付録 A　JSPの作成

A.1　JSPとは

Servletでは、Javaのプログラムの中に文字列定数などの形でHTMLが埋め込まれているのに対し、JSPではHTMLの中にJavaのプログラムが埋め込まれるような形になっている。(ここでは、話を簡単にするために出力形式をHTMLと仮定しているが、他のテキスト形式を出力するJSPも可能である。)

JSPは、自動的にServletに変換されて実行されるので、できることはServletと違いはない。出力すべきHTMLが多くて、プログラム部分が少ない場合は、JSPの方が便利というだけのことである。

ファイル MyDate.jsp

```
<%@ page contentType="text/html; charset=Windows-31J"
         import="java.util.Calendar" %>
<%!  String[] youbi = {"日", "月", "火", "水", "木", "金", "土"}; %>
<%   Calendar cal = Calendar.getInstance(); %>
<html><head></head><body>
  <%= cal.get(Calendar.YEAR) %>年
  <%= cal.get(Calendar.MONTH)+1 %>月
  <%= cal.get(Calendar.DAY_OF_MONTH) %>日
  <%= youbi[cal.get(Calendar.DAY_OF_WEEK)] %>曜日
  <%= cal.get(Calendar.HOUR_OF_DAY) %>時
  <%= cal.get(Calendar.MINUTE) %>分
  <%= cal.get(Calendar.SECOND)) %>秒
</body></html>
```

この例では、Servletの MyDate.javaとほぼ同じ（空白の入り方だけが違う）ページを出力する。

A.2　JSPの主なタグ

ここではJSPのよく使われるタグをいくつか紹介する。

ページディレクティブ: <%@ page 〜 %>　<%@ pageと%>の間にはページディレクティブというJSPページ全体に影響する属性を指定する。

まず、contentTypeという属性を指定するのが一般的である。これは、文字通り出力する内容のタイプである。（静的なコンテンツなら.html, .txtのようなファイルの拡張子でcontentTypeを推測することが可能だが、ServletやJSPはこのように明示的に指定する必要がある。）

また、importという属性でインポートするJavaのクラスあるいは、java.util.*のような形式でパッケージを指定する。クラスやパッケージが複数ある場合はコンマで区切って並べる。

スクリプトレット : <% 〜 %>　<% と %>の間には、文や変数の宣言など、Java のメソッドの中に書くことができる任意のコードの断片を書くことができる。

　ここに書くものは文法的に完全な文である必要はない。例えば次のような書き方も可能である。

ファイル ScriptletTest.jsp

```
<%@ page language="java" contentType="text/html; charset=windows-31j" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>スクリプトレットテスト</title>
</head>
<body>
<h1>スクリプトレットテスト</h1>
<%
   double r = Math.random();
   if ( r < 0.5 ) {
%>
      <span style='color:red'>小吉</span>
<% } else { %>
      <span style='color:green'>大吉</span>
<% } %>
</body>
</html>
```

また、ここで宣言された変数は、Servlet のメソッド内の局所変数に翻訳される。つまり JSP ページが表示されるたびに初期化される。

式: <%= 〜 %>　<%=と%>の間には Java の式を書くことができる。String 型に変換されて出力中に挿入される。

宣言: <%! 〜 %>　<%! と %>の間には変数やメソッドの宣言を書くことができる。特に、ここで宣言された変数は Servlet のインスタンス変数（フィールド）に翻訳される。つまり、Tomcat などの Servlet コンテナが起動している間は（timeout しない限り）ずっと値を保持し続ける変数になる。

コメント : <%-- 〜 --%>　<%--と--%>の間はコメントである。この間の文字は出力に現れない。

ファイル CommentTest.jsp

```
<%@ contentType="text/html; charset=windows-31j" %>
<html>
<head><title>コメントのテスト</title></head>
<body>
<h2>コメントのテスト</h2>
<%-- このコメントは出力に含まれません。 --%>
<!-- このコメントはどうでしょう。 -->
</body>
</html>
```

インクルードディレクティブ : <%@ include 〜 %>　<%@ include file = "〜" %>という形で他のファイルを読み込む。file 属性の値（〜の部分）は相対 URL である。

A.3　暗黙オブジェクト

式やスクリプトレットを簡単にするために、8 つの変数が前もって定義されている。そのうち、よく使われると思われる 6 つを次の表で紹介する。

変数名	クラス	備考（Servlet との対応）
request	HttpServletRequest	doGet や doPost の第 1 引数に相当
response	HttpServletResponse	doGet や doPost の第 2 引数に相当
out	PrintWriter	response.getWriter() で得られるオブジェクトに相当
session	HttpSession	request.getSession() で得られるオブジェクトに相当
config	ServletConfig	getServletConfig() で得られるオブジェクトに相当
application	ServletContext	getServletContext() で得られるオブジェクトに相当

この中のクラスや備考は厳密な説明ではない。例えば out は本当は JspWriter というクラスのオブジェクトで、PrintWriter とは若干の違いがある。ほとんどの場合 PrintWriter クラスに相当すると考えて良い、ということである。

A.4　JavaBeans とは

JavaBeans とは、いくつかの簡単な規約に従う Java のクラスのことである。規約とは例えば次のようなものである。

- クラスは引数なしのコンストラクタを持たなければならない。
- set〜、get〜など、簡単な命名規則に従った public なメソッドを持つ。例えば、foo というプロパティをセットするメソッドは setFoo であり, foo プロパティを読み出すメソッドは getFoo である。（set, get メソッドでは、プロパティ名の最初の文字を大文字にする。）
- java.io.Serializable インタフェースを implements している必要がある。このインタフェースはメソッドのないインタフェースなので、何か特別なメソッドを実装する必要はない。詳しくは java.io.Serializable のドキュメントを参照すること。

次に簡単な、Bean の例を挙げる。

ファイル TestBean.java

```
package beans;

import java.io.Serializable;

public class TestBean implements Serializable {
  private int foo, bar, baz;

  public int getFoo() {
    return foo;
  }
  public void setFoo(int foo) {
    this.foo = foo;
  }

  public int getBar() {
    return bar;
  }
  public void setBar(int bar) {
    this.bar = bar;
  }

  public int getBaz() {
    return baz;
  }
  public void setBaz(int baz) {
    this.baz = baz;
  }

  public int getAns() {
    return (foo+bar)*baz;
  }
}
```

この Bean は、foo, bar, baz という 3 つの読み書き可能なプロパティと、ans という読み出し専用のプロパティを持っている。

Bean は他のクラスで利用されるクラスなので、ちゃんとパッケージ内に作成するほうが良い。この例では package 文を最初に挿入して、beans パッケージのクラスとして宣言している。パッケージはいくつかのクラスやインタフェースをまとめる仕組みである。パッケージにはクラス名やインタフェース名の衝突を避けるという大切な役割がある。

A.5 Servlet と JSP の連携

Bean を介して、Servlet と JSP が連携することができる。例えば、ユーザーがフォームに入力した値のチェックやデータベースのアクセスなどは Servlet 内で行って、表示を JSP に任せる、という書き方ができる。

以下に非常に単純な連携をする Servlet と JSP を挙げる。FooBarBaz.html のフォームにユーザーが入力し、それを FooBarServlet.java が受け取り Bean にデータをまとめる。それを Ans.jsp に渡して表示している。

コードの詳細は、次の節以降で説明する。

ファイル FooBarBaz.html

```
<html><head><title>簡単な Form</title></head>
<body>
<form action='FooBarServlet' method='post'>
数値を入力してください。<br>
その 1<input type='text' size='10' name='param1'><br/>
その 2<input type='text' size='10' name='param2'><br/>
その 3<input type='text' size='10' name='param3'><br/>
<input type='submit' value=' 送信'>
</form>
</body>
</html>
```

ファイル FooBarServlet.java

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import beans.TestBean;

public class FooBarServlet extends HttpServlet {
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
        TestBean bean = new TestBean();
        // 本来はここでエラーのチェックやデータベースの処理などを行なう
        bean.setFoo(Integer.parseInt(request.getParameter("param1")));
        bean.setBar(Integer.parseInt(request.getParameter("param2")));
        bean.setBaz(Integer.parseInt(request.getParameter("param3")));
        request.setAttribute("myBean", bean);

        getServletContext().getRequestDispatcher("/Ans.jsp")
                           .forward(request, response);
    }
}
```

ファイル Ans.jsp

```
<%@ page contentType="text/html; charset=windows-31j" %>
<%-- FooBarServlet でセットした Bean を取り出す --%>
<jsp:useBean id="myBean" scope="request" class="beans.TestBean" />
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-31j">
<title>計算結果</title>
</head>
<body>
<h1>計算結果</h1>
<table border='1'>
<tr><th>属性</th><th>値</th></tr>
<tr><td>Foo</td><td><jsp:getProperty name="myBean" property="foo" /></td></tr>
<tr><td>Bar</td><td><jsp:getProperty name="myBean" property="bar" /></td></tr>
<tr><td>Baz</td><td><jsp:getProperty name="myBean" property="baz" /></td></tr>
<tr><td>Ans</td><td><jsp:getProperty name="myBean" property="ans" /></td></tr>
</table>
</body>
</html>
```

A.6 Bean 関連のタグ

Java のサーバーアプリケーションでは、Servlet や JSP の間でデータを交換するために Bean を用いることが多い。また、JSP には簡便に Bean を扱うためのタグが用意されている。ここではそのような Bean 関連のタグを紹介する。

<jsp:useBean〜/> jsp:useBean タグは、まずは指定された属性を持つ、すでに存在している Bean を見つけようとする。もし、存在しなければ新しく生成する。

よく使われる属性として、次のものがある。

属性	説明
id	Bean の名前である。あとで参照する時に使用する。
scope	Bean の存在するスコープ（後述）である。
class	Bean をこのクラスのインスタンスとして生成する。
type	class 属性と同時に用いて、Bean にこの型を割り当てる。

良く使われるかたちは、

```
<jsp:useBean id="beanName" scope="scope" class="className" />
```

である。

スコープとは jsp:useBean の属性として指定するスコープ (scope) には、次の 4 種類がある。スコープは有効範囲という意味である。

スコープ	説明
page	このタグが使用されている JSP ページ内とそこから静的にインクルードされるページのみで有効
request	フォワード（後述）やインクルードなどにより、同じ request を処理している JSP ページや Servlet 内で有効（Servlet では、request.getAttribute(*beanName*) でこの Bean にアクセスできる。）
session	セッション内で有効（Servletでは、request.getSession().getAttribute(*beanName*) でこの Bean にアクセスできる。）
application	アプリケーション全体で有効（つまり、Tomcat などの Web アプリケーションコンテナのひとつのアプリケーションルートの下に置かれたすべての JSP/Servlet から共有される。Servlet では、getServletContext().getAttribute(*beanName*) でこの Bean にアクセスできる。）

<jsp:getProperty〜/> 以下のかたちで用いる。

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

name 属性は、Bean を作成したときの id 属性に対応する。*beanName* という名前の Bean の、*propertyName* という名前のプロパティを取り出す。

<jsp:setProperty〜/> 以下のかたちで用いる。

```
<jsp:setProperty name="beanName" property="propertyName" value="someString" />
```

または

```
<jsp:setProperty name="beanName" property="propertyName"
                                 value="<%= expression %>" />
```

name 属性は、Bean を作成したときの id 属性に対応する。*beanName* という名前の Bean の、*propertyName* という名前のプロパティを value 属性で指定された値に設定する。

特別な場合として、value 属性を省略することができる。

```
<jsp:setProperty name="beanName" property="propertyName" />
```

この場合は、request.getParameter(*propertyName*) の値が value 属性に指定されたのと同じことになる。

さらに、次のようなかたちで property 属性に*を使用できる。

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

この場合は、request 中のすべての Parameter が Bean のプロパティとして設定される。

A.7 フォワードとリダイレクト

フォワードとは、Servlet（あるいは JSP）から、別の Servlet や JSP にリクエストを転送する処理である。これにより Servlet がリクエストの前処理を行って、他の Servlet や JSP が表示を行うなどの分業が可能になる。

Servlet から他の JSP や Servlet にフォワードするには、まず ServletContext から getRequestDispatcher というメソッドの引数にフォワード先のパスを指定して RequestDispatcher オブジェクトを取り出

す。このパスは /から始まらなくてはならず、現在の Web アプリケーションのコンテキストルートからの相対パスと解釈される。次に RequestDispatcher オブジェクトに対して Request と Response を引数として forwad メソッドを呼び出す。

リダイレクトは、フォワードと似ているが、いったんブラウザ側に転送先の URL を送り、ブラウザがこの URL に改めてリクエストを送る、という違いがある。HttpServletResponse クラスの sendRedirect というメソッドを用いる。sendRedirect の引数は、転送先の URL である。

```
response.sendRedirect(URLString);
```

なお、詳しく説明しないが、同じ HttpServletResponse クラスの encodeRedirectURL メソッドと併用する場合もある。

```
response.sendRedirect(response.encodeRedirectURL(URLString));
```

フォワードの特徴は

- 応答が早い
- 転送するページの間で request スコープのデータを共有することができる

など、であり、リダイレクトの特徴は

- 異なる Web アプリケーションや異なるホストへも転送できる
- ブラウザが転送先の URL を知ることができる

である。必要に応じて使い分ける必要がある。(例えば転送先のページをブックマークの対象として欲しいなら、フォワードではなくリダイレクトを使う必要がある。)

キーワード :

<%@ page 〜 %>タグ, <% 〜 %>タグ, <%= 〜 %>タグ, <%! 〜 %>タグ, <%-- 〜 --%>タグ, <%@ include 〜 %>タグ, 暗黙オブジェクト, JavaBeans, java.io.Serializable インタフェース, <jsp:useBean〜/>タグ, スコープ, <jsp:getProperty〜/>タグ, <jsp:setProperty〜/>タグ, forward メソッド , sendRedirect メソッド ,