4-2-2.ディレクティブ

■ ディレクティブ

はじめに

前回JSPファイルの実行方法を解説しましたが、
今回はJSPファイル特有の記述方法「ディレクティブ」について解説します。
ディレクティブとは、簡単に言うと「このページはこういう属性を持っていますよ」と伝えてあげる為の記述です。
HTMLタグと似た記述となっているので、直感的に理解できると思います。

Step1: ディレクティブについて

前回使用したJSPファイルを例に見てみましょう。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
 <h1>Hello World! </h1>
</body>
</html>
```

上記JSPファイルの一番上に、

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

といった`<%@`から始まり`%>`で終わるソースがありますね。
この記述が **ディレクティブ** です。
基本形はこの形式となり、更にディレクティブには複数の種類があります。

- **pageディレクティブ**
- **includeディレクティブ**
- **taglibディレクティブ**

の3つです。
上記の例は`<%@ page`と書かれているので、
このディレクティブは **pageディレクティブ** ということが確認できますね。
では、この3つのディレクティブがどういう特性を持つのか、1つずつ解説していきます。

Step2: pageディレクティブ

pageディレクティブ内へは、主にページの表示に関する設定を記述します。
先ほどのpageディレクティブ

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

を例に、どのような設定が記述されているか見てみましょう。

?

項目名	説明	例
language	JSPで使用する言語の指定	language="java"
contentType	JSPページのデータ型、文字コードの指定	contentType="text/html; charset=UTF-8"
pageEncoding	JSPページの文字コードの指定	pageEncoding="UTF-8"

使用言語がjava・かつHTML形式のデータをUTF-8形式で出力するといった書き方が上記のソースとなります。
もし文字コードをSHIFT-JISに変更したい、といった要件が発生した際は、
contentType="SHIFT-JIS" にすればOKです。

> **補足**

今回のソースでは `contentType="UTF-8"` に加えて、 `pageEncoding="UTF-8"` と指定していますが、
contentTypeで文字コードを指定すればpageEncodingで再度文字コードを指定しなくても大丈夫です。

頻出属性は上記3属性ですが、この他にも様々な属性が指定可能です。
現場の要件に合わせて適切な物を選択しましょう。

項目名	説明	例
import	JSPで使用するクラスの指定	import="java.util.Date"
session	セッションの有効・無効を指定	session="true"
errorPage	エラーが発生した際の遷移先を指定	errorPage="(遷移先ページのパス)"

上記も属性の一例です。
属性は他にも多く存在しますが、 **一度に全てを覚える必要はありません** 。
ディレクティブの仕様自体は覚えておくべきですが、属性は必要となった際に調べれば問題ないでしょう。

includeディレクティブ

includeディレクティブは、JSPファイルから他のJSPファイルを読み込みたい場合に使用します。
先ほど登場したpageディレクティブの指定方法は`<%@ page`でしたが、
includeディレクティブの場合は`<%@ include`と記述します。

```
<%@ include file="(読み込みたいJSPファイルのパス)" %>
```

上記がincludeディレクティブの構文です。
このディレクティブは、
全JSPファイルで記述している内容を別JSPファイル(1とします)として切り出し、
切り出した1を全JSPファイルから読み込む(include)よう共通化したい
といった場合に使用します。

具体例を見てみましょう。

【include.jsp】

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.*,java.text.SimpleDateFormat"%>
  <div>
   <% Date date = new Date();
      SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
      String formatDate = sdf.format(date);%>
   <%= formatDate %>
  </div>
```

上記サンプルjspファイルを、異なる2つのjspファイルから読み込んでみます。

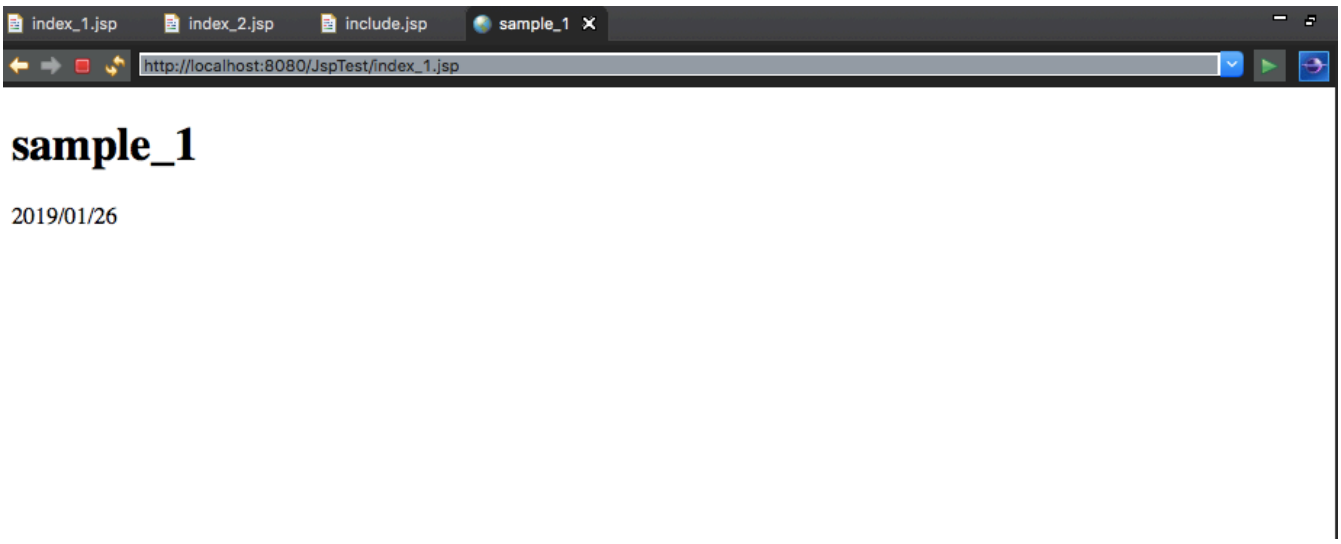
【index_1.jsp】

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>sample_1</title>
</head>
<body>
<h1>sample_1</h1>
 <%@ include file="include.jsp"%>
</body>
</html>
```

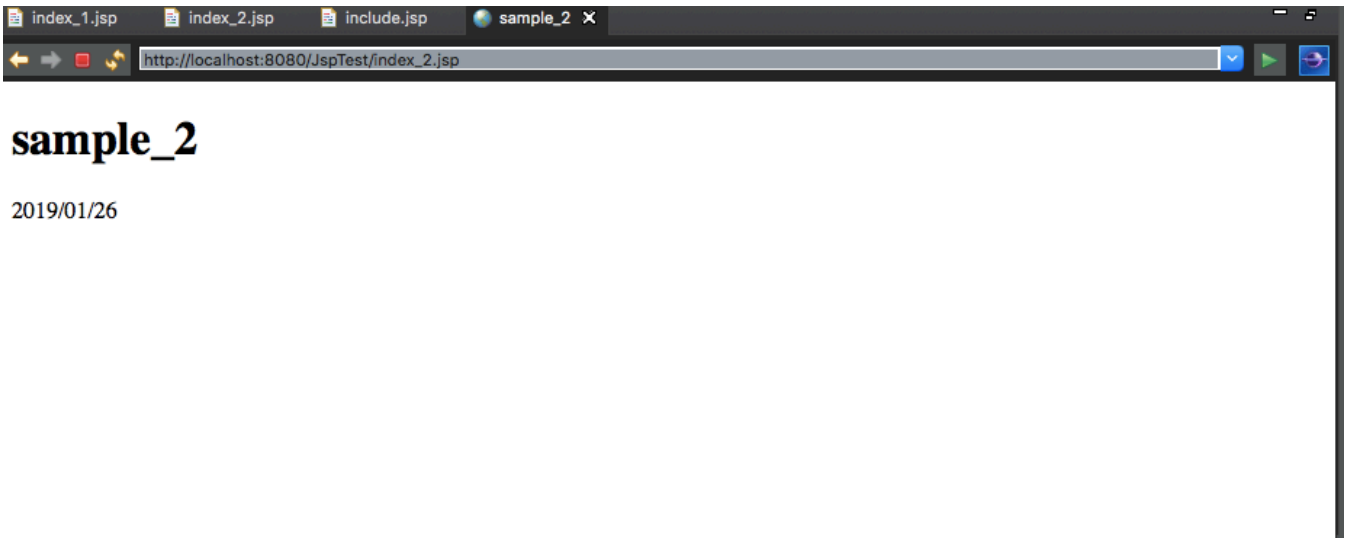
【index_2.jsp】

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>sample_2</title>
</head>
<body>
<h1>sample_2</h1>
 <%@ include file="include.jsp"%>
</body>
</html>
```

【実行結果 index_1.jsp】



【実行結果 index_2.jsp】



いずれも同じ現在年月日が取得されていますね。
このように、includeを使用すれば資材を使いまわすことが可能となり、ソースが読みやすくなります。
現場では **ヘッダーやフッターを共通化する際に使用することも多い** ため、includeの挙動を理解しておきましょう。

Step3: taglibディレクティブ

最後に解説するのがtaglibディレクティブです。
定期的に使用する処理をタグに定義し、このディレクティブで呼び出して使用するといった使い方をします。

```
<%@ taglib uri="(タグライブラリのURI)" prefix="(カスタムタグの接頭辞)" %>
```

上記がtaglibディレクティブの構文です。
構文についてざっくり説明すると、

- **タグライブラリのURI→使用したいタグの情報が記載されている設定ファイル**
- **カスタムタグの接頭辞→JSPからタグを呼び出す際に使用する名称**

といった具合です。
タグの使用にはいくつか手順を踏むので、どのように使用するか例を見てみましょう。

1：タグハンドラクラスの作成
まず、タグを呼び出した際に行いたい処理をクラスに記述します。
今回は「hello world」を表示するサンプルを作成します。

> **1：「hello world」と表示するためのクラスを作成**

サンプルコード(helloTagServlet.java)↓

```
package tags;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.Tag;

public class helloTagServlet implements Tag {
        private PageContext pageContext;
        private Tag parentTag;

        public void setPageContext(PageContext pageContext) {
                this.pageContext = pageContext;
        }

        public void setParent(Tag parentTag) {
                this.parentTag = parentTag;
        }

        public Tag getParent() {
                return this.parentTag;
        }

        public int doStartTag() throws JspException {
                try {
                        JspWriter out = pageContext.getOut();
                        out.print("hello world");
                } catch (Exception e) {
                        throw new JspException(e.getMessage());
                }
                return SKIP_BODY;
        }

        public int doEndTag() throws JspException {
                return EVAL_PAGE;
        }

        public void release() {
        }
}
```

見慣れない構文が多く登場しますが、殆どは定型文(いわゆるおまじないですね)なので深く考えなくて大丈夫です。
ここで着目すべき記述は以下の2点です。

- `doStartTag()`
- `doEndTag()`

`doStartTag()`内へは、JSPからtaglibディレクトリを指定した際に行いたい処理を記述します。
今回はタグを呼び出した際に「hello world」と表示させるため、 `try{}` 内が上記の記述となっています。
その後に続く`doEndTag()`についてですが、このメソッドに内へは`doStartTag()`内の処理が終了した後に行われる処理を記述します。

`return EVAL_PAGE;`は **JSPページにコントローラーで処理を行った結果を返す** といった意味合いを持っており、
今回の場合は「画面にhello worldと表示するよ」という結果を返します。
これも構文なので、処理の流れを把握していれば深く考えなくて大丈夫です。

2：tld　ファイルの作成

tldファイルとは、上記で作成したクラスとJSPファイルを紐づけるための設定ファイルです。

サンプルコード↓(/WEB-INF/lib/hello.tld)

```
<?xml version="1.0" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
            web-jsptaglibrary_2_0.xsd"
        version="2.0">

    <tlib-version>1.0</tlib-version>
    <jsp-version>2.0</jsp-version>
    <short-name>helloTag</short-name>
    <tag>
        <name>helloTag</name>
        <tag-class>tags.helloTagServlet</tag-class>
        <body-content>empty</body-content>
    </tag>
</taglib>
```

<name>タグへ **JSPファイルのtaglibディレクトリから呼び出すための名称** を指定し、
<tag-class>タグへは **上記で作成したタグハンドラクラス** を指定します。

3：web.xmlの作成

web.xmlは、作成したtldファイルとJSPファイルを紐づけるためのファイルです。

サンプルコード↓(/WEB-INF/web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
            web-app_2_4.xsd"
        version="2.4">
    <jsp-config>
        <taglib>
            <taglib-uri>http://www.javaroad.jp/tags/hello</taglib-uri>;
            <taglib-location>/WEB-INF/lib/hello.tld</taglib-location>
        </taglib>
    </jsp-config>
</web-app>
```

<taglib-uri>タグへ **JSPのtaglibディレクトリ**<%@ taglib uri="ooo">**へ指定するuri**を記述し、
<taglib-location>タグへは **tldファイルの配置パス** を指定します。

4：JSPファイルの作成

JSPファイル内へ下記

```
<%@ taglib uri="(タグライブラリのURI)" prefix="(カスタムタグの接頭辞)" %>
```

を指定することで、画面へ「hello world」と表示させます。

サンプルコード↓(hello.jsp)

```
<%@ page contentType="text/html; charset=windows-31j" import="tags.*" %>
<%@ taglib uri="http://www.javaroad.jp/tags/hello" prefix="hello" %>
<html>
<body>
    <hello:helloTag/>
</body>
</html>
```

上記taglibディレクトリ内の **uri=**〇〇〇 へ、xmlファイルで指定した

```
<taglib-uri>http://www.javaroad.jp/tags/hello</taglib-uri>;
```

のタグ内と同じuriを記述し、 **prefix** へはJSPファイル内でタグを使用するための名称(何でも大丈夫です)を指定します。
実際に、指定したprefix **hello** がbody内で

```
<hello:helloTag/>
```

と使用されていることが確認できますね。
hello:の後にtldファイルで指定した<name>タグと同じ値

```
 <name>helloTag</name>
```

を指定することで、タグハンドラクラスへ記述した「hello world」と出力する処理が実行されます。

まとめ

pageディレクティブ・includeディレクティブは記述自体もシンプルであり、
実際に現場で目にする機会も多いでしょう。
ですがtaglibディレクティブに関してはファイル定義がやや複雑なうえ、自身でカスタムタグを
定義する機会はそこまで多くありません。
現時点で事細かに定義方法を覚える必要はありませんが、現場で作成・改修を行うことになった際、
taglibディレクトリについて調べた内容を把握できるようにしておくと良いです。

課題

提出課題はありませんので、一通り学習が終わったら次の章に進んで下さい。

最終更新日時: 2025年 01月 19日(日曜日) 18:34