されど我らが日々
ーJavaサーブレット入門ー

岐阜経済大学　経営学部 経営情報学科　井戸　伸彦

来歴：

0．0版　2004年1月10日

0．1版　2004年6月18日（eclipse環境）

スライドの構成

(0) はじめに

■Ｊａｖａサーブレットに関して、簡単なプログラムを作成します。

■次のスライドと同等の知識を習得済みであることを前提としています。

- Web関係
  - 「ヘンタイ良い子のＷｅｂ講座」
  - 「ツァラトストラ書くWeb　－速習HTML入門－」
  - 「シャボン玉HTML　－フォーム入門－」
- Java関係
  - 「シャバドゥビ、ジャバ　－Java覚書－」
  - 「ドゥビドゥバ、ジャバ　－直感Ｊａｖａのオブジェクト－」
- eclipse関係
  - 「月に吠える　－eclipseによるJavaアプリケーション作成－」
  - 「ただ一疋の青い猫のかげ　－eclipseによるJavaサーブレット作成－」
- JSP
  - 「ウィークエンド・シャッフル－JSP入門－」

■スライド中、次の参照を行っています。

- Ｊａｖａ教科書：「新Ｊａｖａ入門－ビギナー編－」、林晴比古

■説明は直感的な分かりやすさを重視し、厳密には不正確な言い方もしています。

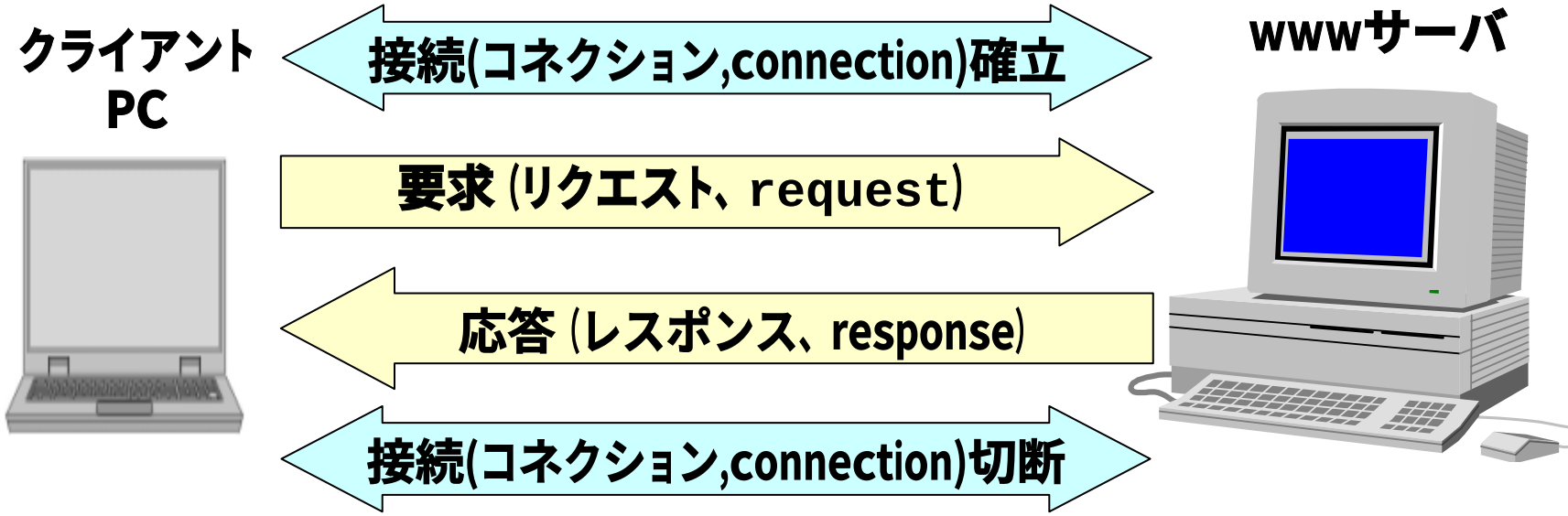
(1) Hello, World

■ありきたりですが、まず、“Hello,World.”のプログラムを作ってみます。

(1．1) HTTP、ステートレス

■HTTP (Webサイトの閲覧に用いるプロトコル) は、サーバ側で状態を持たないという意味で、ステートレス(stateless)であると言います。

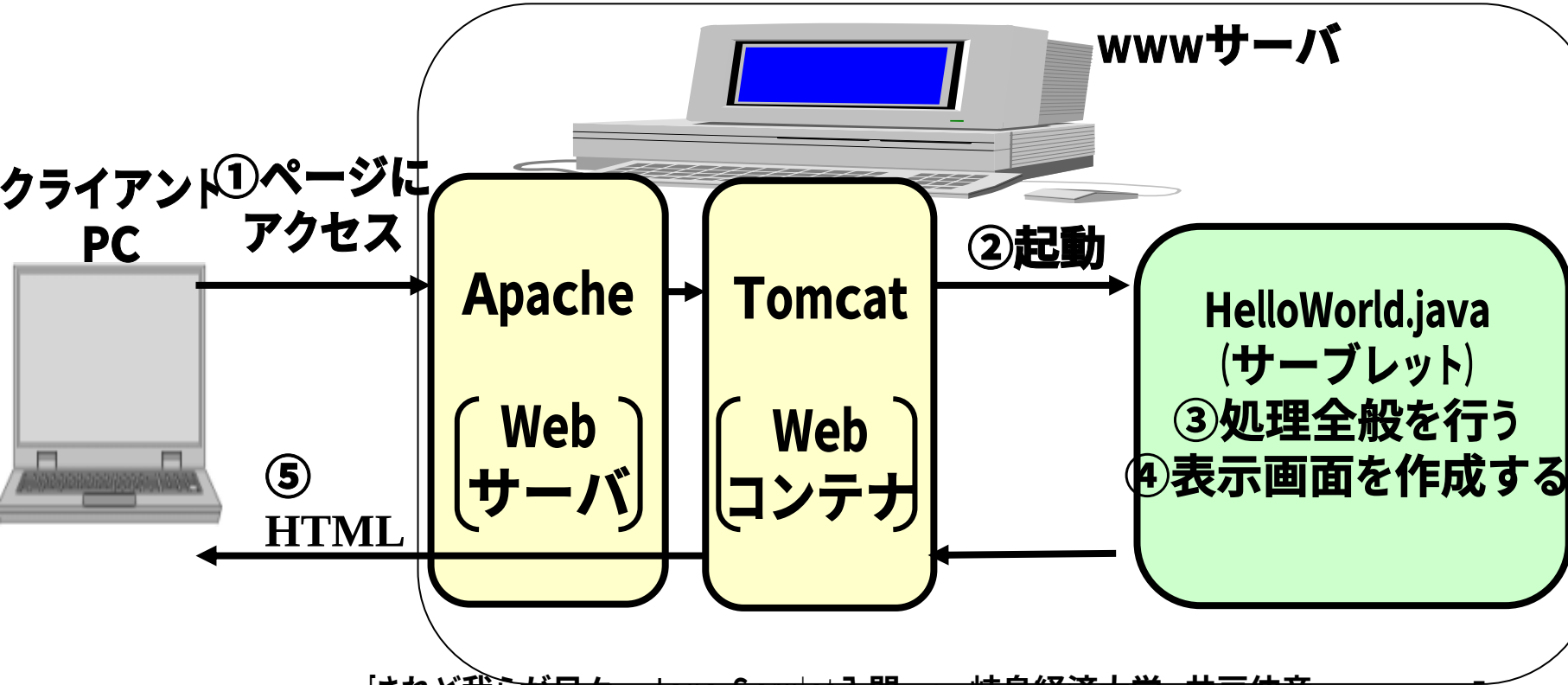
■Webサーバへのアクセスはすべて、次の形を取るという意味に理解しておいてください。



(1．2) システムの構成

■システムの構成は、図のようになっています。

- Apache、Tomcatは、ＪＳＰでも用いました。
- JAVAサーブレットは、コンパイルしておく必要があります。



(1．3) ファイル構成

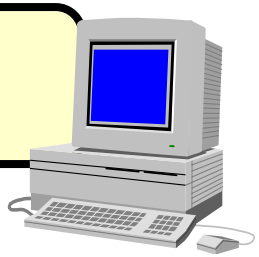
■ファイル構成は、次のとおりです。

- eclipse+lombozにより自動的に構成させます。
- 編集するファイルは、①”web.xml”と、②” HelloWorld.java”の2つです。

http://localhost:8080/idoApp/helloWorld でアクセス

自身のＰＣからの場合。
他のＰＣからでも、
http://(IPアドレス):8080/idoApp/helloWorld
アクセス出来ます。

Tomcat
(各自のLinuxＰＣ)



ファイルサーバ上の
各自のアカウント

yyy (アカウント名) /eclipse/workspace

- servletProject
  - src
    - HelloWorld.java (②)
  - myWeb
    - *WEB-INF*
      - *web.xml (①)*
      - *classes*
        - HelloWorld.class

これらは、この位置、この名前と決まっている (Tomcat)。

(2) HelloWorld.java

■HelloWorld.javaの内容を示します。

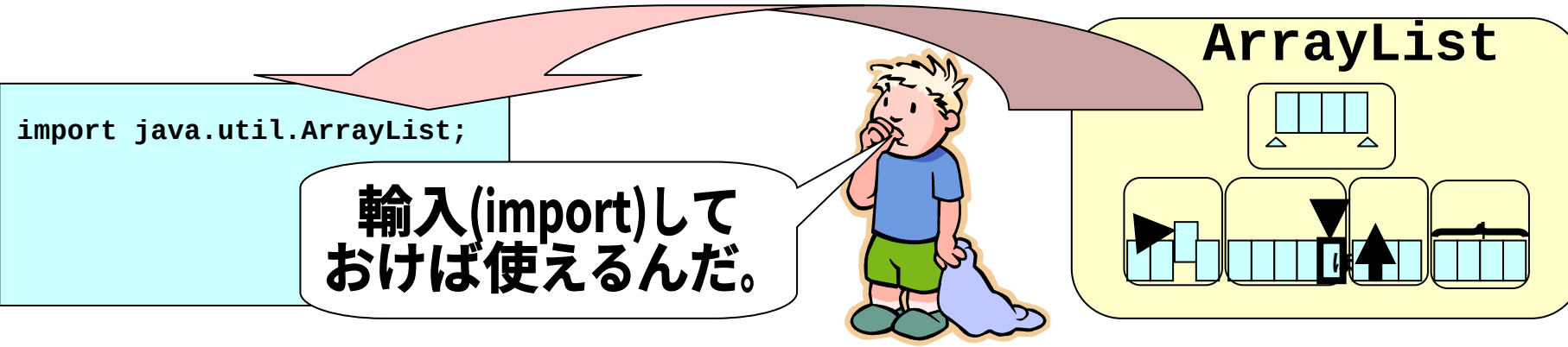
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
  throws ServletException, IOException{
  PrintWriter out=response.getWriter();
  out.println("Hello, World!");
  }
}
```

(2．1) インポート

■クラス“HelloWorld”は、いくつかのクラス・インタフェースをインポートしています。

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

■インポートについては、Java教科書、Javaスライドを参照してください。



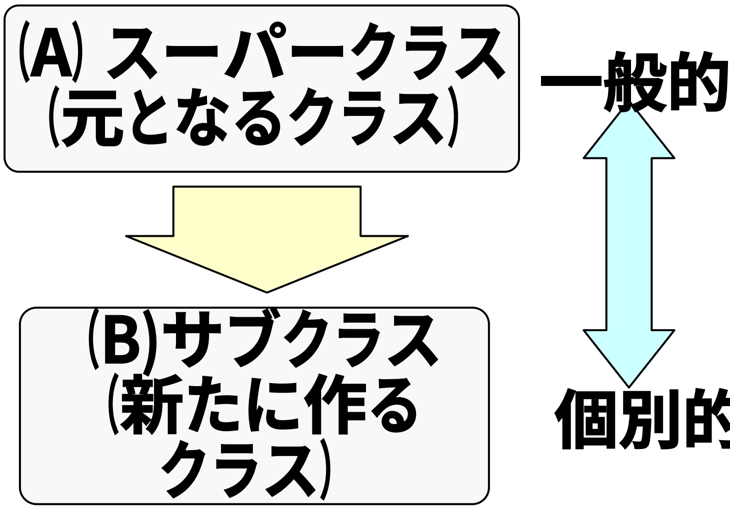
(2.2) 継承

■クラス“HelloWorld”は、クラス“HttpServlet” を継承しています。

```
public class HelloWorld extends HttpServlet{
    :
}
```

■継承については、Java教科書、Javaスライドを参照してください。

- BはAを拡張している。
- BはAを継承している。
- AはBの汎化である
  (これはUMLでの言い方、後述)。



(2．3) 例外処理

■クラス“HelloWorld”は、2つの例外を投げます。

```
public void doGet(...)
 throws ServletException, IOException{
      :
}
```

■例外処理については、Java教科書、Javaスライドを参照してください。

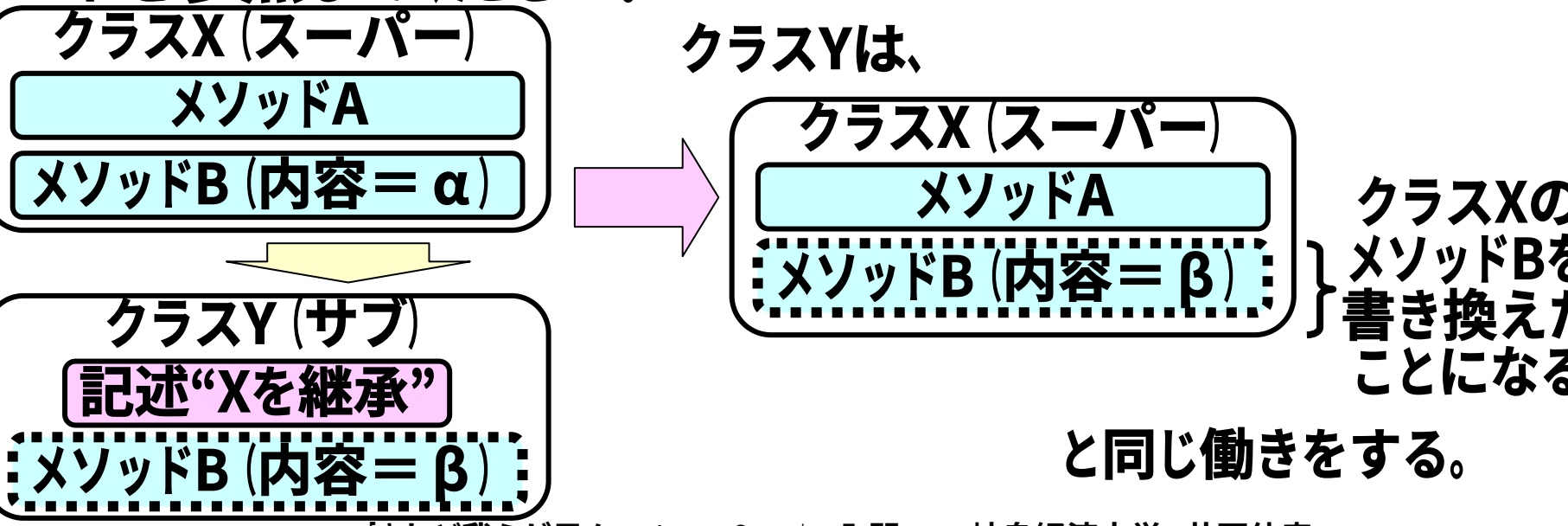
- ここでは、あまり気にしないで結構です。

(2．4) オーバーライド

■クラス“HelloWorld”は、継承したクラス“HttpServlet”のメソッド“doGet”をオーバーライドします。

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
}
```

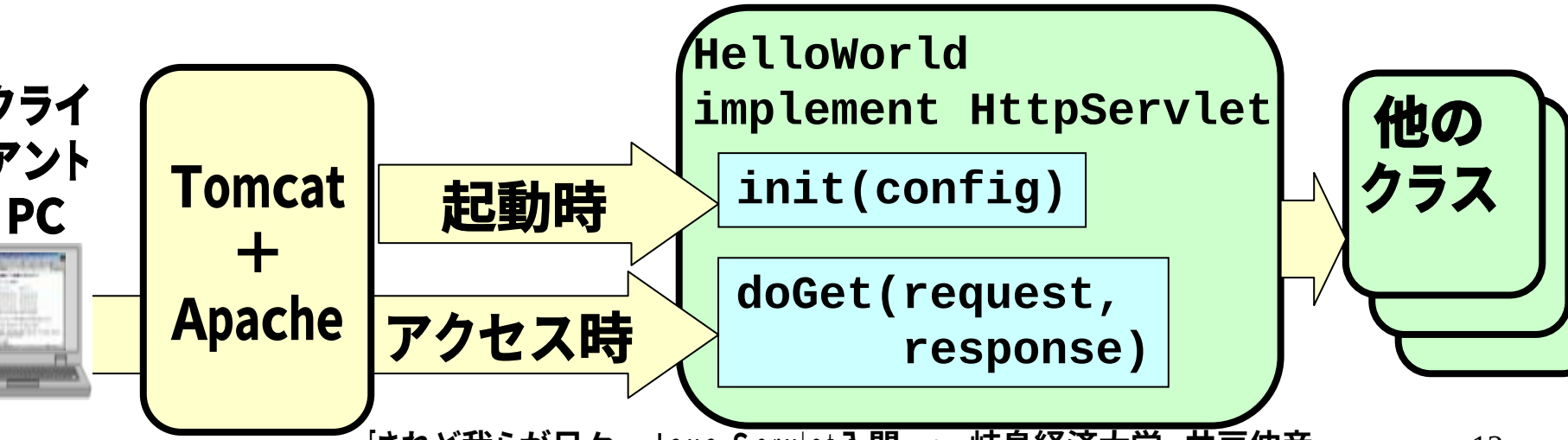
■オーバーライドについては、Java教科書、Javaスライドを参照してください。



(2．5) サーブレットが呼ばれるとき

■tomcatは、“HttpServlet”を継承している“HelloWorld”クラスを、次の2つの場合に呼び出します。

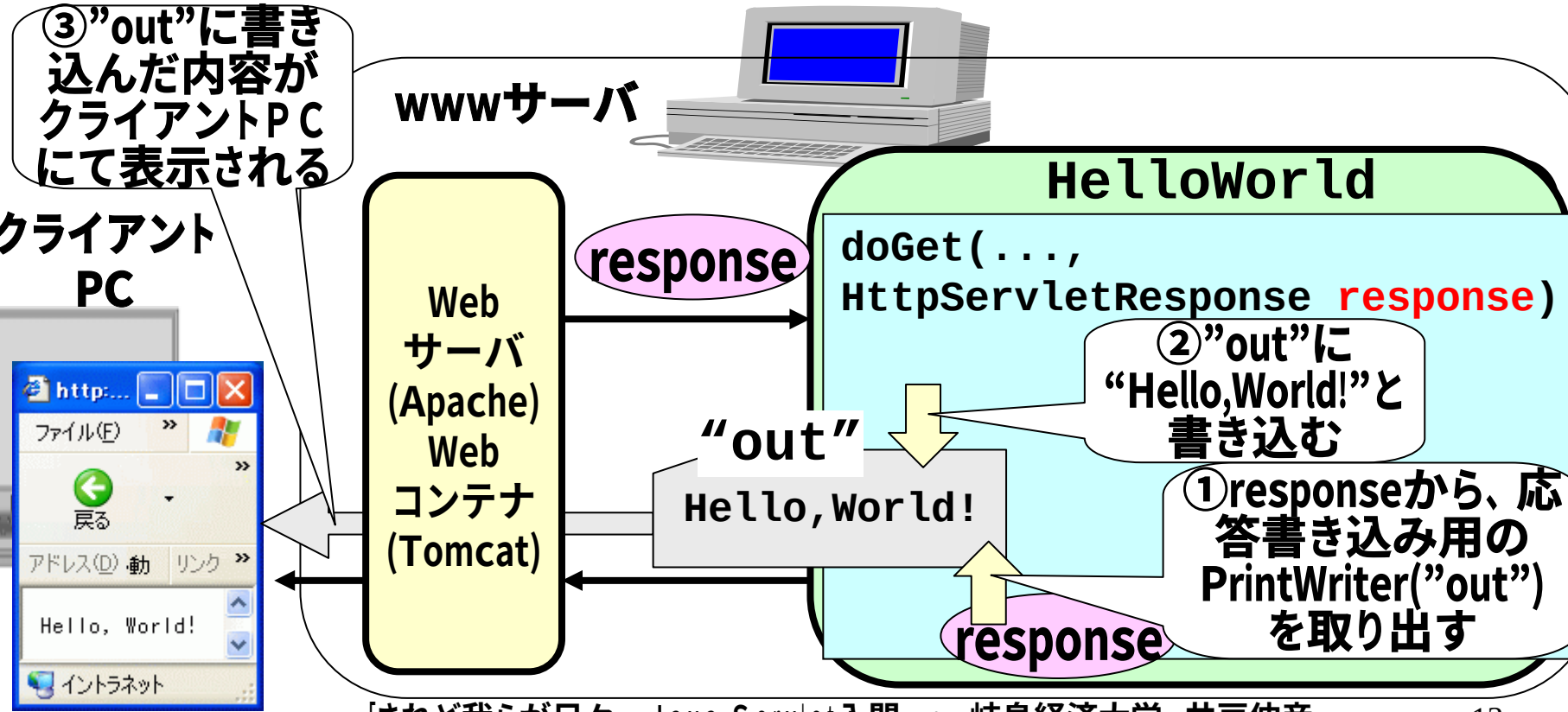
- サーバが立ち上がって、tomcatが起動されたとき、クラス”HelloWorld”のinitメソッドが呼ばれます(今回、なにもしないので、initメソッドはオーバーライドしていません)。
- “http://xxx../idoApp/helloWorld”にアクセスされた時、クラス”HelloWorld”のdoGet (doPost) メソッドが呼ばれます。今回は、このdoGetメソッドをオーバーライドして、”Hello,World!”と表示させます。



(2．6) 表示の書き出し

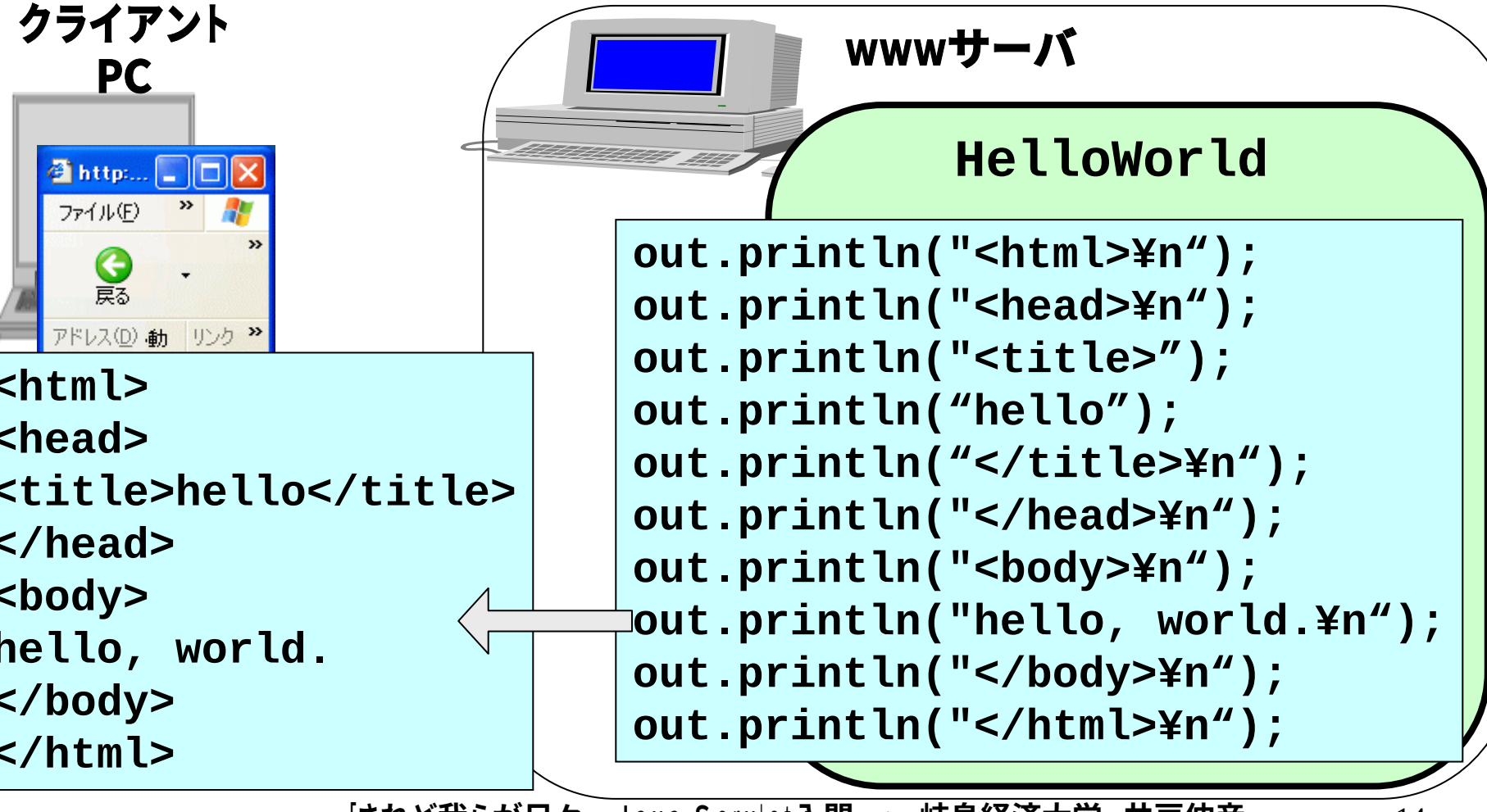
■“Hello,World!”と書き出すプログラム本体（次の2行）は、図のような処理を行っています。

```
①   PrintWriter out=response.getWriter();
②   out.println("Hello, World!");
```

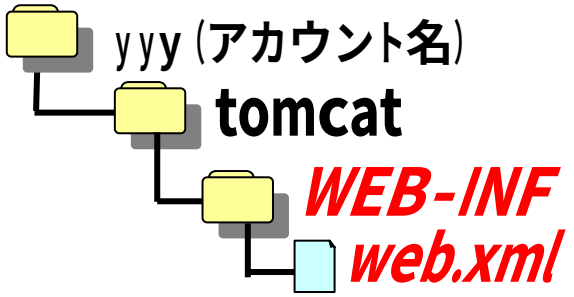


(2．7) HTML文書を表示するには

■PrintWriterクラスの“out”にHTMLを書き込んでいけば、クライアント側にHTML文書を表示できる訳です。



(3) web.xmlファイル

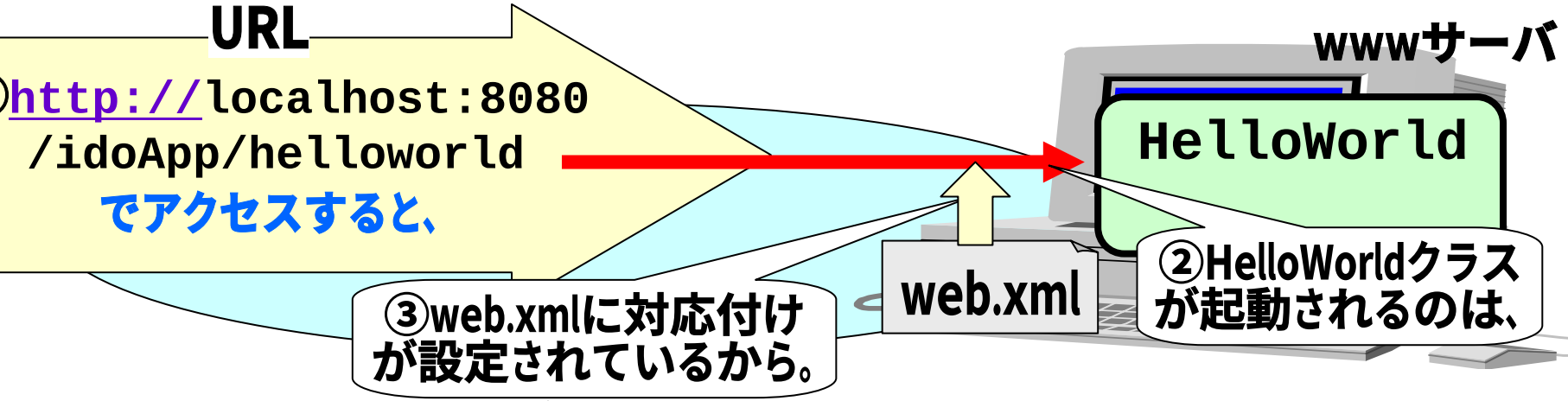


■web.xmlファイルは環境設定の中心となるファイルです。

■HelloWorldのプログラムでは、web.xmlにより次のような設定がなされています。

- サイトにアクセスするURLとこれを受けて動作するサーブレットとの対応付け。

■web.xmlには、その他の設定も行いますが、HelloWorldでは上記の設定のみを行います。



(3．1) ファイルの中身

■web.xmlのファイルの内容を示します。

“ ”(スペース)
は必須です

```
?xml version="1.0" encoding="ISO-8859-1"?>
!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
web-app>
   <servlet>
       <servlet-name>HelloWorldServlet</servlet-name>
       <servlet-class>HelloWorld</servlet-class>
   </servlet>
   <servlet-mapping>
       <servlet-name>HelloWorldServlet</servlet-name>
       <url-pattern>/helloWorld</url-pattern>
   </servlet-mapping>
/web-app>
```

（3．2）XML

■web.xmlは、XML文書として作成されています。

■XMLについては、別途勉強することとします。ここではその形式に慣れてください。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems,
            Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
   <!--アンケートサーブレットの設定（次スライド以降で説明）-->
</web-app>
```

スペース、または、改行は必須です

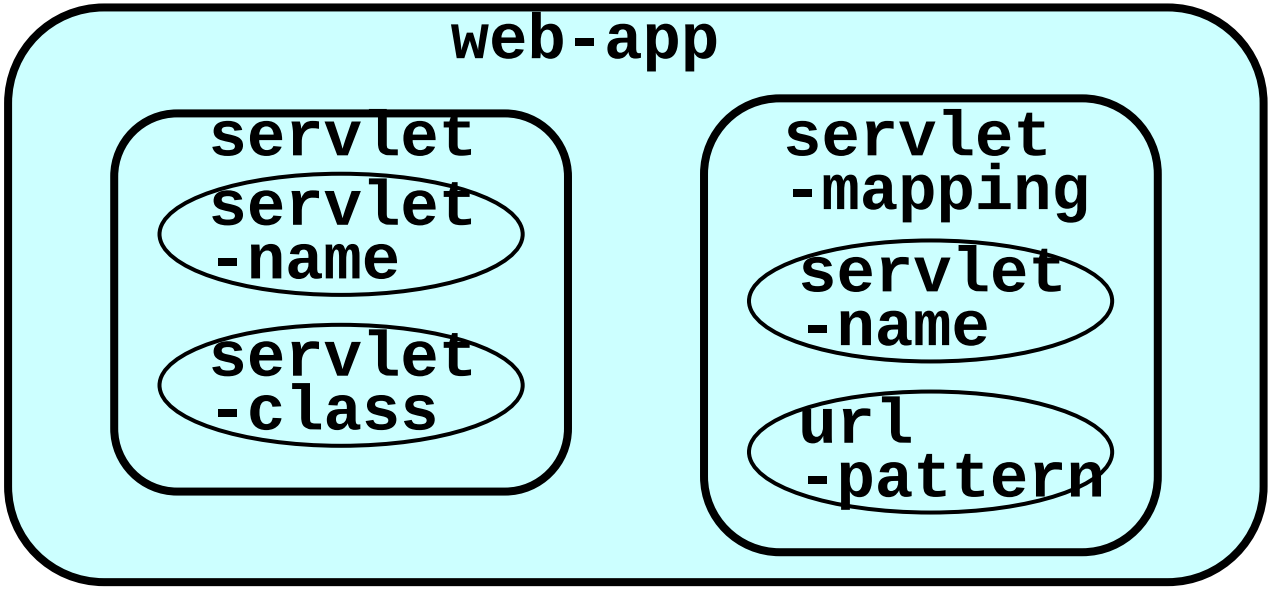
■最初の部分は、“この文書がXMLであること”についての記述です。

■Webアプリケーションに関する実際の記述は、“<web-app>”と“</web-app>”との間に置かれます。

(3．3) 全体像

■全体が、“<web-app>”と“</web-app>”との間に囲まれているように、この中の要素も、“<xxx>”と“</xxx>”とに囲まれる形をしています。

■全体像を示すと、次のとおりとなります。



(3．4) 2つの部分

■“<web-app>”の中身は、2つに分けて説明します。

- ①サーブレットに関する記述
  “<servlet>”、“</servlet>”の間
- ②サーブレットとURLとの関に関する記述
  “<servlet-mapping>”、“</servlet-mapping>”の間

■①②は、サーブレットの数、URLの数だけ作成します。
今回は、HelloWorldにつき、ひとつづつ作成します。

```
<web-app>
  <servlet>
    <!- スライド(3.5)にて説明 -->
  </servlet>

  <servlet-mapping>
    <!- スライド(3.6)にて説明 -->
  </servlet-mapping>
</web-app>
```

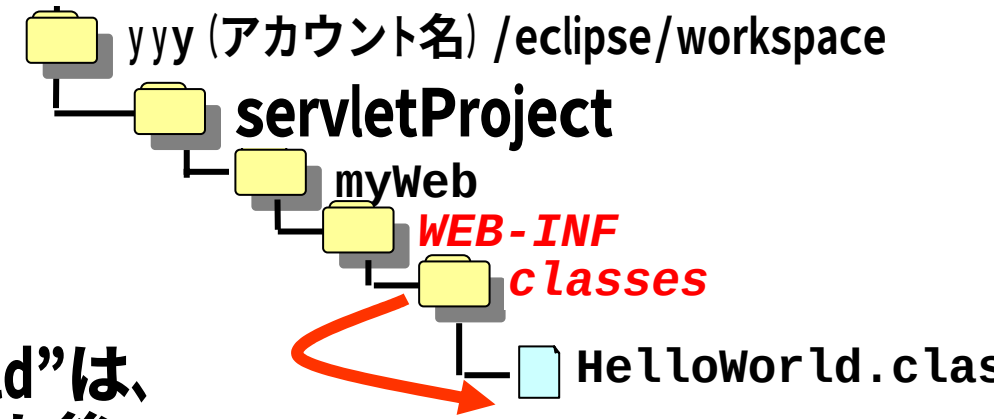
(3．5) servletの部分

■ここではサーブレットの名前と対応するプログラムが定義されています。

```
<servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
</servlet>
```

■上記の記述では、次のとおりとなっています。

- 名前：HelloWorldServlet
- クラス名：HelloWorld



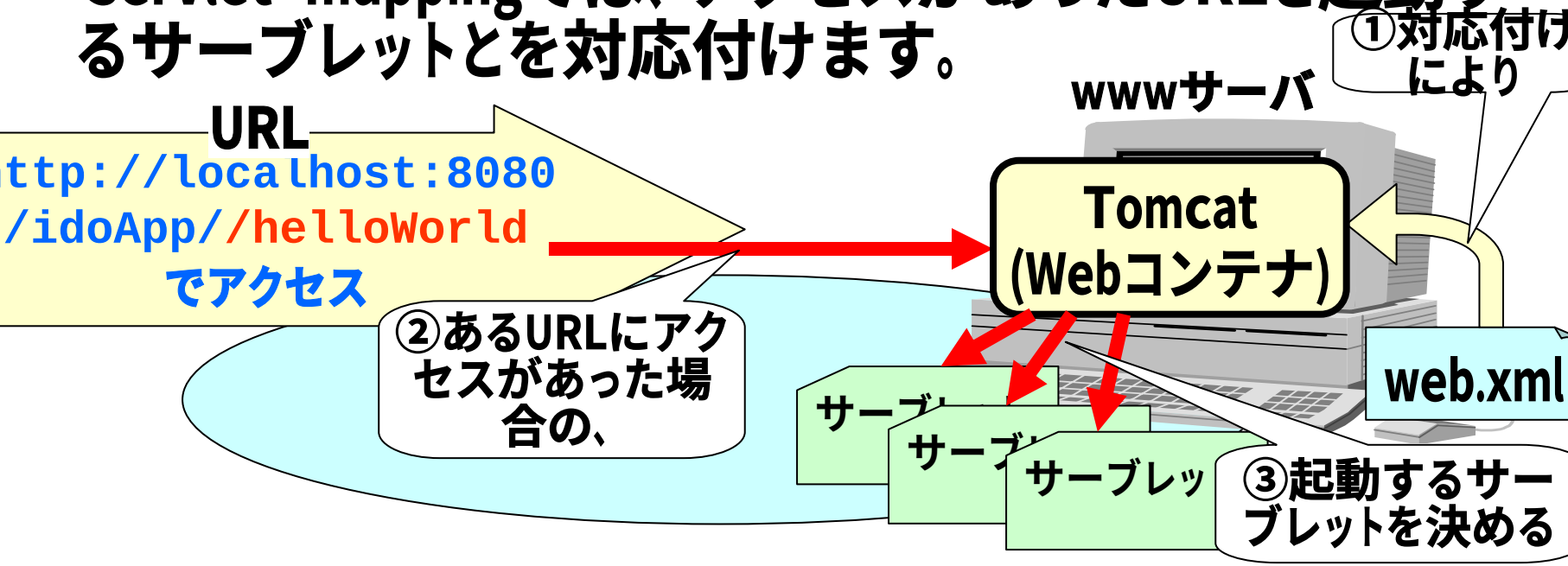
■上記の、クラス名“HelloWorld”は、javacコマンドでコンパイルした後、javaコマンドで実行する際の名前と同じです (実際はサーブレットはjavaコマンドで実行できません)。

```
% cd classes
<コンパイル>
% javac HelloWorld.java
<実行 (実際は実行できない) >
% java HelloWorld
```

これがクラス名

(3．6) servlet-mappingの部分

■servlet-mappingでは、アクセスがあったURLと起動するサーブレットとを対応付けます。



②サーブレット”inquery”（スライド(3.5)で説明した、サーブレットの名前）を起動する

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

①”/helloWord”にアクセスがあったら、

(3．7) URLとの対応

■URL中、tom_yyyの部分は、Apache(Webサーバ)とTomcat (Webコンテナ) の設定ファイルにより、対応付けを決めています(Tomcatインストールの資料参照)。

http: //localhost:8080 /idoApp/helloWorld

プロトコル名　サーバ名+ポート番号 (ドメイン名＋ホスト名)　設定ファイルで対応付け　web.xmlで対応付け

■tomcatは、8080のポートをリッスンしています (「情報ネットワーク論」のスライドを参照してください)。

＜Tomcat＞
(Windows)
C:¥Program Files¥Apache Group¥Tomcat 4.1¥conf¥server.xml
(Linux)/usr/local/jakarta-tomcat-4.1.27/conf/server.xml

```
<Context path="idoApp"
docBase="C:¥Documents and Settings¥JA    1¥My Documents
        ¥workspace¥servletproject¥myweb" />
```

"idoApp"ときたら、このディレクトリの"WEB-INF/web.xml"を見に行く。

(4．1) コンパイル

(eclipseでは、(4.1)(4.2)の内容は気にする必要はありません。)

■コンパイルは、普通に実行すればＯＫです。

```
% cd tomcat/WEB-INF/classes
% javac HelloWorld.java
```

■但し、クラスパスが設定されていないと、エラーが出ます。各々のホームディレクトリにある、“.bashrc”ファイルに、次の2行を加える必要があります。

```
export CATALINA_HOME=/var/tomcat4

CLASSPATH=$CATALINA_HOME/common/lib/servlet.jar:$CLASSPATH
```

■“.bashrc”ファイルについては、

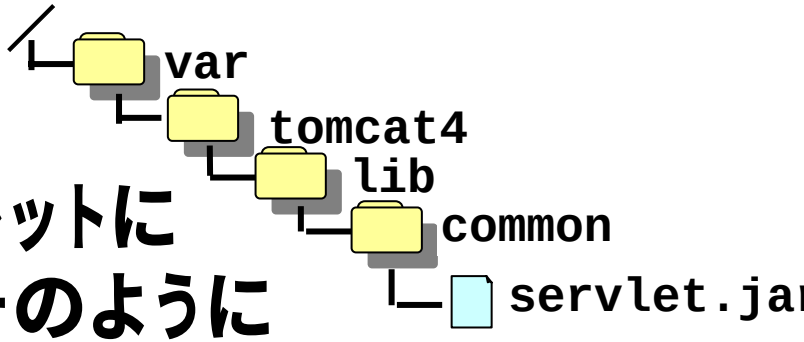
■注意：“.bashrc”ファイルの編集は、慎重に行ってください。ゼミでは、講師に相談の上、行ってください。

(4．2) クラスパス

■スライド(2.1)にて示したとおり、HelloWorld.javaでは次のようなインポートをおこなっています。

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

■他人が作ったプログラムを使っている訳ですが、利用元のプログラムは天から降ってくるわけではありません。



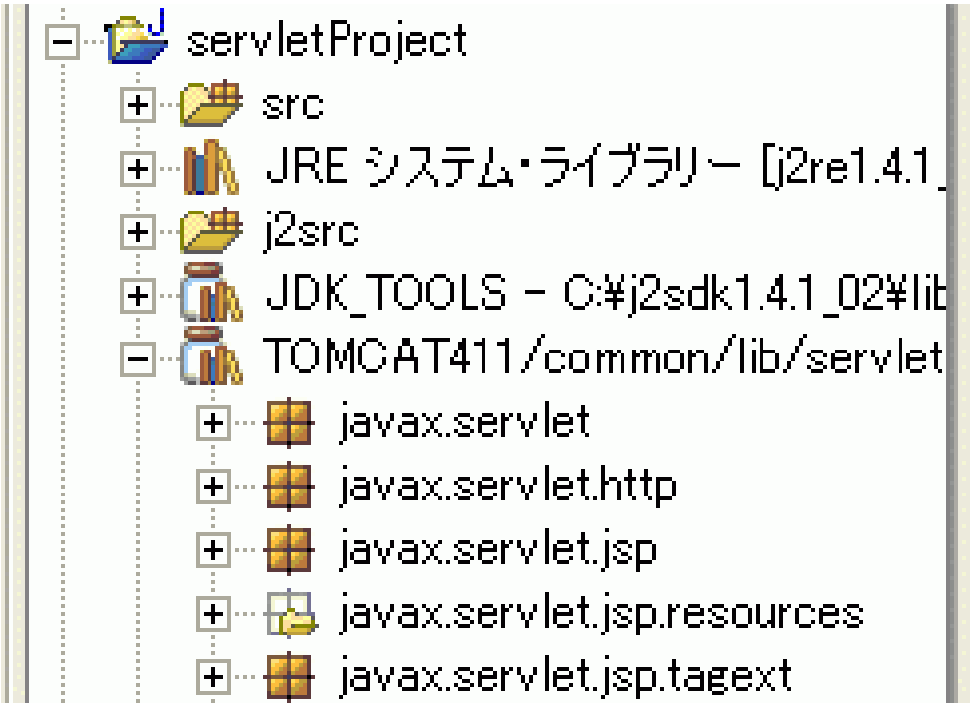
■井戸のサーバでは、サーブレットに関わるプログラムの固まり(そのように理解しておいてください)を、図のように置いています。

■コンパイル時に、これを使えるようにする設定が、前スライドのクラスパスの設定です。

(4．3) eclipseでは。。。

■eclipse+lombozの環境では、(4.1)(4.2)の環境設定は、すでになされています。すなわち、メニュー上でビルドを行うだけでＯＫとなるようになっています。

■eclipse上のパッケージ･エクスプローラ上で、必要なライブラリが用意されていることを確認することが出来ます。



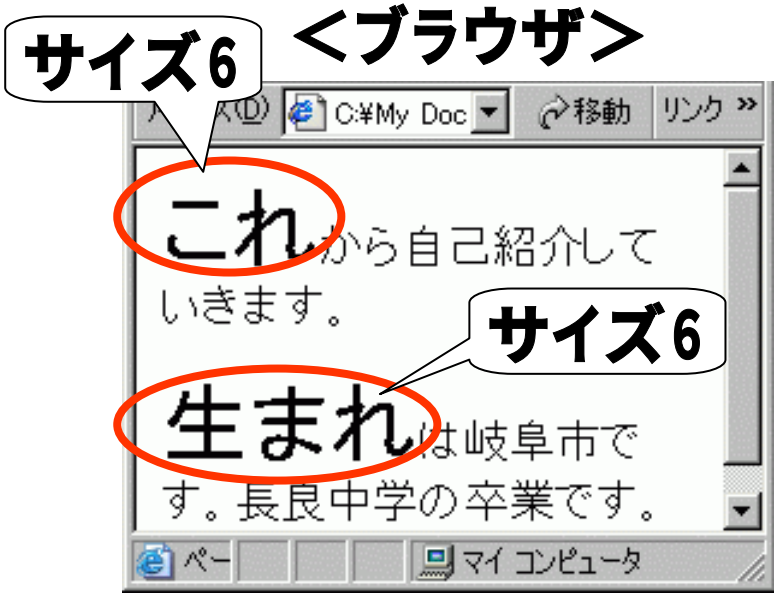
(5) 演習：課題1、課題2

■ (課題1) HelloWorldサーブレットを作成し、実行させてみてください。

- web.xmlを編集した場合、Tomcatの再起動が必要です。
- Tomcatの再起動にはルート権限が必要なため、講師が行います。web.xmlを編集した方は、講師まで申し出てください。

■ (課題2) HelloWorldサーブレットを改造して、アクセスするとつぎのような表示がなされるようにしてください。

<ブラウザ>



(6) Welcomeサーブレット

■これもごく簡単なサーブレットです。

クライアント PC

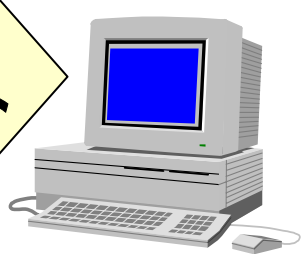
①最初、(注) /idoApp/examples /login.jsp でリクエスト

wwwサーバ (server-ido)

③フォームを表示

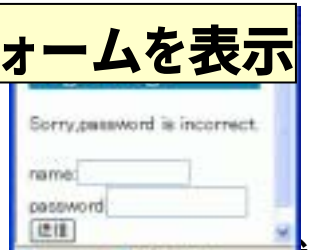
②応答 (レスポンス)

④フォームから、(注) /idoApp/welcomeへリクエスト



⑥メッセージを表示

⑤ (パスワードが正しい) 応答



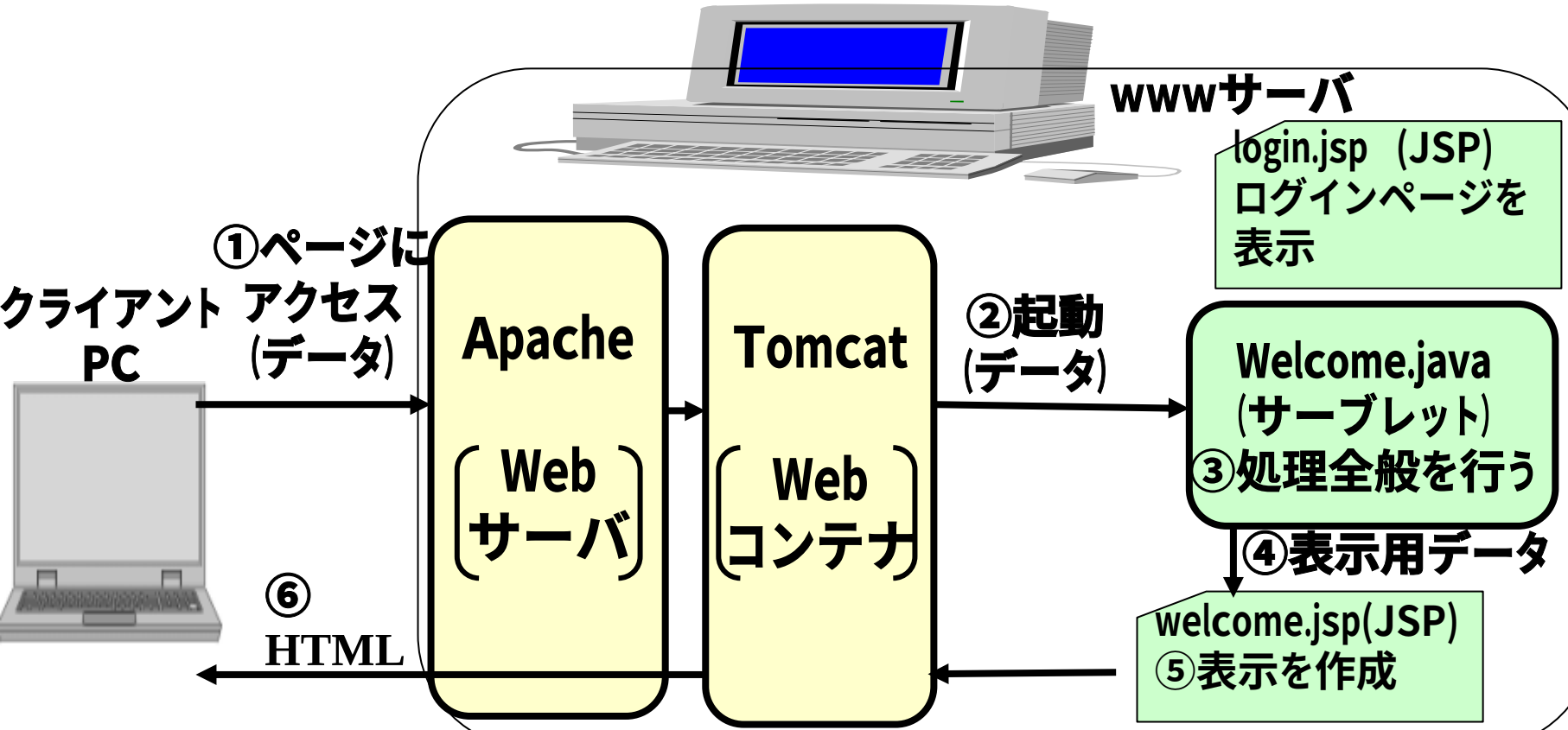
⑥'再度フォームを表示

⑤' (パスワードが誤り) 応答

(注) : http://localhost:8080

(6．1) システムの構成

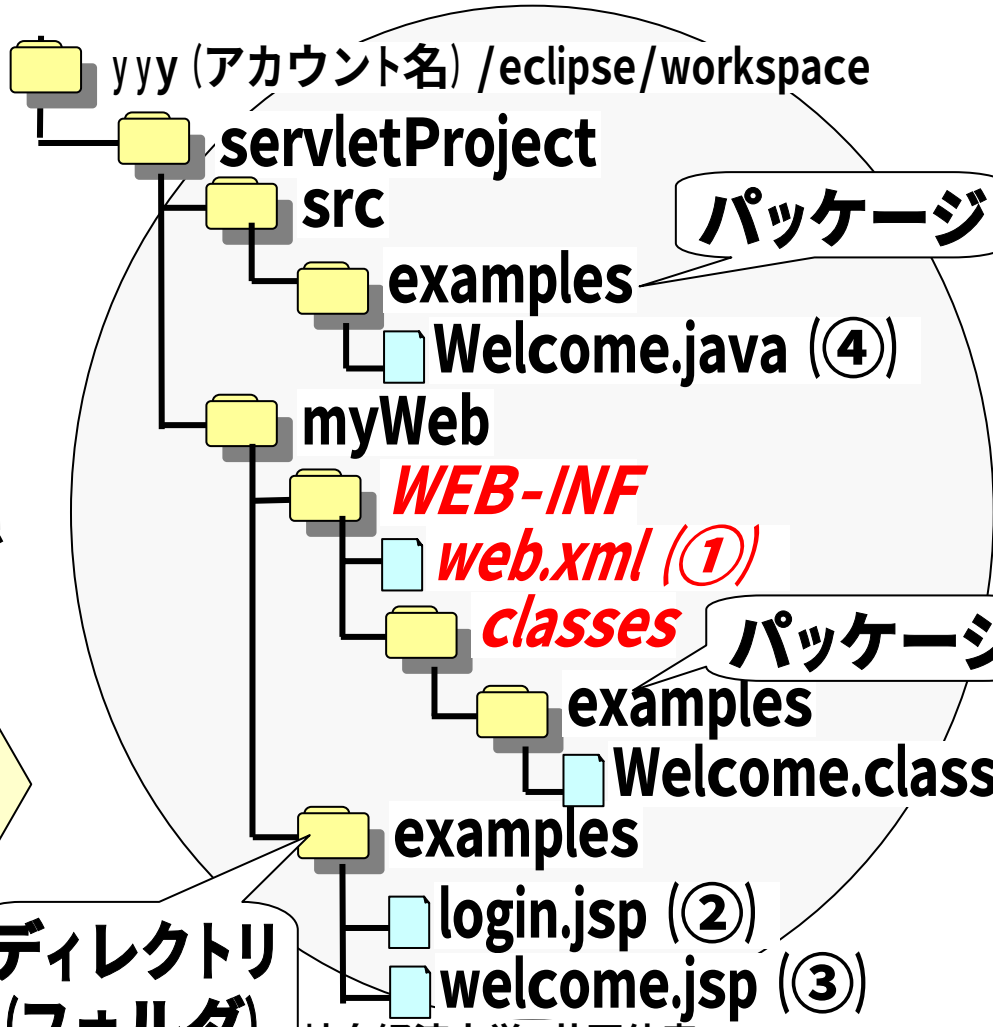
■サーブレットプログラムに加え、ＪＳＰを用います。
■ファイルの関係のイメージは、次の通りです。



(6．2) ファイル構成

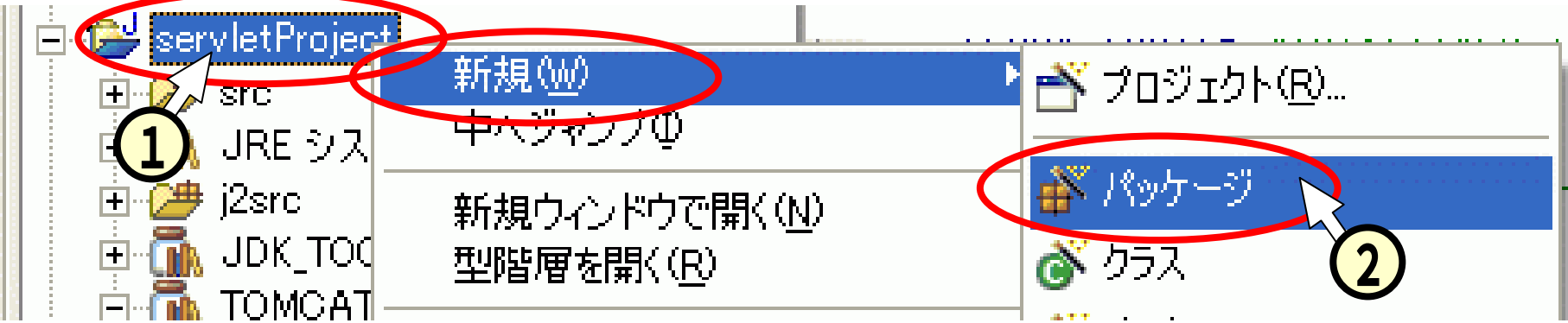
■ファイル構成は、次のとおりです。

- パッケージの“example”と、ディレクトリ(フォルダ)の”examples”とを、eclipse上で作成します。
- 合計4つのファイルを編集します。このうち、① web.xmlは、HelloWorldサーブレットと同じものです。

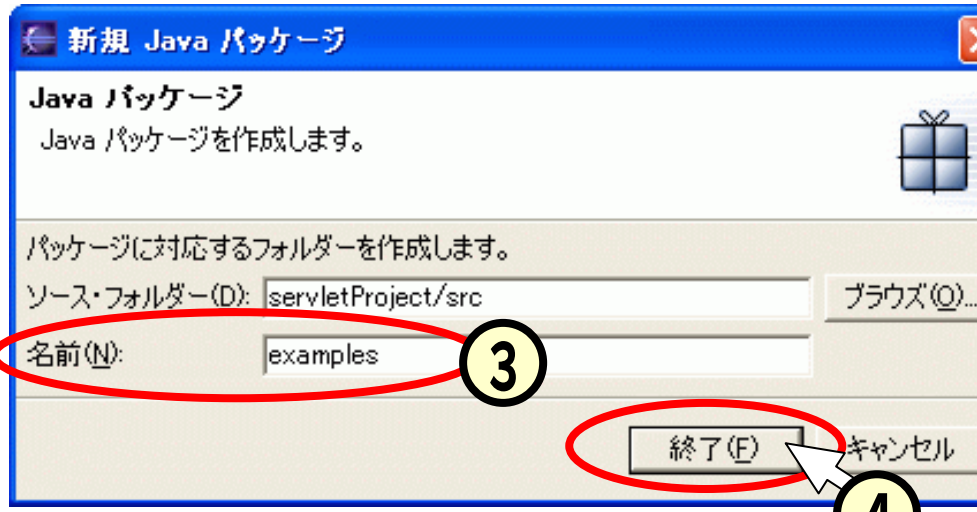


(6．3) eclipse上でのパッケージの作り方

■「パッケージエクスプローラ」上で、プロジェクト(例では“servletProject”)を右クリック(①)し、[新規]-[パッケージ]をクリック(②)します。



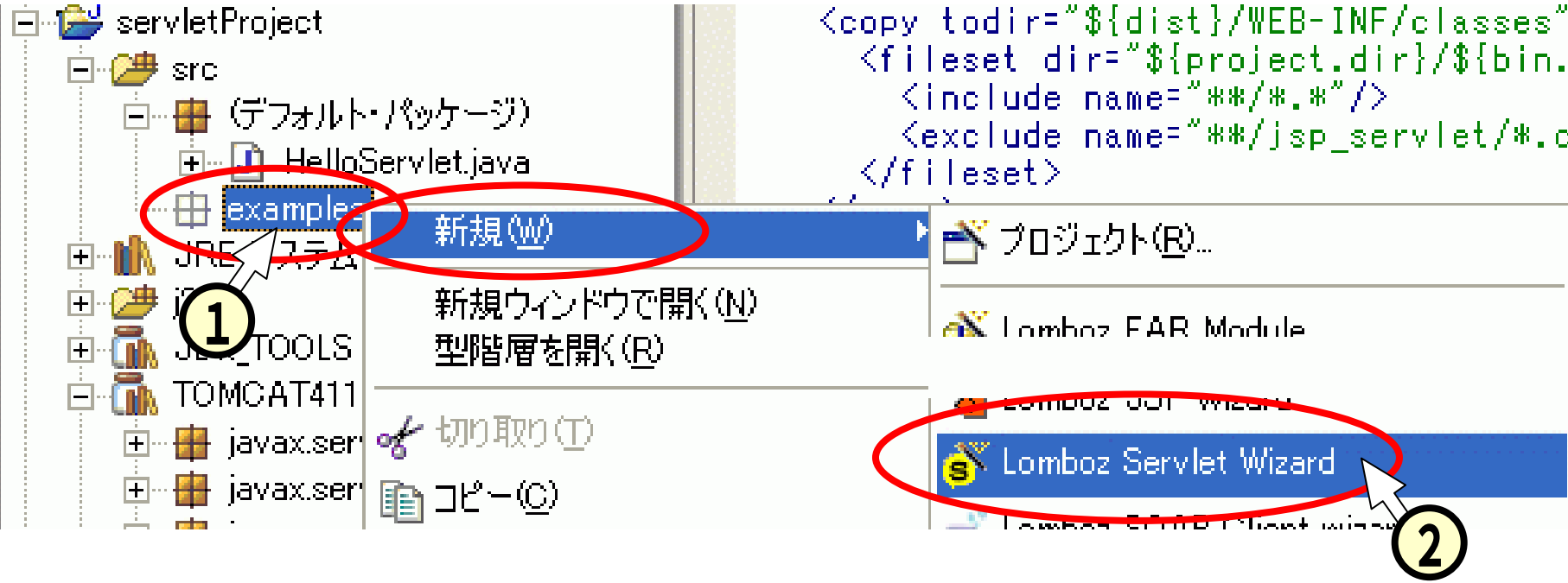
■「新規Javaパッケージ」ウインドウ中で、[名前]の欄にパッケージ名(ここでは“examples”)を入力(③)し、[終了]をクリック(④)します。



(6．4．1) パッケージ内のサーブレットの作り方

■(6.3)で作成した、パッケージ”examples”内に、サーブレット“Welcome”を作ります。

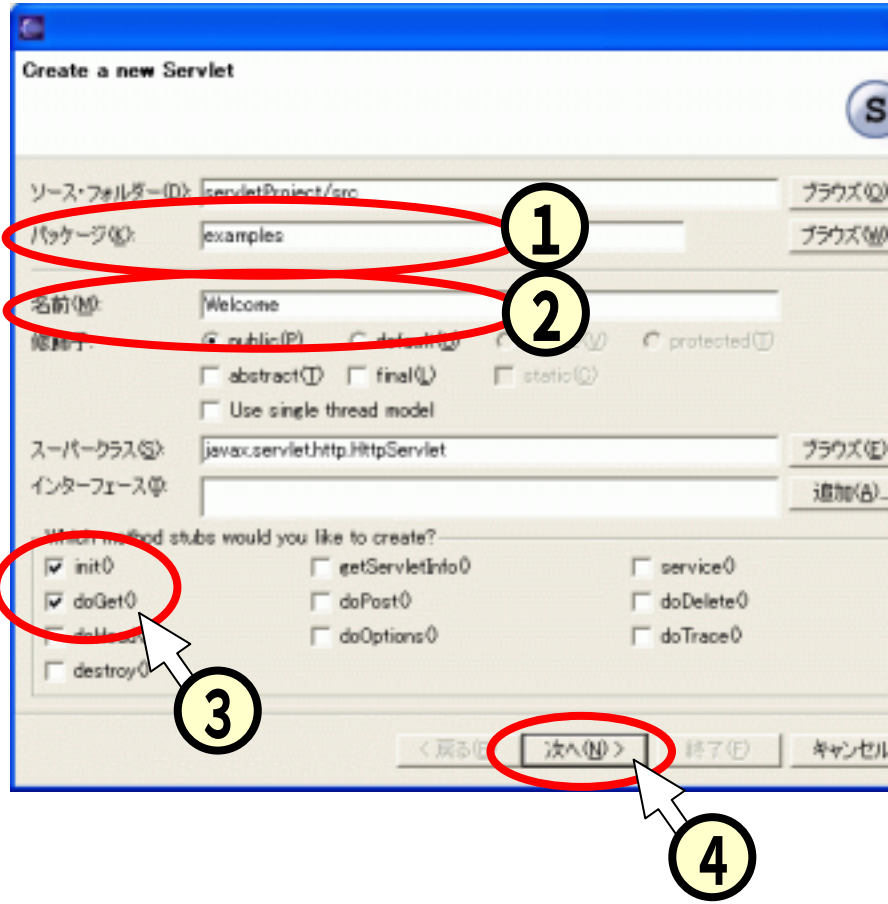
■「パッケージエクスプローラ」上で、パッケージ(例では“examples”)を右クリック(①)し、[新規]-[Lomboz Servlet Wizard]をクリック(②)します。



(6．4．2) 作り方 (続き)

■「Create a new Servlet」ウィンドウにて、[パッケージ]の欄には、“examples”と入力されています (①)。後は、「eclipseによるJavaサーブレットの作成」のスライド (4) と同等の操作を行います。

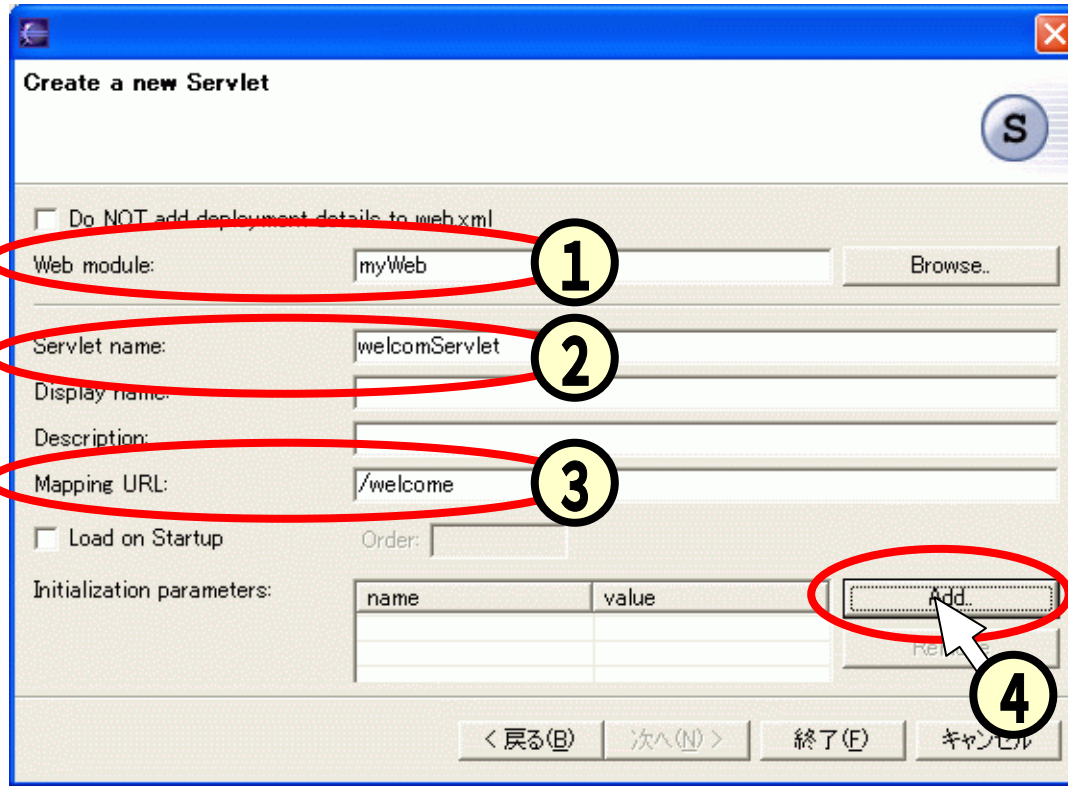
- サーブレットのクラス名 (ここでは、“Welcome”) を入力 (②) します。
- 雛形の指定にて、“init”と“doGet”にチェック (③) を入れます。
- [次へ]をクリック (④) します。



(6．4．3) 作り方 (続き)

■Web.xmlを設定するための情報を入力します。

- Web module
  - ◆myWeb(①)
- Servlet name
  - ◆welcomeServlet (②)
- Mapping URL
  - ◆/welcome(③)

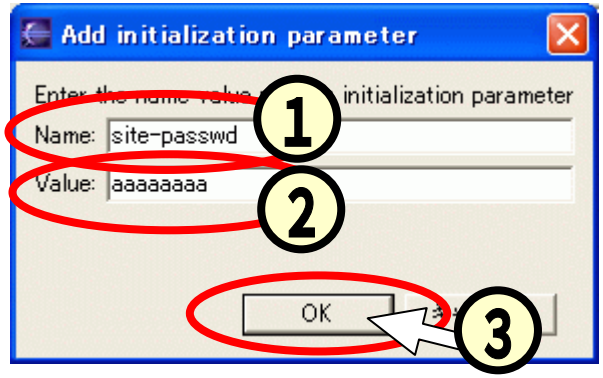


■スライド(10.2.2)で説明する、ファイルweb.xml中の初期化パラメタを設定するため、[Initialization Parameters]の[add]ボタンをクリック (④) します。

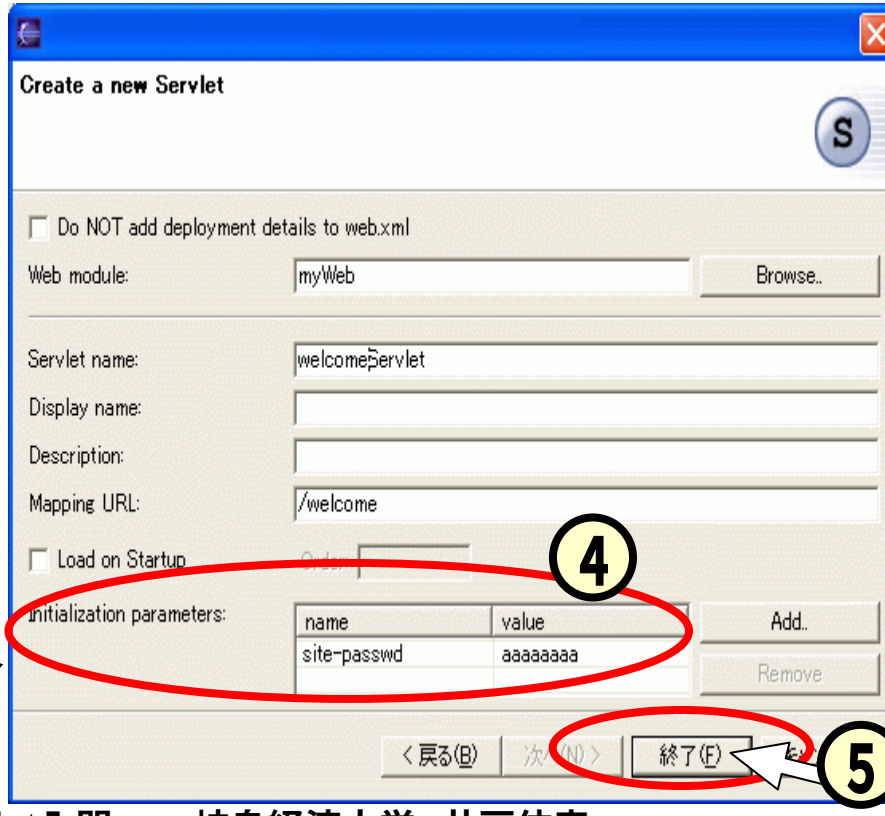
(6．4．4) 作り方 (続き)

■「Add initialization paramter」のダイアログにて、初期化パラメタを追加し、[OK]をクリック (③) します。

- Name
  - ◆site-passwd (①)
- Value
  - ◆aaaaaaaa (②)



■「Create a new Servlet」ウィンドウにて、[Initial parameters]が設定されていることを確認し (④)、[終了]をクリック (⑤) します。



(6．4．5) サーブレット

■出来上がったサーブレットの雛形には、パッケージの宣言がされており(①)、必要なクラスがインポートされています(②)。

①

②

```
package examples;
import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse
public class Welcome extends HttpServlet {
    public void init(ServletConfig config)
    throws ServletException {
        //TODO Method stub generated by Lomboz
    }
    protected void doGet(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        //TODO Method stub generated by Lomboz
    }
}
```

(6．4．6) web.xml

■web.xmlの内容については、スライド(１０)以下で説明します。

```
<web-app>
  <servlet>
    <servlet-name>welcomeServlet</servlet-name>
    <servlet-class>examples.Welcome</servlet-class>
    <init-param>
      <param-name>site-passwd</param-name>
      <param-value>aaaaaaaa</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>welcomeServlet</servlet-name>
    <url-pattern>/welcome</url-pattern>
  </servlet-mapping>
</web-app>
```

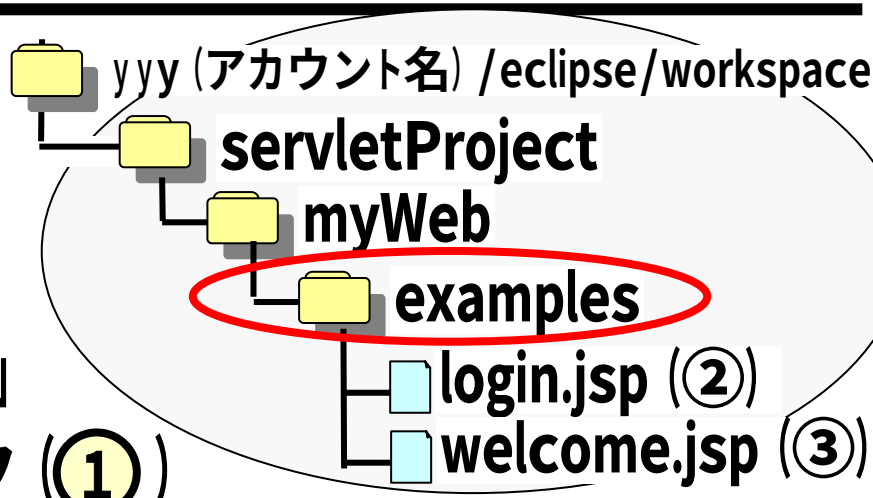
(6．5．1) JSP用のフォルダ、ＪＳＰの作成

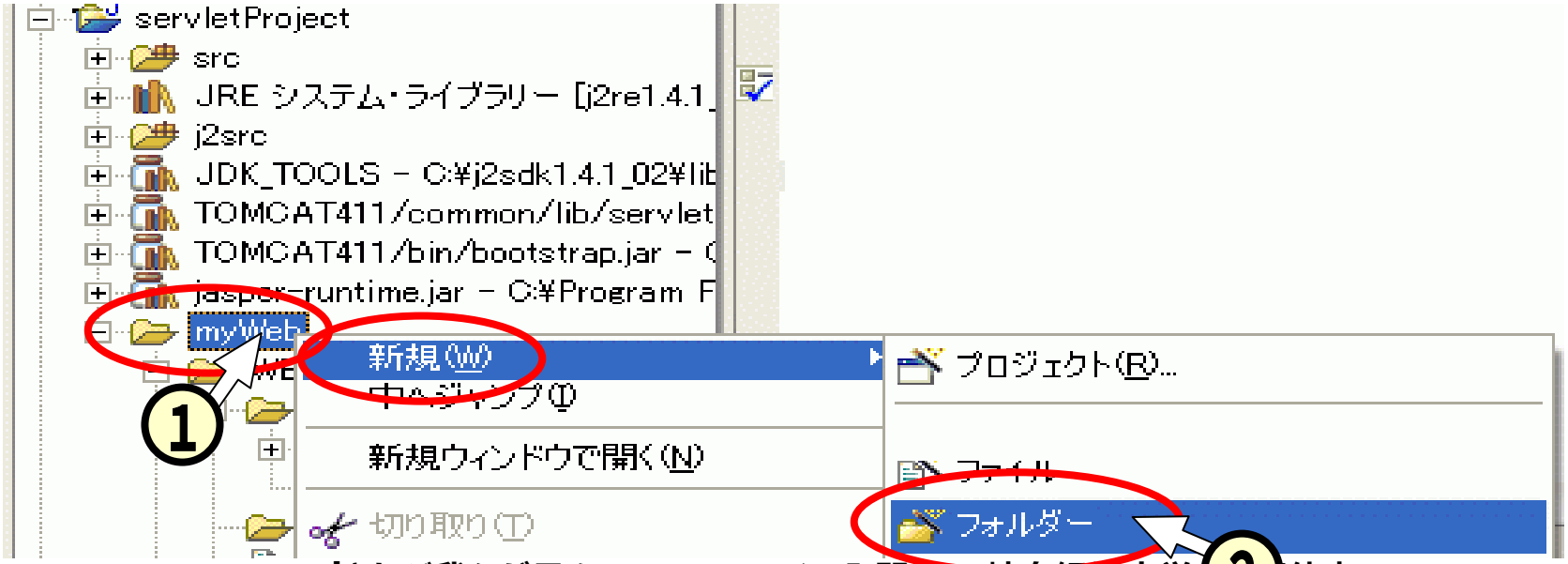
■スライド(6.2)で説明した、
JSP用のディレクトリ
(フォルダ)を作ります。

■「パッケージエクスプローラ」

■上で、“myWeb”を右クリック (①)

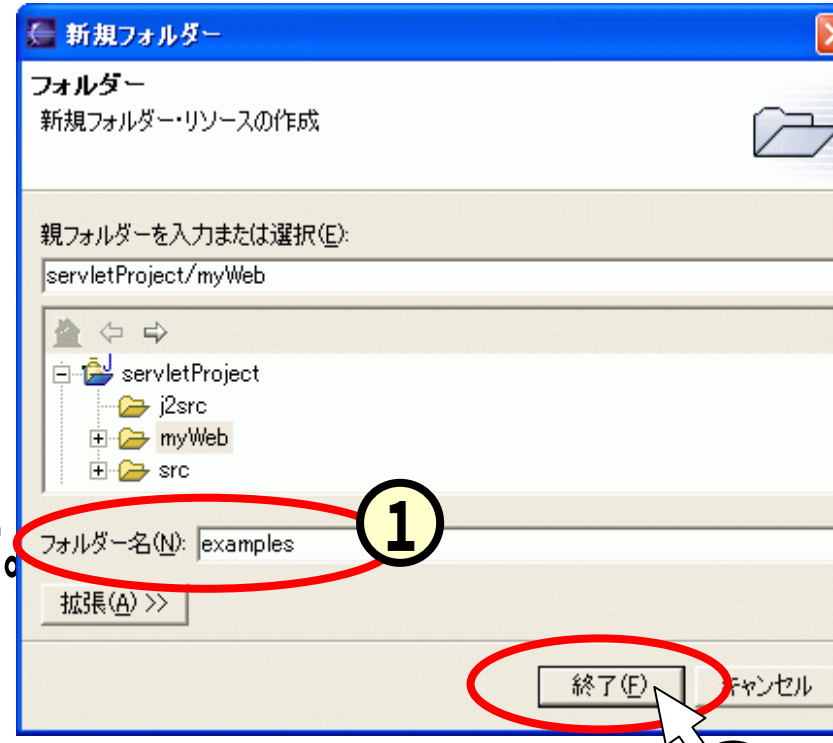
■し、[新規]-[フォルダー]をクリック (②) します。

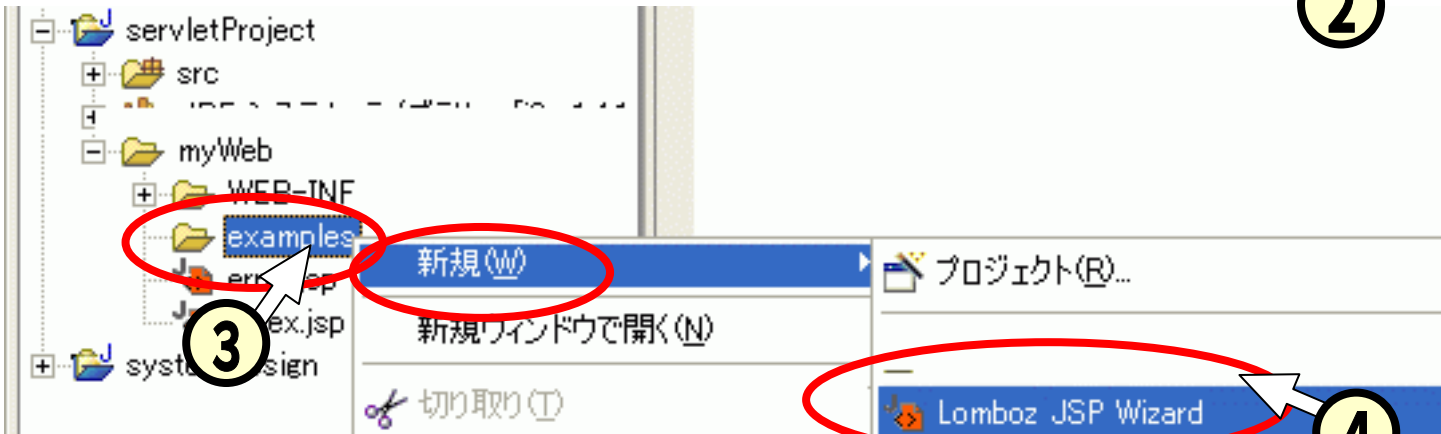




（6．5．2）JSP用のフォルダ、ＪＳＰの作成

- 「新規フォルダー」ウィンドウにて、[フォルダー名]に“examples”と入力（①）し、[終了]をクリック（②）します。
- ＪＳＰを作成するには、今作った“examples”のフォルダーを「パッケージエクスプローラ」中で右クリック（③）し、[新規]-[Lomboz JSP Wizard]をクリック（④）します。
- あとは、「eclipseによるJavaサーブレットの作成」のスライド（3）と同等の操作を行います。





(6．6) login.jsp

■下線部以外は、学習済みであり、理解出来ると思います（下線部については、後で説明します）。

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html>
<head><title>Login Page</title></head>
<body bgcolor="white">
<h2 style="color:white;background-color:#0086b2;">
        Login Page</h2>
<% String error = (String)request.getAttribute("error");
   if(error!=null){ %>
    <%= error %>
<% }%>
<form action="/idoApp/welcome">
name:<input type="text" name="username" size="15" /><br>
password<input type="password" name="passwd" size="15" /><br>
<input type="submit" value="送信"><br>
</form>
</body>
</html>
```

各自で変わります

（7）Welcome.java

■Welcome.javaファイルの内容を、次スライドに亘って示します。次スライドには、doGetの中身のみを示します。

```
package examples;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Welcome extends HttpServlet {
  private String sitePasswd;
  public void init(ServletConfig config)
                              throws ServletException {
    super.init(config);
    sitePasswd = this.getInitParameter("site-passwd");
  }
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws IOException, ServletException
  {
      // 次スライドに示す。
  }
}
```

(7 つづき) Welcome.java

```
request.setCharacterEncoding("UTF-8");
String strName = request.getParameter("username");
String strPasswd = request.getParameter("passwd");
if(!strPasswd.equals(sitePasswd)){
  request.setAttribute→
←("error","Sorry,password is incorrect.");
  RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher→
←("/examples/login.jsp");
  dispatcher.forward(request, response);
  return;
}
if(strName==null){
  strName = "a Mr. or Ms. Unknown";
}
request.setAttribute("name",strName);
RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher→
←("/examples/welcome.jsp");
dispatcher.forward(request, response);
```

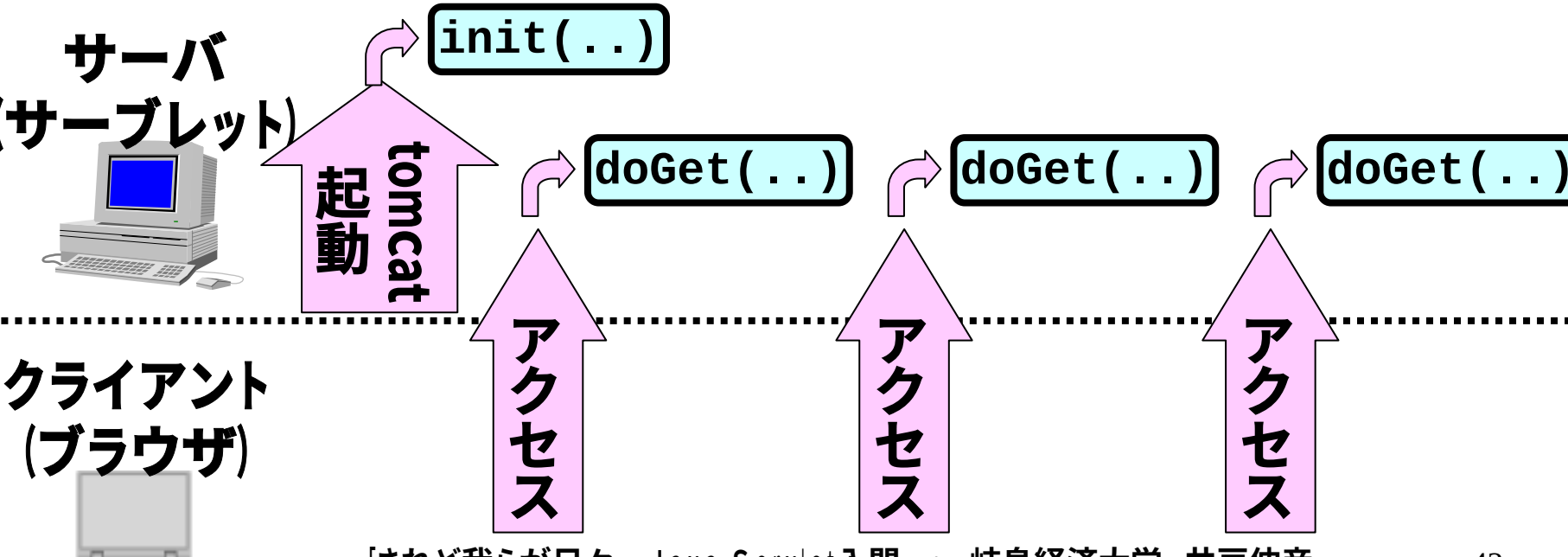
→から←の部分は、実際には改行しません。

(7．1) initメソッド

■今回は、initメソッドもオーバーライドします。

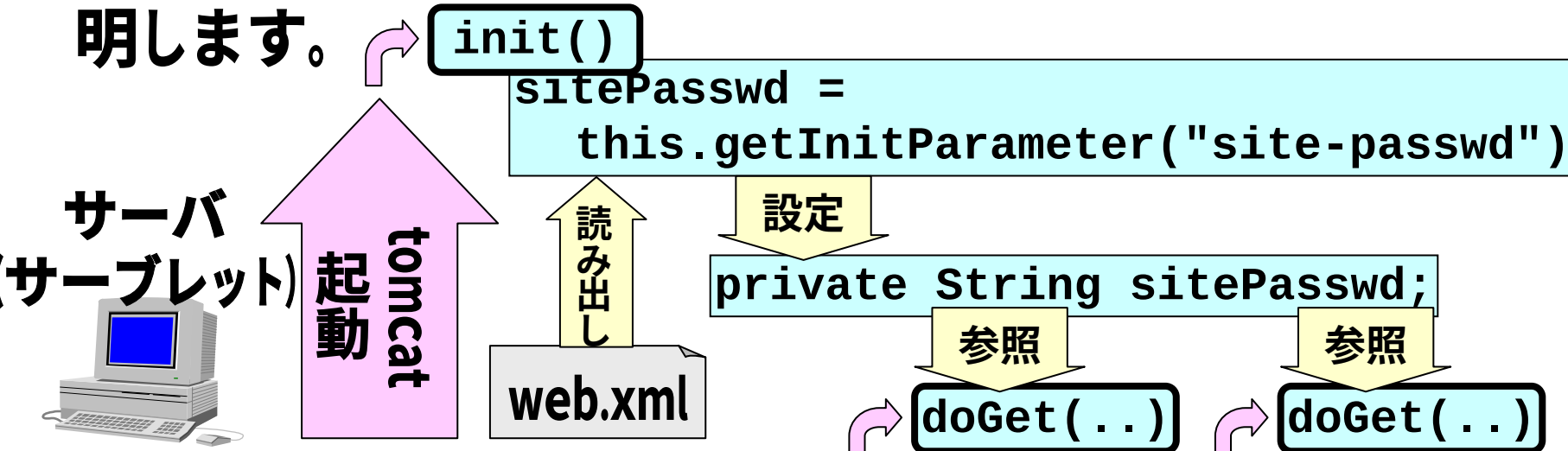
```
private String sitePasswd;
public void init() {
  sitePasswd = this.getInitParameter("site-passwd");
}
```

■スライド(2.5)に示したとおり、initメソッドはTomcatの起動時にのみ呼び出されます。



(7．2) 初期化パラメータの読み出し

- initメソッドでは、初期化パラメータを読み出して、変数“sitePasswd”に設定しています。後で見るdoGetメソッドで、この“sitePasswd”を参照して使っています。
- 初期化パラメータは、web.xmlのファイル内に、パラメータ名とその値が組で記述されています。
- この値を読み出すのが、“getInitParameter”です。
- 初期化パラメータについては、web.xmlの項で再度説明します。



(7．3) doGetメソッドの概要

■doGetメソッドの概要を、ＰＡＤで示します（ＰＡＤについては構造化プログラミングスライドを参照してください）。

■以下、メインの処理である、①②③を説明します。

始まり

①クライアントから送られたデータ（名前、パスワード）を取り出す。(7.4)

Yes

パスワードがsitePasswdと等しくないか？

request中に、errorとして、“Sorry,password is incorrect.“を設定

ログイン画面(JSP)に処理を投げる

終了

Yes

名前が設定されていないか？

名前に、”a Mr. Unknown”を設定

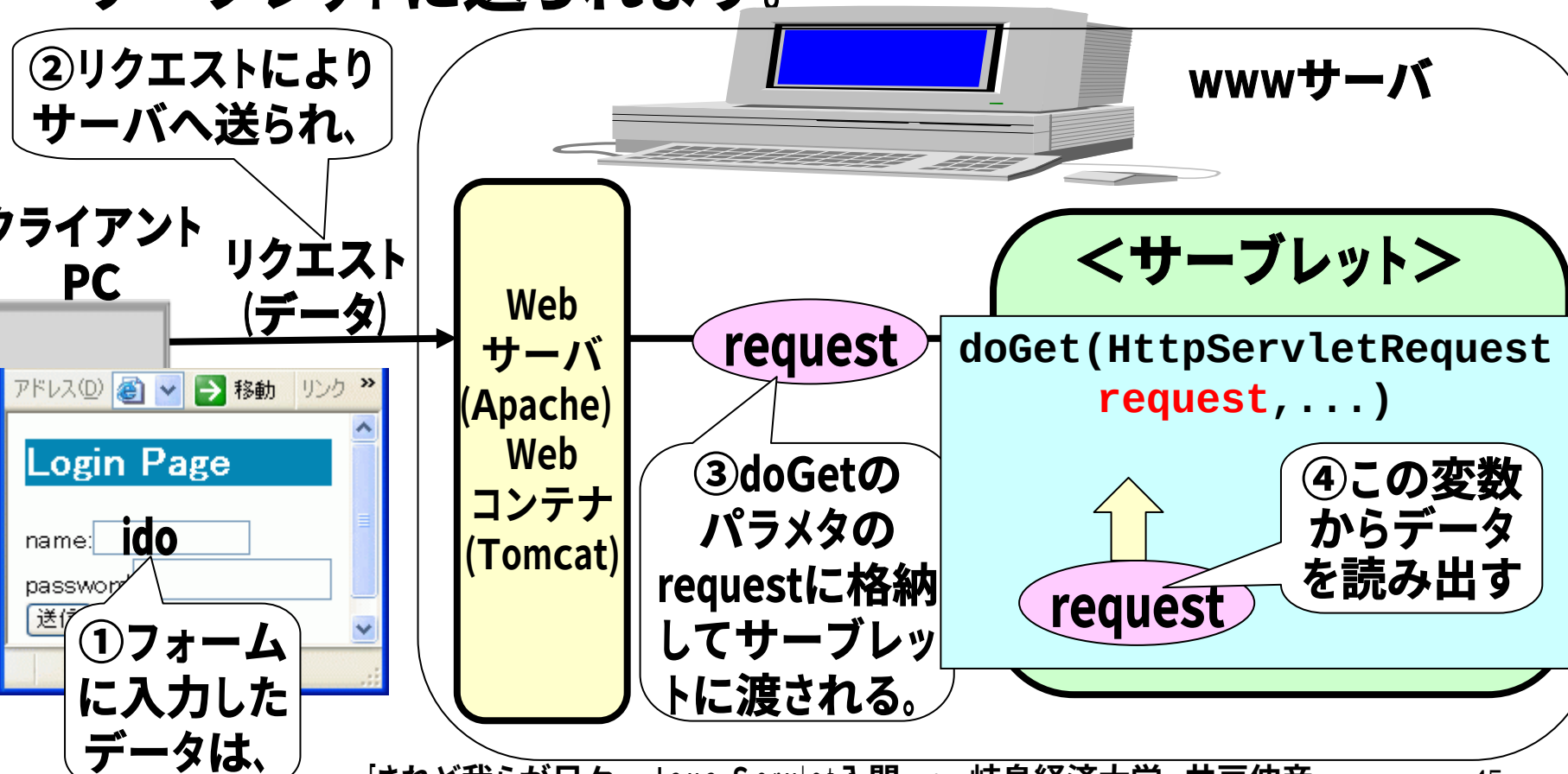
②request中に、nameとして、名前を設定。(7.5.3) (7.5.4)

③ウェルカム(welcome)画面(JSP)に処理を投げる。(7.5.1) (7.5.2)

終わり

(7．4．1) クライアントからのデータ

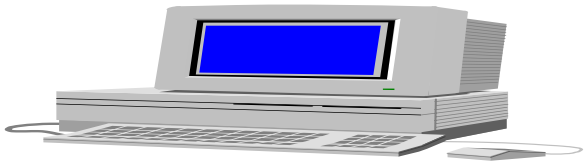
■クライアントからのデータは、doGetメソッドのインプットパラメータである、HttpServletRequestクラスのインスタンス (通常、“request”という名前の変数) に入れてサーブレットに送られます。



(7．4．2) リクエスト･データの読み出しの例

■前スライドに記した仕組みにより、Welcomeサーブレットでは、次のようなデータの読み出しが行われます。

クライアント
PC



wwwサーバ

```
strName ← ido
strPasswd ← abcd
```

getParamter

?username=ido
&passwd=abcd

request
“username”→ido
“password”→abcd

<login.jsp>

```
<input type="text"
       name="username" />
<input type="password“
       name="passwd" />
```

<welcome.java>

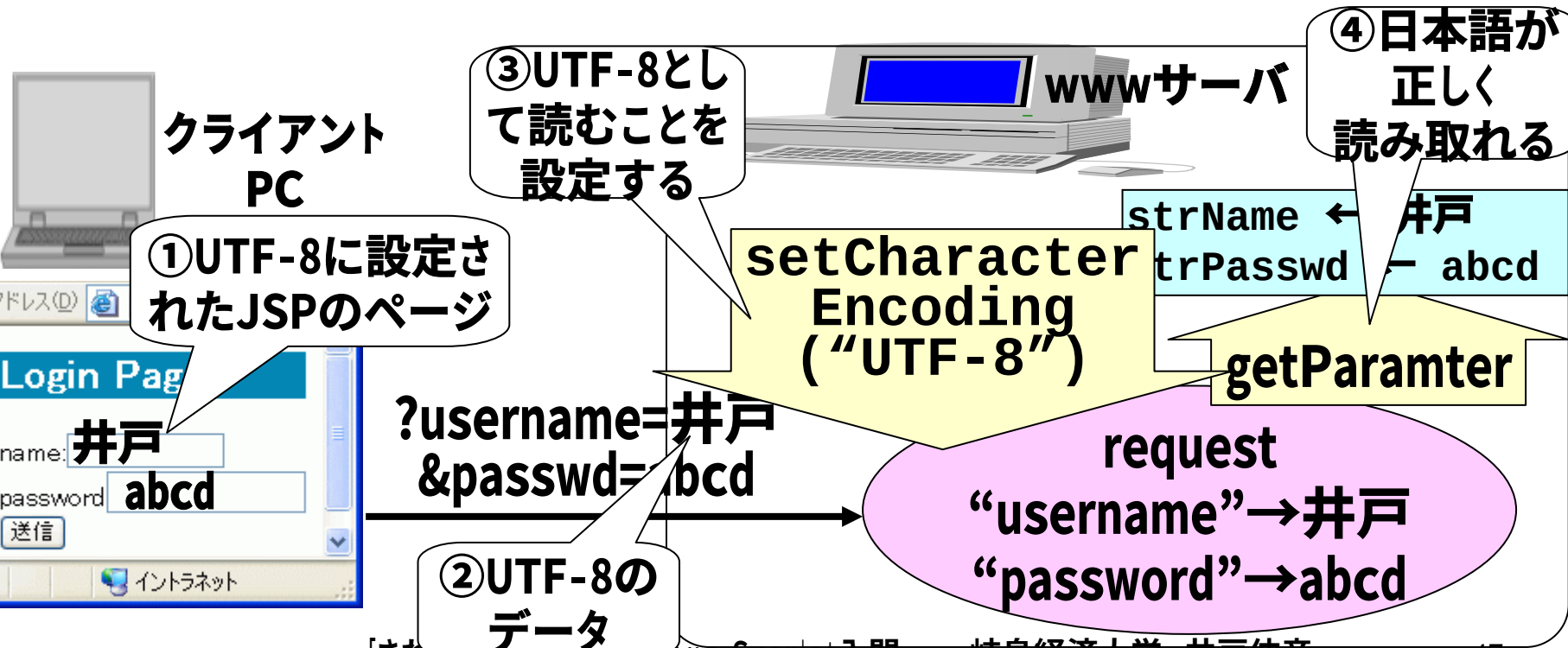
```
String strName =
 request.getParameter("username");
String strPasswd =
 request.getParameter("passwd");
```

(7．4．3) 日本語のエンコード

■本スライドを含めた一連のスライドでは、日本語のエンコードをUTF-8に統一しています。すなわち、login.jspもUTF-8のエンコードで入力を送ってきます。

■この日本語を読み取るために、次の行があります。

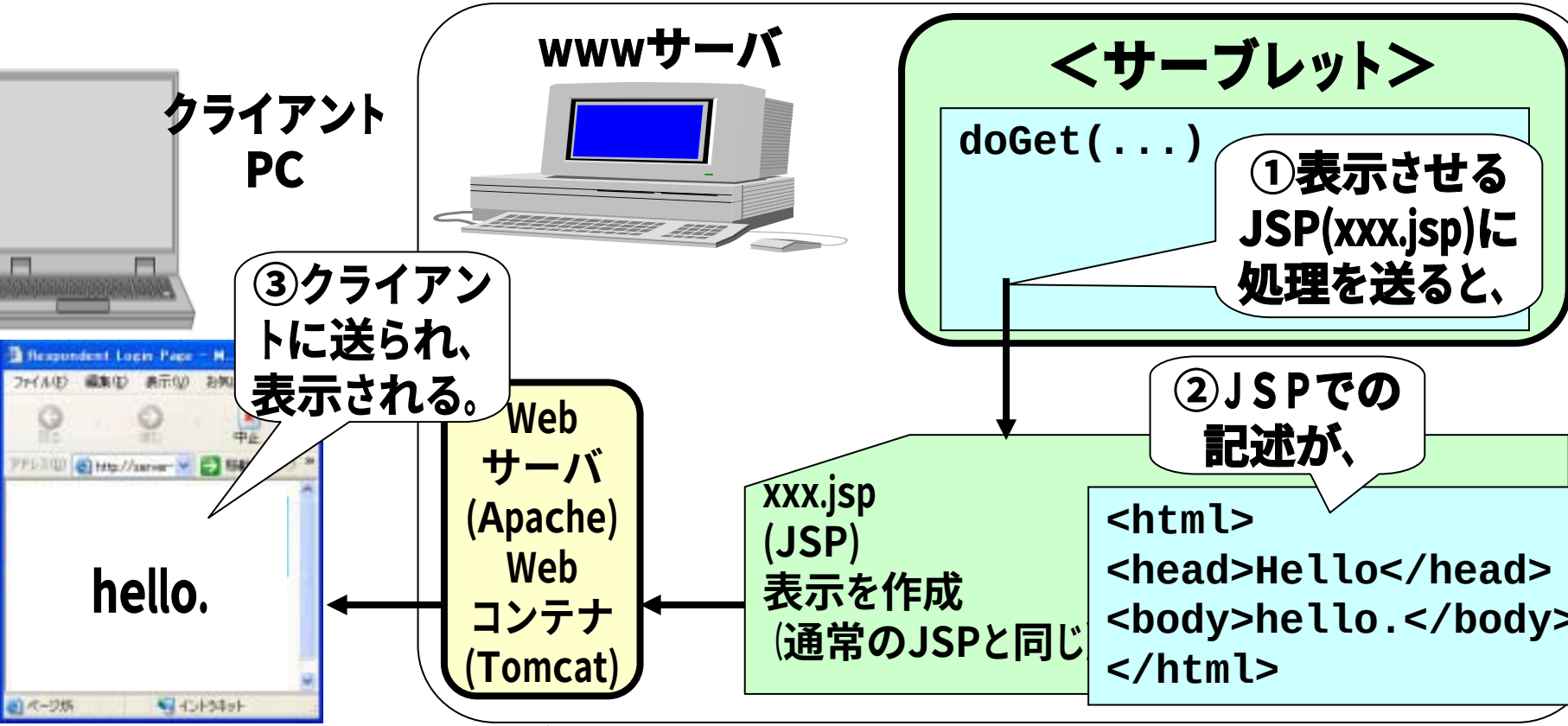
```
request.setCharacterEncoding("UTF-8");
```



(7．5．1) JSPによるページの表示

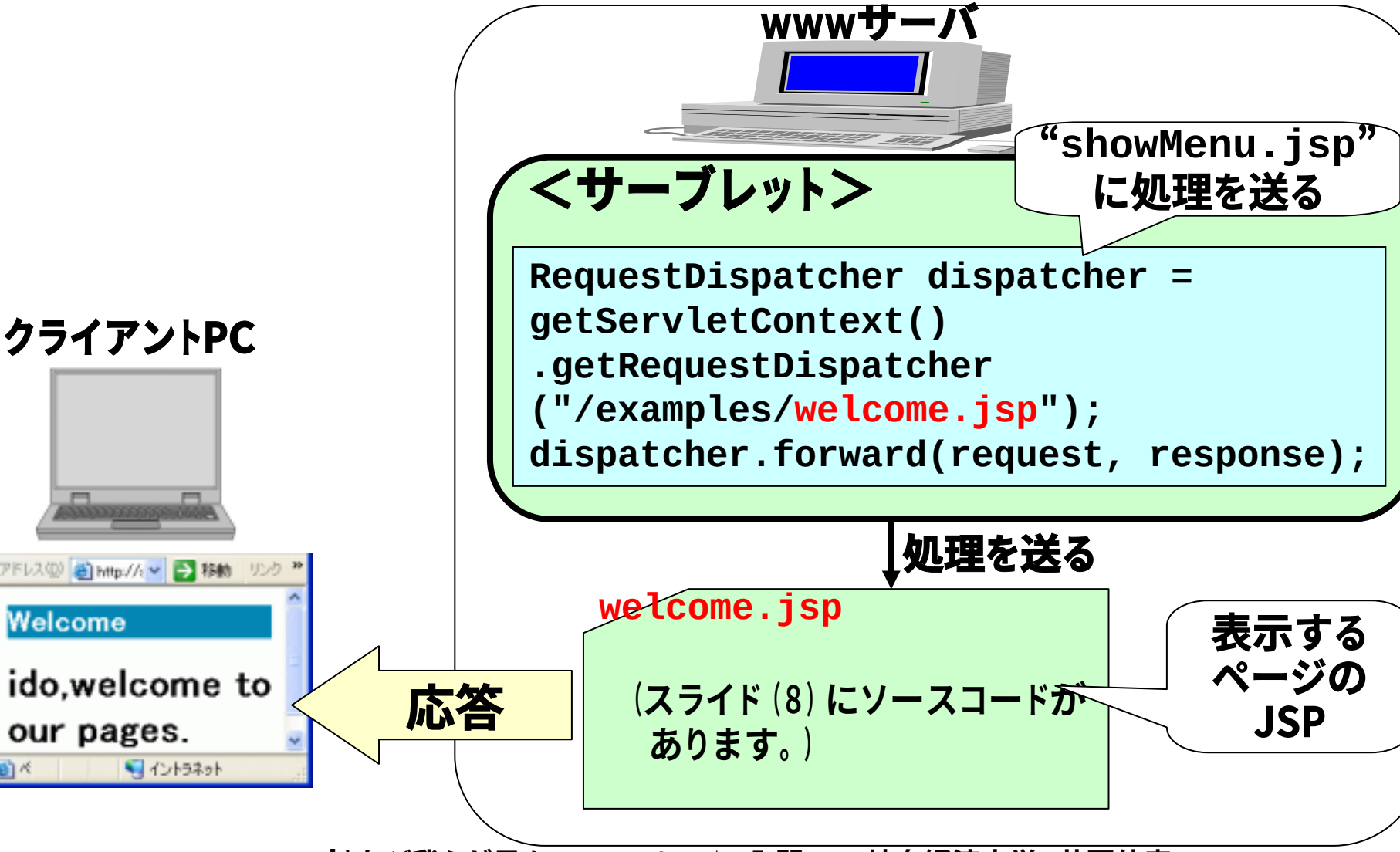
■Welcomeサーブレットでは、JSPを用いて表示を行います。

■サーブレットは、JSPを指定して処理を送ることで、ページの表示が出来ます。



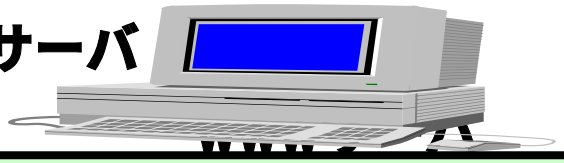
(7．5．2) JSPによる表示の例

■前スライドのコーディングは、次のようになります。



(7．5．3) JSPのデータの引渡し

■サーブレットからJSPへの表示用データも、“request”により引き渡します。



＜サーブレット＞

```
doGet(...){
```

①ＪＳＰに渡すデータを、requestに設定し、

request

②ＪＳＰを起動すると、

クライアントPC

＜JSP＞

<%

③スクリプトレット内で、requestから読み出し、

%>

request

④表示に反映させる。

応答

(7．5．4) JSPのデータの引渡しの例

■welcome.jspのソースは、
スライド (8) にあります。

①“name”という名前のデータに、strNameの値(=ido)を設定する

＜サーブレット＞

setAttribute

request
“name”→ido

```
request.
setAttribute
("name",strName);
```

クライアントPC

②ＪＳＰを起動する

welcome.jsp

③“name”という名前のデータを取り出しunameに設定する

⑤表示用データの読み出し

getAttribute

request
“name”→ido

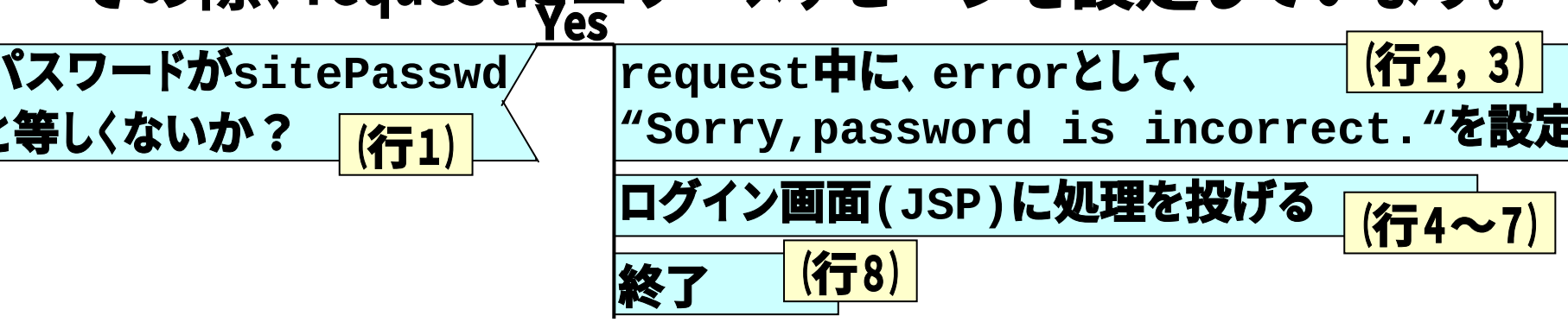
```
<% String uname =
(String)request.
getAttribute("name");
%>
<h1><%= uname %>,
welcome to our pages.</h1>
```

④unameを表示する

応答

(7．6．1) パスワードNG時の処理

■パスワードが一致しない場合、スライド(7.5)に記した方法により、login.jspに表示を送り、処理を終わります。その際、requestにエラーメッセージを設定しています。




```
1. if(!strPasswd.equals(sitePasswd)){
2.    request.setAttribute→
3. ←("error","Sorry,password is incorrect.");
4.    RequestDispatcher dispatcher =
5.    getServletContext().getRequestDispatcher→
6. ←("/examples/login.jsp");
7.    dispatcher.forward(request, response);
8.    return;
9. }
```

(7．6．2) login.jspでの処理

■スライド(6.3)で説明を先送りしたlogin.jspでの処理は、次のことを行っています。

- (行1) エラーメッセージを読み出す。
- (行2) エラーメッセージが設定されているかを判定。
- (行3) 設定されている場合、これを表示。

```
. <% String error = (String)request.getAttribute("error")
.    if(error!=null){ %>
.     <%= error %>
. <% }%>
```

(8) welcome.jsp

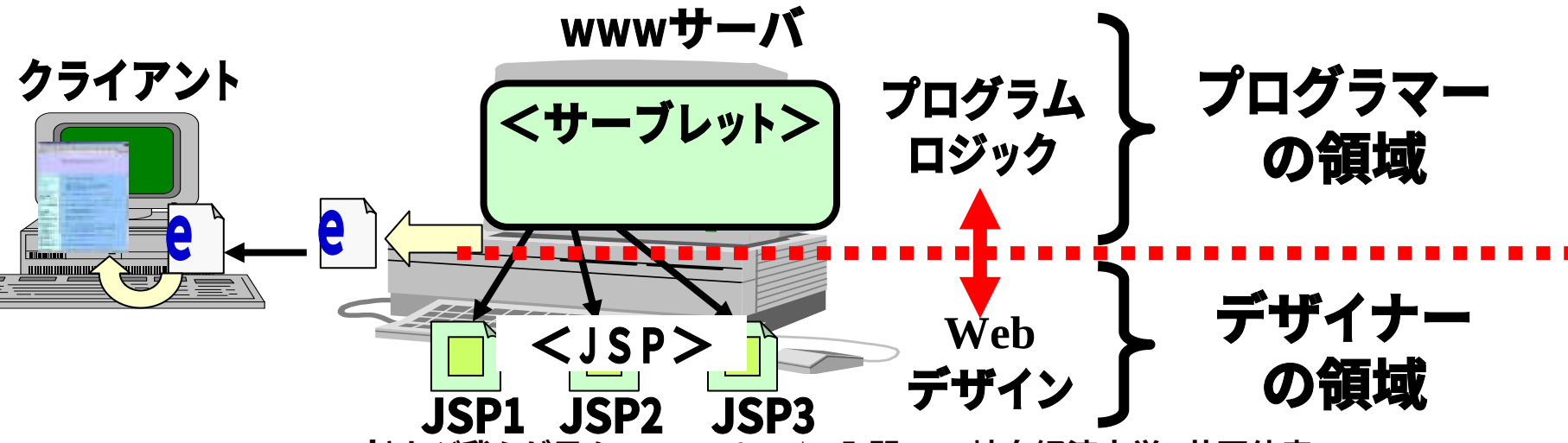
■サーブレットから送られたデータの読み出しは、スライド(7.5.4)で説明しました。

```
<%@ page contentType="text/html;charset=iso-2022-jp" %>
<html>
<head>
<title>Welcome Page</title>
</head>
<body bgcolor="white">
<h2 style="color:white;background-color:#0086b2;">
        Welcome</h2>
<% String uname = (String)request.getAttribute("name");
%>
<h1><%= uname %>,welcome to our pages.</h1>
</body>
</html:html>
```

(9．1) なぜservlet+JSPとするのか？

■Webサイトのデザインは、ビジネスに関わる大切な課題ですが、これは、本職のデザイナーが行うべきです。これに対して、プログラムロジックはプログラマーの領域です。

■デザインの部分はJSPに追い出してデザイナーに任せ、プログラムの部分はサーブレットとしてプログラマーが担当する。これが、servlet+JSPを用いる理由です。



(9．2) ＪＳＰでのタグライブラリ

- 「デザインの領域をＪＳＰに分離した」と言っても、実際には、ＪＳＰのスクリプトレットはプログラムに他なりません。デザインとプログラムとの分離は不完全です。
- ＪＳＰにタグライブラリを用いると、スクリプトレットを使わないように出来ます。タグライブラリについては、別のスライドにて説明します。

```
<logic:iterate id="inquery"
    name="inqueryList"
    property="inqList">

</logic:iterate>
```

タグライブラリを使用

```
<% InquiryList inqueryList
   = (InquiryList)request→
ー.getAttribute("inqueryList");
   ArrayList ar
   = (ArrayList)inqueryList.getInqList();
   Iterator itr = ar.iterator();
   while(itr.hasNext()){
     InquiryWithResponses inquery
     = (InquiryWithResponses)itr.next();
%>
<% } %>
```

同じことをスクリプトレットで実現

(10) web.xml

■Welcomeサーブレットに関するweb.xmlの設定は、①クラスの指定のしかた、②初期化パラメータの部分以外は、HelloWorldサーブレットと同じです。

```
<web-app>
    <servlet>
        <servlet-name>welcomeServlet</servlet-name>
        <servlet-class>examples.Welcome</servlet-class>
        <init-param>
            <param-name>site-passwd</param-name>
            <param-value>aaaaaaaa</param-value>
        </init-param>
    </servlet>
<servlet-mapping>
        <servlet-name>welcomeServlet</servlet-name>
        <url-pattern>/welcome</url-pattern>
    </servlet-mapping>
</web-app>
```

クラスの指定

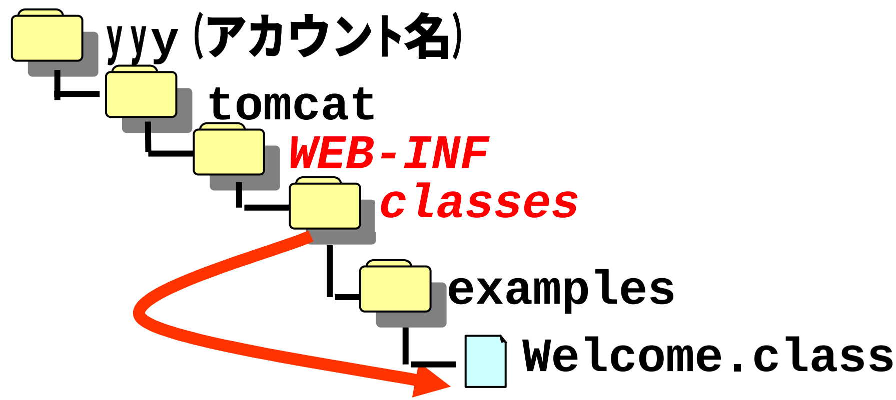
初期化パラメータ

（１０．１）クラスの指定

■クラスの指定は、“examples.Welcome”となります。

```
<servlet>
     <servlet-name>welcomeServlet</servlet-name>
        <servlet-class>examples.Welcome</servlet-class>
</servlet>
```

■上記の、クラス名“examples.Welcome”は、javacコマンドでパッケージ内のＪａｖａファイルをコンパイルした後、javaコマンドで実行する際の名前と同じです（実際はサーブレットはjavaコマンドで実行できません）。パッケージについては、Ｊａｖａ教科書14章を参照してください。



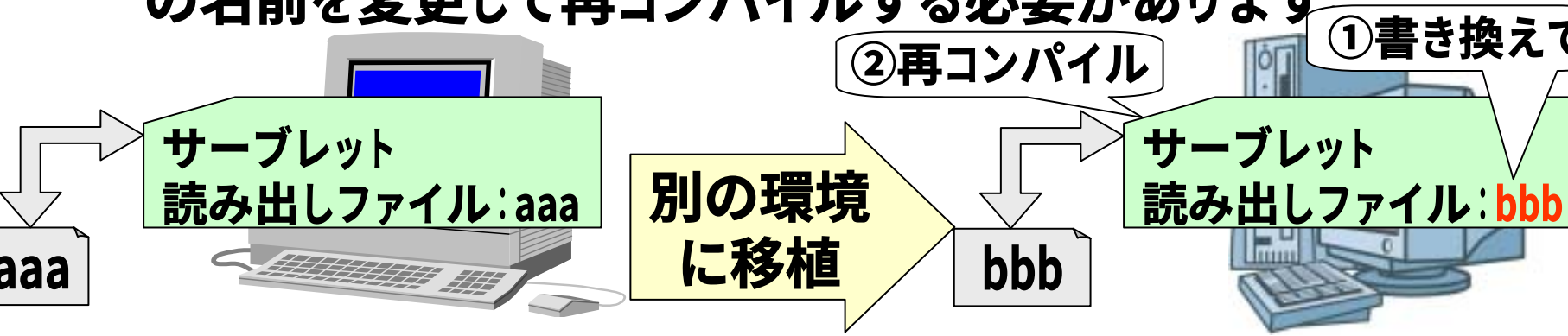

```
% cd classes
＜コンパイル＞
% javac examples/Welcome.java
＜実行（実際は実行できない）＞
% java examples.Welcome
```

これがクラス名

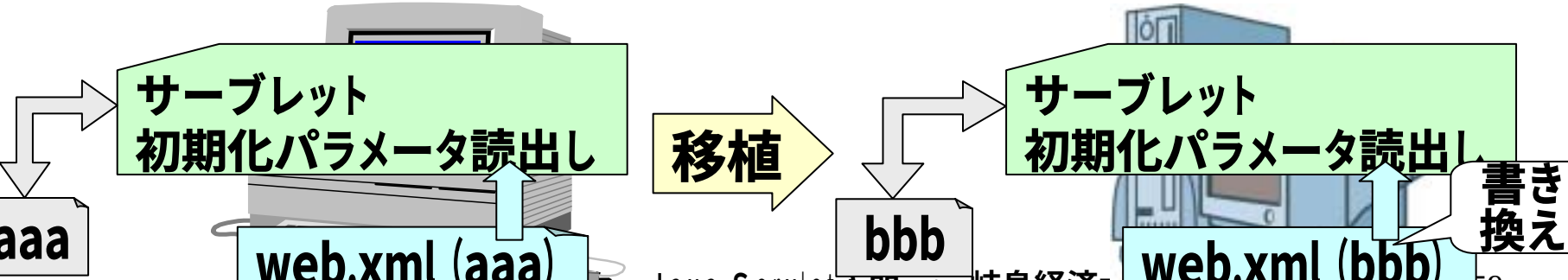
(10.2.1) 初期化パラメータの必要性

■なぜ初期化パラメータを使うのか？

- 例えば、プログラムの中にアンケートファイルのパス名を書き込んでしまうと、別の環境でサーブレットを動かす場合、その名前を変更して再コンパイルする必要があります



- web.xml中の初期パラメータは、サーブレットから一定の方法で読み出すことが出来ます。上記のパス名をここに記しておけば、別の環境に移植しても、再コンパイルは不要です。



(10.2.2) 初期化パラメータ

■web.xmlファイル中の初期化パラメータの設定と、サーブレットでの読み出しは、次のように行います。

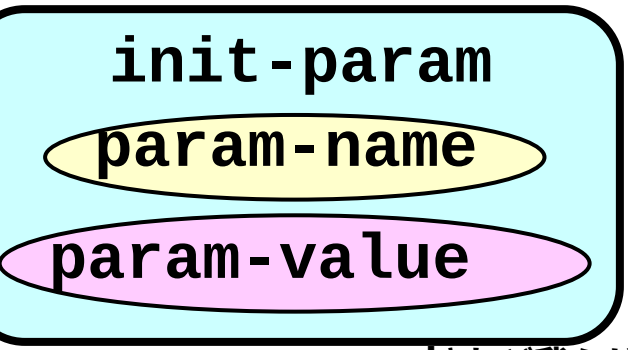
サーブレットの初期化パラメータ読出し

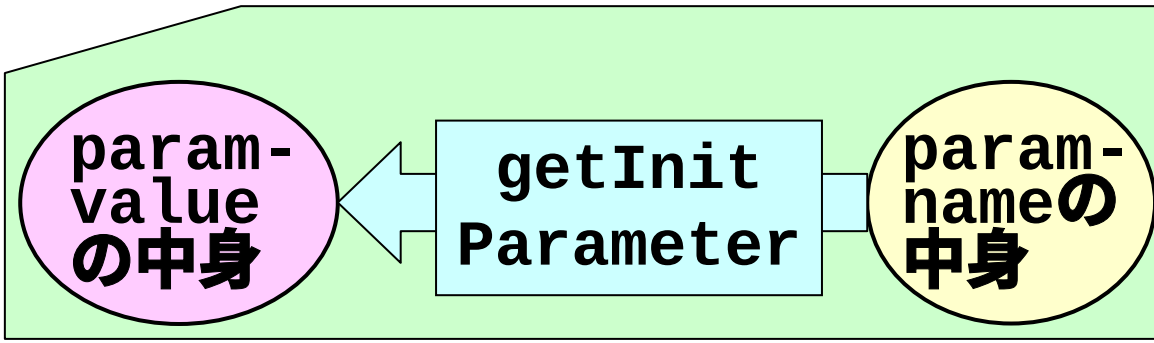
```
sitePasswd = this.getInitParameter("site-passwd");
```

"mypassword"という値が設定される

web.xmlファイル

```
<init-param>
  <param-name>site-passwd</param-name>
  <param-value>aaaaaaaa</param-value>
</init-param>
```



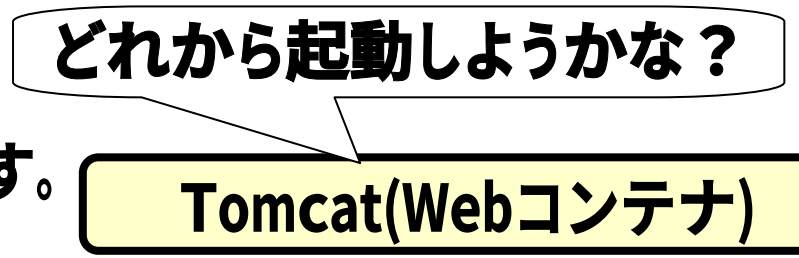


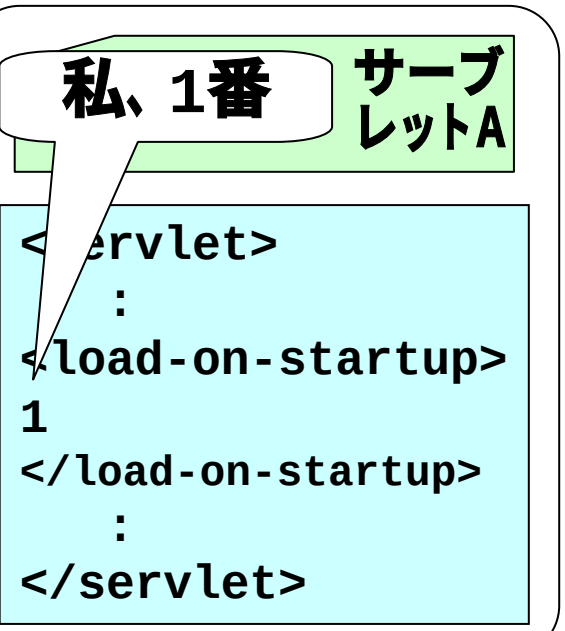
(10．3)起動順序

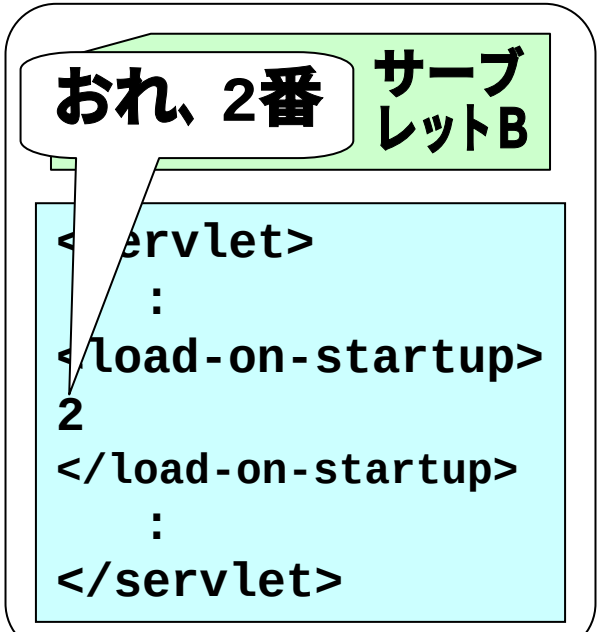
■複数のサーブレットがある場合、tomcat初期化起動時のサーブレット(=init)の起動順序を指定することが出来ます。

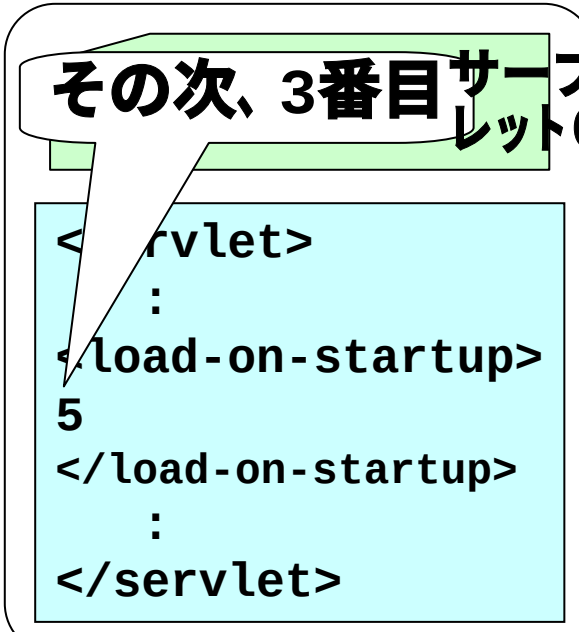
■サーブレット要素の中に、図のようにして<load-on-startup>要素を追加します。

- 整数値を指定します。
- 小さい値ほど、先に起動されます。









(11．1) Java Beansの機能

■直感的には、「タグ(ＪＳＰ)の世界とプログラム(Javaサーブレット)の世界とをデータでつなぐ仕組み」と言えます(タグ：直感的には”<“と”>”で囲んだ書式の命令)。

■すなわち、Java Beans上の同じデータに、

- ＪＳＰではタグでアクセスでき、
- Javaサーブレットからは、プログラムとしてアクセスできます。

```
<%@ page language="java" pageEncoding="EUC-JP" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>Lomboz JSP</titl
</head>
<body bgcolor="#FFFFFF"

<h2 style="color:white;
       Welcome</h2>
<h1><bean:write name="loginForm" property="name" />
,welcome to our pages.</h1>
</body>
</html>
```

タグの世界
(ＪＳＰ)

```
package welcome;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class WelcomeAction extends Action {
  private static final String sitePasswd = "aaaaaaa";
  public Action

    LoginForm l
if(loginForm.ge
      loginForm
    }
    request.setAttribute("loginForm",loginForm);
    return map.findForward("success");
  }
}
```

プログラムの世界
(Ｊａｖａサーブレット)

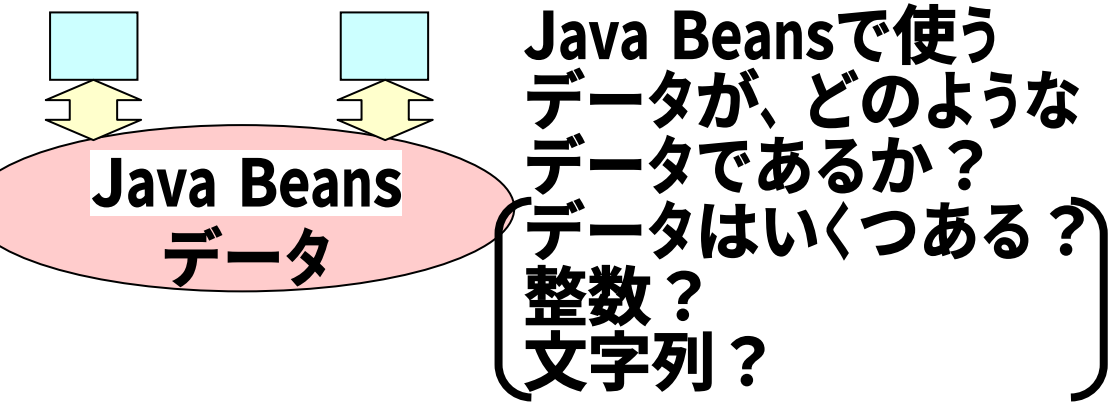
タグで、
データアクセス

Java Beans
データ

プログラム(クラス)
として、
データアクセス

(11．2) Java Beansクラス

■Java Beansで使用するデータが、どのようなデータであるかは、プログラム側でクラス(Java Beansクラス)として記述します。



■Java Beansクラスであるための条件

- 次のような条件になりますが、ここではあまり詳しく触れず、次のスライドで条件を満たすクラスを見ることにします。
  - ①引数のないコンストラクタが定義されている。
  - ②プロパティを参照するアクセサ・メソッドが定義されている。
  - ③Serializableインタフェースを実装する。

(11．3) Java Beansクラスの例

■整数変数“count”だけを持つ（ count ）、簡単なJavaBeansクラスです。

```
package examples;
import java.io.Serializable;

public class DataBeans implements Serializable {

  private int count;

  public void setCount(int count) {
    this.data = count;
  }

  public int getCount() {
    return count;
  }
}
```

③この記述で、Serializableインタフェースを実装していることになります。

宣言したこのデータをＪＳＰとサーブレットで共有します。

このメソッドを、setter(セッター)と呼びます。

このメソッドを、getter(ゲッター)と呼びます。

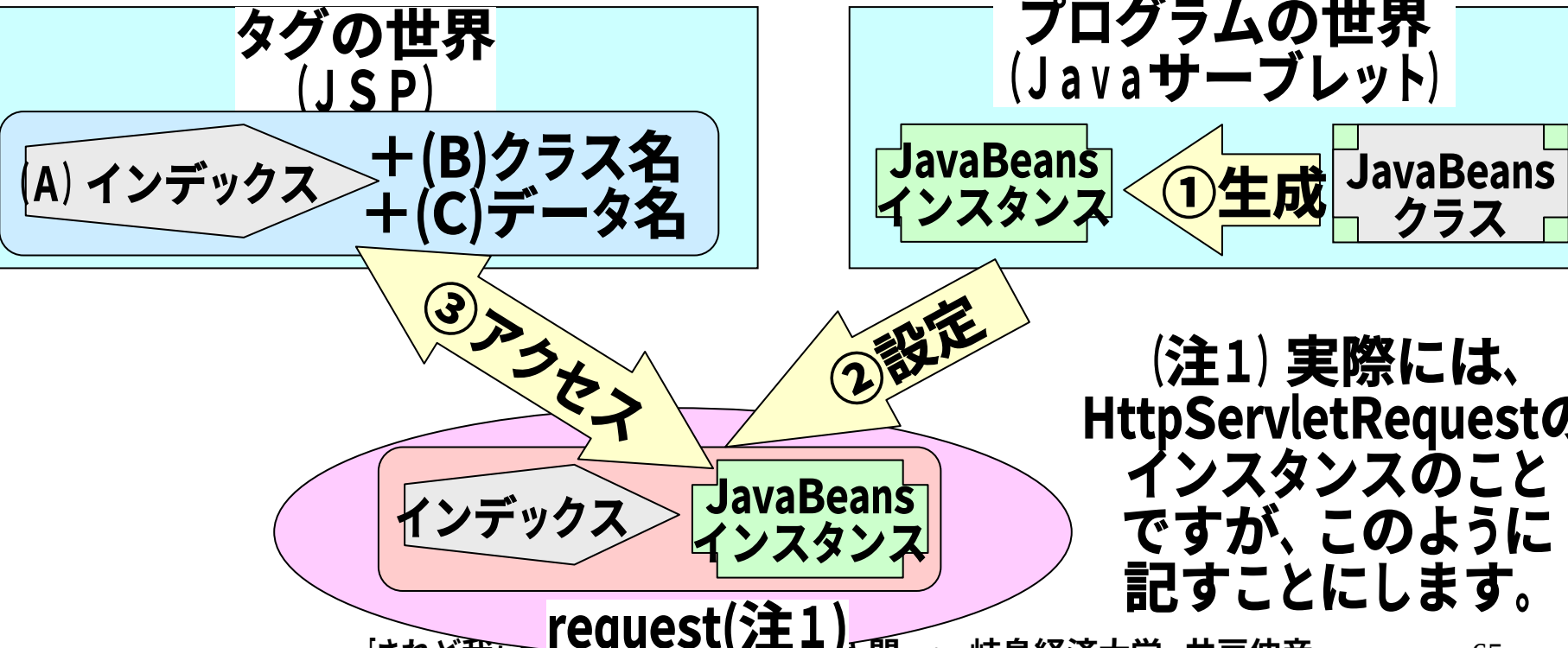
この2つのメソッドが、②アクセサメソッドとなります。

コンストラクタを記述しないと、①の条件を満たすことになります。
コンストラクタを作る場合は、注意しましょう。

(11．4) サーブレットからＪＳＰへ

■サーブレットからＪＳＰへデータを引き渡す場合、

- ①Java Beansクラス (前スライド) のインスタンスを生成します。
- ②そのインスタンスをインデックスをつけて、“request(注1)”に設定します。
- ③JSP側では、(A)インデックス・(B)クラス名・(C)クラスの中のデータ名を用いて、データにアクセスします。



(注1) 実際には、HttpServletRequestのインスタンスのことですが、このように記すことにします。

(11.5) サーブレット側の処理

```
ackage examples;
mport java.io.IOException;
mport javax.servlet.ServletException;
mport javax.servlet.http.*;

ublic class Welcome extends HttpServlet
  protected void doGet(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {

    DataBeans dataBeans = new DataBeans();

    dataBeans.setData(5);

    request.setAttribute("indA",dataBeans);

getServletContext().getRequestDispat   r("/examp
les/welcome.jsp").fo ward(request,re    se);
  }
```

①JavaBeansクラス“DataBean”のインスタンスを生成

Beans中のデータの値を設定

インデックス

②インデックス“indA”をつけて、インスタンスをrequestに設定

(11.6)JSP側の処理

```
<%@ page language="java" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0
transitional//en">
<html>
<head>
<title>Lomboz JSP
</head>
<body bgcolor="#FFFFFF">

<jsp:useBean id="indA"
             class="examples.DataBeans"
             scope="request" />

<p>Java Beans上のデータは、
<jsp:getProperty name ="indA"
                 property="count" />
です。</p>

</body>
</html>
```

JavaBeansを有効化するためのタグ

(A)インデックス

(B)クラス名

スコープ(後述)

(A)インデックス

(C)データ名

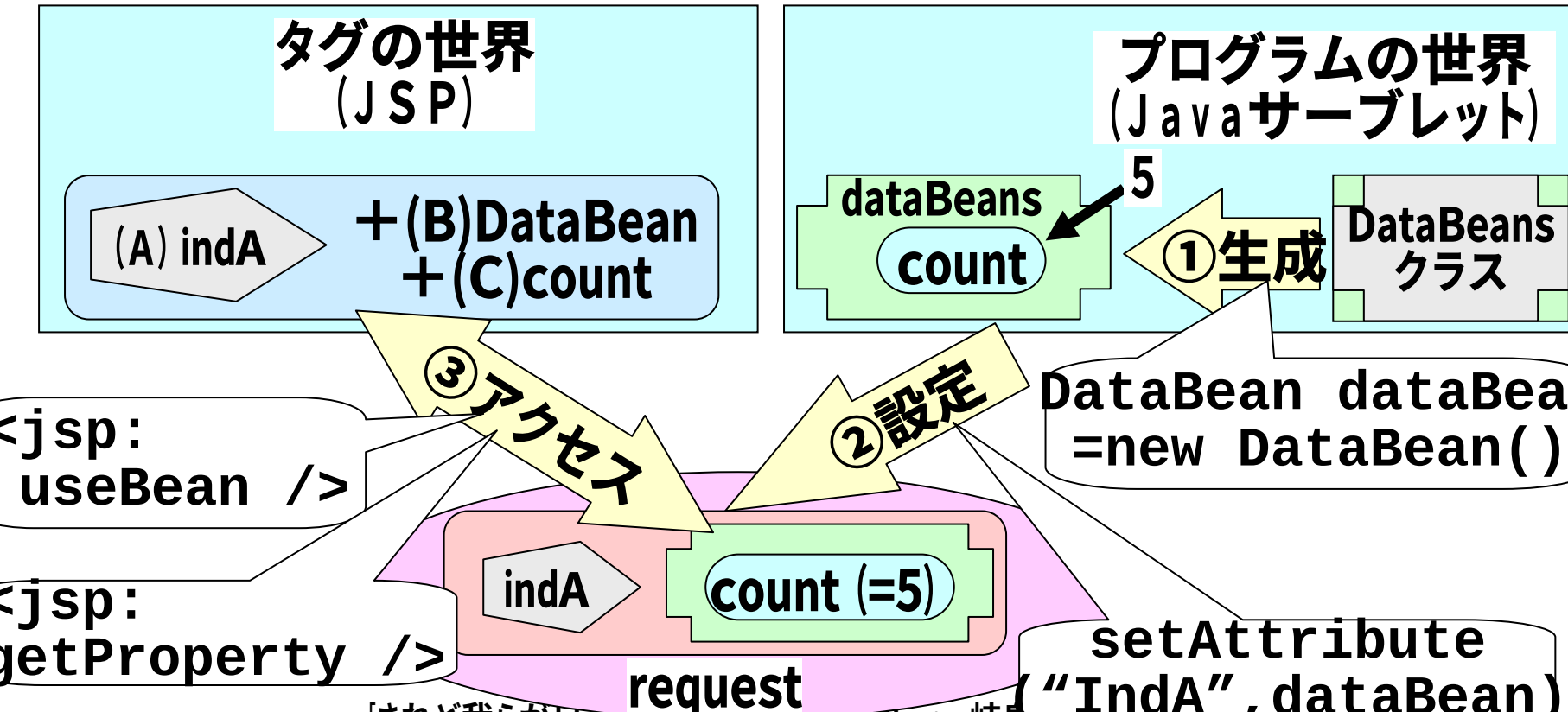
JavaBeansの値を参照するタグ

(11．7) サーブレット・ＪＳＰの動作

■このサーブレットにアクセスすると、次のように表示されます。

Java Beans上のデータは、5 です。

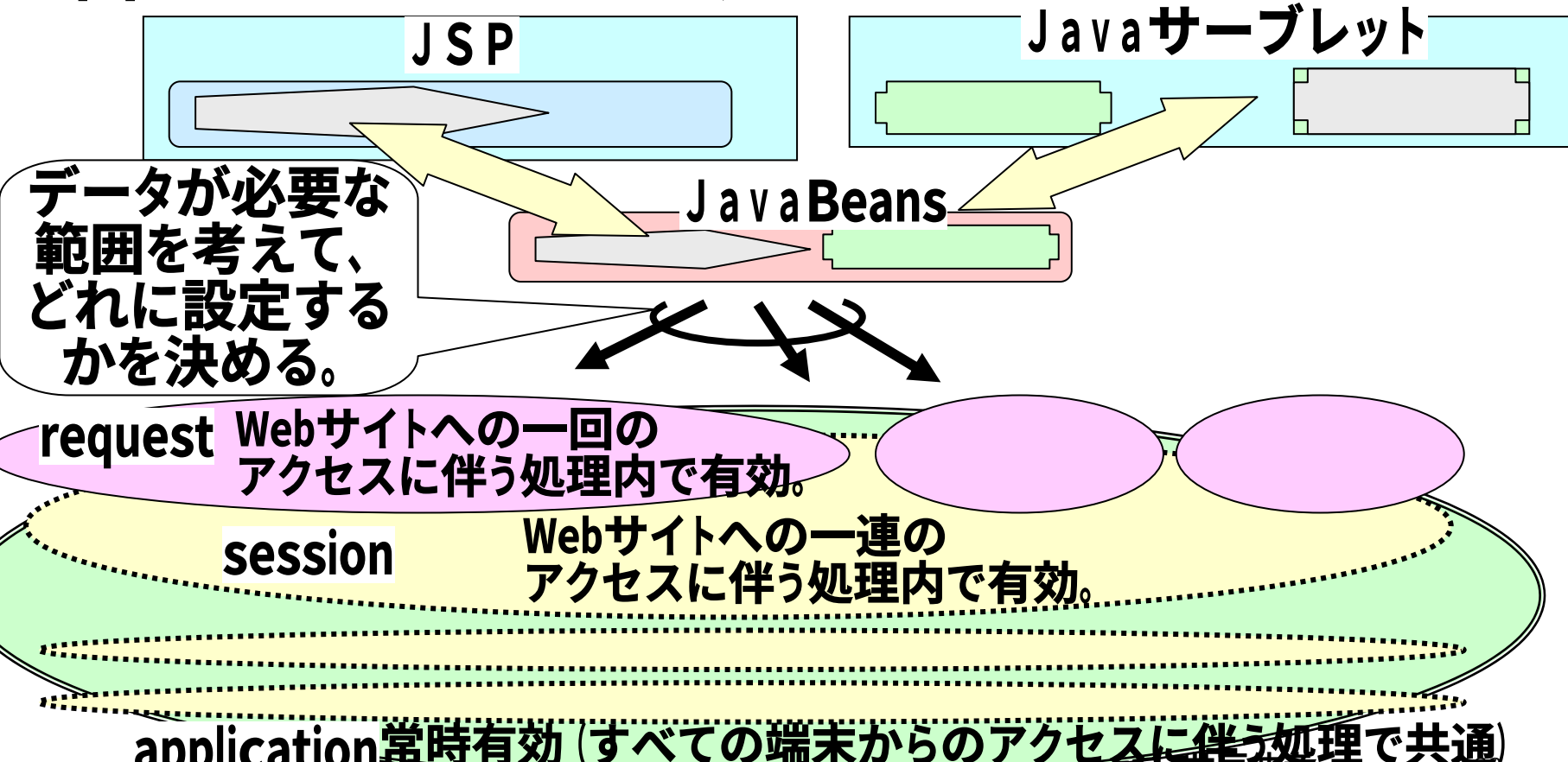
■“5”の値は、サーブレット中で設定している値です。



(11．8) スコープ

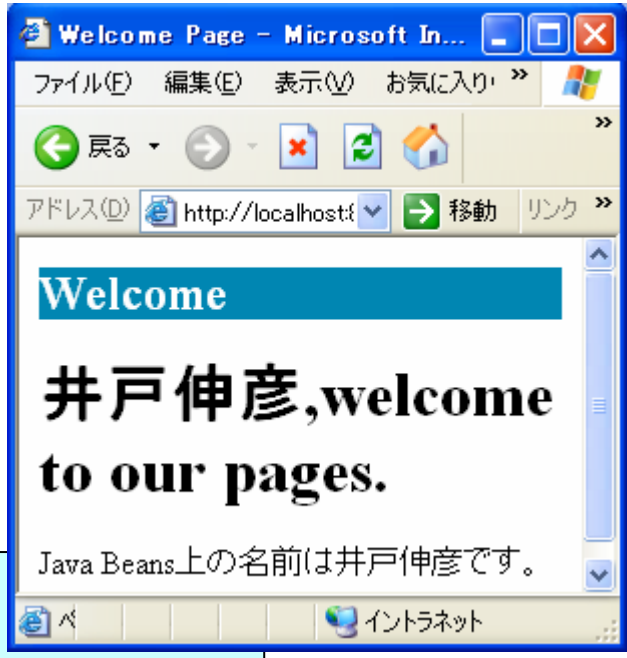
■Java Beansクラスのインスタンスを設定できるものには、page、request、session、applicationがあります。

■これらは、設定したBeans上のデータの値に、どの範囲でアクセスできるかに違いがあります。



(11．9．1) 例題：Java Beansによる表示

■Welcomeサーブレットでの
名前の表示を、Java Beansを
使って行ってください。
(右が表示イメージです)



■解答例：(次スライドに続く)

<NameBean.java>

```
package examples;
import java.io.Serializable;
public class NameBean implements Serializable{
   private String name;

   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
```

（11．9．2）例題：Java Beansによる表示

■解答例：（次スライドに続く）

<Welcome.java>

```
package examples;
(略)
public class Welcome extends HttpServlet {
（略）
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws IOException, ServletException
  {
（略）
    NameBean nameBean = new NameBean();
    nameBean.setName(strName);
    request.setAttribute("indNameBean",nameBean);
    RequestDispatcher dispatcher =
      getServletContext().getRequestDispatcher→
      ←("/examples/welcome.jsp");
    dispatcher.forward(request, response);
  }
}
```

(11．9．3)例題:Java Beansによる表示

■解答例

<welcome.jsp>

```
<%@ page language="java" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC
                "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>Welcome Page</title>
</head>
<body bgcolor="white">
<jsp:useBean id="indNameBean"
                class="examples.NameBean"
                scope="request" />
<h2 style="color:white;background-color:#0086b2;">
          Welcome</h2>
<% String uname = (String)request.getAttribute("name");
%>
<h1><%= uname %>,welcome to our pages.</h1>
<p>Java Beans上の名前は
<jsp:getProperty name ="indNameBean"
                    property="name" />です。</p>
</body>
</html>
```

（１２）演習：課題３、課題４

- （課題３）Welcomeサーブレットを作成し、実行させてみてください。
  - 前述のとおり、web.xmlを編集した場合、Ｔｏｍｃａｔの再起動が必要です。
- （課題４）ＪＳＰの学習で作成した、簡易掲示板を、サーブレット＋ＪＳＰで作成してください。