～Webアプリケーション開発講座～

JSPとMVCモデル

UZUZ COLLEGE ウズウズカレッジ

～ JavaScriptの外部ファイル化 ～

復習

＜Sample_5_07_2（ログイン画面）＞

```
<html>
<head>
  <title>ログイン画面</title>
</head>
<body>
  <h1>ログインSample（入力チェック②）</h1>
  <form action="ExecuteLogin" method="post">
    <p>ユーザーID：
      <br>
      <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
    </p>
    <p>パスワード：<br>
      <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
    </p>
    <input type="submit" value="ログイン" id="ID_SUBMIT" >
  </form>
  <script type = "text/javascript">
    var elmSubmit = document.getElementById("ID_SUBMIT");

    elmSubmit.onclick = function(){
      var elmUserId   = document.getElementById("ID_USER_ID");
      var elmPassword = document.getElementById("ID_PASSWORD");
      var canSubmit = true;
      if(elmUserId.value == "" || elmPassword.value == ""){
        alert("入力漏れの項目があります。");
        canSubmit = false;
      }
      return canSubmit;
    }
  </script>
</body>
</html>
```

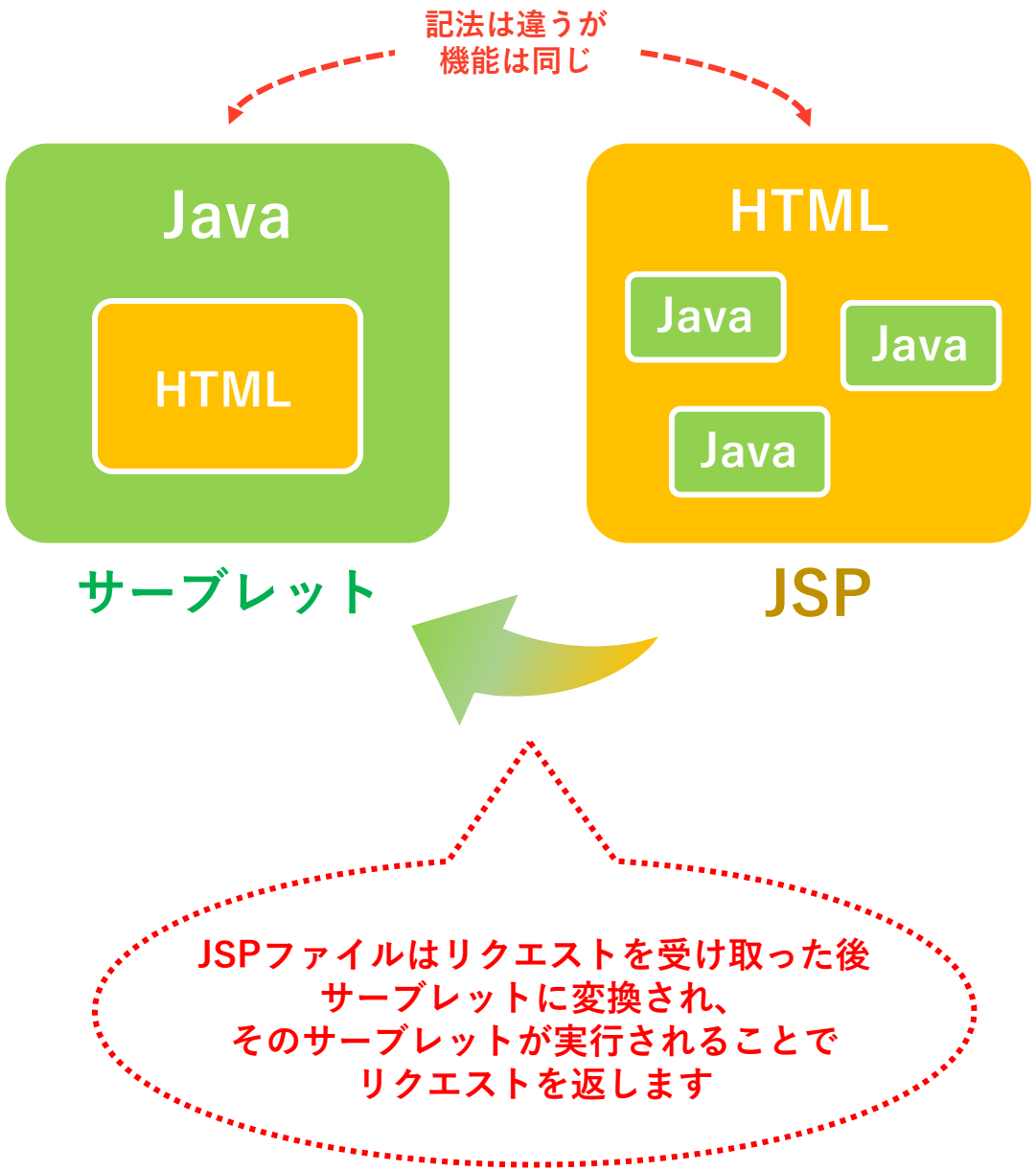
＜Sample_5_07_3（ログイン画面）＞

```
<html>
<head>
  <title>ログイン画面</title>
</head>
<body>
  <h1>ログインSample（入力チェック③）</h1>
  <form action="ExecuteLogin" method="post">
    <p>ユーザーID：
      <br>
      <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
    </p>
    <p>パスワード：<br>
      <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
    </p>
    <input type="submit" value="ログイン" id="ID_SUBMIT" >
  </form>
  <script type="text/javascript" src="js/brank-check.js"></script>
</body>
</html>
```

＜外部ファイル（brank-check.js）＞

```
var elmSubmit = document.getElementById("ID_SUBMIT");
elmSubmit.onclick = function(){
  var elmUserId   = document.getElementById("ID_USER_ID");
  var elmPassword = document.getElementById("ID_PASSWORD");
  var canSubmit = true;
  if(elmUserId.value == "" || elmPassword.value == ""){
    alert("入力漏れの項目があります。");
    canSubmit = false;
  }
  return canSubmit;
}
```

～ JSP ～



≪JSP≫

□**JSP（Java Server Page）**はJavaサーブレットを拡張した技術です。JSPで作成されたプログラムファイル（JSPファイル）はサーバーに配置され、リクエストを受け取るとレスポンスとしてHTMLデータを返します。つまり、**機能としてはサーブレットと大差ありません。**

違いはその記述法にあり、サーブレットはJavaのソースコード中でHTMLをデータとして扱いますが、**JSPはHTMLの中にJavaのソースコードを埋め込む形で記述します。**

HTMLを一行ずつ文字列として扱うサーブレットと異なり、HTML出力処理の記述が非常に楽になります。

□JSPファイルはリクエストされるとサーバーで以下のように処理されてリクエストが返ります。

①JSPファイルがURL指定されるなどしてリクエストされる。

②サーバーはリクエストされたJSPファイルをサーブレットファイルに変換 → コンパイル → 実行 する。

③実行されたサーブレットがHTMLデータをレスポンスとしてクライアントに返す。

～ サンプルを動かしてみよう ～

手順

（１） eclipse上に新しい動的Webプロジェクトを作成する。
プロジェクト名：Sample_5_08_1

（２）プロジェクトの srcパッケージ直下に workパッケージを作成する。

（３）配布のSample_5_08_1フォルダからソースコードを取得して
workパッケージ直下にインポートする。

（４）同じくweb.xmlをドライブから取得して置き換える。
WebContent ＞ WEB-INF ＞ web.xml

（５）サーバーを起動してSearchSurveyBySatisfactionLevel.java
を起動する。

～ サンプルを動かしてみよう ～

手順

（１） eclipse上に新しい動的Webプロジェクトを作成する。
プロジェクト名：Sample_5_08_2

（２）プロジェクトの srcパッケージ直下に workパッケージを作成する。
（３）配布のSample_5_08_2フォルダからソースコードを取得して
workパッケージ直下にインポートする。

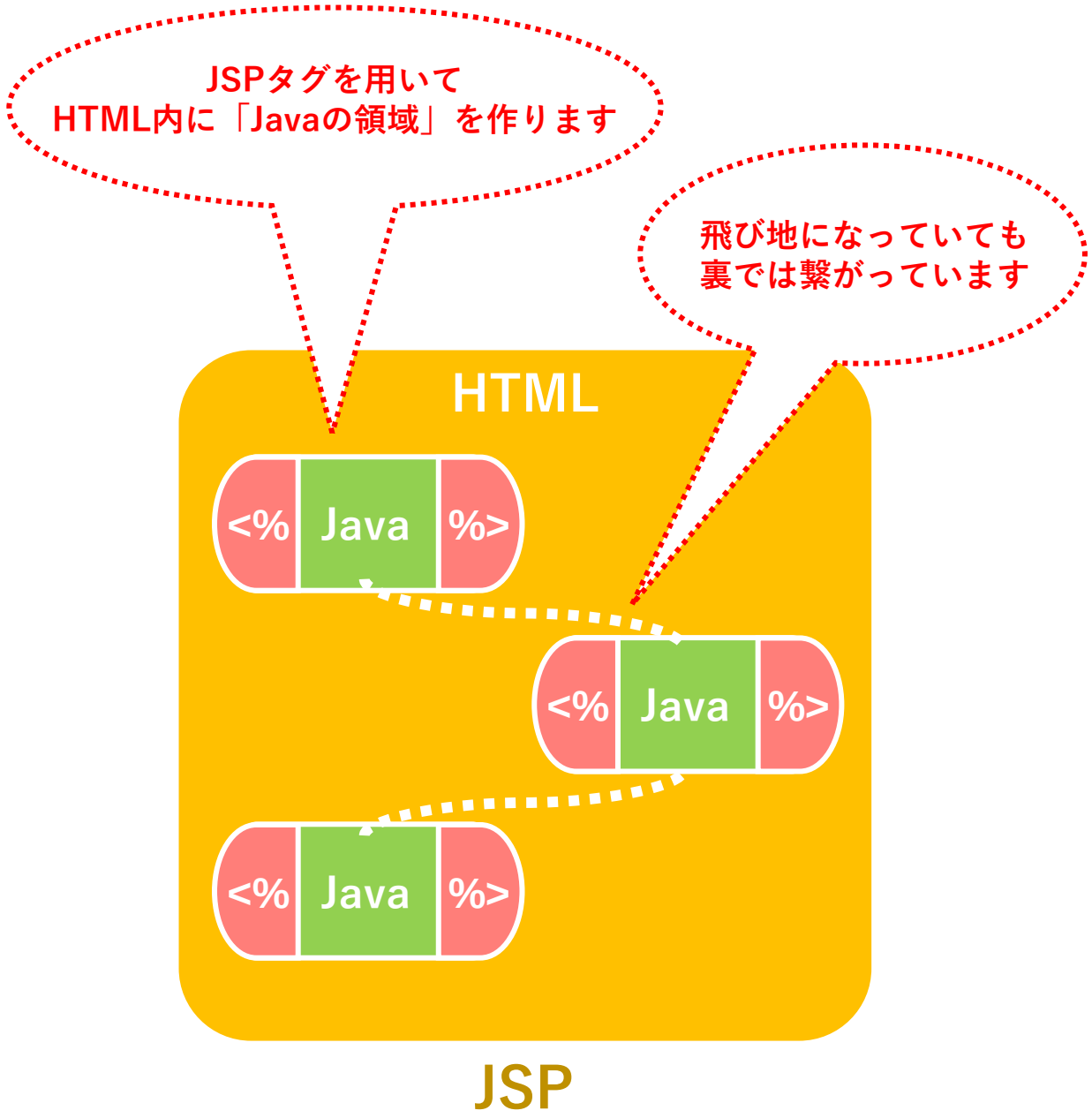
（５）プロジェクトの WebContent直下に jspフォルダを作成する。
（６）配布のSample_5_08_2フォルダから以下のJSPファイルを取得して
jspフォルダ直下にインポートする。

search_survey_by_satisfaction_level.jsp
show_survey_by_satisfaction_level.jsp

（７）配布のweb.xmlを以下に格納する。
WebContent ＞ WEB-INF ＞ web.xml

（８）サーバーを起動してsearch_survey_by_satisfaction_level.jsp
を起動する。

～ JSP ～



≪JSPの文法と扱い≫

□JSPでは**特殊タグ（JSPタグ）**を用いることでHTML**内に**Javaのソースコードを埋め込むことができるようになります。
例えば　<%　と　%>　で囲われた部分はJavaの領域とみなされ、Javaのソースコードと同様に命令文の記述が可能になります。

□JSPタグは一つのファイル内で複数定義することが可能で、**それらは同じスコープ内の処理として扱われます。**
一見飛び地に見えるが裏では繋がっているイメージです。
このJSP内のJavaを管理するスコープを**ページスコープ**と言います。

□JSPファイルは「ファイル名.jsp」という名前で定義し、eclipseの動的Webプロジェクトにおいては WebContentフォルダ の下で管理します。この WebContentフォルダ は外部からアクセス可能な領域で、ここに配置しておけば外部からのリクエストを受け付けることができます。
JSPファイルがリクエストを受け付けるためのURLは以下です。

http://サーバ名/アプリケーション名/WebContentからの相対パス

＜例＞ http://localhost:8080/ServletSample09_1/jsp/Login.jsp

□JSPでJavaを扱う際、始めからあるものとして宣言せずに使用できるオブジェクトが存在し、これを**暗黙オブジェクト**と言います。
request、response、out、sessionなど、サーブレットで頻繁に使うオブジェクトが多く含まれており、サーブレットよりもスッキリしたコードが書けるようになります。

～ JSP ～

各スクリプトレットは同じスコープ（ページスコープ）で管理されている。
つまり、飛び地に見えるが裏でつながっている。

JSP式でHTML中に値を埋め込む。
この領域も他のJSPタグと裏で
繋がっている。

```
<%
for (int i = 0; i < surveyDtoList.size(); i++) {

    //本ループのターゲットとなるDTO（1行分のアンケート結果）をセット
    SurveyDto dto = surveyDtoList.get(i);

%>
      <tr>
        <td><%= dto.getName() %></td>
        <td><%= dto.getAge() %></td>
<%
    //出力内容の分岐（性別）
    switch( dto.getSex() ){
        case 1:
%>
        <td>オス</td>
<%
            break;
        case 2:
%>
        <td>メス</td>
<%
            break;
    }
%>
<%
    //出力内容の分岐（満足度）
    switch( dto.getSatisfactionLevel() ) {
        case 1:
%>
        <td>とても不満</td>
```

＜show_survey_by_satisfaction_level.jsp＞

ページ
スコープ

～ JSP ～

代表的なJSPタグ

タグ	名称	記述内容	解説
<% ～ %>	スクリプトレット	Javaのソースコード全般	囲われている領域にJavaの処理群を記述することが可能です。 1つのJSPファイル内で複数定義することが可能で、それらは同じスコープ内の処理として扱われます。（あるスクリプトレットで宣言した変数はそれ以降に登場するスクリプトレット内でも使用可能） ここで宣言された変数／定数／関数はローカルなものであり、リクエスト処理が終わったら破棄されます。
<%@ page ～ %>	pageディレクティブ	属性名="設定値"	レスポンスのcontentType、文字コード、使用言語など、JSPファイル全体の設定情報を記述することが可能です。 また、import属性を用いることでJavaのクラスをインポートすることも可能です。以下のAPIはデフォルトでインポートされています。 **java.lang.*** **javax.servlet.*** **javax.servlet.jsp.*** **javax.servlet.http.***
<%-- ～ --%>	コメント	コメント	JSPファイルにおけるコメントを記述することが可能です。
<%= ～ %>	式	値を取得できるJavaのソースコード	囲われている領域に値を取得できるJavaのソースコード（変数、演算式、メソッドなど）を記述することで、取得した値をHTMLに埋め込みます。
<%! ～ %>	宣言	変数宣言 定数宣言 関数宣言	囲われている領域で宣言された変数／定数／関数はグローバルなものとしてサーバーに居残り続けます。 リクエスト数や決まりきった定数など、複数のリクエストで共通的に用いるようなものを定義します。複数のリクエストで同時にアクセスされる可能性もあるため設計・実装には注意が必要です。

～ JSP ～

暗黙オブジェクト一覧

オブジェクト名	実装クラス（型）	説明
request	javax.servlet.http.HttpServletRequest	クライアントからのリクエストに関する情報を管理。
response	javax.servlet.http.HttpServletResponse	クライアントに送るレスポンスに関する情報を管理。
session	javax.servlet.http.HttpSession	セッションスコープに関する情報を管理。
application	javax.servlet.ServletContext	コンテナ情報やアプリケーションスコープに関する情報を管理。
out	javax.servlet.jsp.JspWriter	レスポンスとしてクライアントに返す情報を管理。
page	java.lang.Object	当該のJSPファイルに関する情報を管理。
pageContext	javax.servlet.jsp.PageContext	ページスコープ内で共有する情報を管理。
config	javax.servlet.ServletConfig	web.xmlに関する情報を管理。
exception	java.lang.Throwable	例外に関する情報を管理。

～ JSP ～

doGet／doPost
メソッドは不要

ファイルや通信の設定は
pageディレクティブで
まとめてコンパクトに！

```java
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    //レスポンス（出力データ）の文字コードを設定
    response.setContentType("text/html;charset=UTF-8");  //文字コードをUTF-8で設定

    //出力用のストリームの取得
    PrintWriter out = response.getWriter();

    //HTML文書（満足度別アンケート結果検索画面）の出力

    out.println("<html>                                                                ");
    out.println("<head>                                                                ");
    out.println("  <title>満足度別アンケート結果検索</title>                           ");
    out.println("</head>                                                               ");
    out.println("<body>                                                                ");
    out.println("  <h2>満足度別アンケート結果検索</h2>                                ");
    out.println("  <form action=¥"ShowSurveyBySatisfactionLevel¥" method=¥"post¥">     ");
    out.println("    <p> 満足度：                                                      ");
    out.println("      <select name=¥"SATISFACTION_LEVEL¥">                            ");
    out.println("        <option value=¥"5¥">とても満足</option>                       ");
    out.println("        <option value=¥"4¥">満足      </option>                       ");
    out.println("        <option value=¥"3¥">普通      </option>                       ");
    out.println("        <option value=¥"2¥">不満      </option>                       ");
    out.println("        <option value=¥"1¥">とても不満</option>                       ");
    out.println("      </select>                                                       ");
    out.println("    </p>                                                              ");
    out.println("    <input type=¥"submit¥" value=¥"検索¥">                            ");
    out.println("  </form>                                                             ");
    out.println("</body>                                                               ");
    out.println("</html>                                                               ");
}
```

エスケープ処理は不要

出力ストリーム
は不要

＜SearchSurveyBySatisfactionLevel.java＞
（Sample_5_08_1プロジェクト）

Javaの文字列としてではなく
HTML文書をそのまま記載

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

<%--
-------------------------------------------------------------------------------
■■■ファイル名：search_survey_by_satisfaction_level.jsp■■■
概要：JSP
詳細：HTML文書（満足度別アンケート結果検索画面）を出力する。
-------------------------------------------------------------------------------
--%>

<html>
<head>
  <title>満足度別アンケート結果検索</title>
</head>
<body>
  <h2>満足度別アンケート結果検索</h2>
  <form action="./show_survey_by_satisfaction_level.jsp" method="post">
    <p> 満足度：
      <select name="SATISFACTION_LEVEL">
        <option value="5">とても満足</option>
        <option value="4">満足      </option>
        <option value="3">普通      </option>
        <option value="2">不満      </option>
        <option value="1">とても不満</option>
      </select>
    </p>
    <input type="submit" value="検索">
  </form>
</body>
</html>
```

＜search_survey_by_satisfaction_level.jsp＞
（Sample_5_08_2プロジェクト）

～ JSP ～

パッケージ宣言
不要

JSPでは
使用しない

デフォルトで
インポートされている

```java
package work;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**-------------------------------------------------------------------------*
 *■■■ShowSurveyBySatisfactionLevelクラス■■■
 *概要：サーブレット
 *詳細：「survey」テーブルから条件に合致するデータを抽出し、HTML文書（回答一覧画面）を出力する。
 *      ＜条件＞満足度が選択されたものと一致
 *-------------------------------------------------------------------------**/
public class ShowSurveyBySatisfactionLevel extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ShowSurveyBySatisfactionLevel() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        //レスポンス（出力データ）の文字コードを設定
        response.setContentType("text/html;charset=UTF-8");

        //リクエストパラメータを取得
        String receivePrmSatLv     = request.getParameter("SATISFACTION_LEVEL");          //リクエ

        //「survey」テーブルからデータを全件抽出
        List<SurveyDto> surveyDtoList = new ArrayList<SurveyDto>();                      //アンケ
        BusinessLogic   logic         = new BusinessLogic();
        surveyDtoList = logic.executeSelectSurveyBySatisfactionLevel( receivePrmSatLv ); //アンケ

        //出力用のストリームの取得
        PrintWriter out = response.getWriter();

        //HTML文書（回答一覧画面）を出力
        out.println(                "<html>
        out.println(                "  <head>
```

JSPコメント

pageディレクティブ
で定義

JSPでは
使用しない

＜ShowSurveyBySatisfactionLevel.java＞
（Sample_5_08_1プロジェクト）

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

<%@ page import="java.util.ArrayList" %>
<%@ page import="java.util.List"      %>
<%@ page import="work.BusinessLogic"  %>
<%@ page import="work.SurveyDto"      %>

<%--
-------------------------------------------------------------------------
■■■ファイル名：show_survey_by_satisfaction_level.jsp■■■
概要：JSP
詳細：「survey」テーブルから条件に合致するデータを抽出し、HTML文書（回答一覧画面）を出力
      ＜条件＞満足度が選択されたものと一致
-------------------------------------------------------------------------
--%>

<%
//リクエストパラメータを取得
String receivePrmSatLv     = request.getParameter("SATISFACTION_LEVEL");          //リク

//「survey」テーブルからデータを全件抽出
List<SurveyDto> surveyDtoList = new ArrayList<SurveyDto>();                      //アン
BusinessLogic   logic         = new BusinessLogic();
surveyDtoList = logic.executeSelectSurveyBySatisfactionLevel( receivePrmSatLv ); //アン
%>
```

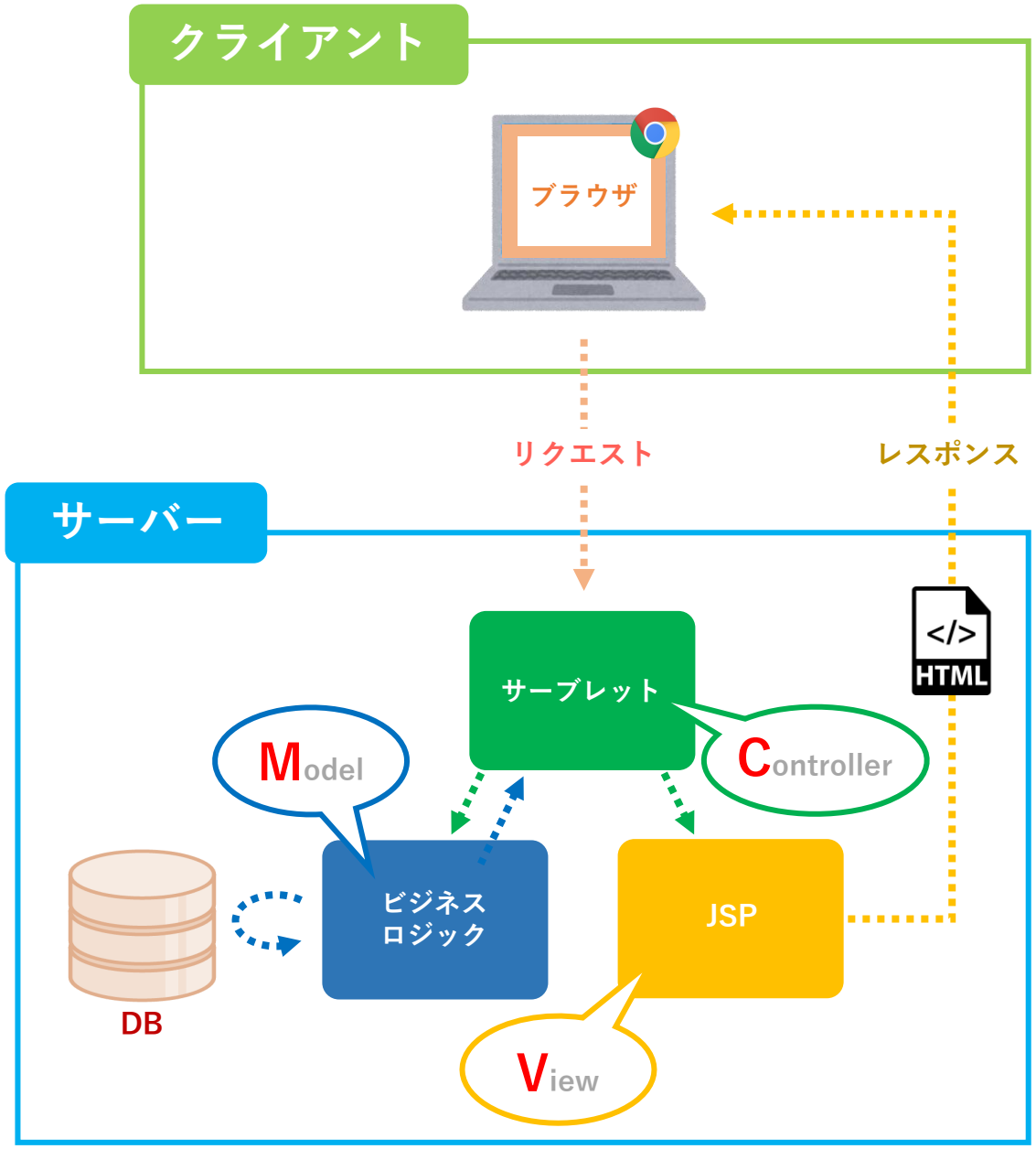
JSPファイルからはパッケージ外
アクセスになるため必要

暗黙オブジェクトであるため
定義もなしに使用可能

＜show_survey_by_satisfaction_level.jsp＞
（Sample_5_08_2プロジェクト）

スクリプトレットで
Javaのソースコードを
そのまま記述

～ MVCモデル ～



≪MVCモデル≫

□MVCモデルはシステム設計方針（デザインパターン）の一種です。
ユーザーがブラウザを通して操作するタイプのWebアプリケーションを構築するうえで長年採用されています。

□MVCは「Model」「View」「Controller」の頭文字を取った単語で、クライアントとの通信が発生するプログラムをこの3つの役割ごとに分割・管理しようというものです。

DAOと同じように役割ごとにソースコードおよび責任範囲を分離することで管理やメンテナンスがしやすくなり、分業も可能になるため開発スピードもアップします。

Model … 主にビジネスロジックを指します。Controllerの指示を受け、複雑な計算やDBアクセスといったデータ周辺の具体的な処理を行います。

View … 主にユーザーインターフェースに関わる処理を管理します。Controllerの指示を受け、HTML周辺の画面情報の生成、クライアントへの送信を行います。主にJSPが担います。

Controller … リクエストを受け取ってからの一連の処理の流れを管理します。必要に応じてModelやViewを呼び出します。mainメソッドのように具体的な処理は他に任せ、あくまで処理の流れだけを記述します。主にサーブレットが担います。

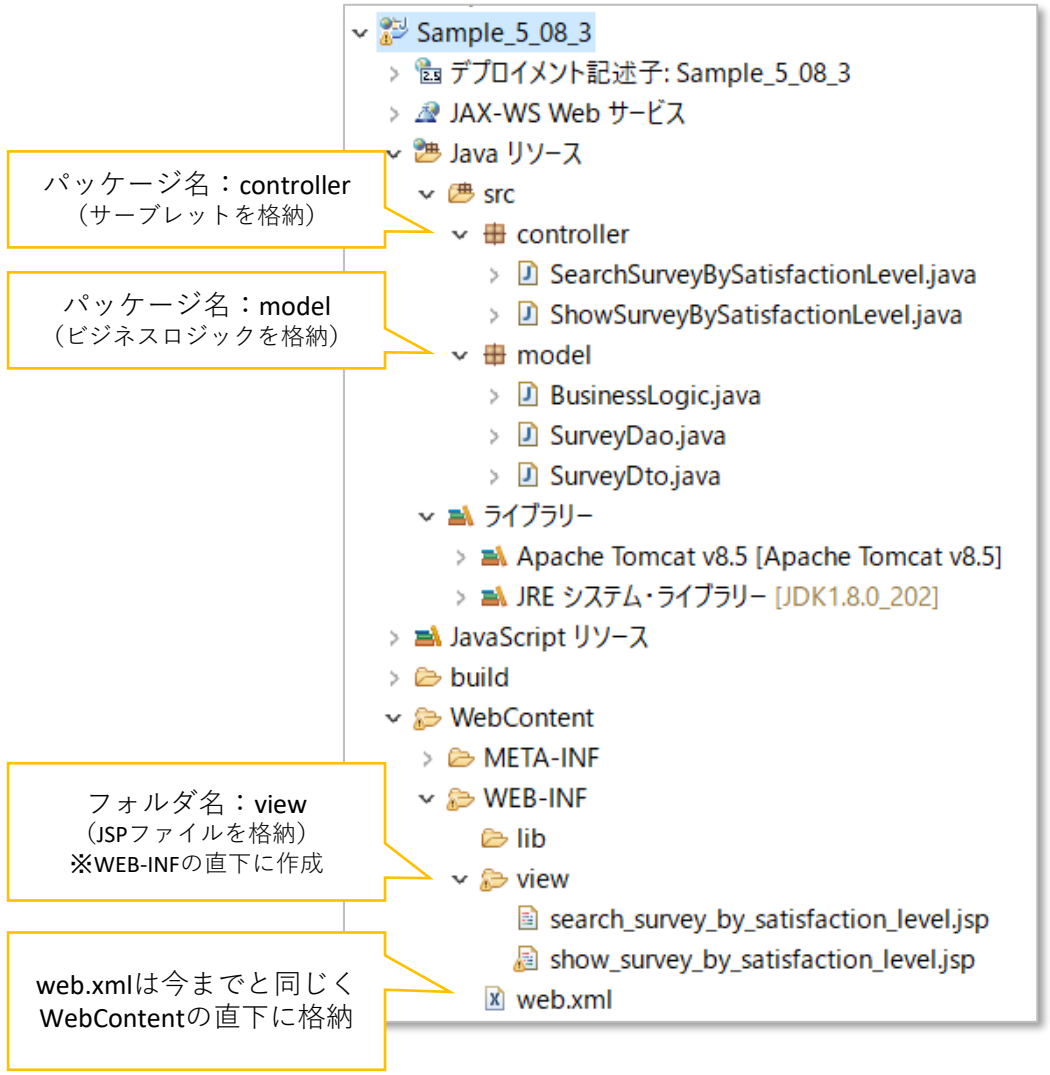
～ サンプルを動かしてみよう ～

手順

（１） eclipse上に新しい動的Webプロジェクトを作成する。
プロジェクト名： Sample_5_08_3

（２）配布のSample_5_08_3フォルダからソースコードを取得し、右画像を参考にパッケージ/フォルダ作成＆ソースコードの配備を行う。

（３）サーバーを起動してSearchSurveyBySatisfactionLevelを起動する。



～ サンプルを動かしてみよう ～

≪Sample_5_08_1≫

ShowSurveyBySatisfactionLevel.java

＜処理①＞
リクエストの取得

＜処理②＞
BusinessLogicを起動
（一覧データの取得）

＜処理③＞
HTMLの生成＆レスポンス送信

≪Sample_5_08_2≫

show_survey_by_satisfaction_level.jsp

＜処理①＞
リクエストの取得

＜処理②＞
BusinessLogicを起動
（一覧データの取得）

＜処理③＞
HTMLの生成＆レスポンス送信

≪Sample_5_08_3≫

ShowSurveyBySatisfactionLevel.java

＜処理①＞
リクエストの取得

＜処理②＞
BusinessLogicを起動
（一覧データの取得）

バトンタッチ！

show_survey_by_satisfaction_level.jsp

＜処理③＞
HTMLの生成＆レスポンス送信

～ サンプルを動かしてみよう ～

リクエスト

≪Sample_5_08_3≫

Controller

ShowSurveyBySatisfactionLevel.java

＜処理①＞
リクエストの取得

＜処理②＞
BusinessLogicを起動
（一覧データの取得）

Model

BusinessLogic.java
SurveyDao.java

一覧データの取得

DB

バトンタッチ！

View

show_survey_by_satisfaction_level.jsp

＜処理③＞
HTMLの生成＆レスポンス送信

レスポンス

～ サンプルを動かしてみよう ～

リクエスト

≪Sample_5_08_3≫

ShowSurveyBySatisfactionLevel.java

＜処理①＞
リクエストの取得

＜処理②＞
BusinessLogicを起動
（一覧データの取得）

どうやってリクエストや
データベースから取得したデータを
受け渡そうか？

バトンタッチ！

show_survey_by_satisfaction_level.jsp

＜処理③＞
HTMLの生成＆レスポンス送信

レスポンス

～ Sample_5_06_1は不完全・・・ ～

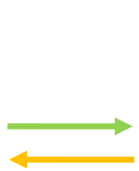
復習

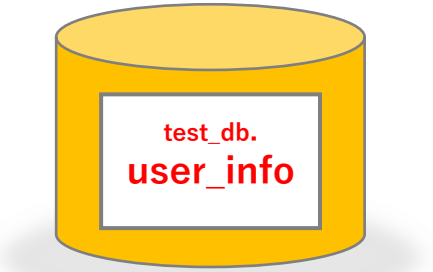
リクエスト

ExecuteLogin

doPost

DBからのデータ取得
&
リダイレクト
（ Home?user_name=MOCO ）





MySQL

リダイレクト

Home

doGet

リクエストパラメータの取得
&
画面の出力

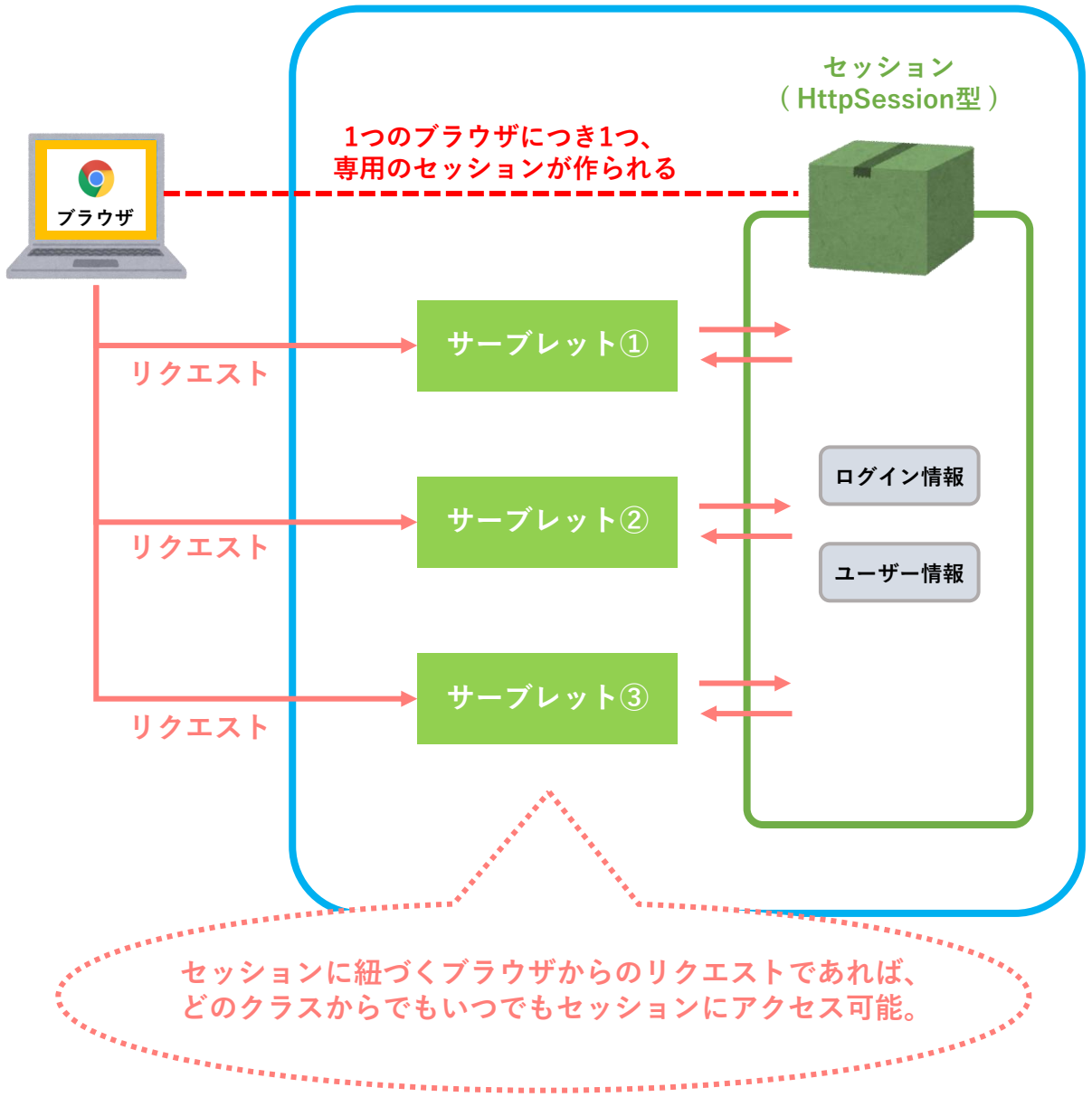
レスポンス

サーブレットからサーブレットに
データを渡すため、
URLにリクエストパラメータを埋め込んで
リダイレクトする！

サーバーの外は危険がいっぱい！
大事なデータを外に出したくない！

～ Cookie／セッションの必要性 ～

復習



≪セッション≫

□同じブラウザからのアクセスに限り、リクエストをまたいで使用することのできるインスタンスのことを**セッション**と言います。
ログイン情報やユーザー情報など、リクエストをまたいで保持しておきたいユーザー固有の情報を格納しておくと便利です。

□セッションはブラウザ固有の存在なので、**ブラウザが閉じられるとセッションも消去されます。**セッションが不要になった際はinvalidateメソッドを使用して明示的に消去することが可能です。
また、一定時間経過で消去する（**セッションタイムアウト**）設定も施すことが可能です。

□以下の要領で使用します。

- セッションの生成
  HttpSession session ＝ request.getSession() ;
- セッションへのデータの保存
  session.setAttribute(“属性名“ , 保存したいインスタンス) ;
  ※保存できるのはインスタンスオブジェクトのみです。プリミティブ型変数は保存できないのでラッパークラスに変換するなどの工夫が必要です。
- セッション上のデータの取得
  (取得するインスタンスの型)session.getAttribute(“属性名“)
  ※getAttributeの戻り値は Object型 であるため、必ず取得するインスタンスの型でキャストする必要があります。
- セッションの消去
  session.invalidate() ;

～ フォワード ～

≪Sample_5_08_3≫

リクエスト

ShowSurveyBySatisfactionLevel.java

＜処理①＞
リクエストの取得

＜処理②＞
BusinessLogicを起動
（一覧データの取得）

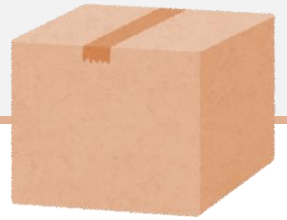
バトンタッチ！
（フォワード）

show_survey_by_satisfaction_level.jsp

＜処理③＞
HTMLの生成＆レスポンス送信

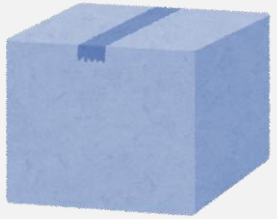
レスポンス

request
（リクエストスコープ）

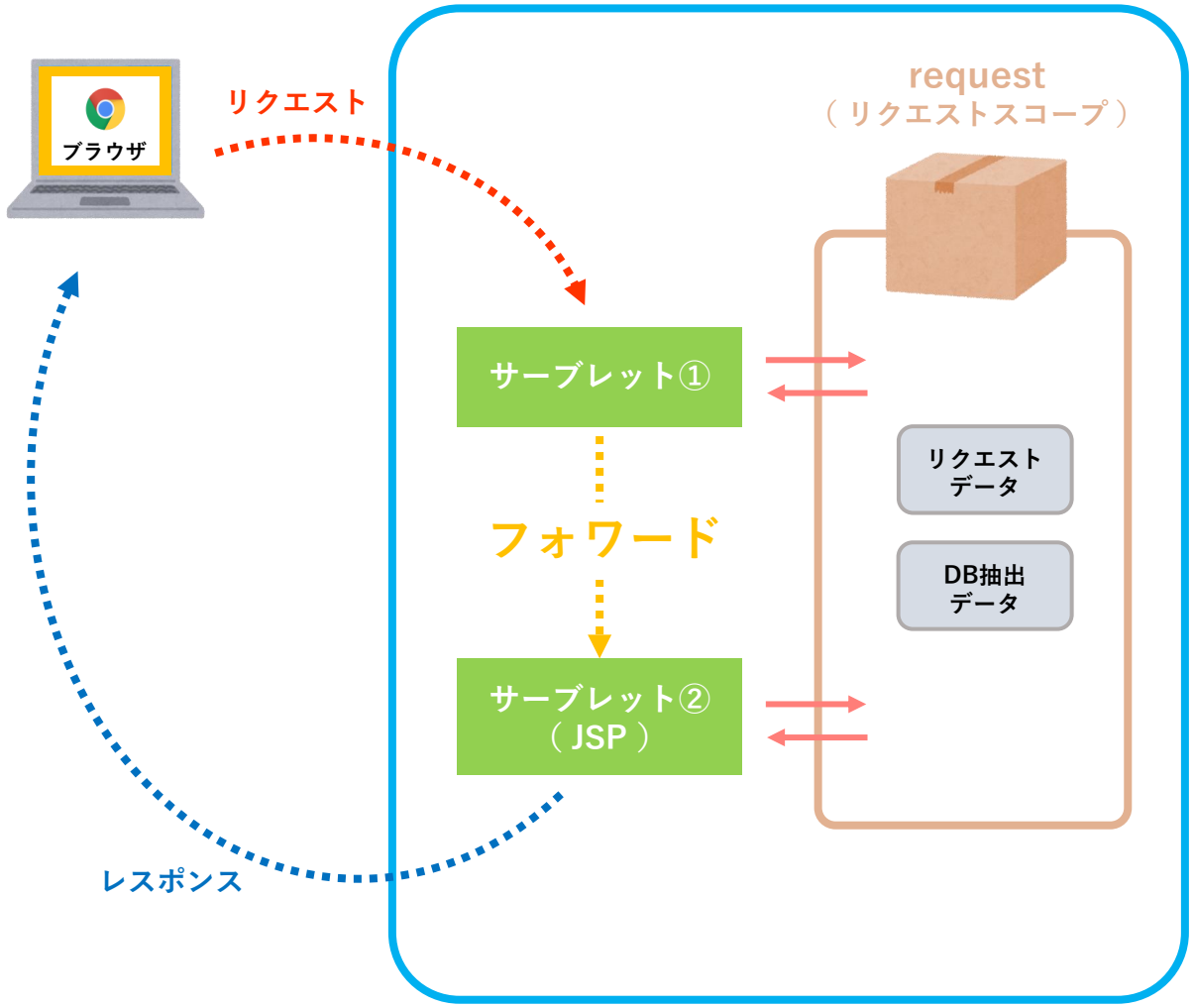


抽出した
アンケートリスト

response



～ フォワード ～



≪フォワードとリクエストスコープ≫

□リクエスト処理を他のリソース（サーブレット、JSPなど）に引き継ぎたい時は**フォワード**を使用します。リダイレクトと異なり、一度のリクエストでリソース間遷移が可能なため無駄なく安全性も保てます。

□フォワードは以下のようにRequestDispatcherインターフェースで定義されるforwardメソッドを使用して行います。

① RequestDispatcherオブジェクトの生成

```
RequestDispatcher dispatch
        = request.getRequestDispatcher("遷移先のパス");
```

② forwardメソッドの起動

```
dispatch.forward( リクエストオブジェクト , レスポンスオブジェクト );
```

□フォワード元と先ではリクエストオブジェクトが引き継がれるため、フォワード先とデータを共有したい場合はこれを利用します。
（この特性からリクエストオブジェクトは**リクエストスコープ**とも呼ばれます。）

- リクエストスコープへのデータの保存

```
request.setAttribute(“属性名“ , 保存したいインスタンス) ;
```

※保存できるのはインスタンスオブジェクトのみです。プリミティブ型変数は保存できないのでラッパークラスに変換するなどの工夫が必要です。

- リクエストスコープ上のデータの取得

```
(取得するインスタンスの型)request.getAttribute(“属性名“)
```

※getAttributeの戻り値は Object型 であるため、必ず取得するインスタンスの型でキャストする必要があります。

【演習】

プロジェクト「Ex_5_07」を
MVCモデルのサイトへと作り替えましょう！

これでラスト！！
最後の一押し！！

【演習】

手順

（１） eclipse上に新しい動的Webプロジェクトを作成する。
プロジェクト名：Ex_5_08

（２）プロジェクト「Sample_5_08_3」を参考に「controller」「model」「view」フォルダを作成する。

（３）プロジェクト「 Ex_5_07 」を流用して演習に取り組む。

～ ポートフォリオを作ってみよう！ ～

【要件】

≪ 基本機能 ≫

☐ ログイン管理機能
・セッションを使ってログイン状態を管理

☐ メッセージ投稿機能
・投稿内容をデータベースに保存
・XSS(クロスサイトスクリプティング)対策

☐ メッセージ一覧表示機能
・投稿内容をデータベースから取得
・動的Webページ

☐ 入力チェック機能
・空欄で投稿できないように警告を出す など
・JavaScriptで実装

☐ MVCモデルで構成

≪ 余裕があれば！ ≫

☐ メッセージ削除機能

☐ メッセージ編集機能

☐ アップロード機能（画像など）

☐ リッチなページ
・CSSによるデザインに優れたページ
・JavaScriptによる動きのあるページ

☐ 外部APIの使用
・SNSログイン など

おつかれさまです！
ここまで来れたことに胸を張ろう！