Java EE

Servlet/JSP編

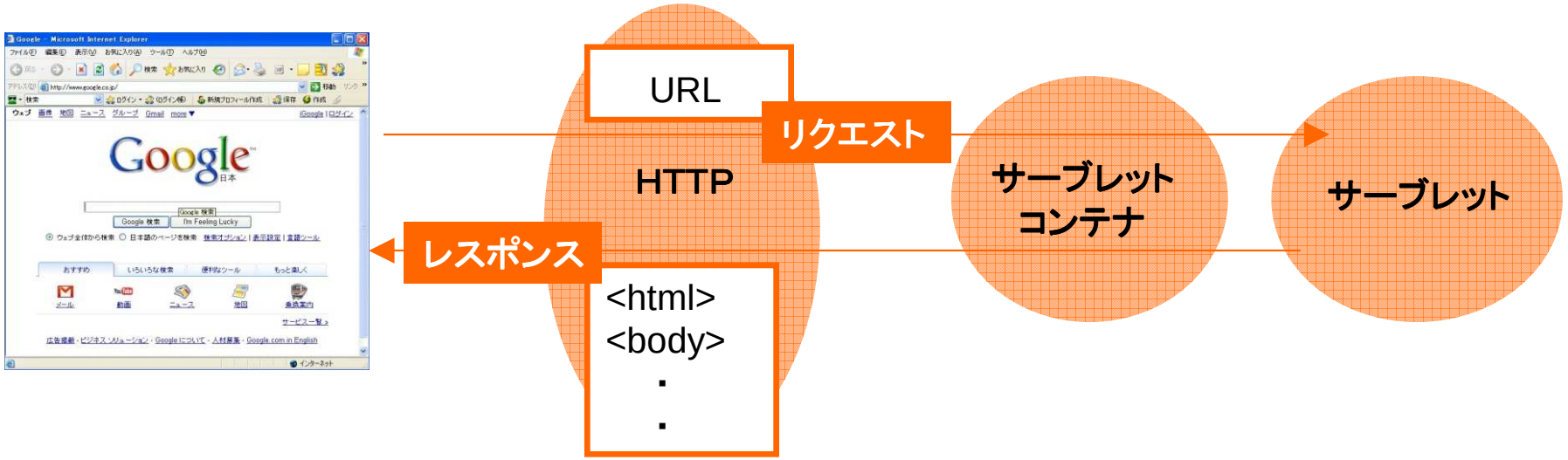
TC. Techno-core Corporation 株式会社テクノコア
http://www.techno-core.jp

サーブレットの仕組みを理解する

サーブレットの仕組み

サーブレットと同じようにクラスを定義してプログラムを作っていくもので、Javaアプリケーションがあります。Javaアプリケーションはクライアント上のJavaVM上でmain()メソッドから実行されます。それに対して、サーブレットはサーバ側のサーブレットのための実行環境(これをAPサーバやサーブレットコンテナと呼びます)上で実行され、doGet()やdoPost()といった名前のメソッドから実行が開始されます。

サーブレットの実行



サーブレットはクライアントからの「リクエスト」を受け取り、処理を行った後「レスポンス」を返すのですが、サーバ／クライアント間の情報のやり取りは、HTTPというプロトコルの上で行われます。処理の開始点となる「リクエスト」は、Webブラウザなどのクライアントから送信されます。リクエストを受け取ったサーブレットは、データベースへの登録など必要な処理を行い、その後「レスポンス」を返します。「レスポンス」はサーバからクライアントに向けて、HTMLなどのデータを含んだ形で送信されます。

Tomcatの設定

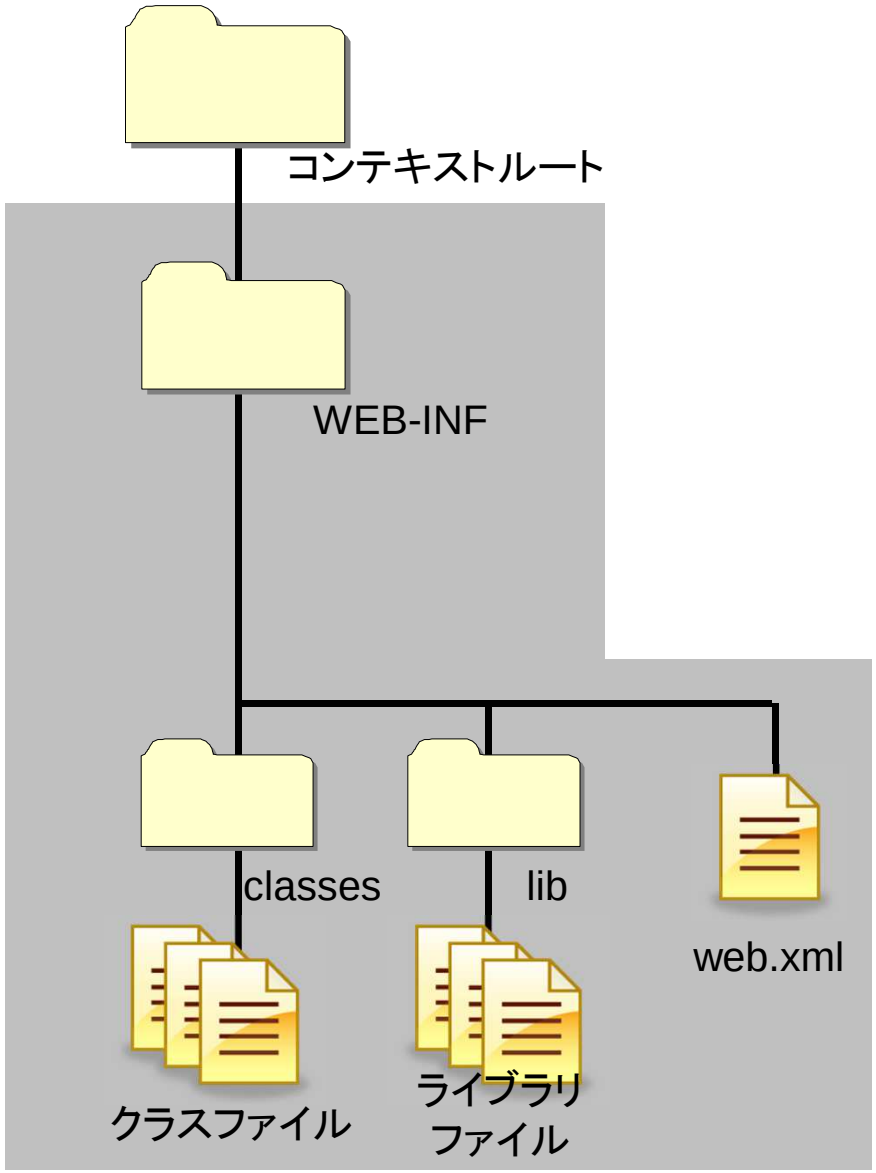
C:¥xampp¥tomcat¥conf¥server.xml を修正します。
（文字化け対策の一環）

```
66          Java AJP  Connector: /docs/config/ajp.html
67          APR (HTTP/AJP) Connector: /docs/apr.html
68          Define a non-SSL HTTP/1.1 Connector on port 8080
69      -->
70      <Connector port="8080" protocol="HTTP/1.1"
71                 connectionTimeout="20000"
72                 redirectPort="8443" 
73                 useBodyEncodingForURI="true" />
74      <!-- A "Connector" using the shared thread pool-->
75      <!--
76      <Connector executor="tomcatThreadPool"
77                 port="8080" protocol="HTTP/1.1"
78                 connectionTimeout="20000"
```

C:¥xampp¥tomcat¥conf¥web.xml を修正します。
（ディレクトリアクセス時にファイル一覧を表示させる）

```
92    <!--                       to be a physical file. [null]                -->
93    <!--                                                                    -->
94    <!--                                                                    -->
95
96      <servlet>
97          <servlet-name>default</servlet-name>
98          <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
99          <init-param>
100             <param-name>debug</param-name>
101             <param-value>0</param-value>
102         </init-param>
103         <init-param>
104             <param-name>listings</param-name>
105             <param-value>true</param-value>
106         </init-param>
107         <load-on-startup>1</load-on-startup>
108     </servlet>
109
110
111   <!-- The JSP page compiler and execution servlet, which is the mechanism  -->
112   <!-- used by Tomcat to support JSP pages.  Traditionally, this servlet    -->
113   <!-- is mapped to the URL pattern "*.jsp".  This servlet supports the     -->
```

初めてのサーブレットを作る



Webアプリケーションの構成

1つのWebアプリケーションは、サーブレットやJSPの他にも、HTMLファイルやgifファイルなどの画像ファイル、音声ファイルなど様々なファイルから構成されます。Java EEでは、これらのファイルの圧縮方法や配置場所などに、いくつかのルールを設けています。そして、その通りに配置しないと動作しません。

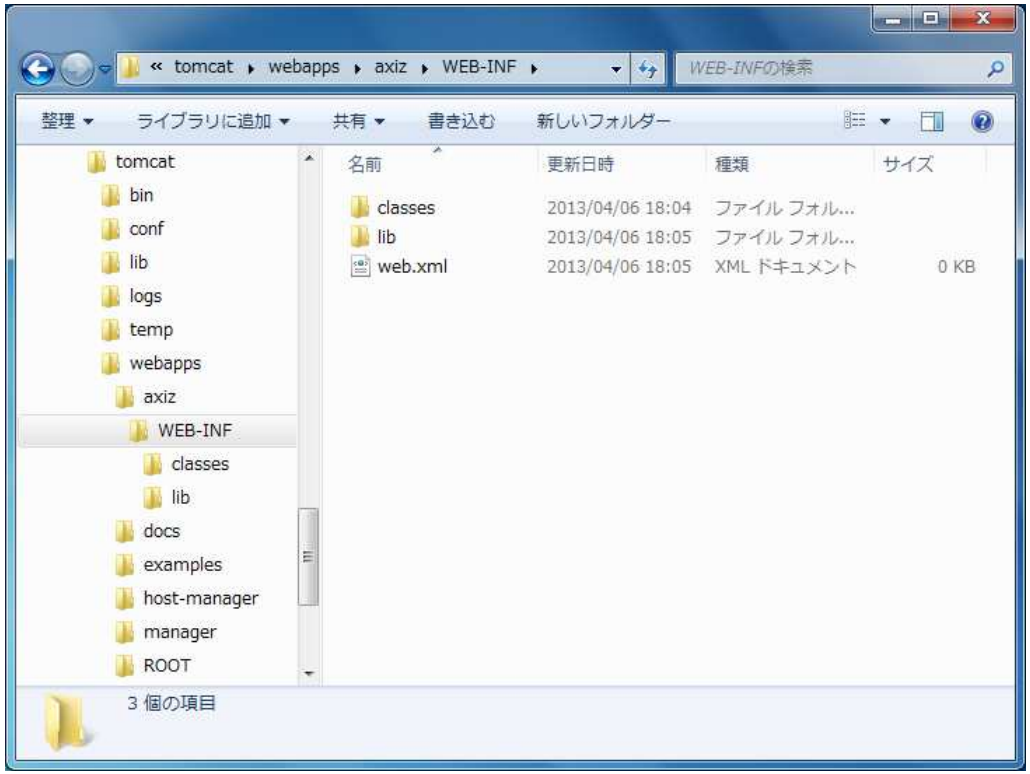
まず、1つのWebアプリケーションに含まれるファイルは、1つのフォルダの中に配置します。このフォルダのことを**Webアプリケーションルートまたはコンテキストルート**と呼び、このフォルダ以下をまとめて**Webアプリケーションまたはコンテキスト**と呼びます。例えば、ECサイトとグループウェアといった2種類のWebアプリケーションを配置した場合は、コンテキストを2つ用意することになります。

Webアプリケーションは、左の図のような階層で構成されます。灰色の部分はJava EEの仕様で決まっており、必ず守らなければなりません。

初めてのサーブレットを作る

コンテキストの作成

まずは、コンテキストを作成します。コンテキストとは簡単に言えばファイルの保存場所、つまりはフォルダのことです。
Tomcatのフォルダの中のwebappsの下(ROOTと同じ階層)にフォルダを作ります。今回は『axiz』にします。
その下の階層には必ず、【WEB-INF】が必要で、さらにその下には【classes】と【lib】というフォルダが必ず必要です。もうひとつ【web.xml】というファイルも必要です。
これらは、コンテキストの作成には必ず必要なものです。名前を変えていいのはコンテキストルート、つまり、『axiz』だけです。ここはなんと言う名前でもいいですし、その下は必ずこのような階層になっています。



初めてのサーブレットを作る

web.xmlの作成

次に、先ほど作成したコンテキストの中にある『web.xml』ファイルの編集を行います。
このweb.xmlは「Webアプリケーション　デプロイメント記述子」または単に「デプロイメント記述子」と呼ばれます。
下記のように記述してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                      http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0">

</web-app>
```

これで、Tomcatが起動します。試しに何かのテストファイルを【WEB-INF】と同じ階層に配置して、
Tomcatを起動してインターネットブラウザで動作確認してみてください。
例えばtest.htmlというファイルを配置した場合は以下のようなURLにアクセスをします。

http://localhost:8080/axiz/test.html

初めてのサーブレットを作る

HelloWorldServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        // 出力
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("Hello World");
        out.println("</body>");
        out.println("</html>");
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```

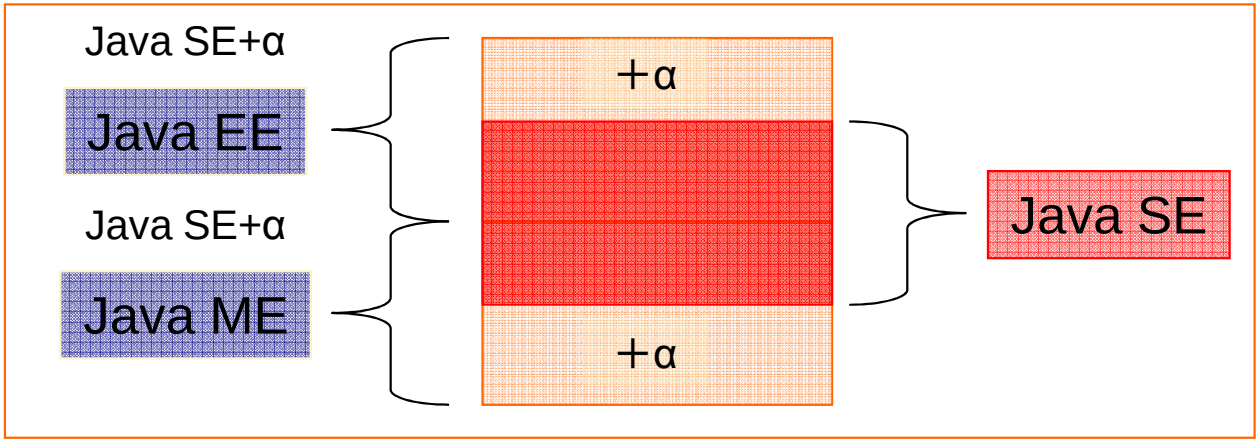
初めてのサーブレットを作る

サーブレットの作成

これまでに、コンテキストの作成、web.xmlの作成が終了しました。次は実際にサーブレットを作成して、動作確認を行います。

まずは、HelloWorldServlet.javaをコンパイルします。そうするとコンパイルエラーが出ますが、これは、皆さんが使えるクラスライブラリが足りないからエラーが表示されているのです。これはデータベースのJDBCに似ています。

さて、これまではJavaのJava SEで開発を行ってきました。下図のように≪＋α≫の機能が必要になります。



この≪＋α≫の機能というのは、実はTomcatインストール時に追加されていて、C:¥xampp¥tomcat¥libの中にある、『servlet-api.jar』というファイルがあることを確認してください。
この『servlet-api.jar』が≪＋α≫に当たり、クラスパスに追加する必要があります。
(なお、Java MEはJava SEの機能の一部しか使用できないため完全な上位エディションではありません)

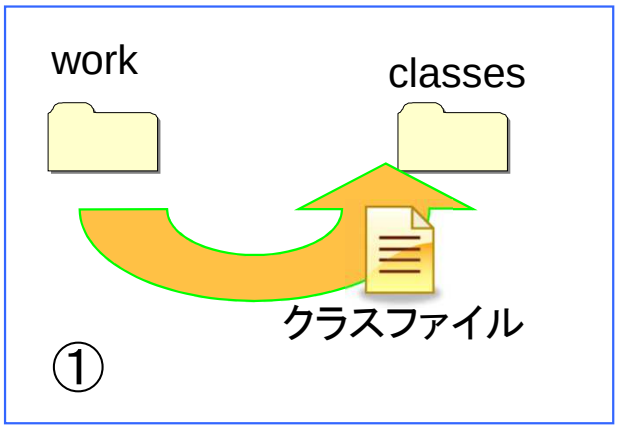
初めてのサーブレットを作る

サーブレットの実行

コンパイルが無事できたら、次は実行してみて下さい。エラーが表示されますね。エラーの内容を読み取ってみると、「mainがありません」と出ています。これは今まで、皆さんがコマンドプロンプト上で動作するJavaのプログラミングをしてきました。今まではVMがmainメソッドを呼び出して、プログラムが実行できていましたが、今回はブラウザがトリガーの役目をしています。つまり、ブラウザからの呼び出し専用のクラスになっています。

実行させるには以下の手順が必要です。

1. コンパイル後のクラスファイルをコンテキストの【classes】に移動する
2. 『web.xml』にサーブレットの登録をする
3. Tomcatの再起動をして、任意のURLにブラウザからアクセスする




```
<servlet>
  <servlet-name>helloWorld</servlet-name>
  <servlet-class>HelloWorldServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>helloWorld</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

紐付け（同じ名前）

クラスの名前(FQN)

②

URLのパターン

③ http://localhost:8080/axiz/hello にアクセス

コンテキストパス

初めてのJSPプログラムを作る

JSPの作成

実は、サーブレットはあまり画面を表示するのが得意ではありません。画面を表示させる（HTMLの生成）には、JSPというものの方が得意です。

では実際にJSPを作ってみましょう。作る場所はコンテキストの【WEB-INF】と同じ階層に配置します(つまりHTMLと同じように配置します)。下記内容を入力してブラウザで確認してみましょう(UTF-8で保存してください)。

hello_jsp_world.jsp

```
<%@page contentType="text/html; charset=UTF-8" %>
<%@page import="java.util.Date" %>
<%
Date time = new Date();
%>
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Hello JSP World!!</title>
</head>
<body>
JSPの世界の皆さんこんにちは！<br>
ただいまの時刻は<%= time %> です。<br>
</body>
</html>
```

確認には下記アドレスを入力します。
http://localhost:8080/axiz/hello_jsp_world.jsp

暗黙オブジェクト

オブジェクト名	説明	備考
pageContext	JSPのオブジェクトを管理する	ページスコープ
request	クライアントからのリクエスト情報を取得する	リクエストスコープ
session	セッション情報を管理する	セッションスコープ
application	アプリケーション情報を管理する	アプリケーションスコープ
response	クライアントへのレスポンス情報を設定する	
out	クライアントへ出力する	
page	JSPページ自身を表す	this
config	JSPページのパラメータを設定する	
exception	例外発生時のエラー情報を取得する	isErrorPage=″true″ の指定が必要

Tips

フォームから送られたデータの文字化け解消

① パラメータごとにエンコーディング(古い方法)

```
String name = request.getParameter("name");
if (name != null) {
    name = new String(name.getBytes("ISO-8859-1"), "UTF-8");
}
```

② 一括エンコーディング(新しい方法)

```
request.setCharacterEncoding("UTF-8");
```

①の方法はTomcat4以前で使われていた方法です。もちろんTomcat4以降でも使える方法ですが、パラメータごとに変換作業が必要なため非常に煩雑になります。Tomcat4以降では②の方法が一般的です。注意点としてはgetParameterしてしまった値は文字化けしたままになるので、**必ずgetParameterよりも先に**setCharacterEncodingメソッドを呼ぶ必要があります。

Tips

EL式

JSP2.0 (Tomcat5.x以降)からJSP内でEL式(Expression Language)が使用できるようになりました。
EL式は ${～} という書式になっており、これを使うことでJSP内でのオブジェクトアクセスのコードが非常に簡略化でき、見通しが良くなります。
下記はEL式のほんの一例です。

従来の方法

```
<%= request.getAttribute("name") %>
```

EL式

```
${ requestScope["name"] }
または
${ requestScope.name }
または
${ name }
```

従来の方法

```
<%
String sNum = request.getParameter("num");
Int num = Integer.parseInt(sNum);
%>
<%= num + 10 %>
```

EL式

```
${ param["num"] + 10 }
または
${ param.num + 10 }
```

Tips

JSTL(JavaServer Pages Standard Tag Library)

JSTLはJSPでよく利用される標準的なカスタムタグをまとめたものです。従来スクリプトレットに書いていた条件分岐やループなどをタグを使って表現することが可能になります。JSTLにもバージョンがあり、下記はJSTLとJSP、Tomcatのバージョン対応表です。またJSTLは単体で使用するよりもEL式と組み合わせて使うことでより一層効果が表れます(ループと配列のようなもんでしょうか)。実際にJSPで使用するためにはjstl.jar、standard.jarをlibフォルダに配置する必要があります。(Tomcatインストール時にexampleもインストールしていれば、webapps¥examples¥WEB-INF¥lib の中にあります)

Servlet/JSP Spec	Apache Tomcat Version	JSTL/Standard Version	Minimum Java Version
3.0/2.2	7.0.x	1.1/1.2	6
2.5/2.1	6.0.x	1.1/1.2	5.0
2.4/2.0	5.5.x	1.1	1.4
2.3/1.2	4.1.x	1.0	1.3
2.2/1.1	3.3.x	-	1.2

Tips

JSTL(JavaServer Pages Standard Tag Library)

JSPでJSTLを使用する場合は<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
のようにtaglibディレクティブでJSTLのURIとPrefixを宣言する必要があります。以下にまとめます。

種類	Prefix	JSTL1.1/1.2のURI	JSTL1.0のURI
core	c	http://java.sun.com/jsp/jstl/core	http://java.sun.com/jstl/core
i18n	fmt	http://java.sun.com/jsp/jstl/fmt	http://java.sun.com/jstl/fmt
xml	x	http://java.sun.com/jsp/jstl/xml	http://java.sun.com/jstl/xml
function	fn	http://java.sun.com/jsp/jstl/functions	-
database	sql	http://java.sun.com/jsp/jstl/sql	http://java.sun.com/jstl/sql

従来の方法

```
<% String name = "AxiZ"; %>
<%= name %>
```

JSTL + EL式

```
<c:set var="name" value="AxiZ" />
${ name }
```

従来の方法

```
<% int num =
      Integer.parseInt(request.getParameter("num")); %>
<% if (num < 10) { %>
   値は10未満です。
<% } else { %>
   値は10以上です。
<% } %>
```

JSTL + EL式

```
<c:set var="num" value="${ param.num }" />
<c:choose>
   <c:when test="${ num < 10 }">値は10未満です。</c:when>
   <c:otherwise>値は10以上です。</c:otherwise>
</c:choose>
```