練習問題の解答

各LESSONの練習問題の解答をまとめています。
間違ってしまった問題は、もう一度しっかりと本文を復習しましょう。

LESSON 02

A 最初の練習問題は、JSPの基本的な構文にフォーカスしています。間違ってしまった場合には、もう一度、しっかりと本文を復習しておきましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson02/p_intro.jsp

① page

ページの処理方法を指定するのは@pageディレクティブです。ディレクティブは<%@ ... %>で囲まなければなりません。

また、@pageディレクティブに複数の属性を指定したい場合には、練習問題のようにまとめて1つのディレクティブで表してもかまいませんし、以下のように複数に分けて記述してもかまいません。要は見やすさの問題です。

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ page buffer="8kb" %>
<%@ page import="java.util.Date" %>
```

ただし、同一の属性（import属性を除く）を同一ページ内で指定することはできません。

② text/html;charset=UTF-8

contentType属性には、出力するコンテンツの種類（MIMEタイプ）と文字コードセットを指定します。指定値のフォーマットは「MIMEタイプ;chartset=文字コード」とします。

③ buffer

バッファーサイズを指定するのは、buffer属性です。

属性値に「8kb」とあることから、容易に類推できたのではないでしょうか。今回の問題とは直接関係ありませんが、属性値は数値であるか文字列であるかに関係なく、必ずダブルクォーテーションでくくる必要があります。

④ import

スクリプトレットで使用するクラスを宣言するのは、import属性の役割です。ここではクラスを1つ指定しているだけですが、複数のクラスを指定したい場合は、

```
<%@ page import="java.util.Date, java.util.Calendar" %>
```

のようにカンマ区切りで記述するか、

```
<%@ page import="java.util.Date" %>
<%@ page import="java.util.Calendar" %>
```

のようにimport属性を複数列記します。

⑤ <%

スクリプトレットは<% ... %>で囲みます。スクリプトレット内には、Javaで記述した1つ以上の命令文を記述できます。

⑥ out.print

文字列を表示するには、outオブジェクトに属するprintメソッド（命令）を使用します。今後、さまざまなオブジェクトとそれに伴うメソッドが登場しますが、「オブジェクト名.メソッド名(引数,...)」という書式は共通していますので、今のうちから慣れておいてください。改行文字を合わせて出力するout.printlnメソッドでもOKです。

⑦ ;

スクリプトレット内では、文はセミコロンで終わらなければなりません。JavaやC系の言語に慣れていない方は意外と忘れがちですので気をつけてください。

LESSON 03

A 変数／配列のさまざまな宣言方法を今一度確認してみましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson03/p_everyone.jsp

① String[]

変数宣言の基本的な構文は「データ型　変数名;」です。配列を宣言する場合、データ型の後ろに「[]」を付加します。

② int

変数iは、for命令のカウンター変数と配列のインデックス番号となるものなので、int型として宣言しておきます。

この問題の場合は、int型よりも範囲の小さいbyte型、short型で宣言しても、自動的にint型に変換されるため、一応、間違いではありません。逆に、対応する範囲の大きいlongやsingle、doubleなどのデータ型ではエラーとなります。

③ i < msgs.length

本文内のサンプルでは、いったん配列サイズを別の変数にセットしていますが、ここではそのまま条件式の一部として表しています。この問題のように、値を1回しか使用しない場合には、あえて変数に格納せずに使用したほうがプログラムがすっきりします。

④ i++

変数iは1回のループのたびに、1ずつ増加させる必要があります。「i = i + 1」としても間違いではありませんが、通常、よりシンプルに記述できる「i++」を採用します。

⑤ msgs[i]

配列内のデータを参照するには、「配列名[インデックス番号]」とします。

LESSON 04

A switch命令の確認です。if命令で同じ内容を書き直してみるのも良い勉強になるでしょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson04/p_quest1.jsp

① p_quest2.jsp

<form>要素のactionオプションには入力データの送信先（処理を行うJSPスクリプト）のパスを指定します。この場合、送信元のp_quest1.jspと送信先のp_quest2.jspとは同一のフォルダーに置かれていることを想定しているので、p_quest2.jspとだけ指定します。

もし異なるフォルダーにファイルがある場合には、"./jsp/p_quest.jsp"（現在のフォルダー配下のjspフォルダーの下にあるp_quest.jsp）のように指定します。

② select

選択ボックスを生成するには<select>要素を使用します。<select>要素は選択肢が多くなっても、コンパクトに画面上に表示することができるフォーム部品です。

③ switch

フォームから受け取った値（1、2、3）によって表示を切り替えます。

このような多岐分岐の場合は、switch構文を使用するのが一般的です。すぐ下にcaseブロックがあるので、答えは簡単にわかると思います。

④ request.getParameter

選択ボックスfoodで指定された値を取得するには、requestオブジェクトのgetParameterメソッドを使います。getParameterメソッドは必ず値を文字列として返すので、これを数値として扱いたい場合には、Integer.parseIntメソッドでいったん文字列を数値に値を変換しなければなりません。

⑤ break;

caseブロックでは、最後にbreak命令を記述する必要があります（さもないと、現在のブロックを処理したあと、次のcaseブロックに制御が移ってしまいます）。

また、問題文には文末を示すセミコロン（;）がありませんので、些細なことですが、これも回答に含めるのを忘れないようにしてください。

誤っている点

- 1行目の「<%! ...」は「<%@ ...」
  ディレクティブは<%@ ... %>で囲まれなければなりません。ちなみに、<%! ... %>は宣言部を表します。

- 15行目と16行目の間には、「%>」が必要

  スクリプトレットは<% ... %>で囲まれなければなりません。ブロックが閉じられない場合、引き続き16行目の<body>要素をスクリプトレットとして解釈しようとするため、構文エラーになってしまいます。

LESSON 05

Javaで利用可能なループ構文のまとめです。

A1 for命令とwhile命令の違いを確認する問題です。変数の管理と終了条件がまとめて表されるfor命令と異なり、while命令はあくまで終了条件のみが管理されている点に注目してください。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson05/p1_kuku.jsp

① i < 10

whileループには終了条件を指定します。外側のループは変数iが10未満の間、繰り返します。別解として「i <= 9」としても意味は同じです。

② int j = 1

whileループは、forループのように、ループ開始時に変数を初期化しません。したがって、このようにループが入れ子になっている場合には、ループ開始時にきちんと変数を初期化しておく必要があります。

③ j < 10

①と同じです。別解として「j <= 9」でも可能です。

④ j++

whileループは、forループのように、ループのたびに変数をカウントアップしません。したがって、ループ内の命令の一環として、きちんとカウントアップを行う必要があります。このような性質から、変数の値でループの終了条件を決めるような繰り返し処理では、forループのほうがスマートにコードを記述できます。

別解として、「j = j + 1」でも可能です。

⑤ i++

④と同じです。別解として「i = i + 1」でも可能です。

A2 ループを制御する3つの命令for、while、do...whileが、これで出揃いました。これに、if、switch命令による分岐を加えれば、原則、大方のプログラムは記述できるはずです。ここで基本的な構文だけはきちんとおぼえておきましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson05/p2_kuku.jsp

● **15行目は「int i = 1;」**

問題文では変数iは宣言されただけで初期化されていません。その他の箇所でも初期化は行っていませんので、このままでiを参照しようとすると、エラーとなってしまいます。

● **18行目と19行目との間に「int j = 1;」を追加する**

変数jは内側のループが開始される都度、初期化する必要があります。外側のループの外で初期化しても、「i = 1」のときのループ（＝初回のループ）で、一度変数jは10に達してしまいますので、その後はiがいくら変化しても、内側のループは意図したように実行されません。

● **23行目の末尾に「;（セミコロン）」を追加する**

意外と忘れがちなところです。do...while命令の末尾のwhileは、最後をセミコロンで終了しなければなりません。

A3 continue命令の動作を確認します。最初のうちは、break命令との区別がわかりにくいかもしれませんが、こういう簡単なサンプルで動作を比較し、しっかりと理解しておきましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson05/p3_sum.jsp

① **result = 0**

先頭にデータ型が記述されていることから、変数の宣言であることがわかります。プログラム内で登場する変数はiとresultですが、iが②で宣言されていることがわかれば、resultだけが残ります。初期値として0をセットするのを忘れないでください。

② **int**

このように、forループのカウンターとして使用される変数は、別に宣言するよりもfor命令の中で宣言したほうがスマートです。

③ **%**

「奇数である」という判定が「値を2で割った余りが0でないこと」ということがわかってしまえば、意外と簡単に解答が導けるかもしれません。ここでは、iを2で割った余りが0（iが偶数）のとき、現在のループをスキップします。%演算子は、日常の数学では使用しないものなので、最初はなじみにくいかもしれませんが、とても有用な演算子の1つですので、これを機会にぜひおぼえておきましょう。

④ **continue**

ようやく登場しました。問題文にもあるcontinue命令はここで使用します。変数iの値が

偶数である場合、変数resultへの足し込みの処理を行わず、ループをスキップします。

⑤　+=

別解として「= result +」でも可能ですが、「+=」のほうがよりJavaらしいスマートな記述です。ここでは、変数resultにiを順番に足し込んでいくことで、ループ終了時に合計値が格納されるようにしてます。

補足　11～14行目は以下のように書き換えても、同じ意味です。ここではカウンター変数iが2ずつ増加していきますので、やはり奇数のみの足し込みが行われます。

```
for(int i = 1;i <= 100;i += 2){ result += i; }
```

LESSON 06

A HTMLフォームから送信されたパラメーター情報をすべて取得する場合も、クエリ情報を扱う場合とまったく変わりありません。ただし、今回は1つのパラメーターが複数の値を持つ場合にも対応してみましたので、この点にも注目してみてください。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson06/p_query1.jsp

① java.util.*（または java.util.Enumeration）

Enumerationオブジェクトを使用する場合には、@pageディレクティブのimport属性で「java.util.*」を指定する必要があります。import属性の詳細については、あとのLESSONでも紹介しますが、とりあえず、決まりごとのようなものだと思っておいてください。

今回の場合、「java.util.*」の記述は「java.util.Enumeration」としてもOKです。

② Enumeration<String>

getParameterNamesメソッドはEnumerationオブジェクトを返します。Enumerationオブジェクトは変数paramsに格納されますので、その前に変数paramsがEnumerationオブジェクト用の箱であることをあらかじめ宣言しておく必要があります。

ここでは、あとの14行目でデータ型のキャスティングが行われて**いない**ことに注目して、ジェネリックス構文で要素の型も指定している点に注目してください。ジェネリックス構文を利用していない場合、nextElementメソッドの戻り値はObjectオブジェクトですので、明示的に値をキャストする必要があります。

③ nextElement

Enumerationオブジェクトから個々の要素を取り出すには、nextElementメソッドを使用します。②でも述べたように、ここではジェネリックス構文を利用して要素の型を明示していますので、nextElementメソッドはString型の値を返します（ジェネリックス構文を利用していない場合は、Object型の値を返します）。

④ getParameterValues

getParameterメソッドは、パラメーターが複数の値を持つ場合にも、先頭の値1つしか返しません。そこで、あらかじめパラメーターが複数の値を持ちうることが想定される場合、getParameterValuesメソッドを使います。

⑤ values
⑥ value

拡張for命令を用いた例です。拡張for命令では、指定された配列／コレクションから順番に値を取り出し、これを仮引数にセットしながら、配列内の要素がなくなるまでループを繰り返します。

余力のある方は、拡張for命令を利用しない記述に書き直してみるのも良い勉強でしょう。

誤っている点

- 13行目の「hasMoreElements」は「hasMoreElements()」
  メソッドを呼び出す場合、引数が存在しない場合でも、カッコを省略することはできません。
- 21行目と22行目の間に「}」が抜けている
  whileループの終了を示す「}」が閉じていません。ループ構文や条件分岐構文が入れ子になっている場合、意外と閉じカッコを忘れがちなものです。このような場合にも、コードにきちんとインデントを付けておくことで、ブロックの開始と終わりが視覚的に確認しやすくなります。心がけましょう。

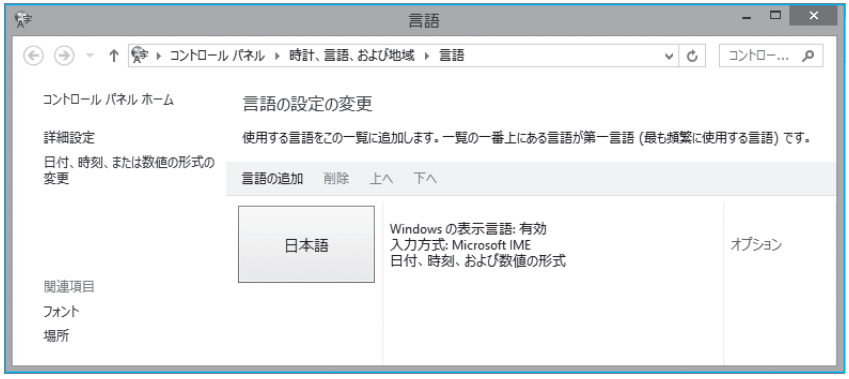
LESSON 07

ヘッダー情報を活用したさまざまな例です。単に構文の確認というに留まらず、実際のアプリケーションでどのように応用できるかをイメージしながら解いていってみてください。

A1 新たなメソッドindexOfが登場しています。今後、さまざまなオブジェクトやメソッドが登場しますが、問題文のように仕様を読むだけで、プログラムの中で応用できるようにする力は重要です。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson07/p1_lang.jsp

ブラウザーの言語設定を変更するには、Internet Explorer 11（Windows 8.1）の場合、[インターネットオプション]で[全般]タブの[言語]を選択すると、[言語の優先順位]ダイアログが開くので、[言語の優先順位の設定]を選択して[言語]ウィンドウを開きます。[言語の追加]で任意の言語を追加できます。また、既存の言語設定を削除する場合には、リストから言語を選択した上で[削除]ボタンをクリックします。



▲ [言語] ダイアログ

① accept-language

ブラウザーの対応言語を表すのは、accept-languageヘッダーです。accept-languageヘッダーを利用することで、ブラウザーの言語によって、ページを切り替えるなどの制御が可能になります。

② !=

accept-languageヘッダーに「ja」という文字が含まれている場合、日本語のメッセージを表示します。indexOfメソッドは指定された部分文字列が元の文字列中に存在しない場合に–1を返しますので、逆に、部分文字列が存在するのはindexOfメソッドが–1以外のときです。否定の否定のような形なので、ちょっと惑わされてしまいそうですが、注意してくだ

さい。

A2 user-agentヘッダーをキーに出力を振り分けることで、クライアントの種類に応じたページを作成できます。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson07/p2_brows.jsp

① user-agent

サイトにアクセスしてきたブラウザーの種類を表すのは、user-agentヘッダーです。user-agentヘッダーには、ブラウザーの種類やバージョン番号、OSなどの情報が含まれています。

② indexOf

クライアントのブラウザーがGoogle Chromeであるかどうかは、user-agentヘッダーにChromeという文字列が含まれているかどうかで判断することができます。user-agentヘッダーが返す具体的な文字列は、自分の目でも確認しておきましょう。

部分文字列の有無を判別するのは、Stringオブジェクトの indexOfメソッドでした。

③ -1

indexOfメソッドの戻り値が-1でないとは、文字列に指定された部分文字列が含まれるということです。問1と空欄の位置が違うだけなので、問1をしっかりと理解できていれば、自動的に答えられたのではないでしょうか。

A3 可能ならば、自分のサイトを訪問したユーザーのrefererヘッダーをログとして記録してみるのもおもしろいかもしれません。自分のサイトがどこからリンクされているのかを知ることができます。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson07/p3_refer.jsp

① referer

refererヘッダーは、現在ページへのリンク元を返します。たとえば、アクセスカウンターで重複したカウントアップを防ぎたい場合などは、refererヘッダーが現在のサイトのドメイン（ホスト名）に等しいかどうかによって、カウントアップするかどうかを決めます（もちろん、それだけでは十分ではありませんが）。

② ||

refererヘッダーがnullである（＝直接、現在のサイトにリンクしてきた）場合、または、

refererヘッダーに現在のホスト名（getServerNameメソッド）が含まれていない場合、ユーザーが他のサイトから現在のページにアクセスしてきたことを示します。

「|||」のような論理演算子を使うことで、このような複合的な条件文を表現できます。

③ ==

②のとおりです。今度は、hostヘッダーの内容がrefererヘッダーに含まれ**ない**場合ですので、indexOfメソッドが–1に等しくなければなりません。問1、2とindexOfメソッドが–1に等しくない場合を見てきたため、思わず「!=」と答えてしまった方は、もう一度コードの流れを見直しましょう。

LESSON 08

細かく見ていくと、なかなか読みでのあるサンプルです。構造を把握して終わりというのではなく、最終的には自分で一からコードを書き起こせるくらいまで、内容を理解してみましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson08/p_cookie1.jsp

A1 ① ウ　② イ　③ ア

A2 ① ウ　② イ　③ ア

LESSON 09

A LESSON 08の練習問題と比較しながら、セッションの構文を確実におさえましょう。また、実際にサンプルを動作させて、「ブラウザを閉じても情報を維持するクッキー」と「ブラウザを閉じると情報を破棄するセッション」との挙動の違いも確認してみましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson09/p_session1.jsp

① getAttribute

セッション情報を取得するのは、暗黙オブジェクトsessionのgetAttributeメソッドの役割です。ここでは、配列namesから取得したフォーム要素名（name、address）をキーに、順にセッション情報を取得しています。

② ?

LESSON 03でも学んだ三項演算子の問題です。ここでは取得したセッション値（values[i]）がnull（未定義値）であるかどうかを判断し、未定義である場合には空文字列をセットしています。この記述がなく、かつセッション値がnullである場合、テキストボックスには「null」という文字列が表示されてしまいますので、要注意です。

前後の「条件式」「値1：値2」という記述から三項演算子が類推できます。

③ setAttribute

セッション情報を設定するのは、暗黙オブジェクトsessionのsetAttributeメソッドです。ここでは、フォームから送信された値を、フォーム要素と同名のキーを持つセッション情報としてセットしています。

④ request.getParameter(names[i])

③のとおりです。

⑤ invalidate

セッションを破棄するのは、invalidateメソッドの役割です。

クライアント側からの見かけ上、セッションはブラウザーを閉じた時点で破棄されますが、これはただ単に紐づけのキーであるセッションIDが破棄されているだけで、セッション情報そのものは明示的にinvalidateメソッドが呼び出されるか、有効期間が経過するまでは削除されない点に注意してください。

LESSON 10

A 文字列中の改行文字を
要素に変換するこのテクニックは、本文中のHTMLエンコードと並んで、Webアプリケーションの中でよく使用されるものです。たとえば、掲示板システムなどで入力された複数行の文章をそのまま表示する場合などには、このテクニックを使って、入力された文字列をあらかじめ処理する必要があります。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson10/p_crlt1.jsp

① **<%!**

JSPページで、ユーザー定義メソッドを宣言するには宣言部（<%! ... %>）を利用します。

② **String**

メソッドの基本的な構文は、以下のとおりです。

```
[アクセス修飾子] 戻り値のデータ型 メソッド名(引数のデータ型 引数,....){
```

ここでは、メソッドの内容を見れば戻り値が文字列型（String）であることが類推できます。

③ **
**

replaceメソッドは、指定した文字列を置き換えます。ここでは文字列中に含まれる改行文字を
要素に置き換えるのでした。

④ **return**

メソッドの戻り値を呼び出し元に返すには、return命令を使用します。

⑤ **transferCRLT**

フォームからの入力値を引数として、transferCRLTメソッドを呼び出しています。

LESSON 11

Calendarクラスを利用した日付操作の理解を確認します。

A1 比較を行う対象が複数になったので、コードは複雑になっていますが、今回の主眼である日付計算の部分はほとんど本文サンプルのままです。見かけの複雑さに惑わされず、コメントも手がかりにしながら骨子をおさえていきましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson11/p_birth.jsp

① getInstance

前後関係からCalendarクラスのインスタンスを生成しようとしているのは明らかです。Calendarクラスのインスタンスはnew演算子ではなく、getInstanceメソッドを介して生成するのでした。

② get

変数yearには、現在の年を格納します。Calendarオブジェクトに含まれる日付／時刻情報はgetメソッドで取得できます。

③ set

Calendarオブジェクトbirthには、i番目のメンバーの今年の誕生日をセットします。Calendarオブジェクトの日付／時刻情報は、setメソッドで設定できます。また、その際、月情報は0～11の範囲で指定しなければならない点に注意してください。

④ ymd[i][0]

「今日の日付」と「i番目のメンバーの誕生日」が等しい場合、誕生日メッセージを表示します。その際、年齢は「(今日の年) – (i番目のメンバーの生年) 」で求めます。

⑤ min

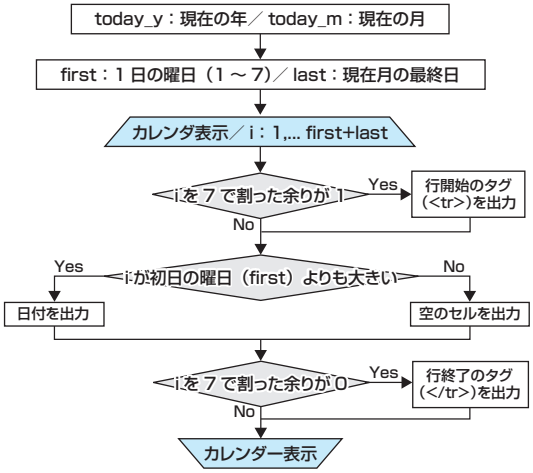
「今日の日付」と「i番目のメンバーの誕生日」が等しくない場合、誕生日がすでに過ぎているかどうかで、今年もしくは来年の誕生日と今日の日付を比較し、その日数差を求めます。現在の日数差が、日数差の最小を格納する変数minよりも小さい場合には、現在の日数差を新たにminに格納すると同時に、変数nameにはi番目のメンバーの名前を格納します。

結果、最終的にループを終了したときには、メンバー全員の中で一番誕生日が近かった人の名前と、その人の誕生日までの日数が変数name、minにセットされているというわけです。

ちょっと込み入ったロジックですが、具体的なデータを想定して、ループを追っていけば、きっと理解できるはずです。がんばってみてください。

A2 Calendarクラス活用の応用編として、ここでは今月のカレンダーを作成してみます。複雑なコードに思われたという方は、リスト内のコメントと以下のフローチャートも参考に、もう一度、コードの全体像を把握してみましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson11/p_calendar.jsp



▲calendar.jspの流れ

① get

コードの流れがわからなくても、とりあえずは構文から日付要素を取得していることは容易にわかると思います。しかし、ここではむしろそれぞれの箇所で「何を目的に」「何を取得しているのか」という点が重要ですので、問題に正解した場合にも、それぞれのコードの意味を理解するように努めてみてください。

特に10行目、「来月の0日目」を設定している部分に注目です。これによって、「今月の最終日」を取得できます。

② %

%演算子は、剰余（割った余り）を求めるための演算子でした。ここでは、現在のカウンター変数iを7で割った余りが1、0である場合に、それぞれ<tr>、</tr>（行の開始と終了）を出力しています。一定間隔で改行したい場合に、このような記述はよく使いますので、おぼえておくと便利でしょう。

③ i - first

実際にカレンダーに表示する日付は、カウンター変数iから今月最初の日（曜日）を差し引くことで求めることができます。

LESSON 12

A 扱っているデータが変わっただけで、コードの流れ自体は本文サンプルのまったくのコピーです。一度解いたら、今度は一から自分でコーディングしてみるなどして、理解を確かなものにしていきましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson12/p_word1.jsp

① java.util.*

連想配列の機能は、HashMapクラスによって実現できます。HashMapクラスは、java.utilパッケージに属しますので、@pageディレクティブのimport属性であらかじめインポートしておく必要があります。

② HashMap<String, String>

上記①の通りです。連想配列の機能を提供するのは、HashMapクラスの役割です。

また、ここではジェネリックス構文で「<キーの型, 値の型>」のように連想配列内のデータ型を明示している点に注目です。ジェネリックス構文を利用しない場合、18行目で連想配列内の要素を取り出す際に、データ型のキャスティングが必要となります。

③ put

単語と意味のセットを、HashMapオブジェクトmapに保存しておきます。

④ containsKey

HashMapオブジェクトに指定されたキーが存在するかどうかを確かめるのは、containsKeyメソッドです。

⑤ ¥t

HashMapオブジェクトの値部分が、タブ文字（¥t）で区切られていることがきちんと認識できていれば、それほど難しい問題ではなかったはずです。HashMapオブジェクトから取得した値を、StringTokenizerクラスを利用して分割します。

LESSON 13

A テキストファイルからデータを検索する基本的な例です。テキストファイルの場合、不特定のデータを検索するには、このように最初から順にデータを舐めていくしか方法がありません。そのため、データが大量になった場合はパフォーマンスの劣化も激しく、あまり好ましくありません。その場合には、極力、データベースを使うようにしましょう。テキストファイルを使うのは、本当に簡易なデータを扱う場合のみです。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson13/p_addbook1.jsp

① java.util.*,java.io.*

FileReader／BufferedReaderクラスはjava.ioパッケージ、StringTokenizerクラスはjava.utilパッケージに、それぞれ属しています。これらのクラスを使用する場合は、あらかじめ該当するパッケージ（クラス）をインポートしておく必要があります。

順番は特に関係ないので「java.io.*,java.util.*」のように書くことも可能ですし、「java.util.StringTokenizer,java.io.FileReader,...」のように、クラス単位にインポートしてもOKです（また、それぞれを異なる@pageディレクティブに分けて記述してもかまいません）。

② application.getRealPath

FileReaderクラスのコンストラクターには、ファイルの絶対パスを渡します。相対パスを絶対パスに変換するには、暗黙オブジェクトapplicationに属するgetRealPathメソッドを利用します。

③ BufferedReader

FileReaderクラスによる読み取りは、パフォーマンス面から効率の良いものではありません。実用の局面では、必ずBufferedReaderクラスを間に介して、バッファリング処理を行いつつ、読み取り処理を行うようにするのが通例です。

④ buf.readLine()

BufferedReaderクラスは、テキストファイルを1行ごとに読み取るreadLineメソッドを提供しています。readLineメソッドは、タブ区切りやCSV形式の定型的なテキストファイルを読み込む場合には特に有効です。

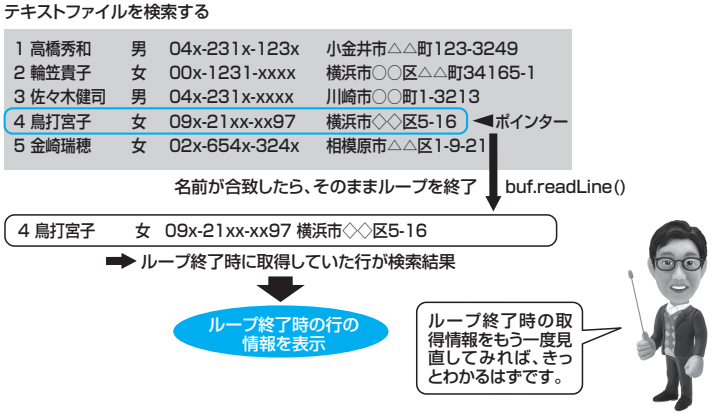
⑤ line

行ごとに読み込まれたテキストは、StringTokenizerクラスを使うことで、タブ文字で分割します。コンストラクターの第1引数には、分割対象の文字列を指定します。

⑥ name.equals(key)

実は、最もわかりにくい箇所かもしれません。

ここで、テキストファイルから取得した名前（name）がフォームから入力された名前（key）と等しい場合、検索対象が見つかったかどうかを示すフラグflagにtrueをセットし、即座にループを脱出します。



▲変数はデータの箱である

この時点で各変数にセットされている情報が検索結果となりますので、これを34～43行目で表示します。

別解として「name.equals(key) == true」としても間違いではありませんが、通常は上の解答のように「== true」を省略した形で記述するのが一般的です。

⑦ flag

別解として「flag == true」でもOKです。

フラグ変数flagがtrueの場合、検索結果を表示します。falseの場合は、検索対象がファイル内に見つからなかったとみなし、「見つからなかった」旨をメッセージ表示します。

LESSON 14

A ファイル書き込みの典型的な手法です。基本的なコードの流れは本文サンプルとなんら変わるところはありませんので、問題を通じて、今一度、テキスト操作の定石を確認してみてください。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson14/p_addrec1.jsp

① StringBuilder

文字列の連結を行う場合、StringBuilderクラスによる処理が効率的です。

② FileWriter

ファイルの書き込みには、FileWriterクラスを利用します。ただし、FileWriterクラスだけでは書き込みの効率が悪いので、一般的にはBufferedWriterクラスを介するのが通例です。

③ true

追加でファイルに書き込む場合は、FileWriterクラスのコンストラクターの第2引数でtrueを指定します。追加書き込みを指定しなかった場合、それまでのデータがクリアされてしまうので、注意が必要です。

④ toString()

BufferedWriterオブジェクトへの書き込みに際しては、StringBuilderオブジェクトの内容を文字列形式に変換する必要があります。

⑤ close()

書き込み終了後は、ファイルをクローズします。

誤っている点

- 1行目の「java.util.*」は「java.io.*」
  コード内で使用しているFileWriter／BufferedWriterなどは、java.ioパッケージに属するクラスです。
- 3行目のsetRequestEncodingはsetCharacterEncodingの誤り。
  メソッド名はすべてがすべて暗記している必要はありませんが、やはり主要なメソッドについては確実に頭の中に刻み込んでおきたいものです。

LESSON 15

正規表現を用いることで、他のアプリケーションでもよく見かける定型的なテクニックを簡単に実現できます。内容を理解したら、自分のプログラムにも積極的に組み込んでみましょう。

A1 よくある入力値チェックの仕組みです。この他、メールアドレスやURL、クレジットカード番号などなど特定の規則に則った文字列を、正規表現を利用することで簡単にチェックできるようになります。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson15/p1_regexp1.jsp

① <%!

4～8行目では、与えられた文字列が郵便番号であるかどうかをチェックするcheckPostNumberメソッドを定義しています。メソッドは宣言部（<%! ... %>）に記述しなければなりません。

② compile

与えられた文字列を正規表現パターンとしてコンパイルします。compileメソッドは、Patternクラスの静的メソッドなので、Pattern.compileの書式で呼び出すことができます（Patternクラスのインスタンスを生成する必要はありません）。

③ ^[0-9]{3}¥¥-[0-9]{4}$

郵便番号を表す正規表現パターンです。「-」は正規表現の予約文字ですので「¥」でエスケープします。また、「¥」はJavaの予約文字なので「¥」でさらにエスケープしています。

また、正規表現パターンの前後に、文字列の開始を表す「^」、終了を表す「$」を指定している点にも要注意です。これがない場合、入力値の中に郵便番号が含まれている（たとえば「270-9871A」のような）文字列でも正しいとみなしてしまうので、注意してください。

④ matcher

正規表現パターンで文字列を検索するのはmatcherメソッドです。matcherメソッドは、マッチング情報を格納したMatcherオブジェクトを返します。

⑤ find

実は、この部分がややわかりにくかったという方も多いかもしれません。正規表現パターンにマッチした部分文字列が存在するかどうかを判定するのは、findメソッドの役割です。ここでは、findメソッドによる判定結果をそのままcheckPostNumberメソッドの戻り値として返しています。

⑥ **request.getParameter("postnum")**

checkPostNumberメソッドの引数として、HTMLフォームから入力された内容を渡す必要があります。

A2 さまざまな局面に応用できるサンプルです。掲示板やブログなどで、表示したい文章に同様の処理をかけることで、文中に含まれるURLに自動的にリンクを張ることができます。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson15/p2_url.jsp

① **java.util.regex.***

正規表現クラスを利用するには、java.util.regexパッケージをインポートしておく必要があります。"java.util.*" は不可ですので、注意してください。

② **Pattern**

正規表現を使用するにあたっては、あらかじめ正規表現パターンをPattern.compileメソッドで解析しておく必要があります。

③ **Pattern.CASE_INSENSITIVE**

compileメソッドの第2引数には、マッチングオプションを設定します。ここでは、大文字小文字を区別しないことを意味する「Pattern.CASE_INSENSITIVE」を指定します。

④ **matcher**

Patternオブジェクトのmatcherメソッドは、Patternオブジェクトに格納された正規表現パターンを使用して、引数で指定された文字列にマッチングするものがないかどうかを検索します。

⑤ **replaceAll**

replaceAllメソッドは、正規表現パターンにマッチングした文字列を指定された文字列で置き換えます。$0は正規表現パターンにマッチングした文字列全体を表します。replaceAllメソッドでは「$0」を利用することで、マッチング文字列を置き換え後の文字列に含めることができます。

LESSON 17

SQLの書き方を練習します。SQLは、直接JSP（Java）に関係するものではありませんが、これからデータベース連携のコードを書いていく上では欠かせないものばかりです。

A1 基本的なINSERT / UPDATE / DELETE命令の練習です。SELECT命令と違って、これらの命令はいつもほぼ同じように表現できます。値を変えながら何度も命令を実際に実行して、基本的な構文を身につけてしまいましょう。

① `INSERT INTO members (name, depart, age) VALUES ('鈴木うめ', '営業部', 22);`
② `UPDATE members SET updated = '2015-04-01' WHERE name = '鈴木うめ';`
③ `DELETE FROM members WHERE name = '鈴木うめ';`

データを変更した結果は「SELECT * FROM members;」で確認してください。

A2 SELECT命令には、さまざまなバリエーションがあります。その中でも、実習で挙げたパターンは特によく利用するものばかりですので、自分で問題を作成して練習を重ねておきましょう。

① `SELECT name, depart FROM members WHERE depart IN ('無所属', '人事部');`
② `SELECT depart, MAX(age) FROM members GROUP BY depart;`
③ `SELECT * FROM members WHERE depart <> '無所属' AND updated IS NOT NULL ORDER BY updated DESC, id ASC;`

ORDER BY句の並び順はASC（昇順）がデフォルトなので、「ORDER BY updated DESC, id」（ASCなし）でも正解です。

LESSON 18

データベース接続に関わる問題です。クラスローダーの考え方、データベース接続文字列などは、いずれも重要なトピックスですので、間違えてしまった人はもう一度、講義を見直してみてください。

A1 %CATALINA_HOME%/lib

クラスライブラリはできるだけ適用範囲が限定された場所に配置するべきです。一般的なライブラリは「%CATALINA_HOME%/webapps/＜アプリケーションルート＞/WEB-INF/lib」フォルダーに配置するべきですが、Connector/JにはTomcatからアクセスできる必要があります。よって、その上位の「%CATALINA_HOME%/lib」に配置します。拡張ライブラリの配置先については、P.201～202も参照してください。

A2

context.xmlの基本的な設定を問う問題です。一度設定してしまえば、なかなか自分で触れることもない部分ですが、逆に、データベースに接続できないという場合、問題となりやすい個所です。間違えた場合にも気づきやすいだけの理解を深めておきましょう。

① password

データベースに接続するためには、ユーザー名／パスワードを指定します。ユーザー名（username属性）はすでに指定されているので、残りはパスワードと推測できます。

② driverClassName

接続に利用するドライバーの完全修飾名を表します。Connector/Jを利用する場合、設定値は「org.gjt.mm.mysql.Driver」で固定です。

③ jdbc:mysql://localhost/jsp10?useUnicode=true&characterEncoding=UTF-8

この設問の一番のポイントです。接続文字列の書式「jdbc:mysql:// ホスト名/ データベース名[?属性情報]」はきちんと理解しておいて、自分の環境でデータベースに接続する場合にも、適切な値に書き換えられるようにしておきましょう。

④ validationQuery

validationQuery属性は、接続の有効性を確認するためのものです。SELECT命令が指定されていることから類推できますね。

LESSON 19

A 本文のサンプルとほぼ同様の流れです。迷ってしまったという方は、もう一度、実習／講義でSQL命令発行の定型的な手順を見直してみましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson19/p_new.jsp

① java.sql.*, javax.naming.*, javax.sql.*

コネクションプーリングを利用してデータベースにアクセスするには、これら3つのパッケージをあらかじめインポートしておく必要があります。

② java:comp/env/jdbc/Jsp10

設定済みのデータソースを呼び出すにはlookupメソッドを使用します。「java:comp/env/」の部分は固定値、「jdbc/Jsp10」にはcontext.xml（コンテキスト情報ファイル）で設定したデータソース名を指定します。

③ INSERT INTO

データを新規に挿入するSQL文は「INSERT INTO」です。基本的なINSERT / UPDATE / DELETE / SELECTの構文は、アプリケーションを作成する中で確実におさえておきたいものです。

④ executeUpdate

レコード（結果セット）を返さないSQL文（INSERT / UPDATE / DELETEなど）を実行する場合には、executeUpdateメソッドを使用します。あとから登場するexecuteQueryメソッドは、結果セットを返すSELECT命令を発行する場合に使用しますので、明確に区別してください。

⑤ close

データベースの使用後は、closeメソッドで接続を切断します。

LESSON 20

A クエリ情報をキーにして一覧画面から詳細画面へと遷移する ── このサンプルの手法は定石とも言えるテクニックの1つです。単純に穴埋め問題を解くだけではなく、コード全体を見直して、自分のアプリケーションで応用できる力を身につけましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson20_21/p_booklist.jsp

① InitialContext

データソースを呼び出すには、InitialContextクラスのlookupメソッドを使用します。

② executeQuery

結果セットを返すSQL命令（SELECT命令）を実行するのは、executeQueryメソッドの役割です。INSERT / UPDATE / DELETE命令などを実行するexecuteUpdateメソッドとは明確に区別してください。

③ DecimalFormat

ここではデータベースから取得した価格（priceフィールド）を「9,999円」の形式で整形しています。数値を整形するのは、DecimalFormatクラスの役割でした。利用可能な書式文字列の意味については、P.182もあわせて参照してください。

④ rs.next()

nextメソッドは、レコードポインターを1つずつずらします。ちなみに、ResultSetオブジェクトには、ポインターを前に移動するpreviousメソッドや先頭に移動するfirstメソッド、最後に移動するlastメソッドなども用意されています。

⑤ rs.getString("isbn")

問題文の条件から、クエリ情報にはレコードのキーとなるisbnフィールドの値を渡すのでした。isbnフィールドの値を取得するには、getStringメソッドを使用します。

⑥ rs.getInt("price")

③のとおりです。ここでは、DecimalFormatクラスを使ってデータベースから取り出した価格情報を整形しています。DecimalFormatオブジェクトのformatメソッドは引数としてint型を受け取りますので、フィールドの値もint型として取得する必要があります。フィールド値をint型で取り出すのは、getIntメソッドです。

⑦ request.getParameter("isbn")

p_booklist.jspからクエリ情報として送出されたキーisbnの値を取得します。
LESSON 06でクエリ情報について勉強したときには、どのような局面で活用されるのか、

なかなかイメージがつかみにくかったかもしれませんが、この例で、ちょっと理解できたのではないでしょうか。

LESSON 21

A 一覧表を処理するスクリプトということで、データベース処理はさておいても、かなり複雑なサンプルとなっています。もしわかりにくいという場合には、まず一覧表として出力されたHTMLのソース（p_edit.jsp）をブラウザーから参照し、どのようなデータが送られるはずなのかをHTMLとして確認したうえで、更新／削除処理部分を見てみるとよいでしょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson20_21/p_edit.jsp

① count = 1

ここでは、一覧表の各行のフォーム部品にユニーク（一意）な名前を付けるために、要素名の後ろに行番号を付加しています。つまり、1行目のtitleであれば対応するテキストボックスの名前は「title1」のようになります。

4行目では、行番号カウント用の変数countを初期化しています。

② rs.next()

LESSON 20の復習です。

取り出した結果セットの内容を全件取り出すためには、nextメソッドがfalseを返す（＝取得するレコードがなくなる）までレコードの取得ループを繰り返します。定石とも言うべき手法ですので、この部分は確実におさえておきましょう。

③ rs.getString("publish")

本文ではcreateOptionメソッドとして切り出していた部分ですね。

選択ボックスにデータベースの値を反映させるためには、p_edit.jspの4～48行目のように、<option>要素の値とデータベースの値を比較照合し、値の等しい<option>要素にselected属性を付加します。これも定石のコードなので、おぼえておきたいところです。

④ <%=count %>

一覧表フォームを送信する場合、処理すべきレコードの数をあらかじめ特定するために、隠しフィールドとしてレコード数を送信します。

⑤ setAutoCommit

ここでは自動コミットモードを無効にし、トランザクションを開始しています。このように、1つのリクエストで複数の処理（更新／削除）を行う場合、トランザクションの利用は必須です。

27行目にcommitメソッドがあることからも、トランザクションの利用を類推できるのではないでしょうか。

⑥ int i = 1; i < count; i++

p_edit.jspから送信されたレコード数分だけforループを繰り返します。

変数iはどこでも宣言されていませんので、for文の中で宣言しなければならないことに注意してください。

⑦ request.getParameter("delete" + i) == null

p_edit.jspで削除サインにチェックが入ったものに関しては、そのレコードを削除し、それ以外のレコードについては、変更の有無にかかわらず、更新することにします。

削除サインがチェックされたかどうかは、リクエスト情報deleteX（Xは連番）の内容がnull（未初期化状態）かどうかで判別します。

⑧ executeUpdate

実際の更新、削除処理を行う部分です。実行の結果、レコードを返さないSQL命令を発行するには、executeUpdateメソッドを用います。戻り値として、ResultSetオブジェクトを返すexecuteQueryメソッドとはしっかりと区別しておきましょう。

⑨ up

一連のデータベース処理のあと、使用したPreparedStatement / Connectionオブジェクトを破棄します。破棄する場合には、オブジェクトを生成したのとは逆順、つまり、PreparedStatement→Connectionの順番で行います。

⑩ db

⑨のとおりです。

LESSON 22

A 区切り文字がタブからカンマになっただけで、出力の流れそのものは実習のそれと同じです。答えに悩んでしまったという人は、もう一度、講義の内容を復習しておきましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson22/p_download.jsp

① application/octet-stream

出力したコンテンツをクライアント側でダウンロードさせるには、コンテンツタイプとして「application/octet-stream」（バイナリデータ）を指定します。

② trimDirectiveWhitespaces

カンマ区切りテキストのように改行や空白に意味のあるフォーマットでは、時として、ディレクティブによる空行が邪魔になります。そこでtrimDirectiveWhitespaces属性をtrueに設定しておくことで、ディレクティブの空行を除去できます。

③ setHeader

ダウンロードファイルのデフォルト名を指定するには、setHeaderメソッドでContent-Dispositionヘッダーを設定します。必ずしも設定しなくてもかまいませんが、まずはクライアントにとってわかりやすい名前をあらかじめ用意しておくのが望ましいでしょう。

④ ResultSetMetaData

結果セットの構成情報（メタ情報）を管理するのは、ResultSetMetaDataクラスの役割です。ResultSetオブジェクトのgetMetaDataメソッドを介して取得できます。

⑤ getColumnCount

ここでは、フィールドの数だけforループを繰り返し、すべてのフィールド値を読み込んでいます。フィールドの数は、ResultSetMetaDataオブジェクト（メタ情報）からgetColumnCountメソッドを介して取得できます。

⑥ i

結果セットのフィールド値には、名前だけでなく、インデックス値からでもアクセスできるのでした。インデックス値にはforループのカウンター変数を利用します。

⑦ ¥r¥n

カンマ区切りテキストでは、1件のレコードを出力するたびに改行文字で区切る必要があります。

LESSON 23

A LESSON 05を見ながら、JSPとサーブレットとの違いをよく比較してみてください。決まりきったルールさえおぼえてしまえば、サーブレットがことさらに難しいことはないことがわかるはずです。

コンパイル済みのP_KukuServlet.classは/WEB-INF/classes/to/msn/wings/lesson23フォルダーに格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson23/P_KukuServlet

① import

サーブレットで外部クラスをインポートする場合は、import命令を使用します。JSPにおける@pageディレクティブのimport属性にあたります。JSPでは省略可能であったjavax.servlet、javax.servlet.httpパッケージも、サーブレットでは明示的にインポートしなければならない点に注意してください。

② @WebServlet

サーブレットを呼び出すためのパスは、@WebServletアノテーションで定義します。@WebServletアノテーションは、サーブレットクラスの先頭で指定するのでした。

③ HttpServlet

サーブレットクラスは、HttpServletクラスを継承して定義する必要があります。HttpServletクラスにはサーブレットとしての最低限の機能が実装されています。サーブレットプログラミングでは、このHttpServletクラスに対して必要な機能を追加していきます。

④ doGet

HttpServletクラスで用意されているメソッドのうち、クライアントからの直接の要求を処理するのが、このdoGetメソッドです。ブラウザーからサーブレットに要求があった場合（正確にはGETリクエストがあった場合）、まずはこのメソッドが呼び出されます。

⑤ setContentType

JSPで言うところの@pageディレクティブのcontentType属性にあたるのが、この部分です。サーブレットによって出力されるコンテンツの型を指定します。

⑥ getWriter

JSPの暗黙オブジェクトoutにあたるのが、PrintWriterクラスです。PrintWriterクラスは、HttpServletResponseオブジェクトresponseのgetWriterメソッドを介して取得できます。

LESSON 24

既存のJSPプログラムをひたすらにサーブレットに置き換える練習です。基本的には決まりきったルールの連なりですので、量をこなして、体でおぼえてしまいましょう。

A1 このサンプルはp1_quest.jspから起動します。quest1.jsp（P.58）とポスト先が異なる程度で、内容はほとんど同じですので、紙面上は割愛しています。

また、コンパイル済みのP1_QuestServlet.classは/WEB-INF/classes/to/msn/wings/lesson24フォルダーに格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson24/p1_quest.jsp

① **javax.servlet.http**

サーブレットを使用する場合には、最低限、java.io、javax.servlet、javax.servlet.http、javax.servlet.annotationパッケージに含まれる6つのクラスをインポートしておく必要があります。

② **@WebServlet**

サーブレットを動作するには、最低限、@WebServletアノテーションでサーブレット呼び出しのパスを宣言しておかなければなりません。web.xml（<servlet>/<servlet-mapping>要素）で宣言する例についても、あわせて復習しておきましょう。

③ **P1_QuestServlet**

サーブレットのクラス名は必ずファイル名と等しくなければいけません。異なる場合はコンパイル時にエラーが発生するので、注意してください。

④ **doPost**

POST形式の<form>から呼び出されたサーブレットでは、まず最初にdoPostメソッドが実行されます。

⑤ **HttpServletRequest**

doPostメソッドは、リクエスト情報を管理するHttpServletRequestオブジェクトと出力を管理するHttpServletResponseオブジェクトとを引数として受け取ります。

⑥ **HttpServletResponse**

⑤のとおりです。

A2

コンパイル済みのP2_QueryServlet.classは/WEB-INF/classes/to/msn/wings/lesson24フォルダー配下に格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson24/P2_QueryServlet?id=namidon&name=kakeya

クエリ情報の部分は自由に変更して、結果の変化を確認してみましょう。

① java.util

Enumerationオブジェクトは、java.utilパッケージに属します。

② HttpServlet

サーブレットクラスは、HttpServletクラスを継承しなければなりません。

③ doGet

直接にURLから指定されて呼び出されるサーブレットでは、最初にdoGetメソッドが実行されます。

④ ServletException

doGetメソッドは、リクエストの処理中に発生した入出力エラーを表すIOException、リクエストの処理中に発生したエラーを表すServletExceptionをスローします。メソッド内部で発生する可能性のある例外は、必ず明示的に宣言しなければなりません。

⑤ setContentType

サーブレットからなんらかのデータを出力する場合には、コンテンツタイプを設定しておく必要があります。これには、HttpServletResponseオブジェクトのsetContentTypeメソッドを呼び出してください。JSPのcontentType属性（@page）に相当します。

⑥ PrintWriter

JSPの暗黙オブジェクトoutにあたるのが、このPrintWriterクラスです。サーブレット内ではgetWriterメソッドで明示的に生成する必要があります。

⑦ nextElement

Enumerationオブジェクトから次の要素を取得するには、hasMoreElements / nextElementメソッドの組み合わせでwhileループを回すのでした。LESSON 06の復習です。

A3 コンパイル済みのP3_CookieServlet.classは、/WEB-INF/classes/to/msn/wings/lesson24フォルダー配下に格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson24/P3_CookieServlet

① **public void**

doGetメソッドを定義する際の決まり文句のようなものです。「public void」は、メソッドが外部のクラスから自由にアクセスでき（public）、戻り値として何も返さない（void）ことを示します。

② **getCookies()**

LESSON 08の復習です。クライアントから出力したクッキーを取得するには、getCookiesメソッドを用います。

③ **response**

生成したCookieオブジェクトをクライアントに送信するには、HttpServerResponseオブジェクトのaddCookieメソッドを用います。

④ **response.setContentType**

JSPでは、@pageディレクティブのcontentType属性で指定していた部分です。

⑤ **out**

JSPでは暗黙オブジェクトとして自由に使うことができたoutオブジェクトも、サーブレットでは自分できちんと定義しなければなりません。ここでは、JSPとの比較をわかりやすくする意味からoutという変数名を使っていますが、もちろん、この部分は自由に変更することが可能です。

LESSON 25

LESSON 24に引き続き、以前勉強したJSPのコードをサーブレットに移植してみます。内部のロジックは以前の復習なので、JSPとサーブレットとの違いという点に集中できるでしょう。

A1 コンパイル済みのP1_ReaderServlet.classは、/WEB-INF/classes/to/msn/wings/lesson25フォルダー配下に格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson25/P1_ReaderServlet

① java.util.StringTokenizer

少しわかりにくいかもしれませんが、③～⑤がわかってしまえば、あとは簡単です。P1_ReaderServletクラスではさまざまなクラスが使われていますが、インポートされていないのは、java.util.StringTokenizerクラスだけです。

```
import java.util.*;
```

のように、パッケージ単位にインポートしてもかまいませんが、ここでは他のimport命令と足並みを揃えて、クラス名まで明示的に記述します。

② application

getServletContextメソッドは、ServletContextオブジェクトを返します。ServletContextオブジェクトは、JSPの暗黙オブジェクトapplicationに相当します。

30行目から、applicationがServletContextオブジェクトを表していることを読み取れます。

③ StringTokenizer

LESSON 14の復習です。文字列を一定の区切り文字（この場合はタブ文字）で分割するには、StringTokenizerクラスを使用します。

④ hasMoreTokens()

StringTokenizerクラスで分割された文字列（トークン）を読み出す場合の典型的な記述です。hasMoreTokensメソッドがtrueの間（＝次のトークンが存在する間）、ループを繰り返し、nextTokenメソッドで次のトークンを読み込むことで、すべてのトークンを処理します。

⑤ nextToken()

④のとおりです。

A2

コンパイル済みのP2_RegexpServlet.classは、/WEB-INF/classes/to/msn/wings/lesson25フォルダー配下に格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson25/P2_RegexpServlet

① java.util.regex

正規表現機能を使うには、java.util.regexパッケージをあらかじめインポートしておく必要があります。

② public class P2_RegexpServlet extends HttpServlet

間違えてしまった方は、サーブレットクラスを定義する構文を今一度復習しておきましょう。「アクセス修飾子 class クラス名 extends HttpServlet」です。

③ public void doGet(HttpServletRequest request, HttpServletResponse response)

doGetメソッドの書式を今一度確認しておきます。引数には、ユーザーリクエストを受け取るHttpServletRequestクラス、クライアントへのレスポンスを管理するHttpServletResponseクラスを指定します。②のクラス定義とともに、サーブレットの最も根本となるところですので、ここだけはしっかりとおぼえておいてください。

④ compile

LESSON 15の復習です。

正規表現パターンは、compileメソッドでコンパイルしたうえで、Patternオブジェクトに格納しておく必要があります。

⑤ find()

正規表現パターンのマッチング結果を表すMatcherオブジェクトの内容は、findメソッドで走査できます。次の検索結果がまだ存在する場合、findメソッドはtrueを返します。

LESSON 26

A 入力フォームp_new.jspは、ほとんどLESSON 19の練習問題p_new.jspと同じなので、ここでは割愛しています。具体的なコードは、配布サンプルからp_new.jspを参照してください。

また、コンパイル済みのP_InsertServlet.classとBook.classは、/WEB-INF/classes/to/msn/wings/lesson26_28フォルダーに格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson26_28/p_new.jsp

① Book

サーブレットからデータ登録用のJavaBeansを呼び出します。データ登録用JavaBeansはBookクラスで定義されています。

② insertInfo

setIsbn / setTitle / setPrice / setPublish / setPublishedメソッドで、一連の入力値をJavaBeansにセットした後、データベースへの登録処理を実行します。Bookクラスのコードを見れば、実際の登録処理がinsertInfoメソッドで定義されているのがわかるはずです。

③ Serializable

クラスがJavaBeansであるための条件の1つ目です。JavaBeansクラスは、Serializableインターフェイスを実装している必要があります。

④ Book()

クラスがJavaBeansであるための条件の2つ目です。JavaBeansクラスは、引数を持たないコンストラクターを持つ必要があります。

ただし、この例のように、コンストラクターで処理する内容がない場合、コンストラクターの記述自体を省略してもかまいません。その場合、コンパイル時に、引数のないデフォルトのコンストラクターが自動的に生成されます。

⑤ setIsbn

クラスがJavaBeansであるための条件の3つ目です。JavaBeansで保持するフィールドはprivate変数として定義し、原則、外部クラスの直接のアクセスを許しません。その代わりに、フィールド値を取得するためのgetXxxxxメソッド、設定するためのsetXxxxxメソッドを定義しておく必要があります。Xxxxxの部分は、フィールド名の頭文字のみを大文字にしたものです。

LESSON 27

A コンパイル済みのP_SearchServlet1.class、P_SearchServlet2.class、Book.classは、/WEB-INF/classes/to/msn/wings/lesson26_28フォルダーに格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson26_28/P_SearchServlet1

① public class P_SearchServlet1 extends HttpServlet {

サーブレット基本構文の復習です。サーブレットのクラスは、HttpServletクラスを継承しており、かつ、アクセス修飾子はpublicでなければなりません。また、クラス名とファイル名は等しくなければなりません。

② Book

getInfosメソッドは静的メソッドです。インスタンスを生成しなくても、クラス名から直接に呼び出せます。

③ Book

拡張for命令は、リストの中に含まれる要素を順に取得します。ここでは、28行目に注目してみると、ArrayListオブジェクトがジェネリック構文で「<Book>」と指定されていますので、取り出される要素もBookオブジェクトであるはずです。

④ dformat.format(info.getPublished())

以前の復習です。データベースから読み込んだ日付型の値を整形するには、SimpleDataFormatオブジェクトのformatメソッドを使用します。

⑤ Book info = Book.getInfo(request.getParameter("isbn"));

前後の文脈から判断する必要があります。ここでは、クエリ情報isbnをキーにデータベースの内容を検索します。getInfoメソッドはstaticメソッドなので、インスタンス化をすることなく、クラス名から直接にアクセスできます。

⑥ Serializable

JavaクラスがJavaBeansであるための条件の1つです。JavaBeansは、必ずSerializableインターフェイスを実装する必要があります。

⑦ ArrayList<Book>

変数listがgetInfosメソッドの戻り値を表していることを読み取れれば、正解を導くのは簡単です。繰り返し登場しているジェネリック構文ですが、改めておさえておきましょう。

⑧ SELECT * FROM book WHERE isbn=?

prepareStatementメソッドで指定するSQL命令には、パラメーターの置き場所（プレースホルダー）を表す「?」を含むことができるのでした。ここでは、isbnフィールドの値をパラメーターとして動的にセットします。

⑨ rs.next()

少々わかりにくいところかもしれません。取得した結果セットは初期状態で「先頭レコードの直前」にありますので、まず最初にnextメソッドでレコードポインターを先頭レコードの位置に移動する必要があります。nextメソッドがfalseを返した場合、該当するレコードは存在しなかったことを意味します。

⑩ finally

finallyブロックで記述された内容は、tryブロックで例外が発生するとしないとにかかわらず、実行されます。データベース接続の切断など、主に使用したリソースの後処理などを記述するのに使用します。

LESSON 28

A p_mvc1.jsp、p_mvc2.jspは/lesson26_28フォルダー、コンパイル済みのP_MvcServlet1.class、P_MvcServlet2.class、Book.classは/WEB-INF/classes/to/msn/wings/lesson26_28フォルダーに、それぞれ格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson26_28/P_MvcServlet1

① **Book**

Bookクラスの getInfos メソッドは、staticキーワードが付けられた静的メソッドです。インスタンス化することなく、直接にクラス名で呼び出せます。

② **request**

JSPページに情報を引き継ぐには、リクエスト属性に値を格納します。リクエスト属性に値をセットするのは、HttpServletRequestオブジェクトrequestのsetAttributeメソッドの役割です。

③ **forward**

リクエスト情報を保持したまま、JSPページに処理を遷移するには、forwardメソッドを使用します。本文を参照して、リダイレクトとフォワードの違いを理解しておいてください。

④ **jsp:useBean**

リクエスト属性に格納されたBeansをJSPページ上で有効化するには、<jsp:useBean>要素を使用します。

⑤ **request**

サーブレットでは、リクエスト属性（requestスコープの変数）にJavaBeansをセットしていますので、JSPページでもrequestスコープで値を参照する必要があります。もしスコープが異なる場合は異なるBeansとみなされてしまい、正しく情報を共有することができないので注意してください。

⑥ **to.msn.wings.lesson26_28.Book**

ジェネリック構文なしのArrayListオブジェクトから拡張for命令で要素を取り出した場合、必ずデータ型をキャストする必要があります。ただし、今回のように異なるパッケージに属するクラスにキャストする場合には、型を完全修飾名で指定するか、あらかじめimport命令でパッケージをインポートしておく必要があります。

⑦ **/lesson26_28/P_MvcServlet2**

@WebServletアノテーションでは、サーブレットの呼び出しパスを指定します。呼び出しのパスは、p_mvc1.jspのリンクから類推できます。アプリケーションルートからの相対パスなので、「/lesson26_28/」の部分を忘れないようにしてください。

⑧ **/lesson26_28/p_mvc2.jsp**

サーブレットでの処理結果をJSPページに転送します。P_MvcServlet1.javaと照らし合わせてみれば、答えはすぐに類推できますね。

⑨ **to.msn.wings.lesson26_28.Book**

<jsp:useBean>要素のclass属性には、インスタンス化するクラスの完全修飾名を指定します。

LESSON 29

JSPページに含まれるスクリプティング要素をひたすらJSTL＋式言語に置き換える練習です。JSTL＋式言語は、いずれも簡易な言語ですので、とにかく量をこなして、身体で感覚をおぼえてしまいましょう。

A1

Coreタグライブラリの中でも最も基本的な<c:set> / <c:forEach>要素の用法を理解しましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson29/p1_every.jsp

① c

JSTLを利用する場合には、あらかじめ@taglibディレクティブで使用を宣言しておく必要があります。prefix属性には、カスタムタグで使用する接頭辞を指定してください。

Coreタグライブラリの接頭辞は一般的に「c」になりますが、いちいちおぼえておかなくても、11、13行目などから容易に類推できるはずです。

② おはようございます, こんにちは, こんばんは

<c:set>要素は、指定された変数に対して値をセットします。ここでは、<c:forEach>要素で利用するために、カンマ区切りの文字列を指定しておきます。<c:forEach>要素は、カンマ区切りの文字列を分解し、取得した個々の部分文字列に対して繰り返し処理を行います。

③ ${msg}

<c:forEach>要素のvar属性で指定された変数は、<c:forEach>要素の配下でのみ参照できます。式言語で変数を参照するには、${変数名}と書きます。

A2

Coreタグライブラリに属する<c:if>要素の用法を理解しましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson29/p2_quest1.jsp

① c:if

Coreタグライブラリで単純分岐を表現するには、<c:if>要素を使用します。<c:if>要素では、条件式をtest属性で指定します。

② ${param['interest'] == 1}

test属性の条件式は、式言語で指定します。リクエスト情報が1であるかどうかで処理を分岐します。なお、演算子「==」は、エイリアスとして「eq」を使用してもかまいません。

③ **${param['interest'] != 1}**

<c:if>要素にはいわゆるelseブロックが存在しません。リクエスト情報が1で**ない**ことを表す条件式を記述する必要があります。なお、演算子「!=」はエイリアスとして「ne」を使用してもかまいません。

A3

Coreタグライブラリに属する<c:choose>要素の用法を理解しましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson29/p3_quest1.jsp

① **c:choose**

条件式が配下に記述されていることから、多岐分岐を表していることがわかるはずです。Coreタグライブラリで多岐分岐を表現するには、<c:choose>要素を使うのでした。Java言語のswitch...case命令と混同して、<c:switch>としないようにしてください。

② **c:when**

<c:choose>要素で個別の条件式を表すには、<c:when>要素を使用します。<c:when>要素は、<c:choose>要素の配下に任意の数だけ記述できます。

③ **c:otherwise**

<c:otherwise>要素は、<c:choose>要素の配下ですべての条件式に合致しなかった場合に実行される内容を記述します。<c:choose>要素の配下に1回だけ記述できます。

A4

<c:forEach>要素のもう1つの用法について理解する問題です。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson29/p4_sum.jsp

① **begin**

<c:forEach>要素でループ回数を指定する場合、開始値をbegin属性で、終了値をend属性で指定します。増分はデフォルトで1ですが、step属性で変更することも可能です。

ループ内で変数値を参照する必要がない場合、var属性は省略可能です。

② **${i % 2 != 0}**

ここではループ変数が奇数の場合にのみ値を加算しますので、test属性には「ループ変数が2で割り切れない場合」を表す条件式を記述します。エイリアスを利用して、${i mod 2 ne 0}と書いても正解です。

③　${result + i}

一般的なJavaのコードでは当たり前の式ですが、式言語になったことで、若干戸惑ったという方もいたかもしれません。このように、自分自身に値を足し込むような表現も可能です。

LESSON 30

LESSON 29に引き続き、JSPページに含まれるスクリプティング要素をJSTL＋式言語に置き換える問題です。LESSON 29に比べると扱う要素の数も多くなっていますが、なんら惑わされることはありません。元のコードと比べながら、対応するアクションタグを当てはめてみてください。

A1

Databaseタグライブラリによるデータベース更新の手法を理解しましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson30/p1_new.jsp

① fmt

JSTLを利用する場合には、あらかじめ@taglibディレクティブで使用を宣言しておく必要があります。prefix属性には、カスタムタグで使用する接頭辞を指定してください。I18nタグライブラリの接頭辞は一般的に「fmt」です。

② fmt:requestEncoding

マルチバイト文字を含むリクエスト情報を受け取る場合には、あらかじめ<fmt:requestEncoding>要素でリクエストの文字エンコーディングを指定する必要があります。直後のvalue属性から容易に想像できるはずです。

③ sql:update

INSERTなど、結果を返さないSQL命令を発行する場合には、<sql:update>要素を使用します。

④ sql:param

SQL命令に対してパラメーターを動的に当てはめるには、<sql:param>要素を使用します。<sql:param>要素では、パラメーターの順番（インデックス）を明示的に指定できませんので、注意してください。

⑤ ${!empty err}

LESSON 29でも学んだように、<c:catch>要素で発生した例外処理の定型的な記述です。例外情報を格納した変数の中身が空であるかによって、例外が発生したかどうかを確認します。

A2 Databaseタグライブラリによるデータベース検索／更新の手法を理解しましょう。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson30/p2_edit.jsp

① sql:setDataSource

<sql:setDataSource>要素は、dataSource属性で指定されたデータソース名に基づいて、データソースへの接続を確立します。

② ${pub == row.publish}

<c:forEach>要素で取り出した出版社名と、データベースから実際に取得した出版社名とが等しい場合に、selected属性を出力します。選択ボックスのデフォルト値を設定する場合の典型的な手法ですので、しっかりとおさえておきたいところです。

③ c:catch

例外処理を定義します。少々わかりにくいかもしれませんが、var属性で指定された変数errが、38行目の<c:if>要素と対応していることがわかれば、比較的容易に類推できるのではないでしょうか。

④ c:forEach

取得した行の数だけループを繰り返します。begin / endなど、後続の属性から容易に類推できるでしょう。

⑤ c:choose

リクエスト情報deleteN（Nは連番）の値が1（チェックされている）であるかどうかによって、更新／削除いずれの処理を実行するかを決定します。配下に、<c:when> / <c:otherwise>要素があることから類推できるはずです。

LESSON 31

A JSTL＋式言語の復習です。これまでの知識をフル動員しながら、空欄を埋めていってください。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson31/P_MvcServlet1

① taglib

タグライブラリを利用する場合には、@taglibディレクティブであらかじめ宣言しておく必要があります。

② ${requestScope['list']}

LESSON 28のP_MvcServlet1.javaもあわせて確認してみましょう。リクエスト属性を取得するには、暗黙オブジェクトrequestScopeを使用します。スコープを明示しない${list}でも可能ですが、スコープ間の名前の重複などによる不具合を避けるためにも、原則、スコープは明示することをお勧めします。

③ fmt:formatNumber

与えられた数値データを整形するのは、<fmt:formatNumber>要素の役割です。value属性で整形対象の値を、pattern属性でフォーマット文字列を指定します。

④ fmt:formatDate

与えられた日付／時刻データを整形するのは、<fmt:formatDate>要素の役割です。value属性で整形対象の値を、pattern属性でフォーマット文字列を指定します。

⑤ yyyy年MM月dd日

フォーマット文字列を利用することで、日付データから自由に必要な値を取り出せます。フォーマット文字列で利用可能な識別子については、P.182もあわせて参照してください。

LESSON 32

カスタムタグの基本的な構文を確認する問題です。コード量も多くなってきて、なかなか頭の整理も難しくなってきたと思います。単に問題を解いて終わりとするのではなく、是非、コードを概観し、全体の流れを把握するようにして見てください。

A1

p1_text2table.jspは/lesson32フォルダーに、コンパイル済みのP1_Text2TableTag.classは/WEB-INF/classes/to/msn/wings/lesson32フォルダーに、それぞれ格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson32/p1_text2table.jsp

① win

@taglibディレクティブのprefix属性には、カスタムタグのプレフィックス（接頭辞）を指定します。接頭辞はp1_text2table.jspの10行目から確認できます。

② TagSupport

タグハンドラークラスは、TagSupport / BodyTagSupportなどの基本クラスを継承して定義します。本体を持たないタグであれば、TagSupportクラスを継承するのが一般的です。

③ public void setPath(String path) { this.path = path; }

カスタムタグの属性を宣言するには、セッターメソッドを利用します。前後からpathプロパティのセッターメソッドがないので、宣言しておきます。

④ doStartTag

開始タグの挙動を定義するのは、doStartTagメソッドの役割です。doStartTagメソッドは、戻り値として開始タグ処理後の挙動を表す定数を返す必要があります。

⑤ short-name

「win」という値から、タグライブラリの略称を設定していることが類推できます。略称は、そのままカスタムタグの接頭辞になるのでした。

⑥ http://www.wings.msn.to/WingsTagLibs-1.0

<uri>要素は、タグライブラリを一意に表す任意のURIを指定します。@taglibディレクティブのuri属性に対応しています。

⑦ Text2Table

<tag>－<name>要素には、カスタムタグの名前を記述します。p1_text2table.jspの10行目から類推できます。

⑧ **to.msn.wings.lesson32.P1_Text2TableTag**

<tag-class>要素には、タグハンドラークラスの完全修飾名を指定します。クラス名だけではダメですので、要注意です。

⑨ **path**

<attribute>－<name>要素は、属性の名前を表します。<attribute>要素はいくつか列記されていますが、抜けているのはpath属性です。

⑩ **true**

<attribute>－<required>要素は、属性が必須であるかどうかを表します。問題文からpath属性は必須なので、trueとしておきます。

A2 p2_textwriter1.jsp、p2_textwriter2.jspは/lesson32フォルダーに、コンパイル済みのP2_TextWriterTag.classは/WEB-INF/classes/to/msn/wings/lesson32フォルダーに、それぞれ格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson32/p2_textwriter1.jsp

① **BodyTagSupport**

タグハンドラークラスは、TagSupport / BodyTagSupportなどの基本クラスを継承しなければなりません。タグが配下にテキストを持つような場合には、BodyTagSupportクラスを継承します。

② **setPath**

アクセサーメソッドです。path属性への設定を表します。

③ **getServletContext**

ServletContextオブジェクト（暗黙オブジェクトapplicationに該当するオブジェクトです）は、タグハンドラー内ではpageContextオブジェクトのgetServletContextメソッドを介して取得できます。

④ **attribute**

カスタムタグの属性情報を設定するのは、<attribute>要素の役割です。<attribute>要素には、属性名や属性の必須／任意、式（Expression）が使用できるかなどを宣言します。

⑤ **path**

<attribute>－<name>要素には、属性名を記述します。

LESSON 33

A タグファイルの基本的な利用方法を問う問題です。dataGrid.jspは/lesson33フォルダーに、DataGrid.tagは/WEB-INF/tagsフォルダーに、それぞれ格納しています。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson33/dataGrid.jsp

① tagdir

タグファイルを利用する場合には、@taglibディレクティブのuri属性の代わりに、tagdir属性を指定する必要があります。

② tag

@tagディレクティブは、タグファイルの処理方法を宣言します。その性質上、タグファイルの先頭に記述するのが一般的です。

③ attribute

タグファイル内部で参照可能な属性を宣言するには、@attributeディレクティブを使用します。@attributeディレクティブでは、属性名（name属性）や必須かどうか（required属性）などを指定します。

④ taglib

タグファイルからJSTLやその他のタグライブラリを利用する場合には、@taglibディレクティブを使用します。これは、一般的なJSPページにおける記述と同じです。

⑤ ${row[col]}

コードの14～20行目では、<c:forEach>要素が2重ループになっている点に注目してください。外側のループで1レコードを、内側のループで個々のフィールドを出力していることが理解できれば、解答は簡単でしょう。ここでは、columnNamesプロパティを介して取得した個々の列名をキーに、フィールド値にアクセスしています。

LESSON 34

A コンパイル済みのP_DbLogFilter.classは/WEB-INF/classes/to/msn/wings/lesson34フォルダーに、web.xmlは/WEB-INFフォルダーに、それぞれ格納しています。/lesson33フォルダー配下のファイルにアクセスすることで、MySQLのlogsテーブルにログ情報が記録されます。logsテーブルの内容は、mysqlクライアントからSELECT文を発行するなどして確認できます。

```
> mysql -u jsp10 -p
password: ****      ⬅（パスワードjsp10を入力）
mysql> USE jsp10;
mysql> SELECT id,access_date,access_time,path,ip_address FROM logs;
```

① implements Filter

フィルター機能を提供するクラスは、Filterインターフェイスを実装していなければなりません。

② doFilter

フィルターの実処理を定義するのは、doFilterメソッドです。

③ chain

FilterChainオブジェクトは、現在の要求に対して実行されるべきすべてのフィルターを管理します。FilterChainオブジェクトchainのdoFilterメソッドは、次のフィルターが存在する場合はそのフィルターを呼び出し、次のフィルターがない場合には、要求のあったもともとのコンテンツを呼び出します。

④ web-app

設定ファイルweb.xmlのルート要素は<web-app>要素です。

⑤ <filter-class>to.msn.wings.lesson34.P_DbLogFilter</filter-class>

<filter>要素は、フィルター機能を記述したクラスとフィルター名とを紐づけます。フィルター機能を記述したクラスを定義するのは、<filter-class>要素です。

⑥ <filter-name>P_DbLogFilter</filter-name>

<filter-mapping>要素は、フィルター名とフィルターの適用範囲を定義します。フィルター名を定義するのは、<filter-name>要素です。

⑦ dispatcher

フィルターをどのタイミングで実行するかを決定します。複数のタイミングを指定したい場合は、<dispatcher>要素を複数列記してください。

LESSON 35

デプロイメントディスクリプターに関する基本的な知識を問う問題です。web.xmlの構造をすべて記憶する必要はまったくありませんが、主要な設定項目は記憶に留めておくことで、きっと役に立つはずです。

A1

以下表の中から3つ以上挙げられれば正解です（ただし、以下は、本書で扱った要素だけですので、実際にはもっと多くの設定要素が存在します）。

▼デプロイメントディスクリプターにおける主要な設定項目

要素名	概要
<context-param>	初期化パラメーターの設定
<error-page>	エラーページの指定
<filter>	フィルタークラスを定義
<filter-mapping>	フィルタークラスの適用範囲をマッピング
<jsp-config>	JSPページに関する共通的な情報を設定
<login-config>	ログイン手段を指定
<security-constraint>	認証の適用範囲を定義
<security-role>	使用するロール名を宣言
<servlet>	サーブレットクラスを定義
<servlet-mapping>	サーブレットクラスと呼び出しのパスをマッピング
<session-config>	セッション情報を定義
<welcome-file-list>	ウェルカムページを指定

A2

認証に関する基本的な設定方法を復習します。適宜、web.xmlを書き換え、以下のURLからアクセスして確認してみてください（配布サンプルのweb.xmlでは、他のサンプルに影響が出ないよう、基本認証の設定はコメントアウトしています）。

練習問題にアクセスするためのURL：
http://localhost:8080/jsp10/lesson35/admin.jsp

① security-constraint

認証の範囲を指定するのは<security-constraint>要素の役割です。配下に<web-resource-collection> / <auth-constraint>などの要素を含むことができます。

② /lesson35/admin.jsp

<url-pattren>要素には、認証を適用するリソースの範囲を指定します。フォルダー全体を制限の対象にする場合は「*」を含めることもできます。

③ <role-name>manager</role-name>

アクセスを認める権限名を指定します。ロール名は、tomcat-users.xmlにおける<user>

要素のroles属性と対応関係にある必要があります。

④ BASIC

<auth-method>要素にはログイン方法を指定します。BASIC（基本認証）の他にFORM、DIGEST、CLIENT-CERTを指定できます。

⑤ security-role

<security-constraint>要素配下で利用するロール名は、あらかじめ<security-role>要素で宣言しておく必要があります。複数のロール名を宣言する場合には、<security-role>要素そのものを並列に記述する必要があります。<security-role>要素の配下に複数の<role-name>要素を記述することはできません。