String name) {
      this.name = name; }

    public String getLanguageComments(){
      if(language.equals("Java")){
         return "The king of OO language.";
      } else if(language.equals("C++")){
         return "Rather too complex for some folks' liking.";
      } else if(language.equals("Perl")){
         return "Ok if you like incomprehensible code.";
      } else {
         return "Sorry, I've never heard of " + language + ".";
      }
   }
}
```

کد 15-18 برنامه languageBean.java

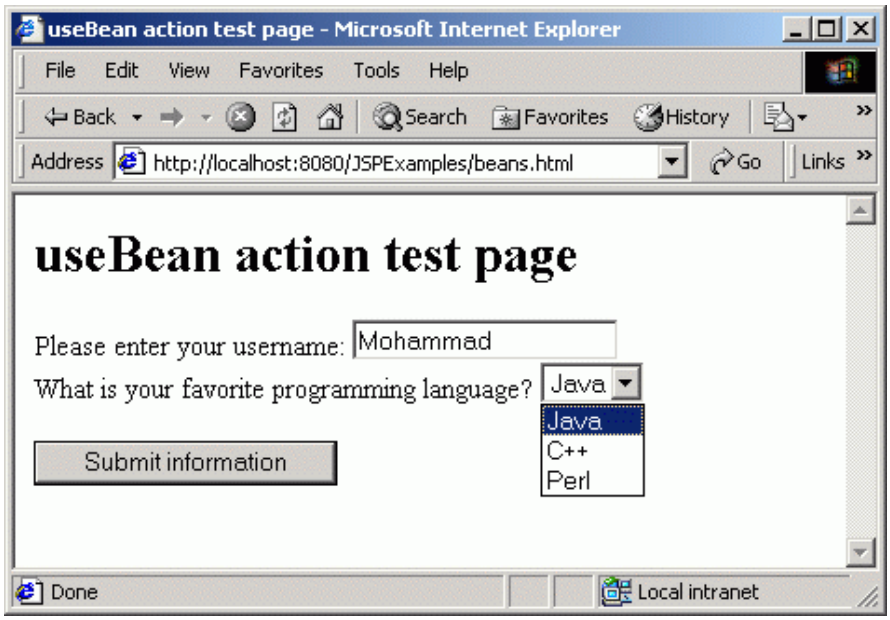
یک برنامه‌نویس خوب، یک کلاس ساده با یک سازنده بدون پارامتر و متدهای set و get برای مقداردهی و بازیابی خاصیت‌های name و language تعریف می‌کند. متد getLanguageComments() با توجه به محتوای خاصیت language توضیحی راجع به آن برمی‌گرداند و به صورت زیر در beans.jsp فراخوانی می شود.

```
<jsp:getProperty name="languageBean" property="language comments"/>
```

فایل‌های beans.jsp و beans.html در دایرکتوری مورد نظر و فایل LanguageBean.java در دایرکتوری C:\JavaPro\Chapter18\JSPExamples\src ذخیره شده و دستور زیر از دایرکتوری src وارد می‌گردد.

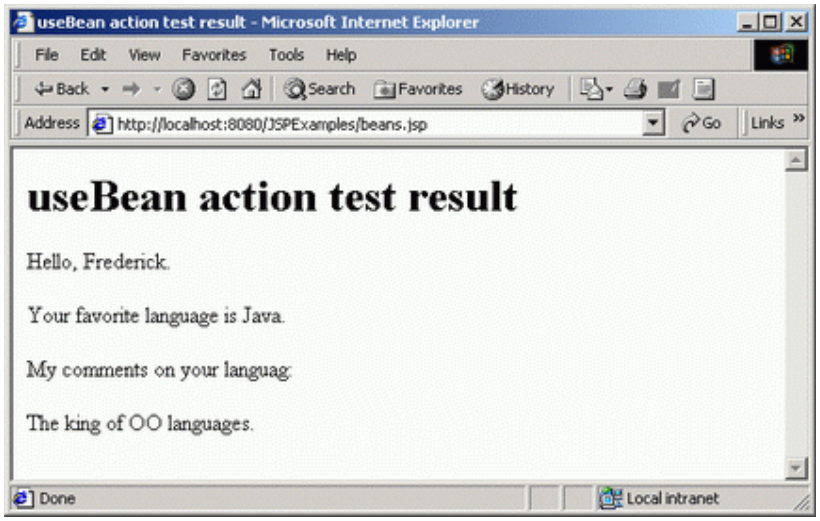
```
javac -d ..\WEB-INF\classes LanguageBean.java
```

سپس آدرس http://localhost:8080/JSPExamples/beans.html در مرورگر وارد شده و خروجی زیر حاصل می شود.



شکل 10-18 خروجی beans.html

با واردکردن نام و زبان برنامه‌نویسی مورد علاقه خود و کلیک کردن روی دکمه submit خروجی زیر ظاهر می شود.



شکل 11-18 خروجی حاصل از اجرای bean

مزیت این روش برنامه‌نویسی، جداسازی کد جاوا از html برای خوانایی برنامه است. انتقال منطق برنامه به LanguageBean باعث سادگی beans.jsp شده، به طوریکه ویرایش آن برای شخصی که مهارت در طراحی صفحات وب داشته ولی با جزییات زبان جاوا آشنایی ندارد، بسیار ساده است.

<jsp:param>

از این action برای تهیه اطلاعات اضافی به صورت زوج‌های نام/مقدار در تگ‌های <jsp:include>، <jsp:forward> و <jsp:plugin> استفاده می شود.

نحوه نگارش آن بدین شکل است.

```
<jsp:param name="paramname" value="paramvalue" />
```

مشخصه‌های قابل دسترس عبارتند از:

- name کلیدی است که برای مشخصه تعریف می شود.
- value مقدار مشخصه است.

<jsp:include>

این action اجازه الحاق منابع ثابت یا پویا مشخص شده توسط URL را به هنگام پردازش درخواست به صفحه جاری JSP می‌دهد. فایل الحاق شده فقط به شیی JspWriter دسترسی دارد و نمی‌تواند cookie یا header را تنظیم کند.

نحوه نگارش این action به یکی از دو شکل زیر است.

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

یـا

```
<jsp:include page="URL" flush="true">
   <jsp:param name="paramname" value="paramvalue" />
   ...
</jsp:include>
```

و مشخصه‌های آن عبارتند از:

- filename مشخص‌کننده فایلی است که الحاق خواهدشد.
- flush در استاندارد jsp1.1 این مقدار همیشه باید "true" باشد و از مقدار "false" پشتیبانی نمی شود. اگر مقدار "true" باشد بافر خروجی قبل از عمل الحاق خالی می شود.

همان طور که گفته شد این action می‌تواند دارای یک یا چند تگ <jsp:param> در داخل بدنه خود باشد. صفحات الحاق شده علاوه بر دسترسی به پارامترهای شیی request صفحه اصلی می‌توانند با استفده از <jsp:param> پارامتر اضافی تعریف کنند. اگر نام پارامترهای جدید و قدیم یکسان باشد، مقدار پارامتر قدیمی دست نخورده می‌ماند ولی مقادیر جدید نسبت به آنچه که وجود داشتند، اولویت دارند. به عنوان مثال اگر شیی request دارای پارامتر param1=myvaluel باشد و پارامتر param1=myvalue2 در تگ <jsp:param> تعریف شود، درخواست دریافت شده در صفحه دوم (فایل الحاق شده) دارای پارامترهای myvalue1 و param1=myvalue2 خواهد بود.

مقادیر پارامترها با استفاده از متد getParameter() رابط javax.Servlet.ServletRequest بازیابی می‌شوند.

درک تفاوت بین دستور include و تگ include بسیار با اهمیت است که بدین شرح می‌باشد.

- دستور include به شکل <%@ include file="filename" %> نوشته شده، به هنگام کامپایل اجرا شده، توسط حامل تجزیه و تحلیل شده و محتویات داخل‌شده توسط آن از نوع ثابت است.
- تگ include به شکل <jsp:include page="filename" /> نوشته شده، به هنگام پردازش درخواست اجرا شده، هرگز مورد تجزیه و تحلیل قرار نمی‌گیرد و محتویات داخل شده توسط آن می‌تواند از نوع ثابت یا پویا باشد.
- درصورت عدم تغییر پی در پی فایل‌ها بهتر است از دستور include استفاده کرد و درصورت تغییر پی درپی آنها بهتر است از تگ include استفاده شود.

در مثال includeAction.jsp زیر از هر دو نوع روش include برای منابع ثابت و پویا استفاده شده‌است.

```
<html>
<head>
   <title>include action test page</title>
</head>
<body>
   <h1>include action test page</h1>
   <h2>using the include directive</h2>
   <%@ include file="included2.html" %>
   <%@ include file="included2.jsp" %>

   <h2>using the include action </h1>
   <jsp:include page="included2.html" flush="true" />
   <jsp:include page="included2.jsp" flush="true" />
</body>
</html>
```

کد 16-18 برنامه includeAction.jsp

فایل الحاق‌شده included2.html دارای خط زیر است.

```
<p>This is some static text in the html file</p>
<p>This is some new text in the html file</p>
```

کد 17-18 محتویات فایل included2.html

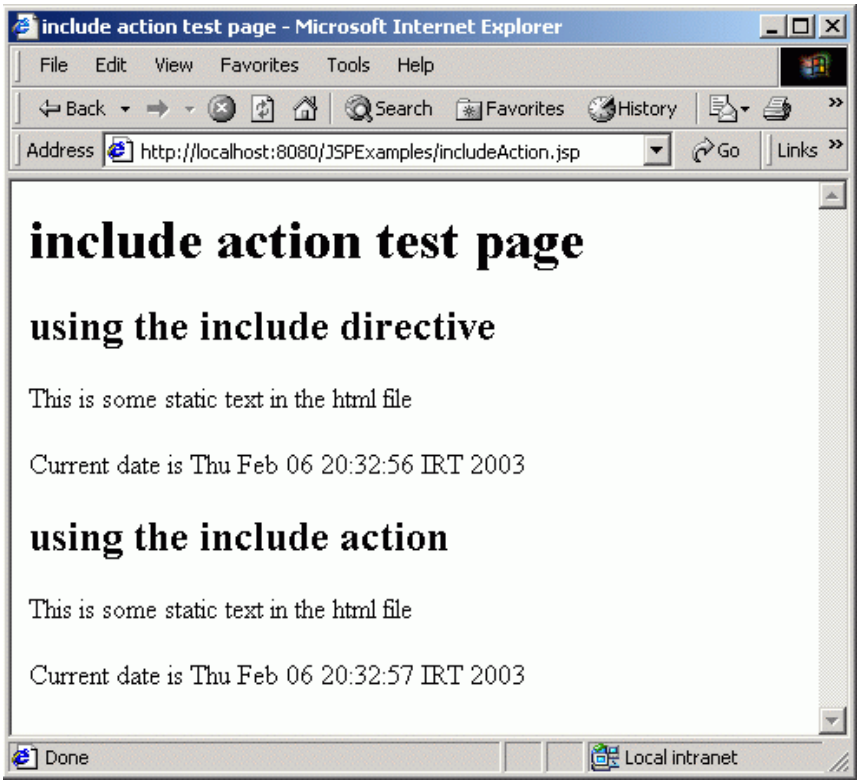
و محتویات فایل included2.jsp چنین است.

```
<%@ page import="java.util.Date" %>
<%= "Current date is " + new Date() %>

<p>This is the new JSP</p>
```

کد 18-18 برنامه included2.jsp

فایل‌ها در دایرکتوری مربوطه ذخیره شده و آدرس زیر در مرورگر وارد می شود.

```
http://localhost:8080/JSPExample/includeAction.jsp
```



شکل 18-10 خروجی includeAction.jsp

اگرچه خروجی این دو نوع include ممکن است یکسان بنظر برسد ولی کمی با یکدیگر تفاوت دارند. جهت درک این تفاوت اندک، بهتر است به کد سرولت تولید شده آن توجه نمود.

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.util.Date;

public class includeAction_jsp {
// ...
   public void _jspService(HttpServletRequest request,
         HttpServletResponse response) throws
            java.io.IOException, ServletException {
      // ...
      out.write("<html>\r\n");
      out.write("<head>\r\n");
      out.write("<title>Include directive test page</title>\r\n");
      out.write("</head>\r\n");
      out.write("<body>\r\n");
      out.write("<h1>Include directive test page </h1>\r\n\r\n");
      out.write("<h2>Using the include directive </h2>\r\n\r\n");
      out.write("<h3>This is some static text in the html file </h3>\r\n\r\n");
      out.print( "Current date is " + new Date() );
      out.write("\r\n\r\n <h2>Using the include action</h2>\r\n\r\n");
      {
         String _jspx_qStr = "";
         out.flush();
         pageContext.include("include2.html" + _jspx_qStr);
       }
      out.write("\r\n");
      {
         String _jspx_qStr = "";
         out.flush();
         pageContext.include("include2.jsp" + _jspx_qStr);
       }
      out.write("\r\n\r\n </body>\r\rn</html>\r\n");
      // …
   }
}
```

کد 19-18 سرولت تولید شده از includeAction.jsp

کد بالا نشان می‌دهد که حامل jsp چطور منابع ذکر شده در دستور include را در صفحه jsp اصلی قرار می‌دهد، درحالی که منابع ذکر شده در در تگ include را به طور پویای فرامی‌خواند. تفاوت این دو روش، زمانی بهتر درک می شود که منابع الحاق شده تغییر کنند (بدون تغییر jsp اصلی).

به فرض محتویات include2.html به صورت زیر تغییر کند.

```
<p>This is some new text in the html file</p>
```

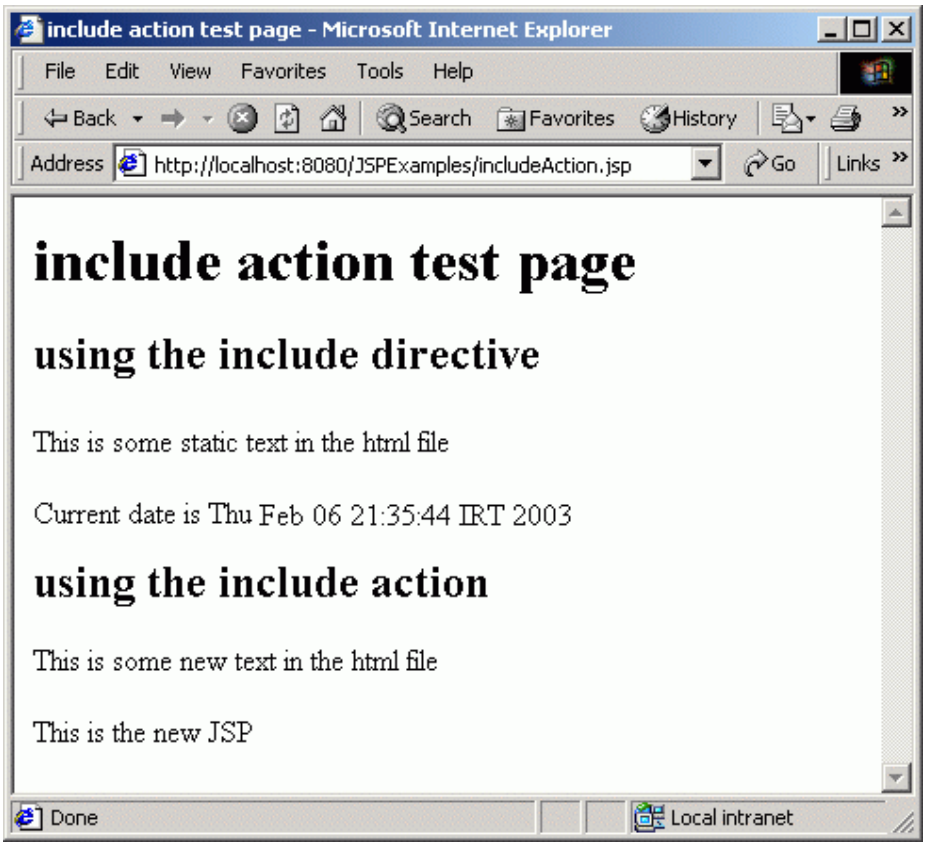
کد 18-20 فایل include2.html

و محتویات include2.jsp به صورت زیر درآید.

```
<p>This is the new JSP</p>
```

کد 18-21 برنامه include2.jsp

حال اگر صفحه JSP دوباره درخواست شود، خروجی بدین شکل خواهدبود.



شکل 12-18 خروجی includeAction.jsp جدید

قسمتی که توسط دستور include الحاق شده، بدون تغییر مانده چراکه صفحه اصلی JSP (فایل includeAction.jsp) تغییر نکرده و بنابراین دوباره کامپایل نشده‌است. ولی قسمت الحاق‌شده توسط تگ <include> تغییر کرده است، چرا که هر زمان صفحه اصلی jsp درخواست شود، این تگ مورد تجزیه و تحلیل قرارمی‌گیرد.

<jsp:forward>

این action اجازه می‌دهد درخواست به یک jsp، سرولت یا یک کد ثابت دیگر ارسال شود. استفاده از آن زمانی مفید است که برنامه‌نویس بخواهد به ازای هر درخواستی که مسدود شده‌است، یک صفحه خاص را نمایش دهد. نحوه نگارش آن به دو صورت زیر می‌باشد.

```
<jsp:forward page="URL" />
```

یا

```
<jsp:forward page="URL">
   <jsp:param name="paramname" value="paramvalue" />
 ...
</jsp:forward>
```

هنگام رسیدن به تگ `<jsp:forward>` اجرای فایل اصلی متوقف شده و بافر پاک می شود. اگر رشته خروجی بافر نشود و چیزی در آن نوشته شود، فراخوانی `<jsp:forward>` باعث بروز استثنای `java.lang.IllegalStateException` خواهد شد. رفتار این `action` دقیقاً مشابه متد `forward()` رابط `javax.Servlet.RequestDispatcher` است.

مثال ساده زیر نحوه استفاده از تگ `<jsp:forward>` را در صفحه `login` نشان می‌دهد. تگ `<form>` این صفحه `HTML` با استفاده از متد `post` اقدام به ارسال درخواست برای `forward.jsp` نموده‌است.

```
<html>
<head>
   <title>forward action test page</title>
</head>
<body>
   <h1>forward action test page</h1>
   <form method="post" action="forward.jsp">
      <p>Please enter your username:
      <input type="text" name="userName">
      <br>and password
      <input type="password" name="password">
      </p>
      <p><input type="submit" value="Log in"></p>
   </form>
</body>
</html>
```

کد 22-18 فایل `forward.html`

`forward.jsp` اولین `jsp` فاقد کد `html` است که تا اینجا دیده شده‌است.

```
<%
if((request.getParameter("userName").equals("Richard"))
   && (request.getParameter("password").equals("xyzzy"))){
%>
   <jsp:forward page="forward2.jsp" />
<% } else { %>
   <%@include file="forward.html" %>
<% } %>
```

کد 23-18 برنامه forward.jsp

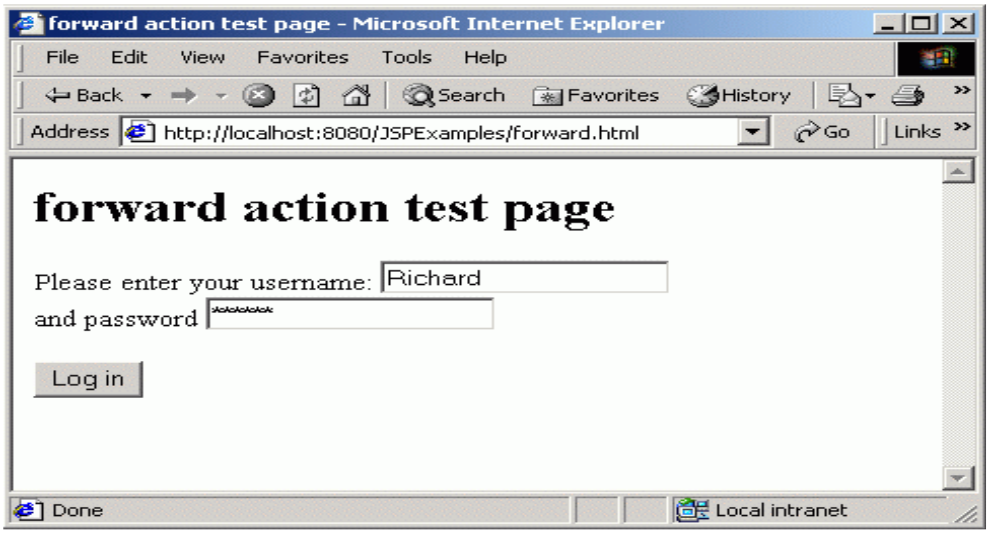
اگر شناسه و رمز عبور کاربر معتبر باشد، درخواست به forward2.jsp ارسال می شود، درغیر اینصورت فرم login دوباره نمایش داده می شود.

سرانجام forward2.jsp یک پیغام خوشامد گویی روی صفحه چاپ می‌کند. از آنجایی که درخواست اصلی شامل پارامترهایی در تگ <form> می‌باشد، لذا آنها برای این JSP ارسال می‌شوند و با استفاده از شیی request می‌توان نام کاربر را نمایش داد.

```
<html>
<head>
   <title>forward action test : login successful!</title>
</head>
<body>
   <h1>forward action test : login successful!</h1>
   <p>welcome, <%=request.getParameter("userName")%></p>
</body>
</html>
```

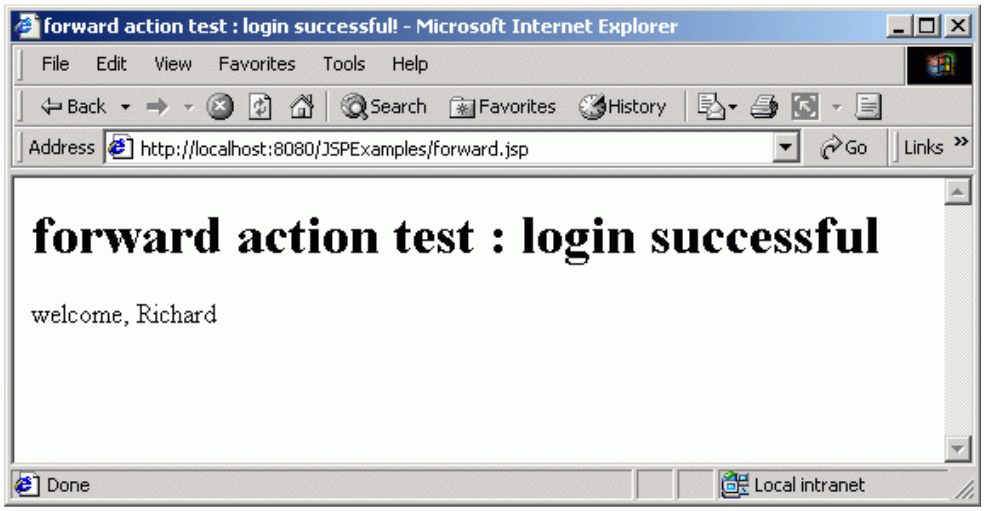
کد 24-18 برنامه forward2.jsp

فایل‌ها در دایرکتوری مورد نظر ذخیره شده و آدرس http://localhost:8080/JSPExamples/forward.html درمرورگر وارد می شود.



شکل 13-18 خروجی forward.html

اگر کاربر Richard رمز عبور خود را صحیح وارد کند، شکل زیر دیده می شود، درغیر اینصورت فرم login دوباره نمایش داده می شود.



شکل 14-18 خروجی forward.jsp درصورت درستبودن رمز عبور

<jsp:plugin>

این action برای تولید تگ‌های خاص html از قبیل <OBJECT> و <EMBED> استفاده می شود که در مرورگر سرویس‌گیرنده اقدام به اجرای plug_in کرده و بعد از جزء Applet یا JavaBean در صفحه html قرار می‌گیرد.

تگ <jsp:plugin> می‌تواند به دلخواه از دو تگ زیر پشتیبانی کند:

- <jsp:params> برای ارسال پارامترهای اضافی به جزء Applet یا javaBean.
- <jsp:fallback> اگر plug_in درمرورگر سرویس‌گیرنده نتواند به هر دلیلی راه‌اندازی شود، این تگ محتویات جایگزین را روی آن مرورگر نمایش می‌دهد. این تگ مشابه تگ <NOFRAME> و مشخصه ALT در HTML است.

نحوه نگارش آن بدین صورت است:

```
<jsp:plugin type="bean|applet" code="objectCode" codebase="objectCodebase"
        align="alignment" archive="archiveList" height="height"
        hspace="hspace" jreversion="jreversion" name="componentName"
        vspace="vspace" width="width" nsplugin="url" ieplugin="url" >
   <jsparams>
      <jsp:param name="paramName" value="paramValue" />
      <jsp:param name="paramName" value="paramValue" />
      ...
   </jsparams>
   <jsp:fallback>Alternate text to display</jsp:fallback>
</jsp:plugin>
```

و مشخصه‌های قابل دسترس آن عبارتند از:

- type نوع جزء جایگزین را معرفی می‌کند، Applet یا Bean و ذکر آن ضروری است.
- code مشابه نحوه نگارش در HTML بوده و ذکر آن ضروری است.
- codebase مشابه نحوه نگارش در HTML بوده و ذکر آن ضرورت ندارد.
- align مشابه نحوه نگارش در HTML بوده و ذکر آن ضرورت ندارد.
- archive مشابه نحوه نگارش در HTML بوده و ذکر آن ضرورت ندارد.
- height مشابه نحوه نگارش در HTML بوده و ذکر آن ضرورت ندارد، ولی بعضی مرورگرها بخاطر مسایل امنیتی صفر را پشتیبانی نمی‌کنند.
- hspace مشابه نحوه نگارش در HTML بوده و ذکر آن ضرورت ندارد.

- jreversion مشخص‌کننده نسخه‌ایی از JER جاوا است که جهت اجرای این جزء موردنیاز می‌باشد و ذکر آن ضرورت ندارد. مقدار پیش‌فرض آن "1.1" است.
- name مشابه نحوه نگارش در HTML بوده و ذکر آن ضرورت ندارد.
- vspace مشابه نحوه نگارش در HTML بوده و ذکر آن ضرورت ندارد.
- title مشابه نحوه نگارش در HTML بوده و ذکر آن ضرورت ندارد.
- width مشابه نحوه نگارش در HTML بوده و ذکر آن ضرورت ندارد، ولی بعضی از مرورگرها بخاطر مسایل امنیتی صفر را پشتیبانی نمی‌کنند.
- nspluginurl URLی است که plug_in جاوا را برای Netscape Nevigator بارگذاری می‌کند. ذکر آن ضرورت ندارد. به طور پیش‌فرض تعریف شده‌است.
- iepluginurl URLی است که plug_in جاوا را برای Internet Explorer بارگذاری می‌کند. ذکر آن ضرورت ندارد. به طور پیش‌فرض تعریف شده‌است.

اشیاء انتزاعی jsp

API سرولت شامل چندین رابط انتزاعی مانند HttpServletRequest، HttpServletResponse، HttpSession و غیره برای برنامه‌نویس است. این رابط‌ها نحوه پیاده‌سازی اشیاء را کپسوله می‌کنند. به عنوان مثال رابط HttpServletRequest داده ارسالی از سرویس‌گیرنده به سرویس‌دهنده، چه از طریق header، چه از طریق پارامترهای <form> و غیره را نشان می‌دهد و متدهایی مانند getParamerer() و getHeader() را برای بازیابی اطلاعات درخواست، درنظر گرفته است.

JSP بر مبنای API سرولت اشیاء انتزاعی را فراهم کرده است. این اشیاء به صورت اتوماتیک در هر صفحه jsp بدون نیاز به نوشتن کد اضافی قابل دسترس بوده و عبارتند از:

- **request** این شیی نشان‌دهنده درخواست سرویس‌گیرنده است که منجر به اجرای متد service() شده‌است. این شیی یک نمونه HttpServletRequest است که دسترسی به Http headers (مثل cookie)، نوع (GET/POST) و پارامترهای درخواست را میسر می سازد. محدوده دسترسی به این شیی، request می‌باشد.
- **response** این شیی نمونه HttpServletResponse و نشان‌دهنده پاسخ سرویس‌دهنده به درخواست سرویس‌گیرنده است. این شیی کدهای وضعیت header متعلق به HTTP را در صفحه JSP به هنگام ارسال خروجی برای سرویس‌گیرنده مقداردهی می‌کند. محدوده دسترسی به این شیی، page است.

- **pageContent** این شیی تنها راه دسترسی به مشخصه‌های صفحه و محلی برای به اشتراک گذاشتن اطلاعات برای یک صفحه JPS می‌باشد. این شیی از نوع javax.Servlet.jsp.PageContext بوده و محدوده دسترسی به آن، page است.
- **session** این شیی نشان‌دهنده session ایجاد شده برای درخواست کاربر است. sessionها به طور اتوماتیک ایجاد شده و همیشه در دسترس می‌باشند (مگر در حالتی که دستور "session="false در بالای صفحه jsp ذکر شده باشد). این شیی از نوع javax.Servlet.http.HttpSession بوده و محدوده دسترسی به آن، session است.
- **application** این شیی نشان‌دهنده ServletContext به دست آمده از شیی ServletConfiguration است که از نوع javax.Servlet.ServletContext بوده و دارای محدوده دسترسی application است.
- **out** شئیی است که رشته خروجی سرویس‌گیرنده در آن نوشته می شود. برای مفید ساختن شیی response، این شیی نسخه بافرشده کلاس java.io.PrintWriter و از نوع javax.Servlet.jsp.JSPWriter است. اندازه بافر را می‌توان توسط مشخصه buffer در دستور page معین نمود.
- **config** این شیی معادل ServletConfig در صفحه JSP جاری است و دارای محدوده دسترسی page می‌باشد.
- page این شیی نمونه‌ای از page پیاده‌سازی شده توسط کلاس Servlet است که توسط درخواست جاری درحال پردازش است. شیی مذکور از نوع java.lang.Object بوده و دارای محدوده تعریف page است. این شیی می‌تواند در یک صفحه به عنوان مترادفی برای عملگر this درنظر گرفته شود.

محدوده تعریف

محدوده تعریف[1] اشیاء JSP یعنی javaBeanها و اشیاء انتزاعی بسیار مهم و حیاتی است و مشخص‌کننده آن است که چه مدت و از کدام JSP آن شیی قابل دسترس است. به عنوان مثال شیی session محدوده‌ای فراتر از یک صفحه دارد، بنابراین می‌توان از آن برای تمام

[1] Scope

درخواست های یک کاربر استفاده کرد. شیی application می‌تواند گروهی از صفحات jsp را سرویس‌دهی کند که با یکدیگر یک برنامه وب را تشکیل می‌دهند.

محدوده‌های jsp در داخل به contextها متکی هستند. context یک حامل مخفی برای منابع و یک رابط برای ارتباط با محیط است. به عنوان مثال سرولت در یک context (یک نمونه از ServletContext) اجرا می شود. هر اطلاعاتی که سرولت راجع به سرویس‌دهنده نیاز داشته باشد از این context قابل استخراج بوده و هر چیزی که در سرویس‌دهنده بخواهد با سرولت ارتباط برقرار کند از طریق context آن وارد می شود.

بنابراین هر چیزی در JSP با یک Context در ارتباط بوده و هر Context نیز محدوده تعریف خاص خود را دارد. آنچه که به هنگام کامپایل یک تگ خاص bean با مقادیر متفاوت scope اتفاق می‌افتد را می‌توان در زیر مشاهده کرد.

محدوده Page

یک شیی با این محدوده تعریف به javax.Servlet.jsp.PageContext محدود خواهد شد. این رابطه خیلی ساده بوده، بدین مفهوم که شیی مذکور در مدتی که صفحه در حال پاسخ‌دهی به درخواست جاری است، در شیی PageContext قرار دارد. یک شیی با این محدوده را می‌توان با متدهای getAtribute() شیی انتزاعی PageContext مورد دسترسی قرار داد.

اشیایی که توسط تگ <jsp:uesBean> مورد استفاده قرار می‌گیرند، دارای محدوده تعریف پیش‌فرض page هستند.

محدوده Request

یک شیی با این محدوده تعریف به javax.Servlet.ServletRequest محدود شده و با متدهای getAttribute() شیی انتزاعی request قابل دسترس می‌باشد. مرجع شیی تا زمانی قابل دسترس است که شیی HttpRequest وجود داشته باشد، یعنی درصورت ارسال درخواست به صفحات دیگر (توسط تگ forward) یا استفاده از تگ <jsp:include> باز هم این شیی قابل دسترس خواهد بود.

محدوده Session

یک شیی با این محدوده تعریف به javax.Servlet.ServletContext محدود شده و با متدهای getAttribute() شیی انتزاعی session قابل دسترس می‌باشد.

محدوده Application

یک شیی با این محدوده تعریف به javax.Servlet.ServletContext محدود شده و با متدهای getAttribute() شیی انتزاعی application قابل دسترس می‌باشد.

تگ‌های معادل XML

استاندارد JSP1.1 برای صفحات JSP یک نگارش مبتنی بر XML تهیه کرده‌است که معادل دستورات و اجزاء Script موجود در JSP می‌باشد. درصورت استفاده از این نگارش، صفحات JSP را می‌توان به عنوان مستندات XML تلقی نمود.

دستورات[1]

برای دستور

```
<%@ directiveName  ATTRIBUTES %>
```

در JSP معادل XML زیر تهیه شده‌است.

```
<jsp:directive:directiveName  ATTRIBUTES />
```

این موضوع برای کلیه دستورات JSP به غیر از taglib صادق است.

اجزاء Script

نحوه تعریف <%! declaration code %> در JSP به صورت زیر در XML معادل‌سازی شده‌است.

```
<jsp:declaration> declaration code </jsp:declaration>
```

[1] Directives

Scriptlet

Scriptlet در JSP به شکل <% scriptlet code %> و معادل آن در XML به شکل زیر می‌باشد.

```
<jsp:scriptlet> scriptlet code </jsp:scriptlet>
```

Expression

Expression در JSP به شکل <%= expression code %> بوده و معادل آن در XML به شکل زیر می‌باشد.

```
<jsp:expression> expression  code </jsp:expression>
```

Action

درحال حاضر نحوه نگارش Actionهای JSP براساس XML است. تنها تفاوت این دو در قرار دادن مقدار مشخصه‌ها در علامت " " است.

اصول طراحی JSP

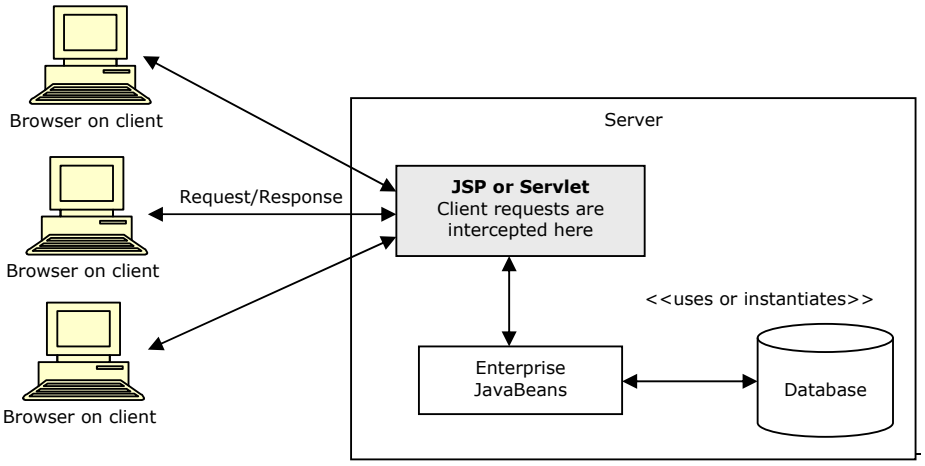
تا اینجا راجع به اینکه صفحات JSP چگونه در حامل وب اجرا می شوند ، تگ های JSP و غیره صبحت شد ، در ادامه فصل روشها و چارچوب های طراحی برنامه های کاربردی استفاده از صفحات JSP مورد بحث قرار می گیرد.

هدف اصلی در طراحی اینگونه برنامه های کاربردی ، جداسازی لایه منطق تجاری[1] از لایه نمایش[2] است. دو خط مشی شاخص برای طراحی برنامه های کاربردی مبتنی بر JSP وجود دارد:

- Client-Server یا Page-Centric: درخواست های سرویس گیرنده مستقیما به صفحه JSP که درخواست ها را پردازش و خروجی (پاسخ) را می سازد، فرستاده می شود.
- Dispatcher یا N-Tier: دراین روش طراحی یک JSP یا سرولت به عنوان میانجی[3] (واسط) یا کنترل کننده عمل می کند و درخواست های سرویس گیرنده را برای صفحات JSP یا JavaBeans های مربوطه ارسال می کند.

معماری Client-Server یا Page-Centric

در این روش طراحی صفحات JSP یا سرولت ها به منابع سیستم (برای مثال بانک اطلاعاتی) یا از طریق یک JavaBeans به طور مستقیم دسترسی دارد و خروجی را نیز خود تولید می کند.

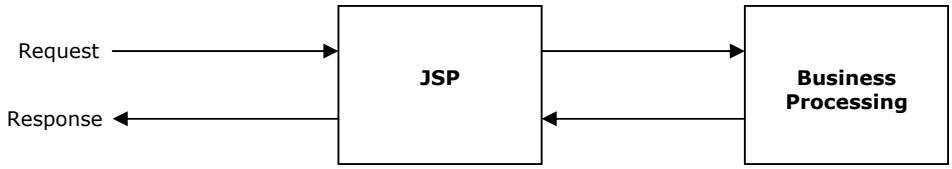


[1] Business Logic
[2] Presentation
[3] Mediator

مزیت این راه حل سادگی برنامه نویسی وتولید آسان صفحات JSP است و نویسنده صفحات وب براحتی می تواند با توجه به نوع درخواست و وضعیت منابع صفحات پویا را تولید نماید.
اما این روش مقیاس پذیر نیست و استفاده بی رویه از قطعه کدهای باعث تولید بیش از حد کد جاوا در بین صفحات JSP می شود . این امر شاید از دیدگاه برنامه نویس جاوا مشکل به نظر نرسد، اما نگهداشت و پشتیانی سیستم را با مشکل مواجه می سازد. دو معماری مختلف در این روش مطرح می شود : Page-View و Page-View with Bean.

معماری Page-View

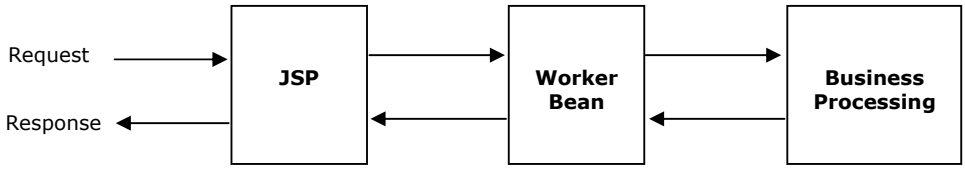
این معماری اولیه بر فراخوانی مستقیم صفحه JSP برای پردازش درخواست و استفاده از صفحات حاوی کدهای جاوا و تگ های HTML برای تولید خروجی برای سرویس گیرنده تاکید می کند.
این راه حل دارای چندین مزیت است ، شروع سریع و آسان تولید از نظر برنامه نویس و پیچیدگی کم. تمام قطعه کدهای جاوادربین تگ های HTML قرار می‌گیرند ، بنابراین تغییرات به قسمتهای خاص از برنامه معطوف می شود. شکل زیر این معماری رانشان می دهد :



مشکل بزرگ این معماری ، توسعه ناپذیری سیستم است. درصورتیکه مقیاس سیستم گسترش یابد محدودیت های این معماری بروز می کند ، قرار گرفتن بیش از حد کد مربوط به منطق تجاری درصفحات JSP تولید ، نگهداشت و تغییر سیستم را با مشکل مواجه خواهد ساخت. این معماری برای سیستم های با مقیاس کوچک توصیه می شود. همان طور که در ادامه خواهید دید استفاده از یک سرولت یاJSP واسط (میانی)وJavaBeans باعث می شود لایه منطق تجاری برنامه از لایه رابط کاربر جدا و به کلاسهایJavaBeans منتقل شود و قابلیت استفاده مجدد از کد افزایش یابد.

معماری Page-View with Bean

این معماری زمانی مورد استفاده قرار می گیرد که کدهای منطق تجاری و نیز کدهای دسترسی به منابع اطلاعاتی بیش از حد در صفحات JSP وارد شود. این معماری به سمت طراحی پیچیده تر گام بر می دارد ، شکل زیر این معماری را نشان می دهد:



در این معماری کدهای مربوط به منطق تجاری و منابع اطلاعاتی‌از صفحات JSP جدا و در کلاس های JavaBeans قرار می گیرند. این تغییر ساختار باعث خوانایی و ساده تر شدن صفحات JSP با استفاده محدود قطعه کدهای جاوا در آن می شود و نقش های طراح صفحات وب و برنامه نویس جاوا کاملا از هم جدا می شود.
این بدین معنی است که یک برنامه نویس می تواند یک JavaBeans ایجاد و بدون نیاز به تغییر تگ های HTML و صفحات JSP ، این کلاس را تغییر یا تعریف مجدد نماید.

طراحی Dispatcher

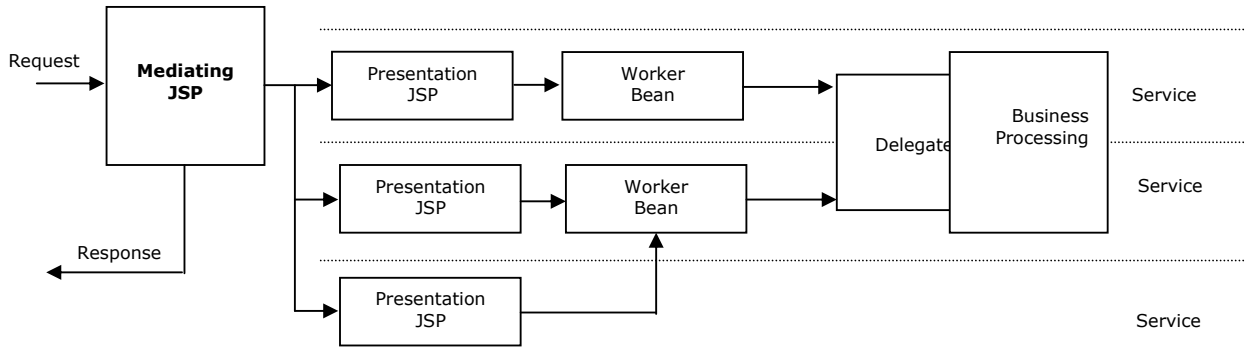
در این طراحی که به راه حل چندلایه نیز معروف است، یک سرولت یا JSP به عنوان یک کنترل کننده یا میانجی عمل می کند و درخواست های سرویس گیرنده را برای صفحات JSP یا کلاسهای JavaBeans می فرستد.این راه حل شامل سه معماری Mediator-View ، Mediator-Composite View و Service to Worker است.

معماری Mediator-View

ساختن سرویس های مشترک مانند شناسایی کاربر ، بیرون از سرولت یا JSP واسط ، باعث حذف کدهای تکراری از صفحات JSP می شود. برای مثال می توان یک سرولت برای کنترل معتبر بودن کاربر در نظر گرفت و به عنوان تنها نقطه ورودی سیستم درنظرگرفت.

در این الگو از سرولت یا JSP واسط با لایه نمایش کار می کند و کلاسهای JavaBeans به سرویس دهی درخواست های سرویس گیرنده ها می پردازند. شکل زیر نشان می دهد که چگونه

در این معماری سرویس ها توزیع شده اند. سرولت یاJSP واسط کنترل درخواست را شروع می کند و سپس آن را به کلاسهایJavaBeans محول می کند. Presentation JSP در شکل زیر مشخصه های شیء JavaBeans را با استفاده از اطلاعات وارده شده توسط کاربر مقدار دهی می کند و سپس از این شیء برای آماده سازی نمایش اطلاعات کمک می گیرد:



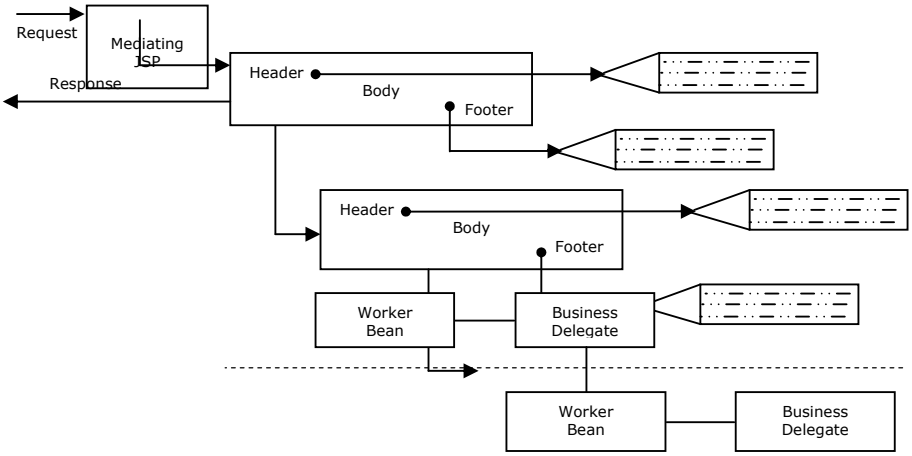
معماری Mediator-Composite View

گاهی اوقات سیستم هایی با الگوهای متن پویا[1] ساخته می شوند ، به عبارت دیگر محتویات صفحات به صورت پویا تولید می شوند و الگوهای متن که اطلاعات را دربرگرفته است نیز تغییر می کند. درنظر بگیرید header و footer در یک صفحه JSP که اطلاعاتی مانند وضعیت تغییرات را نمایش می دهند. منطقی نیست که برای هر تغییر این الگو ها ، صفحات JSP دوباره کامپایل شوند چراکه این کار کندی اجرا و بارگذاری منابع را در پی‌دارد. هنگامی که header و footer مستقیما در صفحه JSP قرار بگیرند ، به ازای هر تغییر این اطلاعات ، صفحه JSP نیز تغییر کندو نیاز به کامپایل مجدد داشته باشد.

نه تنها فقط صفحات ثابت مانند صفحات HTML قابل درج در صفحات JSP هستند ، بلکه می توان صفحات پویایJSP را نیز در صفحات دیگر قرار داد. در این حالت لایه نمایش از تعدادی صفحات ثابت و پویا تشکیل می شود که در کنار هم صفحه قابل نمایش برای کاربر را می سازند. هر یک از این صفحات خود می تواند شامل چندین صفحه داخلی باشد. اگرچه الگوهای HTML به عنوان منابع ثابت در نظر گرفته می شوند ، بدین معنی که در زمان اجرا به صورت پویا تولید می شوند . اما این قطعه کدها به صورت بخشی پویا از رابط

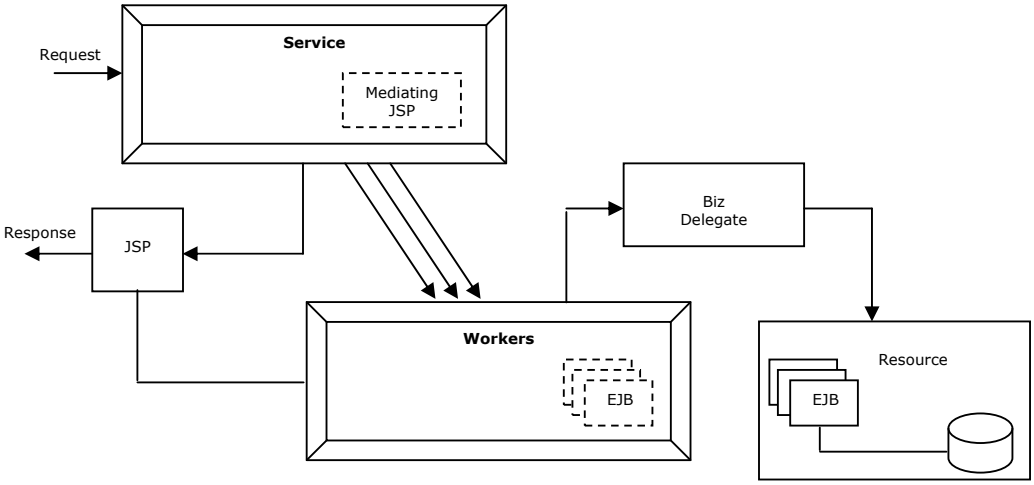
[1] Dynamic Template Text

کاربرمحسوب می شوند. چرا که هریک از این الگوها می توانند به صورت متناوب با به روزرسانی های فوری ، تغییر نمایند. شکل زیر این معماری را نشان می دهد:



معماری Service to Worker

شمای کلی این معماری در شکل زیر نمایش داده شده است. یک کلاس JavaBeans وظیفه پردازش منطق تجاری سیستم و دسترسی به اطلاعات را برعهده دارد. مشابه هر معماری توزیع کننده ، یک سرولت یا JSP واسط وظیفه کنترل درخواست های سرویس گیرنده را برعهده دارد و امکان پردازش سرویس های مشترک را فراهم می آورد. پس از اینکه کلاس JavaBeans پردازش منطق را به اتمام برساند ، جزء میانی این معماری ، صفحات JSP مناسب را برای تولید لایه نمایش فراخوانی‌می کند :



در این معماری تفکیک لایه های کنترل کننده و نمایش بیشتر نمود پیدا می کند ، چرا که صفحات JSP کدهای منطق تجاری را مستقیما فراخوانی نمی کنند بلکه از طریق JavaBeans های از پیش تعریف شده این کار صورت می گیرد. بسته به چرخه کار[1] مورد نیاز سیستم می توان از کنترل کننده ها برای تصمیم گیری در وضعیت های مختلف استفاده کرد.

[1] Workflow

الگوی Model-View-Controller

الگوی MVC امکان جداسازی لایه منابع اطلاعاتی را از لایه های دسترسی و پردازش اطلاعات فراهم می سازد . یک سیستم مبتنی بر MVC به سه قسمت مدل (Model) ، دید (View) و کنترل کننده (Controller) تقسیم می شود . View، رابط کاربر است که می تواند توسط صفحات JSP پیاده سازی شود. کنترل کننده توسط یک یا چند سرولت پیاده سازی می شود ، مدل نیز می تواند [1] EJB یا سایر عناصر دسترسی به بانک اطلاعاتی باشد .

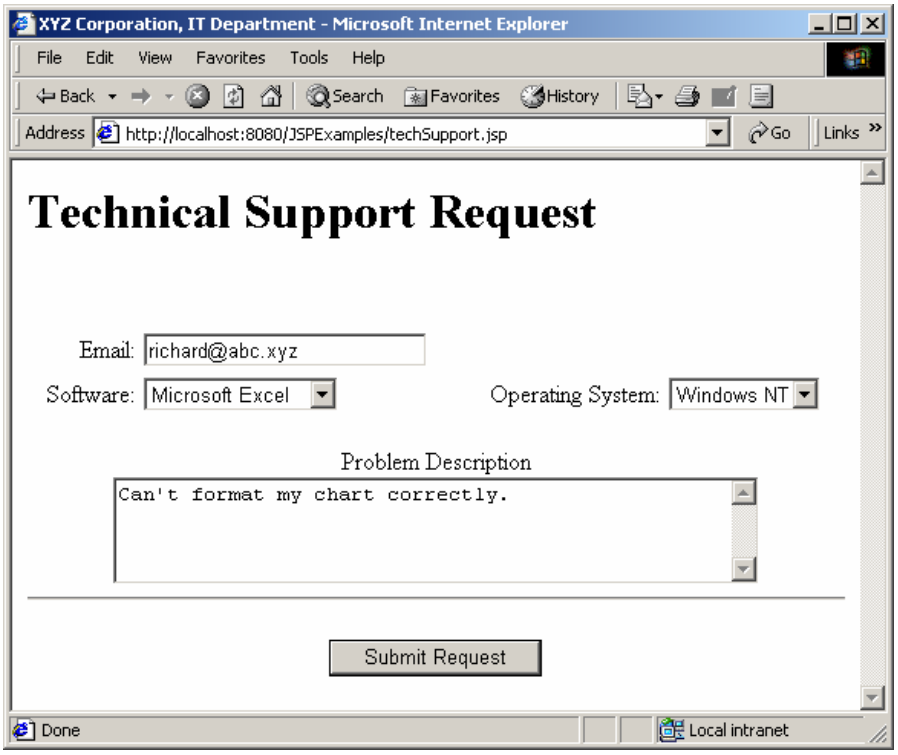
MVC برای سیستمهای بزرگ و توزیع شده که اطلاعات از طریق چندین روش دسترسی پردازش می شوند میتواند مفید واقع شود. الگوی MVC یک گزینه مناسب برای تولید برنامه ها به صورت همزمان و پیمانه ای[2] توسط برنامه نویسان مختلف است .
ویژگی شاخص MVC جداسازی وظایف و مسئولیت ها است . برای مثال وظیفه View تنها نمایش اطلاعات است و جوابگو یا مسئول بروزرسانی بانک اطلاعاتی نیست . کنترل کننده مسئول انتخاب یک View مناسب و تغییرات Model است . گاهی اوقات Model شامل منطق تجاری و توابع بروزرسانی اطلاعات نیز می شود.
سرویس گیرنده درخواست را به سرولت کنترل کننده می فرستد و کنترل کننده با Model متناظر ، برای انجام عمل درخواست شده توسط سرویس گیرنده ، ارتباط برقرار می کند و بسته به نیاز سرویس گیرنده ، View مناسب را نمایش می دهد .

---

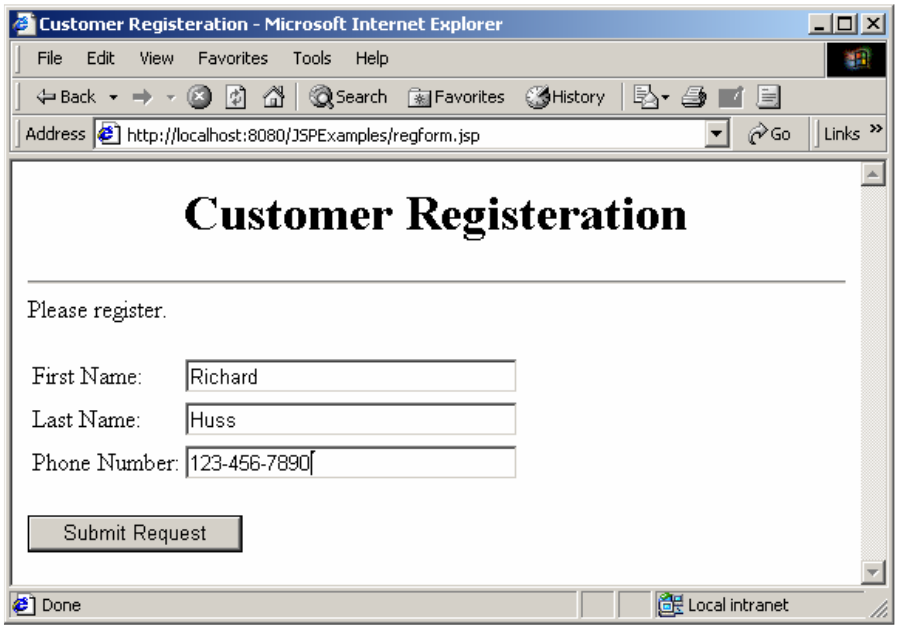
[1] Enterprise Java Bean
[2] Modular

یک مثال کاربردی

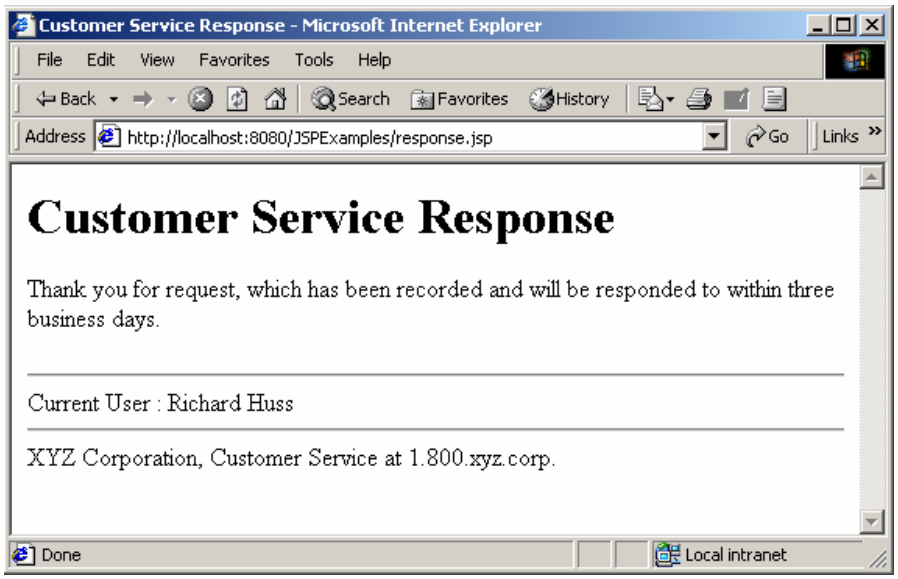
در انتهای این فصل مثال Technical Support فصل چهاردهم سرولت با راه حل مبتنی بر JSP طراحی و پیاده سازی می شود. منطق تجاری تفاوتی با مثال فصل چهاردهم ندارد. کاربر به صفحه اصلی سایت مراجعه می کند و فرم اطلاعات Request Technical Support شامل آدرس الکترونیکی و جزئیات مشکل خود را وارد می کند:



با ارسال فرم از طرف کاربر ، سیستم بررسی می کند که آیا کاربر قبلا در سیستم تعریف شده است(با توجه به آدرس الکترونیکی) ، اگر کاربر جدید باشد فرم ثبت نام کاربر نمایش داده می شود:



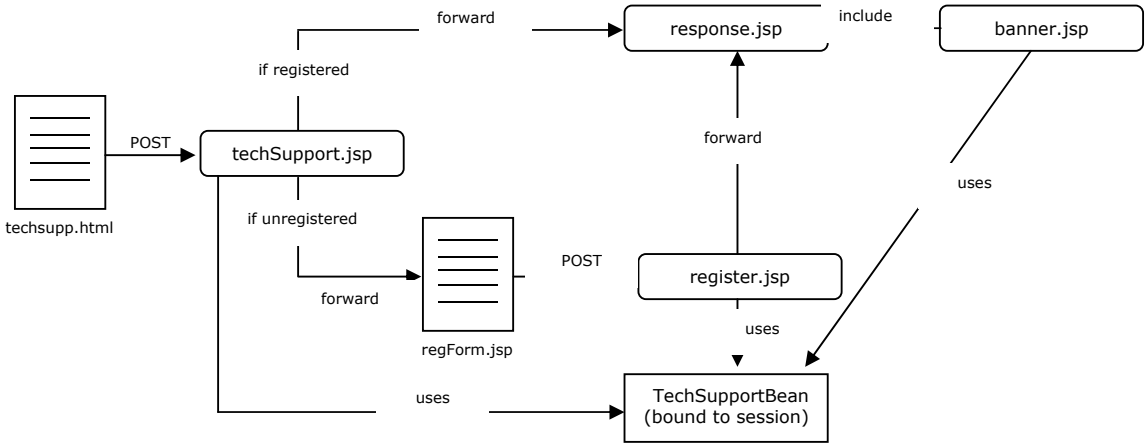
پس از تکمیل فرم توسط کاربر و ارسال آن ، صفحه تایید دریافت درخواست کاربر نمایش داده می شود:



(اگر آدرس الکترونیکی کاربر در سیستم تعریف شده باشد ، بجای فرم ثبت نام کاربر این صفحه نمایش داده می شود.)

طراحی برنامه

ساختار برنامه همان طور که گفته شد مشابه مثال فصل چهاردهم است با این تفاوت که تمام سرولت ها با صفحات JSP جایگزین می شود و منطق برنامه به یک JavaBeans به نام TechSupportBean منتقل می شود:



مطابق نمودار بالا سرولت TechSupportServlet با techSupport.jsp ، register.html با regform.jsp ، RegisterCustomerServlet با register.jsp ، ResponseServlet با response.jsp و BannerServlet با banner.jsp جایگزین می شود.

کلاس TechSupportBean نقش "مدل" منطق برنامه را ایفا می کند. و در طی فرآیند پردازش درخواست کاربر مورد استفاده قرار می گیرد : یک شئیی از این کلاس در صفحه techSupport.jsp ایجاد و حوزه اعتبار آن در محدوده session تعریف می شود و در سایر صفحات JSP به کار گرفت می شود. این کلاس کدهای دسترسی به بانک اطلاعاتی را کپسوله می نماید و از قرار گرفتن این کدها در بدنه اصلی صفحات JSP جلوگیری می کند.

صفحه اول

صفحه `techsupp.html` فرم ورود به سیستم است و نسبت به مثال فصل چهاردهم تغییر محسوسی نکرده است ، به جز اینکه از متد `POST` برای فرستادن اطلاعات به صفحه `techSupport.jsp` استفاده می شود:

```
<html>
   <head>
      <title>XYZ Corporation, IT Department</title>
   </head>
   <body>
      <h1>Technical Support Request</h1>
      <hr><br>
      <center>
         <form action="/JSPTechSupport/techSupport.jsp" method="POST">
            <table align="center" width="100%" cellspacing="2" cellpadding="2">
            <tr><td align="right">Email:</td>
               <td><input type="text" name="email" align="left" size="25"></td>
            </tr>
            <tr><td align="right">Software:</td>
               <td><select name="software" size="1">
                     <option value="Word">Microsoft Word</option>
                     <option value="Excel">Microsoft Excel</option>
                     <option value="Access">Microsoft Access</option>
                  </select></td>
               <td align="right">Operating System:</td>
               <td><select name="os" size="1">
                     <option value="95">Windows 95</option>
                     <option value="98">Windows 98</option>
                     <option value="NT">Windows NT</option>
                  </select></td></tr>
            </table>
         <br>Problem Description<br>
            <textarea name="problem" cols="50" rows="4"></textarea>
            <hr><br>
            <input type="submit" name="submit" value="Submit Request">
         </form>
      </center>
   </body>
</html>
```

این صفحه در فایل پیکربندی web.xml به عنوان صفحه اصلی برنامه تعریف شده است :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
   "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
   <welcome-file-list>
      <welcome-file>techsupp.html</welcome-file>
   </welcome-file-list>
</web-app>
```

صفحه پردازش درخواست

هنگامیکه کاربر یک درخواست پشتیبانی فنی را ارسال می کند ، اطلاعات آن به صفحه techSupport.jsp فرستاده می شود. این صفحه حاوی کدهای جزئیات نمایش رابط کاربر نیست ، بلکه یک شیء از کلاس TechSupportBean با محدوده session ایجاد می شودو با استفاده از تگ <jsp:setProperty> مشخصه های email ، os ، software و problem با داده های فرستاده شده توسط کاربر مقدار دهی می شوند :

```
<%@ page errorPage="/error.jsp" %>
<jsp:useBean id="techSupportBean" scope="session" class="TechSupportBean"/>
<jsp:setProperty name="techSupportBean" property="*"/>
```

سپس با فراخوانی متد registerSupportRequest() اطلاعات ارسال شده در جدول SUPP_REQUESTS ذخیره می شود و نیز جدول CUSTOMERS بررس می شود که آیا آدرس الکترونیکی کاربر قبلا ثبت شده است یا خیر :

```
<% techSupportBean.registerSupportRequest();%>
```

در انتها ، متد isRegistered() برای اطمینان از ثبت درخواست کاربر فراخوانی می شود ، اگر عمل ثبت با موفقیت انجام شده باشد، صفحه response.jsp ،در غیراین صورت صفحه regform.jsp نمایش داده می شود :

```
<% if(techSupportBean.isRegistered()) { %>
   <jsp:forward page="response.jsp" />
<% } else { %>
   <jsp:forward page="regform.jsp" />
<% } %>
```

کلاس TechSupportBean

چگونه کلاس TechSupportBean پردازش های مربوط به techSupport.jsp را انجام می دهد ؟ این کلاس در ابتدا شامل تعریف متغیرها ، متد getter و setter است :

```
import java.sql.*;

public class TechSupportBean {
   private String email;
   private String software;
   private String os;
   private String problem;
   private String firstName;
   private String lastName;
   private String phoneNumber;

   private String driver = "COM.cloudscape.core.JDBCDriver";
   private String protocol =
                  "jdbc:cloudscape:c:/ProJava/chapter18/JSPTechSupport/WEB-
                       INF/db;autocommit=true";

   public TechSupportBean() {
      try {
         Class.forName(driver);
      } catch(ClassNotFoundException e){
      }
   }

   public void setEmail(String email) {
      this.email = email;
   }

   // ... and similary for the software, os problem, firstname, lastname,
   // and phoneNumber properties

   public String getEmail() {
      return email;
   }

   // ... and similary for the software, os problem, firstname, lastname,
   // and phoneNumber properties

   // ... more properties and methods...
}
```

در سازنده این کلاس داریور JDBC با مشخص کردن نام داریور و آدرس بانک اطلاعاتی بارگذاری شده است . در یک روش برنامه نویسی پیشرفته مشخصات بانک اطلاعاتی در یک فایل xml یا property تعریف و برنامه با خواندن و پردازش این فایل ها ، داریور بانک اطلاعاتی بارگذاری می شود.

متدهای getter وsetter برای مقداردهی و واکشی مشخصه های عمومی کلاس شامل firstName ، lastName و phoneNumber بایستی نوشته شوند. این مشخصه ها توسط فیلدهای بانک اطلاعاتی یا اطلاعات ارسال شده توسط کاربر مقداردهی می شوند.

در واقع چون شیء techSupportBean در محدوده session تعریف شده است، این مشخصه های در تمام صفحات JSP قابل دسترس می باشد.

کار اصلی برنامه در متد registerSupport () انجام می شود. از کلاس Sequencer بدون هیچ تغییر نسبت به فصل چهاردهم استفاده شده است. (این کلاس یک شماره درخواست یکتا از جدول SEQ_NO تولید می کند.) با این تفاوت که متد registerSupportRequest () درخواست کاربر را خودش ارسال می کند ، بلکه تنها فقط مقدار مشخصه registered که از نوع boolean است را تعیین می کند . اگر مقدار این مشخصه برابر true باشد یعنی اینکه کاربر قبلا ثبت نام شده است و مشخصه های آن شامل firstName ، lastName و phoneNumber مقدار دهی می شود.

```
private boolean registered;

   public void registerSupportRequest() throws SQLException {
      int requestId = 0;
      Connection connection = null;
      String insertSql =
                        "INSERT INTO SUPP_REQUESTS VALUES(?, ?, ?, ?, ?)";
      String selectSql =
                  "SELECT CUSTOMERS.FNAME, CUSTOMERS.LNAME,
                        CUSTOMERS.PHONE FROM CUSTOMERS"
                     + " WHERE CUSTOMERS.EMAIL = ?";
       try {
         connection = DriverManager.getConnection(protocol);
          PreparedStatement insertStatement =
                              connection.prepareStatement(insertSql);
         requestId = Sequencer.getNextNumber(protocol);

         insertStatement.setInt(1, requestId);
         insertStatement.setString(2, email);
         insertStatement.setString(3, software);
         insertStatement.setString(4, os);
         insertStatement.setString(5, problem);

         insertStatement.executeUpdate();

         // Now verify if the customer is registered or not.
         PreparedStatement selectStatement =
            connection.prepareStatement(selectSql);
         selectStatement.setString(1, email);
```

```
            ResultSet rs = selectStatement.executeQuery();
            if(rs.next()){
               setFirstName(rs.getString("FNAME"));
               setLastName(rs.getString("LNAME"));
               setPhoneNumber(rs.getString("PHONE"));

               // The customer was registered - we can go straight to the
               // response page
               registered = true;
            } else {
               // Customer is not registered - need to go to registration form
               registered = false;
            }
         } finally {
            if(connection != null){
               try {
                  connection.close();
               } catch(SQLException sqle){}
            }
         }
```

فرم ثبت نام

اگر کاربر قبلا در سیستم ثبت نشده باشد ، درخواست به `regform.jsp` ارسال می شود و فرم ثبت اطلاعات کاربر نمایش داده می شود. در این صفحه یک فرم با متد ارسال`POST` تعریف شده است :

```
<html>
   <head>
      <title>Customer Registeration</title>
   </head>
   <body>
      <center><h1>Customer Registeration</h1></center>
      <hr>
       Please register.
         <form action="/JSPTechSupport/register.jsp" method="POST">
            <table  border ="0" align="center">
            <tr><td>First Name:</td>
               <td><input type="text" name="firstName" size="30"></td></tr>
            <tr><td>Last Name:</td>
               <td><input type="text" name="lastName" size="30"></td></tr>
            <tr><td>Phone Number:</td>
               <td><input type="text" name="phoneNumber" size="30"></td></tr>
            <tr><td colspan="2">
              <br><input type="submit" name="submit" value="Submit Request">
            </tr>
         </form>
      </center>
   </body>
</html>
```

با پرکردن اطلاعات توسط کاربر ، صفحه register.jsp فراخوانی می شود:

```
<%@ page errorPage="/error.jsp" %>

<jsp:useBean id="techSupportBean" scope="session" class="TechSupportBean" />
<jsp:setProperty name="techSupportBean" property="*"/>

<% techSupportBean.registerCustomer();%>

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

این صفحه بسیار مشابه techSupport.jsp است ، مشخصه های شیء
techSupportBean با تگ <jsp:setProperty> مقدار دهی می شوند.
و سپس با فراخوانی متد registerCustomer() اطلاعات کاربر در بانک اطلاعاتی ذخیره می شود.

```
public void registerCustomer() throws SQLException {

      int requestId = 0;
      Connection connection = null;
      String insertSql =
               "INSERT INTO CUSTOMERS VALUES(?, ?, ?, ?)";
      try {
         connection = DriverManager.getConnection(protocol);

         PreparedStatement insertStatement =
              connection.prepareStatement(insertSql);
         insertStatement.setString(1, email);
         insertStatement.setString(2, firstName);
         insertStatement.setString(3, lastName);
         insertStatement.setString(4, phoneNumber);

         insertStatement.executeUpdate();

      } finally {
         if(connection != null){
            try {
               connection.close();
            } catch(SQLException sqle){}
         }
      }
   }
```

و در نهایت درخواست کاربر به صفحه response.jsp ارسال می شود.

صفحات response **و** banner

در هر دو صورت ثبت نام بودن یا نبودن کاربر در سیستم ، درخواست کاربر به صفحه response.jsp ارسال می شود. این صفحه یک پیغام ساده مبنی بر دریافت درخواست کاربر و نیز همچنین با استفاده از دستورinclude مشخصات کاربر را چاپ می کند:

```
<%@ page errorPage="/error.jps" %>
<html>
   <head>
      <title>Customer Service Response</title>
   </head>
   <body>
      <h1>Customer Service Request Received</h1>

      <p>Thank you for your request, which has been recorded and will be
      responsded to within three business days.
      </p>
      <%@ include file="/banner.jsp"  %>
   </body>
</html>
```

صفحه banner.jsp با رجوع به شیء techSupportBean نام و نام خانوادگی کاربررا چاپ می کند :

```
<jsp:useBean id="techSupportBean" scope="session" class="TechSupportBean"/>

<hr>
Current User:
<jsp:getProperty name="techSupportBean" property="firstName" />
<jsp:getProperty name="techSupportBean" property="lastName" />
<hr>
XYZ Corporation, Customer Service at 1.800.xyz.corp.<br>
```

صفحه کنترل خطا

برای کنترل خطا و نمایش پیغام خطا از صفحه error.jsp استفاده شده است ، error.jsp در صفحات دیگر برنامه با استفاده از مشخصه errorPage در تگ <%@page> به عنوان صفحه کنترل خطا تعریف شده است.

```
<%@page isErrorPage="true"  %>
<html>
   <head>
      <title>XYZ Corporation, IT Department</title>
   </head>
   <body>
      <h1>Technical Support </h1>
      <p>We're sorry, an error occurred processing your request.</p>
      <p>You got a <%=exception%></p>
   </body>
</html>
```

راه اندازی سیستم

در پایان کار ، تنها کار باقیمانده حصول اطمینان از اینکه فایلهای برنامه در محل صحیح خود قرار گرفته باشند :

- صفحات JSP در دایرکتوری C:\JavaPro\Chapter18\JSPTechSupport
- فایل web.xml در دایرکتوری C:\JavaPro\Chapter18\JSPTechSupport\WEB-INF
- فایل های TechSupport.java و Sequencer.java در دایرکتوری C:\JavaPro\Chapter18\JSPTechSupport\src
- فایل های جاوا با استفاده از دستور خطی زیر در دایرکتوری دایرکتوریC:\JavaPro\Chapter18\JSPTechSupport\classes

```
java – d ..\WEB-INF\classes *.java
```

- راه اندازی بانک اطلاعاتی ، در صورتی که CloudScape ، Microsoft Access و یا MySQL به عنوان بانک اطلاعاتی انتخاب شده باشد ، بانک اطلاعاتی در دایرکتوری

C:\JavaPro\Chapter18\JSPTechSupport\db ایجاد شود و جداول مورد نیاز طبق دستورات زیر ایجاد شود :

```
CREATE TABLE SUPP_REQUESTS(REQUEST_ID INTEGER PRIMARY KEY,
                    EMAIL VARCHAR(40),
                    SOFTWARE VARCHAR(40),
                    OS VARCHAR(40),
                    PROBLEM VARCHAR(256));

CREATE TABLE SEQ_NO(NEXT_NO INTEGER);

INSERT INTO SEQ_NO VALUES(0);

CREATE TABLE CUSTOMERS(EMAIL VARCHAR(40) PRIMARY KEY,
                  FNAME VARCHAR(15),
                  LNAME VARCHAR(15),
                  PHONE VARCHAR(12));
```

- در فایل پیکربندی server.xml مسیر برنامه معرفی‌گردد :

```
<Context path="/JSPTechSupport"
   docBase="c:\ProJava/Chapter18/JSPTechSupport">
</Context>
```

- پس از راه اندازی و اجرای سرویس دهنده Tomcat آدرس زیر در مروگر وب وارد شود:

```
http://localhost:8080/JSPTechSupport
```

خلاصه فصل

در این فصل درباره گرامر JSP ، سازنده ها ، تگ ها و روشهای طراحی سیستم های چند لایه با استفاده از JSP و در انتها طراحی و پیاده سازی یک برنامه ساده صحبت شد. با آشنایی اولیه با HTML حال می توانید برنامه های کاربردی مبتنی بر JSP را تولید نمائید.

پرسش‌های فصل

1- صفحه JSP بعد از کامپایل شدن به چه چیزی تبدیل می شود.

الف- اپلت ب- سرولت

ج- برنامه‌کاربردی د- هیچکدام

2- کدامیک از موارد زیر، متد استانداردی نیست که به عنوان بخشی از چرخه حیات JSP فراخوانی شود.

الف- jspInit() ب- jspService()

ج- _jspService() د- jspDestroy()

3- اگر قرار باشد متد مقداردهی اولیه JSP تحریف شود، آن متد باید در بین کدامیک از تگ‌های زیر تعریف شود.

الف- <@ @> ب- <%@ %>

ج- <% %> د- <%! %>

4- به هنگام استفاده از JavaBean در JSP کدامیک از موارد زیر نمی‌تواند به عنوان یک محدوده تعریف استفاده شود.

الف- application ب- session

ج- request د- request

5- اشیاء ضمنی JSP همانند request، response و out فقط در متد _jspService() قابل رویت می‌باشند.

الف- درست ب- نادرست

6- تفاوت اصلی استفاده از <jsp:forward> و HttpServletResponse.sendRedirect() در چیست؟

الف- forward روی سرویس‌گیرنده اجرا شده، درحالی که متد sendRedirect() روی سرویس‌دهنده اجرا می شود.

ب- forward روی سرویس‌دهنده اجراشده، درحالی که متد sendRedirect() روی سرویس‌گیرنده اجرا می شود.

ج- هر دو به طور مجزا اجرا می‌شوند.

7- کدامیک از جملات زیر، صفحه JSP کامپایل‌شده را به عنوان رابط singleThreadModel پیاده‌سازی می‌کند.

الف- <%@ page isThreadSafe = "false" %>

ب- <%@ page isThreadSafe = "true" %>

8- درصورت استفاده از کدامیک از موارد توضیحات نویسی معتبر در صفحات JSP، کاربـر عادی می‌تواند آنها را ببیند.

الف-

```
<%--
My comments
<% out.println("Hello World"); %>
--%>
```

ب-

```
<!-- (c) 2000 jGuru.com -->
```

ج-

```
<% // For Loop
for (int i=1; i<4; i++) {
%>
   <H<%=i%>>Hello</H<%=i%>>
<%}%>
```

د-

```
<% /** Yet another comment */
   JavaDoc Rules
 %>
```

9- چطور یک سرولت می‌تواند صفحه خالی JSP را فراخوانی کند.

الف- از این قابلیت حمایت نمی شود.

ب- زمانی که سـرولت بروز یک استثناء را درک کند، به طور اتوماتیک توسط فراخوانی صفحه JSP آن را کنترل می‌کند.

ج- سرولت نیازمند آن است که RUL صفحه خطای مشخصی را درخواست کند. در این شرایط استثناء رخ داده نیز به عنوان یک خصیصه به نام javax.servlet.jsp.jspException ارسال می شود.

د- سـرولت نیازمند آن اسـت که پاسـخ را به یـک صفحه خطـای مشـخص redirect کند. در این شرایط استثناء به روزشده نیز در یک Cookie ذخیره می شود.

10- به هنـگام استفاده از یـک JavaBean برای دریافت همه پارامترهای یـک فرم، چه property در کـد زیر باید برای مقداردهی اتوماتیک همه پارامترها مورد استفاده قرارگیرد.

```
<jsp:useBean id="fBean" class="govi.FormBean" scope="request" />
<jsp:setProperty name="fBean" property="???" />
<jsp:forward page="/servlet/JSP2Servlet" />
```

الف- * ب- all ج- @ د- =