JSP

コーディング規約

・Code Conventions for the JavaServer Pages Technology Version 1.x Language
・Code Conventions for the JavaServer Pages Technology Version 1.x Language

■抄訳

Why Have Code Conventions? 何故コーディング規約をもつのか

コーディング規約はいくつかの理由でプログラマと web コンテンツ開発者にとって重要である。Code conventions are important to programmers and web content developers for a number of reasons:

1. They improve the readability of software artifacts

ソフトウェア著作物の読みやすさを改善する

2. They reduce training management and effort

訓練の管理と努力を低減する

3. They leverage organizational commitment towards standardization

標準化のために組織への帰属意識をてこ入れする

File Names and Locations ファイルの名前と位置 File naming gives tool vendors and web containers a way to determine file types and interpret them accordingly. The following table lists our recommended file suffixes and locations. 以下の表は我々が推奨するファイル接尾辞と位置を一覧する。

File Type	File Suffix	Recommended Location
JSP technology	.jsp	<context root>/<subsystem path>/
JSP fragment	.jsp	<context root>/<subsystem path>/
--	.jspf	<context root>/WEB-INF/jspf/<subsystem path>/
cascading style sheet	.css	<context root>/css/
JavaScript technology	.js	<context root>/js/
HTML page	.html	<context root>/<subsystem path>/
web resource	.gif .jpg etc.	<context root>/images/
tag library descriptor	.tld	<context root>/WEB-INF/tld/

There are a few things to keep in mind when reading the table above. First, 上の表を読むときにいくつか心に留めておく事が有る。最初に、<context root> は web アプリケーションのコンテキストの

ルート (.war ファイル内のルートディレクトリ) である。次に、<subsystem path> は動的なものと静的な web ページコンテンツの論理的なグループに使う。<context root> is the root of the context of the web application (the root directory inside a .war file). Second, <subsystem path> is used to provide refined logical grouping of dynamic and static web page contents. For a small web application, this may be an empty string. 小さなウェブアプリケーションに対してこれは多分空の文字列だろう。三つ目は、他の JSP ページにインクルードされる Third, we use the term JSP fragment to refer to a JSP page that can be included in another JSP page. Note that in the JSP 2.0 Specification, the term "JSP segment" is used instead as the term "JSP fragment" is overloaded. JSP fragments can use either .jsp or .jspf as a suffix, and should be placed either in /WEB-INF/jspf or with the rest of the static content, respectively. JSP fragments that are not complete pages should always use the .jspf suffix and should always be placed in /WEB-INF/jspf. Fourth, though the JSP specification recommends both .jspf and .jsp as possible extensions for JSP fragments, we recommend .jspf as .jsf might be used by the JavaServer Faces specification.

Finally, it is in general a good practice to place tag library descriptor files and any other non-public content under WEB-INF/ or a subdirectory underneath it. In this way, the content will be inaccessible and invisible to the clients as the web container will not serve any files underneath WEB-INF/.

An optional welcome file's name, as declared in the welcome-file element of the deployment descriptor (web.xml), should be index.jsp if dynamic content will be produced, or index.html if the welcome page is static.

When internationalizing JSP files, we recommend that you group JSP pages into directories by their locale. For example, the US English version of index.jsp would appear under /en_US/index.jsp whereas the Japanese version of the same file would appear under /ja_JP/index.jsp. The Java Tutorial provides additional information about internationalizing Java code in general, and the book Designing Enterprise Applications with the J2EE Platform provides information about internationalization for web applications.

File Organization ファイルの構成 A well-structured source code file is not only easier to read, but also makes it quicker to locate information within the file. In this section, we'll introduce the structures for both JSP and tag library descriptor files.JSP File / JSP Fragment File

よく構造化されたソースコードファイルは読みやすいだけでなくこのセクションでは JSP とタグライブラリ記述ファイルの両方について構造を導入する。

1 つの JSP ファイルは列挙された順に以下のセクションを含む。A JSP file consists of the following sections in the order they are listed:

1. **Opening comments 開始のコメント**
2. **JSP page directive(s) ページディレクティブ**
3. **Optional tag library directive(s) タグライブラリディレクティブ**
4. **Optional JSP declaration(s) 宣言**
5. **HTML and JSP code HTML と JSP コード**

Opening Comments 開始のコメント A JSP file or fragment file begins with a server side style comment:JSP ファイルやフラグメントファイルはサーバサイドスタイルのコメントで始まる

```
<%--
  - Author(s):
  - Date:
  - Copyright Notice:
  - @(#)
  - Description:
  --%>
```

```
<%--
  - 著作者：
  - 作成日：
  - 著作権表示：
  - @(#)
  - 概要説明：
  --%>
```

This comment is visible only on the server side because it is removed during JSP page translation. Within this comment are the author(s), the date, and the copyright notice of the revision, an identifier and a description about the JSP page for web developers. The combination of characters "@(#)" is recognized by certain programs as indicating the start of an identifier. While such programs are seldom used, the use of this string does no harm. In addition, this combination is sometimes appended by "Id" for the identification information to be automatically inserted into the JSP page by some version control programs. The Description part provides concise information about the purpose of the JSP page. It does not span more than one paragraph.

In some situations, the opening comments need to be retained during translation and propagated to the client side (visible to a browser) for authenticity and legal purposes. This can be achieved by splitting the comment block into two parts; first, the client-side style comment:

```
<!--
  - Author(s):
  - Date:
  - Copyright Notice:
  -->
```

and then a shorter server side style comment:

```
<%--
  - @(#)
  - Description:
  --%>
```

JSP Page Directive(s)

A JSP page directive defines attributes associated with the JSP page at translation time. The JSP specification does not impose a constraint on how many JSP page directives can be defined in the same page. So the following two Code Samples are equivalent (except that the first example introduces two extra blank lines in the output):

Code Sample 1:

```
<%@ page session="false" %>
<%@ page import="java.util.*" %>
<%@ page errorPage="/common/errorPage.jsp" %>
```

If the length of any directive, such as a page directive, exceeds the normal width of a JSP page (80 characters), the directive is broken into multiple lines:

Code Sample 2:

```
<%@ page    session="false"
            import="java.util.*"
            errorPage="/common/errorPage.jsp"
%>
```

In general, Code Sample 2 is the preferred choice for defining the page directive over Code Sample 1. An exception occurs when multiple Java packages need to be imported into the JSP pages, leading to a very long import attribute:

```
<%@ page    session="false"
            import="java.util.*,java.text.*,
                    com.mycorp.myapp.taglib.*,
                    com.mycorp.myapp.sql.*, ..."
...
%>
```

In this scenario, breaking up this page directive like the following is preferred:

```
<%-- all attributes except import ones --%>
<%@ page
...
%>
<%-- import attributes start here --%>
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
```

...

Note that in general the import statements follow the local code conventions for Java technology. For instance, it may generally be accepted that when up to three classes from the same package are used, import should declare specific individual classes, rather than their package. Beyond three classes, it is up to a web developer to decide whether to list those classes individually or to use the ".*" notation. In the former case, it makes life easier to identify an external class, especially when you try to locate a buggy class or understand how the JSP page interacts with Java code. For instance, without the knowledge of the imported Java packages as shown below, a web developer will have to search through all these packages in order to locate a Customer class:

<%@ page import="com.mycorp.bank.savings.*" %><%@ page import="com.thirdpartycorp.cashmanagement.*" %><%@ page import="com.mycorp.bank.foreignexchange.*" %>...

In the latter case, the written JSP page is neater but it is harder to locate classes. In general, if a JSP page has too many import directives, it is likely to contain too much Java code. A better choice would be to use more JSP tags (discussed later in this article).Optional Tag Library Directive(s)

A tag library directive declares custom tag libraries used by the JSP page. A short directive is declared in a single line. Multiple tag library directives are stacked together in the same location within the JSP page's body:

<%@ taglib uri="URI1" prefix="tagPrefix1" %><%@ taglib uri="URI2" prefix="tagPrefix2" %>...

Just as with the page directive, if the length of a tag library directive exceeds the normal width of a JSP page (80 characters), the directive is broken into multiple lines:

<%@ taglib

```
uri="URI2"
prefix="tagPrefix2"
```

%>

Only tag libraries that are being used in a page should be imported.

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.Optional JSP Declaration(s)

JSP declarations declare methods and variables owned by a JSP page. These methods and variables are no different from declarations in the Java programming language, and therefore the relevant code conventions should be followed. Declarations are preferred to be contained in a single <%! ... %> JSP declaration block, to centralize declarations within one area of the JSP page's body. Here is an example:Disparate declaration blocks, Preferred declaration block

```
<%! private int hitCount; %>
<%! private Date today; %>
...
<%! public int getHitCount() {
        return hitCount;
    }
%>
```

```
<%!
    private int hitCount;
    private Date today;

    public int getHitCount() {
        return hitCount;
    }
%>
```

HTML and JSP Code

This section of a JSP page holds the HTML body of the JSP page and the JSP code, such JSP expressions, scriptlets, and JavaBeans instructions.Tag Library Descriptor

A tag library descriptor (TLD) file must begin with a proper XML declaration and the correct DTD statement. For example, a JSP 1.2 TLD file must begin with:

<?xml version="1.0" encoding="ISO-8859-1" ?><!DOCTYPE taglib

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

This is immediately followed by a server-side style comment that lists the author(s), the date, the copyright, the identification information, and a short description about the library:

<!--

```
   - Author(s):
   - Date:
   - Copyright Notice:
   - @(#)
   - Description:
   -->
```

The rules and guidelines regarding the use of these elements are the same for those defined for JSP files/fragment files.

The rest of the file consists of the following, in the order they appear below:

```
    * Optional declaration of one tag library validator
    * Optional declaration of event listeners
    * Declaration of one or more available tags
```

It is recommended that you always add the following optional sub-elements for the elements in a TLD file. These sub-elements provide placeholders for tag designers to document the behavior and additional information of a TLD file, and disclose them to web component developers.TLD Element JSP 1.2 RecommendedSub-element JSP 1.1 RecommendedSub-elementattribute (JSP 1.2) description   init-param (JSP 1.2) description   tag display-name, description, example name, infotaglib uri, display-name, description uri, infovalidator (JSP 1.2) description   variable (JSP 1.2) description   Indentation

Indentations should be filled with space characters. Tab characters cause different interpretation in the spacing of characters in different editors and should not be used for indentation inside a JSP page. Unless restricted by particular integrated development environment (IDE) tools, a unit of indentation corresponds to 4 space characters. Here is an example:

<myTagLib:forEach var="client" items="${clients}">

```
    <myTagLib:mail value="${client}" />
```

</myTagLib:forEach>

A continuation indentation aligns subsequent lines of a block with an appropriate point in the previous line. The continuation indentation is in multiple units of the normal indentation (multiple lots of 4 space characters):

```
<%@ page    attribute1="value1"

                 attribute2="value2"
                 ...
                 attributeN="valueN"

%>
```

Indentation of Scripting Elements

When a JSP scripting element (such as declaration, scriptlet, expression) does not fit on a single line, the adopted indentation conventions of the scripting language apply to the body of the element. The body begins from the same line for the opening symbol of the element <%=, and from a new line for the opening symbol <%=. The body is then terminated by an enclosing symbol of the element (%>) on a separate line. For example:

```
<%= (Calendar.getInstance().get(Calendar.DAY_OF_WEEK)

        =Calendar.SUNDAY) ?
    "Sleep in" :
    "Go to work"

%>
```

The lines within the body not containing the opening and the enclosing symbols are preceded with one unit of normal indentation (shown as   in the previous example) to make the body distinctively identifiable from the rest of the JSP page.Compound Indentation with JSP, HTML and Java

Compound indentation, for JSP elements intermingled with Java scripting code and template text (HTML), is necessary to reduce the effort of comprehending a JSP source file. This is because the conventional normal indentation might make seeing the JSP source file difficult. As a general rule, apply an extra unit of normal indentation to every element introduced within another one. Note that this alters the indentations of the final output produced for the client side to render for display. The additional indentations, however, are usually ignored (by the browser) and have no effect on the rendered output on the browser. For instance, adding more space characters before a <TABLE> tag does not change the position of a table. So, applying this convention for indentation makes this looks nicer:

```
<table>
    <%  if ( tableHeaderRequired ) { %>
        <tr>
            <th>Last Name</th>
            <th>First Name</th>
        </tr>
    <%  } %>
    <c:forEach var="customer" items="${customers}">
        <tr>
            <td><c:out value="${customer.lastName}"/></td>
            <td><c:out value="${customer.firstName}"/></td>
        </tr>
    </c:forEach>
</table>
```

than this:

```
<table>
    <%  if ( tableHeaderRequired ) { %>
    <tr>
        <th>Last Name</th>
        <th>First Name</th>
    </tr>
    <% } %>
    <c:forEach var="customer" items="${customers}">
    <tr>
        <td><c:out value="${customer.lastName}"/></td>
        <td><c:out value="${customer.firstName}"/></td>
    </tr>
    </c:forEach>
</table>
```

Comments

Comments are used sparingly to describe additional information or purposes of the surrounding code. Here we look at two types for JSP files: JSP and client-side comments.JSP Comments

JSP comments (also called server-side comments) are visible only on the server side (that is, not propagated to the client side). Pure JSP comments are preferred over JSP comments with scripting language comments, as the former is less dependent on the underlying scripting language, and will be easier to evolve into JSP 2.0-style pages. The following table illustrates this:Line JSP scriptlet with scripting language comment Pure JSP commentsingle

```
<% /**  ...  */ %>
<% /*  ...  */ %>
<% //  ...  %>
```

```
<%--  ...  --%>
```

multiple

```
<%
/*
 *
...
 *
 */
%>
```

```
<%--
  -
 ...
  -
  -- %>
```

```
<%
//
//
```

```
    ...
    //
    %>
```

Client-Side Comments

Client-side comments (<!-- ... -->) can be used to annotate the responses sent to the client with additional information about the responses. They should not contain information about the behavior and internal structure of the server application or the code to generate the responses.

The use of client-side comments is generally discouraged, as a client / user does not need or read these kinds of comments directly in order to interpret the received responses. An exception is for authenticity and legality purposes such as the identification and copyright information as described above. Another exception is for HTML authors to use a small amount of HTML comments to embody the guidelines of the HTML document structures. For example:

<!-- toolbar section -->

    ...

<!-- left-hand side navigation bar -->

    ...

<!-- main body -->

    ...

<!-- footer -->

    ...

Multiline Comment Block

A multiline comment block, be it JSP or client-side, is decorated with the dash character "-". In the XML specification, the double-dash string "--" is not allowed within an XML comment block. Thus, for compatibility and consistency with this specification, no double-dash string is used to decorate comment lines within a multiline comment block. The following table illustrates this preference using a client-side comment block:Preferred Non-XML compliant

```
<!--
  - line 1
  - line 2
...
  -->
```

```
<!--
  -- line 1
  -- line 2
...
  -->
```

JSP Declarations

As per the Java code convention, declarations of variables of the same types should be on separate lines:Not recommended Recommended

```
<%! private int x, y; %>
```

```
<%! private int x; %>
<%! private int y; %>
```

JavaBeans components should not be declared and instantiated using JSP declarations but instead should use the <jsp:useBean> action tag.

In general, JSP declarations for variables are discouraged as they lead to the use of the scripting language to weave business logic and Java code into a JSP page which is designed for presentation purposes, and because of the overhead of managing the scope of the variables.JSP Scriptlets

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 deemphasizes the use of scriptlets). In the following examples, for each data type representation of the customers, a different scriptlet must be written:

customers as an array of Customers

```
<table>
    <%  for ( int i=0; i<customers.length; i++ ) { %>
        <tr>
            <td><%= customers[i].getLastName() %></td>
            <td><%= customers[i].getFirstName() %></td>
        </tr>
    <%  } %>
</table>
```

customers as an Enumeration

```
<table>
    <%  for ( Enumeration e = customers.elements();
            e.hasMoreElements(); ) {
            Customer customer = (Customer)e.nextElement();
    %>
        <tr>
            <td><%= customer.getLastName() %></td>
            <td><%= customer.getFirstName() %></td>
        </tr>
    <%  } %>
</table>
```

However, if a common tag library is used, there is a higher flexibility in using different types of customers. For instance, in the JSP Standard Tag Library, the following segment of JSP code will support both array and Enumeration representations of customers:

```
<table>
    <c:forEach var="customer" items="${customers}">
        <tr>
            <td><c:out value="${customer.lastName}"/></td>
            <td><c:out value="${customer.firstName}"/></td>
        </tr>
    </c:forEach>
</table>
```

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. For example, the following code fetches the names of customers from the database directly and displays them to a client:

<%

```
// NOT RECOMMENDED TO BE DONE AS A SCRIPTLET!


Connection conn = null;
try {
    // Get connection
    InitialContext ctx = new InitialContext();
    DataSource ds = (DataSource)ctx.lookup("customerDS");
    conn = ds.getConnection();


    // Get customer names
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT name FROM customer");


    // Display names
    while ( rs.next() ) {
        out.println( rs.getString("name") + "<br>");
    }
} catch (SQLException e) {
    out.println("Could not retrieve customer names:" + e);
} finally {
    if ( conn != null )
        conn.close();
}
```

%>

The following segment of JSP code is better as it delegates the interaction with the database to the custom tag myTags:dataSource which encapsulates and hides the dependency of the database code in its implementation:

<myTags:dataSource

```
name="customerDS"
```

```
        table="customer"
        columns="name"
        var="result" />
```

<c:forEach var="row" items="${result.rows}">

```
        <c:out value="${row.name}" />
        <br />
```

</c:forEach>

result is a scripting variable introduced by the custom tag myTags:dataSource to hold the result of retrieving the names of the customers from the customer database. The JSP code can also be enhanced to generate different kinds of outputs (HTML, XML, WML) based on client needs dynamically, without impacting the backend code (for the dataSource tag). A better option is to delegate this to a front controller servlet which performs the data retrieval and provide the results to the JSP page through a request-scoped attribute. See the Enterprise section of Java BluePrints for an example.

In summary:

```
      * JSP scriptlets should ideally be non-existent in the JSP page so that the JSP page is
    independent of the scripting language, and business logic implementation within the JSP page is
    avoided.
      * If not possible, use value objects (JavaBeans components) for carrying information to and from
    the server side, and use JSP scriptlets for transforming value objects to client outputs.
      * Use custom tags (tag handlers) whenever available for processing information on the server
    side.
```

JSP Expressions

JSP Expressions should be used just as sparingly as JSP Scriptlets. To illustrate this, let's look as the following three examples which accomplish equivalent tasks:

Example 1 (with explicit Java code):

```
        <%= myBean.getName() %>
```

Example 2 (with JSP tag):

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

Example 3 (with JSTL tag):

```
        <c:out value="${myBean.name}" />
```

Example 1 assumes that a scripting variable called myBean is declared. The other two examples assume that myBean is a scoped attribute that can be found using PageContext.findAttribute(). The second example also assumes that myBean was introduced to the page using <jsp:useBean>.

Of the three examples, the JSTL tag example is preferred. It is almost as short as the JSP expression, it is just as easy to read and easier to maintain, and it does not rely on Java scriptlets (which would require the web developer to be familiar with the language and the API calls). Furthermore, it makes the page easier to evolve into JSP 2.0-style programming, where the equivalent can be accomplished by simply typing ${myBean.name} in template text. Whichever choice is adopted, it should be agreed on amongst web developers and consistent across all produced JSP pages in the same project. It should be noted that the JSTL example is actually slightly different in that it gets the value of myBean from the page context instead of from a local Java scripting variable.

Finally, JSP expressions have preference over equivalent JSP scriptlets which rely on the syntax of the underlying scripting language. For instance,

```
<%= x %>
```

is preferred over

```
<% out.print( x ); %>
```

White Space

White space further enhances indentation by beautifying the JSP code to reduce comprehension and maintenance effort. In particular, blank lines and spaces should be inserted at various locations in a JSP file where necessary.Blank Lines

Blank lines are used sparingly to improve the legibility of <strike>the</strike>a JSP page, provided that they do not produce unwanted effects on the outputs. For the example below, a blank line inserted between two JSP expressions inside an HTML <PRE> block call causes an extra line inserted in the HTML output to be visible in the client's browser. However, if the blank line is not inside a <PRE> block, the effect is not visible in the browser's output.JSP statements HTML output to client

```
<pre>
<%= customer.getFirstName() %>
<%= customer.getLastName() %>
</pre>
```

```
Joe
Block
```

```
<pre>
<%= customer.getFirstName() %>


<%= customer.getLastName() %>
</pre>
```

```
Joe
Block

<%= customer.getFirstName() %>

<%= customer.getLastName() %>



Joe Block
```

Blank Spaces

A white space character (shown as) should be inserted between a JSP tag and its body. For instance, the following

```
<%= customer.getName() %>
```

is preferred over

```
<%=customer.getName()%>
```

There should also be space characters separating JSP comment tags and comments:

<%--

```
- a multi-line comment broken into pieces, each of which
- occupying a single line.
--%>
```

<%-- a short comment --%>

Naming Conventions

Applying naming conventions makes your web component elements easier to identify, classify and coordinate in projects. In this section, we will look at these conventions specific to JSP technology.JSP Names

A JSP (file) name should always begin with a lower-case letter. The name may consist of multiple words, in which case the words are placed immediately adjacent and each word commences with an upper-case letter. A JSP name can be just a simple noun or a short sentence. A verb-only JSP name should be avoided, as it does not convey sufficient information to developers. For example:

```
perform.jsp
```

is not as clear as

```
performLogin.jsp
```

In the case of a verb being part of a JSP name, the present tense form should be used, since an action by way of backend processing is implied:

```
showAccountDetails.jsp
```

is preferred over

```
showingAccountDetails.jsp
```