How can I avoid Java code in JSP files, using JSP 2?

Asked 14 years, 5 months ago Modified 1 year, 11 months ago Viewed 322k times

1770

I know that something like the following three lines

```
<%= x+1 %>
<%= request.getParameter("name") %>
<%! counter++; %>
```

is an old school way of coding and in JSP version 2 there exists a method to avoid Java code in JSP files. What are the alternative JSP 2 lines, and what is this technique called?

java jsp scriptlet

Share
Improve this question
Follow

edited Dec 28, 2022 at 23:44

starball
48.3k • 28 • 187 • 848

asked Jul 5, 2010 at 7:24

chmoelders
18.7k • 6 • 23 • 25

22 @Koray Tugay, as long as the counter variable is declared somewhere prior to its use, then it's most certainly valid... – Sheldon R. Feb 26, 2015 at 14:30

@SheldonR. This is valid: <%= counter++ %> or this: <%! int counter = 0; int x = counter++; %> but not: <%! int counter = 0; counter++; %> – Koray Tugay Oct 22, 2017 at 17:36

@KorayTugay, what I meant was if the variable counter was declared in an earlier script block, it should be valid in a later block. But ultimately, J2EE programmers these days should be using EL variables instead of scriptlets, anyway... – Sheldon R. Mar 29, 2018 at 15:43

31 Answers

Sorted by: Highest score (default)

1 2 Next

2030

The use of *scriptlets* (those `<% %>` things) in JSP is indeed highly discouraged since the birth of *taglibs* (like JSTL) and EL (Expression Language, those `${}` things) way back in 2001.

The major disadvantages of *scriptlets* are:

1. **Reusability:** you can't reuse scriptlets.
2. **Replaceability:** you can't make scriptlets abstract.
3. **OO-ability:** you can't make use of inheritance/composition.
4. **Debuggability:** if scriptlet throws an exception halfway, all you get is a blank page.
5. **Testability:** scriptlets are not unit-testable.
6. **Maintainability:** per saldo more time is needed to maintain mingled/cluttered/duplicated code logic.

~~Sun~~ Oracle itself also recommends in the JSP coding conventions to avoid use of *scriptlets* whenever the same functionality is possible by (tag) classes. Here are several cites of relevance:

> From JSP 1.2 Specification, it is highly recommended that the JSP Standard Tag Library (JSTL) be used in your web application to help **reduce the need for JSP scriptlets** in your pages. Pages that use JSTL are, in general, easier to read and maintain.
>
> ...
>
> Where possible, **avoid JSP scriptlets** whenever tag libraries provide equivalent functionality. This makes pages easier to read and maintain, helps to separate business logic from presentation logic, and will make your pages easier to evolve into JSP 2.0-style pages (JSP 2.0 Specification supports but de-emphasizes the use of scriptlets).
>
> ...
>
> In the spirit of adopting the model-view-controller (MVC) design pattern to reduce coupling between the presentation tier from the business logic, **JSP scriptlets should not be used** for writing business logic. Rather, JSP scriptlets are used if necessary to transform data (also called "value objects") returned from processing the client's requests into a proper client-ready format. Even then, this would be better done with a front controller servlet or a custom tag.

How to replace *scriptlets* entirely depends on the sole purpose of the code/logic. More than often this code is to be placed in a fullworthy Java class:

- If you want to invoke the **same** Java code on *every* request, less-or-more regardless of the requested page, e.g. checking if a user is logged in, then

implement a filter and write code accordingly in `doFilter()` method. E.g.:

```
  public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws ServletException, IOException {
      if (((HttpServletRequest) request).getSession().getAttribute("user") =
null) {
          ((HttpServletResponse) response).sendRedirect("login"); // Not log
in, redirect to login page.
      } else {
          chain.doFilter(request, response); // Logged in, just continue
request.
      }
  }
```

When mapped on an appropriate `<url-pattern>` covering the JSP pages of interest, then you don't need to copypaste the same piece of code overall JSP pages.

- If you want to invoke some Java code to **process a GET request**, e.g. preloading some list from a database to display in some table, if necessary based on some query parameters, then implement a servlet and write code accordingly in `doGet()` method. E.g.:

```
  protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      try {
          List<Product> products = productService.list(); // Obtain all
products.
          request.setAttribute("products", products); // Store products in
request scope.
          request.getRequestDispatcher("/WEB-
INF/products.jsp").forward(request, response); // Forward to JSP page to
display them in a HTML table.
      } catch (SQLException e) {
          throw new ServletException("Retrieving products failed!", e);
      }
  }
```

This way dealing with exceptions is easier. The DB is not accessed in the midst of JSP rendering, but far before the JSP is been displayed. You still have the possibility to change the response whenever the DB access throws an exception. In the above example, the default error 500 page will be displayed which you can anyway customize by an `<error-page>` in `web.xml`.

- If you want to invoke some Java code to **process a POST request**, such as gathering data from a submitted HTML form and doing some business stuff with it (conversion, validation, saving in DB, etcetera), then implement a servlet and write code accordingly in `doPost()` method. E.g.:

```
  protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      String username = request.getParameter("username");
      String password = request.getParameter("password");
```

```
        User user = userService.find(username, password);

        if (user != null) {
            request.getSession().setAttribute("user", user); // Login user.
            response.sendRedirect("home"); // Redirect to home page.
        } else {
            request.setAttribute("message", "Unknown username/password. Please
retry."); // Store error message in request scope.
            request.getRequestDispatcher("/WEB-INF/login.jsp").forward(request
response); // Forward to JSP page to redisplay login form with error.
        }
    }
```

This way dealing with different result page destinations is easier: redisplaying the form with validation errors in case of an error (in this particular example you can redisplay it using `${message}` in EL), or just taking to the desired target page in case of success.

- If you want to invoke some Java code to **control** the execution plan and/or the destination of the request and the response, then implement a servlet according to the MVC's Front Controller Pattern. E.g.:

```
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        try {
            Action action = ActionFactory.getAction(request);
            String view = action.execute(request, response);

            if (view.equals(request.getPathInfo().substring(1)) {
                request.getRequestDispatcher("/WEB-INF/" + view +
".jsp").forward(request, response);
            } else {
                response.sendRedirect(view);
            }
        } catch (Exception e) {
            throw new ServletException("Executing action failed.", e);
        }
    }
```

Or just adopt an MVC framework like JSF, Spring MVC, Wicket, etc so that you end up with just a JSP/Facelets page and a JavaBean class without the need for a custom servlet.

- If you want to invoke some Java code to **control the flow** inside a JSP page, then you need to grab an (existing) flow control taglib like JSTL core. E.g. displaying `List<Product>` in a table:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
...
<table>
    <c:forEach items="${products}" var="product">
        <tr>
            <td>${product.name}</td>
            <td>${product.description}</td>
            <td>${product.price}</td>
```

```
            </tr>
        </c:forEach>
    </table>
```

With XML-style tags which fit nicely among all that HTML, the code is better readable (and thus better maintainable) than a bunch of scriptlets with various opening and closing braces (*"Where the heck does this closing brace belong to?"*). An easy aid is to configure your web application to throw an exception whenever *scriptlets* are still been used by adding the following piece to `web.xml`:

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <scripting-invalid>true</scripting-invalid>
    </jsp-property-group>
</jsp-config>
```

In Facelets, the successor of JSP, which is part of the Java EE provided MVC framework JSF, it is already **not** possible to use *scriptlets*. This way you're automatically forced to do things "the right way".

- If you want to invoke some Java code to **access and display** "backend" data inside a JSP page, then you need to use EL (Expression Language), those `${}` things. E.g. redisplaying submitted input values:

  ```
  <input type="text" name="foo" value="${param.foo}" />
  ```

  The `${param.foo}` displays the outcome of `request.getParameter("foo")`.

- If you want to invoke some **utility** Java code directly in the JSP page (typically `public static` methods), then you need to define them as EL functions. There's a standard functions taglib in JSTL, but you can also easily create functions yourself. Here's an example how JSTL `fn:escapeXml` is useful to prevent XSS attacks.

  ```
  <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
  ...
  <input type="text" name="foo" value="${fn:escapeXml(param.foo)}" />
  ```

  Note that the XSS sensitivity is in no way specifically related to Java/JSP/JSTL/EL/whatever, this problem needs to be taken into account in **every** web application you develop. The problem of *scriptlets* is that it provides no way of builtin preventions, at least not using the standard Java API. JSP's successor Facelets has already implicit HTML escaping, so you don't need to worry about XSS holes in Facelets.

See also:

- What's the difference between JSP, Servlet and JSF?
- How does Servlet, ServletContext, HttpSession and HttpServletRequest/Response work?
- Basic MVC example with JSP, Servlet and JDBC
- Design patterns in Java web applications
- Hidden features of JSP/Servlet

Share

Improve this answer

Follow

edited Jun 6, 2022 at 12:00

answered Jul 5, 2010 at 14:19

BalusC

1

121 +1 Great answer. But don't go dogmatic, sometime using scriptlets IS ok, but that should be the exception that proves the rule. – svachon Jul 6, 2010 at 1:58

5 @BalusC A few times I've been stuck with java classes that didn't follow the getter/setter pattern. IMHO that is a case where a scripltet does the job. – svachon Jul 6, 2010 at 14:30

43 @svachon: I'd wrap those classes with own javabean classes and use them instead. – BalusC Jul 6, 2010 at 14:37

233

As a Safeguard: Disable Scriptlets For Good

As another question is discussing, you can and always should disable scriptlets in your `web.xml` web application descriptor.

I would always do that in order to prevent any developer adding scriptlets, especially in bigger companies where you will lose overview sooner or later. The `web.xml` settings look like this:

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
     <scripting-invalid>true</scripting-invalid>
  </jsp-property-group>
</jsp-config>
```

Share
Improve this answer
Follow

edited May 23, 2017 at 10:31
Community Bot
1 • 1

answered Aug 14, 2011 at 21:59

Stefan Schubert-Peters
5,459 • 2 • 21 • 21

17 @MrSpoon tracked down an answer for you. As per this answer + comment, this turns off scriptlets `<% %>`, scriptlet expressions `<%! %>`, and scriptlet declarations `<%= %>`. That means directives `<%@ %>` and comments `<%-- --%>` remain enabled and usable, so you can still do comments and includes. – Marsh Jul 29, 2015 at 21:55

112

JSTL offers tags for conditionals, loops, sets, gets, etc. For example:

```
<c:if test="${someAttribute == 'something'}">
   ...
</c:if>
```

JSTL works with request attributes - they are most often set in the request by a Servlet, which *forwards* to the JSP.

Share
Improve this answer
Follow

edited Jul 22, 2016 at 12:17

answered Jul 5, 2010 at 7:28

Bozho
597k • 147 • 1.1k • 1.2k

56

You can use JSTL tags together with EL expressions to avoid intermixing Java and HTML code:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<html>
    <head>
    </head>
    <body>

        <c:out value="${x + 1}" />
        <c:out value="${param.name}" />
        // and so on

    </body>
</html>
```

Share
Improve this answer
Follow

edited Oct 8, 2018 at 15:36

answered Jul 5, 2010 at 8:45

Behrang Saeedzadeh
47.8k ● 27 ● 124 ● 165

36

There are also component-based frameworks, such as Wicket, that generate a lot of the HTML for you.

The tags that end up in the HTML are extremely basic and there is virtually no logic that gets mixed in. The result is almost empty-like HTML pages with typical HTML elements. The downside is that there are a lot of components in the **Wicket** API to learn and some things can be difficult to achieve under those constraints.

Share
Improve this answer
Follow

edited Aug 25, 2020 at 21:06

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Mar 22, 2011 at 18:24

tsand
501 ● 4 ● 5

36

In the MVC architectural pattern, JSPs represent the view layer. Embedding Java code in JSPs is considered a bad practice.

You can use JSTL, freeMarker, and velocity with JSP as a "template engine".

The data provider to those tags **depends on frameworks** that you are dealing with. Struts 2 and WebWork as an implementation for the MVC pattern uses OGNL "very interesting technique to expose Beans properties to JSP".

Share
Improve this answer
Follow

edited Feb 13, 2021 at 10:28

eis
53.4k ● 14 ● 155 ● 202

answered Feb 4, 2011 at 10:41

Sami Jmii
470 ● 7 ● 17

Experience has shown that JSP's have some shortcomings, one of them being hard to avoid mixing markup with actual code.

29

If you can, then consider using a specialized technology for what you need to do. In Java EE 6 there is JSF 2.0, which provides a lot of nice features including gluing Java beans together with JSF pages through the `#{bean.method(argument)}` approach.

Share Improve this answer Follow

answered Jul 5, 2010 at 8:30

Thorbjørn Ravn Andersen

75.3k ● 34 ● 199 ● 352

4 Old answer but I can not resist to tell that JSF is one of the most horrendous invention in Java space. Try to make a single (HTTP GET like) link and you will understand why. – Alex Aug 17, 2015 at 8:11

@Alex but still better. Feel free to recommend something even better.
– Thorbjørn Ravn Andersen Dec 22, 2016 at 9:05

@ThorbjørnRavnAndersen scriptlets – Criticize SE actions means ban Jul 19, 2021 at 15:30

@user253751 Yes, scriptlets is the "mixing markup with actual code" shortcoming mentioned.
– Thorbjørn Ravn Andersen Jul 21, 2021 at 11:58

@ThorbjørnRavnAndersen markup is actual code – Criticize SE actions means ban Jul 21, 2021 at 12:20

28

If you simply want to avoid the drawbacks of Java coding in JSP you can do so even with scriplets. Just follow some discipline to have minimal Java in JSP and almost no calculation and logic in the JSP page.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<% // Instantiate a JSP controller
MyController clr = new MyController(request, response);

// Process action, if any
clr.process(request);

// Process page forwarding, if necessary

// Do all variable assignment here
String showMe = clr.getShowMe();%>

<html>
    <head>
    </head>
    <body>
        <form name="frm1">
            <p><%= showMe %>
            <p><% for(String str : clr.listOfStrings()) { %>
            <p><%= str %><% } %>

            // And so on
        </form>
```

```
    </body>
</html>
```

Share

Improve this answer

Follow

edited Aug 25, 2020 at 20:57

Peter Mortensen

31.6k • 22 • 109 • 133

answered May 11, 2012 at 16:45

dipu

1,340 • 3 • 16 • 28

27

Learn to customize and write your own tags using JSTL

Note that EL is **EviL** (runtime exceptions and refactoring).

Wicket may be evil too (performance and toilsome for small applications or simple view tier).

Example from *java2s*

This must be added to the web application's web.xml

```
<taglib>
    <taglib-uri>/java2s</taglib-uri>
    <taglib-location>/WEB-INF/java2s.tld</taglib-location>
</taglib>
```

Create file *java2s.tld* in the */WEB-INF/*

```
<!DOCTYPE taglib
  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
   "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<!-- A tab library descriptor -->
<taglib xmlns="http://java.sun.com/JSP/TagLibraryDescriptor">
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>Java2s Simple Tags</short-name>

    <!-- This tag manipulates its body content by converting it to upper case
    -->
    <tag>
        <name>bodyContentTag</name>
        <tag-class>com.java2s.BodyContentTag</tag-class>
        <body-content>JSP</body-content>
        <attribute>
          <name>howMany</name>
        </attribute>
    </tag>
</taglib>
```

Compile the following code into *WEB-INF\classes\com\java2s*

```
package com.java2s;

import java.io.IOException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.BodyContent;
import javax.servlet.jsp.tagext.BodyTagSupport;

public class BodyContentTag extends BodyTagSupport{
    private int iterations, howMany;

    public void setHowMany(int i){
        this.howMany = i;
    }

    public void setBodyContent(BodyContent bc){
        super.setBodyContent(bc);
        System.out.println("BodyContent = '" + bc.getString() + "'");
    }

    public int doAfterBody(){
        try{
            BodyContent bodyContent = super.getBodyContent();
            String bodyString  = bodyContent.getString();
            JspWriter out = bodyContent.getEnclosingWriter();

            if ( iterations % 2 == 0 )
                out.print(bodyString.toLowerCase());
            else
                out.print(bodyString.toUpperCase());

            iterations++;
            bodyContent.clear(); // empty buffer for next evaluation
        }
        catch (IOException e) {
            System.out.println("Error in BodyContentTag.doAfterBody()" +
e.getMessage());
            e.printStackTrace();
        } // End of catch

        int retValue = SKIP_BODY;

        if ( iterations < howMany )
            retValue = EVAL_BODY_AGAIN;

        return retValue;
    }
}
```

Start the server and load the bodyContent.jsp file in the browser:

```
<%@ taglib uri="/java2s" prefix="java2s" %>
<html>
    <head>
        <title>A custom tag: body content</title>
    </head>
    <body>
        This page uses a custom tag manipulates its body content.Here is its
output:
        <ol>
```

```
            <java2s:bodyContentTag howMany="3">
            <li>java2s.com</li>
            </java2s:bodyContentTag>
        </ol>
    </body>
</html>
```

Share

Improve this answer

Follow

edited Aug 25, 2020 at 21:02

Peter Mortensen

31.6k • 22 • 109 • 133

answered Aug 17, 2011 at 20:02

tomasb

1,673 • 2 • 22 • 29

though reusability of components is fine it targets some field – tomasb Aug 20, 2011 at 20:44

26

Wicket is also an alternative which completely separates Java from HTML, so a designer and programmer can work together and on different sets of code with little understanding of each other.

Look at Wicket.

Share

Improve this answer

Follow

edited Aug 25, 2020 at 21:05

Peter Mortensen

31.6k • 22 • 109 • 133

answered May 20, 2011 at 20:42

msj121

2,842 • 4 • 30 • 58

23

You raised a good question and although you got good answers, I would suggest that you get rid of JSP. It is outdated technology which eventually will die. Use a modern approach, like template engines. You will have very clear separation of business and presentation layers, and certainly no Java code in templates, so you can generate templates directly from web presentation editing software, in most cases leveraging WYSIWYG.

And certainly stay away of filters and pre and post processing, otherwise you may deal with support/debugging difficulties since you always do not know where the variable gets the value.

Share

Improve this answer

Follow

edited Apr 20, 2014 at 0:02

glen3b

693 • 1 • 8 • 22

answered Jul 12, 2011 at 20:20

Dmitriy R

613 • 1 • 5 • 12

15 JSP is itself a template engine – WarFox Jul 13, 2011 at 7:28

1 You can mix business and presentation layers just as well with most other template engines because most of them are turing complete. I think stringtemplate.org is one of the not-turing-complete exceptions. – mihca Jun 17, 2021 at 6:07

21

No matter how much you try to avoid, when you work with other developers, some of them will still prefer scriptlet and then insert the evil code into the project. Therefore, setting up the project at the first sign is very important if you really want to reduce the scriptlet code. There are several techniques to get over this (including several frameworks that other mentioned). However, if you prefer the pure JSP way, then use the JSTL tag file. The nice thing about this is you can also set up master pages for your project, so the other pages can inherit the master pages

Create a master page called base.tag under your WEB-INF/tags with the following content

```
<%@tag description="Overall Page template" pageEncoding="UTF-8"%>

<%@attribute name="title" fragment="true" %>

<html>
  <head>
    <title>
       <jsp:invoke fragment="title"></jsp:invoke>
    </title>

  </head>
  <body>
    <div id="page-header">
       ....
    </div>
    <div id="page-body">
      <jsp:doBody/>
    </div>
    <div id="page-footer">
      .....
    </div>
  </body>
</html>
```

On this mater page, I created a fragment called "title", so that in the child page, I could insert more codes into this place of the master page. Also, the tag `<jsp:doBody/>` will be replaced by the content of the child page

Create child page (child.jsp) in your WebContent folder:

```
<%@ taglib prefix="t" tagdir="/WEB-INF/tags" %>

<t:base>
    <jsp:attribute name="title">
        <bean:message key="hello.world" />
    </jsp:attribute>

    <jsp:body>
    [Put your content of the child here]
```

```
    </jsp:body>
</t:base>
```

`<t:base>` is used to specify the master page you want to use (which is base.tag at this moment). All the content inside the tag `<jsp:body>` here will replace the `<jsp:doBody/>` on your master page. Your child page can also include any tag lib and you can use it normally like the other mentioned. However, if you use any scriptlet code here (`<%= request.getParameter("name") %>` ...) and try to run this page, you will get a `JasperException because  Scripting elements ( <%!, <jsp:declaration, <%=, <jsp:expression, <%, <jsp:scriptlet ) are disallowed here`. Therefore, there is no way other people can include the evil code into the jsp file

Calling this page from your controller:

You can easily call the child.jsp file from your controller. This also works nice with the struts framework

Share Improve this answer Follow

answered Jan 26, 2013 at 18:07

Thai Tran
9,915 ● 7 ● 46 ● 66

1 "No matter how much you try to avoid, when you work with other developers, some of them will still prefer scriptlet and then insert the evil code into the project." See "As a Safeguard: Disable Scriptlets For Good" answer. – Lluis Martinez Jul 22, 2019 at 20:52

Or you could just explain why scriptlets are evil and if you can't explain it, maybe they're not actually evil after all. – Criticize SE actions means ban Jul 19, 2021 at 15:31

21

In order to avoid Java code in JSP files, Java now provides tag libraries, like JSTL.

Also, Java has come up with JSF into which you can write all programming structures in the form of tags.

Share
Improve this answer
Follow

edited Aug 25, 2020 at 21:08

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Feb 23, 2011 at 6:14

mahesh
4,763 ● 11 ● 45 ● 61

18

Use JSTL tag libraries in JSP. That will work perfectly.

Share
Improve this answer

edited Aug 25, 2020 at 21:04

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Jun 5, 2011 at 4:23

Chandra Sekhar
16.5k ● 10 ● 70 ● 90

Follow

Just use the JSTL tag and EL expression.

17

Share Improve this answer Follow

answered Jul 11, 2011 at 6:59

tanglei

347 ● 2 ● 6

17

If somebody is really against programming in more languages than one, I suggest GWT. Theoretically, you can avoid all the JavaScript and HTML elements, because Google Toolkit transforms all the client and shared code to JavaScript. You won't have problem with them, so you have a webservice without coding in any other languages. You can even use some default CSS from somewhere as it is given by extensions (smartGWT or Vaadin). You don't need to learn dozens of annotations.

Of course, if you want, you can hack yourself into the depths of the code and inject JavaScript and enrich your HTML page, but really you can avoid it if you want, and the result will be good as it was written in any other frameworks. I it's say worth a try, and the basic GWT is well-documented.

And of course many fellow programmers hereby described or recommended several other solutions. GWT is for people who really don't want to deal with the web part or to minimize it.

Share

Improve this answer

Follow

edited Aug 25, 2020 at 20:54

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Jan 21, 2013 at 16:04

CsBalazsHungary

813 ● 14 ● 31

1 Doesn't really answer the OP's question. – Evan Donovan Apr 2, 2014 at 16:46

1 @EvanDonovan well, practically it does give an answer. You don't need to mess with Java codes mixing up with other languages. I admit it uses Java for coding, but it will be translated to JS without Java calls. But the question's scope is how to avoid the chaos of the classic JSP. And GWT technology solves that. I added up this answer since nobody mentioned it, but relevant since it is an alternative to JSP. I didn't want to answer the question's whole scope, but to add a valuable information for people who are ooking for alternatives.
– CsBalazsHungary Apr 3, 2014 at 4:17

16

Using scriptlets in JSPs is not a good practice.

Instead, you can use:

1. JSTL tags
2. EL expressions
3. Custom Tags- you can define your own tags to use.

Please refer to:

1. http://docs.oracle.com/javaee/1.4/tutorial/doc/JSTL3.html
2. EL

Share

Improve this answer

Follow

edited Nov 22, 2013 at 16:02

RustyTheBoyRobot
5,945 ● 4 ● 37 ● 55

answered Jul 22, 2013 at 5:59

kapil das
2,101 ● 1 ● 28 ● 29

16

A neat idea from the Python world is *Template attribute languages*; TAL was introduced by Zope (therefore a.k.a. "Zope Page Templates", ZPT) and is a standard, with implementations in PHP, XSLT and Java as well (I have used the Python/Zope and PHP incarnations). In this class of templating languages, one of the above examples could look like this:

```
<table>
    <tr tal:repeat="product products">
        <td tal:content="product/name">Example product</td>
        <td tal:content="product/description">A nice description</td>
        <td tal:content="product/price">1.23</td>
    </tr>
</table>
```

The code looks like ordinary HTML (or XHTML) plus some special attributes in an XML namespace; it can be viewed with a browser and safely be tweaked by a designer.

There is support for macros and for internationalisation and localisation as well:

```
<h1 i18n:translate="">Our special offers</h1>
<table>
    <tr tal:repeat="product products">
        <td tal:content="product/name"
            i18n:translate="">Example product</td>
        <td tal:content="product/description"
            i18n:translate="">A nice description</td>
        <td tal:content="product/price">1.23</td>
    </tr>
</table>
```

If translations of the content are available, they are used.

I don't know very much about the Java implementation, though.

Share

Improve this answer

Follow

edited Aug 25, 2020 at 20:55

Peter Mortensen

31.6k • 22 • 109 • 133

answered May 21, 2012 at 7:51

Tobias

2,561 • 3 • 27 • 40

1 JSP has since dec 2009 been succeeded by Facelets which supports this stuff. Facelets is also XML based. See also among others the Facelets chapter in Java EE 6 tutorial and `ui:xxx` tags in Facelts VDL. – BalusC May 23, 2012 at 18:53

I don't know Facelets very well, but IIRC it is all about writing classes which implement custom XML elements. The TAL/ZPT way is to have templates which contain true (X)HTML with special attributes which fill or replace the original elements; thus, you can view the working template and see a prototype with nice dummy content. I'm not sure Facelets allow to tweak the original HTML elements (w/o an additional namespace) using custom attributes. – Tobias Jul 31, 2012 at 14:28

I just had another look at this Facelets stuff. It contains all sorts of validation facilities etc. and thus follows a completely different philosophy than TAL. The TAL way is, "Keep the logic out of the template as cleanly as possible; have all the complicated stuff be done by the controller which feeds it." You won't ever give a Facelets template to a designer to have him/her tweak it; it's just not possible. Regarding the generated content - it's just like using `tal:replace="structure (expression)"` attributes all the time. – Tobias Mar 23, 2014 at 8:51

14

Sure, replace `<%! counter++; %>` by an event producer-consumer architecture, where the business layer is notified about the need to increment the counter, it reacts accordingly, and notifies the presenters so that they update the views. A number of database transactions are involved, since in future we will need to know the new and old value of the counter, who has incremented it and with what purpose in mind. Obviously serialization is involved, since the layers are entirely decoupled. You will be able to increment your counter over RMI, IIOP, SOAP. But only HTML is required,

which you don't implement, since it is such a mundane case. Your new goal is to reach 250 increments a second on your new shiny E7, 64GB RAM server.

I have more than 20 years in programming, most of the projects fail before the sextet: Reusability Replaceability OO-ability Debuggability Testability Maintainability is even needed. Other projects, run by people who only cared about functionality, were extremely successful. Also, stiff object structure, implemented too early in the project, makes the code unable to be adapted to the drastic changes in the specifications (aka agile).

So I consider as procrastination the activity of defining "layers" or redundant data structures either early in the project or when not specifically required.

Share
Improve this answer
Follow

edited Nov 22, 2017 at 12:22

Alex Weitz
3,359 ● 5 ● 38 ● 61

answered Mar 15, 2014 at 17:17

Razvan
157 ● 1 ● 2

11

Technically, JSP are all converted to Servlets during runtime.

JSP was initially created for the purpose of the decoupling the business logic and the design logic, following the MVC pattern. So JSP is technically all Java code during runtime.

But to answer the question, *tag libraries* are usually used for applying logic (removing Java code) to JSP pages.

Share
Improve this answer
Follow

edited Aug 25, 2020 at 20:39

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Jun 14, 2013 at 9:31

mel3kings
9,375 ● 3 ● 63 ● 70

9

How can I avoid Java code in JSP files?

You can use tab library tags like **JSTL** in addition to Expression Language (**EL**). But EL does not work well with JSP. So it's is probably better to drop JSP completely and use **Facelets**.

Facelets is the first non JSP page declaration language designed for **JSF (Java Server Faces)** which provided a simpler and more powerful programming model to JSF developers as compare to JSP. It resolves different issues occurs in JSP for web applications development.

Feature Name	JSP	Facelets
Pages are compile	A Servlet that gets executed each time the page renders. The UI Component hierarchy is built by the presence of custom tags in the page.	An abstract syntax tree that, when executed, builds a UIComponent hierarchy
Handling of tag attributes	All tag attributes must be declared in a TLD file.	Tag attributes are completely dynamic and automatically map to properties, attributes and ValueExpressions on UIComponent instances
Page template	Not supported directly. For templating must go outside of core JSP	Page templating is a core feature of Facelets
Performance	Due to the common implementation technique of compiling a JSP page to a Servlet, performance can be slow	Facelets is simpler and faster than JSP
EL Expressions	Expressions in template text cause unexpected behavior when used in JSP	Expressions in template text operate as expected.
JCP Standard	Yes, the specification is separate from the implementation for JSP	No, the specification is defined by and is one with the implementation.

Source

Share
Improve this answer
Follow

edited Aug 25, 2020 at 20:22

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Dec 24, 2015 at 11:47

Yster
3,295 ● 5 ● 34 ● 48

I definitely second this response. JSF with or without Facelets. I thought developing in JSP had largely ceased more than 10 years ago. I last wrote JSPs in 2000! – casgage Feb 1, 2017 at 2:52

9

If we use the following things in a Java web application, Java code can be eliminated from the foreground of the JSP file.

1. Use the MVC architecture for a web application
2. Use JSP Tags

a. Standard Tags

b. Custom Tags

3. Expression Language

Share
Improve this answer
Follow

edited Aug 25, 2020 at 20:59

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Apr 2, 2012 at 13:52

Ajay Takur
6,216 ● 5 ● 44 ● 58

5

Using Scriptlets is a very old way and *not* recommended. If you want directly output something in your JSP pages, just use **Expression Language (EL)** along with **JSTL**.

There are also other options, such as using a templating engine such as Velocity, Freemarker, Thymeleaf, etc. But using plain JSP with EL and JSTL serves my purpose most of the time and it also seems the simplest for a beginner.

Also, take note that it is not a best practice to do business logic in the view layer. You should perform your business logic in the *service* layer, and pass the output result to your views through a *controller*.

Share
Improve this answer
Follow

edited Aug 25, 2020 at 20:16

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Nov 30, 2017 at 15:59

adn.911
1,314 ● 3 ● 17 ● 32

4

Use a Backbone.js or AngularJS-like JavaScript framework for UI design and fetch the data using a REST API. This will remove the Java dependency from the UI completely.

Share
Improve this answer
Follow

edited Aug 25, 2020 at 20:12

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Jun 20, 2018 at 13:49

Sumesh TG
2,575 ● 2 ● 16 ● 29

4

Nothing of that is used anymore, my friend. My advice is to decouple the view (CSS, HTML, JavaScript, etc.) from the server.

In my case, I do my systems handling the view with Angular and any data needed is brought from the server using REST services.

Believe me, this will change the way you design.

Share
Improve this answer
Follow

edited Aug 25, 2020 at 20:14

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Dec 12, 2017 at 19:45

Eduardo
2,280 ● 23 ● 30

> It will change the way we design, but not always in a good way with good results. – Yuriy N. Feb 9, 2022 at 16:11

> It will change the way we design, but not always in a good way with good results. We will face 1. Longer initial loading 2. slow indexing by Google, no indexing by other search engines 3. performance problems on low-end mobile devices. SPA frameworks should be used for SPA. Creating content websites using SPA frameworks, especially heavy ones like Angular or React is a way to trouble. – Yuriy N. Feb 9, 2022 at 16:18

4

JSP 2.0 has a feature called **"Tag Files"**, and you can write tags without external Java code and `tld`. You need to create a `.tag` file and put it in `WEB-INF\tags`. You can even create a directory structure to package your tags.

For example:

```
/WEB-INF/tags/html/label.tag

<%@tag description="Rensders a label with required css class"
pageEncoding="UTF-8"%>
<%@attribute name="name" required="true" description="The label"%>

<label class="control-label control-default"  id="${name}Label">${name}</label>
```

Use it like

```
<%@ taglib prefix="h" tagdir="/WEB-INF/tags/html"%>
<h:label  name="customer name" />
```

Also, you can read the tag body easily:

```
/WEB-INF/tags/html/bold.tag
<%@tag description="Bold tag" pageEncoding="UTF-8"%>
<b>
  <jsp:doBody/>
</b>
```

Use it:

```
<%@ taglib prefix="h" tagdir="/WEB-INF/tags/bold"%>
<h:bold>Make me bold</h:bold>
```

The samples are very simple, but you can do lots of complicated tasks here. Please consider you can use other tags (for example: `JSTL` which has controlling tags like `if/forEcah/chosen` text manipulation like `format/contains/uppercase` or even SQL tags `select/update`), pass all kind parameters, for example `Hashmap`, access `session`, `request`, ... in your tag file too.

Tag File are so easy developed as you did not need to restart the server when changing them, like JSP files. This makes them easy for development.

Even if you use a framework like Struts 2, which have lots of good tags, you may find that having your own tags can reduce your code a lot. You can pass your tag parameters to struts and this way customize your framework tag.

You can use tags not only to avoid Java, but also minimize your HTML codes. I myself try to review HTML code and build tags a lot as soon as I see code duplicates start in my pages.

(Even if you end up using Java in your JSP code, which I hope not, you can encapsulate that code in a tag.)

Share

Improve this answer

Follow

edited Aug 25, 2020 at 20:20

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Apr 12, 2016 at 14:55

Alireza Fattahi

45.3k ● 16 ● 131 ● 184

3

A lot of the answers here go the "use a framework" route. There's zero wrong with that. However I don't think it really answers your question, because frameworks may or may not use JSPs, nor are they designed in any way with removing java use in JSPs as a primary goal.

The only good answer to your question "how do I avoid using Java in a JSP" is: you can't.

That's what JSPs are for - using Java to render HTML with dynamic data/logic. The follow up question might be, how much java should I use in my JSPs.
Before we answer that question, you should also ponder, "do I need to use JSPs to build web content using Java?" The answer to that last one is, no. There are many alternatives to JSPs for developing web facing applications using Java. Struts for example does not force you to use JSPs - don't get me wrong, you can use them and many implementations do, but you don't absolutely have to. Struts doesn't even force you to use any HTML. A JSP doesn't either, but let's be honest, a JSP producing no HTML is kinda weird. Servlets, famously, allow you to serve any kind of content you like over HTTP dynamically. They are *the* primary tech behind pretty much everything java web - JSPs are just HTML templates for servlets, really.
So the answer to how much java you should put in a JSP is, "as little as possible". I of course have java in my JSPs, but it consists exclusively of tag library definitions, session and client variables, and beans encapsulating server side objects. The <%%> tags in my HTML are almost exclusively property calls or variable expressions. Rare exceptions include ultra-specific calculations pertaining to a single page and unlikely to ever be reused; bugfixes stemming from page-specific issues only applying to one page; last minute concatenations and arithmetic stemming from unusual requirements limited in scope to a single page; and other similar cases. In a code set of 1.5 million lines, 3000 JSPs and 5000 classes, there are maybe 100 instances of such unique snippets. It would have been quite possible to make these changes in classes or tag library definitions, but it would have been inordinately complex due to the specificity of each case, taken longer to write and debug, and taken more time as a result to get to

my users. It's a judgement call. But make no mistake, you cannot write JSPs of any meaning with "no java" nor would you want to. The capability is there for a reason.

Share Improve this answer Follow

answered Aug 28, 2020 at 0:03

Philippe Damerval

96 ● 4

That is some wall of text. Can you break it down? – Peter Mortensen Dec 29, 2022 at 1:26

2

1. Make your values and parameters inside your servlet classes
2. Fetch those values and parameters within your JSP using JSTL/Taglib

> The good thing about this approach is that your code is also HTML like code!

Share Improve this answer Follow

answered Dec 1, 2018 at 16:04

Mehdi

3,753 ● 3 ● 37 ● 65

2

Java itself is a very nice language, but heavy usage in the enterprise environment made its standard solutions extremely (ridiculously) difficult. Examples: JSTL, JSF, Wicket, etc.

Here is a super lightweight approach to creating a backend in Java:

- don't use JSP (or any other template engine) at all;
- use plain HTML templates;
- use JSOUP to parse HTML templates into Document object;
- modify Document object using its very intuitive jQuery like methods;
- return Document.toString() as response to request.

I use it for one of my side projects (hosted Digitalocean $5 droplet, Nginx, Tomcat) and it is very fast: according to Googlebot average response time is about 160 ms.

JSoup

Share

Improve this answer

Follow

edited Jan 3, 2023 at 8:53

answered Jan 22, 2022 at 19:17

Yuriy N.

6,003 ● 2 ● 46 ● 38

Can you add a reference to JSOUP? (But ********** ***without*** ********** "Edit:", "Update:", or similar - the answer should appear as if it was written today). – Peter Mortensen Dec 29, 2022 at 1:28

1

As many answers says, use JSTL or create your own custom tags. Here is a good explanation about creating custom tags.

Share

Improve this answer

Follow

edited Aug 25, 2020 at 20:17

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Jul 7, 2017 at 20:24

Vikas

7,167 ● 4 ● 29 ● 45

1 2 Next

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.