プログラミング Java 3　第 6 回：JSP と Java Beans 入門

ビジネス・ロジックとページ・プレゼンテーションの分離: JavaBeans のコール

１．Ｊａｖａ　Ｂｅａｎｓとは

1.1 Java Beans の定義
Java Beans は、次のように定義されている。

"A Java Bean is a reusable software component written in Java that can be manipulated visually in a builder tool."

ここでのキーワードは、次のようなものである。

1.1.1 ソフトウエア・コンポーネント
コンポーネントとは、ソフトウエアの自己完備した部品のことであり、簡単な例ではボタン、ラベルなどが挙げられる。もちろんこのような可視的なものでなくともコンポーネントとして捉える。この部品が複数集まって組み立てられ、それぞれの間で動作が関連付けされてアプリケーションを作り上げる。

また、これらの部品は一定の規則とデザインパターンに沿って連携動作するように作られなければいけない。

Java Beans は Java 言語によるコンポーネントモデルである。ソフトウエア・コンポーネントとしてはこの他に Microsoft 社の COM をベースとした Active X がある。こちらは Java Beans とは異なり、Windows プラットフォームのみをターゲットとしたものである。

1.1.2.ビジュアル開発
コンポーネントモデルはプログラム的にソフトウエア部品として使用され､アプリケーションを作り上げられなければならないが、加えてビジュアルに部品の組み立て、すなわち開発が行われることが重要なポイントである。Symantec Visual Cafe や Borland JBuilder などのビジュアル開発ツールによってコンポーネントモデルは更に強力な武器となっている。

Java の開発元である Sun から Java Beans のビジュアルな動作確認のためのツールとして BDK(JavaBeans Development Kit)と呼ばれるものが提供されている。

1.2. Java Beans の特徴

1.2.1.Java 言語そのもの
Java Beans を作る際に特別なものは、何も付加されておらず既存の Java 言語そのものでよい。

1.2.2.コンパクトで簡単
Bean はシンプルであり、簡単に開発して、簡単に使用できる。

1.2.3.ポータブル
100% pure Java のため Java の実行環境をサポートするすべてのプラットフォームで動作する。すなわち Java と同様に「Write once,use everywhere」が成り立つ。

1.3　Java Beans の概要

1.3.1 プロパティ(Property)---属性
プロパティは名前によって参照される Bean の属性のこと。通常は、それぞれのプロパティの値を取得するメソッド(getter)、設定するメソッド(setter)をこのプロパティ毎に作ることが必要である。

ビジュアル開発ツールでは GUI によるプロパティエディタが作成され、これよりプロパティ値の編集が可能となる。

1.3.2 メソッド(Method)
Bean のメソッドは通常の Java プログラムのメソッドと変わるところはない。このメソッドは Bean のクラスにより公開された public メソッドである。これらのメソッドにより目的のコンポーネントにアクセスし操作等を行う。

1.3.3 イベント(Event)
あるコンポーネントから他のコンポーネントへの通知を送ることであり、これも通常の Java における「委任イベントモデル」と呼ばれるイベントと同義である。

Bean はイベント発生元であるのでイベントソースとしての作成が必要となる。イベントの発生は fire(発火)と呼ばれる。すなわち、登録先のコンポーネントでのイベントの発火は、登録されたコンポーネントのメソッドを呼び出すことにより実現される。

このようなイベントを通知するための登録は、イベントソースのコンポーネントの登録用メソッドの呼び出しにより行われる。１つのイベントソースに対し複数のリスナー(イベントの通知を受けるコンポーネント)の登録が可能である。

1.3.4 イントロスペクション(Introspection)---認識処理
イントロスペクションとは、Bean コンポーネントのプロパティ、メソッド、イベントを公開し、それを他のコンポーネントが認識するようにすることである。イントロスペクションは実行時にも行われ、またビジュアル開発ツールによる設計時にも行われる。このイントロスペクションはデフォルトで自動的に行われるようになっている。

Bean のプロパティ、メソッドなどを自動的に解析する機構をリフレクション(Reflection)と呼ぶ。リフレクションはまず、Bean のクラスを解析し、そのメソッドを認識する。次にいくつかのデザインパターンからその Bean のプロパティとイベントを認識する。したがって決められたデザインパターンによって Bean を作っている場合には､特別なこと無しにリフレクションによりイントロスペクションが行われる。

このようなリフレクションによる方法のほかに､Bean の情報を Bean 作成者が明示する方法として BeanInfo クラスを実装する方法がある。複雑な Bean の場合はリフレクションによると、あまりにも不要なものまでが編集対象とされてしまうのでこの BeanInfo クラスの実装が必要となってくる場合がしばしばある。ただし、この BeanInfo クラスと Bean とは本来別個のものとして扱われるので、Bean がこれにより大きなサイズを占めるようなことにはならないし、いっしょに持ち運ぶ必要もない。

1.3.5 カスタマイズ(Customize)---編集機能
ビジュアル開発ツールでコンポーネントを組み立ててアプリケーションを作成する場合、編集用のユーザインターフェイスを用いて、プロパティをカスタマイズすることが可能になる。ビジュアル開発ツールは、Bean のプロパティを認識して、これを編集するためのプロパティシートを作成する。

また、Bean が複雑な場合には、このようなデフォルトのプロパティーシートでは簡単に編集できないようなことが起こりうる。このような場合には、デフォルトのプロパティシートとは別に、その Bean を専用に編集するためのカスタマイザを作成することができる。したがって、市販のアプリケーション(例えば Excel など)で見かけるような、ユーザをガイドして手順どおりにカスタマイズさせるウイザードを作成することもできる。

カスタマイザも Bean とは切り離されたものである。Java Beans では Bean の機能のみを実装し、それ以外のものは別のところで実装されるというのが Java Beans アーキテクチャの基本である。

1.3.6.パーシステンス(Persistence)----保存/復元機能
Bean は多様なアプリケーションの部品として使われるために、様々な種類のストレージ(保存)機能をサポートする。Java オブジェクトの状態を自動的に保存/復元するための仕掛けとして、Java Object Serialization が用意されている。これは Java のシステムに標準でサポートされているパーシステンス機能である。これにより Bean は、その時々の状態を保存することができ、再び継続することが可能なポータブル部品となる。

２．　Ｊａｖａ　Ｂｅａｎｓの例

```
package beans;
public class NameBean {
  String newName="";
  public void NameBean() {  }
  public String getNewName() {
    return newName;
  }
  public void setNewName(String newName) {
    this.newName = newName;
  }
}
```

図１．Ｊａｖａ　Ｂｅａｎ　NameBean の内容。

3. Java Beans と JSP

JSP テクノロジによって、静的なページ・プレゼンテーションを決定する HTML コードの開発と、ビジネス・ロジックを処理して動的コンテンツを表示する Java コードの開発を分離できます。したがって、HTML の知識はあるが Java には不慣れなプレゼンテーションとレイアウトの担当と、Java の知識はあるが HTML には不慣れなコード担当との間で、メンテナンス作業を簡単に分割できます。

典型的な JSP ページでは、ほとんどの Java コードとビジネス・ロジックは、JSP ページに埋め込まれているコード内にはありません。かわりに、JSP ページから起動する JavaBeans または Enterprise JavaBeans 内にあります。

JSP テクノロジには、JavaBeans クラスのインスタンスを定義および作成するため、次の構文が用意されています。

<jsp:useBean id="pageBean" class="mybeans.NameBean " scope="page" />

この例では、mybeans.NameBean クラスのインスタンス pageBean を作成します。scope パラメータについては、この章で後述します。

この Bean インスタンスは、このページの後の方で、次の例のように使用できます。

Hello <%= pageBean.getNewName() %> !

この例では、pageBean の newName 属性が、たとえば、名前「John」（ユーザー入力などで設定）の場合は、「Hello John !」が出力されます。

ビジネス・ロジックをページ・プレゼンテーションから分離すると、ビジネス・ロジックと動的コンテンツを担当する Java のエキスパート（NameBean クラスのコードを所有およびメンテナンスする担当者）と、アプリケーション・ユーザーが参照する Web ページの静的なプレゼンテーションとレイアウトを担当する HTML のエキスパート（JSP ページの.jsp ファイルのコードを所有およびメンテナンスする担当者）の間で、作業を分担できます。

JavaBeans で使用するタグ（たとえば、JavaBean インスタンスを宣言するための useBean、Bean プロパティにアクセスするための getProperty と setProperty）の詳細は、「標準アクション: JSP タグ」で説明します。

JSP ページと代替マークアップ言語

JavaServer Pages テクノロジは通常、動的 HTML の出力に使用されますが、JSP 仕様では、構造化されたテキスト・ベースのドキュメント出力に関する追加の型もサポートしています。JSP トランスレータは、JSP 要素の外にあるテキストは処理しないため、Web ページに適切なテキストは、通常 JSP ページにも適切です。

JSP ページは、HTTP リクエストから情報を取得し、データベース・サーバーから（たとえば、SQL データベース問合せを使用して）情報にアクセスします。この情報を、必要に応じて結合および処理し、動的コンテンツを持つ HTTP レスポンスに取り込みます。コンテンツは、HTML、DHTML、XHTML、XML などにフォーマットできます。

JSP 構文の要素の概要

「JSP ページの例」では、JSP 構文の単純な例を示しました。ここでは、トップレベルの構文カテゴリを説明します。

- ディレクティブ: JSP ページ全体に関する情報を伝達します。
- スクリプト要素: 宣言、式、スクリプトレット、コメントなどの Java コーディング要素です。
- オブジェクトとスコープ: JSP オブジェクトは、明示的または暗黙的に作成でき、指定されたスコープ内（JSP ページやセッション内など）でアクセスできます。
- 操作: オブジェクトを作成したり、JSP レスポンスの出力ストリームに影響を与えます。

この項では、基本的な構文とコード例も含めて、各カテゴリについて説明します。Bean プロパティの変換およびカスタム・タグ・ライブラリ（カスタム・アクションに使用されます）の概要についても説明します。詳細は、Sun 社の JSP 仕様を参照してください。

ディレクティブ

ディレクティブは、JSP ページ全体に関する指示を JSP コンテナに提供します。この情報は、ページの変換または実行で使用されます。基本的な構文は、次のとおりです。

<%@ *directive attribute1*="*value1*" *attribute2*="*value2*"... %>

JSP 仕様では、次のディレクティブをサポートします。

- page
- include
- taglib

page ディレクティブ

ページに依存する属性を指定します。たとえば、スクリプト言語、コンテンツ・タイプ、文字エンコード、拡張するクラス、インポートするパッケージ、使用するエラー・ページ、JSP ページの出力バッファ・サイズ、バッファが満杯になったときにバッファを自動的にフラッシュするかどうかなどがあります。例:

```
<%@ page language="java" import="packages.mypackage" errorPage="boof.jsp" %>
```

また、自動フラッシュを有効にし、JSP ページの出力バッファ・サイズを 20KB に設定する場合は、次のようにします。

```
<%@ page autoFlush="true" buffer="20kb" %>
```

include ディレクティブ

変換時に JSP ページに挿入されるテキストまたはコードを含むリソースを指定します。例:

```
<%@ include file="/jsp/userinfopage.jsp" %>
```

リソースへのページ相対パスまたはコンテキスト相対パスのいずれかを指定します。ページ相対パスおよびコンテキスト相対パスの詳細は、「JSP ページのリクエスト」を参照してください。

スクリプト要素

JSP スクリプト要素には、JSP ページに表示可能な Java コードの次のカテゴリが含まれます。

- 宣言
- 式
- スクリプトレット
- コメント

評価され、必要に応じて文字列値に変換され、ページ内の検出された場所に表示される Java 式です。

JSP 式は、セミコロンで終了せず、<%=...%>タグ内に含まれます。例:

```
<P><B> Today is <%= new java.util.Date() %>. Have a nice day! </B></P>
```

スクリプトレット

ページのマークアップ言語内に記述される、Java コードの断片です。スクリプトレットまたはコードの一部分は、1 行または複数行の Java コードで構成されている場合があります。このスクリプトレットを JSP ページの HTML コード内で使用すると、条件ブランチやループなどを設定できます。JSP スクリプトレットは<%...%>スクリプトレット・タグ内に含まれ、通常の Java 構文を使用します。

例 1:

```
<% if (pageBean.getNewName().equals("")) { %>
    I don't know you.
<% } else { %>
    Hello <%= pageBean.getNewName() %>.
<% } %>
```

コメント

Java コード内に埋め込まれたコメントと同様に、JSP コード内に埋め込まれた、開発者のコメントです。コメントは、<%--...--%>構文内に含まれます。例:

```
<%-- Execute the following branch if no user name is entered. --%>
```

HTML のコメントとは異なり、JSP のコメントは、ユーザーがブラウザでページのソースを表示しても参照できません。

JSP **オブジェクトとスコープ**

このマニュアルの JSP オブジェクトという用語は、JSP ページで宣言されるか、または JSP ページからアクセス可能な Java クラス・インスタンスを指します。JSP オブジェクトは、次のいずれかです。

- 明示的: 明示的なオブジェクトは、JSP ページのコード内で宣言および作成され、選択した scope の設定に従って、その JSP ページおよびその他のページにアクセスできます。

または

- 暗黙的: 暗黙的なオブジェクトは、基礎となる JSP 機能によって作成され、特定のオブジェクト型に固有の scope 設定に従って、JSP ページの Java スクリプトレットまたは式からアクセスできます。

JSP ページ内のオブジェクトは、明示的か暗黙的かに関係なく、特定のスコープ内でアクセス可能です。明示的なオブジェクト(jsp:useBean 操作で作成された JavaBean インタンスなど)の場合は、次の構文を使用して明示的にスコープを設定できます。

scope="*scopevalue*"

スコープには、次の 4 種類があります。

- scope="page"(デフォルトのスコープ): オブジェクトには、そのオブジェクトが作成された JSP ページ内からのみアクセスできます。page スコープのオブジェクトは、暗黙的な pageContext オブジェクトに格納されます。page スコープは、ページの実行が停止すると終了します。

  JSP ページの実行中に、ユーザーがページをリフレッシュすると、すべての page スコープのオブジェクトについて新規インスタンスが作成されます。

- scope="request": オブジェクトには、そのオブジェクトを作成した JSP ページと同じ HTTP リクエスト・サービスを提供している JSP ページからアクセスできます。request スコープのオブジェクトは、暗黙的な request オブジェクトに格納されます。request スコープは、HTTP リクエストが完了すると終了します。
- scope="session": オブジェクトには、そのオブジェクトを作成した JSP ページと同じ HTTP セッションを共有する JSP ページからアクセスできます。session スコープのオブジェクトは、暗黙的な session オブジェクトに格納されます。session スコープは、HTTP セッションがタイムアウトになるか、または無効になると終了します。
- scope="application": オブジェクトには、単一の Java Virtual Machine 上で、そのオブジェクトを作成した JSP ページと同じ Web アプリケーションで使用される JSP ページからアクセスできます。これは、Java の静的変数の概念と同じです。application スコープのオブジェクトは、暗黙的な application サーブレット・コンテキスト・オブジェクトに格納されます。application スコープは、アプリケーション自体が終了するか、JSP コンテナまたはサーブレット・コンテナが停止すると終了します。

標準アクション: JSP **タグ**

JSP 操作の要素により、JSP ページの実行中に発生する操作(Java オブジェクトをインスタンス化し、ページに対して使用可能にするなど)が行われます。次のような操作が含まれます。

- JavaBean インスタンスを作成し、そのプロパティにアクセスする操作
- 別の HTML ページ、JSP ページまたはサーブレットに実行を転送する操作
- 外部リソースを JSP ページにインクルードする操作

標準アクションについては、JSP 仕様に一連のタグが定義されています。JSP ページのコードは、この章で前述したディレクティブとスクリプト要素を使用して作成できますが、ここで説明する標準的なタグを使用すると、機能性や利便性が向上します。

JSP の標準アクションに対する一般的なタグの構文は、次のとおりです。

<jsp:*tag attr1*="*value1*" *attr2*="*value2*" ... *attrN*="*valueN*">
...*body*...
</jsp:*tag*>

ボディがない場合は、次のとおりです。

<jsp:*tag attr1*="*value1*", ..., *attrN*="*valueN*" />

JSP 仕様には、次の標準アクション・タグが含まれます。次に、各タグについて簡単に説明します。

- jsp:usebean
- jsp:setProperty
- jsp:getProperty
- jsp:param
- jsp:include
- jsp:forward
- jsp:plugin

jsp:useBean タグ

jsp:useBean タグは、Java 型のインスタンス（通常は JavaBean クラス）にアクセスし、Java 型のインスタンスを作成します。また、インスタンスを指定の名前または ID に関連付けます。インスタンスは、指定したスコープのスクリプト変数として、その ID を介して使用できます。スクリプト変数の概要は、「カスタム・タグ・ライブラリ」を参照してください。スコープの詳細は、「JSP オブジェクトとスコープ」を参照してください。

主な属性は、class、type、id および scope です（使用頻度は多くありませんが、後述の beanName 属性もあります）。

id 属性を使用して、インスタンス名を指定します。JSP コンテナは最初に、指定したスコープ内で、指定した型の指定の ID によってオブジェクトを検索します。オブジェクトが見つからない場合、コンテナはそのオブジェクトの作成を試みます。

class 属性は、JSP コンテナが必要に応じてインスタンス化できるクラスを指定するために使用します。クラスは、抽象クラスにはできません。また、引数のないコンストラクタが必要です。type 属性は、JSP コンテナがインスタンス化できない型（インタフェース、抽象クラス、あるいは引数のないコンストラクタを持たないクラスのうちいずれか）を指定するために使用します。インスタンスがすでに存在する場合やインスタンス化可能なクラスのインスタンスが型に割り当てられる場合は、type を使用します。次に、典型的な 3 種類の使用例を示します。

- ターゲット・スコープにすでに存在しているインスタンスを指定するには、type と id を使用します。
- クラスのインスタンス名を指定するには、class と id を使用します。インスタンスは、ターゲット・スコープにすでに存在しているインスタンスか、JSP コンテナによって新規作成されるインスタンスのいずれかです。
- インスタンス化するクラス、およびインスタンスを割り当てる型を指定するには、class、type および id を使用します。この場合、クラスは、型に対して正式に割当て可能であることが必要です。

scope 属性を使用して、インスタンスのスコープを指定します。ページ・コンテキスト・オブジェクトに関連付けるインスタンスの場合は page、HTTP リクエスト・オブジェクトに関連付けるインスタンスの場合は request、HTTP セッション・オブジェクトに関連付けるインスタンスの場合は session、サーブレット・コンテキストに関連付けるインスタンスの場合は application です。

class 属性のかわりに、beanName 属性を使用することもできます。この場合は、クラス名のかわりにシリアライズ可能なリソースを指定するオプションがあります。beanName 属性を使用すると、JSP コンテナは java.beans.Beans クラスの instantiate()メソッドを使用してインスタンスを作成します。

次の例では、型 MyIntfc の request スコープ・インスタンス reqobj を使用しています。MyIntfc はインタフェースであり、直接インスタンス化できないため、reqobj がすでに存在している必要があります。

<jsp:useBean id="reqobj" type="mypkg.MyIntfc" scope="request" />

次の例では、クラス PageBean の page スコープ・インスタンス pageobj を使用しています（必要に応じて、このインスタンスを最初に作成します）。

<jsp:useBean id="pageobj" class="mybeans.PageBean" scope="page" />

次の例では、クラス SessionBean のインスタンスを作成し、そのインスタンスを型 MyIntfc の変数 sessobj に割り当てています。

<jsp:useBean id="sessobj" class="mybeans.SessionBean"
type="mypkg.MyIntfc scope="session" />

jsp:setProperty タグ

jsp:setProperty タグは、1 つ以上の Bean プロパティを設定します。Bean は、jsp:useBean タグにすでに指定されている必要があります。指定のプロパティに値を直接指定したり、指定のプロパティの値を関連の HTTP リクエストのパラメータから取得したり、HTTP リクエストのパラメータから一連のプロパティと値を繰り返し取得できます。

次の例では、pageBean インスタンス（前述の jsp:useBean の例で定義済）の user プロパティの値を「Smith」に設定します。

```
<jsp:setProperty name="pageBean" property="user" value="Smith" />
```

次の例では、HTTP リクエストの username というパラメータの値セットに従って、pageBean インスタンスの user プロパティを設定します。

```
<jsp:setProperty name="pageBean" property="user" param="username" />
```

Bean のプロパティとリクエスト・パラメータが同じ名前（user）の場合は、次のようにプロパティを設定できます。

```
<jsp:setProperty name="pageBean" property="user" />
```

次の例では、HTTP リクエストのパラメータで、Bean のプロパティ名とリクエストのパラメータ名を繰り返し一致させ、対応するリクエストのパラメータ値に従って Bean のプロパティ値を設定します。

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

jsp:setProperty タグを使用すると、文字列以外のプロパティの値を指定する場合でも、文字列入力を使用できます。これは、バックグラウンドで変換が行われるためです。

jsp:getProperty タグ

jsp:getProperty タグは、Bean のプロパティ値を読み取って Java 文字列に変換し、暗黙的な out オブジェクトに配置します。これによって、文字列値を出力として表示できます。Bean は、jsp:useBean タグにすでに指定されている必要があります。文字列に変換する場合、プリミティブ型は直接変換され、オブジェクト型は、java.lang.Object クラスに指定されている toString()メソッドを使用して変換されます。

次の例では、pageBean Bean の user プロパティの値を、out オブジェクトに出力します。

```
<jsp:getProperty name="pageBean" property="user" />
```

jsp:param タグ

jsp:param タグは、jsp:include タグ、jsp:forward タグおよび jsp:plugin タグ（後述）とともに使用できます。

```
<jsp:param name="username" value="Smith" />
```

jsp:include タグ

jsp:include タグは、リクエスト時にページが表示されると、静的または動的な追加リソースをページに挿入します。リソースは、相対 URL（ページ相対またはアプリケーション相対）を使用して指定します。例:

```
<jsp:include page="/templates/userinfopage.jsp" flush="true" />
```

flush 属性を「true」に設定すると、jsp:include 操作の実行時に、バッファがブラウザにフラッシュされます。JSP 仕様および OC4J JSP コンテナは、「true」または「false」のいずれかの設定をサポートします。デフォルトは「false」です（JSP 仕様 1.1 でサポートされる設定は「true」のみで、flush は必須属性です）。

jsp:forward タグ

jsp:forward タグは、現行のページの実行を事実上終了し、その出力を破棄し、新規ページ（HTML ページ、JSP ページまたはサーブレットのいずれか）をディスパッチします。

jsp:forward タグを使用するには、JSP ページをバッファする必要があります。page ディレクティブに buffer="none"は設定できません。この操作では、バッファがクリアされ、コンテンツはブラウザに出力されません。

jsp:include の場合と同様、次の 2 番目の例に示すように、jsp:param タグを持つ操作ボディも指定できます。

```
<jsp:forward page="/templates/userinfopage.jsp" />
```

jsp:plugin タグ

jsp:plugin タグによって、指定のアプレットまたは JavaBean がクライアント・ブラウザで実行されます。必要に応じて、Java プラグイン・ソフトウェアのダウンロードが先行して実行されます。

次の例は、Sun 社 JSP 仕様 1.2 からの抜粋で、アプレットのプラグインの使用方法を示しています。

```
<jsp:plugin type=applet code="Molecule.class" codebase="/html" >
    <jsp:params>
        <jsp:param name="molecule" value="molecules/benzene.mol" />
    </jsp:params>
    <jsp:fallback>
        <p> Unable to start the plugin. </p>
    </jsp:fallback>
</jsp:plugin>
```

ARCHIVE、HEIGHT、NAME、TITLE、WIDTH など、その他のパラメータも jsp:plugin タグ内で使用できます。これらのパラメータの使用方法は、一般的な HTML 仕様に従います。

4. 実習

4.1 ＪＳＰ開発

Notepad (Notepad++, Eclipse 等)を開く。

下記の Lecture_6_bean_1.jsp の内容を入力する （図2)。

```
<HTML>
  <HEAD>
    <TITLE> Programming 3, Lecture 6 (1)</TITLE>
  </HEAD>
  <BODY>
    <%@ include file = "logo.htm" %>
    <BR>
    JSP and Java Beans: using methods <B>jsp:setProperty</B> and <B>jsp:getProperty</B>
   (Case 1)
    <BR><BR>
    <%@ page import="beans.NameBean"  %>
    <% if (request.getParameter("newName") == null) { %>
         Enter your name:
    <% } else { %>
        <jsp:useBean id="pageBean" class="beans.NameBean" scope="page" />
        <jsp:setProperty name="pageBean" property="newName" param="newName" />
        Hello
        <B><jsp:getProperty name="pageBean" property="newName" /></B>
        !
    <% } %>
    <FORM METHOD="GET" ACTION="Lecture_6_bean_1.jsp" >
      <INPUT NAME = "newName"/>
      <INPUT TYPE = "submit" VALUE="Submit Your Name"/>
   </FORM>
  </BODY>
</HTML>
```

図 2. JSP Lecture_6_bean_1.jsp の内容。 ロケーション：E:¥TEMP¥

編集されたファイル Lecture_6_ bean_1.jsp を E:¥TEMP¥に保存する。

4.2 FFFTP をスタートし、下記の Setup します：

1) ホスト名： isd-si.doshisha.ac.jp
2) ユーザ名： guest
3) パスワード：guest

接続してから、FFTP の右のパネルの Web-server フォルダー（Download と Upload の時）/1XXXXXX/ のフォルダを開く（1XXXXXXX は学生 ID です）

4.3 JSP Upload

FFTP で Web-server のフォルダー/1XXXXXXX/にファイル Lecture_6_bean_1.jsp を Upload する。

4.4 JSP Access

Web アクセス: ＪＳＰのＵＲＬ（Internet Explorer のアドレス・バー）:

http://isd-si.doshisha.ac.jp/teaching/programming_3/1XXXXXXX/Lecture_6_bean_1.jsp

図 3. JSP Lecture_6_bean_1.jsp の最初のアクセス

URL : http://isd-si.doshisha.ac.jp/teaching/programming_3/1XXXXXXX/Lecture_6_bean_1.jsp

図 4. JSP Lecture_6_bean_1.jsp の「John」を入力し、「Submit Your Name」の押してからの結果

URL : http://isd-si.doshisha.ac.jp/teaching/programming_3/ 1XXXXXXX/Lecture_6_bean_1.jsp

4.5 JSP の実行結果を確認

図 4 に表示された Web Browser の画面にマウスの右ボタンを押し、ページのソースを示すと、ページのソースを確認（図 5)。このソースは JSP（Beans のアクセスを含めて）の実行結果です。実行結果と JSP の内容（図 2）を比べる。

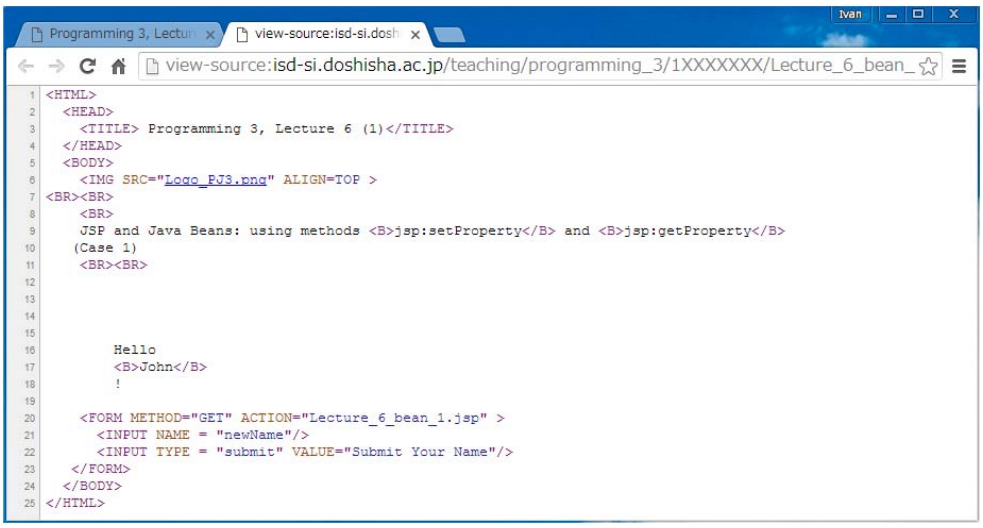

図 5. JSP Lecture_6_bean_1.jsp の実行結果。

4.6  ＪＳＰ開発

Notepad (Notepad++, Eclipse 等)を開く。

下記の Lecture_6_bean_2.jsp の内容を入力する （図 6)。

```
<HTML>
  <HEAD>
    <TITLE> Programming 3, Lecture 6 (2)</TITLE>
  </HEAD>
  <BODY>
    <%@ include file = "logo.htm" %>
    <BR>
    JSP and Java Beans: using methods <B>pageBean.setNewName</B> and <B>pageBean.setNewName</B>
   (Case 2)
    <BR><BR>
    <%@ page import="beans.NameBean"  %>
    <% String name = request.getParameter("newName");
       if (name == null) { %>
         Enter your name:
    <% } else { %>
         Hello
         <jsp:useBean id="pageBean" class="beans.NameBean" scope="page" />
         <% pageBean.setNewName(name); %><B>
         <%= pageBean.getNewName() %></B>
       !
    <% } %>
    <FORM METHOD="GET" ACTION="Lecture_6_bean_2.jsp" >
      <INPUT NAME = "newName"/>
      <INPUT TYPE = "submit" VALUE="Submit"/>
    </FORM>
  </BODY>
</HTML>
```

図 6. JSP Lecture_6_bean_2.jsp の内容。 ロケーション：E:¥TEMP¥

編集されたファイル Lecture_6_ bean_2.jsp を E:¥TEMP¥に保存する。

4.7  FFTP をスタートし、下記の Setup します：

1) ホスト名： isd-si.doshisha.ac.jp
2) ユーザ名： guest
3) パスワード：guest

接続してから、FFTP の右のパネルの Web-server フォルダー（Download と Upload の時）/1XXXXXX/ のフォルダを開く （1XXXXXXX は学生 ID です)

4.8  JSP Upload

FFTP で Web-server のフォルダー/1XXXXXXX/にファイル Lecture_6_bean_2.jsp を Upload する。

4.9 JSP Access

Web アクセス:JSP の URL (Internet Explorer のアドレス・バー) :

http://isd-si.doshisha.ac.jp/teaching/programming_3/1XXXXXXX/Lecture_6_bean_2.jsp

図 7. JSP Lecture_6_bean_2.jsp の最初のアクセス

URL : http://isd-si.doshisha.ac.jp/teaching/programming_3/1XXXXXXX/Lecture_6_bean_2.jsp

図 8. JSP Lecture_6_bean_2.jsp の「John」を入力し、「Submit Your Name」の押してからの結果

URL : http://isd-si.doshisha.ac.jp/teaching/programming_3/ 1XXXXXXX/Lecture_6_bean_2.jsp

4.10 JSP の実行結果を確認

図 8 に表示された Web Browser の画面にマウスの右ボタンを押し、ページのソースを示すと、ページのソースを確認（図 9)。このソースは JSP（Beans のアクセスを含めて）の実行結果です。実行結果と JSP の内容（図 6）を比べる。

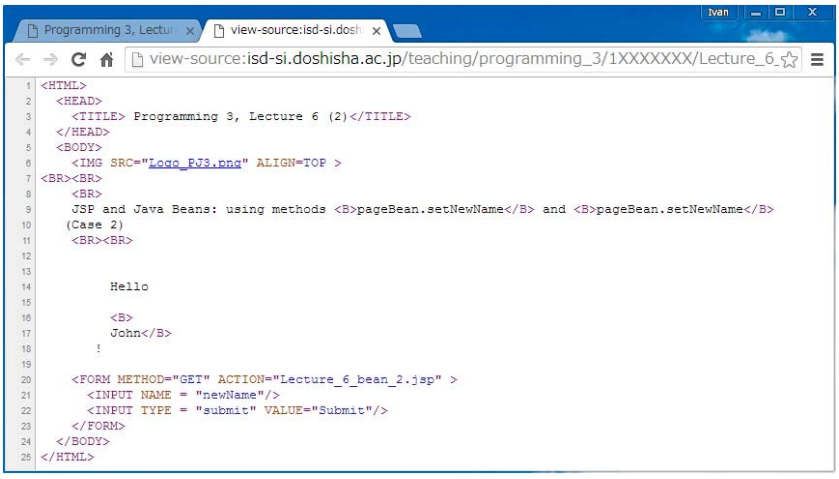

図 9. JSP Lecture_6_bean_2.jsp の実行結果。