令和 6 年度文部科学省委託「専修学校による地域産業中核的人材養成」事業

情報 DX エンジニア教材資料

JDBC プログラミング
サーブレット／ JSP

IT 分野 DX 人材養成のモデルプログラム開発と実証事業

目 次

JDBC プログラミング

1.DB 連携

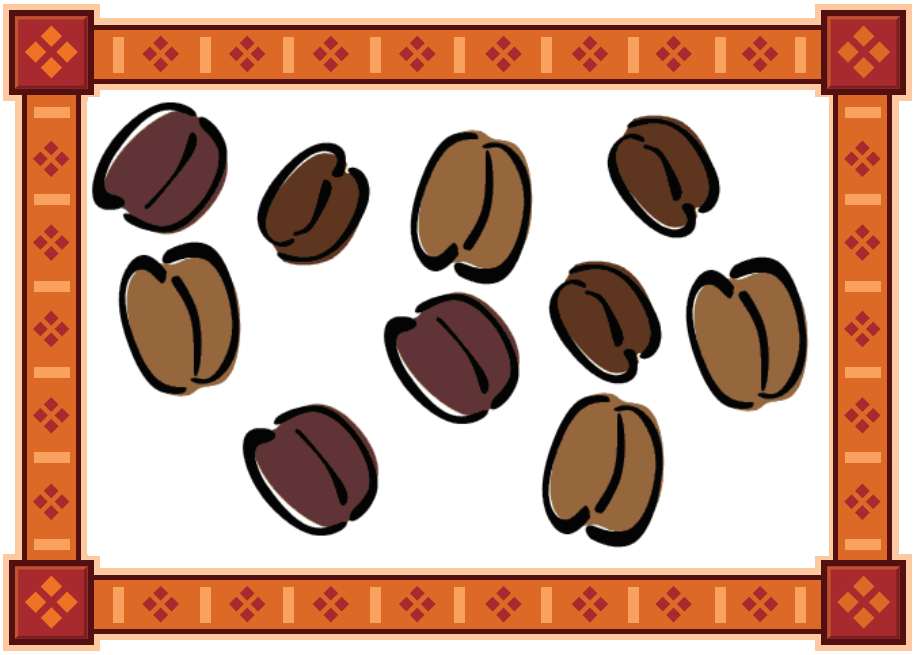
付録

サーブレット ／ JSP

1. サーブレット ／ JSP

付録

JDBCプログラミング



１．DB連携

１－１　DB連携の仕組み

１－２　JDBCを利用したDBアクセス

１－３　DBアクセスの高速化

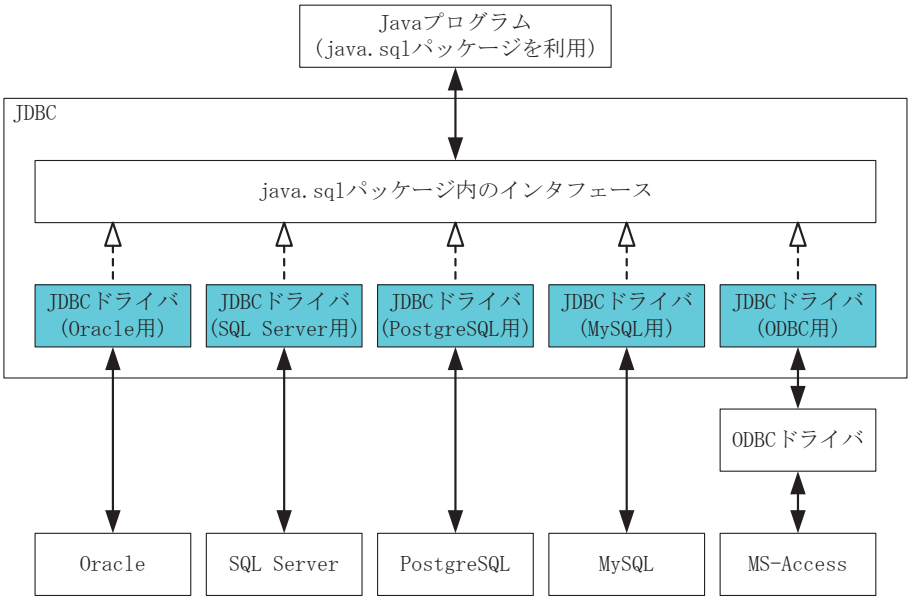
１－４　デザインパターンの導入

1-1　DB連携の仕組み

Java言語の標準クラスライブラリには，データベース（以下，DB）を利用するためのAPI（JDBC：Java DataBase Connectivity）が含まれています。JDBCは，プログラムがDBに接続する，プログラムがDBにSQL文を送信する，プログラムがDBから実行結果を受信するためのインタフェース及びクラス群です。

JDBCは，個々のDBによる違いを吸収するために，全DBに共通するメソッドを定義したインタフェース群（java.sqlパッケージ内のインタフェース）と，DBの仕様を考慮してインタフェースを実装したクラス群（各JDBCドライバ）によって構成されます。このため，JDBCを使用したJavaプログラムは，DBの種類によらない汎用性の高いものになります。

［さまざまなJDBCドライバ］



なお，MS-Accessなどの簡易DBは，JDBCから直接アクセスできません。そのため，Microsoft社が提供するODBC（Open DataBase Connectivity）を経由して簡易DBにアクセスします。

本教材では，JDBCを使用して，JavaプログラムからMySQLに接続するプログラムの作成方法を説明します。

JDBCの設定については，別冊「MySQLインストールマニュアル　実習環境の構築」を参照してください。

1-2　JDBCを利用したDBアクセス

Javaプログラム（Javaアプリケーション）から，DB（MySQL）にアクセスします。

Javaプログラムの作成手順は，次のとおりです。

〔手順〕

①java.sqlパッケージをインポートする。

例.
```
import  java.sql.DriverManager;      //JDBCドライバ管理
import  java.sql.Connection;         //DB接続管理
import  java.sql.Statement;          //SQL文の実行管理
import  java.sql.ResultSet;          //SQL文の実行結果
import  java.sql.SQLException;       //SQL関連の例外
```

②JDBCドライバをロードする。

java.lang.ClassクラスのforName()メソッドを呼び出して，使用するJDBCドライバをロードします。なお，Java仮想マシンは，コンパイル／実行時にjava.langパッケージを自動的にインポートします。

例.
```
:
class  SQLExecution
{
    public  static  void  main(String[]  args)
    {
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
        }
        catch(ClassNotFoundException e)
        {
            System.out.println(
                "JDBCドライバが見つかりません。");
            e.printStackTrace();
        }
    }
}
```

※Classクラスの forName()メソッドは，引数に指定されたクラスを探します。

※ClassNotFoundExceptionクラスは，クラス名の不明を表す例外です。

③DBMSに接続する（DBMSとのコネクションを確立する）。

java.sql.DriverManagerクラスのgetConnection()メソッドを呼び出して，DBMS（DataBase Management System）に接続します。

例.

```
class  SQLExecution
{
    public  static  void  main(String[]  args)
    {
        Connection con = null;
        :
        try
        {
            con = DriverManager.getConnection(
                "jdbc:mysql://localhost:65534/受注管理DB",
                "user1", "pass1");
        }
        :
        catch(SQLException e)
        {
            System.out.println(
                "DBアクセス時にエラーが発生しました。");
            e.printStackTrace();
        }
    }
}
```

※DriverManagerクラスのgetConnection()メソッドは，引数に指定されたDBMS（MySQL）及びDB（受注管理DB）を探して，データソース情報を含んだインスタンスの参照を返します。

また，MySQLのユーザID（例：user1）とパスワード（例：pass1）を指定します。

④DBMSにSQL文を送信する準備を行う。

Connectionインタフェースのcreate Statement()メソッドを呼び出して，DBMSにSQL文を送信，DBMSから実行結果を受信するための準備を行います。

例.　:

```
class  SQLExecution
{
    public  static  void  main(String[]  args)
    {
        Connection con = null;
        Statement stmt = null;
        :
        try
        {
            :
            stmt = con.createStatement();
        }
        :
        catch(SQLException e)
        {
            System.out.println(
                "DBアクセス時にエラーが発生しました。");
            e.printStackTrace();
        }
    }
}
```

※Connectionインタフェースのcreate Statement()メソッドは，DBMSにSQL文を送信，DBMSから実行結果を受信するため情報を含んだインスタンスの参照を返します。

⑤DBMSにSQL文を送信する／DBMSから実行結果を受信する。

StatementインタフェースのexecuteQuery()メソッドを呼び出して，DBMSにSQL文を送信します。また，DBMSからSQL文の実行結果を受信します。

例.　：

```
class  SQLExecution
{
    public  static  void  main(String[]  args)
    {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
         :
        try
        {
            :
            rs = stmt.executeQuery(
                    "SELECT  商品ID, 商品名, 単価
                     FROM  商品マスタ;");
        }
         :
        catch(SQLException e)
        {
            System.out.println(
                "DBアクセス時にエラーが発生しました。");
            e.printStackTrace();
        }
    }
}
```

※StatementインタフェースのexecuteQuery()メソッドは，引数に指定されたSQL文をDBMSに送信して，実行結果を含んだインスタンスの参照を返します。なお，引数には，SQL文（SELECT文）を文字列として指定します。

※DBMSにINSERT文／UPDATE文／DELETE文を送信する場合は，StatementインタフェースのexecuteUpdate()メソッドを呼び出します。このメソッドは，引数に指定されたSQL文（INSERT文／UPDATE文／DELETE文）をDBMSに送信して，処理したレコード件数（int型）を返します。

⑥実行結果を含んだインスタンスからデータを取り出す。

ResultSetインタフェースのgetString()メソッドやgetInt()メソッドなどを呼び出して，実行結果を含んだインスタンスからデータを取り出します。

例.　:

```
class  SQLExecution
{
    public  static  void  main(String[]  args)
    {
        :
        try
        {
            :
            while(rs.next() == true)
            {
                String pId = rs.getString("商品ID");
                String pName = rs.getString("商品名");
                int pPrice = rs.getInt("単価");
            }
        }
        :
        catch(SQLException e)
        {
            System.out.println(
                "DBアクセス時にエラーが発生しました。");
            e.printStackTrace();
        }
    }
}
```

※ResultSetインタフェースのnext()メソッドは，実行結果（表）のどの行を参照しているかを示す“カーソル”を１つ下に移動します（初期状態では，先頭行の１つ上にカーソルがあります)。このメソッドは，true（カーソル位置に行がある）／false（カーソル位置には行がない）を返します。

※ResultSetインタフェースのgetString()メソッドは，引数に指定された列名のデータを含むStringインスタンスの参照を返します。

※ResultSetインタフェースのgetInt()メソッドは，引数に指定された列名のデータをint型の整数として返します。

⑦DBMSから切断する（DBMSとのコネクションを解放する）。

ResultSetインタフェースのclose()メソッド，Statementインタフェースのclose()メソッド，Connectionインタフェースのclose()メソッドを順に呼び出して，各接続を逆順に閉じます。

例.

```
class  SQLExecution
{
    public  static  void  main(String[]  args)
    {
        :
        catch(SQLException e)
        {
            System.out.println(
                "DBアクセス時にエラーが発生しました。");
        }
        finally
        {
            try
            {
                if(rs != null)
                {
                    rs.close();
                }
            }
            catch(SQLException e)
            {
                System.out.println(
                    "DBアクセス時にエラーが発生しました。");
                e.printStackTrace();
            }
```

```
                try
                {
                    if(stmt != null)
                    {
                        stmt.close();
                    }
                }
                catch(SQLException e)
                {
                    System.out.println(
                        "DBアクセス時にエラーが発生しました。");
                    e.printStackTrace();
                }
                try
                {
                    if(con != null)
                    {
                        con.close();
                    }
                }
                catch(SQLException e)
                {
                    System.out.println(
                        "DBアクセス時にエラーが発生しました。");
                    e.printStackTrace();
                }
            }
        }
}
```

※ResultSetインタフェースのclose()メソッドは，実行結果を含んだインスタンスに資源の解放を伝えます。

※Statementインタフェースのclose()メソッドは，DBMSから実行結果を受信するため情報を含んだインスタンスに資源の解放を伝えます。

※Connectionインタフェースのclose()メソッドは，データソース情報を含んだインスタンスに資源の解放を伝えます。

※各close()メソッドの呼出しは，finally{}の中に記述します。

実際に，JDBCを使用したプログラムを見てみましょう。プログラム1-1では，DB（受注管理DB）の商品マスタ表から全行を取得して表示しています。

プログラム 1-1　（ファイル名：SQLExecution.java）

```
1  import  java.sql.*;
2  class  SQLExecution
3  {
4     public  static  void  main(String[]  args)
5     {
6        Connection con = null;
7        Statement stmt = null;
8        ResultSet rs = null;
9        try
10       {
11          Class.forName("com.mysql.cj.jdbc.Driver");
12          con = DriverManager.getConnection(
13                "jdbc:mysql://localhost:65534/受注管理DB",
14                "user1", "pass1");
15          stmt = con.createStatement();
16          rs = stmt.executeQuery(
17                "SELECT  商品ID, 商品名, 単価  FROM  商品マスタ;");
18          System.out.println("商品ID¥t商品名¥t¥t¥t単価");
19          while(rs.next() == true)
20          {
21             String pId = rs.getString("商品ID");
22             String pName = rs.getString("商品名");
23             int pPrice = rs.getInt("単価");
24             System.out.println(
25                pId + "¥t" + pName + "¥t¥t¥¥" + pPrice);
26          }
27       }
28       catch(ClassNotFoundException e)
29       {
```

（次ページに続く）

```
30              System.out.println("JDBCドライバが見つかりません。");
31              e.printStackTrace();
32          }
33          catch(SQLException e)
34          {
35              System.out.println("DBアクセス時にエラーが発生しました。");
36              e.printStackTrace();
37          }
38          finally
39          {
40              try
41              {
42                  if(rs != null)
43                  {
44                      rs.close();
45                  }
46              }
47              catch(SQLException e)
48              {
49                  System.out.println(
50                      "DBアクセス時にエラーが発生しました。");
51                  e.printStackTrace();
52              }
53              try
54              {
55                  if(stmt != null)
56                  {
57                      stmt.close();
58                  }
59              }
60              catch(SQLException e)
61              {
62                  System.out.println(
63                      "DBアクセス時にエラーが発生しました。");
```

```
64                  e.printStackTrace();
65              }
66              try
67              {
68                  if(con != null)
69                  {
70                      con.close();
71                  }
72              }
73              catch(SQLException e)
74              {
75                  System.out.println(
76                      "DBアクセス時にエラーが発生しました。");
77                  e.printStackTrace();
78              }
79          }
80      }
81  }
```

実行結果　(表示のズレは問わない)

商品ID	商品名	単価
F-001	ストロベリー	¥1580
F-002	ピーチ	¥1480
F-003	マンゴ	¥1480
F-004	トロピカル	¥1300
F-005	オレンジ	¥1480
H-001	ローズヒップ	¥1800
H-002	カモミール	¥1600
H-003	レモンバーム	¥1700
H-004	ペパーミント	¥1600
H-005	ハイビスカスフラワー	¥1800
O-001	アイリッシュモルト	¥1400
T-001	ダージリン	¥2300
T-002	アッサム	¥1600
T-003	セイロン	¥1500
T-004	キャラメル	¥1400
T-005	アールグレイ	¥1500
T-006	バニラチャイ	¥1780
T-007	ドゥドロップス	¥1600

練習 1-1　レベル ☆

DB（受注管理DB）の商品マスタ表から，商品IDが'F-'で始まる行を取得して表示するプログラムを作成しなさい。（ファイル名：Ren1_1.java）

実行結果　（表示のズレは問わない）

商品ID	商品名	単価
F-001	ストロベリー	¥1580
F-002	ピーチ	¥1480
F-003	マンゴ	¥1480
F-004	トロピカル	¥1300
F-005	オレンジ	¥1480

練習 1-2　レベル ☆

DB（受注管理DB）の商品マスタ表から，単価が1,500～2,000円の行を取得して表示するプログラムを作成しなさい。なお，各行は単価の降順，商品IDの昇順に整列して表示すること。（ファイル名：Ren1_2.java）

実行結果　（表示のズレは問わない）

商品ID	商品名	単価
H-001	ローズヒップ	¥1800
H-005	ハイビスカスフラワー	¥1800
T-006	バニラチャイ	¥1780
H-003	レモンバーム	¥1700
H-002	カモミール	¥1600
H-004	ペパーミント	¥1600
T-002	アッサム	¥1600
T-007	ドゥドロップス	¥1600
F-001	ストロベリー	¥1580
T-003	セイロン	¥1500
T-005	アールグレイ	¥1500

練習 1-3　レベル ☆

DB（受注管理DB）の商品マスタ表に，新しい行｛'F-006'，'ピンクグレープフルーツ'，1400｝を挿入するプログラムを作成しなさい。なお，SQL文（INSERT文）の実行結果として，Statementインタフェースの executeUpdate()メソッドを呼び出したときの返却値（処理したレコード件数）をコンソールに出力すること。

ヒント　（ファイル名：Ren1_3.java）

```
 :
public  static  void  main(String[]  args)
{
    try
    {
        :
        stmt = con.createStatement();
        int result = stmt.executeUpdate("INSERT  ･･･
        System.out.println("挿入したレコード件数：" + result);
        :
    }
     :
```

実行結果　（商品IDが重複するレコードが存在しない場合）

```
挿入したレコード件数：1
```

※受注管理DBを開いて内容を確認すること。

実行結果　（商品IDが重複するレコードが存在する場合）

```
DBアクセス時にエラーが発生しました。
java.sql.SQLIntegrityConstraintViolationException: Duplicate entry 'F-006'
for key 'PRIMARY'
 :
```

練習 1-4　レベル ☆

DB（受注管理DB）の商品マスタ表の，商品IDが'F-006'の行の単価を1,470円に更新するプログラムを作成しなさい。なお，SQL文（UPDATE文）の実行結果として，Statementインタフェースの executeUpdate()メソッドを呼び出したときの返却値（処理したレコード件数）をコンソールに出力すること。

ヒント　（ファイル名：Ren1_4.java）

```
:
public  static  void  main(String[]  args)
{
      try
      {
            :
            stmt = con.createStatement();
            int result = stmt.executeUpdate("UPDATE  ･･･
            System.out.println("更新したレコード件数：" + result);
            :
      }
      :
```

実行結果　（商品IDが重複するレコードが存在する場合）

```
更新したレコード件数：1
```

※受注管理DBを開いて内容を確認すること。

実行結果　（商品IDが重複するレコードが存在しない場合）

```
更新したレコード件数：0
```

練習 1-5　レベル ☆

DB（受注管理）の商品マスタ表から，商品IDが'F-006'の行を削除するプログラムを作成しなさい。なお，SQL文（DELETE文）の実行結果として，Statementインタフェースの executeUpdate()メソッドを呼び出したときの返却値（処理したレコード件数）をコンソールに出力すること。

ヒント　（ファイル名：Ren1_5.java）

```
:
public  static  void  main(String[]  args)
{
      try
      {
            :
            stmt = con.createStatement();
            int result = stmt.executeUpdate("DELETE  ･･･
            System.out.println("削除したレコード件数：" + result);
            :
      }
      :
```

実行結果　（商品IDが重複するレコードが存在する場合）

```
削除したレコード件数：1
```

※受注管理DBを開いて内容を確認すること。

実行結果　（商品IDが重複するレコードが存在しない場合）

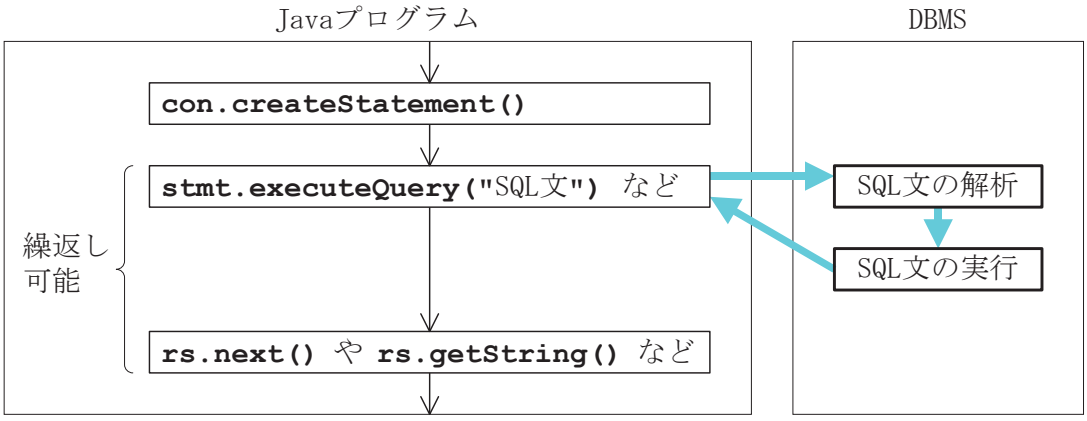
```
削除したレコード件数：0
```

1-3　DBアクセスの高速化

StatementインタフェースのexecuteQuery()メソッドとexecuteUpdate()メソッドは，引数に指定されたSQL文をDBMSに送信します。

これに伴い，DBMSでは，受け取ったSQL文を解析し，効率良く実行するための計画（表やインデックスへのアクセス方法，表の結合の順序など）を決定します。この作業をSQL文の“最適化（optimization)”，または“コンパイル（compile)”といいます。

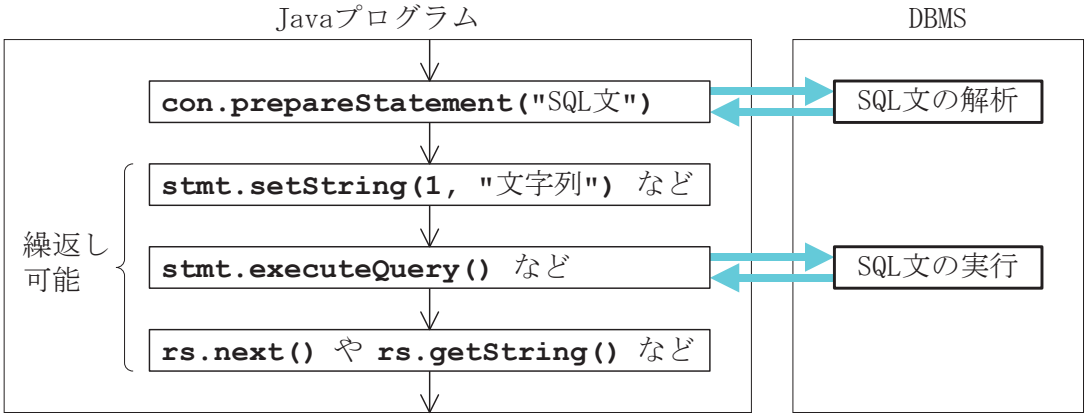
【Statementインタフェースを利用した場合】



DBMSは，受け取ったSQL文が過去に実行したものと同じであっても，新たに最適化を行います。また，最適化によって決定された実行計画は再利用できません。JDBCでは，これらを改善するために，PreparedStatementインタフェースを用意しています。

PreparedStatementインタフェースを利用したJavaプログラムでは，executeQuery()メソッドやexecuteUpdate()メソッドを呼び出す前に，SQL文の最適化を行います。

【PreparedStatementインタフェースを利用した場合】



setString()メソッドは，解析されたSQL文のWHERE句内のプレースホルダ“?（クエスチョン)”に，引数に指定された文字列を埋め込みます。また，executeQuery()メソッドは，解析されたSQL文（文字列がプレースホルダに埋め込まれたSQL文）を実行します。

prepareStatement()メソッドの呼出しによってDBMSが決定した実行計画は，同じSQL文を使用する限り，何度でも再利用できます。したがって，同じSQL文を繰返し実行するJavaプログラムの場合は，PreparedStatementインタフェースを利用すると最適化にかかる時間が短縮されるので，より高速なDBアクセスを実現することができます。

実際に，PreparedStatementインタフェースを使用したプログラムを見てみましょう。プログラム1-2では，DB（受注管理）の商品マスタ表から，商品コードが検索キーワードで始まる行を取得して表示しています。

プログラム 1-2　（ファイル名：SQLExecution2.java）

```
1  import  java.util.Scanner;
2  import  java.sql.*;
3  class  SQLExecution2
4  {
5     public  static  void  main(String[]  args)
6     {
7        Scanner sc = new Scanner(System.in);
8        Connection con = null;
9        PreparedStatement stmt = null;
10       ResultSet rs = null;
11       try
12       {
13          Class.forName("com.mysql.cj.jdbc.Driver");
14          con = DriverManager.getConnection(
15             "jdbc:mysql://localhost:65534/受注管理DB",
16             "user1", "pass1");
17          String sql
```

（次ページに続く）

```
18                  = "SELECT  商品ID, 商品名, 単価  FROM  商品マスタ
19                      WHERE  商品ID  LIKE  ?;";
20              //一度だけ最適化を行う
21              stmt = con.prepareStatement(sql);
22
23              String keyword;
24              System.out.print("商品IDの検索キーワード：");
25              keyword = sc.next();              //検索キーワード読込み
26              while(keyword.equals("end") != true)
27              {
28                  //SQL文に検索キーワードを埋め込む
29                  stmt.setString(1, keyword + "%");
30                  //SQL文を実行する
31                  rs = stmt.executeQuery();
32                  System.out.println("商品ID¥t商品名¥t¥t¥t単価");
33                  while(rs.next() == true)
34                  {
35                      String pId = rs.getString("商品ID");
36                      String pName = rs.getString("商品名");
37                      int pPrice = rs.getInt("単価");
38                      System.out.println(
39                          pId + "¥t" + pName + "¥t¥t¥¥" + pPrice);
40                  }
41                  rs.close();
42                  System.out.println();
43
44                  System.out.print("商品IDの検索キーワード：");
45                  keyword = sc.next();          //検索キーワード読込み
46              }
47          }
48          catch(ClassNotFoundException e)
49          {
50              System.out.println("JDBCドライバが見つかりません。");
51              e.printStackTrace();
```

```
52          }
53          catch(SQLException e)
54          {
55              System.out.println("DBアクセス時にエラーが発生しました。");
56              e.printStackTrace();
57          }
58          finally
59          {
60              try
61              {
62                  if(rs != null)
63                  {
64                      rs.close();
65                  }
66              }
67              catch(SQLException e)
68              {
69                  System.out.println(
70                      "DBアクセス時にエラーが発生しました。");
71                  e.printStackTrace();
72              }
73              try
74              {
75                  if(stmt != null)
76                  {
77                      stmt.close();
78                  }
79              }
80              catch(SQLException e)
81              {
82                  System.out.println(
83                      "DBアクセス時にエラーが発生しました。");
84                  e.printStackTrace();
```

（次ページに続く）

```
85              }
86              try
87              {
88                  if(con != null)
89                  {
90                      con.close();
91                  }
92              }
93              catch(SQLException e)
94              {
95                  System.out.println(
96                      "DBアクセス時にエラーが発生しました。");
97                  e.printStackTrace();
98              }
99          }
100         sc.close();
101     }
102 }
```

・22行目の「con.prepareStatement(sql)」は，引数に指定されたSQL文をDBMSに送信，DBMSから実行結果を受信するための情報を含んだインスタンスの参照を返します。なお，SQL文内の“?”については，後で文字列／整数値／実数値などに置換することができます。

・30行目の「stmt.setString(1, keyword + "%")」は，18～20行目の文字列（SQL文）内の１（setString()メソッドの第１引数の整数値）番目の“?”に，検索キーワード等（setString()メソッドの第２引数の文字列）を埋め込みます。なお，PreparedStatementインタフェースには，setString()メソッドの他に，setInt()メソッドやsetDouble()メソッドなども用意されています。

・32行目の「stmt.executeQuery()」は，既に最適化されているSQL文を実行して，その実行結果を受信します。

実行結果　（表示のズレは問わない）

```
商品IDの検索キーワード：F-  Enter
商品ID          商品名                  単価
F-001           ストロベリー            ¥1580
F-002           ピーチ                  ¥1480
F-003           マンゴ                  ¥1480
F-004           トロピカル              ¥1300
F-005           オレンジ                ¥1480

商品IDの検索キーワード：0-  Enter
商品ID          商品名                  単価
0-001           アイリッシュモルト      ¥1400

商品IDの検索キーワード：end  Enter
```

<<セキュリティ対策にもなるPreparedStatementインタフェース>>

PreparedStatementインタフェースは，DBアクセスの高速化と同時にDBMSのセキュリティ対策にも貢献します。

今回のSQL文のように，SQL文に検索条件などを後から指定する場合に開発者が意図しない不正な文字列（情報を引き出すSQL文やデータベースを改ざんするSQL文など）を指定してDBMSを不正に操作されてしまう可能性があります。これをSQLインジェクション攻撃と呼び，主にWebアプリケーションでフォームから文字の入力ができるシステムに対して行われる攻撃です。

PreparedStatementインタフェースを使用すると，万が一不正なSQL文を入力されSQLインジェクション攻撃を試みられても，プレースホルダに値が埋め込まれる際に，JDBCドライバがシングルクォートなどを自動的にエスケープシーケンスに変換し，単なる文字列としか認識されなくなり，SQL文として解釈されずSQLインジェクション攻撃を防げます。

練習 1-6　レベル ☆☆

練習1-2で作成したプログラムにおいて，PreparedStatementインタフェースを使用して，キーボードから入力した２つの整数（単価）をそれぞれ下限，上限としてSQL文に埋め込んで実行するように変更しなさい。

ヒント　（ファイル名：Ren1_6.java）

```
import java.util.Scanner;
:
class  Ren1_6
{
   public  static  void  main(String[]  args)
   {
       Scanner sc = new Scanner(System.in);
       Connection con = null;
       PreparedStatement stmt = null;
       ResultSet rs = null;
       try
       {
           :
           String sql = "SELECT  ･･･
           stmt = con.prepareStatement(sql);

           int lowerPrice;
           int upperPrice;
           System.out.print("単価の下限：");
           lowerPrice = sc.nextInt();       //単価の下限を読み込む
           System.out.print("単価の上限：");
           upperPrice = sc.nextInt();       //単価の上限を読み込む
           while(lowerPrice > 0 && upperPrice > 0)
           {
               stmt.setInt(  ･･･
               stmt.setInt(  ･･･
```

```
            rs =  ･･･
             :
            rs.close();
            System.out.println();

            System.out.print("単価の下限：");
            lowerPrice = sc.nextInt();
            System.out.print("単価の上限：");
            upperPrice = sc.nextInt();
        }
    }
     :
```

実行結果　（表示のズレは問わない）

```
単価の下限：1500  Enter
単価の上限：2000  Enter
商品ID          商品名                  単価
H-001           ローズヒップ            ¥1800
H-005           ハイビスカスフラワー    ¥1800
T-006           バニラチャイ            ¥1780
H-003           レモンバーム            ¥1700
H-002           カモミール              ¥1600
H-004           ペパーミント            ¥1600
T-002           アッサム                ¥1600
T-007           ドゥドロップス          ¥1600
F-001           ストロベリー            ¥1580
T-003           セイロン                ¥1500
T-005           アールグレイ            ¥1500

単価の下限：0  Enter
単価の上限：0  Enter
```

練習 1-7　レベル ☆☆

PreparedStatementインタフェースを使用して，DB（受注管理DB）の受注表と顧客マスタ表を結合した表から，受注日がキーボードから入力した整数（西暦年）の行を取得して表示するプログラムを作成しなさい。なお，コンソールには受注日，顧客ID，顧客名，フリガナのデータを出力し，各行は受注日の降順に出力すること。（ファイル名：Ren1_7.java）

実行結果　（表示のズレは問わない）

```
受注日の西暦年：2005  Enter
受注日          顧客ID      顧客名          フリガナ
2005/12/22      5           小島　千里      コジマ　チサト
2005/12/01      3           西方　樹里      ニシカタ　ジュリ
2005/11/26      1           鈴木　隆文      スズキ　タカフミ

受注日の西暦年：0  Enter
```

練習 1-8　レベル ☆☆

PreparedStatementインタフェースを使用して，DB（受注管理DB）の受注明細表から，商品IDごとの数量合計がキーボードから入力した整数（基準）以上の行を取得して表示するプログラムを作成しなさい。なお，コンソールには商品ID，数量合計のデータを出力し，各行は数量合計の降順，商品IDの昇順に出力すること。（ファイル名：Ren1_8.java）

実行結果　（表示のズレは問わない）

```
数量合計の基準：10  Enter
商品ID          数量合計
F-001           83
T-005           60
T-001           48
T-006           47
T-002           36
F-003           29
H-001           23
T-003           17
O-001           12
F-005           11
H-002           10

数量合計の基準：0  Enter
```

1-4　デザインパターンの導入

デザインパターンとは，ソフトウェア設計時に起こる典型的な問題と，それに対する解決策をまとめたものです。現在では， GoFデザインパターン（23種類）やJ2EEパターン（21種類）など，さまざまなデザインパターンが公開されており，ソフトウェア設計時にこれらのデザインパターンを採用することで，開発者の経験不足を補うことができます。

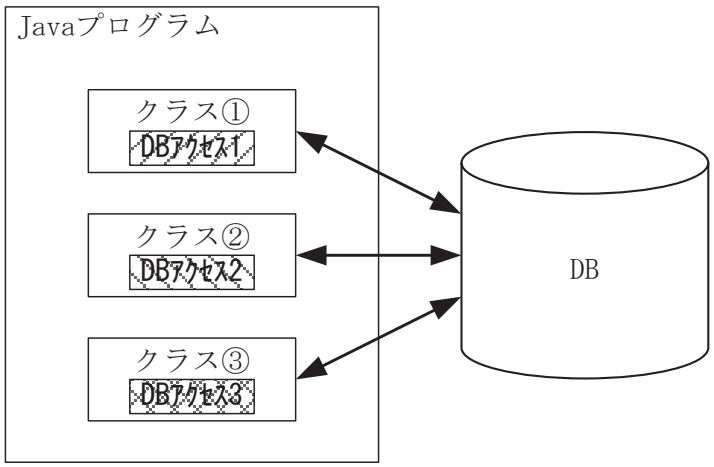
J2EEパターンは，J2EE（企業の業務システムや電子商取引などで使われるサーバに必要な機能をまとめたもの）を使用したソフトウェアの設計時に利用されるデザインパターンで，Java言語を発表したSun Microsystems社（現Oracle社）が提唱しています。

本教材では，DBアクセスに関連するJ2EEパターンのうち，「DAO」と「DTO」を導入したJavaプログラムの作成方法を説明します。

1-4-1　DAO

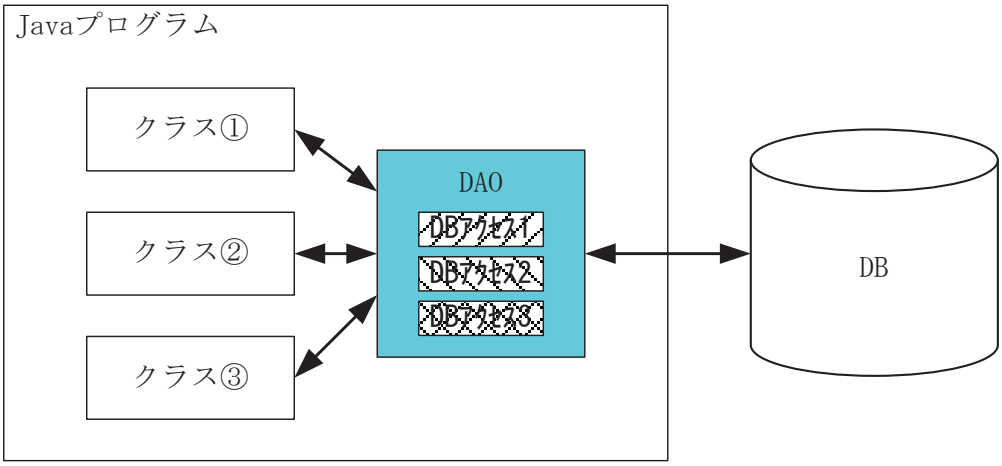
DAO（Data Access Object）とは，Javaプログラム（多数のクラス）に点在するDBアクセス処理を集めたオブジェクトのことであり，DAOを導入したJavaプログラムは，ソフトウェアの保守性が向上します。

［DAOを導入しないJavaプログラム］



図「DAOを導入しないJavaプログラム」では，DBMS変更（例えば，MySQLからOracleへの変更など）時に，クラス①～③のプログラムをそれぞれ修正しなければいけません。

［DAOを導入したJavaプログラム］

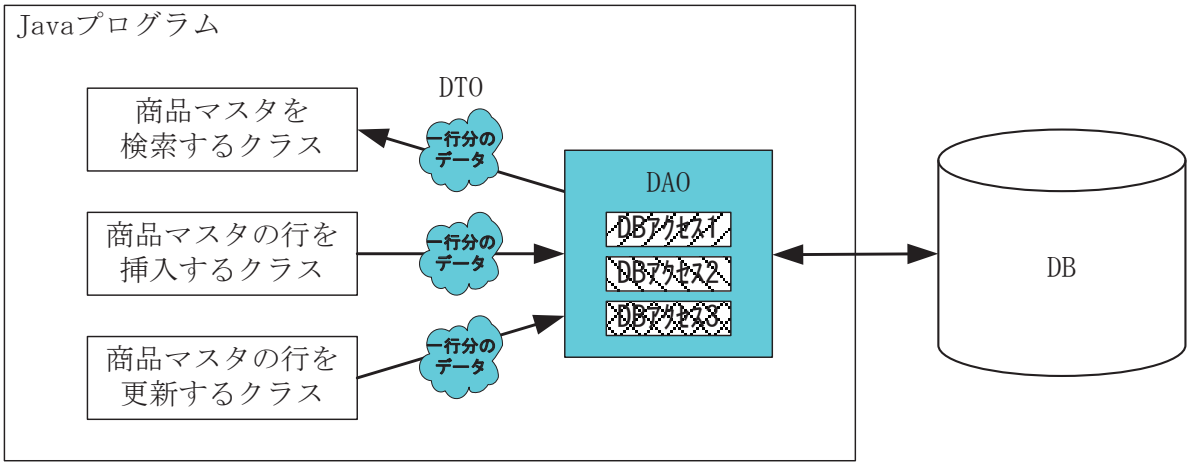


一方，図「DAOを導入したJavaプログラム」では，DBMS変更時にDAOクラスのプログラムだけ修正する（①～③は一切修正しない）ことになります。

1-4-2　DTO

DTO（Data Transfer Object）とは，DBを利用するクラス（図「DAOを導入したJavaプログラム」内のクラス①～③）-DAOクラス間でやり取りされる，DBの一行分のデータをもったオブジェクトのことであり，DBMSにSQL文（SELECT文）を送信して実行した結果（表）をDTO（一行分のデータ）の集まりとして表現するときに使用します。そのため，DTOクラスは，なるべくJavaBeansに準拠して定義します。

［DTOを導入したJavaプログラム］



1-4-3　DAOとDTOを導入したJavaプログラム

実際に，DAOとDTOを導入したプログラムを見てみましょう。プログラム1-3～1-5では，DB（受注管理DB）の商品マスタ表から全行を取得して表示しています。ここでは，SELECT文で指定した列（35ページのプログラムの62行目）を参考にしてDTOクラスを設計していることに注意してください。

プログラム 1-3　（ファイル名：ProductBean.java）

```
1  import  java.io.Serializable;
2
3  //ProductBeanクラス（DTO）の定義
4  public  class  ProductBean  implements  Serializable
5  {
6      private  String  id;                    //商品ID
7      private  String  name;                  //商品名
8      private  int  price;                    //単価
9
10     public  ProductBean(String  id, String  name, int  price)
11     {
12         setId(id);
13         setName(name);
14         setPrice(price);
15     }
16
17     public  void  setId(String  id)         //商品IDを設定する
18     {
19         this.id = id;
20     }
21
22     public  String  getId()                 //商品IDを取得する
23     {
24         return id;
25     }
26
```

```
27      public  void  setName(String  name)      //商品名を設定する
28      {
29          this.name = name;
30      }
31
32      public  String  getName()                //商品名を取得する
33      {
34          return name;
35      }
36
37      public  void  setPrice(int  price)       //単価を設定する
38      {
39          this.price = price;
40      }
41
42      public  int  getPrice()                  //単価を取得する
43      {
44          return price;
45      }
46  }
```

プログラム 1-4 （ファイル名：OrderControlDBAccess.java）

```
1  import  java.util.ArrayList;
2  import  java.sql.*;
3
4  //OrderControlDBAccessクラス（DAO）の定義
5  public  class  OrderControlDBAccess
6  {
7      //DBとの接続を確立する
8      private  Connection  createConnection()
9      {
10         Connection con = null;
11         try
12         {
13             Class.forName("com.mysql.cj.jdbc.Driver");
14             con = DriverManager.getConnection(
15                 "jdbc:mysql://localhost:65534/受注管理DB",
16                 "user1", "pass1");
17         }
18         catch(ClassNotFoundException e)
19         {
20             System.out.println("JDBCドライバが見つかりません。");
21             e.printStackTrace();
22         }
23         catch(SQLException e)
24         {
25             System.out.println("DB接続時にエラーが発生しました。");
26             e.printStackTrace();
27         }
28         return con;
29     }
30
31     //DBとの接続を閉じる
32     private  void  closeConnection(Connection  con)
33     {
```

```
34          try
35          {
36              if(con != null)
37              {
38                  con.close();
39              }
40          }
41          catch(SQLException e)
42          {
43              System.out.println("DB切断時にエラーが発生しました。");
44              e.printStackTrace();
45          }
46      }
47
48      //商品リストを取得する
49      public  ArrayList<ProductBean>  findAllProducts()
50      {
51          Connection con = createConnection();
52          PreparedStatement stmt = null;
53          ResultSet rs = null;
54          ArrayList<ProductBean> list
55              = new ArrayList<ProductBean>();
56          try
57          {
58              if(con != null)
59              {
60                  String sql =
61                          "SELECT  商品ID, 商品名, 単価  FROM  商品マスタ;";
62                  //一度だけ最適化を行う
63                  stmt = con.prepareStatement(sql);
64                  rs = stmt.executeQuery();
65                  while(rs.next() == true)
66                  {
```

（次ページに続く）

```
67                  String pId = rs.getString("商品ID");
68                  String pName = rs.getString("商品名");
69                  int pPrice = rs.getInt("単価");
70                  ProductBean bean
71                      = new ProductBean(pId, pName, pPrice);
72                  list.add(bean);
73              }
74          }
75      }
76      catch(SQLException e)
77      {
78          System.out.println(
79              "DBアクセス時にエラーが発生しました（商品検索）。");
80          e.printStackTrace();
81      }
82      finally
83      {
84          try
85          {
86              if(rs != null)
87              {
88                  rs.close();
89              }
90          }
91          catch(SQLException e)
92          {
93              System.out.println(
94                  "DBアクセス時にエラーが発生しました。");
95              e.printStackTrace();
96          }
97          try
98          {
99              if(stmt != null)
100             {
```

```
101                     stmt.close();
102                 }
103             }
104             catch(SQLException e)
105             {
106                 System.out.println(
107                     "DBアクセス時にエラーが発生しました。");
108                 e.printStackTrace();
109             }
110         }
111         closeConnection(con);
112         return list;
113     }
114 }
```

プログラム 1-5 （ファイル名：SQLExecution3.java）

```
1  import  java.util.ArrayList;
2  class  SQLExecution3
3  {
4      public  static  void  main(String[]  args)
5      {
6          //DAOを生成する
7          OrderControlDBAccess dao = new OrderControlDBAccess();
8          //DAO から商品リストを取得する
9          ArrayList<ProductBean> list = dao.findAllProducts();
10         System.out.println("商品ID¥t商品名¥t¥t¥t単価");
11         for(ProductBean bean : list)
12         {
13             System.out.println(bean.getId() + "¥t"
14                 + bean.getName() + "¥t¥t¥¥" + bean.getPrice());
15         }
16     }
17 }
```

実行結果　(表示のズレは問わない)

商品ID	商品名	単価
F-001	ストロベリー	¥1580
F-002	ピーチ	¥1480
F-003	マンゴ	¥1480
F-004	トロピカル	¥1300
F-005	オレンジ	¥1480
H-001	ローズヒップ	¥1800
H-002	カモミール	¥1600
H-003	レモンバーム	¥1700
H-004	ペパーミント	¥1600
H-005	ハイビスカスフラワー	¥1800
O-001	アイリッシュモルト	¥1400
T-001	ダージリン	¥2300
T-002	アッサム	¥1600
T-003	セイロン	¥1500
T-004	キャラメル	¥1400
T-005	アールグレイ	¥1500
T-006	バニラチャイ	¥1780
T-007	ドゥドロップス	¥1600

練習 1-9　レベル ☆☆☆

プログラム1-4（ファイル名：OrderControlDBAccess.java）に，DB（受注管理DB）の顧客マスタ表から全行を取得して表示する機能を追加しなさい。なお，コンソールには顧客ID，顧客名のデータを出力すること。

ヒント　（ファイル名：CustomerBean.java）

```
import  java.io.Serializable;

//CustomerBeanクラス（DTO）の定義
public  class  CustomerBean  implements  Serializable
{
    :
}
```

ヒント　（ファイル名：OrderControlDBAccess.java）

```
    :
    //顧客リストを取得する
    public  ArrayList<CustomerBean>  findAllCustomers()
    {
        Connection con = createConnection();
        PreparedStatement stmt = null;
        ResultSet rs = null;
        ArrayList<CustomerBean> list
            = new ArrayList<CustomerBean>();
        try
        {
            :
        }
        catch(SQLException e)
        {
            System.out.println(
                "DBアクセス時にエラーが発生しました（顧客検索）。");
```

```
            e.printStackTrace();
        }
        :
        closeConnection(con);
        return list;
    }
}
```

プログラム　（ファイル名：SQLExecution4.java）

```
1  import  java.util.ArrayList;
2  class  SQLExecution4
3  {
4      public  static  void  main(String[]  args)
5      {
6          //DAOを生成する
7          OrderControlDBAccess dao = new OrderControlDBAccess();
8          //DAO から顧客リストを取得する
9          ArrayList<CustomerBean> list = dao.findAllCustomers();
10         System.out.println("顧客ID¥t顧客名");
11         for(CustomerBean bean : list)
12         {
13             System.out.println(
14                 bean.getId() + "¥t" + bean.getName());
15         }
16     }
17 }
```

実行結果　(表示のズレは問わない)

顧客ID	顧客名
1	鈴木　隆文
2	金子　実
3	西方　樹里
4	橘　正潤
5	小島　千里
6	五十嵐　鈴与
7	太田　誠人
8	横田　進
9	市山　桃子
10	佐藤　美佑
11	中村　大輔
12	宮　正澄
13	谷口　柚香
14	小林　和佳子
15	武山　勝
16	杉田　裕太
17	宮崎　沙耶子
18	荒瀬　陽子
19	安藤　由紀子
20	金子　紀子

練習 1-10　レベル ☆☆☆

プログラム1-4（ファイル名：OrderControlDBAccess.java）に，DB（受注管理DB）の顧客マスタ表，商品マスタ表，受注表，受注明細表を結合した表から，都道府県ごとの売上合計を取得して表示する機能を追加しなさい。なお，コンソールには都道府県名，売上合計のデータを出力し，各行は売上合計の降順にすること。

ヒント　（ファイル名：DivisionBean.java）

```
import  java.io.Serializable;

//DivisionBeanクラス（DTO）の定義
public  class  DivisionBean  implements  Serializable
{
    :
}
```

ヒント　（ファイル名：OrderControlDBAccess.java）

```
    :
    //都道府県リストを取得する
    public  ArrayList<DivisionBean>  findAllDivisions()
    {
        Connection con = createConnection();
        PreparedStatement stmt = null;
        ResultSet rs = null;
        ArrayList<DivisionBean> list
            = new ArrayList<DivisionBean>();
        try
        {
            :
        }
        catch(SQLException e)
        {
```

```
            System.out.println(
                "DBアクセス時にエラーが発生しました（都道府県検索）。");
            e.printStackTrace();
        }
        :
        closeConnection(con);
        return list;
    }
}
```

プログラム　（ファイル名：SQLExecution5.java）

```
1  import  java.util.ArrayList;
2  class  SQLExecution5
3  {
4      public  static  void  main(String[]  args)
5      {
6          //DAOを生成する
7          OrderControlDBAccess dao = new OrderControlDBAccess();
8          //DAO から都道府県リストを取得する
9          ArrayList<DivisionBean> list = dao.findAllDivisions();
10         System.out.println("都道府県名¥t売上合計");
11         for(DivisionBean bean : list)
12         {
13             System.out.println(
14                 bean.getName() + "¥t¥¥" + bean.getPrice());
15         }
16     }
17 }
```

実行結果　(表示のズレは問わない)

都道府県名	売上合計
東京都	¥221140
千葉県	¥163860
福島県	¥79440
神奈川県	¥51360
静岡県	¥44160
石川県	¥39600
福岡県	¥27700
長野県	¥20500
埼玉県	¥10300
大阪府	¥5080
宮城県	¥3580
愛知県	¥1480

付録

受注管理DBのデータ

付録　受注管理DBのデータ

“顧客マスタ”表

顧客ID	顧客名	フリガナ	郵便番号	都道府県
1	鈴木　隆文	スズキ　タカフミ	247-0022	神奈川県
2	金子　実	カネコ　ミノル	108-0071	東京都
3	西方　樹里	ニシカタ　ジュリ	270-0013	千葉県
4	橘　正潤	タチバナ　セイジュン	390-0303	長野県
5	小島　千里	コジマ　チサト	153-0042	東京都
6	五十嵐　鈴与	イガラシ　スズヨ	810-0001	福岡県
7	太田　誠人	オオタ　マコト	206-0021	東京都
8	横田　進	ヨコタ　ススム	970-8003	福島県
9	市山　桃子	イチヤマ　モモコ	414-0045	静岡県
10	佐藤　美佑	サトウ　ミウ	156-0057	東京都
11	中村　大輔	ナカムラ　ダイスケ	330-0052	埼玉県
12	宮　正澄	ミヤ　マサズミ	135-0043	東京都
13	谷口　柚香	タニグチ　ユカ	981-0961	宮城県
14	小林　和佳子	コバヤシ　ワカコ	169-0073	東京都
15	武山　勝	ムヤマ　マサル	464-0092	愛知県
16	杉田　裕太	スギタ　ユウタ	545-0037	大阪府
17	宮崎　沙耶子	ミヤザキ　サヤコ	270-0034	千葉県
18	荒瀬　陽子	アラセ　ヨウコ	920-0961	石川県
19	安藤　由紀子	アンドウ　ユキコ	111-0035	東京都
20	金子　紀子	カネコ　ノリコ	243-0035	神奈川県

住所	電話番号
横浜市栄区庄戸 0-24-70	045-888-0000
港区白金台 0-100	03-7777-0000
松戸市小金きよしケ丘 0-5　ビューハイツ 700	045-888-0000
松本市浅間温泉 0-500	0263-11-0000
目黒区青葉台 0-30	03-6666-0000
福岡市中央区天神 0-500 BP ハイツ 100	092-000-0000
多摩市連光寺 0-16-3　エクセレントタワー1100	042-333-0000
いわき市平下平窪 0-300	046-22-0000
伊東市玖須美元和田 026-100	0557-33-0000
世田谷区上北沢 0-200	03-3333-0000
さいたま市浦和区本太 0-30　レーベンハイム 700	048-888-0000
江東区塩浜 0-4-000	03-5555-0000
仙台市青葉区桜ケ丘 0-400	022-555-0000
新宿区百人町 0-100	03-3333-0000
名古屋市千種区茶屋が坂 0-50　サンビレッジ 300	052-777-0000
大阪市阿倍野区帝塚山 0-300	06-8888-0000
松戸市新松戸 0-500　プラスステージ 100	047-333-0000
金沢市香林坊 0-8-200	076-333-0000
台東区西浅草 0-12-000	03-3333-0000
厚木市愛甲 036-10	045-333-0000

“商品マスタ”表

商品ID	商品名	単価
F-001	ストロベリー	1580
F-002	ピーチ	1480
F-003	マンゴ	1480
F-004	トロピカル	1300
F-005	オレンジ	1480
H-001	ローズヒップ	1800
H-002	カモミール	1600
H-003	レモンバーム	1700
H-004	ペパーミント	1600
H-005	ハイビスカスフラワー	1800
O-001	アイリッシュモルト	1400
T-001	ダージリン	2300
T-002	アッサム	1600
T-003	セイロン	1500
T-004	キャラメル	1400
T-005	アールグレイ	1500
T-006	バニラチャイ	1780
T-007	ドゥドロップス	1600

“受注”表

受注ID	受注日	顧客ID
1	2005/11/26	1
2	2005/12/01	3
3	2005/12/22	5
4	2006/01/11	7
5	2006/01/18	2
6	2006/01/26	9
7	2006/02/01	10
8	2006/02/10	14
9	2006/02/22	1
10	2006/03/06	6
11	2006/03/16	18
12	2006/04/11	19
13	2006/04/24	11
14	2006/04/25	20
15	2006/05/02	9
16	2006/05/17	12
17	2006/05/29	15
18	2006/06/05	7
19	2006/06/13	16
20	2006/06/28	13
21	2006/06/28	8
22	2006/07/04	4
23	2006/07/11	10
24	2006/07/13	11
25	2006/07/25	9

受注ID	受注日	顧客ID
26	2006/07/28	17
27	2006/08/03	20
28	2006/08/10	18
29	2006/08/11	10
30	2006/08/22	17
31	2006/08/30	12
32	2006/09/07	4
33	2006/09/22	8
34	2006/10/04	19
35	2006/10/18	1
36	2006/10/24	6
37	2006/10/26	9
38	2006/10/27	17

“受注明細”表

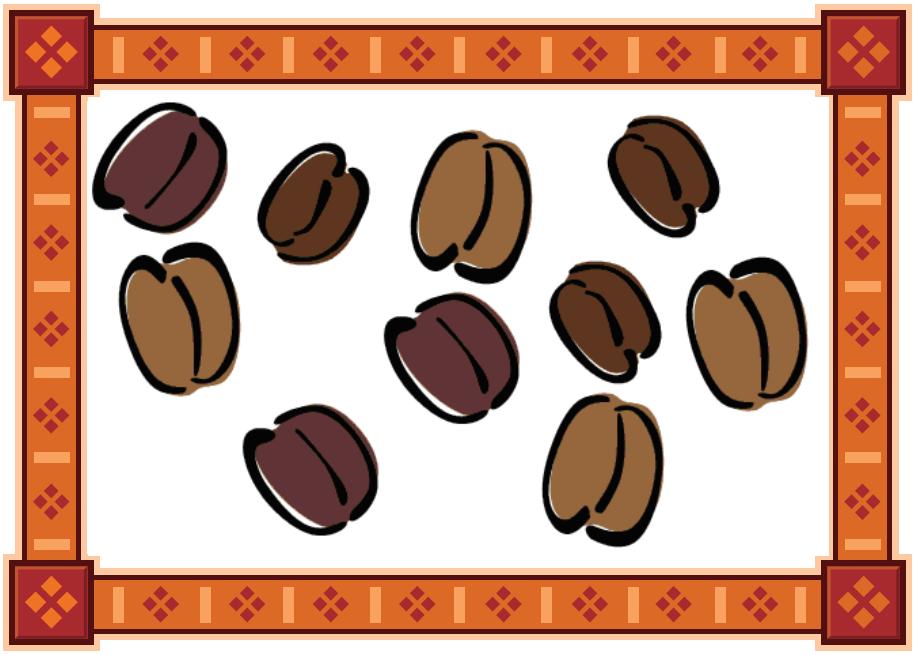
明細ID	受注ID	商品ID	数量
1	1	T-001	5
2	1	T-005	5
3	1	F-001	2
4	2	T-002	10
5	2	T-003	10
6	2	T-006	20
7	2	O-001	10
8	2	F-001	10
9	3	T-001	3
10	4	O-001	2
11	4	F-005	2
12	4	H-001	2
13	5	T-007	2
14	5	F-003	2
15	6	T-003	1
16	6	T-007	1
17	7	T-006	1
18	8	F-001	10
19	8	H-004	1
20	8	T-001	6
21	9	H-004	1
22	9	T-005	1
23	10	F-003	3
24	10	H-002	1
25	11	H-004	1

明細ID	受注ID	商品ID	数量
26	12	F-001	3
27	13	F-004	1
28	14	F-003	1
29	14	T-004	1
30	14	T-005	5
31	15	H-004	1
32	16	T-001	1
33	16	T-004	1
34	17	F-003	1
35	18	F-001	10
36	18	T-002	20
37	18	T-005	20
38	19	T-006	1
39	19	T-007	1
40	19	H-003	1
41	20	T-006	1
42	20	H-001	1
43	21	T-001	1
44	21	T-005	4
45	21	F-005	3
46	22	T-003	1
47	22	H-001	2
48	23	F-001	3
49	23	T-001	3
50	23	T-002	5

明細ID	受注ID	商品ID	数量
51	23	T-003	3
52	23	T-005	2
53	23	H-002	2
54	24	H-001	5
55	25	T-003	2
56	25	T-006	2
57	26	H-001	3
58	26	H-002	5
59	27	T-006	8
60	28	F-001	10
61	28	F-003	10
62	28	F-005	5
63	29	T-004	6
64	30	T-002	1
65	30	T-005	1
66	30	T-006	2
67	31	T-005	2
68	31	T-001	12
69	32	T-001	2
70	32	H-001	6
71	33	T-001	10
72	33	T-006	10
73	33	F-001	5
74	33	F-003	10
75	33	H-002	1

明細ID	受注ID	商品ID	数量
76	33	H-004	1
77	34	F-003	2
78	34	H-001	4
79	35	F-005	1
80	36	T-001	5
81	36	T-006	2
82	36	H-003	2
83	36	H-004	2
84	37	T-005	20
85	37	F-004	1
86	37	H-002	1
87	38	F-001	30

サーブレット／JSP



１．サーブレット／JSP

１－１　Webアプリケーションの仕組み

１－２　サーブレット基礎

１－３　JSP基礎

１－４　MVCアーキテクチャ

1-1　Webアプリケーションの仕組み

JavaServlet（以下，Javaサーブレット）とは，Webコンテナ（Javaサーブレットを実行できるWebサーバ）上で動作するJavaプログラムのことで，主に動的Webページ（Webブラウザからの閲覧要求ごとに作成するWebページ）の配信に利用されます。

Javaサーブレットでは，Javaプログラム内にHTML（HyperText Markup Language）のタグを埋め込みます。

プログラム　（ファイル名：HelloWorld.java）

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

@WebServlet("/HelloWorld")
public  class  HelloWorld  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse  response)
        throws  ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("ようこそJSP／サーブレットの世界へ！");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```

図１　Javaサーブレットのプログラム（HelloWorld.java）

Javaサーブレット（Javaソースコード）は，JavaコンパイラによってJavaバイトコードに変換します。なお，Webブラウザからの閲覧要求時には，Javaサーブレット（Javaバイトコード）はWebコンテナによって実行されます。

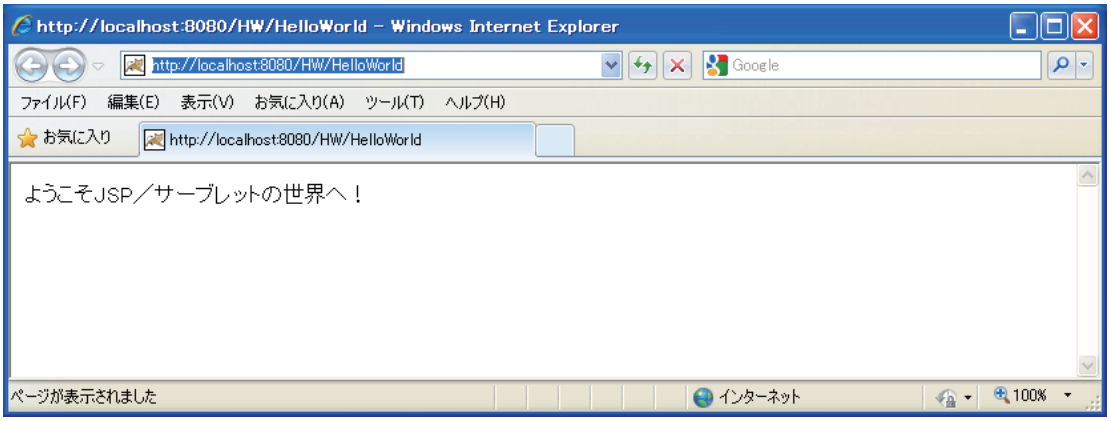


図２　Javaサーブレットの実行結果

また，Javaサーブレットと同様のテクノロジとして，JSP（Java Server Pages）があります。JSPでは，HTML文書内にJavaソースコードを埋め込みます。

プログラム　（ファイル名：HelloWorld.jsp）

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html>
<body>

<%
    out.print("ようこそJSP／サーブレットの世界へ！");
%>

</body>
</html>
```

図３　JSPのプログラム（HelloWorld.jsp）

JSP（HTML文書＋Javaソースコード）は，「～.jsp」というファイル名で保存します（この時点でのJavaコンパイラによる変換は不要です）。また，JSPは，Webコンテナ起動時にJavaサーブレット（Javaソースコード）に自動変換され，JavaコンパイラによってJavaバイトコードに自動変換されます。なお，Webブラウザからの閲覧要求時には，JSP（Javaバイトコード）はWebコンテナによって実行されます。

[Eclipseを使用して開発する場合の注意]

Eclipseを使用して開発する場合，次の点に注意してください。

－ importされるクラスの表示が異なる

Eclipseで新規Javaサーブレットを作成するとimport宣言が自動で挿入されます。

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

２ページのプログラムでは「*」で一括してimportしていたクラスを個別でimportしています。

また，Eclipseのバージョンによっては「*」で一括してimportを記述しても，この例のように個別のimportに自動的に変換される場合もありますが，Javaサーブレット実行には影響ありません。

1-2　サーブレット基礎

1-2-1　JavaServletの仕組み

Webブラウザ-Webコンテナ間の情報のやり取りは，“HTTP（HyperText Transfer Protocol）”というプロトコルに準拠します。特に，WebブラウザからWebコンテナに送信される情報を“リクエスト”，WebコンテナからWebブラウザに送信される情報を“レスポンス”といいます。リクエストにはWebブラウザ情報やフォームに入力されたデータなどが含まれ，レスポンスにはHTMLで記述されたWebページ情報などが含まれます。

また，Webコンテナは，Webブラウザからリクエストを受け取ると，Javaサーブレット内のdoGet()メソッドまたはdoPost()メソッドを呼び出します。これらのメソッドは，呼出し元から引数としてリクエスト（HttpServletRequestインタフェース型）を受け取り，自身の処理部分でデータ処理やDBアクセス処理などを行い，呼出し元に引数としてレスポンス（HttpServletResponseインタフェース型）を返します。

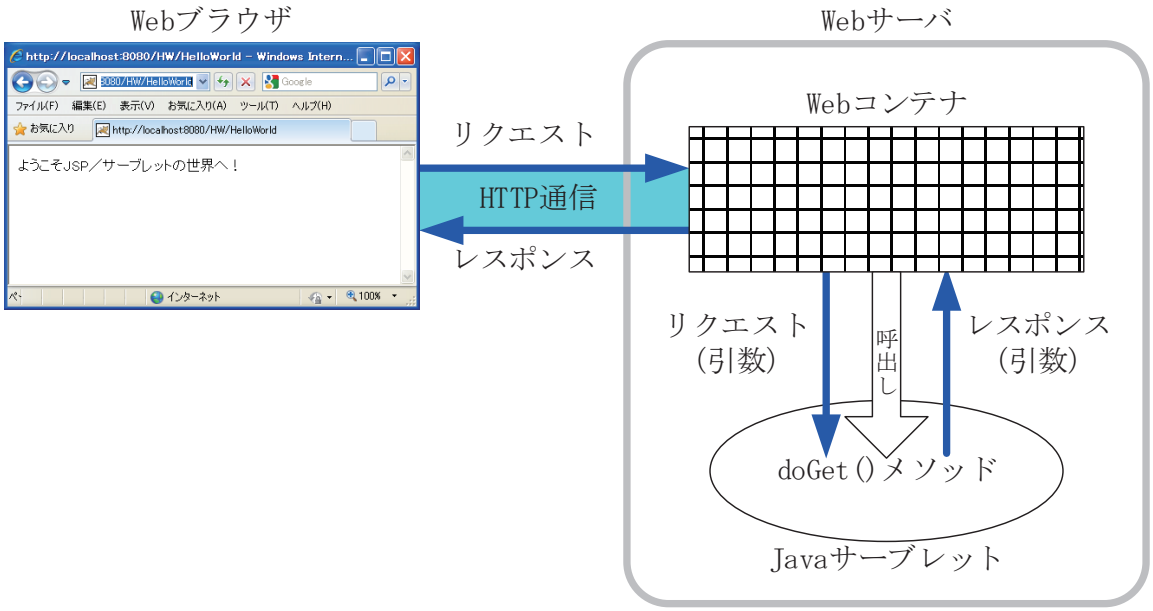


図4　リクエストとレスポンス

Javaサーブレットは，javax.servlet.httpパッケージに登録されているHttpServletクラスのサブクラスとして定義します。HttpServletクラスには，Javaサーブレットの初期化や入出力処理などの基本機能や，処理部分が空のdoGet()メソッドなどが定義されています。したがって，Javaサーブレットでは，HttpServletクラスから継承したdoGet()メソッドを

オーバライドして，その処理部分に独自処理を記述します。

Javaサーブレットの主な構造は，次のとおりです。

```
import  javax.servlet.*;                                          ①
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;

@WebServlet("/HelloWorld")
public  class  HelloWorld  extends  HttpServlet                   ②
{
    protected  void  doGet(                                       ③
        HttpServletRequest  request,
        HttpServletResponse  response)
        throws  ServletException, IOException
    {
        response.setContentType(
            "text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<body>");
        out.println(
            "ようこそJSP／サーブレットの世界へ！");
        out.println("</body>");
        out.println("</html>");

        out.close();
    }
}
```

①Javaサーブレットのクラス内で使用するパッケージをインポートする。

・javax.servletパッケージ

ServletExceptionクラスなどが登録されています。

・javax.servlet. annotationパッケージ

WebServletクラスなどが登録されています。

・javax.servlet.httpパッケージ

HttpServletクラス，HttpServletRequestインタフェース，HttpServletResponseインタフェース，HttpSessionインタフェースなどが登録されています。

・java.ioパッケージ

IOExceptionクラス，PrintWriterクラスなどが登録されています。

②Javaサーブレットのクラスを定義する。

HttpServletクラスのサブクラスとして定義します。HttpServletクラスには抽象メソッドは含まれていないので，③は任意です。

③doGet()メソッドやdoPost()メソッドを定義する。

Javaサーブレットのクラス内では，HttpServletクラスから継承したdoGet()メソッドやdoPost()メソッドに対してオーバライドを行い，独自処理を定義します。各メソッドの仕様は，次のとおりです。

表１　doGet()メソッドの仕様

項目	内容
メソッド名	doGet()
引数	［第１引数］request（HttpServletRequest型，リクエスト） ［第２引数］response（HttpServletResponse型，レスポンス）
返却値	なし（void型）
修飾子	protected
例外	ServletException，IOException
概要	HTTPの“GETメソッド”に対応する。

※GETメソッドによるリクエストの送信の場合，WebブラウザからWebコンテナへの送信時は，フォームに入力されたデータがWebブラウザのURL欄に表示されます。

表２　doPost()メソッドの仕様

項目	内容
メソッド名	doPost()
引数	［第１引数］request（HttpServletRequest型，リクエスト） ［第２引数］response（HttpServletResponse型，レスポンス）
返却値	なし（void型）
修飾子	protected
例外	ServletException，IOException
概要	HTTPの“POSTメソッド”に対応する。

※POSTメソッドによるリクエストの送信の場合，WebブラウザからWebコンテナへの送信時は，フォームに入力されたデータをWebブラウザのURL欄に表示しません。したがって，パスワードや個人情報を送信するときには，POSTメソッドによるリクエストの送信を行わなければいけません。

実際に，Javaサーブレットのプログラムを見てみましょう。プログラム1-1では，Javaサーブレットは，Webブラウザからのリクエストに対するレスポンスとして，Webブラウザにログインページを出力しています。

プログラム 1-1　（ファイル名：LoginPageServlet.java　URL：/LoginPage）

```
1  import  javax.servlet.*;
2  import  javax.servlet.annotation.*;
3  import  javax.servlet.http.*;
4  import  java.io.*;
5
6  @WebServlet("/LoginPage")
7  public  class  LoginPageServlet  extends  HttpServlet
8  {
9      protected  void  doGet(
10         HttpServletRequest  request,
11         HttpServletResponse  response)
12         throws  ServletException, IOException
13     {
```

（次ページに続く）

```
14          response.setContentType("text/html;charset=UTF-8");
15          PrintWriter out = response.getWriter();
16          out.println("<html>");
17          out.println("<head><title>ログインページ</title></head>");
18          out.println("<body>");
19          out.println("<form action='LoginPage' method='POST'>");
20          out.println("<p>");
21          out.println("ID");
22          out.println("<input type='text' size='50' name='id' />");
23          out.println("</p>");
24          out.println("<p>");
25          out.println("PASS");
26          out.println("<input type='password' size='50' name='pass' />");
27          out.println("</p>");
28          out.println("<input type='submit' value='ログイン' />");
29          out.println("</form>");
30          out.println("</body>");
31          out.println("</html>");
32
33          out.close();
34      }
35  }
```

Javaサーブレット利用時のURLは

http://localhost:8080　～　/LoginPage

になります。なお，上記の“/LoginPage”部分は，6行目の「@WebServlet("/LoginPage")」によって決定されます。これを「WebServletアノテーション」と呼びます。

また，上記の“～”部分は，コンテキスト記述子で決定します。

コンテキスト記述子（server.xml）を更新します。

WebアプリケーションをWebコンテナに登録します。これに伴い，Javaサーブレット利用時のURLの一部が決定します。

実際に，コンテキスト記述子を見てみましょう。

※Eclipse使用時は，server.xmlの記述は，Tomcatサーバにプロジェクトを登録すると自動的に更新されます。

プログラム 1-2　（ファイル名：server.xml）

```
124                 :
125                 <Context docBase="CommonApp" path="/CommonApp"
126                     reloadable="true" />
127             </Host>
128         </Engine>
129     </Service>
130 </Server>
```

・コンテキスト記述子は，Webコンテナのインストール時に作成されます。

・コンテキスト記述子の内部は，XML形式で記述します。

・<Host>タグ内の<Context>タグには，docBase属性，path属性，reloadable属性を記述します（125，126行目）。

- docBase属性　：Webアプリケーションルート名を記述します。
- path属性　：Javaサーブレット利用時のURLの一部を記述します。先頭には必ず‘/’を付加しなければいけません。
- reloadable属性　：Webアプリケーションに変更があったときに，Webコンテナが自動的にそれを再ロードするかどうか（true：再ロードする／false：再ロードしない）を設定します。

プログラム1-2では，Javaサーブレット利用時のURLは

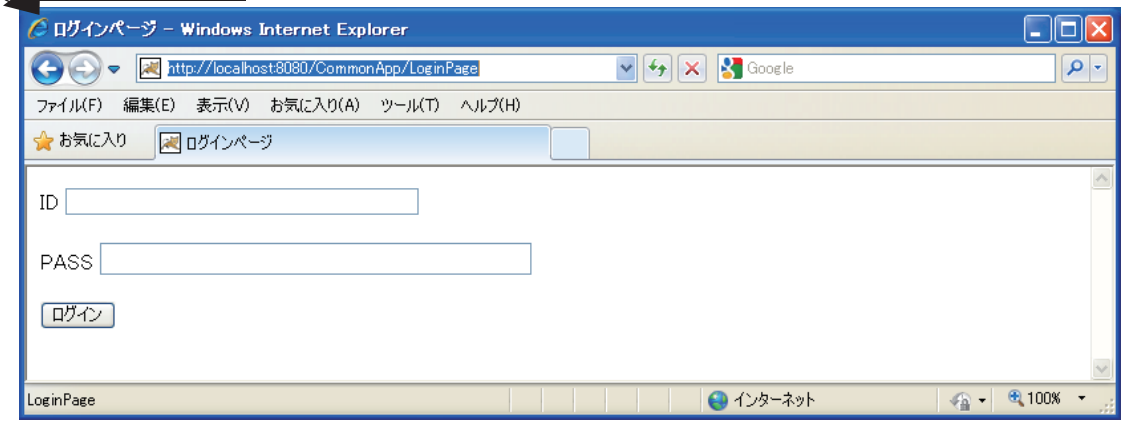
```
http://localhost:8080/CommonApp/LoginPage
```

になります。

プログラム1-1，1-2で作成・更新したJavaサーブレットと配備記述子は，次の場所に保存されています。

図５　Javaサーブレットと配備記述子の保存場所

実行結果　(URL：http://localhost:8080/CommonApp/LoginPage)



練習 1-1　レベル ☆

学校の期末試験の英語，国語，数学，理科，社会の得点を表示するJavaサーブレットのプログラムを作成しなさい。なお，各得点はint型の二次元配列scoreに記録されているものとします。

二次元配列

score	（英語）	（国語）	（数学）	（理科）	（社会）
（1人目）	67	72	54	78	83
（2人目）	55	84	51	48	91
（3人目）	70	57	62	66	96

ヒント　（ファイル名：Ren1_1_Servlet.java　URL：/ExamScore）

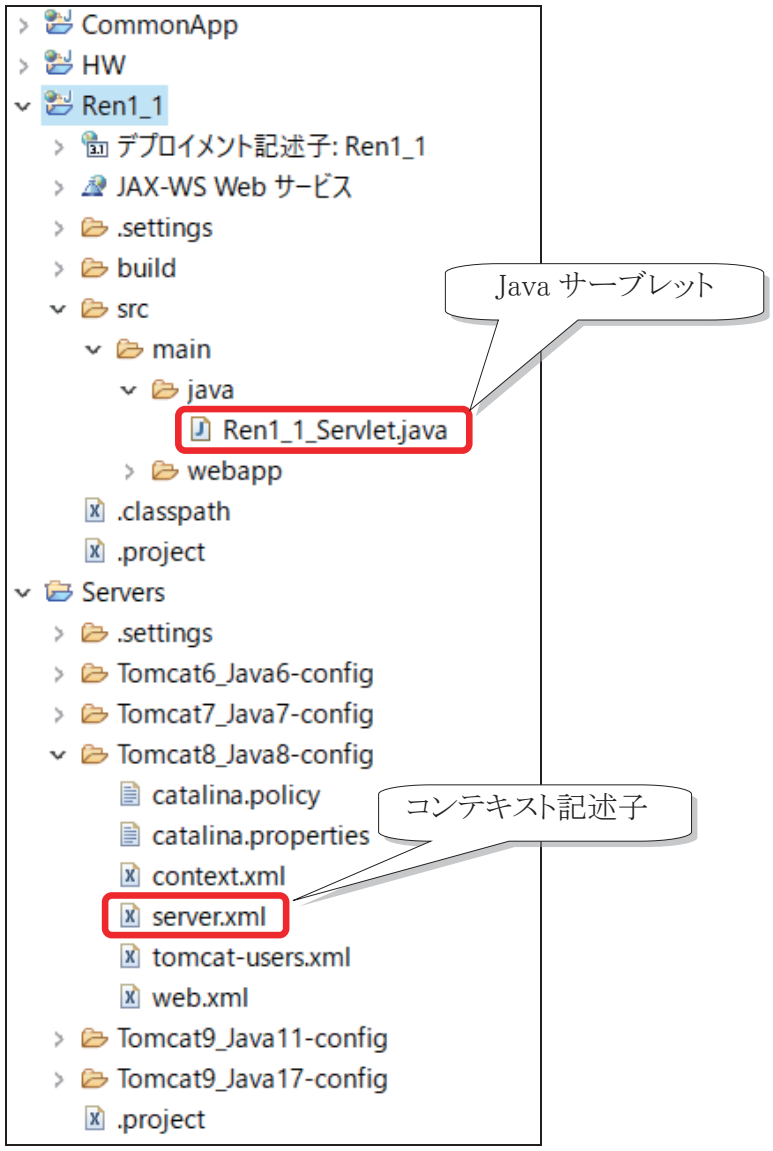
```
import  javax.servlet.*;
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;

@WebServlet("/ExamScore")
public  class  Ren1_1_Servlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse  response)
        throws  ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //学校の期末試験の英語，国語，数学，理科，社会の得点
        int[][] score = {{67, 72, 54, 78, 83},
                         {55, 84, 51, 48, 91},
                         {70, 57, 62, 66, 96}};
        out.println("<html>");
        out.println("<head><title>期末試験の得点</title></head>");
```

```
        out.println("<body>");
        out.println("<div align='center'>");
        out.println("<table border='1'>");
        out.println("<tr><th></th><th>英語</th><th>国語</th>"
            + "<th>数学</th><th>理科</th><th>社会</th></tr>");
        int i;          //行の添字
        int j;          //列の添字
        for(i = 0; i < score.length; i++)
        {
            :
            for(j = 0; j < score[i].length; j++)
            {
                :
            }
            :
        }
        out.println("</table>");
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");

        out.close();
    }
}
```

プログラム　（ファイル名：server.xml）

```
124                    :
125                    <Context docBase="Ren1_1" path="/Ren1_1"
126                        reloadable="true" />
127                </Host>
128            </Engine>
129        </Service>
130    </Server>
```

［ファイルの保存場所］



実行結果　(URL：http://localhost:8080/Ren1_1/ExamScore)

	英語	国語	数学	理科	社会
1人目	67	72	54	78	83
2人目	55	84	51	48	91
3人目	70	57	62	66	96

練習 1-2　レベル ☆

受注した東京都の顧客情報を表示するJavaサーブレットのプログラムを作成しなさい。なお，各顧客情報はStringクラス型の二次元配列infoに記録されているものとします。

二次元配列
info

（受注ID）	（受注日）	（顧客ID）	（顧客名）
0003	2005/12/22	005	小島　千里
0004	2006/01/11	007	太田　誠人
0005	2006/01/18	002	金子　実
0007	2006/02/01	010	佐藤　美佑
0008	2006/02/10	014	小林　和佳子
0012	2006/04/11	019	安藤　由紀子
0016	2006/05/17	012	宮　正澄
0018	2006/06/05	007	太田　誠人
0023	2006/07/11	010	佐藤　美佑
0029	2006/08/11	010	佐藤　美佑
0031	2006/08/30	012	宮　正澄
0034	2006/10/04	019	安藤　由紀子

ヒント　（ファイル名：Ren1_2_Servlet.java　URL:/TokyoCustomerInfo）

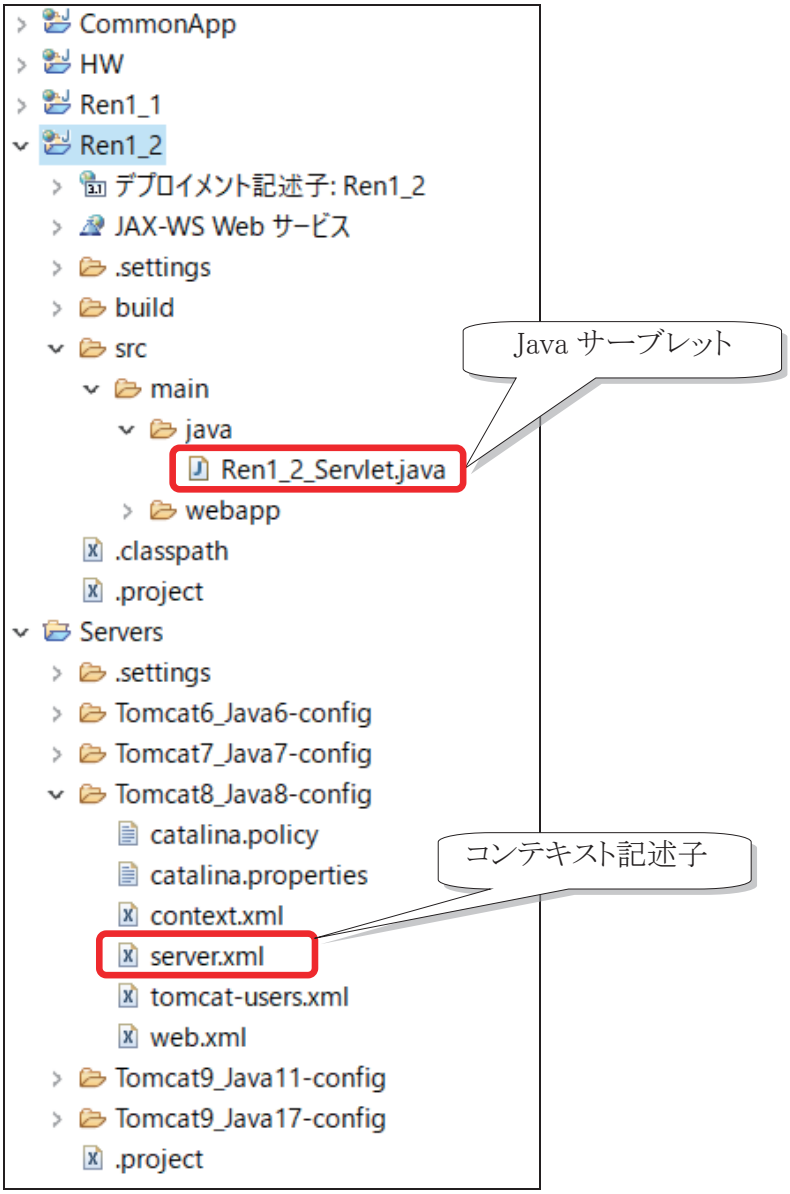
```
import  javax.servlet.*;
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;

@WebServlet("/TokyoCustomerInfo")
public  class  Ren1_2_Servlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse  response)
        throws  ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
```

```
        //受注した東京都の顧客情報
        String[][] info = {
            {"0003", "2005/12/22", "005", "小島　千里"},
            {"0004", "2006/01/11", "007", "太田　誠人"},
            {"0005", "2006/01/18", "002", "金子　実"},
            {"0007", "2006/02/01", "010", "佐藤　美佑"},
            {"0008", "2006/02/10", "014", "小林　和佳子"},
            {"0012", "2006/04/11", "019", "安藤　由紀子"},
            {"0016", "2006/05/17", "012", "宮　正澄"},
            {"0018", "2006/06/05", "007", "太田　誠人"},
            {"0023", "2006/07/11", "010", "佐藤　美佑"},
            {"0029", "2006/08/11", "010", "佐藤　美佑"},
            {"0031", "2006/08/30", "012", "宮　正澄"},
            {"0034", "2006/10/04", "019", "安藤　由紀子"}};

        out.println("<html>");
        out.println("<head><title>受注した東京都の顧客情報</title></head>");
        out.println("<body>");
        out.println("<div align='center'>");
        out.println("<table border='1'>");
        :
        out.println("</table>");
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");

        out.close();
    }
}
```

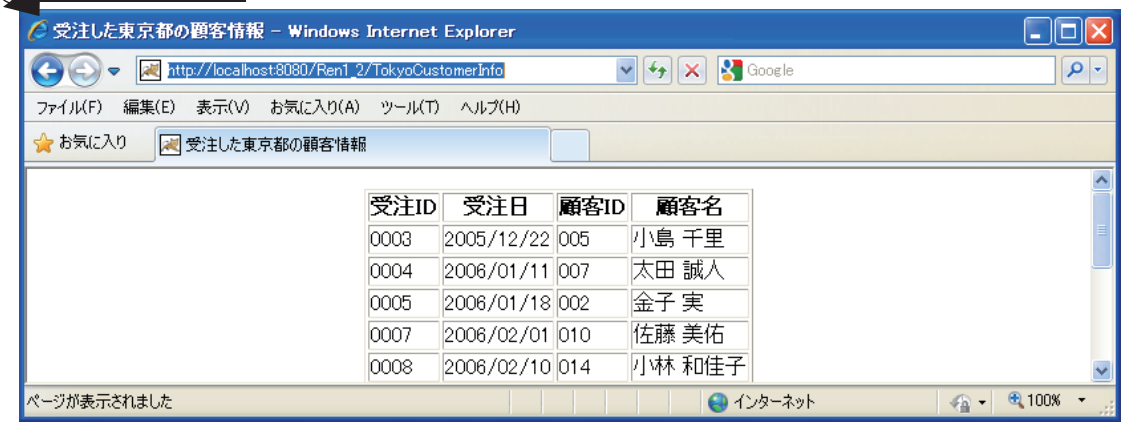
プログラム　（ファイル名：server.xml）

```
124                    :
125                    <Context docBase="Ren1_2" path="/Ren1_2"
126                        reloadable="true" />
127                </Host>
128            </Engine>
129        </Service>
130    </Server>
```

［ファイルの保存場所］



実行結果　(URL：http://localhost:8080/Ren1_2/TokyoCustomerInfo)



受注ID	受注日	顧客ID	顧客名
0003	2005/12/22	005	小島 千里
0004	2006/01/11	007	太田 誠人
0005	2006/01/18	002	金子 実
0007	2006/02/01	010	佐藤 美佑
0008	2006/02/10	014	小林 和佳子

※実行結果はスクロールして確認する（表のレコード数：12）。

練習 1-3　レベル ☆☆

JDBCプログラミングのプログラム1-3で作成したProductBeanクラス，練習1-9で作成したCustomerBeanクラス，練習1-10で作成したDivisionBeanクラスとOrderControlDBAccessクラスを使用して，DB（受注管理DB）の商品マスタ表から全レコードを取得して表示するJavaサーブレットのプログラムを作成しなさい。

ヒント　（ファイル名：Ren1_3_Servlet.java　URL：/ProductList）

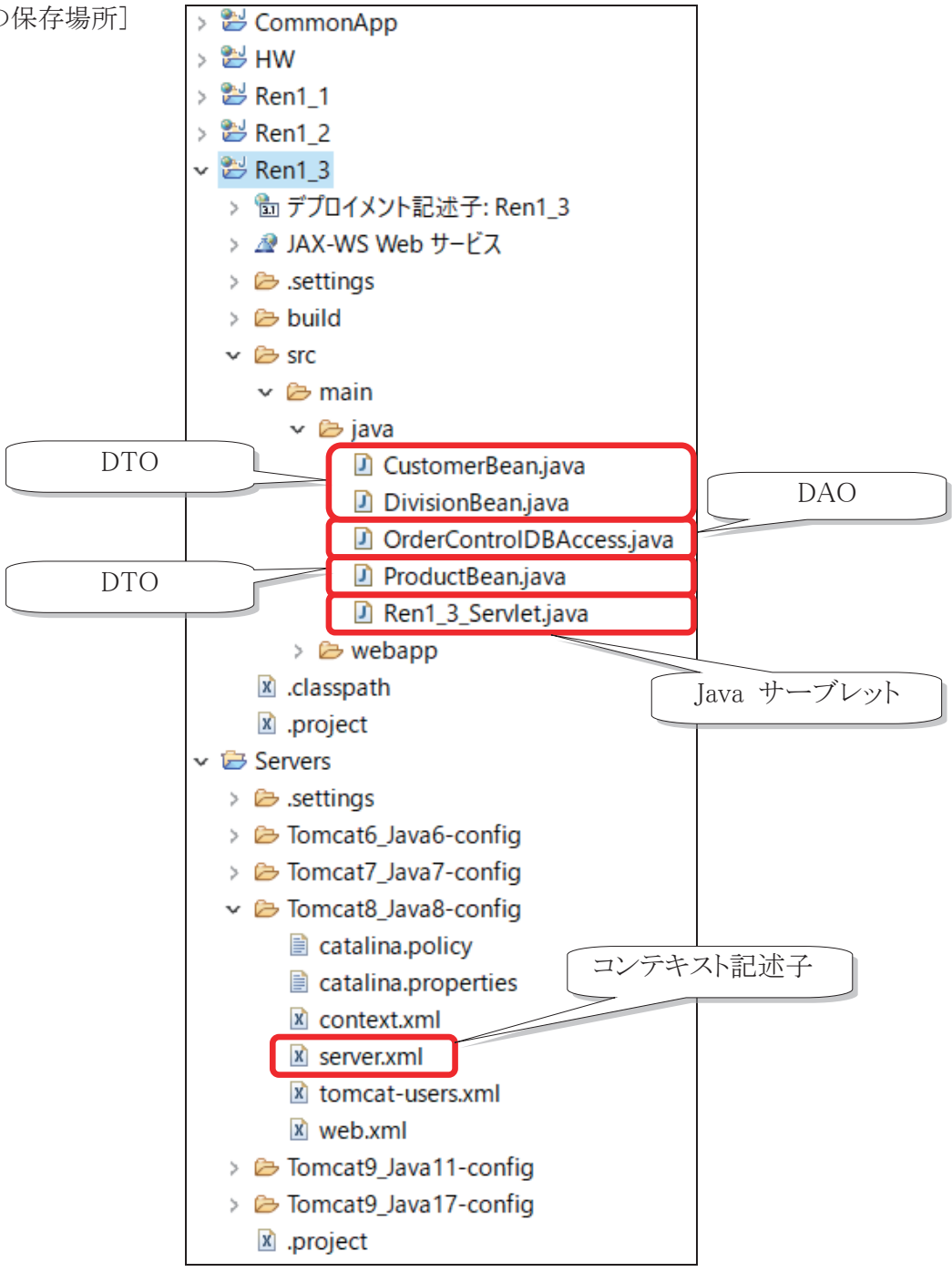
```
import  javax.servlet.*;
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;
import  java.util.ArrayList;

@WebServlet("/ProductList")
public  class  Ren1_3_Servlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse  response)
        throws  ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //DAOを生成する。
        OrderControlDBAccess dao = new OrderControlDBAccess();
        //DAOから商品マスタ表の全レコードを取得する。
        ArrayList<ProductBean> list = dao. findAllProducts();

        out.println("<html>");
        out.println("<head><title>商品マスタ表</title></head>");
        out.println("<body>");
```

```
        out.println("<div align='center'>");
        out.println("<table border='1'>");
        out.println("<caption>商品マスタ表</caption>");
        out.println(
            "<tr><th>商品ID</th><th>商品名</th><th>単価</th></tr>");
        for(ProductBean bean : list)
        {
            :
            out.print("<td>" + bean.getId() + "</td>");
            :
        }
        out.println("</table>");
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```

プログラム　（ファイル名：server.xml）

```
124                 :
125                 <Context  docBase="Ren1_3" path="/Ren1_3"
126                     reloadable="true" />
127             </Host>
128         </Engine>
129     </Service>
130 </Server>
```

［ファイルの保存場所］


CommonApp
HW
Ren1_1
Ren1_2
Ren1_3
デプロイメント記述子: Ren1_3
JAX-WS Web サービス
.settings
build
src
main
java
CustomerBean.java
DivisionBean.java
OrderControlDBAccess.java
ProductBean.java
Ren1_3_Servlet.java
webapp
.classpath
.project
Servers
.settings
Tomcat6_Java6-config
Tomcat7_Java7-config
Tomcat8_Java8-config
catalina.policy
catalina.properties
context.xml
server.xml
tomcat-users.xml
web.xml
Tomcat9_Java11-config
Tomcat9_Java17-config
.project
DTO
DTO
DAO
Java サーブレット
コンテキスト記述子

実行結果　(URL：http://localhost:8080/Ren1_3/ProductList)

商品マスタ表

商品ID	商品名	単価
F-001	ストロベリー	¥1580
F-002	ピーチ	¥1480
F-003	マンゴ	¥1480
F-004	トロピカル	¥1300
F-005	オレンジ	¥1480
H-001	ローズヒップ	¥1800
H-002	カモミール	¥1600
H-003	レモンバーム	¥1700
H-004	ペパーミント	¥1600
H-005	ハイビスカスフラワー	¥1800
O-001	アイリッシュモルト	¥1400
T-001	ダージリン	¥2300
T-002	アッサム	¥1600
T-003	セイロン	¥1500
T-004	キャラメル	¥1400
T-005	アールグレイ	¥1500
T-006	バニラチャイ	¥1780
T-007	ドゥドロップス	¥1600

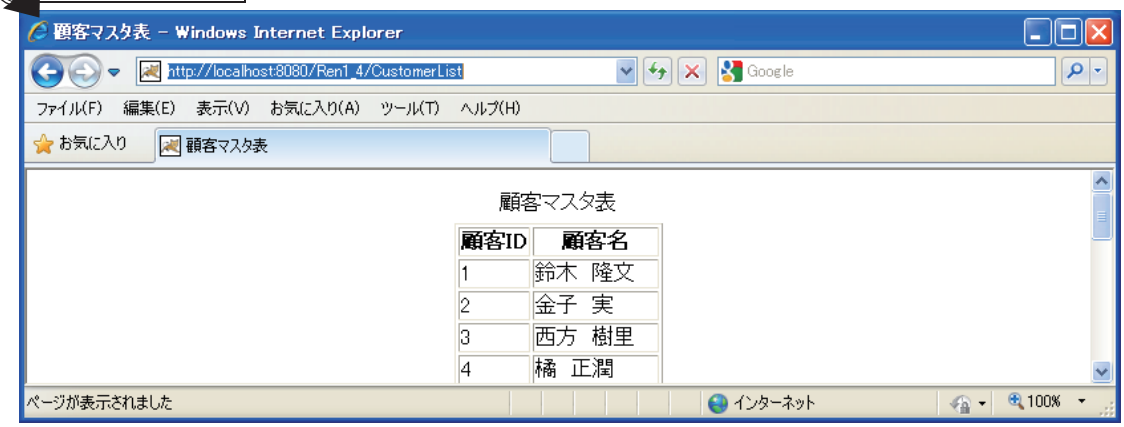
練習 1-4　レベル ☆☆

JDBCプログラミングのプログラム1-3で作成したProductBeanクラス，練習1-9で作成したCustomerBeanクラス，練習1-10で作成したDivisionBeanクラスとOrderControlDBAccessクラスを使用して，DB（受注管理DB）の顧客マスタ表から全レコードを取得して表示するJavaサーブレットのプログラムを作成しなさい。（ファイル名：Ren1_4_Servlet.java
URL：/CustomerList）

プログラム　（ファイル名：server.xml）

```
124                  :
125                  <Context  docBase="Ren1_4" path="/Ren1_4"
126                      reloadable="true" />
127              </Host>
128          </Engine>
129      </Service>
130  </Server>
```

実行結果　（URL：http://localhost:8080/Ren1_4/CustomerList）



※実行結果はスクロールして確認する（表のレコード数：20）。

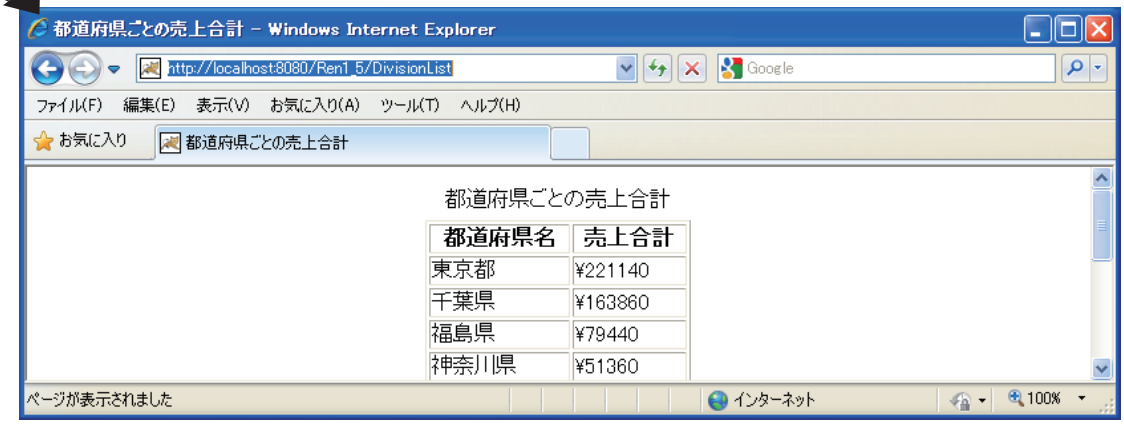
練習 1-5　レベル ☆☆

JDBCプログラミングのプログラム1-3で作成したProductBeanクラス，練習1-9で作成したCustomerBeanクラス，練習1-10で作成したDivisionBeanクラスとOrderControlDBAccessクラスを使用して，DB（受注管理DB）の顧客マスタ表，商品マスタ表，受注表，受注明細表を結合した表から，都道府県ごとの売上合計を取得して表示するJavaサーブレットのプログラムを作成しなさい。（ファイル名：Ren1_5_Servlet.java　URL：/DivisionList）

プログラム　（ファイル名：server.xml）

```
124                  :
125                  <Context  docBase="Ren1_5" path="/Ren1_5"
126                      reloadable="true" />
127              </Host>
128          </Engine>
129      </Service>
130  </Server>
```

実行結果　（URL：http://localhost:8080/Ren1_5/DivisionList）



※実行結果はスクロールして確認する（表のレコード数：12）。

1-2-2　フォームからデータを受け取る

Webブラウザに表示されたフォームにデータを入力して「ログイン」ボタンを押した場合，Webブラウザはそのデータを Webコンテナに送信します。

実際に，Javaサーブレットのプログラムを見てみましょう。

プログラム 1-1〔一部再掲〕　（ファイル名：LoginPageServlet.java）

```
13          :
14          out.println("<html>");
15          out.println("<head><title>ログインページ</title></head>");
16          out.println("<body>");
17          out.println("<form action='LoginPage' method='POST'>");
18          out.println("<p>");
19          out.println("ID");
20          out.println("<input type='text' size='50' name='id' />");
21          out.println("</p>");
22          out.println("<p>");
23          out.println("PASS");
24          out.println("<input type='password' size='50' name='pass' />");
25          out.println("</p>");
26          out.println("<input type='submit' value='ログイン' />");
27          out.println("</form>");
28          out.println("</body>");
29          out.println("</html>");
30          :
```

・17行目の<form>タグでは，action属性に“LoginPage”，method属性に“POST”を指定しています。このため，「ログイン」ボタン押下時には，URLが

http://localhost:8080/CommonApp/LoginPage

のJavaサーブレットに，POSTメソッドによるリクエストの送信を行います。

・20行目の<input>タグでは，name属性に“id”を指定しています。このため，「ログイン」ボタン押下時には，入力されたデータを“id=データ”の形式でリクエストに添付します。

・24行目の<input>タグでは，name属性に“**pass**”を指定しています。このため，「ログイン」ボタン押下時には，入力されたデータを“**pass=データ**”の形式でリクエストに添付します。

一方，リクエストを受信したJavaサーブレット側では，リクエストに添付されたデータ（リクエストパラメータ）を取り出します。リクエストパラメータの取得は，doGet()メソッド及びdoPost()メソッドの第１引数に記録された参照のインスタンス内のgetParameter()メソッドを使用します。

表３　getParameter()メソッドの仕様

項目	内容
メソッド名	getParameter()
引数	name　（String型，リクエストパラメータ名）
返却値	リクエストパラメータ値　（String型）
修飾子	public
例外	なし
概要	リクエストからリクエストパラメータ名nameに対応するリクエストパラメータ値を返す。リクエストパラメータ名nameが存在しない場合は，nullリテラル値「null」を返す。

実際に，Javaサーブレットのプログラムを見てみましょう。

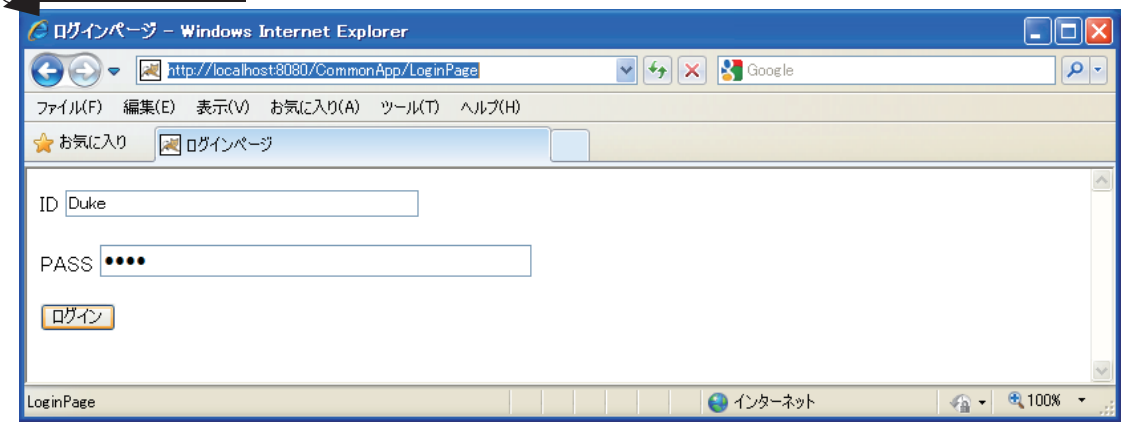
プログラム 1-3　（ファイル名：LoginPageServlet.java）

```
      :
34    protected  void  doPost(
35        HttpServletRequest  request,
36        HttpServletResponse  response)
37        throws  ServletException, IOException
38    {
39        //リクエストパラメータidを取り出す
40        String id = request.getParameter("id");
41        //リクエストパラメータpassを取り出す
42        String pass = request.getParameter("pass");
43
44        System.out.println("id  :" + id);        //コンソール出力
45        System.out.println("pass:" + pass);      //コンソール出力
46    }
47  }
```

・34～46行目は，30ページ17行目の<form>タグのmethod属性の“POST”の指定に従って，doPost()メソッドを定義しています。

・40，42行目のgetParameter()メソッドは，引数に指定した名前のリクエストパラメータ名に対応するリクエストパラメータ値を文字列（Stringクラス型）として返します。リクエストパラメータ名が存在しない場合は，nullリテラル値「null」を返します。

実行結果　(URL：http://localhost:8080/CommonApp/LoginPage)



[Tomcatコンソール]

```
id   : Duke
pass : java
```

1-2-3　インクルード／フォワード

Javaサーブレット（クラス）が他クラスを利用する場合，次の２つの方法があります。

① Javaサーブレットが**他クラス（非Javaサーブレット）**を利用する場合

Javaサーブレットのプログラム（クラス）では，標準クラスライブラリや独自クラスを利用することができます。その方法は，生成したインスタンス内のメソッドの呼出しやスタティックメソッドの呼出しなど，これまで学習してきたクラスの利用方法と同じです。

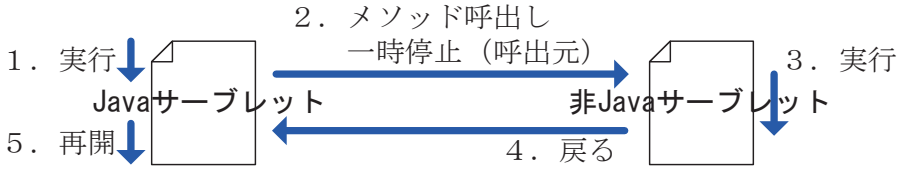


図６　他クラス（非Javaサーブレット）のメソッドを呼び出す場合

② Javaサーブレットが**他クラス（Javaサーブレット）**を使用する場合

Javaサーブレットのプログラム（クラス）では，他のJavaサーブレット（クラス）を利用することができます。その方法は，他のJavaサーブレットの“インクルード”，他のサーブレットへの“フォワード”です（メソッドの呼出しではありません）。

－ インクルード

：JavaサーブレットＡ（インクルード元）実行中に，他のJavaサーブレットＢ（インクルード先）をインクルードすると，JavaサーブレットＡの実行が一時停止して，JavaサーブレットＢの実行が開始されます。さらに，JavaサーブレットＢの実行が終了すると，JavaサーブレットＡの実行が再開されます。

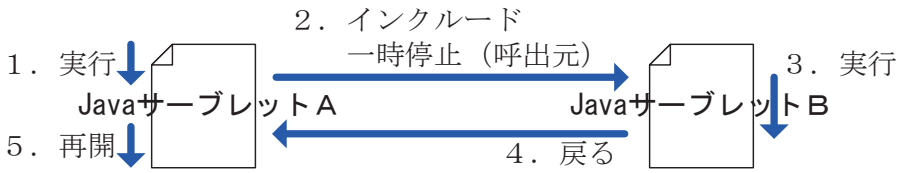


図７　他クラス（Javaサーブレット）をインクルードする場合

－ フォワード

：JavaサーブレットＡ（フォワード元）実行中に，他のJavaサーブレットＢ（フォワード先）にフォワードすると，JavaサーブレットＡの実行が終了して，JavaサーブレットＢの実行が開始されます。なお，JavaサーブレットＢの実行が終了しても，JavaサーブレットＡの実行は再開されません。

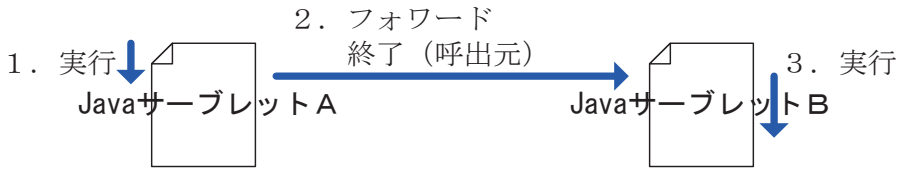


図８　他クラス（Javaサーブレット）をフォワードする場合

他のJavaサーブレットの“インクルード”や，他のサーブレットへの“フォワード”は，サーブレットコンテキスト（≒Webコンテナ）から取得したリクエストディスパッチャ（≒他のJavaサーブレットを実行する環境など）に対して指示します。サーブレットコンテキストの取得は，HttpServletクラスから継承したgetServletContext()メソッドを使用します。また，リクエストディスパッチャの取得は，サーブレットコンテキスト内のgetRequestDispatcher()メソッドを使用します。

表４　getServletContext()メソッドの仕様

項目	内容
メソッド名	getServletContext()
引数	なし
返却値	サーブレットコンテキスト　(ServletContext型)
修飾子	public
例外	なし
概要	サーブレットコンテキストを返す。

表５　getRequestDispatcher()メソッドの仕様

項目	内容
メソッド名	getRequestDispatcher()
引数	name　(String型，Javaサーブレット名)
返却値	リクエストディスパッチャ　(RequestDispatcher型)
修飾子	public
例外	なし
概要	リクエストディスパッチャを返す。

また，インクルードとフォワードは，リクエストディスパッチャ内のinclude()メソッドとforward()メソッドを使用します。

表６　include()メソッドの仕様

項目	内容
メソッド名	include()
引数	[第１引数] request (HttpServletRequest型，リクエスト) [第２引数] response (HttpServletResponse型，レスポンス)
返却値	なし (void型)
修飾子	public
例外	ServletException，IOException
概要	Javaサーブレットをインクルードする。

表７　forward()メソッドの仕様

項目	内容
メソッド名	forward()
引数	[第１引数] request (HttpServletRequest型，リクエスト) [第２引数] response (HttpServletResponse型，レスポンス)
返却値	なし (void型)
修飾子	public
例外	ServletException，IOException
概要	Javaサーブレットにフォワードする。

ここで，Webブラウザから送信されたデータ（id，pass）から利用者認証を行うために，次のDTO（プログラム1-4），DAO（プログラム1-5），DB（利用者管理DB）を準備しておきます。

プログラム 1-4　（ファイル名：UserBean.java）

```
1  import  java.io.Serializable;
2
3  //UserBeanクラス（DTO）の定義
4  public  class  UserBean  implements  Serializable
5  {
6      private  String  id;                    //ID
7      private  String  name;                  //氏名
8
9      public  UserBean(String  id, String  name)
10     {
11         setId(id);
12         setName(name);
13     }
14
15     public  void  setId(String  id)         //IDを設定する
16     {
17         this.id = id;
18     }
19
20     public  String  getId()                 //IDを取得する
21     {
22         return id;
23     }
24
25     public  void  setName(String  name)     //氏名を設定する
26     {
27         this.name = name;
28     }
29
```

（次ページに続く）

```
30      public  String  getName()              //氏名を取得する
31      {
32           return name;
33      }
34  }
```

プログラム 1-5　（ファイル名：UserControlDBAccess.java）

```
1   import  java.sql.*;
2
3   //UserControlDBAccessクラス（DAO）の定義
4   public  class  UserControlDBAccess
5   {
6       //DBとの接続を確立する
7       private  Connection  createConnection()
8       {
9           Connection con = null;
10          try
11          {
12              Class.forName(
13                  "com.mysql.cj.jdbc.Driver");
14              con = DriverManager.getConnection(
15                      "jdbc:mysql://localhost:65534/利用者管理DB",
16                      "user1",
17                      "pass1");
18          }
19          catch(ClassNotFoundException e)
20          {
21              System.out.println("JDBCドライバが見つかりません。");
22              e.printStackTrace();
23          }
24          catch(SQLException e)
25          {
26              System.out.println("DB接続時にエラーが発生しました。");
27              e.printStackTrace();
```

```
28          }
29          return con;
30      }
31
32      //DBとの接続を閉じる
33      private  void  closeConnection(Connection  con)
34      {
35          try
36          {
37              if(con != null)
38              {
39                  con.close();
40              }
41          }
42          catch(SQLException e)
43          {
44              System.out.println("DB切断時にエラーが発生しました。");
45              e.printStackTrace();
46          }
47      }
48
49      //IDとpassが一致する利用者を取得する
50      public  UserBean  findUserByIdAndPass(String  id, String  pass)
51      {
52          Connection con = createConnection();
53          PreparedStatement stmt = null;
54          ResultSet rs = null;
55          UserBean bean = null;
56          try
57          {
58              String sql = ("SELECT  ID,  氏名  FROM  利用者マスタ"
59              + "  WHERE BINARY ID = ? AND BINARY パスワード = ? ;");
60              //一度だけ最適化を行う
```

（次ページに続く）

```
61              stmt = con.prepareStatement(sql);
62              //SQL文に検索キーワードを埋め込む
63              stmt.setString(1, id);
64              stmt.setString(2, pass);
65              //SQL文を実行する
66              rs = stmt.executeQuery();
67              if(rs.next() == true)
68              {
69                  String name = rs.getString("氏名");
70                  bean = new UserBean(id, name);
71              }
72          }
73          catch(SQLException e)
74          {
75              System.out.println(
76                  "DBアクセス時にエラーが発生しました（利用者認証）。");
77              e.printStackTrace();
78          }
79          finally
80          {
81              try
82              {
83                  if(rs != null)
84                  {
85                      rs.close();
86                  }
87              }
88              catch(SQLException e)
89              {
90                  System.out.println(
91                      "DBアクセス時にエラーが発生しました。");
92                  e.printStackTrace();
93              }
94              try
```

```
95              {
96                  if(stmt != null)
97                  {
98                      stmt.close();
99                  }
100             }
101             catch(SQLException e)
102             {
103                 System.out.println(
104                     "DB アクセス時にエラーが発生しました。");
105                 e.printStackTrace();
106             }
107         }
108         closeConnection(con);
109         return bean;
110     }
111 }
```

実際に，Javaサーブレットのプログラムを見てみましょう。

プログラム 1-6　（ファイル名：LoginPageServlet.java）

```
31      :
32      protected  void  doPost(
33          HttpServletRequest  request,
34          HttpServletResponse   response)
35          throws  ServletException, IOException
36      {
37          //リクエストパラメータidを取り出す
38          String id = request.getParameter("id");
39          //リクエストパラメータpassを取り出す
40          String pass = request.getParameter("pass");
41          System.out.println("id :" + id);    //コンソール出力
42          System.out.println("pass:" + pass); //コンソール出力
43          //サーブレットコンテキストを取得する
44          ServletContext sc = getServletContext();
45          //リクエストディスパッチャを取得する
46          RequestDispatcher rd
47              = sc.getRequestDispatcher("/LoginProcess");
48          //他のJavaサーブレットにフォワードする
49          rd.forward(request, response);
50      }
51  }
```

・44行目のgetServletContext()メソッドは，サーブレットコンテキスト（≒Webコンテナ）の参照を返します。

・47行目のgetRequestDispatcher()メソッドは，リクエストディスパッチャ（≒引数で指定されたJavaサーブレットを実行する環境など）の参照を返します。なお，引数には，先頭にスラッシュ記号“/”を付与したJavaサーブレット名を指定します。

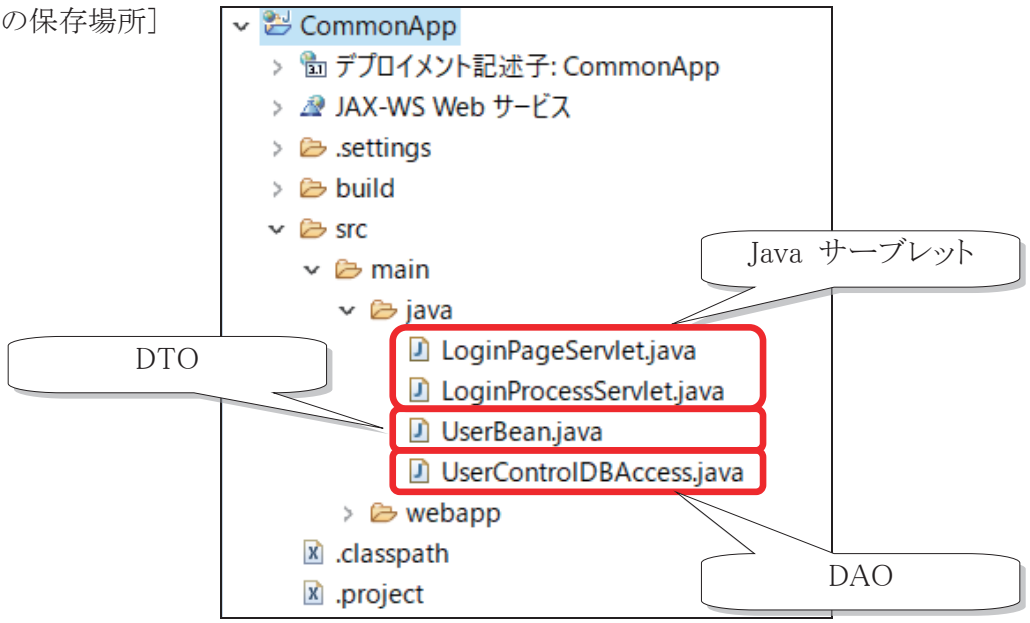
・49行目のforward()メソッドは，リクエストディスパッチャに指定したJavaサーブレットに“フォワード”しています。なお，リクエストディスパッチャに指定したJavaサーブレットの実行が終了しても，50行目以降のプログラムは実行されません。

プログラム 1-7　（ファイル名：LoginProcessServlet.java　URL：/LoginProcess）

```
1  import  javax.servlet.*;
2  import  javax.servlet.annotation.*;
3  import  javax.servlet.http.*;
4  import  java.io.*;
5
6  @WebServlet("/LoginProcess")
7  public  class  LoginProcessServlet  extends  HttpServlet
8  {
9      protected  void  doPost(
10         HttpServletRequest  request,
11         HttpServletResponse   response)
12         throws  ServletException, IOException
13     {
14         //リクエストパラメータidを取り出す
15         String id = request.getParameter("id");
16         //リクエストパラメータpassを取り出す
17         String pass = request.getParameter("pass");
18         UserControlDBAccess dao = new UserControlDBAccess();
19         UserBean bean = dao.findUserByIdAndPass(id, pass);
20         if(bean != null)
21         {
22             System.out.println("認証成功");       //コンソール出力
23         }
24         else
25         {
26             System.out.println("認証失敗");       //コンソール出力
27         }
28     }
29 }
```

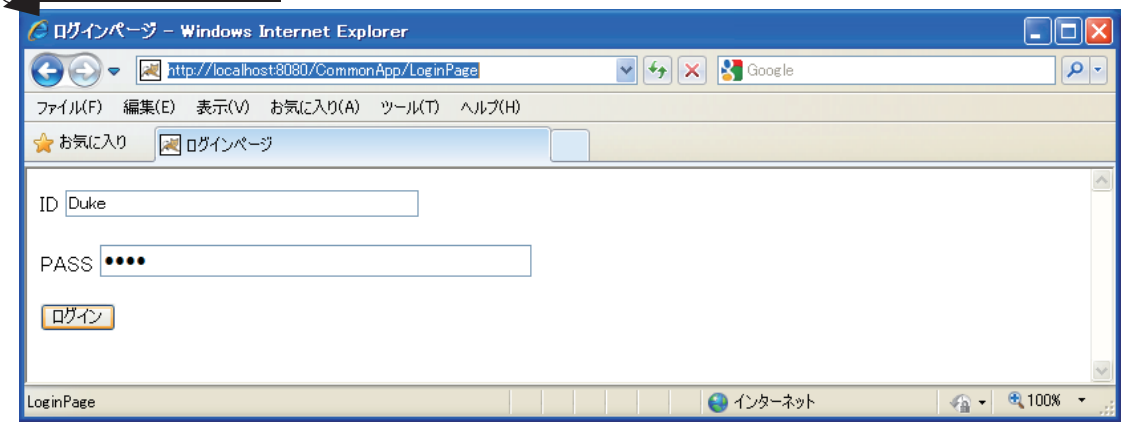
・19行目のfindUserByIdAndPass()メソッドは，リクエストパラメータ（id, pass）を使用して利用者認証を行い，認証成功時にはUserBeanクラス型のインスタンスの参照，認証失敗時にはnullリテラル値「null」を返します。

［ファイルの保存場所］



※以降はコンテキスト記述子（Server.xml）の場所は変更が無いので省略します。

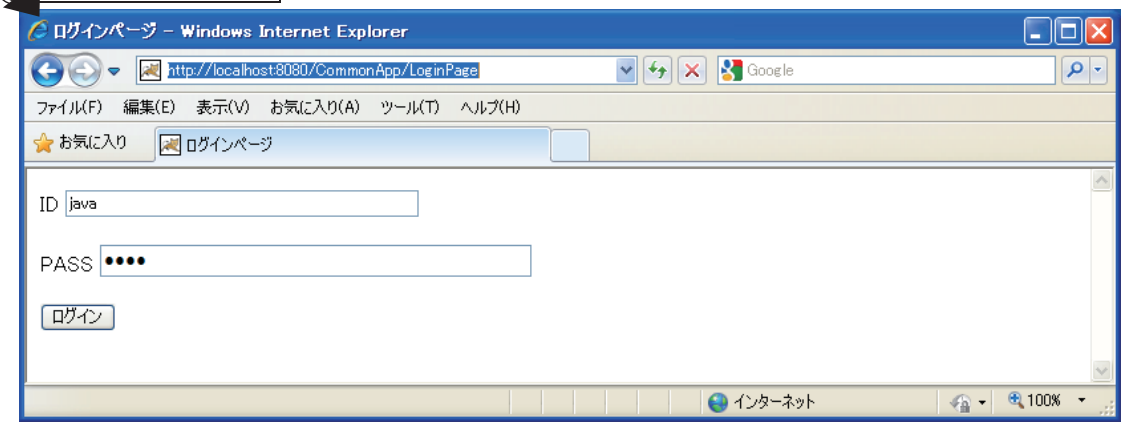
実行結果①　(URL：http://localhost:8080/CommonApp/LoginPage)



[Tomcatコンソール]

```
認証成功
```

実行結果②　(URL：http://localhost:8080/CommonApp/LoginPage)



[Tomcatコンソール]

```
認証失敗
```

プログラム1-6，1-7では，Javaサーブレット（LoginPageServletクラス）から，他のJavaサーブレット（LoginProcessServletクラス）にフォワードしましたが，WebブラウザのURLは“http://localhost:8080/CommonApp/LoginPage”のまま変わっていません。WebブラウザのURLを変更する方法は，「1-2-4　リダイレクション」で詳しく説明します。

練習 1-6　レベル ☆☆☆

JDBCプログラミングのプログラム1-3で作成したProductBeanクラス，練習1-9で作成したCustomerBeanクラス，練習1-10で作成したDivisionBeanクラスとOrderControlDBAccessクラスを使用して，DB（受注管理DB）の商品マスタ表から全レコードを取得して表示するJavaサーブレットと，DBの顧客マスタ表から全レコードを取得して表示するJavaサーブレットのプログラムをそれぞれ作成しなさい。なお，Ren1_6_Servletサーブレットでは，他のJavaサーブレットをインクルードすること。

ヒント　（ファイル名：Ren1_6_Page1_Servlet.java　URL：/ProductList）

```
import  javax.servlet.*;
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;
import  java.util.ArrayList;

@WebServlet("/ProductList")
public  class  Ren1_6_Page1_Servlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse   response)
        throws  ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //DAOを生成する。
        OrderControlDBAccess dao = new OrderControlDBAccess();
        //DAOから商品マスタ表の全レコードを取得する。
        ArrayList<ProductBean> list = dao.findAllProducts();

        out.println("<div align='center'>");
```

```
        out.println("<table border='1'>");
        out.println("<caption>商品マスタ表</caption>");
        out.println("<tr><th>商品コード</th><th>商品名</th>"
            + "<th>単価</th></tr>");
        for(ProductBean bean : list)
        {
            :
            out.print("<td>" + bean.getId() + "</td>");
            :
        }
        out.println("</table>");
        out.println("</div>");
    }                                   //「out.close();」は記述しない。
}
```

ヒント　（ファイル名：Ren1_6_Page2_Servlet.java　URL：/CustomerList）

```
import  javax.servlet.*;
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;
import  java.util.ArrayList;

@WebServlet("/CustomerList")
public  class  Ren1_6_Page2_Servlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse   response)
        throws  ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
```

（次ページに続く）

```
        //DAOを生成する。
        OrderControlDBAccess dao = new OrderControlDBAccess();
        //DAOから顧客マスタ表の全レコードを取得する。
        ArrayList<CustomerBean> list = dao.findAllCustomers();
        out.println("<div align='center'>");
        out.println("<table border='1'>");
        out.println("<caption>顧客マスタ表</caption>");
        out.println("<tr><th>顧客ID</th><th>顧客名</th></tr>");
        for(CustomerBean bean : list)
        {
            :
            out.print("<td>" + bean.getId() + "</td>");
            :
        }
        out.println("</table>");
        out.println("</div>");
    }                                   //「out.close();」は記述しない。
}
```

ヒント　（ファイル名：Ren1_6_Servlet.java　URL：/VariousData）

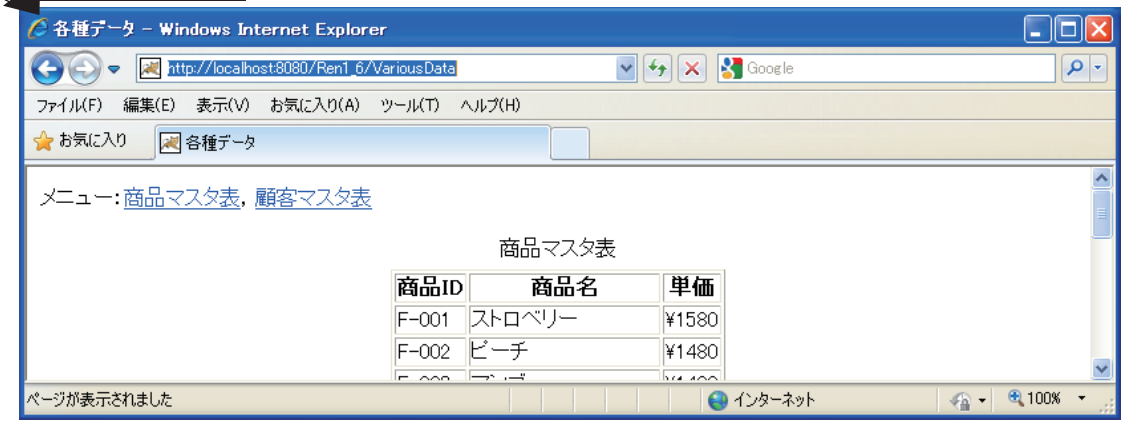
```
import  javax.servlet.*;
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;

@WebServlet("/VariousData")
public  class  Ren1_6_Servlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse   response)
        throws  ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
```

```
        PrintWriter out = response.getWriter();

        //サーブレットコンテキストを取得する
        ServletContext sc = getServletContext();
        //リクエストパラメータdispを取り出す
        String disp = request.getParameter("disp");
        out.println("<html>");
        out.println("<head><title>各種データ</title></head>");
        out.println("<body>");
        out.println("<p>メニュー："
            + "<a href='VariousData?disp=ProductList'>商品マスタ表</a>, "
            + "<a href='VariousData?disp=CustomerList'>顧客マスタ表</a>"
            + "</p>");
        if(disp == null || disp.equals("ProductList") == true)
        {
            :
        }
        else if(disp.equals("CustomerList") == true)
        {
            :
        }
        out.println("</body>");
        out.println("</html>");

        out.close();
    }
}
```

プログラム　（ファイル名：Server.xml）

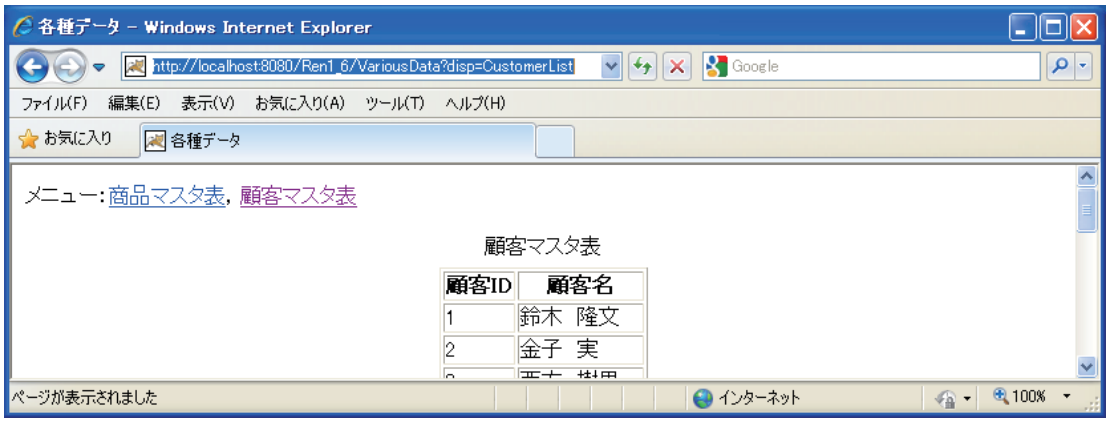
```
124                    :
125                    <Context docBase="Ren1_6" path="/Ren1_6"
126                        reloadable="true" />
127                </Host>
128            </Engine>
129        </Service>
130    </Server>
```

実行結果　（URL：http://localhost:8080/Ren1_6/VariousData）



※実行結果はスクロールして確認する（表のレコード数：18）。

「顧客マスタ表」クリック時　　「商品マスタ表」クリック時



※実行結果はスクロールして確認する（表のレコード数：20）。

1-2-4　リダイレクション

ログインページが表示されているWebブラウザでは，利用者認証の成功時にはメインページが表示されます。同時に，WebブラウザのURLも変わらなければいけません。

Javaサーブレットから，URLが異なるJavaサーブレット／JSP／HTMLファイルなどを利用するには，“リダイレクション”を行います。リダイレクションは，doGet()メソッドおよびdoPost()メソッドの第２引数に記録された参照のインスタンス内のsendRedirect()メソッドを使用します。

表８　sendRedirect()メソッドの仕様

項目	内容
メソッド名	sendRedirect()
引数	name　(String型，Javaサーブレット名)
返却値	なし　　(void型)
修飾子	public
例外	IOException，IllegalStateException
概要	引数に指定されたJavaサーブレットにリダイレクトする。

実際に，Javaサーブレットのプログラムを見てみましょう。

プログラム 1-8　（ファイル名：LoginProcessServlet.java）

```
15          :
16          UserControlDBAccess dao = new UserControlDBAccess();
17          UserBean bean = dao.findUserByIdAndPass(id, pass);
18          if(bean != null)
19          {
20             System.out.println("認証成功");
21             response.sendRedirect("MainPage");
22          }
23          else
24          {
25             System.out.println("認証失敗");
26             response.sendRedirect("LoginPage?status=failure");
27          }
28      }
29  }
```

・21行目のsendRedirect()メソッドは，Webコンテナから Webブラウザに，引数で指定されたJavaサーブレットのURL“http://localhost:8080/CommonApp/MainPage”を含んだレスポンスを送信します（レスポンスを受信したWebブラウザは，新しいURLのWebコンテナにリクエストを送信します）。なお，引数には，Javaサーブレット名を指定します。

・26行目も同様です。さらに，URLにリクエストパラメータ“status=failure”を添付しています（Javaサーブレット名とリクエストパラメータの間にはクエスチョン記号“?”，複数のリクエストパラメータの間にはアンパサンド記号“&”を付与します）。

プログラム 1-9　（ファイル名：MainPageServlet.java　URL：/MainPage）

```
1  import  javax.servlet.*;
2  import  javax.servlet.annotation.*;
3  import  javax.servlet.http.*;
4  import  java.io.*;
5
6  @WebServlet("/MainPage")
7  public  class  MainPageServlet  extends  HttpServlet
8  {
9      protected  void  doGet(
10         HttpServletRequest  request,
11         HttpServletResponse   response)
12         throws  ServletException, IOException
13     {
14         response.setContentType("text/html;charset=UTF-8");
15         PrintWriter out = response.getWriter();
16
17         out.println("<html>");
18         out.println("<head><title>メインページ</title></head>");
19         out.println("<body>");
20         out.println("<p>○○○さん，こんにちは。</p>");
21         out.println("</body>");
22         out.println("</html>");
23
24         out.close();
25     }
26     protected  void  doPost(
27         HttpServletRequest  request,
28         HttpServletResponse  response)
29         throws  ServletException, IOException
30     {
31         doGet(request, response);
32     }
33 }
```

・31行目は，9～25行目のdoGet()メソッドを呼び出しています。

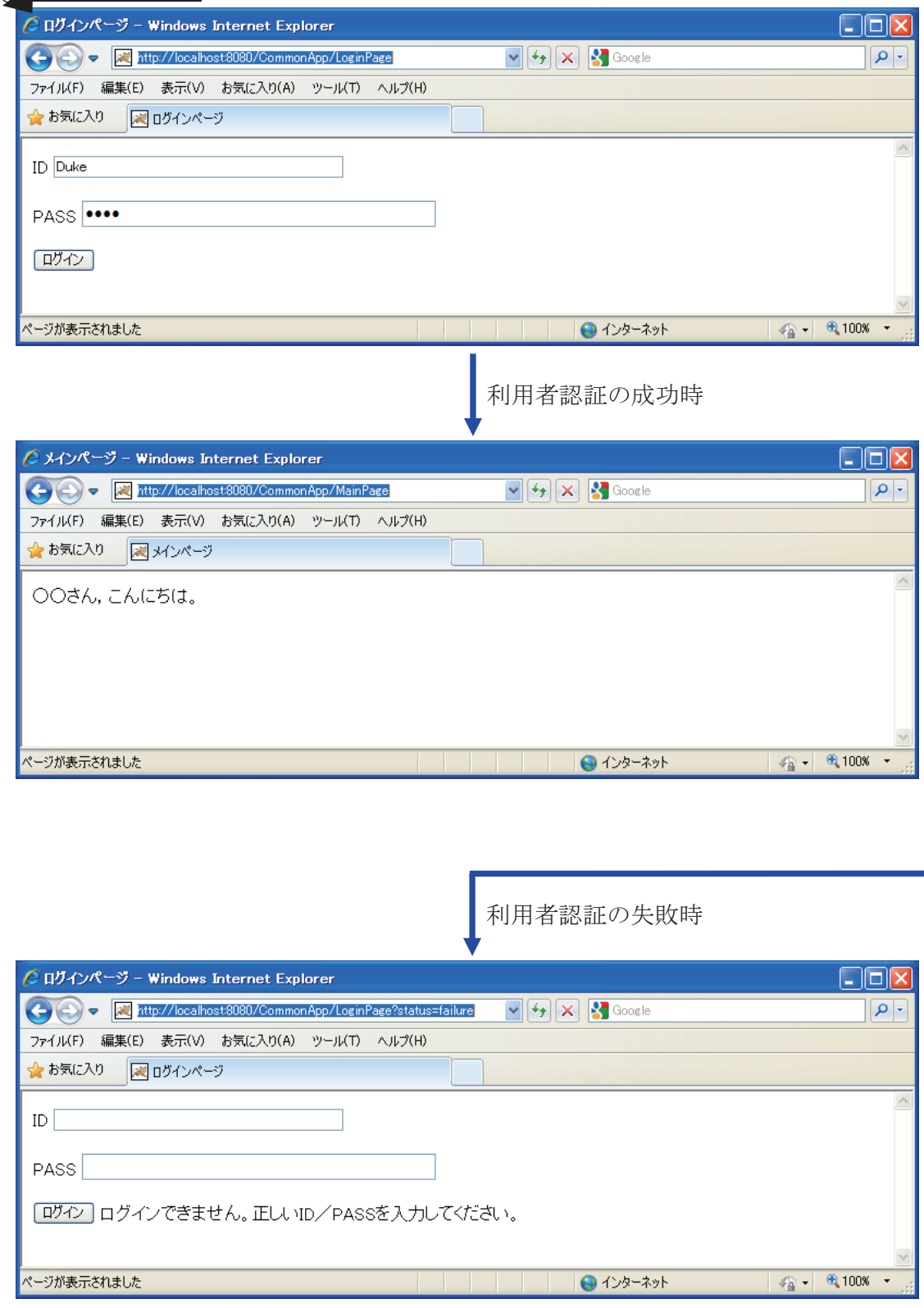
プログラム 1-10　（ファイル名：LoginPageServlet.java）

```
13          :
14          out.println("<html>");
15          out.println("<head><title>ログインページ</title></head>");
16          out.println("<body>");
17          out.println("<form action='LoginPage' method='POST'>");
18          out.println("<p>");
19          out.println("ID");
20          out.println("<input type='text' size='50' name='id' />");
21          out.println("</p>");
22          out.println("<p>");
23          out.println("PASS");
24          out.println("<input type='password' size='50' name='pass' />");
25          out.println("</p>");
26          out.println("<input type='submit' value='ログイン' />");
27          String msg = request.getParameter("status");
28          if(msg != null && msg.equals("failure") == true)
29          {
30              out.println(
31                  "ログインできません。正しいID／PASSを入力してください。");
32          }
33          out.println("</form>");
34          out.println("</body>");
35          out.println("</html>");
36          :
```

・27行目のgetParameter()メソッドは，リクエストパラメータ名“status”に対応するリクエストパラメータ値を文字列（String型）として返します。存在しない場合はnullリテラル値「null」を返します。

・28～32行目は，リクエストパラメータ名“status”が存在し，かつリクエストパラメータ値が文字列リテラル値“failure”と一致する場合に，「ログイン」ボタンの右横に再入力を促すメッセージを表示します。

実行結果　(URL：http://localhost:8080/CommonApp/LoginPage)



1-2-5　フォワードとリダイレクションの違い

フォワードとリダイレクションは，どちらも他のJavaサーブレットを呼び出しますが，内部の処理方式が異なっています。

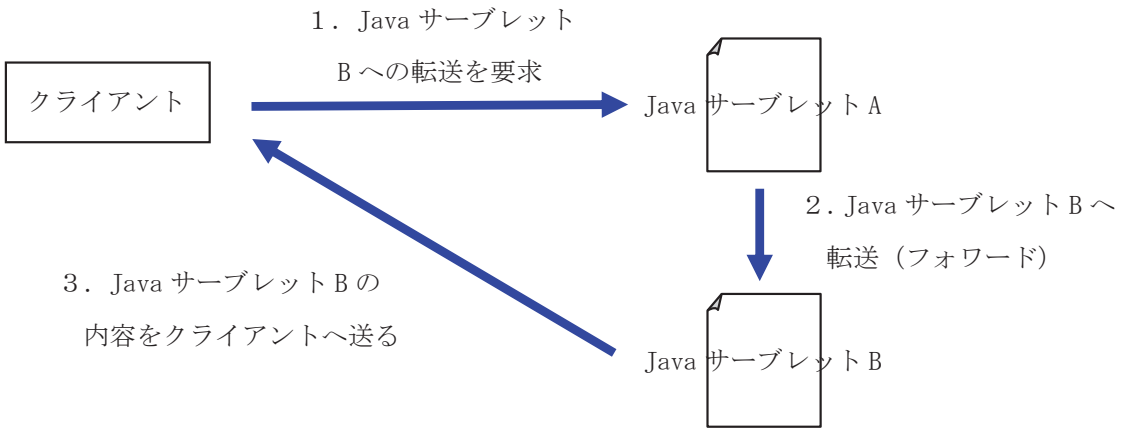


図９　フォワードの処理の遷移

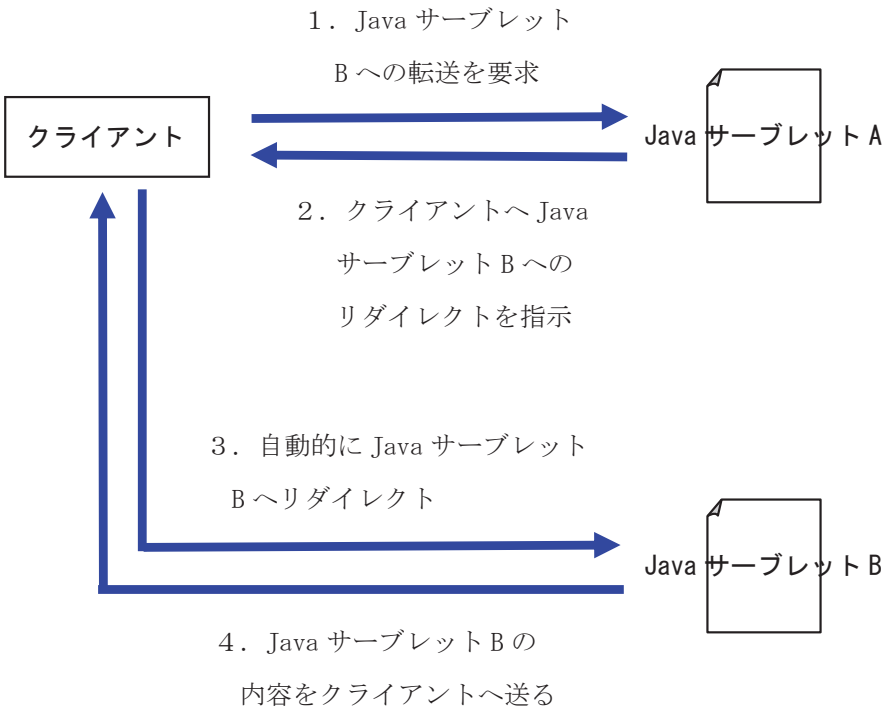


図10　リダイレクションの処理の遷移

フォワードとリダイレクションの主な特徴の違いは次の通りです。

表９　フォワードとリダイレクションの比較

項目	フォワード	リダイレクション
パフォーマンス	良い（リクエスト発生1回）	悪い（リクエスト発生2回）
リクエスト情報	引き継ぐ（同一リクエストの為）	引き継げない(別のリクエストが発生している為)
別サーバへ転送	できない(同一サーバ内の転送に限られる)	できる
URLの変更	されない	される

リダイレクションとフォワードの使い分けは次の通りです。

ーフォワードを使う場合

・同一サーバ内のJavaサーブレットを呼び出す場合

・WebブラウザのURLを変更しなくても問題ない場合

ーリダイレクションを使う場合

・別のサーバのJavaサーブレットを呼び出す場合

・WebブラウザのURLを変更しなければならない場合

練習 1-7　レベル ☆☆☆

JDBCプログラミングのプログラム1-3で作成したProductBeanクラス，練習1-9で作成したCustomerBeanクラス，練習1-10で作成したDivisionBeanクラスとOrderControlDBAccessクラスを使用して，DB（受注管理DB）の商品マスタ表から全レコードを取得して表示するJavaサーブレットと，DBの顧客マスタ表から全レコードを取得して表示するJavaサーブレットのプログラムをそれぞれ作成しなさい。なお，Ren1_7_Servletサーブレットでは，他のJavaサーブレットにリダイレクトすること。

ヒント　（ファイル名：Ren1_7_Page1_Servlet.java　URL：/ProductList）

```
import  javax.servlet.*;
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;
import  java.util.ArrayList;

@WebServlet("/ProductList")
public  class  Ren1_7_Page1_Servlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse  response)
        throws  ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //DAOを生成する。
        OrderControlDBAccess dao = new OrderControlDBAccess();
        //DAOから商品マスタ表の全レコードを取得する。
        ArrayList<ProductBean> list = dao.findAllProducts();

        out.println("<html>");
```

```
        out.println("<head><title>各種データ</title></head>");
        out.println("<body>");
        out.println("<p>メニュー："
            + "<a href='VariousData?disp=ProductList'>商品マスタ表</a>, "
            + "<a href='VariousData?disp=CustomerList'>顧客マスタ表</a>"
            + "</p>");
        out.println("<div align='center'>");
        out.println("<table border='1'>");
        out.println("<caption>商品マスタ表</caption>");

        out.println("<tr><th>商品ID</th><th>商品名</th>"
            + "<th>単価</th></tr>");
        for(ProductBean bean : list)
        {
            :
            out.print("<td>" + bean.getId() + "</td>");
            :
        }
        out.println("</table>");
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");

        out.close();
    }
}
```

ヒント　（ファイル名：Ren1_7_Page2_Servlet.java　URL：/CustomerList）

```
import  javax.servlet.*;
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;
import  java.util.ArrayList;
```

（次ページに続く）

```
@WebServlet("/CustomerList")
public  class  Ren1_7_Page2_Servlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse  response)
        throws  ServletException, IOException
    {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //DAOを生成する。
        OrderControlDBAccess dao = new OrderControlDBAccess();
        //DAOから顧客マスタ表の全レコードを取得する。
        ArrayList<CustomerBean> list = dao.findAllCustomers();

        out.println("<html>");
        out.println("<head><title>各種データ</title></head>");
        out.println("<body>");
        out.println("<p>メニュー："
            + "<a href='VariousData?disp=ProductList'>商品マスタ表</a>, "
            + "<a href='VariousData?disp=CustomerList'>顧客マスタ表</a>"
            + "</p>");
        out.println("<div align='center'>");
        out.println("<table border='1'>");
        out.println("<caption>顧客マスタ表</caption>");
        out.println("<tr><th>顧客ID</th><th>顧客名</th></tr>");
        for(CustomerBean bean : list)
        {
            :
            out.print("<td>" + bean.getId() + "</td>");
            :
        }
        out.println("</table>");
```

```
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```

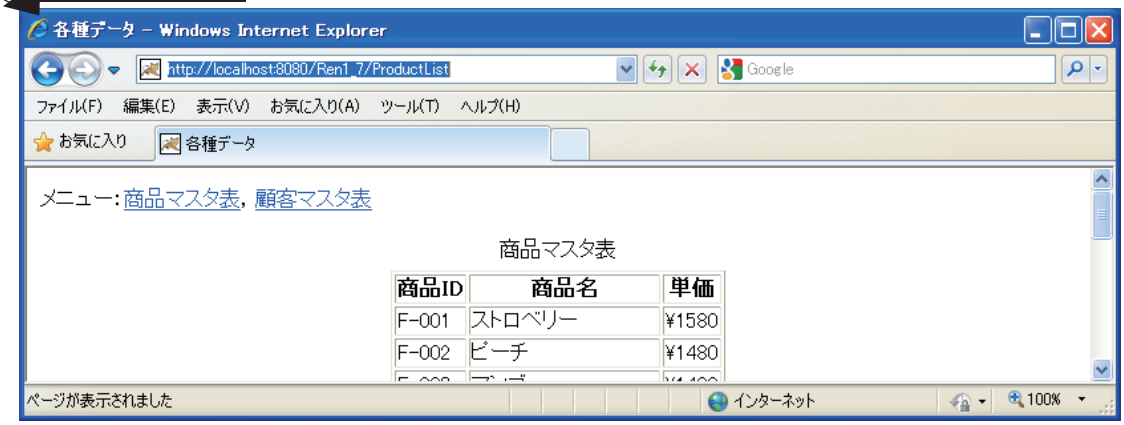
ヒント　（ファイル名：Ren1_7_Servlet.java　URL：/VariousData）

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

@WebServlet("/VariousData")
public  class  Ren1_7_Servlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse  response)
        throws  ServletException, IOException
    {
        //リクエストパラメータdispを取り出す
        String disp = request.getParameter("disp");

        if(disp == null || disp.equals("ProductList") == true)
        {
            :
        }
        else if(disp.equals("CustomerList") == true)
        {
            :
        }
    }
}
```

プログラム　（ファイル名：server.xml）

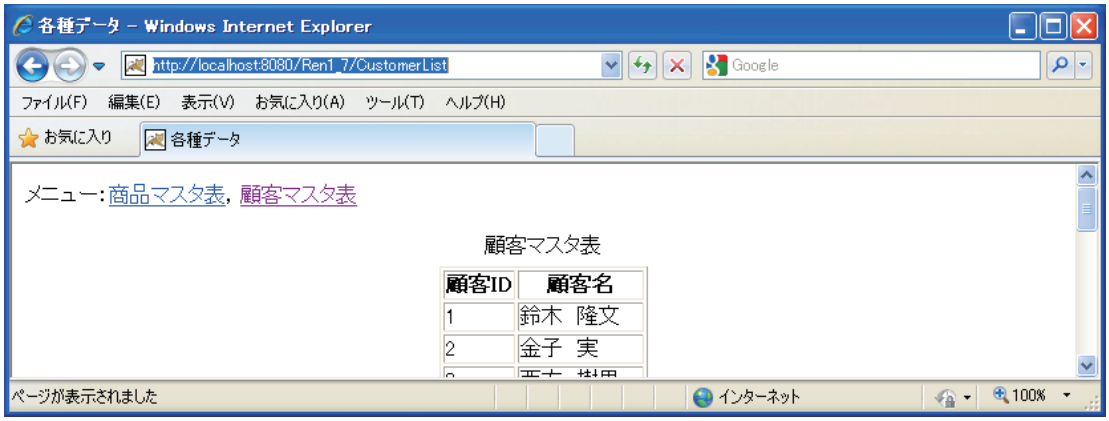
```
124                    :
125                    <Context docBase="Ren1_7" path="/Ren1_7"
126                        reloadable="true" />
127                </Host>
128            </Engine>
129        </Service>
130    </Server>
```

実行結果　（URL：http://localhost:8080/Ren1_7/VariousData）



※実行結果はスクロールして確認する（表のレコード数：18）。

「顧客マスタ表」クリック時　　「商品マスタ表」クリック時



※実行結果はスクロールして確認する（表のレコード数：20）。

1-2-6　スコープ

Webアプリケーションにはスコープ（scope）と呼ばれる概念があります。スコープは“範囲”という意味であり，Javaサーブレットからアクセス可能な，あるインスタンスの生存期間を表します。Javaサーブレットで定義されている三つのスコープは，次のとおりです。

表10　Javaサーブレットで定義されている三つのスコープ

種類	概要
リクエストスコープ	・Webブラウザからの１回のリクエストを有効範囲とする。 ・HttpServletRequestインタフェース型のインスタンスに“属性名”と“属性値”を格納する。
セッションスコープ	・特定のWebブラウザとWebコンテナ間における継続的な通信セッションを有効範囲とする。 ・HttpSessionインタフェース型のインスタンスに“属性名”と“属性値”を格納する。
アプリケーションスコープ	・Webアプリケーション全体を有効範囲とする。 ・ServletContextインタフェース型のインスタンスに“属性名”と“属性値”を格納する。

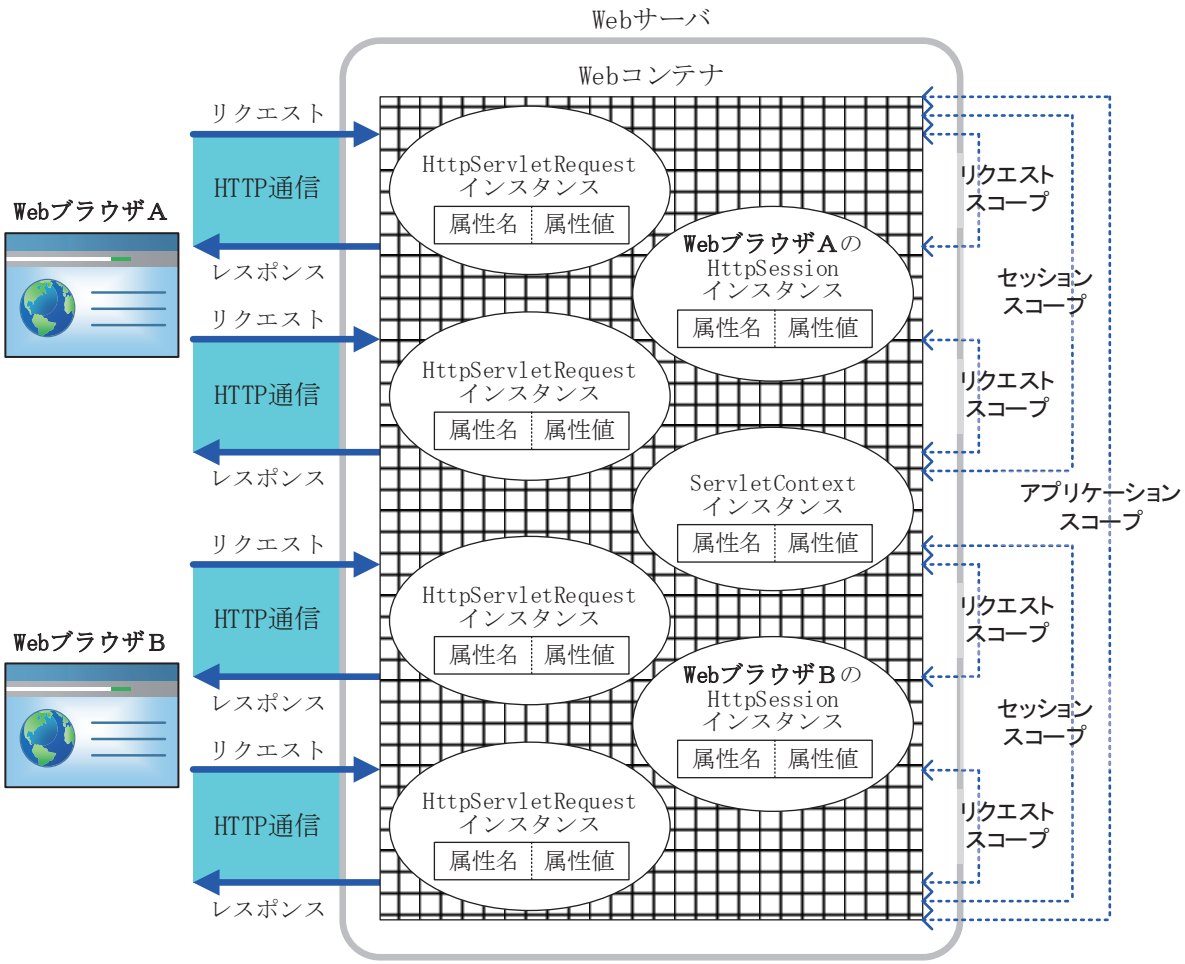
※「リクエストスコープ」は，Webアプリケーションで最も多く利用されるスコープです。Javaサーブレットのget()メソッドやpost()メソッドの引数requestを使用します。

※「セッションスコープ」は，一般的に利用者がWebアプリケーションにログインしたときに開始し，ログアウトしたときに終了します。すなわち，WebアプリケーションはWebブラウザごとにスコープを管理します。
（使用例．73ページ「プログラム1-13」，77ページ「プログラム1-14」）

※「アプリケーションスコープ」は，Webアプリケーションが一つだけ管理します（すべてのWebブラウザが同じスコープを使用します）。そのため，セキュリティの観点から，このスコープはほとんど利用されません。

※スコープへのアクセスは，setAttribute()メソッド，getAttribute()メソッド，removeAttribute()メソッドを使用します。詳しくは「1-2-8　通信セションとデータ共有」で説明します。

［Javaサーブレットで定義されている三つのスコープ］



1-2-7 設定ファイル

定数や表示するメッセージが多くなり，それらをプログラム内に記述していると記述箇所が分散して管理が複雑になります。

そこで，規定値などの定数や表示させるメッセージなどをプロパティファイルと呼ばれる設定ファイルに一括して記述しておき，それをプログラム内で読み込ませることで管理がしやすく，プログラムの変更にも対応しやすくなります。

プロパティファイルは拡張子を「.properties」のファイルで作成し，内容は「キー＝値」の形式で羅列します。

プログラム　（ファイル名：Message.properties）

```
fail=ログインに失敗しました。正しいID／PASSを入力してください。
```

実際に，プロパティファイルを見てみましょう。

設定ファイルの読み込みができるように，プログラム1-10を一部修正します。

プログラム 1-11　（ファイル名：LoginPageServlet.java）

```
 1  import  javax.servlet.*;
 2  import  javax.servlet.annotation.*;
 3  import  javax.servlet.http.*;
 4  import  java.io.*;
    import  java.util.*;
13          :
14         PrintWriter out = response.getWriter();
15         // Message.propertiesファイルを読み込む
16         Properties pr = new Properties();
17         Reader rd = new InputStreamReader(
18             LoginPageServlet.class.getResourceAsStream(
19                 "Message.properties"),"UTF-8");
20         pr.load(rd);
21         rd.close();
22
```

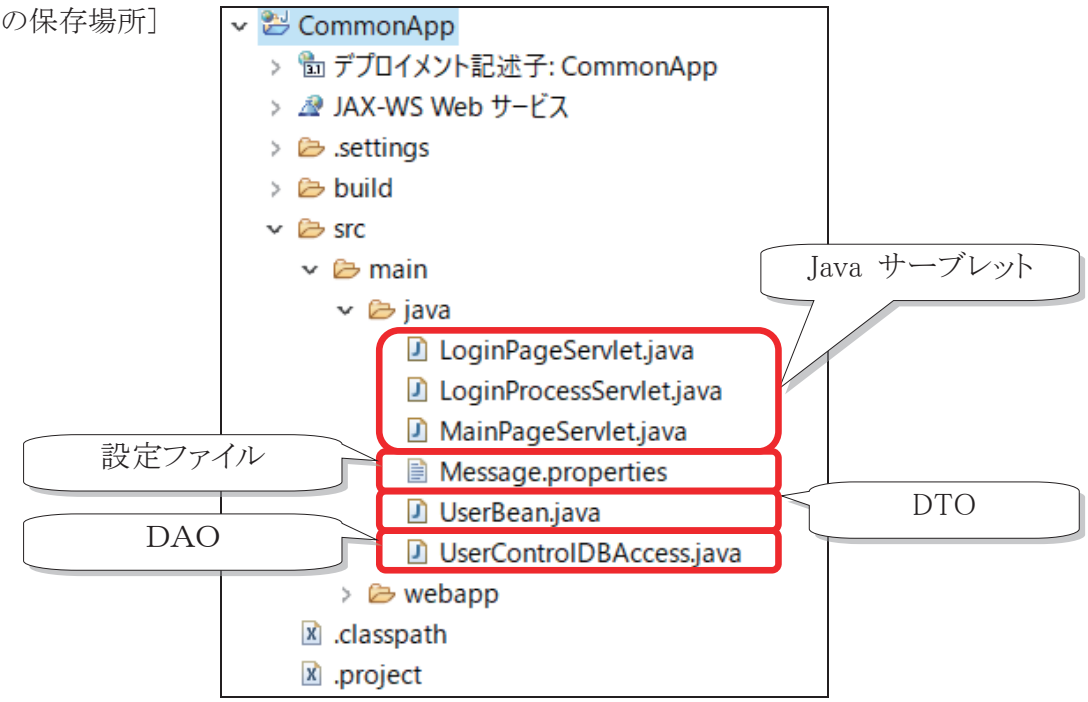
```
23          out.println("<html>");
24          out.println("<head><title>ログインページ</title></head>");
25          out.println("<body>");
26          out.println("<form action='LoginPage' method='POST'>");
27          out.println("<p>");
28          out.println("ID");
29          out.println("<input type='text' size='50' name='id' />");
30          out.println("</p>");
31          out.println("<p>");
32          out.println("PASS");
33          out.println(
34              "<input type='password' size='50' name='pass' />");
35          out.println("</p>");
36          out.println("<input type='submit' value='ログイン' />");
37          String msg = request.getParameter("status");
38          if(msg != null && msg.equals("failure") == true)
39          {
40             out.println(
41                 "ログインできません。正しいID／PASSを入力してください。");
42             out.println(pr.getProperty("fail"));
43          }
44          :
```

・16～21行目は，Propertiesクラスのインスタンスを作成し，プロパティファイルからキーと値を一覧で取得します。文字コードは“UTF-8”を指定します。

・42行目は，プロパティファイルから取得したキーと値の一覧から，キー名“fail”に対応する値を取得して表示します。

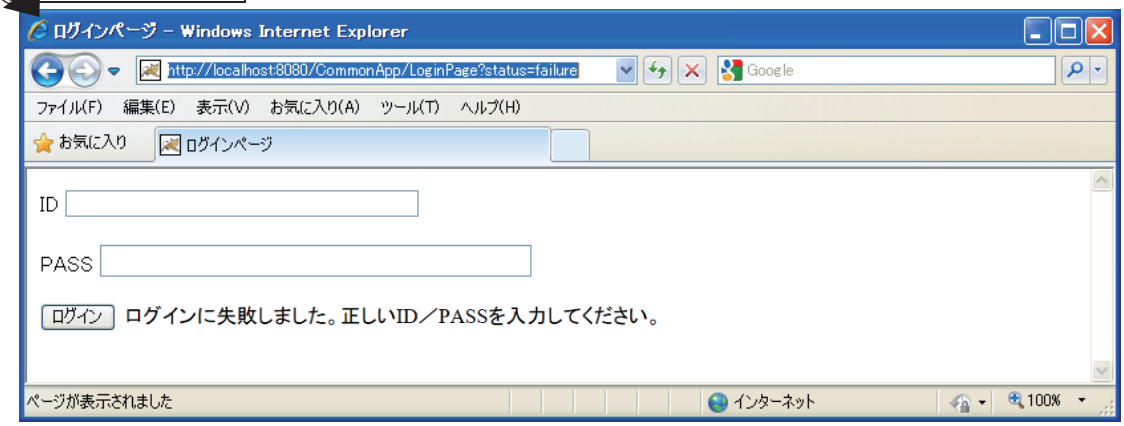
・正しく実行できたら，Message.propertiesファイルのメッセージを書き換えて実行し，変更が簡単にできることを確認してみましょう。

［ファイルの保存場所］



※うまく実行できない場合「Message.properties」ファイルを右クリック「移動(V)…」を選択して一度javaフォルダ以外へ移動し，再び右クリック「移動(V)…」を選択してjavaフォルダへ移動し直すことで正しく実行できるようになります。

実行結果　(URL：http://localhost:8080/CommonApp/LoginPage)



利用者認証の失敗時

1-2-8　通信セッションとデータ共有

Webコンテナでは，通常，個々のWebブラウザを識別することはできません。プログラム1-8～1-10では，同じWebブラウザにログインページとメインページを表示していましたが，Webコンテナ側では同じWebブラウザとの通信であることを認識していません。これは，Webブラウザ-Webコンテナ間で利用される通信プロトコル“HTTP”の仕組みによるものです。

通信セッションとは，**特定のWebブラウザとWebコンテナの継続的な通信状態**のことです。Webブラウザが異なる場合は，通信セッションも異なります。また，通信セッションを一旦終了して再び開始した場合は，たとえWebブラウザが同じであっても新たな通信セッションになります。Webブラウザ-Webコンテナ間で通信セッションを利用すると，Webコンテナ側では同じWebブラウザとの通信であることを認識できます。

通信セッションは，Webブラウザ側にCookie（クッキー：有効期限が設定されたデータファイル）を保存することで実現します。Webブラウザ-Webコンテナ間で通信セッションを利用する手順は，次のとおりです。

〔手順〕

① Webブラウザは，Webコンテナにリクエストを送信する。
② Webコンテナは，Webブラウザを特定するための“セッションID”を生成して，それを含んだレスポンスをWebブラウザに送信する（通信セッションが確立する）。

（HTTP通信（１回目））

③ Webブラウザは，レスポンスから“セッションID”を取り出して，それをCookieとして保存する。
④ Webブラウザは，Cookie内から“セッションID”を取り出して，それを含んだリクエストをWebコンテナに送信する。
⑤ Webコンテナは，リクエストから“セッションID”を取り出して，それが有効か無効かを判断する。有効である場合は，Webブラウザを特定したことになる。
⑥ Webコンテナは，Webブラウザにレスポンスを送信する。

（HTTP通信（２回目～））

新しく開始した通信セッションや既存の通信セッションの取得は，doGet()メソッドおよびdoPost()メソッドの第１引数に記録された参照のインスタンス内のgetSession()メソッドを使用します。

表11　getSession()メソッドの仕様

項目	内容
メソッド名	getSession()
引数	create（boolean型，動作指定）
返却値	通信セッション　（HttpSession型）
修飾子	public
例外	なし
概要	通信セッションを返す。

また，getSession()メソッドの引数に指定するtrue／falseによって，このメソッドの動作が変わります。

表12　getSession()メソッドの動作の違い

	まだ通信セッションが確立されていない場合	既に通信セッションが確立されている場合
getSession(true)	新しく開始した通信セッションの参照を返す。	既存の通信セッションの参照を返す。
getSession(false)	nullリテラル値「null」を返す。	既存の通信セッションの参照を返す。

通信セッションを確立すると，Webコンテナ内の複数のJavaサーブレット間でデータを共有することができます。また，共有データの操作（登録／取得／削除）は，getSession()メソッドから返された参照のインスタンス内のsetAttribute()メソッド，getAttribute()メソッド，removeAttribute()メソッドを使用します。

表13　setAttribute()メソッドの仕様

項目	内容
メソッド名	setAttribute()
引数	［第１引数］name　（String型，属性名） ［第２引数］value　（Object型，属性値）
返却値	なし　（void型）
修飾子	public
例外	IllegalStateException
概要	通信セッションに属性名nameと属性値valueを登録する。既に同じ属性名が登録されている場合は，属性値valueを上書きする。

表14　getAttribute()メソッドの仕様

項目	内容
メソッド名	getAttribute()
引数	name　（String型，属性名）
返却値	属性値（Object型）
修飾子	Public
例外	IllegalStateException
概要	通信セッション内で属性名nameに対応する属性値を返す。属性名nameが存在しない場合は，nullリテラル値「null」を返す。

表15　removeAttribute()メソッドの仕様

項目	内容
メソッド名	removeAttribute()
引数	name　（String型，属性名）
返却値	なし　（void型）
修飾子	public
例外	IllegalStateException
概要	通信セッション内で属性名nameと，対応する属性値を削除する。

実際に，Javaサーブレットのプログラムを見てみましょう。

プログラム 1-12　（ファイル名：LoginProcessServlet.java）

```
15          :
16          UserControlDBAccess dao = new UserControlDBAccess();
17          UserBean bean = dao.findUserByIdAndPass(id, pass);
18          if(bean != null)
19          {
20              //新しい通信セッションを開始する
21              HttpSession session = request.getSession(true);
22              //通信セッションに属性名“user”と属性値beanを登録する
23              session.setAttribute("user", bean);
24              response.sendRedirect("MainPage");
25          }
26          else
27          {
28              response.sendRedirect("LoginPage?status=failure");
29          }
30      }
31  }
```

・21行目のgetSession()メソッドは，引数に指定された「true」によって，新しく開始した通信セッションの参照を返します。

・23行目のsetAttribute()メソッドは，引数に指定された文字列リテラル値“user”，変数beanに記録された参照のインスタンス（UserBeanクラス型）をそれぞれ属性名，属性値とし，これらを通信セッションに登録します。これに伴い，同じ通信セッションのJavaサーブレットは，共有データを利用することができます。

プログラム 1-13　（ファイル名：MainPageServlet.java）

```
12          :
13          PrintWriter out = response.getWriter();
14          //Message.propertiesファイルを読み込む
15          Properties pr = new Properties();
```

（次ページに続く）

```
16          Reader rd = new InputStreamReader(
17              MainPageServlet.class.getResourceAsStream(
18              "Message.properties"),"UTF-8");
19          pr.load(rd);
20          rd.close();
21          out.println("<html>");
22          out.println("<head><title>メインページ</title></head>");
23          out.println("<body>");
24          out.println("<p>○○○さん, こんにちは。</p>");
25          //既存の通信セッションを取得する
26          HttpSession session = request.getSession(false);
27          if(session != null)
28          {
29              UserBean bean
30                  = (UserBean)session.getAttribute("user");
31              if(bean != null)
32              {
33                  out.println("<p>" + bean.getName() +
34                      pr.getProperty("register") + "</p>");
35              }
36              else
37              {
38                  out.println("<p>" +
39                      pr.getProperty("guest") + "</p>");
40              }
41          }
42          else
43          {
44              out.println("<p>" + pr.getProperty("guest") + "</p>");
45          }
46          out.println("</body>");
47          out.println("</html>");
48          :
```

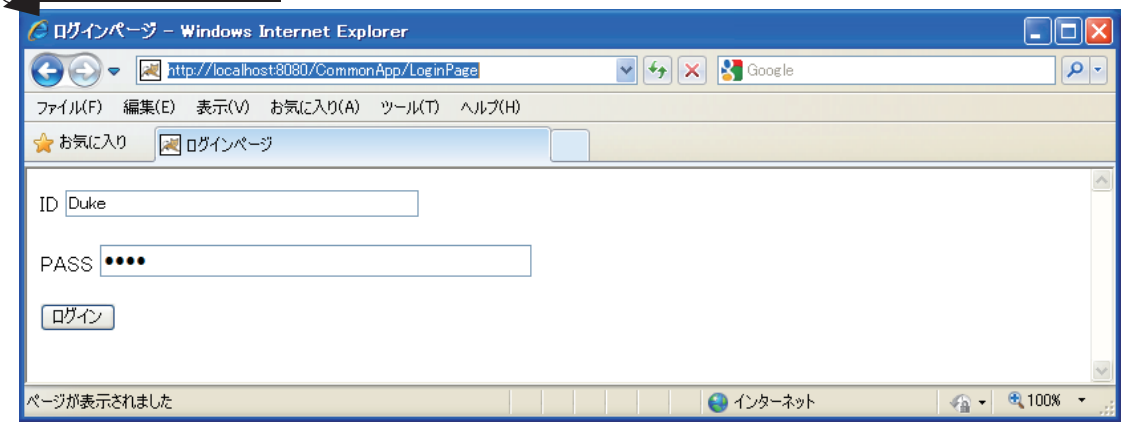
・26行目のgetSession()メソッドは，引数に指定された「false」によって，既存の通信セッションの参照，またはnullリテラル値「null」を返します。
・30行目のgetAttribute()メソッドは，引数に指定された文字列リテラル値“user”を属性名とし，対応する属性値を通信セッションから取得して，その参照を返します。なお，属性名が存在しない場合は，nullリテラル値「null」を返します。
・34行目は，プロパティファイルから取得したキーと値の一覧から，キー名“register”に対応する値を取得して表示し，39行目と44行目は，取得したキーと値の一覧から，キー名“guest”に対応する値を取得して表示します。

プログラム　（ファイル名：Message.properties）

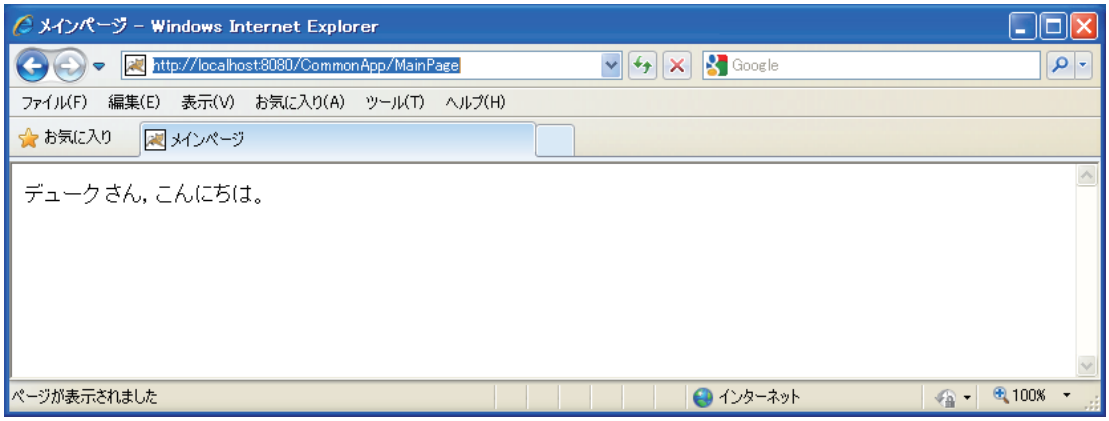
```
fail=ログインに失敗しました。正しいID／PASSを入力してください。
register=さん、こんにちは。
guest=ゲストさん、こんにちは。
```

実行結果①

(URL：http://localhost:8080/CommonApp/LoginPage)

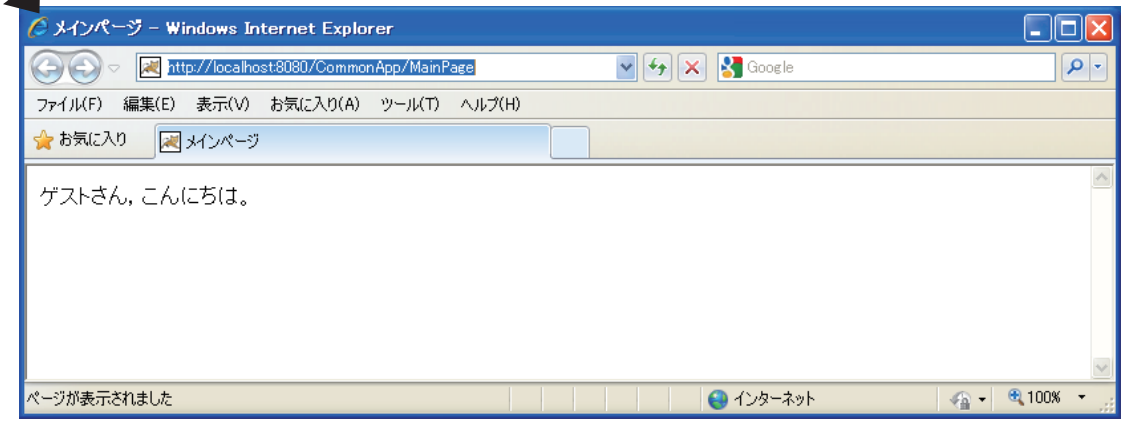


利用者認証の成功時



実行結果②

(URL：http://localhost:8080/CommonApp/MainPage)



※実行結果は，すべてのWebブラウザを閉じてから確認する。

通信セッションは，Webブラウザが閉じられたり，セッションタイムアウト時間（標準設定で30分間）が経過したりすると，自動的に終了します。また，Javaサーブレットから通信セッションを終了させることができます。通信セッションの終了は，getSession()メソッドから返された参照のインスタンス内のinvalidate()メソッドを使用します。

表16　invalidate()メソッドの仕様

項目	内容
メソッド名	invalidate()
引数	なし
返却値	なし　(void型)
修飾子	public
例外	IllegalStateException
概要	通信セッションを終了させる。

実際に，Javaサーブレットのプログラムを見てみましょう。

プログラム 1-14　（ファイル名：LogoutProcessServlet.java　URL：/LogoutProcess）

```
1  import  javax.servlet.*;
2  import  javax.servlet.annotation.*;
3  import  javax.servlet.http.*;
4  import  java.io.*;
5
6  @WebServlet("/LogoutProcess")
7  public  class  LogoutProcessServlet  extends  HttpServlet
8  {
9      protected  void  doGet(
10         HttpServletRequest  request,
11         HttpServletResponse  response)
12         throws  ServletException, IOException
13     {
14         //既存の通信セッションを取得する
15         HttpSession session = request.getSession(false);
16         if(session != null)
```

（次ページに続く）

```
17          {
18              //通信セッションを終了する
19              session.invalidate();
20          }
21          response.sendRedirect("LoginPage");
22      }
23  }
```

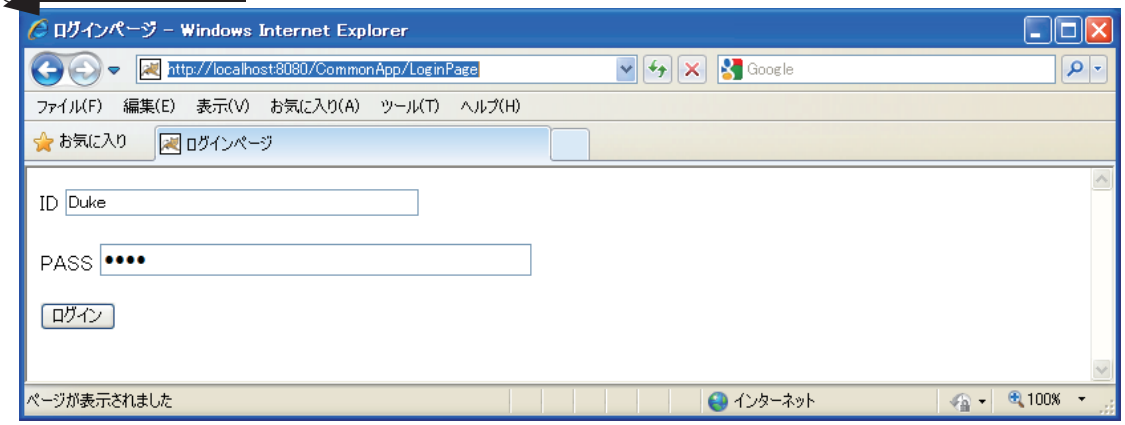
・19行目のinvalidate()メソッドは，通信セッションを終了させます。

プログラム 1-15　（ファイル名：MainPageServlet.java）

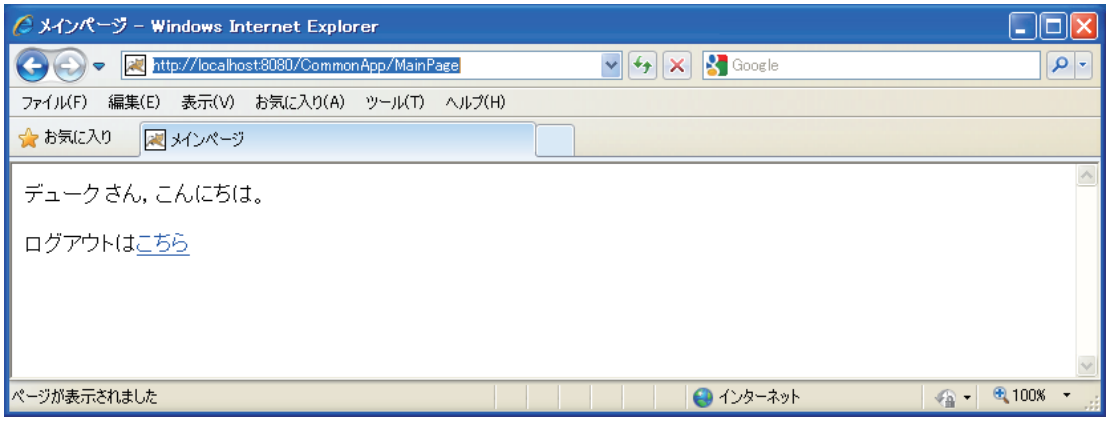
```
17          :
18          //既存の通信セッションを取得する
19          HttpSession session = request.getSession(false);
20          if(session != null)
21          {
22              UserBean bean
23                  = (UserBean)session.getAttribute("user");
24              if(bean != null)
25              {
26                  out.println("<p>" + bean.getName() +
27                      pr.getProperty("register") + "</p>");
28              }
29              else
30              {
31                  out.println("<p>" +
32                      pr.getProperty("guest") + "</p>");
33              }
34          }
35          else
36          {
37              out.println("<p>" + pr.getProperty("guest") + "</p>");
38          }
39          out.println("ログアウトは<a href='LogoutProcess'>こちら</a>");
```

```
40          out.println("</body>");
41          out.println("</html>");
42           :
```

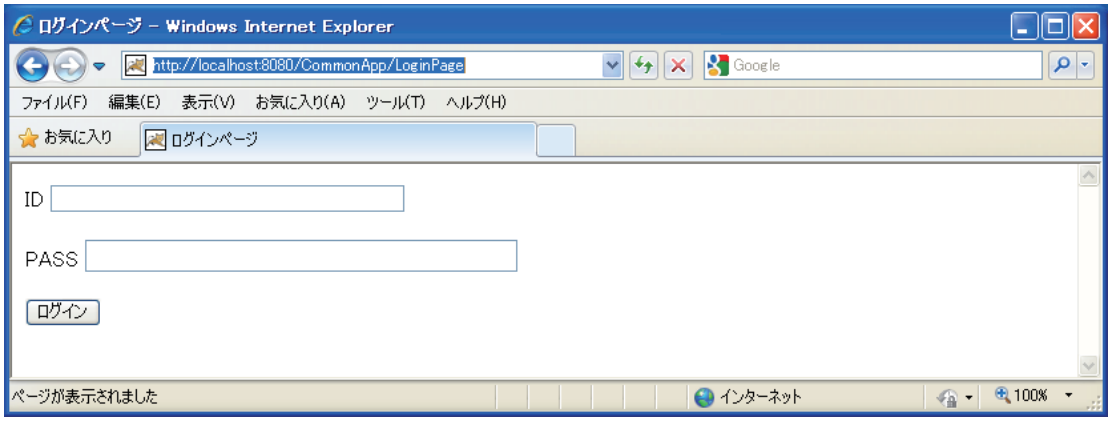
実行結果　(URL：http://localhost:8080/CommonApp/LoginPage)



利用者認証の成功時



ログアウト時



練習 1-8　レベル ☆☆☆

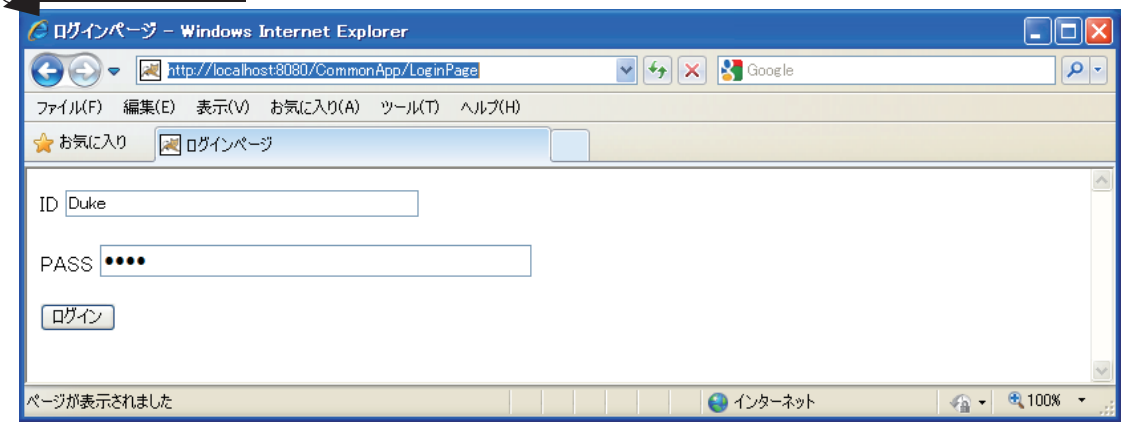
プログラム1-12において，通信セッションのセッションタイムアウト時間を５分（300秒）間に設定しなさい。なお，セッションタイムアウト時間の設定には，getSession()メソッドから返された参照のインスタンス内のsetMaxInactiveInterval()メソッドを使用すること（インストールしたAPIドキュメントで，setMaxInactiveInterval()メソッドの仕様を調べること)。

ヒント　（ファイル名：LoginProcessServlet.java）

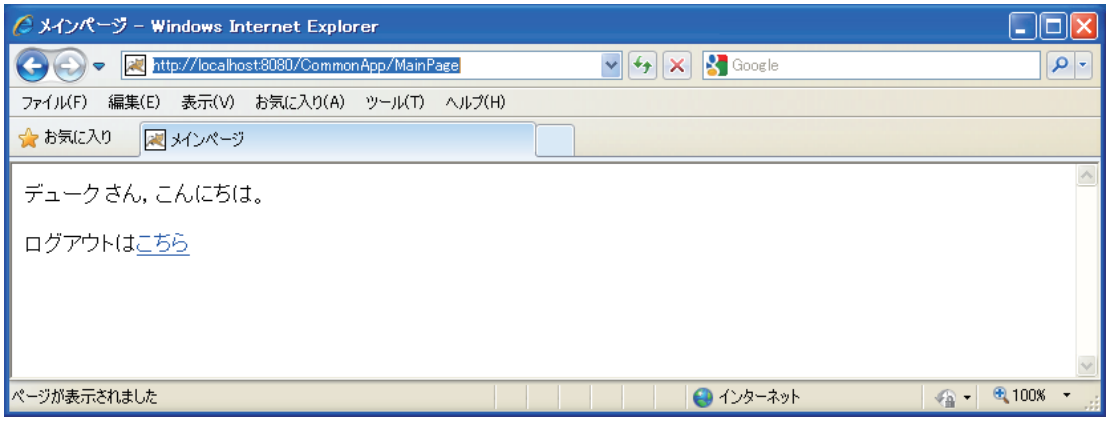
```
        :
        UserControlDBAccess dao = new UserControlDBAccess();
        UserBean bean = dao.findUserByIdAndPass(id, pass);
        if(bean != null)
        {
            //新しい通信セッションを開始する
            HttpSession session = request.getSession(true);
            //通信セッションのセッションタイムアウト時間を設定する
            :
            //通信セッションに属性名“user”と属性値beanを登録する
            session.setAttribute("user", bean);
            response.sendRedirect("MainPage");
        }
        else
        {
            response.sendRedirect("LoginPage?status=failure");
        }
    }
}
```

実行結果

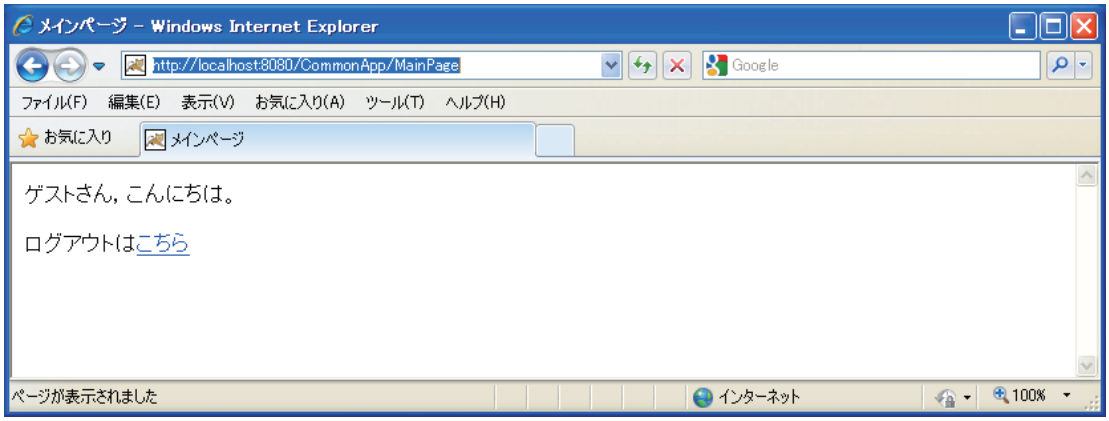
（URL：http://localhost:8080/CommonApp/LoginPage）



利用者認証の成功時



5分後のページ更新時



※動作確認が終了した後は，セッションタイムアウト時間の設定（点線枠内）をプログラムから削除しておくこと。

1-3　JSP基礎

1-3-1　JSPの仕組み

JSPは，HTML文書内にJavaソースコードを埋め込んで記述します。このとき，プログラマによるコンパイル作業は不要です。Webコンテナでは，Webブラウザから初めてリクエストを受け取ると，JSPをJavaサーブレット（Javaソースコード）に自動変換して，Javaコンパイラによって Javaサーブレット（Javaバイトコード）に自動変換して，その実行結果をレスポンスとしてWebコンテナに返します。なお，２回目以降のリクエストでは，Javaサーブレットへの自動変換は行われません。

[Apache Tomcatの場合]
変換後のJavaサーブレットは
「[Eclipseインストールフォルダ]¥Tomcat8¥work¥Catalina¥localhost」
フォルダ内に保存されます。

JSPの主な構造は，次のとおりです。

```
<%@ page contentType="text/html;charset=UTF-8" %>          ①

<html>                                                     ②
<body>

<%                                                         ③
    out.print("ようこそJSP／サーブレットの世界へ！");
%>

</body>                                                    ②
</html>
```

①JSPに関する情報をWebコンテナに設定する。

pageディレクティブを使用して，レスポンスの文字コードを指定したり，無名パッケージ以外のパッケージに登録された他クラス（非Javaサーブレット）をインポートします。

②HTML文書を記述する。

HTMLやXHTMLのタグを使用して文書を記述します。また，標準アクション（使用頻度が高い機能［インクルード／フォワード，リクエストパラメータ関連の操作など］）のタグを記述することもできます。

③スクリプトを記述する。

スクリプトレットやスクリプト式を使用してJavaソースコードを記述します。

・スクリプトレット

計算式，制御文，インスタンス生成，メソッド呼出しなどのJavaソースコードを「<%」と「%>」で囲みます。

・スクリプト式

リクエストに出力したい変数，計算式やメソッド呼出しなどのJavaソースコードを「<%=」と「%>」で囲みます。

実際に，JSPのプログラムを見てみましょう。プログラム1-16では，プログラム1-11と同じログインページを出力しています。JSPのプログラムは動的Webプロジェクト「CommonApp2」を作成し，記述してください。

プログラム 1-16　（ファイル名：LoginPage.jsp）

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2
3  <html>
4  <head><title>ログインページ</title></head>
5  <body>
6  <form action='LoginProcess' method='POST'>
7  <p>
8  ID
9  <input type='text' size='50' name='id' />
10 </p>
11 <p>
12 PASS
13 <input type='password' size='50' name='pass' />
14 </p>
15 <input type='submit' value='ログイン' />
```

（次ページに続く）

```
16
17  <%
18      String msg = request.getParameter("status");
19      if(msg != null && msg.equals("failure") == true)
20      {
21  %>
22
23  ログインできません。正しいID／PASSを入力してください。
24
25  <%
26      }
27  %>
28
29  </form>
30  </body>
31  </html>
```

JSPは，そのプログラムだけでは実行できません。次のような“コンテキスト記述子”を作成／更新します。

※Eclipse使用時は，Server.xmlの記述は，Tomcatサーバにプロジェクトを登録すると自動的に更新されます。

プログラム 1-17　（ファイル名：Server.xml）

```
124                 :
125                 <Context docBase="CommonAppFolder2" path="/CommonApp2"
126                     reloadable="true" />
127             </Host>
128         </Engine>
129     </Service>
130 </Server>
```

なお，プログラム1-16で作成したJSPは，次の場所(webappフォルダ直下)に保存します。

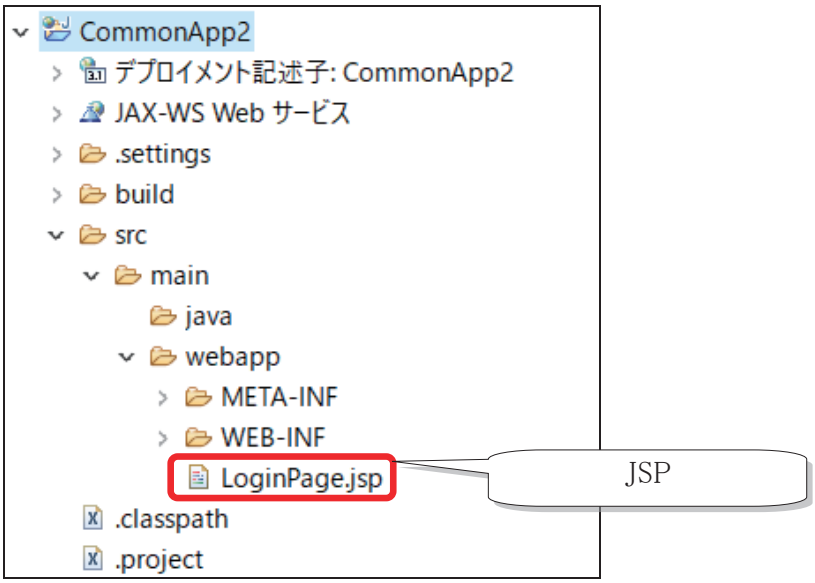
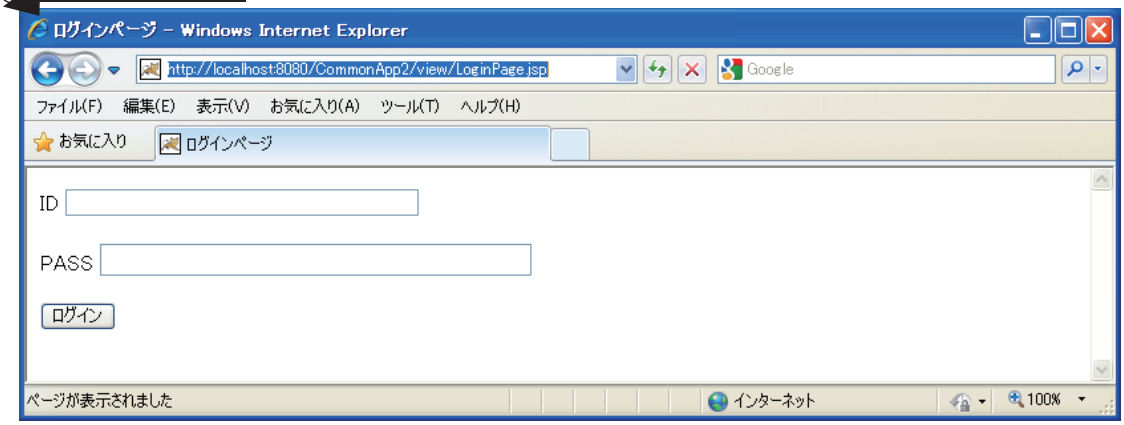


図１１　JSPの保存場所

実行結果　(URL：http://localhost:8080/CommonApp2/LoginPage.jsp)



※URL：http://localhost:8080/CommonApp2/LoginPage.jsp?status=failureも確認すること。

練習 1-9　レベル ☆

学校の期末試験の英語，国語，数学，理科，社会の得点を表示するJSPのプログラムを作成しなさい。なお，各得点はint型の二次元配列scoreに記録されているものとします。

二次元配列

score	（英語）	（国語）	（数学）	（理科）	（社会）
（１人目）	67	72	54	78	83
（２人目）	55	84	51	48	91
（３人目）	70	57	62	66	96

ヒント　（ファイル名：ExamScore.jsp）

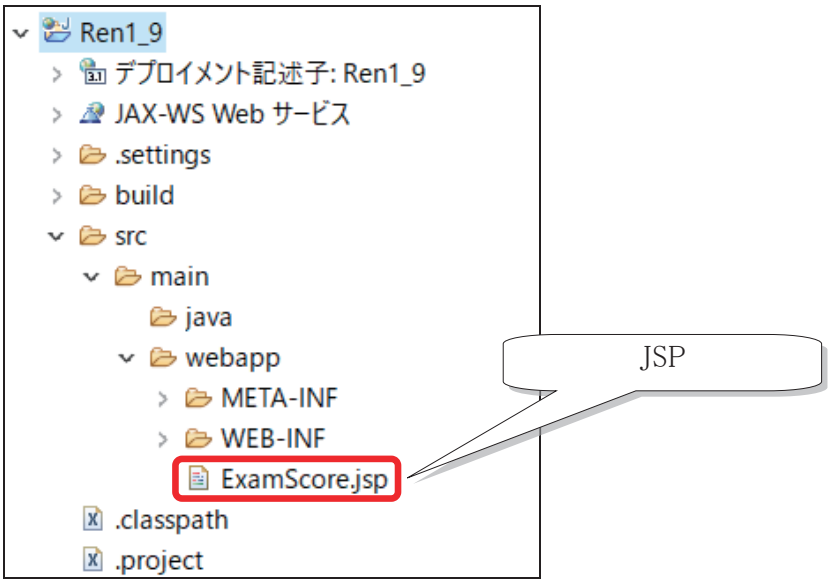
```
<%@ page contentType="text/html;charset=UTF-8" %>

<%
    //学校の期末試験の英語，国語，数学，理科，社会の得点
    int[][] score = {{67, 72, 54, 78, 83},
                     {55, 84, 51, 48, 91},
                     {70, 57, 62, 66, 96}};
%>

<html>
<head><title>期末試験の得点</title></head>
<body>
<div align='center'>
<table border='1'>
<tr><th></th>
    <th>英語</th>
    <th>国語</th>
    <th>数学</th>
    <th>理科</th>
    <th>社会</th></tr>
```

```
<%
    int i;              //行の添字
    int j;              //列の添字
    for(i = 0; i < score.length; i++)
    {
%>
<tr><th><%=(i + 1) %>人目</th>
<%
        for(j = 0; j < score[i].length; j++)
        {
%>
 :
<%
        }
%>
</tr>
<%
    }
%>

</table>
</div>
</body>
</html>
```

プログラム　（ファイル名：Server.xml）

```
124                  :
125                  <Context docBase="Ren1_9" path="/Ren1_9"
126                      reloadable="true" />
127              </Host>
128          </Engine>
129      </Service>
130  </Server>
```

［ファイルの保存場所］



実行結果　(URL：http://localhost:8080/Ren1_9/ExamScore.jsp)

	英語	国語	数学	理科	社会
1人目	67	72	54	78	83
2人目	55	84	51	48	91
3人目	70	57	62	66	96

練習 1-10　レベル ☆

受注した東京都の顧客情報を表示するJSPのプログラムを作成しなさい。なお，各顧客情報はStringクラス型の二次元配列infoに記録されているものとします。

二次元配列

info

（受注ID）	（受注日）	（顧客ID）	（顧客名）
0003	2005/12/22	005	小島　千里
0004	2006/01/11	007	太田　誠人
0005	2006/01/18	002	金子　実
0007	2006/02/01	010	佐藤　美佑
0008	2006/02/10	014	小林　和佳子
0012	2006/04/11	019	安藤　由紀子
0016	2006/05/17	012	宮　正澄
0018	2006/06/05	007	太田　誠人
0023	2006/07/11	010	佐藤　美佑
0029	2006/08/11	010	佐藤　美佑
0031	2006/08/30	012	宮　正澄
0034	2006/10/04	019	安藤　由紀子

ヒント　（ファイル名：TokyoCustomerInfo.jsp）

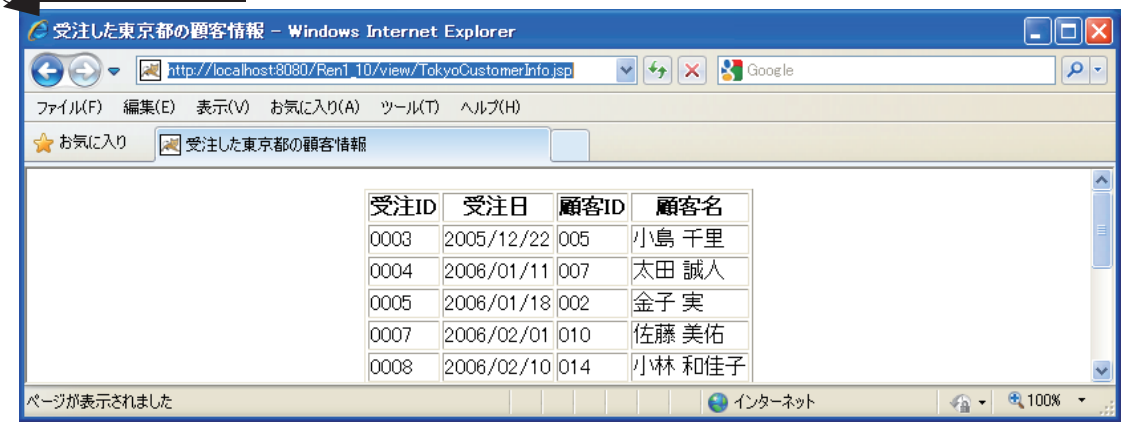
```
<%@ page contentType="text/html;charset=UTF-8" %>

<%
    //受注した東京都の顧客情報
    String[][] info = {
        {"0003", "2005/12/22", "005", "小島　千里"},
        {"0004", "2006/01/11", "007", "太田　誠人"},
        {"0005", "2006/01/18", "002", "金子　実"},
        {"0007", "2006/02/01", "010", "佐藤　美佑"},
        {"0008", "2006/02/10", "014", "小林　和佳子"},
        {"0012", "2006/04/11", "019", "安藤　由紀子"},
        {"0016", "2006/05/17", "012", "宮　正澄"},
        {"0018", "2006/06/05", "007", "太田　誠人"},
        {"0023", "2006/07/11", "010", "佐藤　美佑"},
        {"0029", "2006/08/11", "010", "佐藤　美佑"},
        {"0031", "2006/08/30", "012", "宮　正澄"},
```

```
        {"0034", "2006/10/04", "019", "安藤　由紀子"}};
%>

<html>
<head><title>受注した東京都の顧客情報</title></head>
<body>
<div align='center'>
<table border='1'>
<tr>  <th>受注ID</th><th>受注日</th>
      <th>顧客ID</th><th>顧客名</th></tr>
<%
    :
%>

</table>
</div>
</body>
```

プログラム　（ファイル名：Server.xml）

```
124                 :
125                 <Context docBase="Ren1_10" path="/Ren1_10"
126                     reloadable="true" />
127             </Host>
128         </Engine>
129     </Service>
130 </Server>
```

実行結果　(URL：http://localhost:8080/Ren1_10/TokyoCustomerInfo.jsp)



受注ID	受注日	顧客ID	顧客名
0003	2005/12/22	005	小島 千里
0004	2006/01/11	007	太田 誠人
0005	2006/01/18	002	金子 実
0007	2006/02/01	010	佐藤 美佑
0008	2006/02/10	014	小林 和佳子

※実行結果はスクロールして確認する（表のレコード数：12）。

練習 1-11　レベル ☆☆

JDBCプログラミングのプログラム1-3で作成したProductBeanクラス，練習1-9で作成したCustomerBeanクラス，練習1-10で作成したDivisionBeanクラスとOrderControlDBAccessクラスを**一部変更**して使用し，DB（受注管理DB）の商品マスタ表から全レコードを取得して表示するJSPのプログラムを作成しなさい。

プログラム　（ファイル名：ProductBean.java）

```
1  package model;
2
3  import java.io.Serializable;
4
5  //ProductBeanクラス（DTO）の定義
6  public class ProductBean implements Serializable
7  {
8      private String id;                    //商品ID
9      private String name;                  //商品名
10     private int price;                    //単価
11      :
```

プログラム　（ファイル名：CustomerBean.java）

```
1  package model;
2
3  import java.io.Serializable;
4
5  //CustomerBeanクラス（DTO）の定義
6  public class CustomerBean implements Serializable
7  {
8      private String id;                    //顧客ID
9      private String name;                  //顧客名
10      :
```

プログラム　（ファイル名：DivisionBean.java）

```
1  package  model;
2
3  import  java.io.Serializable;
4
5  //DivisionBeanクラス（DTO）の定義
6  public  class  DivisionBean  implements  Serializable
7  {
8      private  String  name;                    //都道府県名
9      private  int  price;                      //売上合計
10     :
```

プログラム　（ファイル名：OrderControlDBAccess.java）

```
1  package  model;
2
3  import  java.util.ArrayList;
4  import  java.sql.*;
5
6  //OrderControlDBAccess クラス（DAO）の定義
7  public  class  OrderControlDBAccess
8  {
9      :
```

ヒント　（ファイル名：ProductList.jsp）

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ page import="model.OrderControlDBAccess, model.ProductBean,
                 java.util.*" %>
<%
    //DAO を生成する。
    OrderControlDBAccess dao = new OrderControlDBAccess();
    //DAO から商品マスタ表の全レコードを取得する。
    ArrayList<ProductBean> list = dao.findAllProducts();
%>
```

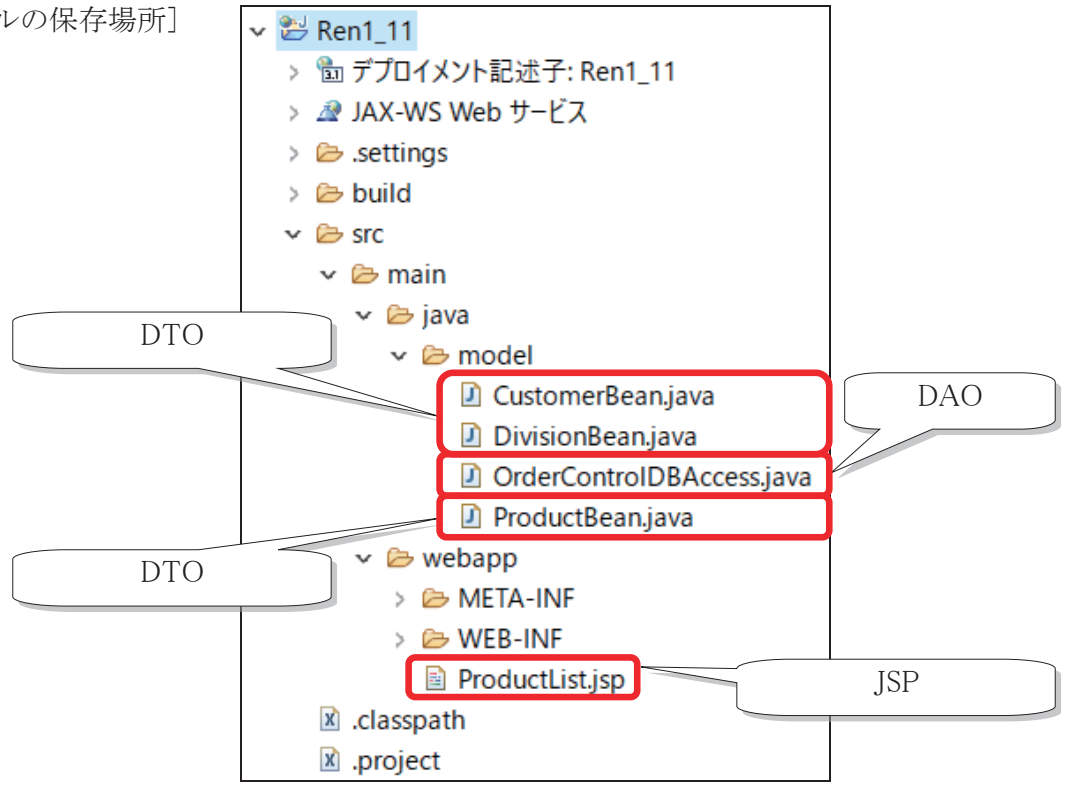
（次ページに続く）

```
<html>
<head><title>商品マスタ表</title></head>
<body>
<div align='center'>
<table border='1'>
<caption>商品マスタ表</caption>
<tr><th>商品ID</th><th>商品名</th><th>単価</th></tr>

<%
    for(ProductBean bean : list)
    {
%>
 :
<%
    }
%>

</table>
</div>
</body>
</html>
```

プログラム　（ファイル名：Server.xml）

```
                :
                <Context docBase="Ren1_11" path="/Ren1_11"
                    reloadable="true" />
            </Host>
        </Engine>
    </Service>
</Server>
```

［ファイルの保存場所］


Ren1_11
デプロイメント記述子: Ren1_11
JAX-WS Web サービス
.settings
build
src
main
java
model
CustomerBean.java
DivisionBean.java
OrderControlDBAccess.java
ProductBean.java
webapp
META-INF
WEB-INF
ProductList.jsp
.classpath
.project
DTO
DTO
DAO
JSP

実行結果　(URL：http://localhost:8080/Ren1_11/ProductList.jsp)

商品マスタ表

商品ID	商品名	単価
F-001	ストロベリー	¥1580
F-002	ピーチ	¥1480
F-003	マンゴ	¥1480
F-004	トロピカル	¥1300
F-005	オレンジ	¥1480
H-001	ローズヒップ	¥1800
H-002	カモミール	¥1600
H-003	レモンバーム	¥1700
H-004	ペパーミント	¥1600
H-005	ハイビスカスフラワー	¥1800
O-001	アイリッシュモルト	¥1400
T-001	ダージリン	¥2300
T-002	アッサム	¥1600
T-003	セイロン	¥1500
T-004	キャラメル	¥1400
T-005	アールグレイ	¥1500
T-006	バニラチャイ	¥1780
T-007	ドゥドロップス	¥1600

練習 1-12　レベル ☆☆

JDBCプログラミングのプログラム1-3で作成したProductBeanクラス，練習1-9で作成したCustomerBeanクラス，練習1-10で作成したDivisionBeanクラスとOrderControlDBAccessクラスを**一部変更**して使用し，DB（受注管理DB）の顧客マスタ表から全レコードを取得して表示するJSPのプログラムを作成しなさい。（ファイル名：CustomerList.jsp）

プログラム　（ファイル名：Server.xml）

```
124                  :
125                  <Context docBase="Ren1_12" path="/Ren1_12"
126                      reloadable="true" />
127              </Host>
128          </Engine>
129      </Service>
130  </Server>
```

実行結果　（URL：http://localhost:8080/Ren1_12/CustomerList.jsp）

※実行結果はスクロールして確認する（表のレコード数：20）。

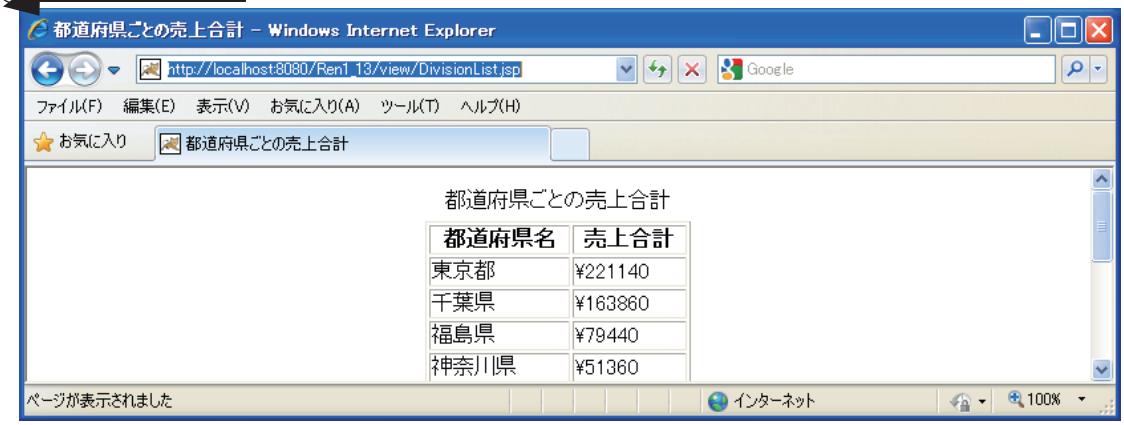
練習 1-13　レベル ☆☆

JDBCプログラミングのプログラム1-3で作成したProductBeanクラス，練習1-9で作成したCustomerBeanクラス，練習1-10で作成したDivisionBeanクラスとOrderControlDBAccessクラスを**一部変更**して使用し，DB（受注管理DB）の顧客マスタ表，商品マスタ表，受注表，受注明細表を結合した表から，都道府県ごとの売上合計を取得して表示するJSPのプログラムを作成しなさい。（ファイル名：DivisionList.jsp）

プログラム　（ファイル名：Server.xml）

```
124                 :
125                 <Context docBase="Ren1_13" path="/Ren1_13"
126                     reloadable="true" />
127             </Host>
128         </Engine>
129     </Service>
130 </Server>
```

実行結果　(URL：http://localhost:8080/Ren1_13/DivisionList.jsp)



※実行結果はスクロールして確認する（表のレコード数：12）。

1-3-2　ディレクティブ

JSPにはディレクティブと呼ばれるJSPの設定を記述する箇所があります。ディレクティブは「<%@」で始まり「%>」で終わります。この中にディレクティブ名と属性の内容を指定します。

ディレクティブには指示する内容によって「pageディレクティブ」「includeディレクティブ」「taglibディレクティブ」の3種類があります。

表17　ディレクティブ

ディレクティブ名	概要
page	JSPのページに関する指示・指定を行う
include	includeディレクティブを記述した位置に,他のJSPやHTMLファイルを読み込むことができる
taglib	定期的に使われる処理を記述したもの（タグ）をまとめておくことができる

ここではディレクティブの中でも特に重要なpageディレクティブについて見てみましょう。

練習1-11の「ProductList.jsp」を見ると,先頭にpageディレクティブが記述されています。

プログラム〔一部再掲〕　（ファイル名：ProductList.jsp）

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <%@ page import="model.OrderControlDBAccess, model.ProductBean,
3                   java.util.*" %>
         :
```

1行目の「**page contentType="text/html;charset=UTF-8"**」はデータ型と文字コードの指定のための項目で，データ型「HTMLファイル」文字コード「UTF-8」を指定しています。

2～3行目の「**page import="model.OrderControlDBAccess, model.ProductBean, java.util.*"**」はJSPでJavaクラスをimportしています。複数のクラスをimportする場合は「,」で区切って列挙します。

pageディレクティブの代表的な項目は次の通りです。

表18　pageディレクティブの代表的な項目

項目名	概要
contentType	JSPのデータ型と文字コードを指定する
import	JavaクラスをJSPにimportする
session	通信セッションの有効/無効を指定する。「true」で通信セッション有効,「false」で通信セッション無効
info	JSPに関する任意の情報を記述する
errorPage	エラーが発生した場合に表示するページのURLを指定する。指定がない場合はサーバのデフォルトエラーページが表示される
isErrorPage	そのJSPがエラー表示用のページかどうかを指定する。上記のerrorPageでエラー表示先として指定された場合は「true」を設定しなければならない
isElIgnored	「1-3-4　EL式」で解説するEL式を使用できるかどうかを指定する。EL式を使用できないようにする場合は「true」を指定する

1-3-3　暗黙オブジェクト

JSPのスクリプトを記述する部分では，Webブラウザから受け取ったリクエストに対応するインスタンス，Webブラウザに返すレスポンスに対応するインスタンス，通信セッションに対応するインスタンスなどを，あらかじめ宣言することなく利用できます。これらのインスタンスは暗黙オブジェクトと呼ばれ，９種類が用意されています。代表的な暗黙オブジェクトは，次のとおりです。

表19　代表的な暗黙オブジェクト

暗黙オブジェクト	概要
request	リクエストに対応するインスタンスの参照（HttpServletRequest型）
responce	レスポンスに対応するインスタンスの参照（HttpServletResponse型）
out	レスポンスへの出力用インスタンスの参照（JspWriter型）
session	通信セッションに対応するインスタンスの参照（HttpSession型）

※暗黙オブジェクト「request」は，request.getParameter()メソッドの呼出しなどで使用します。
（使用例．86ページ「プログラム1-16」18行目）

※暗黙オブジェクト「out」は，out.print()メソッドやout.println()メソッドの呼出しなどで使用します。
（使用例．84ページ「JSPの主な構造（図）」中段）

※暗黙オブジェクト「session」は，session.getAttribute()メソッドの呼出しなどで使用します。
（使用例．109ページ「プログラム1-18」11行目）

実際に，JSPのプログラムを見てみましょう。

プログラム 1-18　（ファイル名：MainPage.jsp）

```
1 <%@ page contentType="text/html;charset=UTF-8" %>
2 <%@ page import="model.UserBean, java.util.*" %>
```

```
3
4  <html>
5  <head><title>メインページ</title></head>
6  <body>
7
8  <%
9      if(session != null)
10     {
11         UserBean bean = (UserBean)session.getAttribute("user");
12         if(bean != null)
13         {
14 %>
15 <p><%=bean.getName() %>さん，こんにちは。</p>
16 <%
17         }
18         else
19         {
20 %>
21 <p>ゲストさん，こんにちは。</p>
22 <%
23         }
24     }
25     else
26     {
27 %>
28 <p>ゲストさん，こんにちは。</p>
29 <%
30     }
31 %>
32
33 <p>ログアウトは<a href='LogoutProcess'>こちら</a></p>
34 </body>
35 </html>
```

・11行目のsession.getAttribute()メソッドは，暗黙オブジェクト「session」を使用しています。

なお，実行結果を確認するには，プログラム1-4，1-5，1-12，1-14，を一部修正して使用します。ページ遷移はパフォーマンスを向上させるためリダイレクションからフォワードに変更します。

プログラム 1-19　（ファイル名：UserBean.java）

```
1  package model;
2
3  import java.io.Serializable;
4
5  //UserBeanクラス（DTO）の定義
6  public class UserBean implements Serializable
7  {
8      private String id;                    //ID
9      private String name;                  //氏名
10     :
```

プログラム 1-20　（ファイル名：UserControlDBAccess.java）

```
1  package model;
2
3  import java.sql.*;
4
5  //UserControlDBAccessクラス（DAO）の定義
6  public class UserControlDBAccess
7  {
8      :
```

プログラム 1-21　（ファイル名：LoginProcessServlet.java）

```
1  package  model;
2
3  import  javax.servlet.*;
4  import  avax.servlet.annotation.*
5  import  javax.servlet.http.*;
6  import  java.io.*;
          :
16        //サーブレットコンテキストを取得する
17        ServletContext sc = getServletContext();
18        UserControlDBAccess dao = new UserControlDBAccess();
19        UserBean bean = dao.findUserByIdAndPass(id, pass);
20        if(bean != null)
21        {
22            //新しい通信セッションを開始する
23            HttpSession session = request.getSession(true);
24            //通信セッションに属性名“user”と属性値beanを登録する
25            session.setAttribute("user", bean);
26            response.sendRedirect("MainPage");
27            //リクエストディスパッチャを取得する
28            RequestDispatcher rd =
29              sc.getRequestDispatcher("/MainPage.jsp");
30            rd.forward(request, response);
31        }
32        else
33        {
34            response.sendRedirect("LoginPage?status=failure");
35            request.setAttribute("status", "failure");
36            //リクエストディスパッチャを取得する
37            RequestDispatcher rd =
38              sc.getRequestDispatcher("/LoginPage.jsp");
39            rd.forward(request, response);
40        }
41        :
```

プログラム 1-22　（ファイル名：LogoutProcessServlet.java）

```
10          :
11          //既存の通信セッションを取得する
12          HttpSession session = request.getSession(false);
13          if(session != null)
14          {
15              //通信セッションを終了する
16              session.invalidate();
17          }
18          response.sendRedirect("LoginPage");
19          //サーブレットコンテキストを取得する
20          ServletContext sc = getServletContext();
21          //リクエストディスパッチャを取得する
22          RequestDispatcher rd =
23            sc.getRequestDispatcher("/LoginPage.jsp");
24          rd.forward(request, response);
25      }
26  }
```

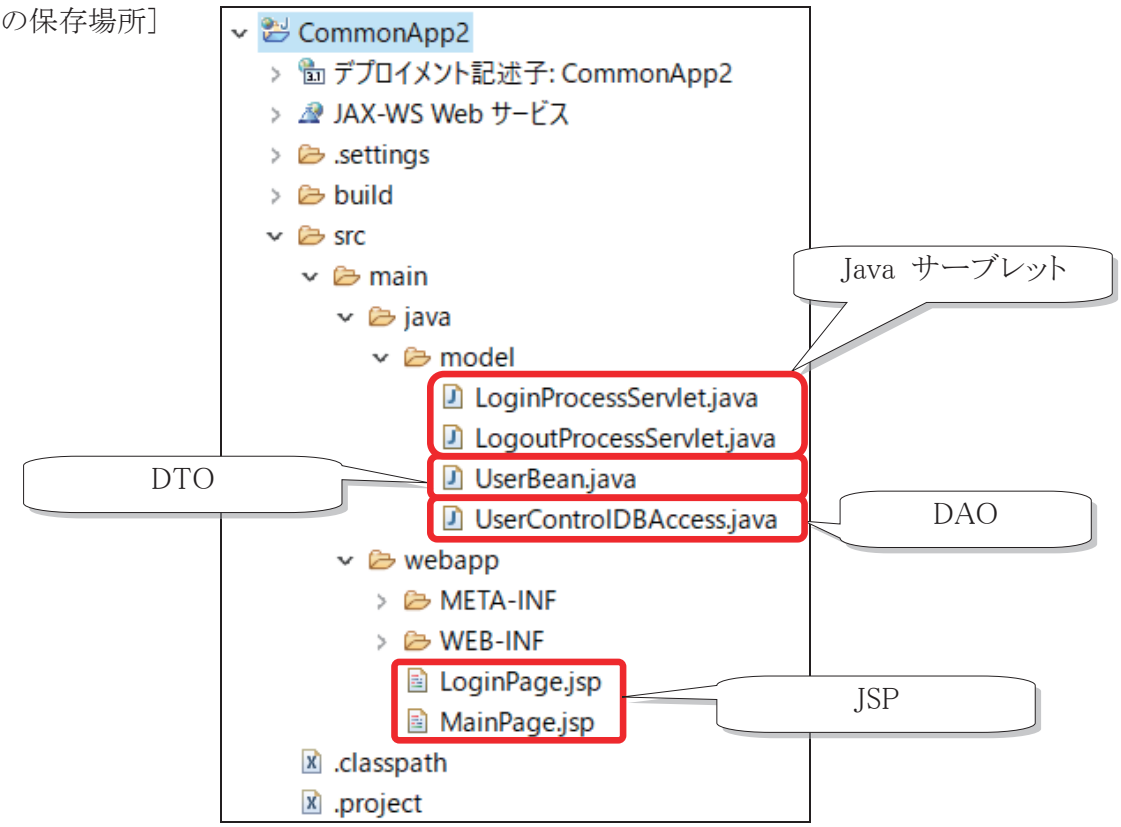
プログラム 1-23　（ファイル名：LoginPage.jsp）

```
14          :
15  <input type='submit' value='ログイン' />
16
17  <%
18          String msg = request.getParameter("status");
19          String msg = (String)request.getAttribute("status");
20          if(msg != null && msg.equals("failure") == true)
21          {
22  %>
23  ログインできません。正しいID／PASSを入力してください。
24  <%
25          :
```

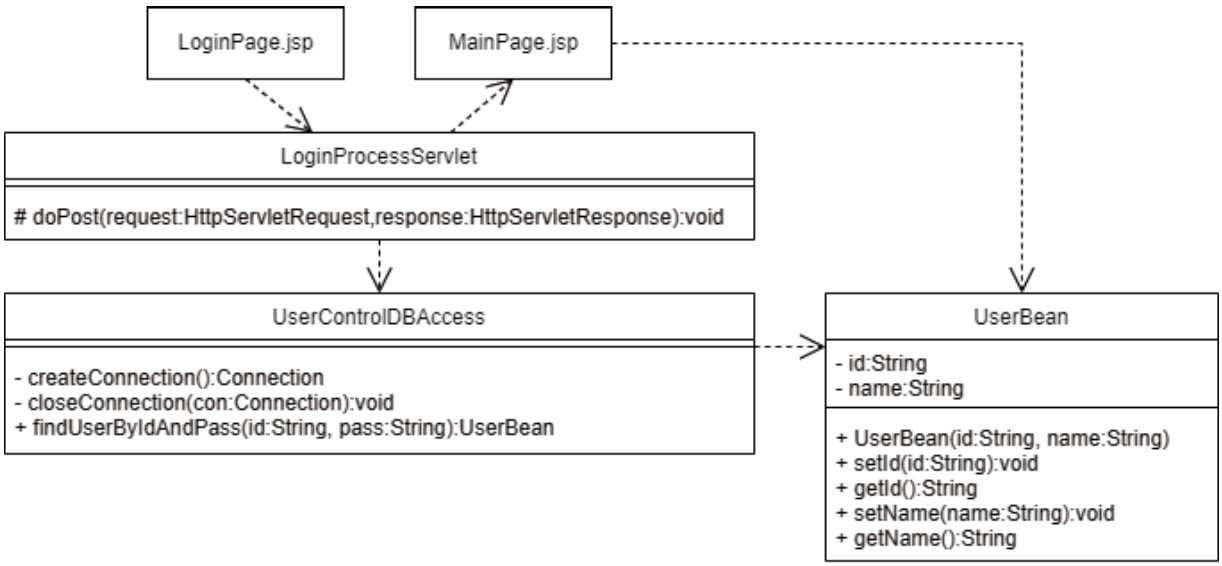
・フォワードではURLが変更されないため，「“URL”？“リクエストパラメータ名”＝“リクエストパラメータ値”」の方法でリクエストパラメータをページ間で受け渡しすることができません。よって，送信側のJavaサーブレット（プログラム1-21）35行目のrequest.setAttribute()メソッドで，リクエストパラメータ名“**status**”とリクエストパラメータ値“**failure**”を設定し，受信側のJSP（プログラム1-23）19行目のrequest.getAttribute()メソッドでリクエストパラメータを取得するよう変更します。

・今回の例ではパフォーマンスを向上させる目的でフォワードを使用しましたが，パフォーマンス向上よりもURLの変更を優先しなければならない場合はリダイレクションを使用します。

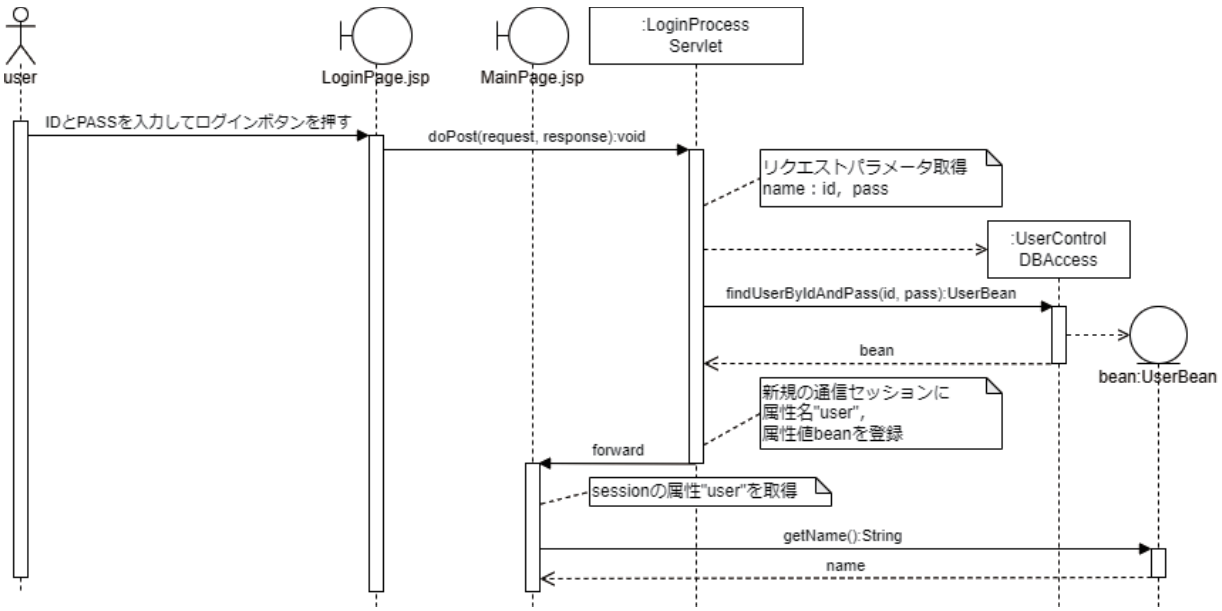
［ファイルの保存場所］



［クラス図（利用者認証の成功時）］

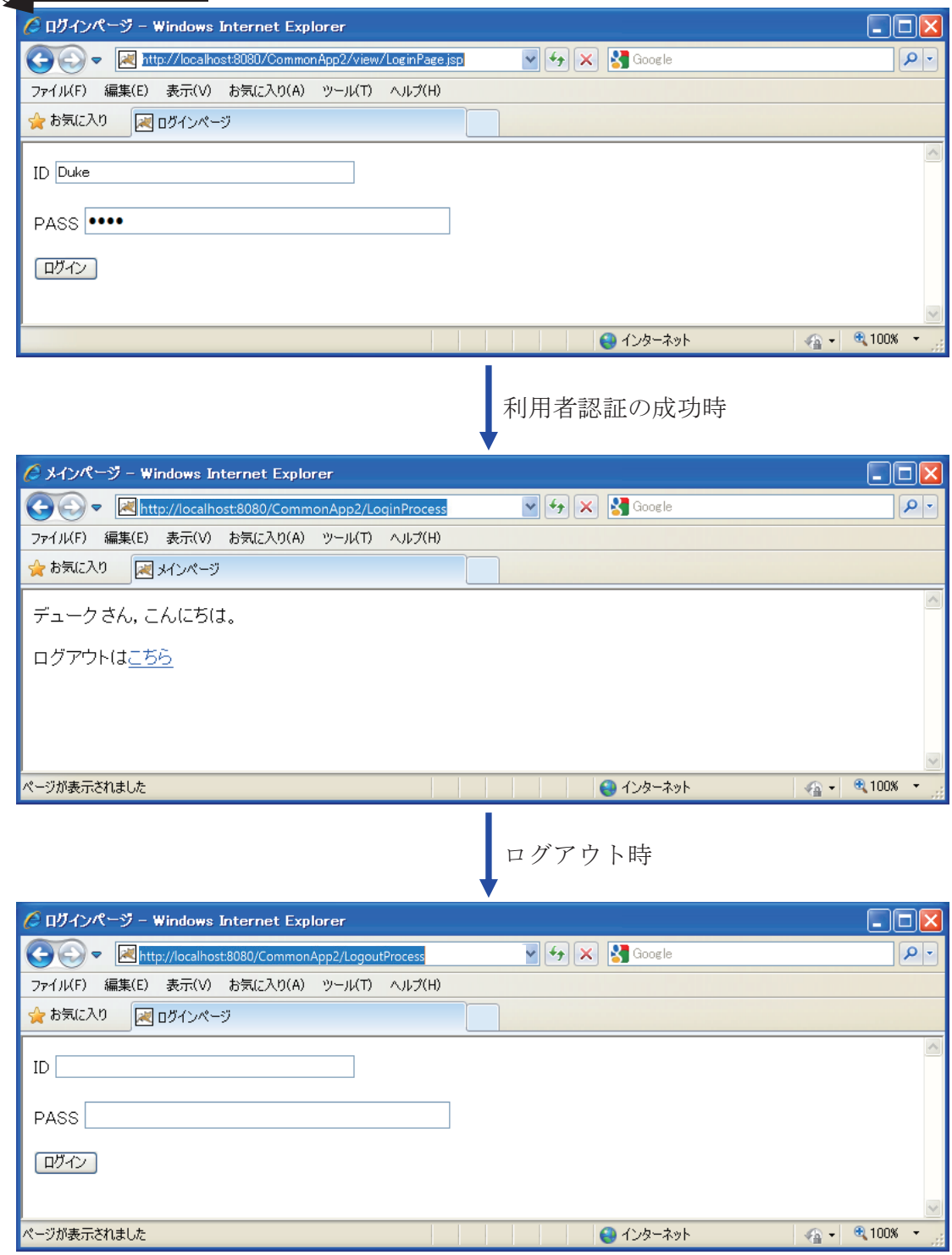


［シーケンス図（利用者認証の成功時）］



実行結果

(URL：http://localhost:8080/CommonApp2/LoginPage.jsp)



JSPのスクリプトを記述する部分では，Javaサーブレットで定義されている三つのスコープ「リクエストスコープ」，「セッションスコープ」，「アプリケーションスコープ」の他に，「ページスコープ」が利用できます。

表20　JSPで定義されている四つのスコープ

種類	概要
ページスコープ	・JSPのプログラム内を有効範囲とする。 ・暗黙オブジェクト「page」（HttpJspPageクラスのインスタンス）に“属性名”と“属性値”を格納する。
リクエストスコープ	・Webブラウザからの１回のリクエストを有効範囲とする。 ・暗黙オブジェクト「request」（HttpServletRequestクラスのインスタンス）に“属性名”と“属性値”を格納する。
セッションスコープ	・特定のWebブラウザとWebコンテナ間における継続的な通信セッションを有効範囲とする。 ・暗黙オブジェクト「session」（HttpSessionクラスのインスタンス）に“属性名”と“属性値”を格納する。
アプリケーションスコープ	・Webアプリケーション全体を有効範囲とする。 ・暗黙オブジェクト「application」（ServletContextクラスのインスタンス）に“属性名”と“属性値”を格納する。

※「ページスコープ」は，JSPプログラムの実行ごとにスコープを管理します。

※「リクエストスコープ」，「セッションスコープ」，「アプリケーションスコープ」を利用すると，JSPとJavaサーブレットのプログラム間で“属性値”をやり取りできます。ただし，「アプリケーションスコープ」はセキュリティの観点から，ほとんど利用されません。

1-3-4　EL式

EL式を利用することで，JSPにセッションに保存された情報や，サーブレットから受け渡された情報などをJSPに簡単に表示させることができます。

EL式（Expression Language）は式言語とも呼ばれています。EL式ではごく簡単な演算を行うこともできます。

EL式の記述は「${式}」という形式です。

実際に，JSPのプログラムを見てみましょう。

プログラム 1-24　（ファイル名：MainPage.jsp）

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <%@ page import="model.UserBean, java.util.*" %>
3
4  <html>
5  <head><title>メインページ</title></head>
6  <body>
7
8  <%
9      if(session != null)
10     {
11         UserBean bean = (UserBean)session.getAttribute("user");
12         if(bean != null)
13         {
14 %>
15 <p><%=bean.getName() %>さん，こんにちは。</p>
16 <%
17         }
18         else
19         {
20 %>
21 <p>ゲストさん，こんにちは。</p>
22 <%
23         }
```

```
24      }
25      else
26      {
27  %>
28  <p>ゲストさん，こんにちは。</p>
29  <%
30      }
31  %>
32  <p>${user.name}さん，こんにちは。</p>
33  <p>ログアウトは<a href='LogoutProcess'>こちら</a></p>
34  </body>
35  </html>
```

・32行目の${user.name}がEL式です。既存の属性名「user」の通信セッションに記憶された参照のインスタンス（userBean型）から「name」の値を取得し，表示します。

EL式を用いるとsession.getAttribute()メソッドを使わずにセッションに記憶されている値を表示できるため，通信セッションの有無をチェックする必要がありません（通信セッションが存在しない場合は「${user.name}」の箇所に何も表示されないだけで例外は発生しません)。

しかし，このままでは通信セッションが存在しない場合，値を取得することができないため「さん，こんにちは。」だけが表示されてしまいます。

そのため，EL式では簡単な演算や条件式が記述できるようになっており，セッションが存在しなかった場合の対処を行うことができます。

実際に，JSPのプログラムを見てみましょう。

プログラム 1-25　（ファイル名：MainPage.jsp）

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <%@ page import="model.UserBean, java.util.*" %>
3  <html>
4
```

（次ページに続く）

```
5  <head><title>メインページ</title></head>
6  <body>
7
8  <p>${user.name}さん，こんにちは。</p>
9  <p>${(empty user.name) == false ? user.name : "ゲスト"}
10         さん，こんにちは</p>
11 <p>ログアウトは<a href='LogoutProcess'>こちら</a></p>
12 </body>
13 </html>
```

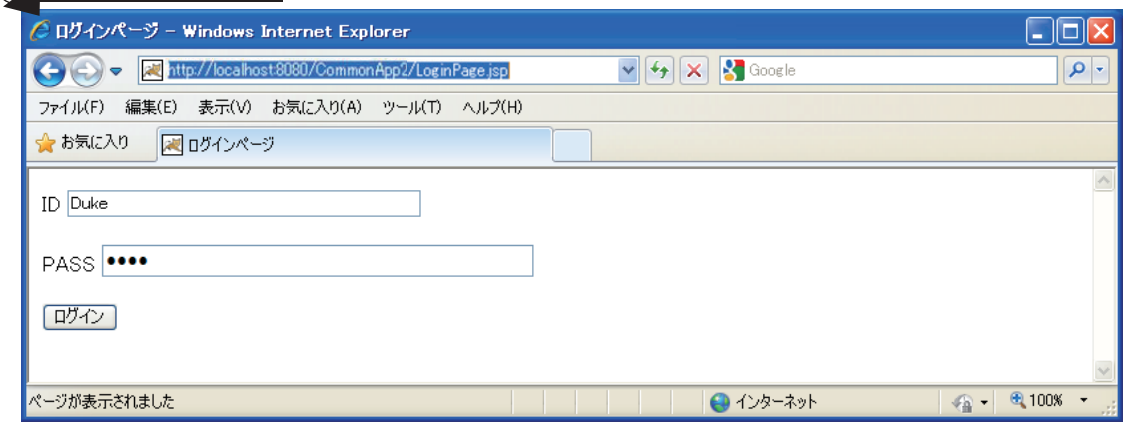
・9行目の${(empty　user.name) == false ? user.name : “ゲスト”}がEL式の条件式です。
EL式の条件式は${（論理式）?（真の場合）:（偽の場合）}と記述します。
「empty　user.name」はEL式のempty演算子で，後に続く値（user.name）が空なら「true」を返し，空でなければ「false」を返します。

今回の例では，属性名userの通信セッションが存在し，user.nameが空でなければuser.nameの値を表示し，user.nameが空もしくは属性名userの通信セッションが存在しない場合は「ゲスト」と表示します。

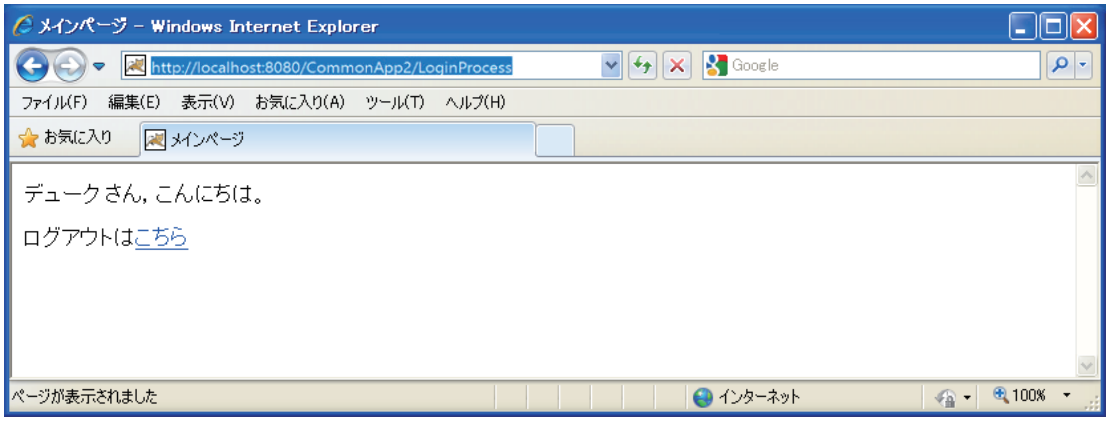
ただし，EL式はあくまでも簡単に表示させるための式なので，通信セッションの有無によってJSPの処理を大きく分岐させたり，通信セッションに記憶させている値を変更したりといった複雑な処理を行うことはできません。

実行結果①

(URL：http://localhost:8080/CommonApp2/LoginPage.jsp)

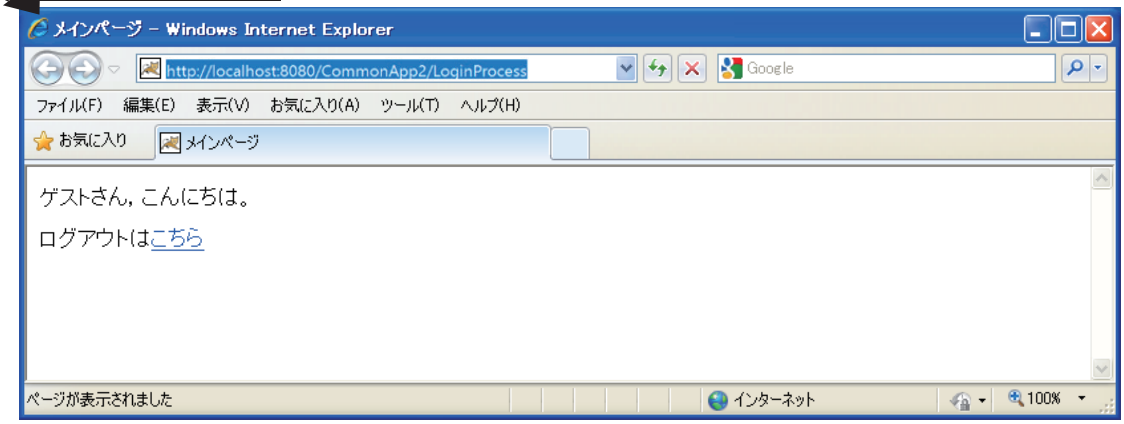


利用者認証の成功時



実行結果②

※セッションが存在しない場合



※実行結果は，すべてのWebブラウザを閉じてから確認する。

1-3-5　インクルード／フォワード

JSPのHTML文書を記述する部分では，標準アクションのタグを使用して，インクルード／フォワード，リクエストパラメータ関連の操作などを簡単に行うことができます。代表的な標準アクションは，次のとおりです。

表21　代表的な標準アクション

標準アクション	概要
include	他のJSP／Javaサーブレット（Javaバイトコード）をインクルードする。
forward	他のJSP／Javaサーブレット（Javaバイトコード）にフォワードする。
param	インクルード先／フォワード先にデータをリクエストパラメータとして渡す。
useBean	JavaBeans準拠のクラスからインスタンスを生成する。
setProperty	JavaBeans準拠のインスタンス内のフィールドに値を設定する。
getProperty	JavaBeans準拠のインスタンス内のフィールドから値を取得する。

※インクルード／フォワードの考え方は，Javaサーブレットのものと同じです。
詳細は，「1-2-3　インクルード／フォワード」を参照しくてください。

ここでは，標準アクション「include」と「forward」のタグについて説明します。

表22　標準アクション「include」のタグ仕様

項目	内容
タグ	〈jsp:include page="インクルード先" flush="モード"〉 ～ 〈/jsp:include〉
属性値	インクルード先 …インクルード先のJSPのファイル名，Javaサーブレットの名称を指定する。 モード …true　：インクルード前にバッファをクリアする。 false：インクルード前にバッファをクリアしない。

表23　標準アクション「forward」のタグ仕様

項目	内容
タグ	〈jsp:forward page="フォワード先"〉 ～ 〈/jsp:forward〉
属性値	フォワード名 …フォワード先のJSPのファイル名，Javaサーブレットの名称を指定する。

実際に，JSPのプログラムを見てみましょう。

プログラム 1-26　（ファイル名：MainPage.jsp）

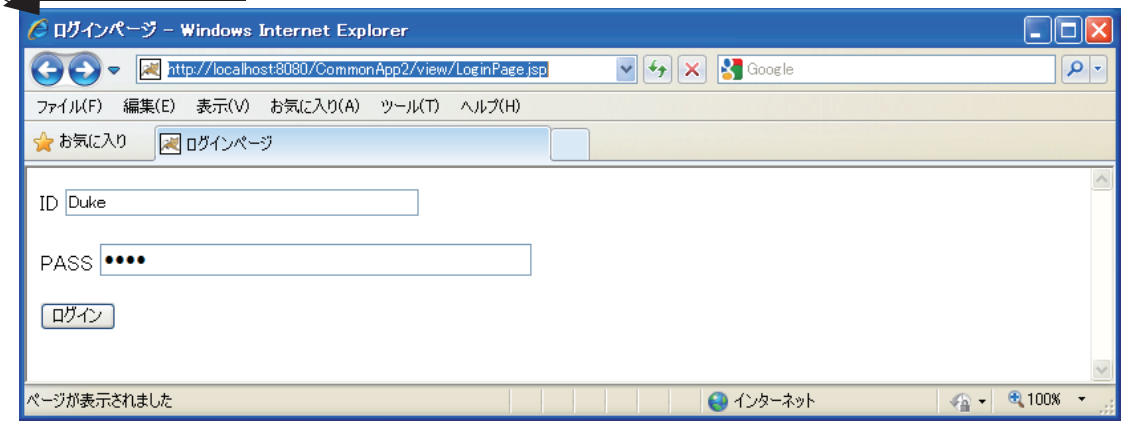
```
7   :
8   <p>ログアウトは<a href='LogoutProcess'>こちら</a></p>
9   <jsp:include page="SubPage.jsp" flush="true"></jsp:include>
10  </body>
11  </html>
```

・9行目の〈jsp:include〉タグでは，page属性に“SubPage.jsp”，flush属性に“true”を指定しています。このため，MainPage.jspの実行時には，SubPage.jspの出力結果が含まれます。

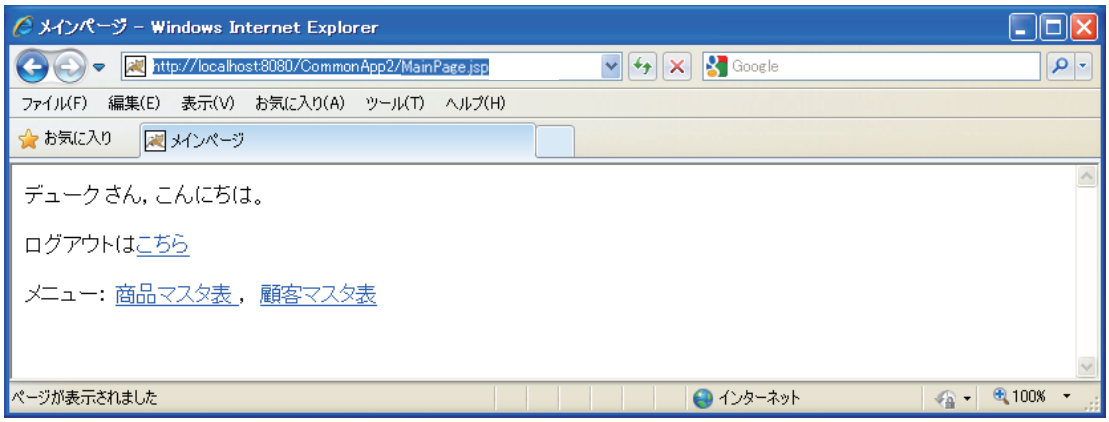
プログラム 1-27　（ファイル名：SubPage.jsp）

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2
3  <p>メニュー：
4  <a href='ProductList' target='frame'>
5  商品マスタ表
6  </a>,
7  <a href='CustomerList' target='frame'>
8  顧客マスタ表
9  </a>
10 </p>
```

実行結果　（URL：http://localhost:8080/CommonApp2/LoginPage.jsp）

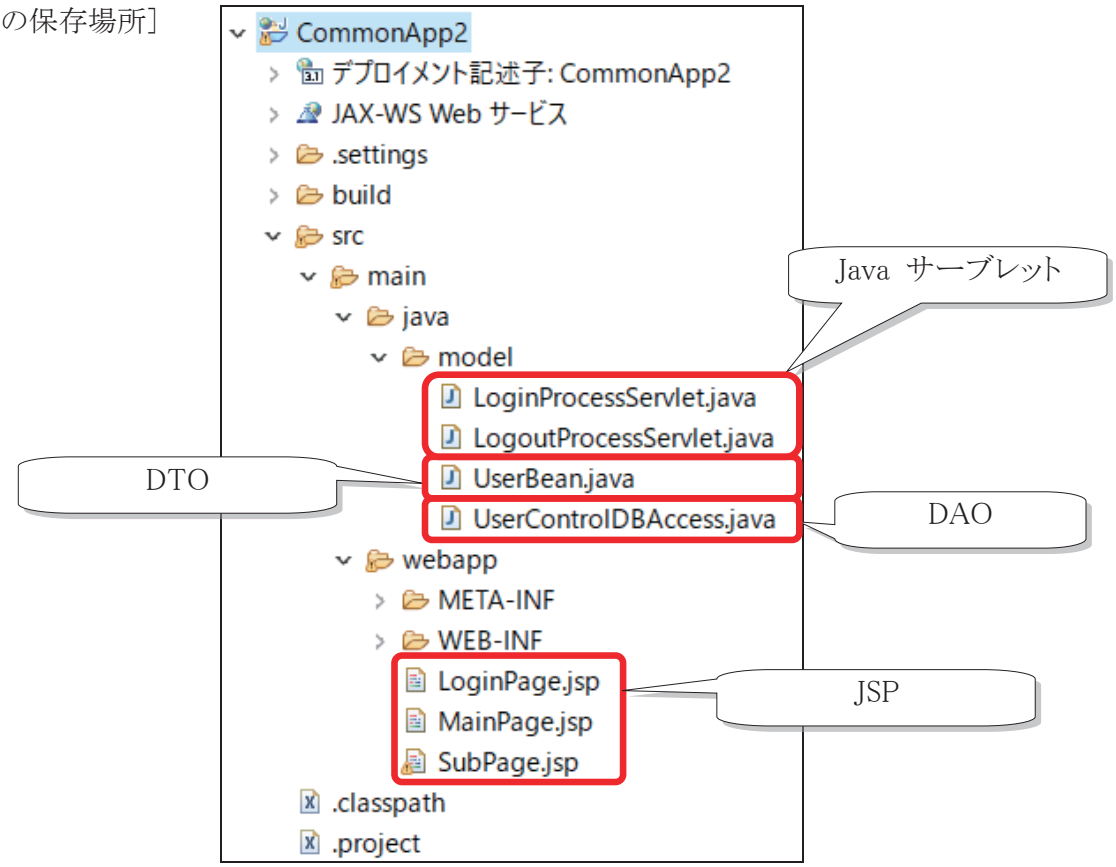


利用者認証の成功時



※「商品マスタ表」，「顧客マスタ表」リンクをクリックしないでください。

［ファイルの保存場所］


CommonApp2
デプロイメント記述子: CommonApp2
JAX-WS Web サービス
.settings
build
src
main
java
model
LoginProcessServlet.java
LogoutProcessServlet.java
UserBean.java
UserControlDBAccess.java
webapp
META-INF
WEB-INF
LoginPage.jsp
MainPage.jsp
SubPage.jsp
.classpath
.project
Java サーブレット
DTO
DAO
JSP

1-3-6　複数選択可能項目のデータを受け取る

これまではWebブラウザから送信された情報を受け取るためにgetParameter()メソッドを使用していますが，複数選択可能なチェックボックスやリストボックスなどからデータを受け取るにはgetParameterValues()メソッドを使用します。

実際に，JSP／Javaサーブレットのプログラムを見てみましょう。

プログラム 1-28　（ファイル名：LoginPage.jsp）

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2
3  <html>
4  <head>
5  <title>ログインページ</title>
6  </head>
7  <body>
8  <form action='LoginProcess' method='POST'>
9  <p>ID<input type='text' size='50' name='id' /></p>
10 <p>PASS<input type='password' size='50' name='pass' /></p>
11 <p>学習科目を選択してください。
12 <input type="checkbox" name="subject" value="java">java：
13 <input type="checkbox" name="subject" value="サーブレット">
14 サーブレット：
15 <input type="checkbox" name="subject" value="JSP">JSP：
16 <input type="checkbox" name="subject" value="SQL">SQL：
17 <input type="checkbox" name="subject" value="アルゴリズム">
18 アルゴリズム
19 </p>
20 <input type='submit' value='ログイン' />
21     :
```

・12～18行目の<input>タグで複数選択可能なチェックボックスを作成しています。同一グループになるようにname属性にすべて同じ“subject”を指定しています。
value属性に指定した値が，実際にサーブレットに渡されるデータになります。

プログラム 1-29　（ファイル名：LoginProcessServlet.java）

```
    :
@WebServlet("/LoginProcess")
public class LoginProcessServlet extends HttpServlet
{
    protected void doPost(
        HttpServletRequest request,
        HttpServletResponse  response)
        throws ServletException, IOException
    {
        //文字コードをUTF-8に対応させる
        request.setCharacterEncoding("UTF-8");
        //リクエストパラメータidを取り出す
        String id = request.getParameter("id");
        //リクエストパラメータpassを取り出す
        String pass = request.getParameter("pass");
        //リクエストパラメータsubjectを取り出す
        String[] subject = request.getParameterValues("subject");
        //サーブレットコンテキストを取得する
        ServletContext sc = getServletContext();
        UserControlDBAccess dao = new UserControlDBAccess();
        UserBean bean = dao.findUserByIdAndPass(id, pass);
        if(bean != null)
        {
            //新しい通信セッションを開始する
            HttpSession session = request.getSession(true);
            //通信セッションに属性名“user”と属性値beanを登録する
            session.setAttribute("user", bean);
            //通信セッションに属性名“subject”と属性値subjectを登録する
            session.setAttribute("subject", subject);
    :
```

・19行目のrequest.setCharacterEncoding("UTF-8");は，サーブレットの文字コードをUTF-8に指定しています。指定しないとWebブラウザから全角文字を受け取ったときに

文字化けすることがあります。

・25行目のgetParameterValues()メソッドでWebページからname属性“subject”のデータを一括で受け取っています。getParameterValues()メソッドはvalue属性に指定された文字列を配列の形で返します。

・37行目は通信セッションにgetParameterValues()メソッドで取得した値を属性名“subject”として登録しています。

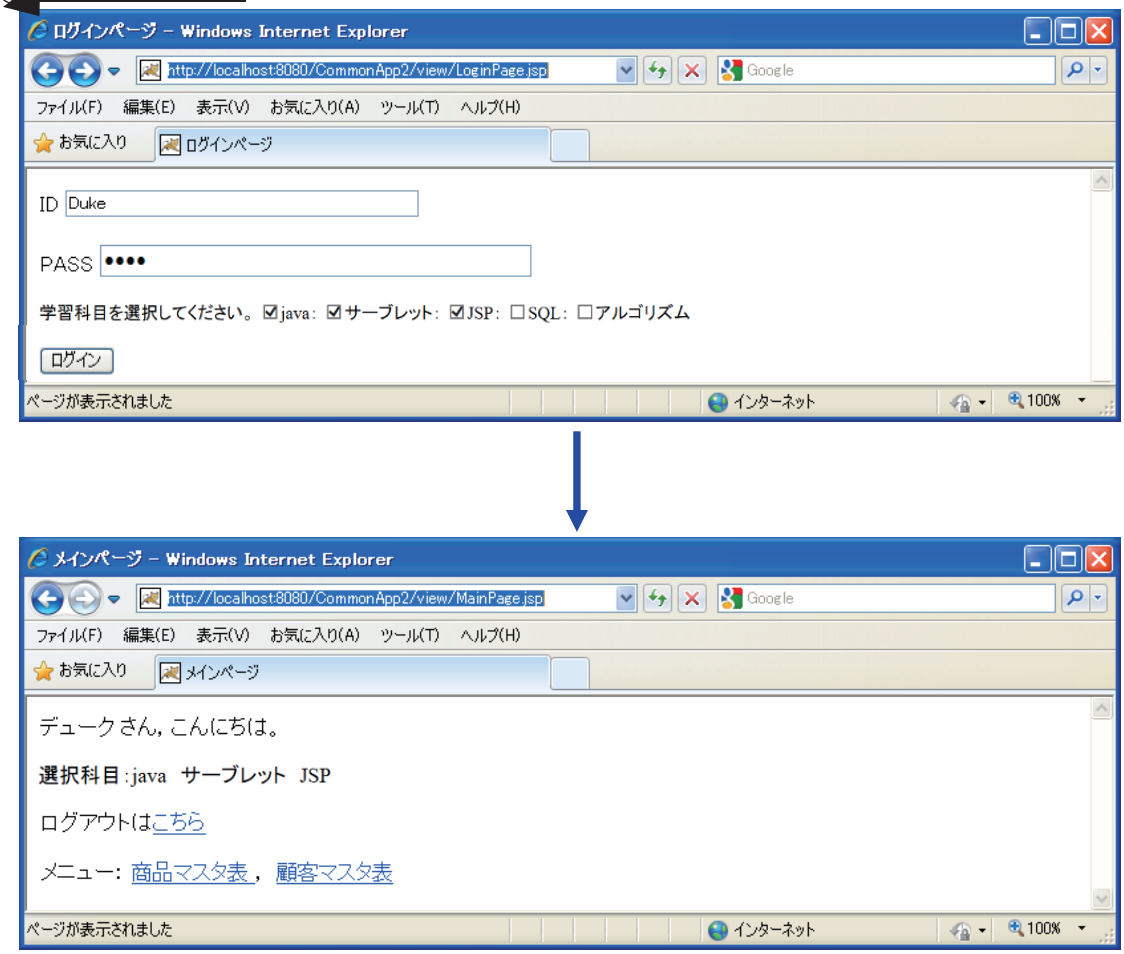
プログラム 1-30　（ファイル名：MainPage.jsp）

```
 :
<body>
<p>${(empty user.name) == false ? user.name : "ゲスト"}
さん，こんにちは</p>
<p>選択科目：${subject[0]}  ${subject[1]}  ${subject[2]}
${subject[3]}  ${subject[4]}</p>
<p>ログアウトは<a href='LogoutProcess'>こちら</a></p>
<jsp:include page="SubPage.jsp" flush="true"></jsp:include>
</body>
</html>
```

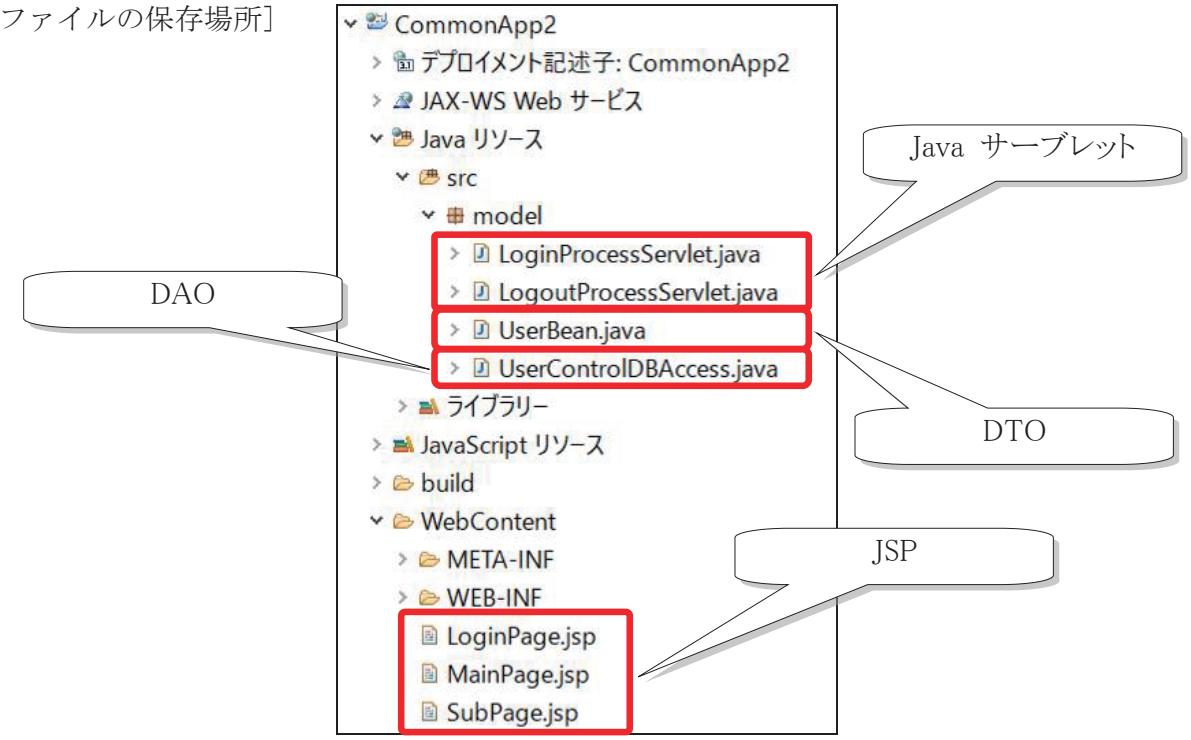
・9～10行目はEL式を使って通信セッションの属性名“subject”に記憶されている文字を表示しています。EL式での配列の表示は通常の配列と同様に“配列名[添え字]”と記述します。

EL式では，配列に値が記憶されていない箇所については表示されないだけでエラーにはなりません。

実行結果　(URL：http://localhost:8080/CommonApp2/LoginPage.jsp)



［ファイルの保存場所］


CommonApp2
デプロイメント記述子: CommonApp2
JAX-WS Web サービス
Java リソース
src
model
LoginProcessServlet.java
LogoutProcessServlet.java
UserBean.java
UserControlDBAccess.java
ライブラリー
JavaScript リソース
build
WebContent
META-INF
WEB-INF
LoginPage.jsp
MainPage.jsp
SubPage.jsp
Java サーブレット
DAO
DTO
JSP

1-4　MVCアーキテクチャ

1-4-1　MVCモデル

Webアプリケーションでは，MVCアーキテクチャを利用して開発作業と保守作業を効率よく行います。MVCアーキテクチャは，Webアプリケーション全体をModel，View，Controllerの３つに分けて考えます。

[MVCアーキテクチャ]

・Model

Webアプリケーションで必要となるデータを保持します。また，Webアプリケーションの機能を実現するための本質的なデータ処理を担当します。

Javaテクノロジを利用したWebアプリケーションでは，“JavaBeansに準拠したクラス”や“DTO（Data Transfer Object）及びDAO（Data Access Object）のクラス”などが該当します。

・View

Webブラウザに返すHTML文書を作成します。ただし，HTML文書に埋め込むためのデータ処理はModelに委譲します。

Javaテクノロジを利用したWebアプリケーションでは，“JSP”が該当します。

・Controller

Webブラウザから受け取ったデータをチェックしてModelに渡します。また，Webアプリケーションの機能を実現するための非データ処理やView呼出しなどを担当します。

Javaテクノロジを利用したWebアプリケーションでは，“Javaサーブレット”が該当します。

また，Webコンテナ内の，MVCアーキテクチャを利用したWebアプリケーションの動作順序は，次のとおりです。

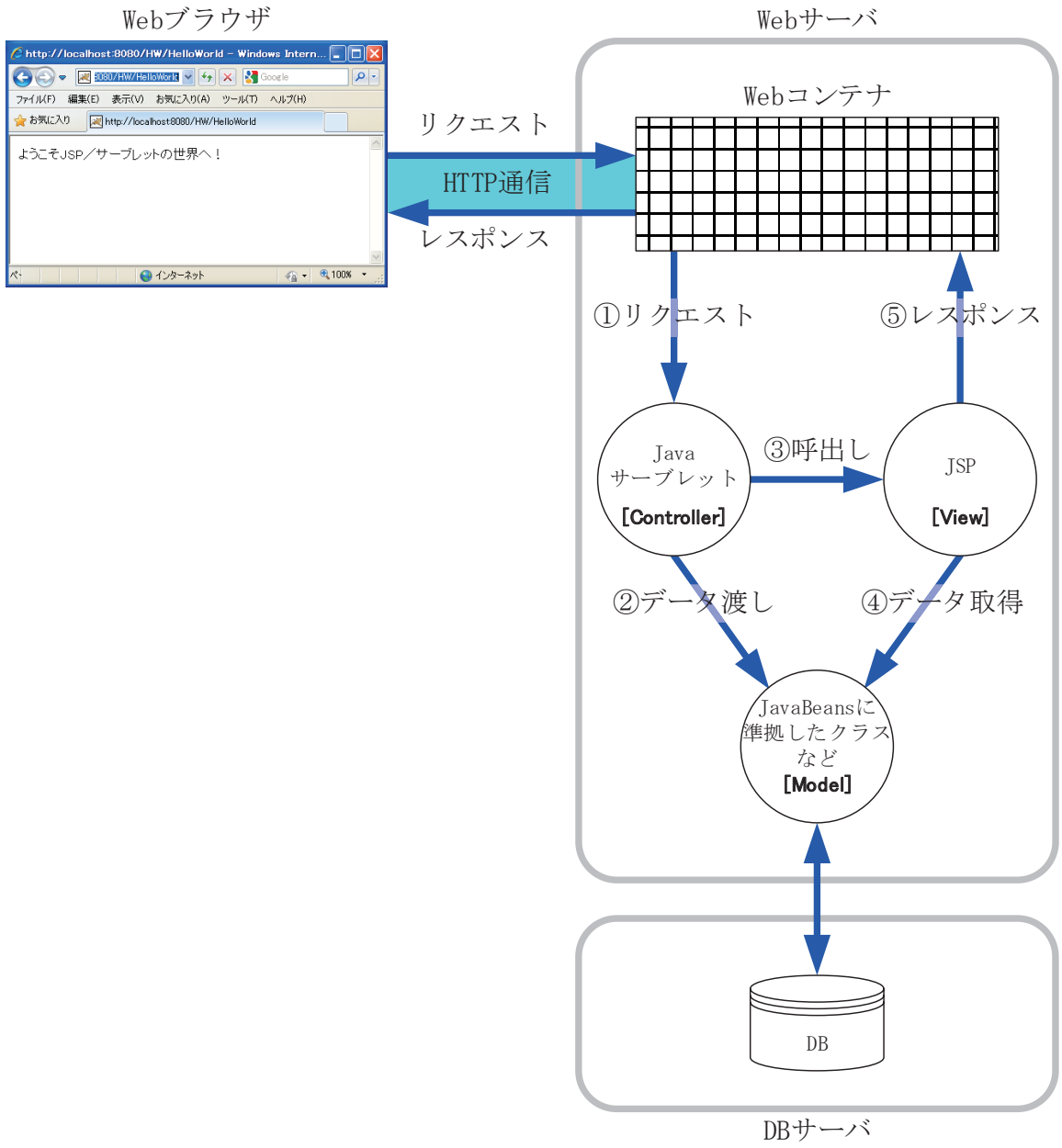


図１２　MVCアーキテクチャを利用したWebアプリケーションの動作順序

実際に，MVCアーキテクチャを利用したJSP／Javaサーブレットのプログラムを見てみましょう。

プログラム 1-31　（ファイル名：UserList.jsp）

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <%@ page import="model.UserBean, java.util.*" %>
3
4  <html>
5  <head><title>利用者マスタ表</title></head>
6  <body>
7
8  <%
9      ArrayList<UserBean> list
10         = (ArrayList)session.getAttribute("list");
11     if(list == null)
12     {
13 %>
14 エラーが発生しました。<br />
15 <%
16     }
17     else if(list.size() > 0)
18     {
19 %>
20 <div align='center'>
21 <table border='1'>
22 <caption>利用者マスタ表</caption>
23 <tr><th>ID</th><th>氏名</th></tr>
24 <%
25         for(UserBean bean : list)
26         {
27 %>
28 <tr><td><%=bean.getId() %></td>
29     <td><%=bean.getName() %></td></tr>
30
```

```
31  <%
32          }
33  %>
34  </table>
35  </div>
36  <%
37      }
38      else
39      {
40  %>
41  <p>利用者マスタ表のレコード数は０件です。</p>
42  <%
43      }
44  %>
45
46  </body>
47  </html>
```

・10行目のsession.getAttribute()メソッドは，暗黙オブジェクト「session」を使用して通信セッションから属性名“list”に対応する属性値（DTO）を取得します。なお，View（JSP）で使用するModel（DTO）は，プログラム1-32のController（Javaサーブレット）で通信セッションに属性名“list”で登録されます。

プログラム 1-32　（ファイル名：UserListServlet.java　URL：/UserList）

```
1  package  ctrl;
2
3  import  javax.servlet.*;
4  import  javax.servlet.annotation.*;
5  import  javax.servlet.http.*;
6  import  java.io.*;
7  import  java.util.ArrayList;
8  import  model.UserControlDBAccess;
9  import  model.UserBean;
```

（次ページに続く）

```
@WebServlet("/UserList")
public class UserListServlet extends HttpServlet
{
    protected void doGet(
        HttpServletRequest  request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        //DAOを生成する。
        UserControlDBAccess dao = new UserControlDBAccess();
        //DAOから利用者マスタ表の全レコードを取得する。
        ArrayList<UserBean> list = dao.findAllUsers();

        //新しい通信セッションを開始する。
        HttpSession session = request.getSession(true);
        //通信セッションに属性名"list"と属性値listを登録する。
        session.setAttribute("list", list);
        //サーブレットコンテキストを取得する
        ServletContext sc = getServletContext();
        //リクエストディスパッチャを取得する
        RequestDispatcher rd =
            sc.getRequestDispatcher("/UserList.jsp");
        rd.forward(request, response);
    }
}
```

なお，実行結果を確認するには，プログラム1-19をそのまま使用し，プログラム1-20，1-22，1-29を一部修正して使用します。

プログラム 1-33　（ファイル名：UserControlDBAccess.java）

```
package model;

import java.sql.*;
import java.util.ArrayList;
    :
```

```
113     :
114     //利用者リストを取得する
115     public  ArrayList<UserBean>  findAllUsers()
116     {
117         Connection con = createConnection();
118         PreparedStatement stmt = null;
119         ResultSet rs = null;
120         ArrayList<UserBean> list = new ArrayList<UserBean>();
121         try
122         {
123             String sql = "SELECT ID, 氏名 FROM 利用者マスタ;";
124             stmt = con.prepareStatement(sql);
125             rs = stmt.executeQuery();
126             while(rs.next() == true)
127             {
128                 String uId = rs.getString("ID");
129                 String uName = rs.getString("氏名");
130                 UserBean bean = new UserBean(uId, uName);
131                 list.add(bean);
132             }
133         }
134         catch(SQLException e)
135         {
136             System.out.println(
137                 "DB アクセス時にエラーが発生しました（利用者検索）。");
138             e.printStackTrace();
139         }
140         finally
141         {
142             try
143             {
144                 if(rs != null)
145                 {
```

（次ページに続く）

```
146                 rs.close();
147             }
148         }
149         catch(SQLException e)
150         {
151             System.out.println(
152                 "DB アクセス時にエラーが発生しました。");
153             e.printStackTrace();
154         }
155         try
156         {
157             if(stmt != null)
158             {
159                 stmt.close();
160             }
161         }
162         catch(SQLException e)
163         {
164             System.out.println(
165                 "DB アクセス時にエラーが発生しました。");
166             e.printStackTrace();
167         }
168     }
169     closeConnection(con);
170     return list;
171   }
172 }
```

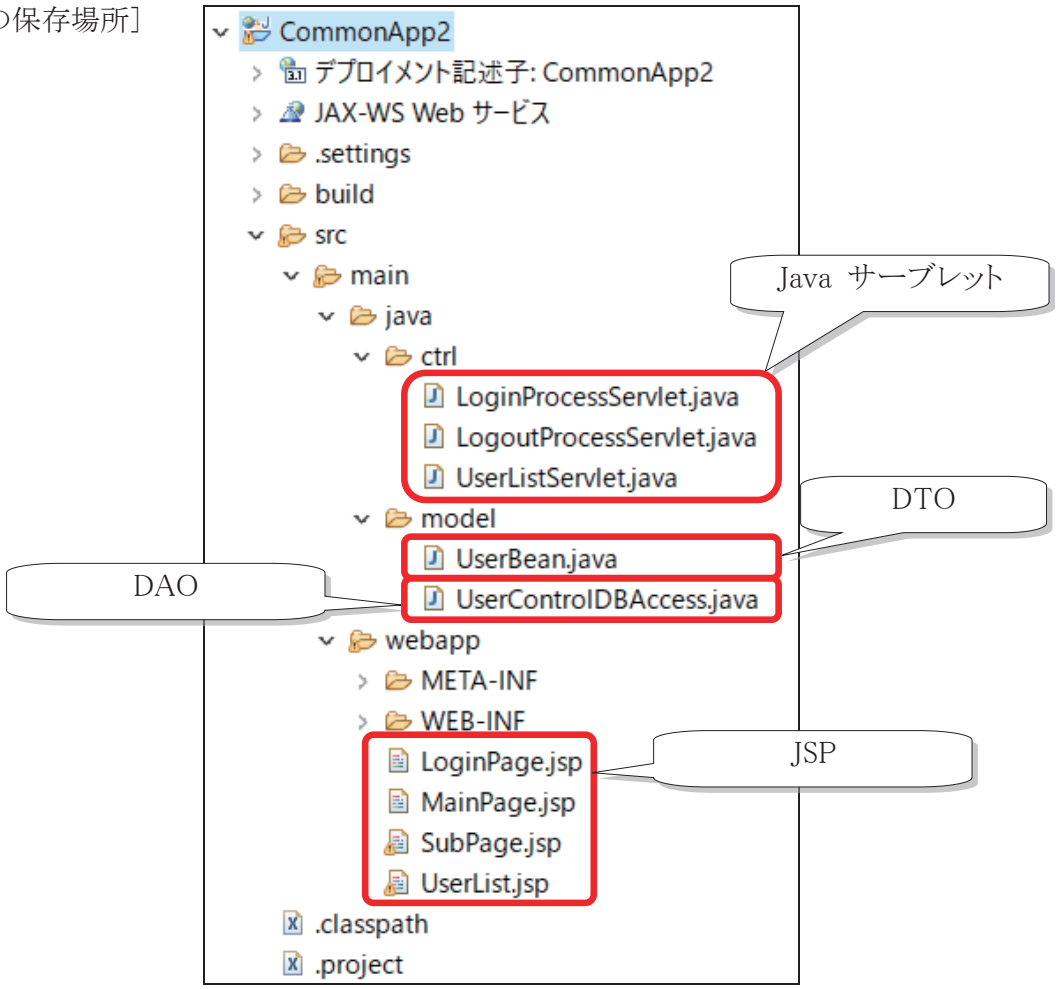
プログラム 1-34　（ファイル名：LoginProcessServlet.java）

```
1  package ctrl;
2
3  import javax.servlet.*;
4  import javax.servlet.annotation.*;
5  import javax.servlet.http.*;
6  import java.io.*;
7  import model.UserControlDBAccess;
8  import model.UserBean;
9
10 public class LoginProcessServlet extends HttpServlet
11 {
12     :
```

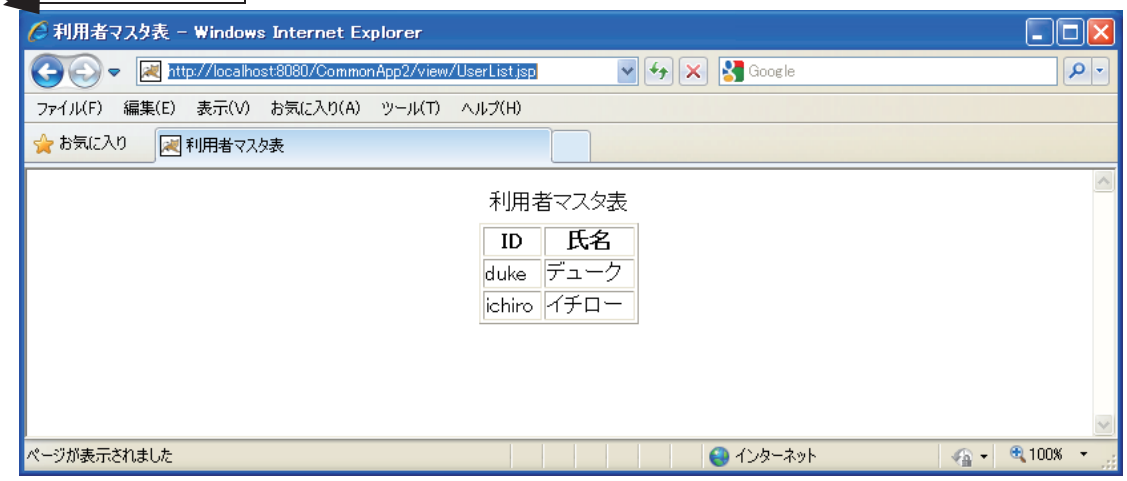
プログラム 1-35　（ファイル名：LogoutProcessServlet.java）

```
1  package ctrl;
2
3  import javax.servlet.*;
4  import javax.servlet.annotation.*;
5  import javax.servlet.http.*;
6  import java.io.*;
7
8  public class LogoutProcessServlet extends HttpServlet
9  {
10     :
```

［ファイルの保存場所］



実行結果　(URL：http://localhost:8080/CommonApp2/UserList)



1-4-2 フロントコントローラパターン

MVCアーキテクチャで有効な設計パターンとして，現在のWebアプリケーション制作ではフロントコントローラパターンが広く用いられています。

フロントコントローラパターンとは，Webアプリケーションのすべてのリクエストの入口を一つのコントローラに集約し，リクエストの内容に応じてその先に実行する処理を振り分ける手法です。

例えば，同じログイン画面を使用していても，管理者としてログインする場合とユーザとしてログインする場合でその後に表示されるページを自動的に振り分けるアプリケーション等です。

実際に，Javaサーブレットのプログラムを見てみましょう。LoginProcessServletにフロントコントローラの機能を実装します。

プログラム 1-36　（ファイル名：LoginProcessServlet.java）

```
42      :
43          else
44          {
45              //ユーザ名「admin」パスワード「password」指定時は
46              //管理者として/UserList へフォワード
47              if(id.equals("admin") && pass.equals("password"))
48              {
49                  //リクエストディスパッチャを取得する
50                  RequestDispatcher rd =
51                      sc.getRequestDispatcher("/UserList");
52                  rd.forward(request, response);
53              }
54              else
55              {
56                  request.setAttribute("status", "failure");
57                  //リクエストディスパッチャを取得する
58                  RequestDispatcher rd =
59                      sc.getRequestDispatcher("/LoginPage.jsp);
```

（次ページに続く）

```
60                  rd.forward(request, response);
61              }
62          }
63      }
64  }
```

・47～53行目はIDに“admin”パスワードに“password”が指定されたときに“/UserList”へフォワードします。これにより通常ユーザのID／パスワードを入力したときには“/MainPage.jsp”へフォワード，管理者のID／パスワードを入力した時には“/UserList”へフォワード、と分岐できるようになり，LoginProcessServlet.javaがフロントコントローラとしてすべてのログイン処理を集約するようになります。

なお，フォワード処理の際に転送元と転送先の送信方式は同じになります。転送元であるLoginProcessServletへPOSTでアクセスされた場合は，転送先のプログラムUserListServletもPOSTでアクセスします。

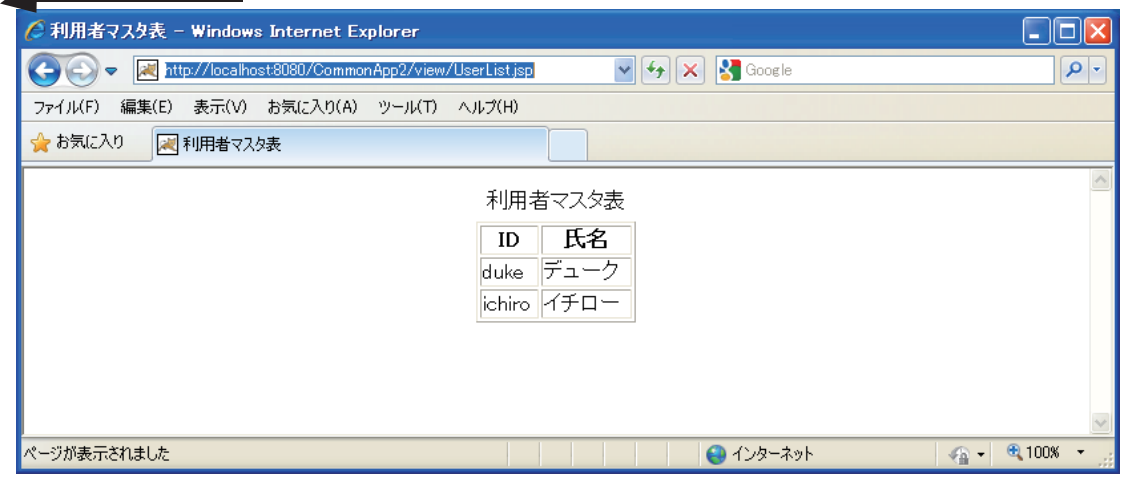
UserListServletにはdoPost()メソッドが実装されていないのでdoPost()メソッドを追加し，本来の処理を行っているdoGet()メソッドを呼び出すように変更します。

プログラム 1-37　（ファイル名：UserListServlet.java）

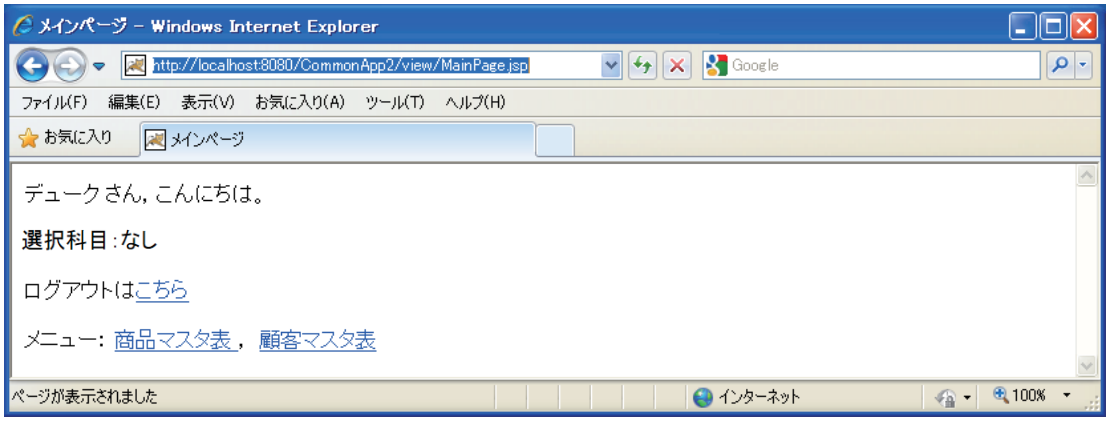
```
42      :
43      protected void doPost(
44          HttpServletRequest request, HttpServletResponse response)
45          throws ServletException, IOException
46      {
47          //LoginProcessServlet.java から送信されてくる時は Post なので
48          //本来の処理を行う doGet メソッドを呼び出す。
49          this.doGet(request, response);
50      }
51  }
```

・49行目の“this.doGet(request, response);”はUserListServletのdoGet()メソッドを呼び出しています。それによりPOSTでUserListServletが呼び出されても本来の処理が実行できるようになります。

実行結果　(URL：http://localhost:8080/CommonApp2/LoginPage.jsp)



LoginPage.jspでID“admin”パスワード“password”を入力した場合



LoginPage.jspでID“Duke”パスワード“java”を入力した場合

練習 1-14　レベル ☆☆☆

練習1-11で作成したProductBeanクラス，CustomerBeanクラス，DivisionBeanクラスとOrderControlDBAccessクラスを使用して，DB（受注管理DB）の商品マスタ表から全レコードを取得して表示する，MVCアーキテクチャを利用したJSP／Javaサーブレットのプログラムを作成しなさい。

ヒント　（ファイル名：ProductList.jsp）

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ page import="model.ProductBean, java.util.*" %>

<html>
<head><title>商品マスタ表</title></head>
<body>

<%
    ArrayList<ProductBean> list
        = (ArrayList)session.getAttribute("list");
    if(list == null)
    {
%>
エラーが発生しました。<br />
<%
    }
    else if(list.size() > 0)
    {
%>
<div align='center'>
<table border='1'>
<caption>商品マスタ表</caption>
<tr><th>商品ID</th><th>商品名</th><th>単価</th></tr>
<%
        for(ProductBean bean : list)
```

```
        {
%>
 :
<%
        }
%>
</table>
</div>
<%
    }
    else
    {
%>
商品マスタ表のレコード数は０件です。<br />
<%
    }
%>

</body>
</html>
```

ヒント　（ファイル名：ProductListServlet.java　URL：/ProductList）

```
package  ctrl;

import  javax.servlet.*;
import  javax.servlet.annotation.*;
import  javax.servlet.http.*;
import  java.io.*;
import  java.util.ArrayList;
import  model.OrderControlDBAccess;
import  model.ProductBean;

@WebServlet("/ProductList")
```

（次ページに続く）

```
public  class  ProductListServlet  extends  HttpServlet
{
    protected  void  doGet(
        HttpServletRequest  request,
        HttpServletResponse   response)
        throws  ServletException, IOException
    {
        :
        //サーブレットコンテキストを取得する
        ServletContext sc = getServletContext();
        //リクエストディスパッチャを取得する
        RequestDispatcher rd = sc.getRequestDispatcher("/ProductList.jsp");
        rd.forward(request, response);
    }
}
```

プログラム　（ファイル名：Server.xml）

```
124                 :
125                 <Context  docBase="Ren1_14" path="/Ren1_14"
126                     reloadable="true" />
127             </Host>
128         </Engine>
129     </Service>
130 </Server>
```

実行結果　(URL：http://localhost:8080/Ren1_14/ProductList)

商品マスタ表

商品ID	商品名	単価
F-001	ストロベリー	¥1580
F-002	ピーチ	¥1480
F-003	マンゴ	¥1480
F-004	トロピカル	¥1300

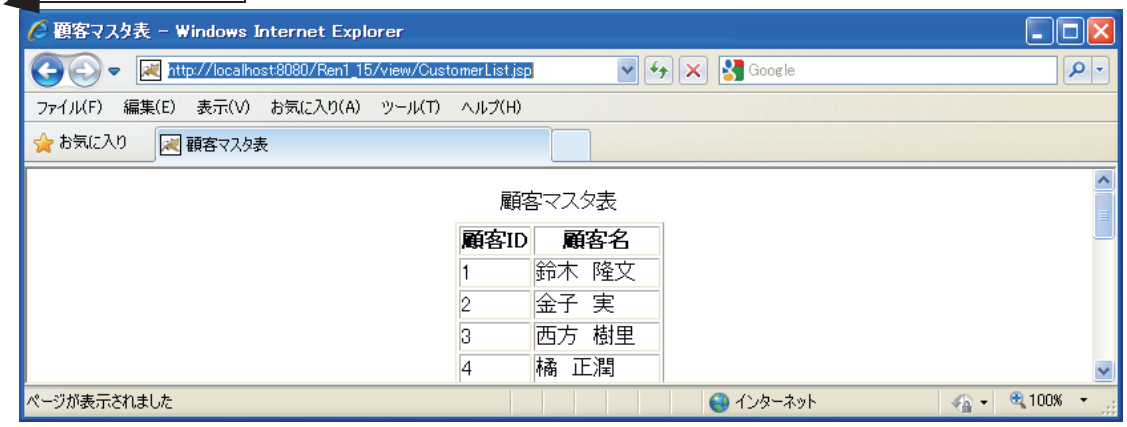
練習 1-15　レベル ☆☆☆

練習1-11で作成したProductBeanクラス，CustomerBeanクラス，DivisionBeanクラスとOrderControlDBAccessクラスを使用して，DB（受注管理DB）の顧客マスタ表から全レコードを取得して表示する，MVCアーキテクチャを利用したJSP／Javaサーブレットのプログラムを作成しなさい。（ファイル名：CustomerList.jsp，CustomerListServlet.java）

プログラム　（ファイル名：server.xml）

```
124                 :
125                 <Context  docBase="Ren1_15" path="/Ren1_15"
126                     reloadable="true" />
127             </Host>
128         </Engine>
129     </Service>
130 </Server>
```

実行結果　（URL：http://localhost:8080/Ren1_15/CustomerList）



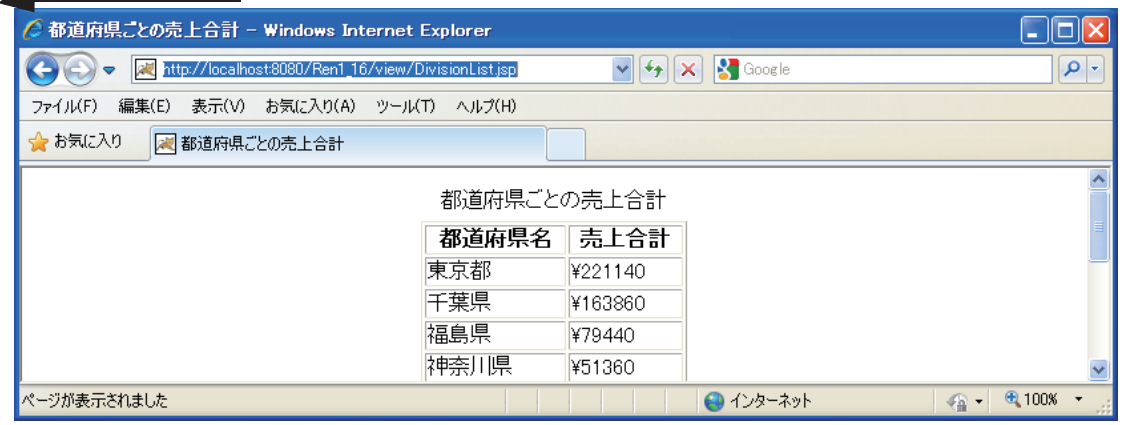
練習 1-16　レベル ☆☆☆

練習1-11で作成したProductBeanクラス，CustomerBeanクラス，DivisionBeanクラスとOrderControlDBAccessクラスを使用して，DB（受注管理DB）の顧客マスタ表，商品マスタ表，受注表，受注明細表を結合した表から，都道府県ごとの売上合計を取得して表示する，MVCアーキテクチャを利用したJSP／Javaサーブレットのプログラムを作成しなさい。（ファイル名：DivisionList.jsp，DivisionListServlet.java）

プログラム　（ファイル名：server.xml）

```
124                 :
125                 <Context  docBase="Ren1_16" path="/Ren1_16"
126                     reloadable="true" />
127             </Host>
128         </Engine>
129     </Service>
130 </Server>
```

実行結果　（URL：http://localhost:8080/Ren1_16/DivisionList）



練習 1-17　レベル ☆☆☆

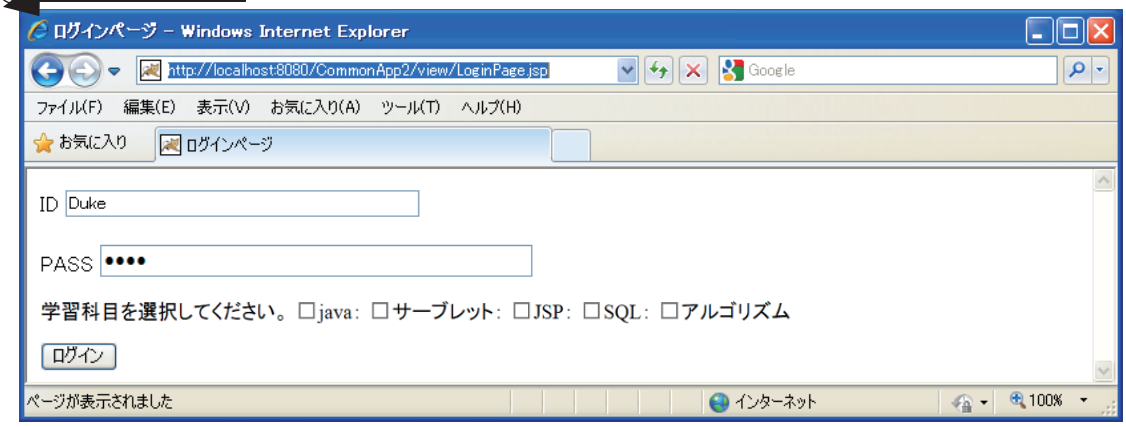
プログラム1-27，1-30～1-31，1-33，1-35～1-37において，練習1-14と1-15で作成したJSP／Javaプログラムを結合しなさい。

プログラム　（ファイル名：SubPage.jsp）

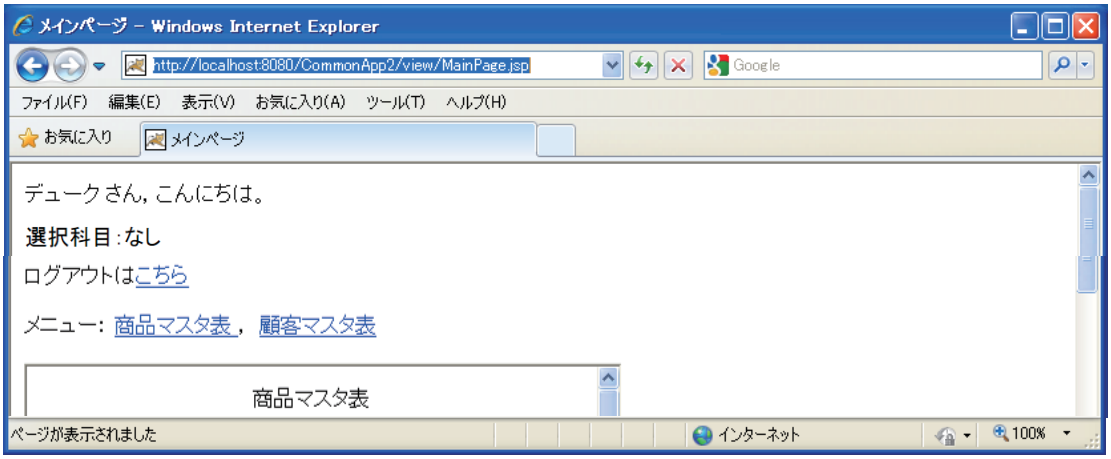
```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2
3  <p>メニュー：
4      <a href='ProductList' target='frame'>
5          商品マスタ表
6      </a>,
7      <a href='CustomerList' target='frame'>
8          顧客マスタ表
9      </a>
10 </p>
11 <iframe name='frame'
12         src='ProductList'
13         frameborder='1'
14         width='450'
15         height='450'>
16 </iframe>
```

実行結果

（URL：http://localhost:8080/CommonApp2/LoginPage.jsp）



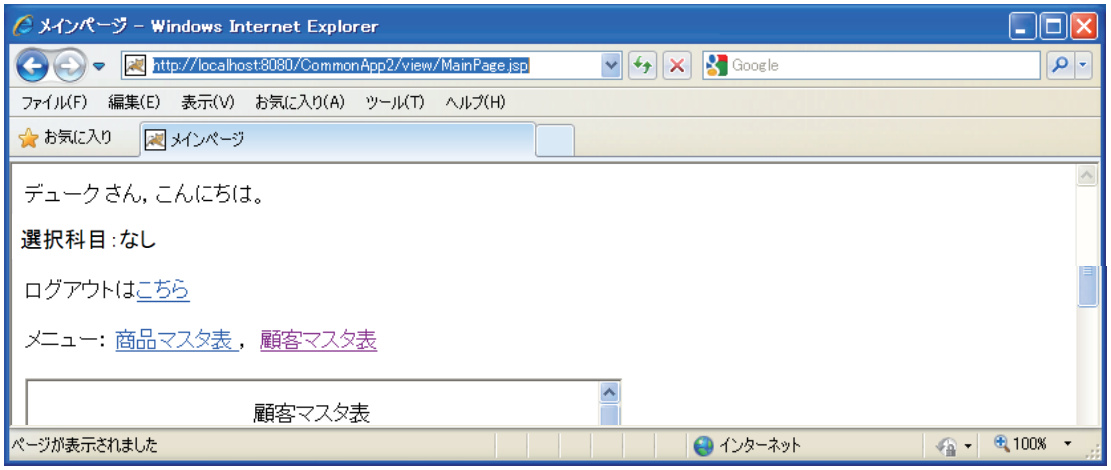
利用者認証の成功時



※実行結果はスクロールして確認する（表のレコード数：18）。

「顧客マスタ表」クリック時

「商品マスタ表」クリック時



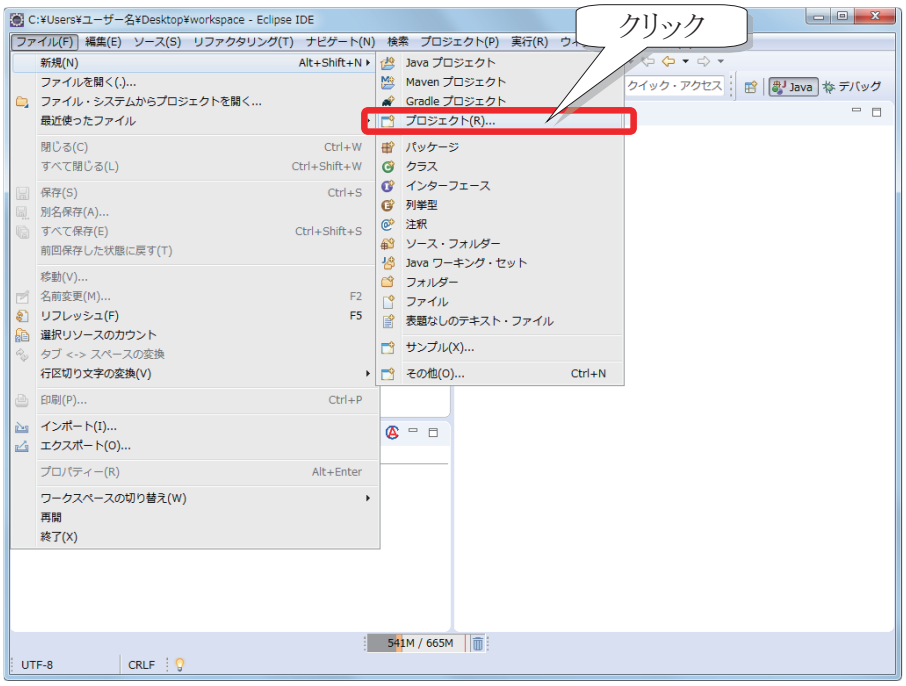
※実行結果はスクロールして確認する（表のレコード数：20）。

付録

付録 1　Webアプリ開発環境の構築

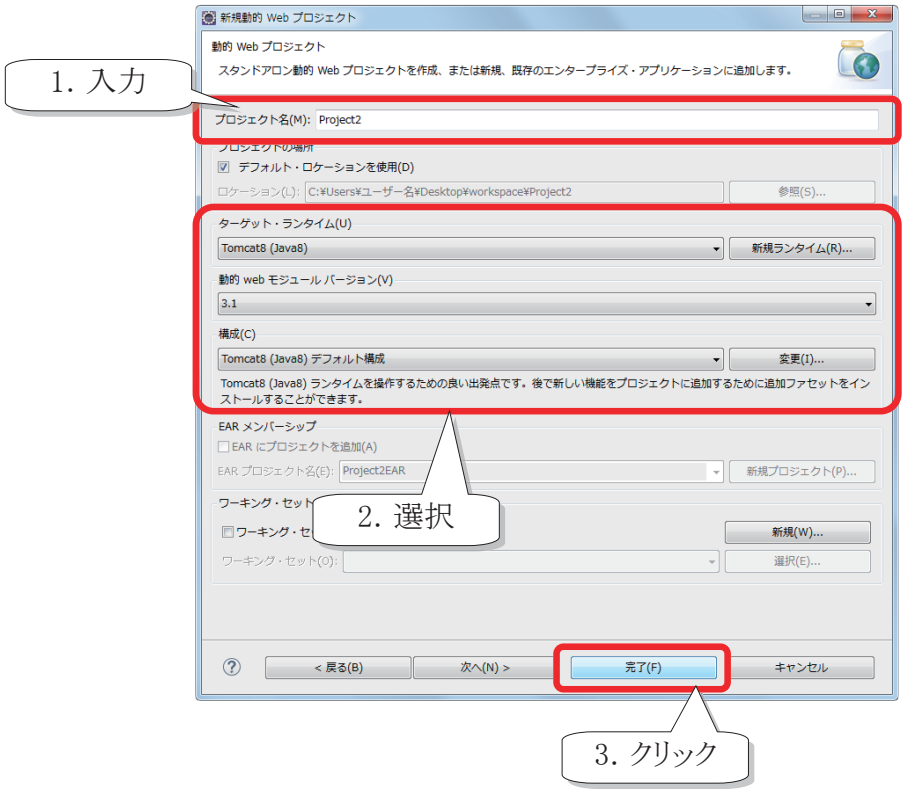
(1) 動的Webプロジェクトを作成します。次の①～⑨の順に進めてください。

① 「ファイル(F)」メニューの「新規(N)」をポイントして，「プロジェクト(R)...」をクリックする。

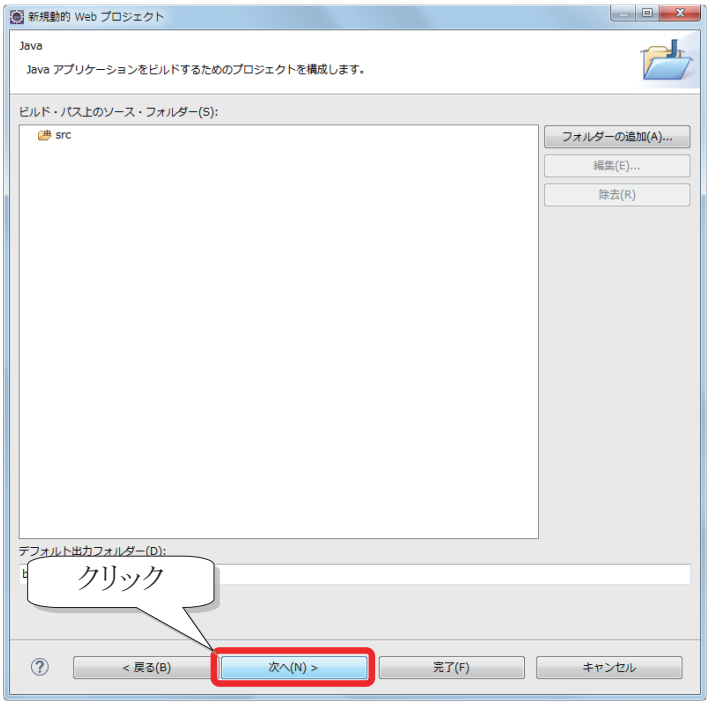


② ウィザード一覧から「Web」フォルダをダブルクリックして開き，「動的 Web プロジェクト」を選択して，「次へ(N) >」ボタンをクリックする。

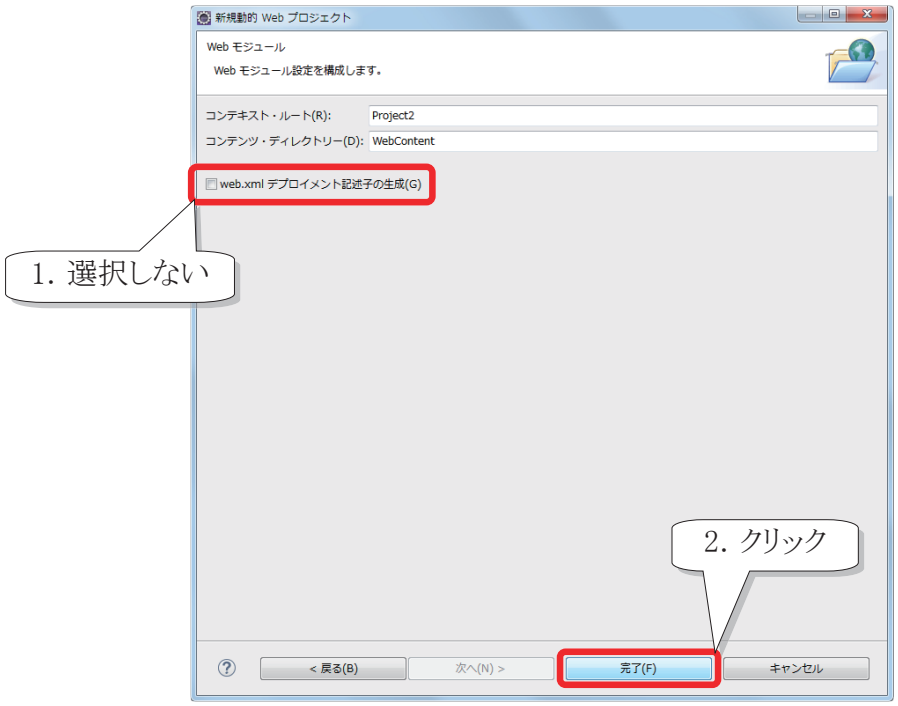
③ 「プロジェクト名(M)」に適切なプロジェクト名を入力して，「ターゲット・ランタイム(U)」に「Tomcat（Java8）」，「動的 web モジュール バージョン(V)」に「3.1」，「構成(C)」に「Tomcat8（Java8）デフォルト構成」を選択して，「次へ(N) >」ボタンをクリックする。



④　「次へ(N) >」ボタンをクリックする。

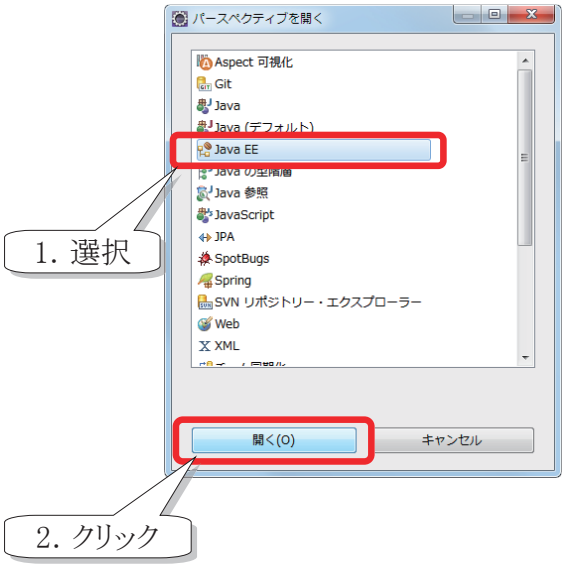


⑤　「web.xmlデプロイメント記述子の生成(G)」のチェックを外し，「完了(F)」ボタンをクリックする。

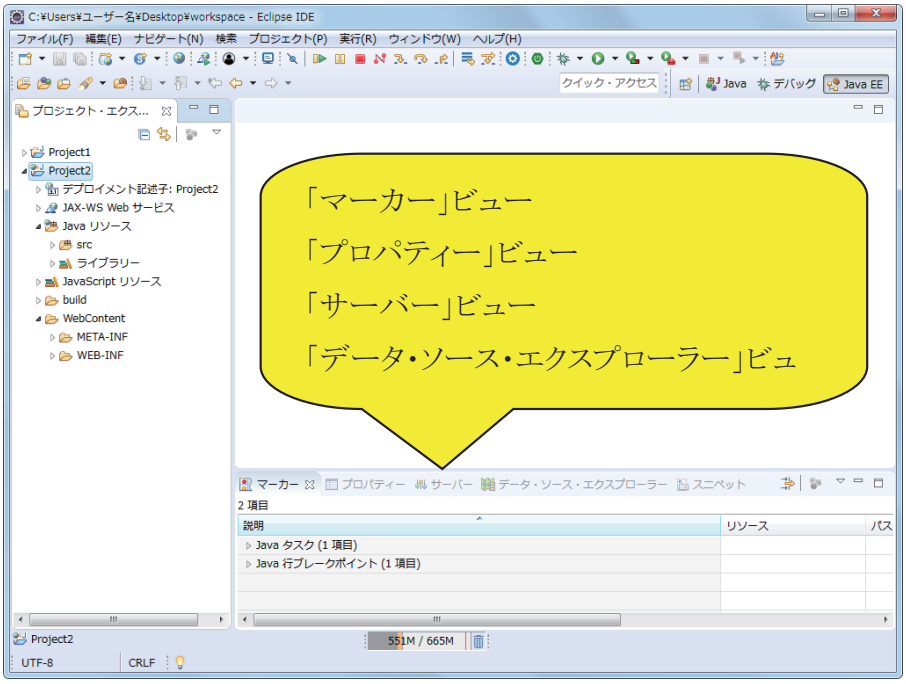


⑥　「ウィンドウ(W)」メニューの「パースペクティブ(R)」，「パースペクティブを開く(O)」の順にポイントして，「その他(O)...」をクリックする。

⑦　一覧から「Java EE」を選択して，「開く(O)」ボタンをクリックする。



⑧　不要なビューを閉じる。



- 「マーカー」ビュー

  … マーカーが付けられた箇所の一覧を表示する。

- 「プロパティー」ビュー

  … ファイルのプロパティの管理状況を表示する。

- 「サーバー」ビュー

  … サーバの管理状況を表示する。

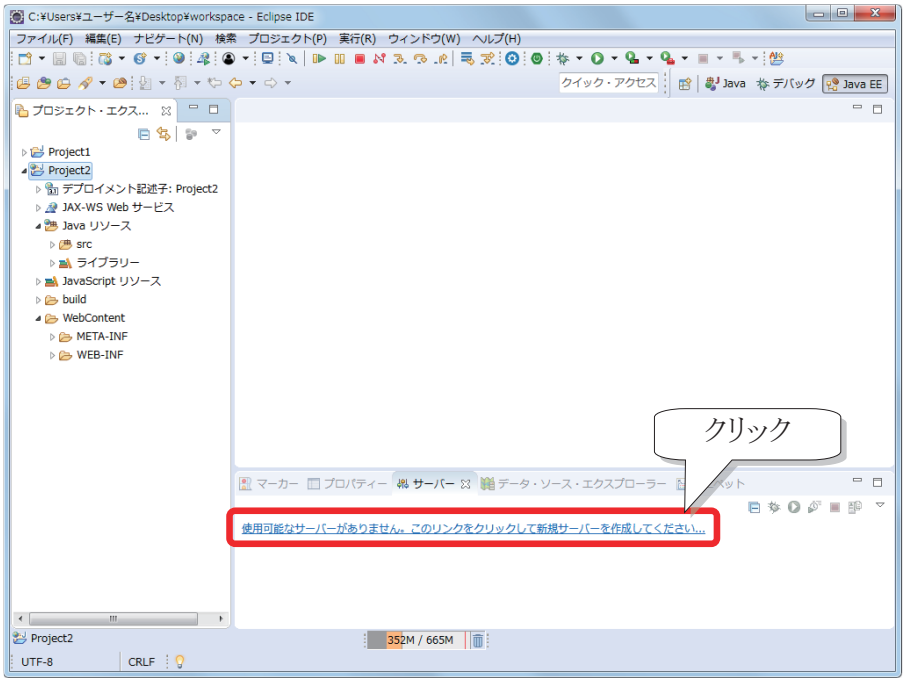
- 「データ・ソース・エクスプローラー」ビュー

  … データベースの管理状況を表示する。

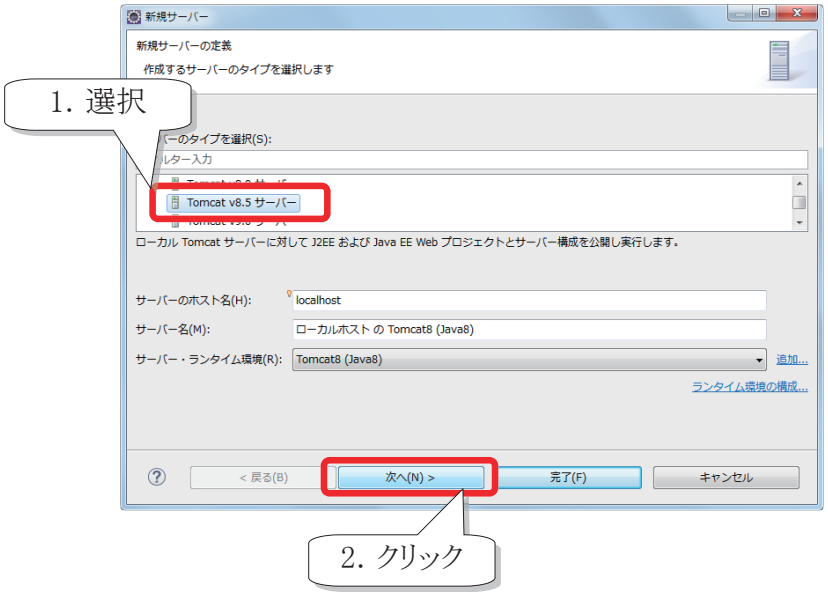
- 「スニペット」ビュー

  … スニペット（使用頻度が高いコードの登録）の一覧などを表示する。

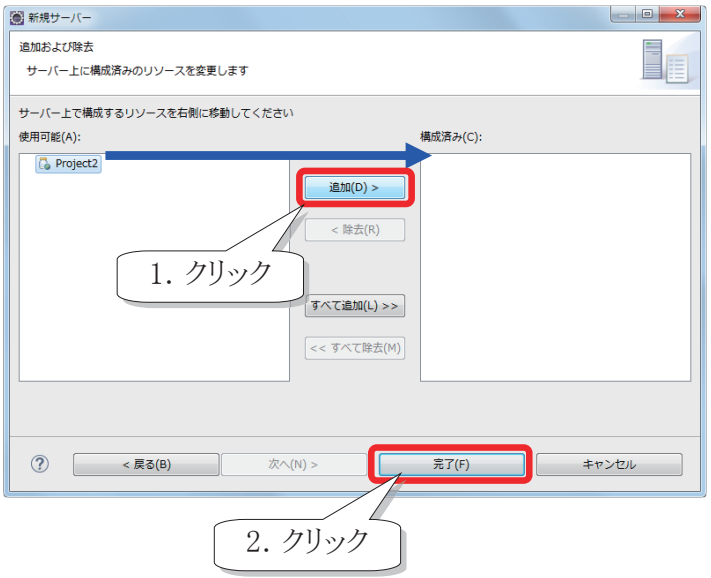
⑨　サーバー・ビューのタブをポイントして，ビュー内の「使用可能なサーバーがありません。このリンクをクリックして新規サーバーを作成してください...」リンクをクリックする。



サーバのタイプ一覧から「Apache」の「Tomcat v8.5 サーバー」を選択して，「次へ(N) >」ボタンをクリックする。

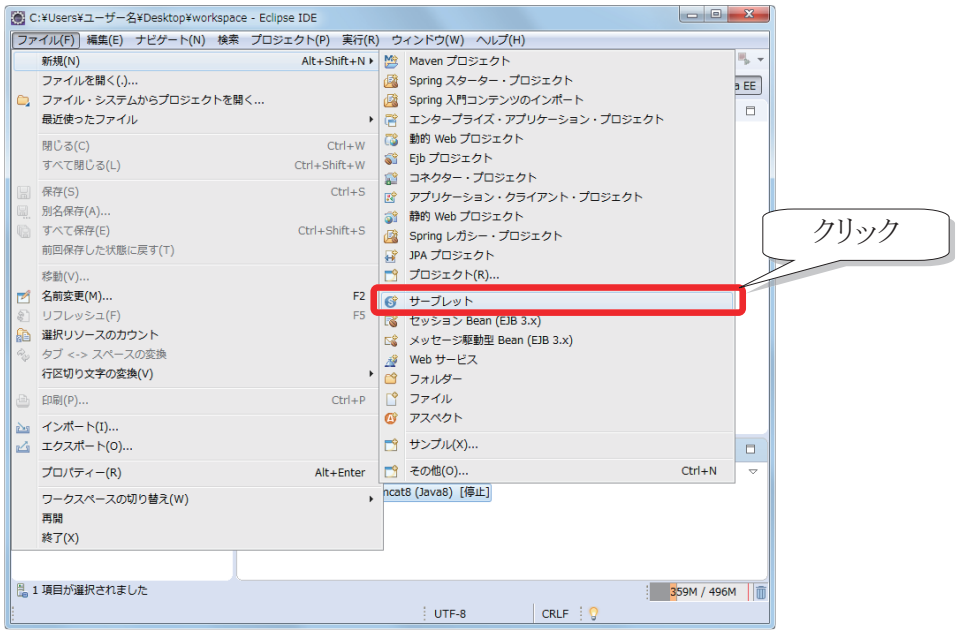


当該プロジェクト名を指定して，「追加(D) >」ボタンをクリックして，「完了(F)」ボタンをクリックする。

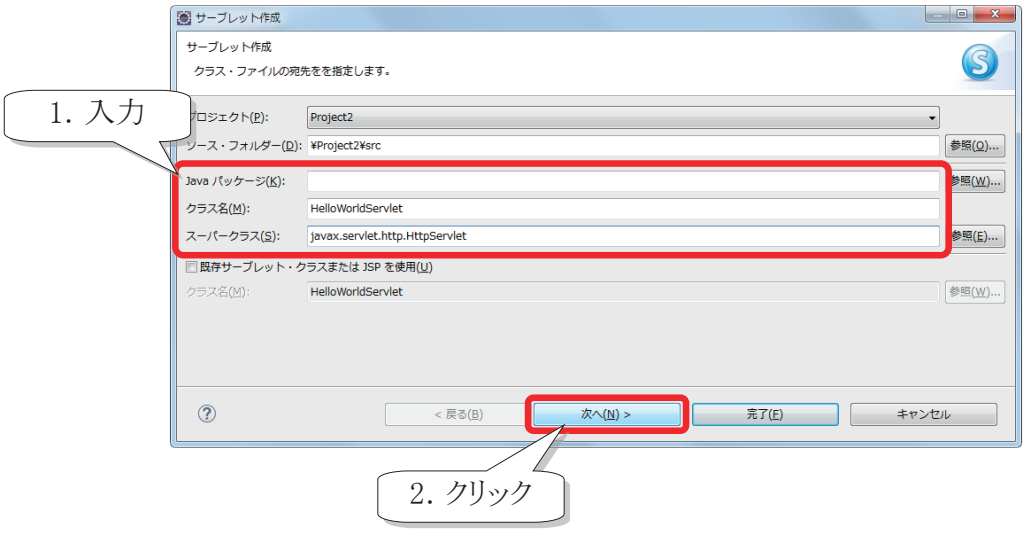


(2) サーブレットを作成します。次の①～⑧の順に進めてください。

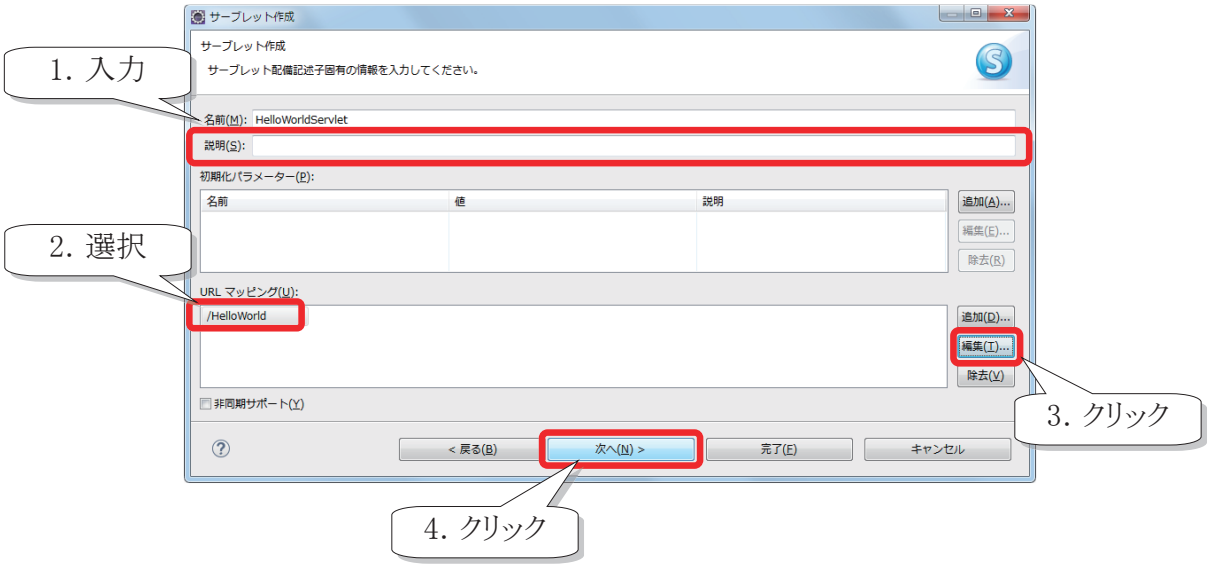
① 「ファイル(F)」メニューの「新規(N)」をポイントして，「サーブレット」をクリックする。



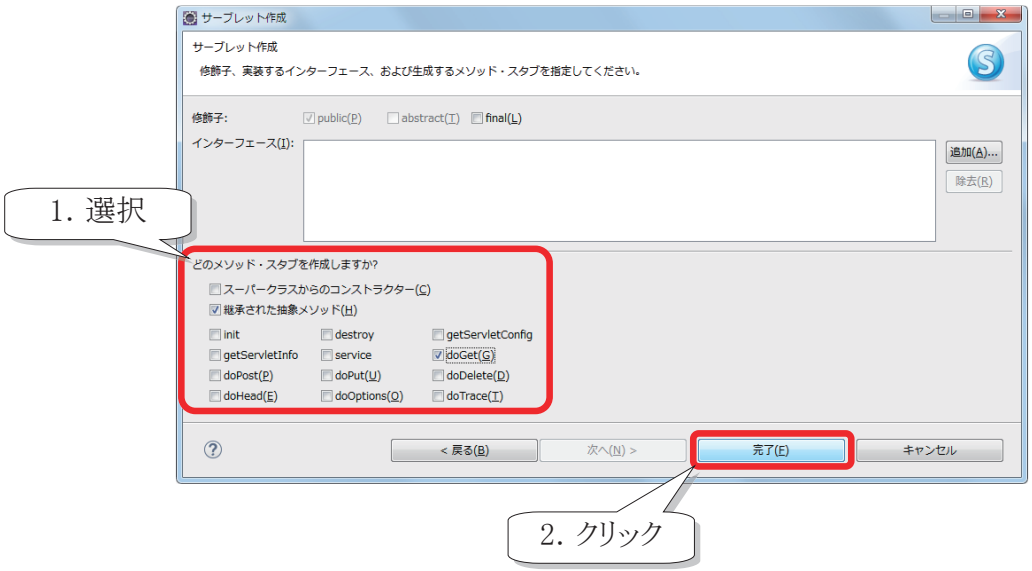
② 「Java パッケージ(K)」に適切なパッケージ名を入力（省略可）して，「クラス名(M)」に適切なクラス名を入力して，「スーパークラス(S)」に適切なスーパクラス名を入力して，「次へ(N) >」ボタンをクリックする。



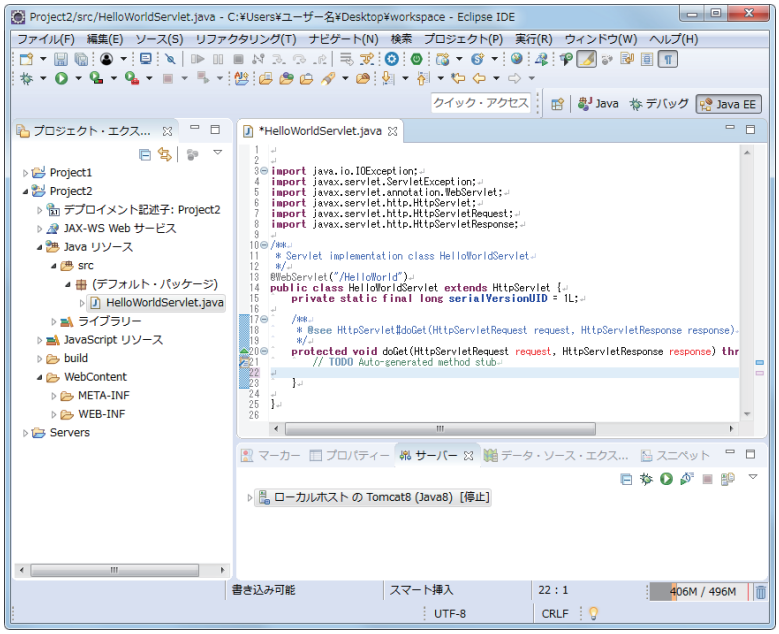
③　「名前(M)」に適切なサーブレット名を入力して，「URL マッピング(U)」の既存値を選択してから「編集(T)...」ボタンをクリックして，「パターン(P)」に適切なURLパターンを入力して，「OK」ボタンをクリックして，「次へ(N) >」ボタンをクリックする。



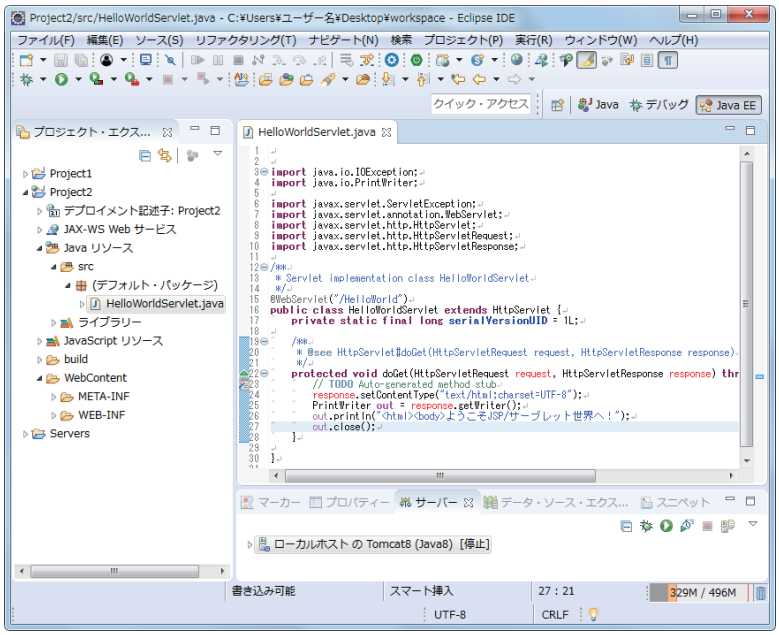
④　「スーパークラスからのコンストラクター(C)」を選択解除して，「doGet(G)」または「doPost(P)」を選択して，「完了(F)」ボタンをクックする。



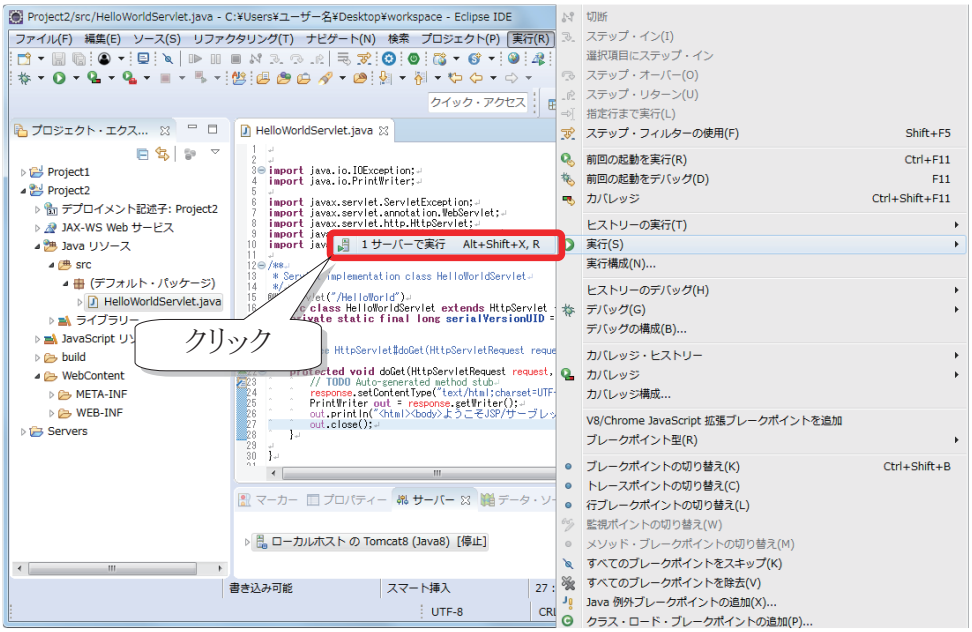
⑤　エディタ・ビューに表示されたクラス内に表示された不要なコードを削除する。



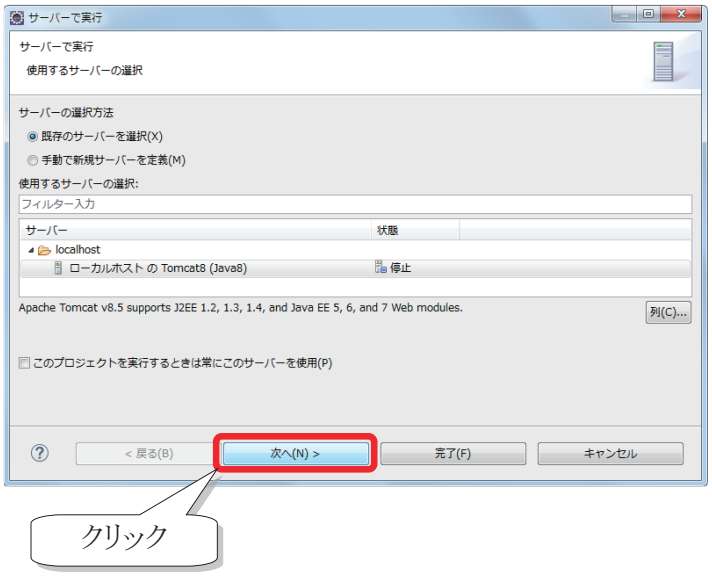
⑥　エディタ・ビューに表示されたクラス内に適切なソースコードを追加して，「ファイル(F)」メニューの「保存(S)」をクリックする。このとき，ソースコードがコンパイルされるので，問題・ビューに表示された内容を確認して，入力したソースコードを変更して，「ファイル(F)」メニューの「保存(S)」をクリックする。



⑦　「実行(R)」メニューの「実行(S)」をポイントして，「1 サーバーで実行」をクリックする。

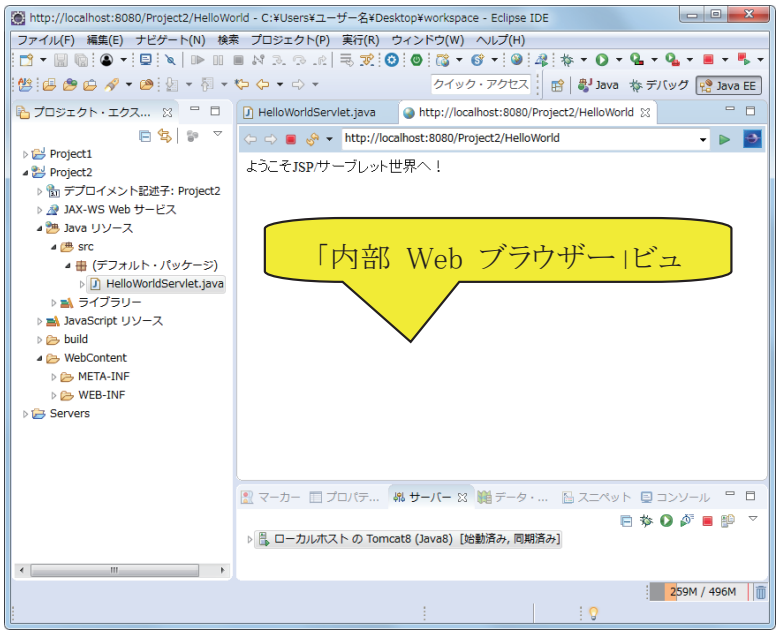


「次へ(N) >」ボタンをクリックする。

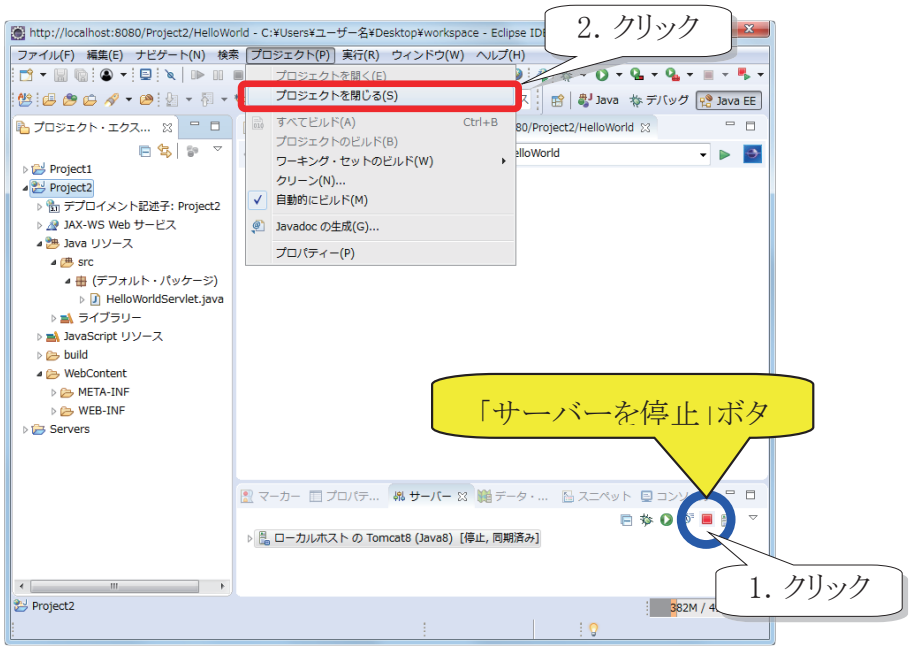


「完了(F)」ボタンをクリックする。

このとき，実行結果が「内部 Web ブラウザー」ビューに表示される。



⑧　サーブレットの開発が終わった場合は，「サーバーを停止」ボタンをクリックして，パッケージ・エクスプローラー・ビュー上のプロジェクト名をクリックした後に，「プロジェクト(P)」メニューの「プロジェクトを閉じる(S)」をクリックする。

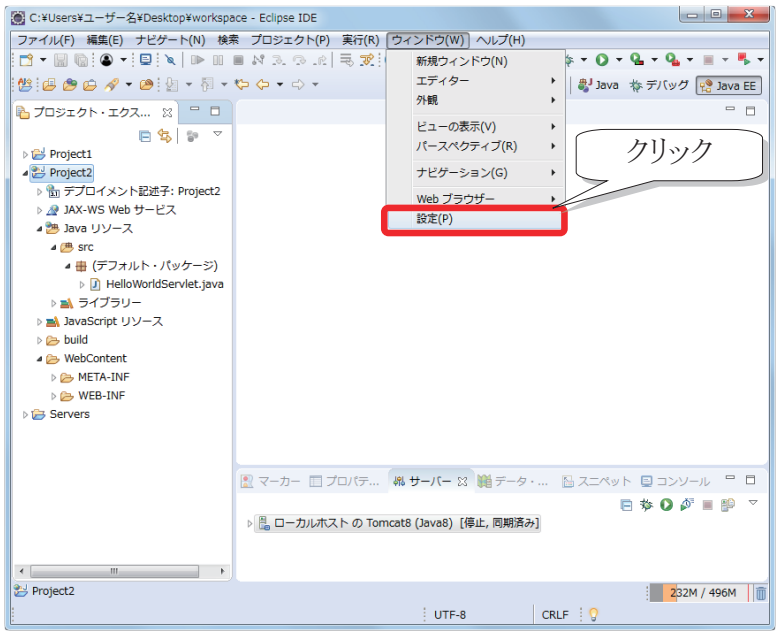


なお，当該プロジェクトのサーブレットを再実行する場合は，「プロジェクト(P)」メニューの「プロジェクトを開く(E)」をポイントする。

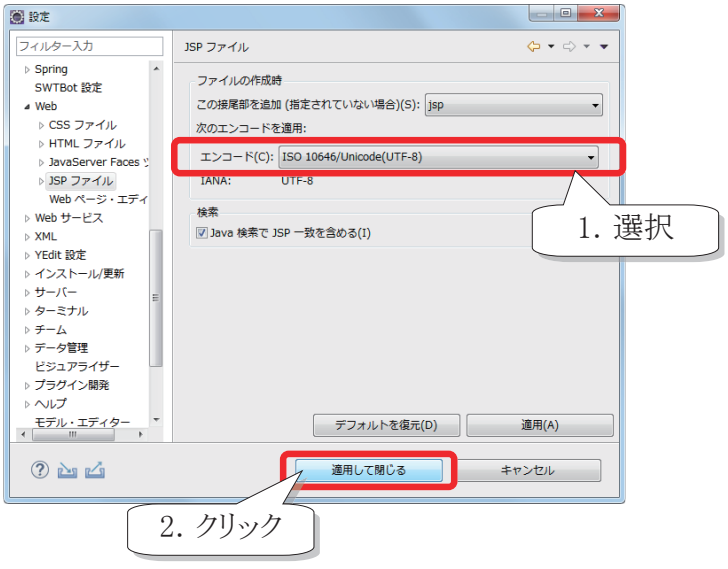
(3) JSPを作成します。次の①～⑩の順に進めてください。

① (1)の①～⑧を実行する。また，サーバー・ビュー内の「ローカル・ホストのTomcat8（Java8）」が未設定の場合は，(1)の⑨を実行する。

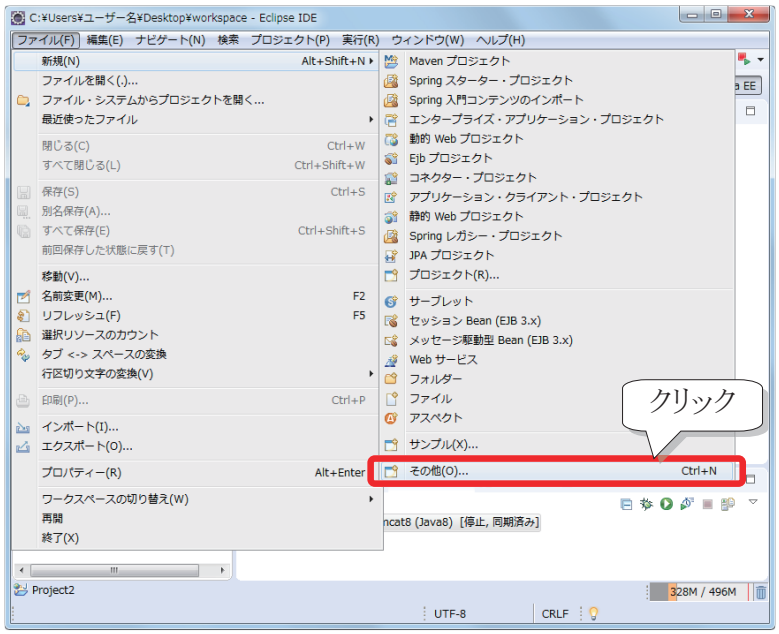
② 「ウィンドウ(W)」メニューの「設定(P)」をクリックする。



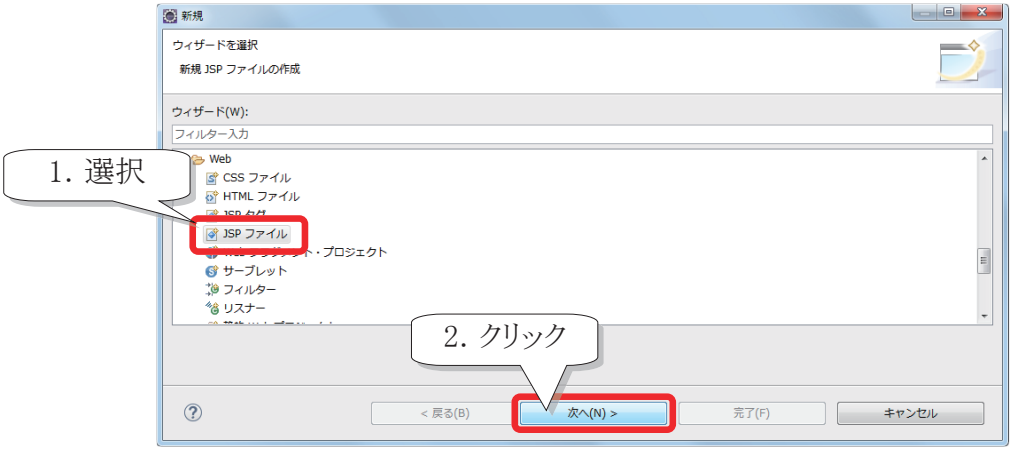
③ 「Web」の「JSPファイル」をクリックして，エンコードに「ISO 10646/Unicode (UTF-8)」を選択して，「適用して閉じる」ボタンをクリックする。



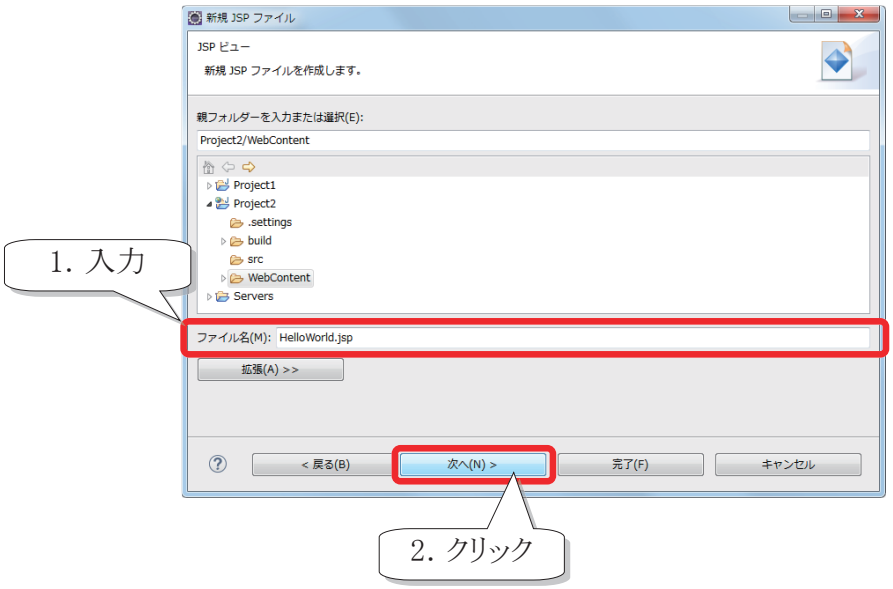
④　「ファイル(F)」メニューの「新規(N)」をポイントして，「その他(O)...」をクリックする。



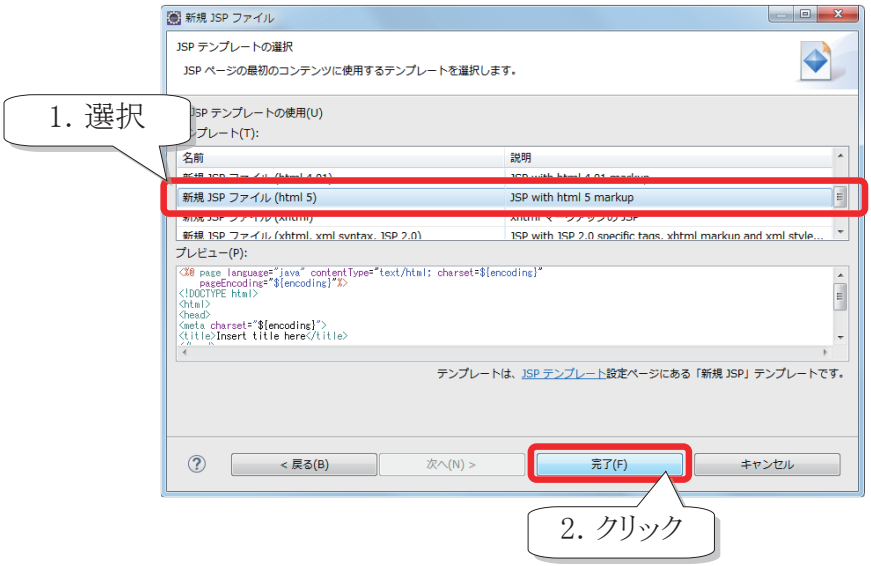
⑤　ウィザード一覧から「Web」フォルダをダブルクリックして開き，「JSPファイル」を選択して，「次へ(N) >」ボタンをクリックする。



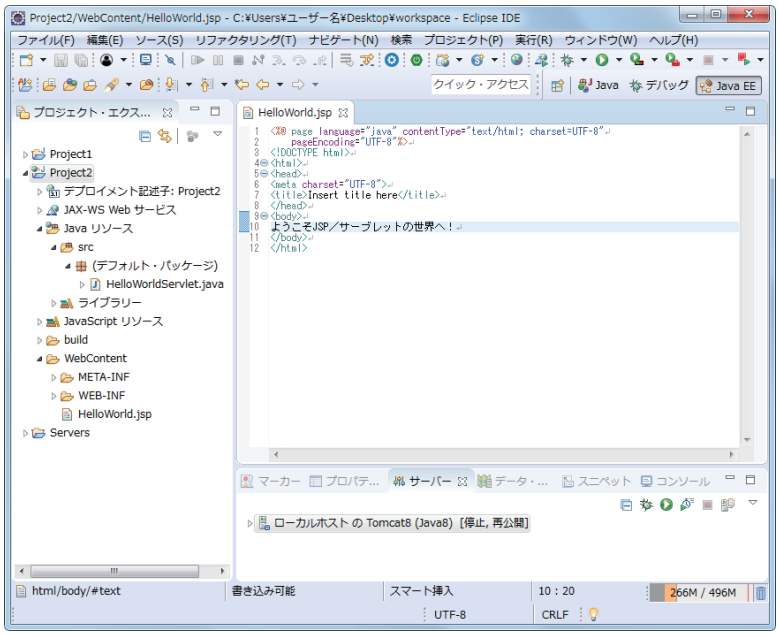
⑥　「ファイル名(M)」に適切なファイル名を入力して，「次へ(N) >」ボタンをクリックする。



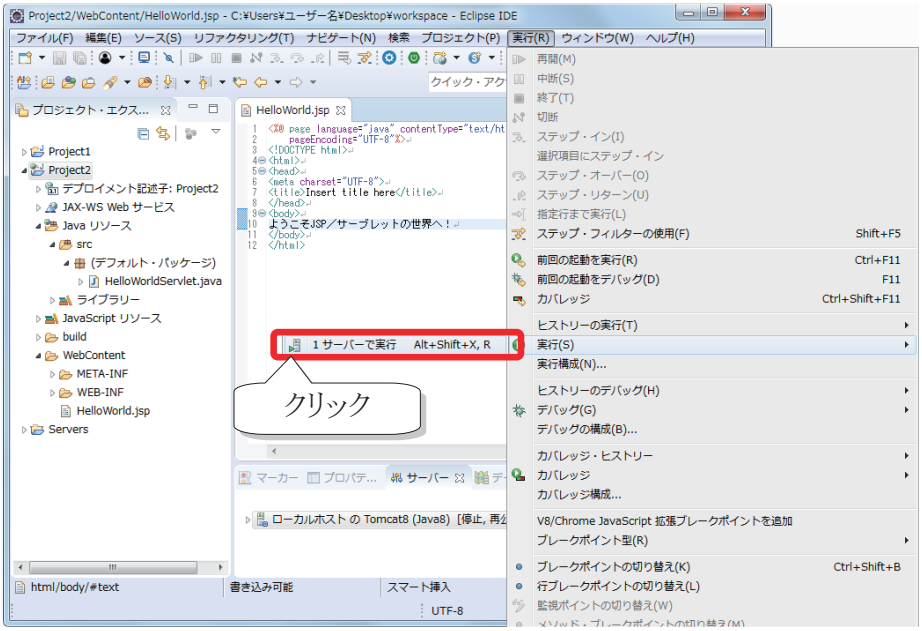
⑦　テンプレート一覧から「新規 JSP ファイル(html 5)」または「新規 JSP ファイル(xhtml)」を選択して，「完了(F)」ボタンをクリックする。



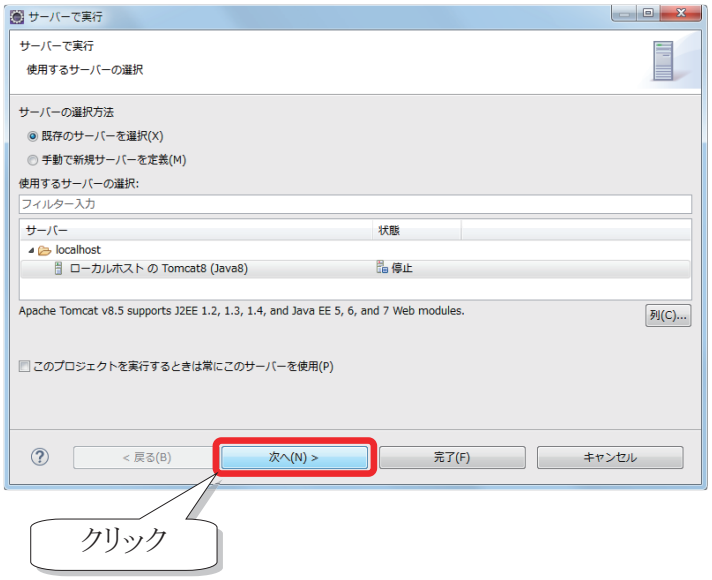
⑧　エディタ・ビューに表示されたHTMLまたはXHTMLタグ内に適切なコードを追加して，「ファイル(F)」メニューの「保存(S)」をクリックする。



⑨　「実行(R)」メニューの「実行(S)」をポイントして，「1 サーバーで実行」をクリックする。



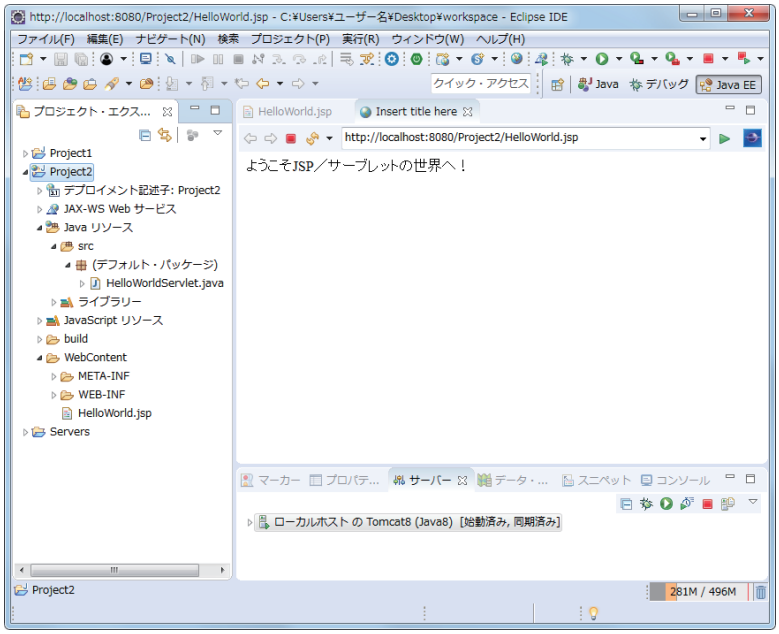
「次へ(N) >」ボタンをクリックする。



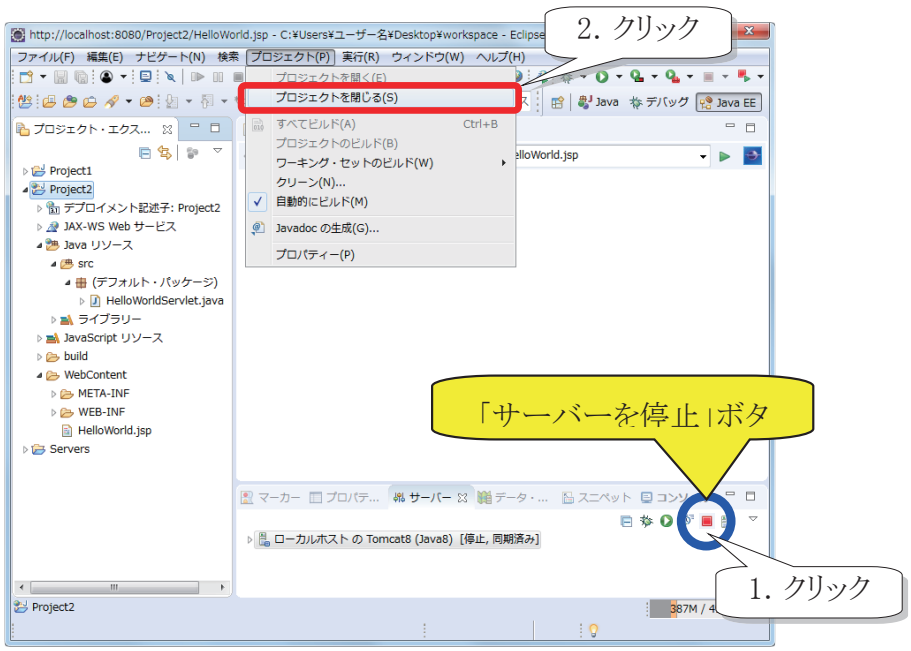
「完了(F)」ボタンをクリックする。

確認ダイアログ「サーバーを再起動しますか？」が表示された場合は，「サーバー再起動(S)」を選択して，「OK」ボタンをクリックする。

このとき，実行結果が「内部 Web ブラウザー」ビューに表示される。



⑩　JSPの開発が終わった場合は，「サーバーを停止」ボタンをクリックして，パッケージ・エクスプローラー・ビュー上のプロジェクト名をクリックした後に，「プロジェクト(P)」メニューの「プロジェクトを閉じる(S)」をクリックする。



なお，当該プロジェクトのJSPを再実行する場合は，「プロジェクト(P)」メニューの「プロジェクトを開く(E)」をクリックする。

付録２　Javaロギング入門

2-1　ロギングとロガー

アプリケーションが出力する運用情報は，一般的に「ログ（log)」と呼ばれます。ログには“日付”，“時刻”，“ログ発生元（クラス名，メソッド名)”，“ログレベル”，“メッセージ”などが含まれます。ログは「ログファイル」に蓄積して，システム監査などに使用します。

ログをログファイルに書き出すことを「ロギング（logging)」，ロギングを行うプログラムを「ロガー（logger)」といいます。

2-2　Loggerクラス

Java言語は，Loggerクラス（ロガー）が標準クラスライブラリとして提供しています。このクラスはjava.util.Loggerパッケージに定義されていますので，使用時はパッケージのインポートが必要です。

Loggerクラスが出力するログレベルには，次の７種類があります。

ログレベル			Loggerクラスのメソッド名
低	Level.FINEST	[詳細レベル(高)]	finest()メソッド
↕	Level.FINER	[詳細レベル(中)]	finer()メソッド
↕	Level.FINE	[詳細レベル(小)]	fine()メソッド
↕	Level.CONFIG	[設定]	config()メソッド
↕	Level.INFO	[情報]	info()メソッド
↕	Level.WARNING	[警告]	warning()メソッド
高	Level.SEVERE	[致命的]	severe()メソッド

Webコンテナ“Tomcat”には，ロギング設定ファイル“logging.properties”が配備されています。

ロギング設定ファイル　C:¥Program Files¥Tomcat8¥conf¥logging.properties

このファイル内でログレベルとコンソールハンドラをそれぞれ“INFO”に設定していますので，ログファイル“catalina.YYYY-MM-DD.log”にはログレベルが“Level.INFO”，“Level.WARNING”，“Level.SEVERE”のログのみ書き出されます。

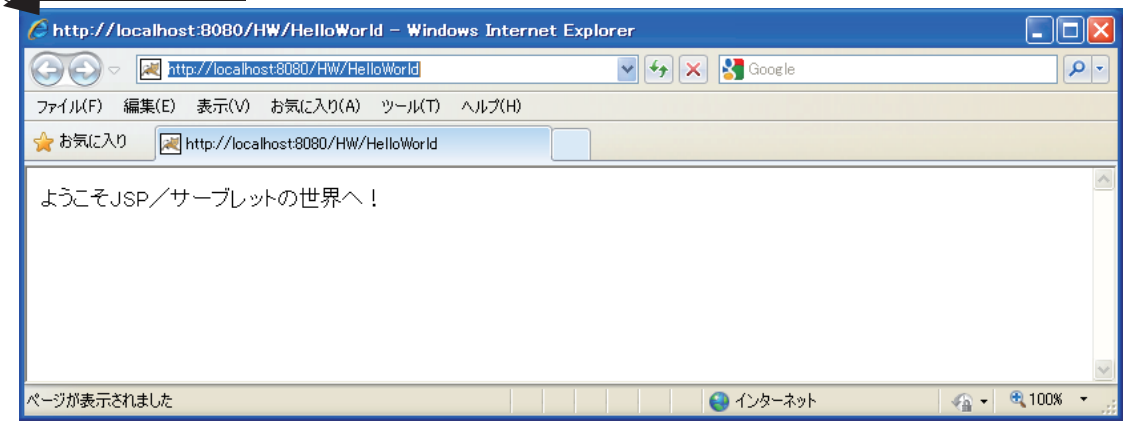
ログファイル　C:¥Program Files¥Tomcat8¥logs¥catalina.YYYY-MM-DD.log

実際に，ロギングを実装したプログラムを見てみましょう。

```
1  import  javax.servlet.*;
2  import  javax.servlet.annotation.*;
3  import  javax.servlet.http.*;
4  import  java.io.*;
5  import  java.util.logging.Logger;
6
7  @WebServlet("/HelloWorld")
8  public  class  HelloWorld  extends  HttpServlet
9  {
10     protected  void  doGet(
11         HttpServletRequest  req,
12         HttpServletResponse  res)
13         throws  ServletException, IOException
14     {
15         Logger logger
16             = Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);
17         String agent = req.getHeader("User-Agent");
18         logger.info(agent);
19
20         res.setContentType("text/html;charset=UTF-8");
21         PrintWriter out = res.getWriter();
22
23         out.println("<html>");
24         out.println("<body>");
25         out.println("ようこそJSP／サーブレットの世界へ！");
26         out.println("</body>");
27         out.println("</html>");
28
29         out.close();
30     }
31 }
```

・5行目は，java.util.loggingパッケージに定義されたLoggerクラスをインポートします。
・15，16行目のgetLogger()メソッドは，Loggerクラス型のインスタンスの参照を取得します。このメソッドは引数にスタティック変数GLOBAL_LOGGER_NAMEを指定しているので，簡易的にログを記録するローバルロガーを使用します。なお，Webアプリケーションごとにログを記録するには，個別にロガーを用意して使用します。
・17行目のgetHeader()メソッドは，WebブラウザからWebコンテナに送信されるリクエスト内のHTTPヘッダ情報“User-Agent（クライアントのWebブラウザなどの情報)”の値を文字列として返します。なお，このメソッド呼出しは，ロギングを実装したプログラムの必須コードではありません。
・18行目のinfo()メソッドは，引数に指定したメッセージ（文字列）をログレベル“INFO”でログファイルに記録します。

実行結果　(URL：http://localhost:8080/HW/HelloWorld)



[Tomcatコンソール]，[ログファイル“catalina.YYYY-MM-DD.log”]

```
5 16, 2012 5:23:25 午後 HelloWorld doGet
情報: Mozilla/4.0 (compatible; MSIE 6.0; Windows XP)
```

※表示フォーマット

日付（月　日，年），時刻（時：分：秒　午前／午後）　クラス名　メソッド名
ログレベル：Webブラウザ情報

Java言語では，Loggerクラスの他にApacheソフトウェア財団が開発した「Apache log4j」が有名です。これはorg.apache.log4jパッケージに定義されたLoggerクラスなので，別途クラスパスなどの設定が必要です。

付録３　利用者管理DBのデータ

“利用者マスタ”表

ID	パスワード	氏名
Duke	java	デューク
Ichiro	suzuki	イチロー

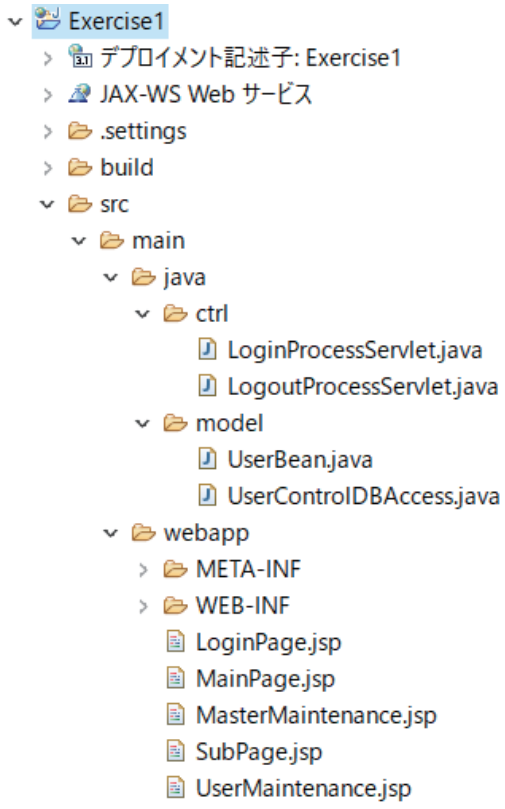
付録４　演習問題

演習課題-1　レベル ☆

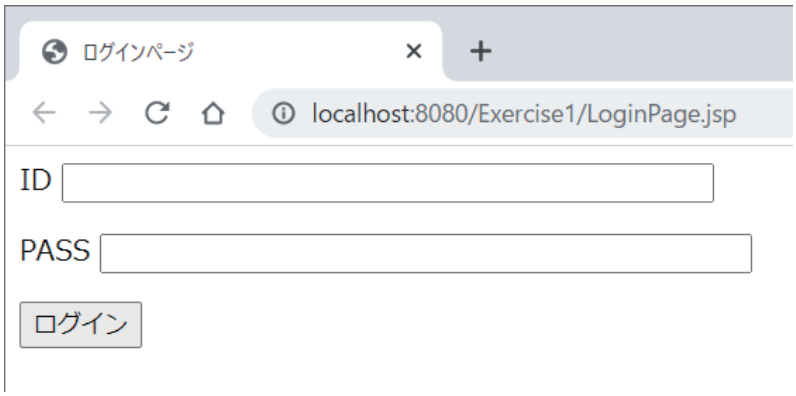
ログイン画面からのログイン処理を下記の要件を満たすように作成しなさい。

- 利用者マスタに登録されているID・パスワード（例：ID「Duke」パスワード「java」）でログインした場合は「情報表示メニュー」へ進む。
- ID「admin」パスワード「jsp」でログインした場合は「利用者マスタメンテメニュー」へ進む。また，ID「master」パスワード「servlet」でログインした場合は「マスタメンテメニュー」へ進む。
- 必ずログイン失敗の処理も作成すること。
- プログラムは必ずMVCモデル（フロントコントローラパターン）を採用すること。
- ViewはJSPで作成すること。
- 画面遷移は必ずフォワードを使用すること。
- JSPの画面レイアウト等は実行例を参考に作成すること。
- ログイン情報の保持には通信セション（有効時間 15分）を使用すること。
- プログラム名は自由（下図は参考）とする。分かりやすく簡潔な名称とすること。

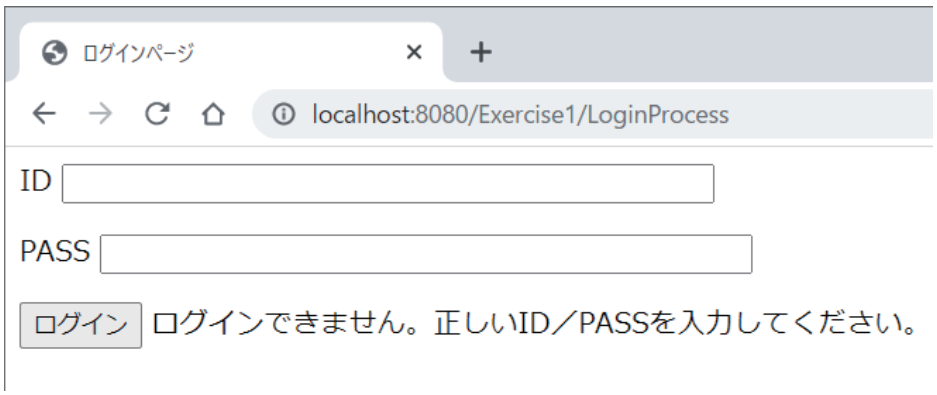
［ファイルの保存場所］



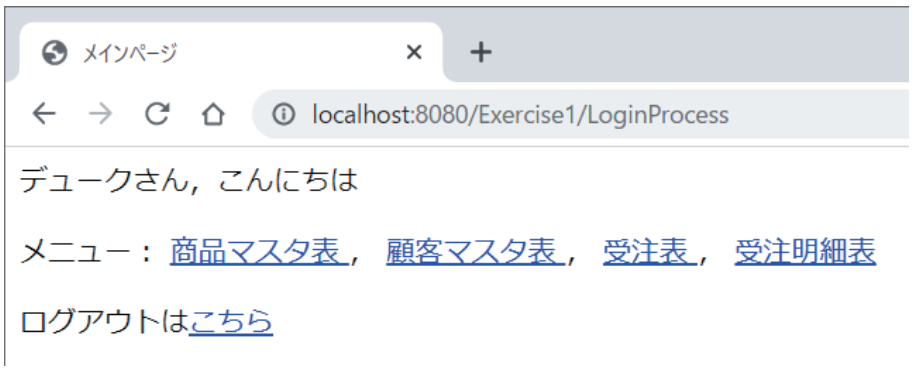
実行結果　(URL：http://localhost:8080/Exercise1/LoginPage.jsp)



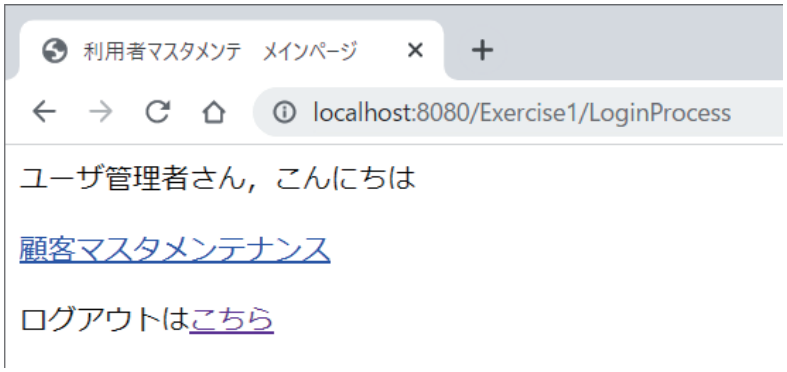
(ログイン画面)



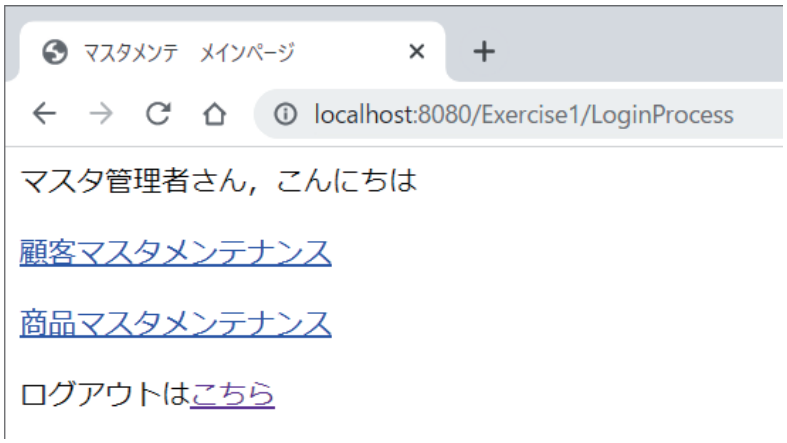
(ログイン失敗)



(ID：Duke，パスワード：java　でログイン時)



（ID：admin，パスワード：jsp　でログイン時）

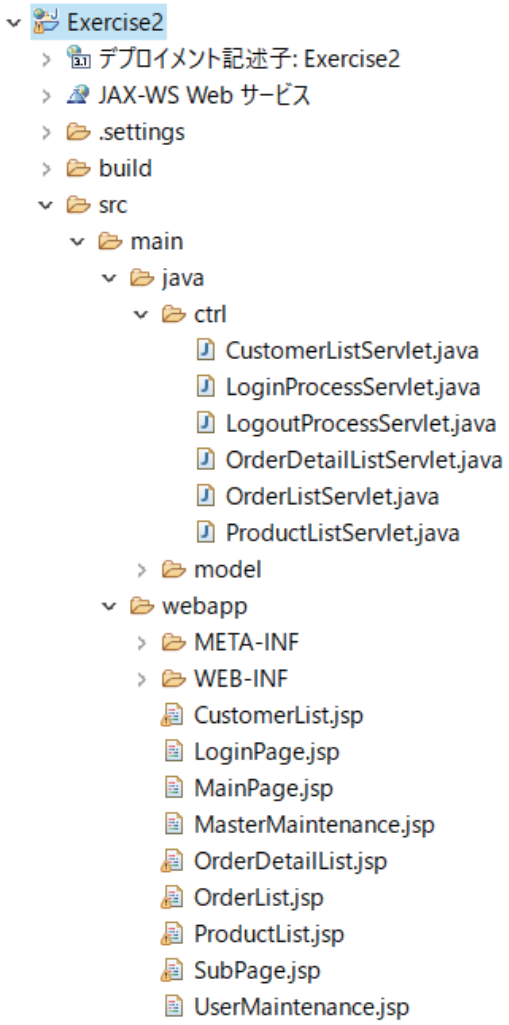


（ID：master，パスワード：servlet　でログイン時）

演習課題-2　レベル ☆

利用者マスタに登録されているID・パスワード（例：ID「Duke」パスワード「java」）でログインした後の画面で，実行結果を参考にして「商品マスタ表」「顧客マスタ表」「受注表」「受注明細表」を表示させるプログラムを作成しなさい。なお，それぞれの表はすべてのフィールドを表示させること。

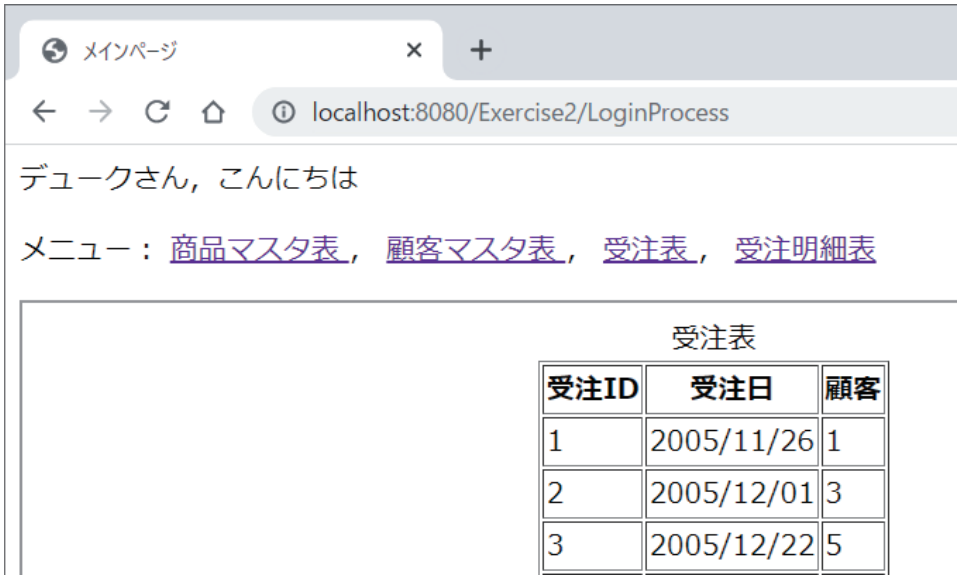
［ファイルの保存場所］



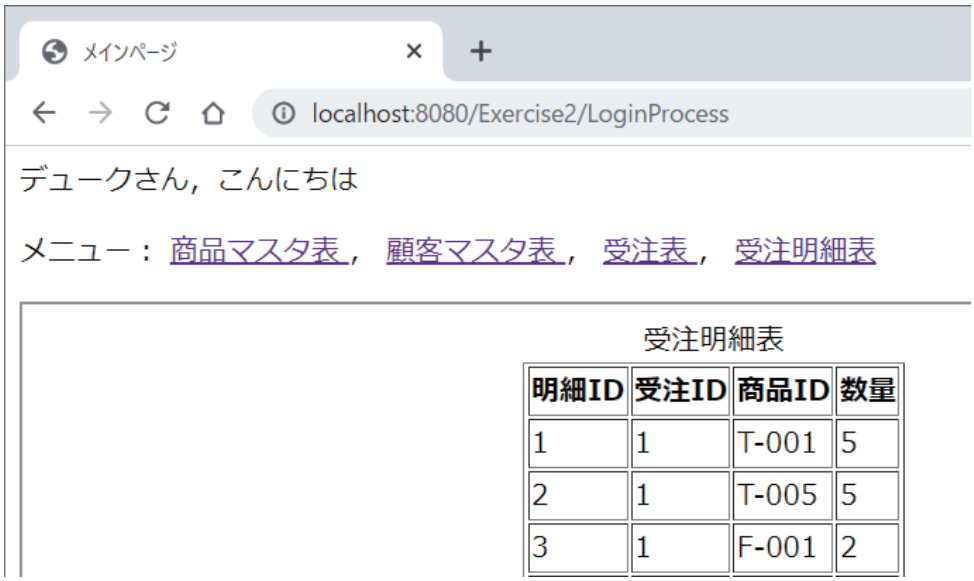
実行結果　(URL：http://localhost:8080/Exercise2/LoginPage.jsp)

（「商品マスタ表」クリック時）

（「顧客マスタ表」クリック時）



（「受注表」クリック時）

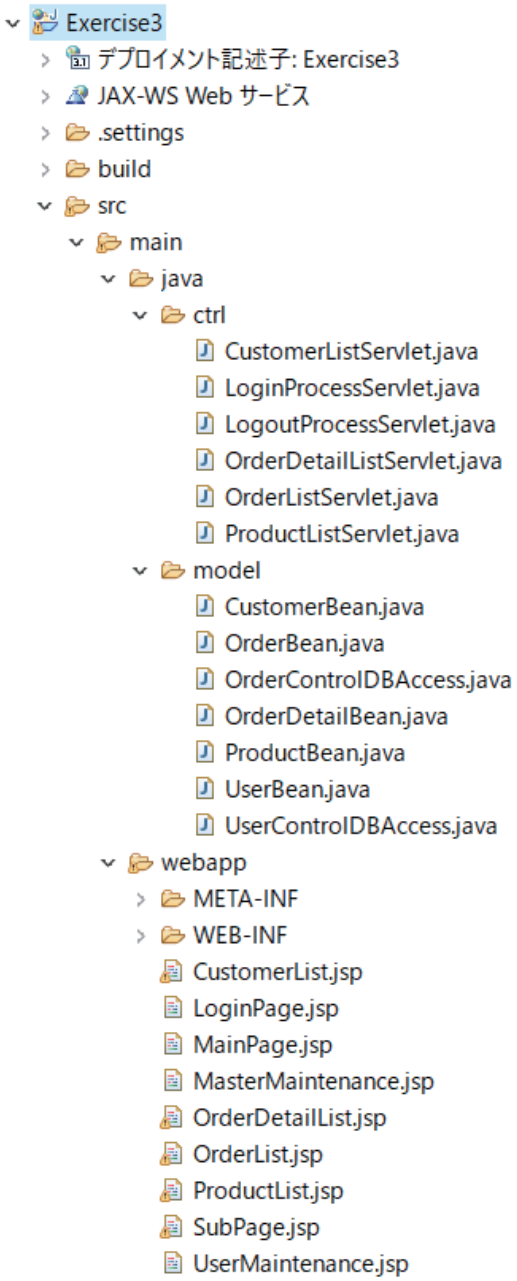


（「受注明細表」クリック時）

演習課題-3　レベル ☆☆

顧客マスタ表について，実行結果を参考にして指定した都道府県のみを表示するように変更しなさい。

［ファイルの保存場所］



実行結果　(URL：http://localhost:8080/Exercise3/LoginPage.jsp)

(都道府県テキストボックスに「東京都」と入力し，抽出ボタンをクリックした状態)

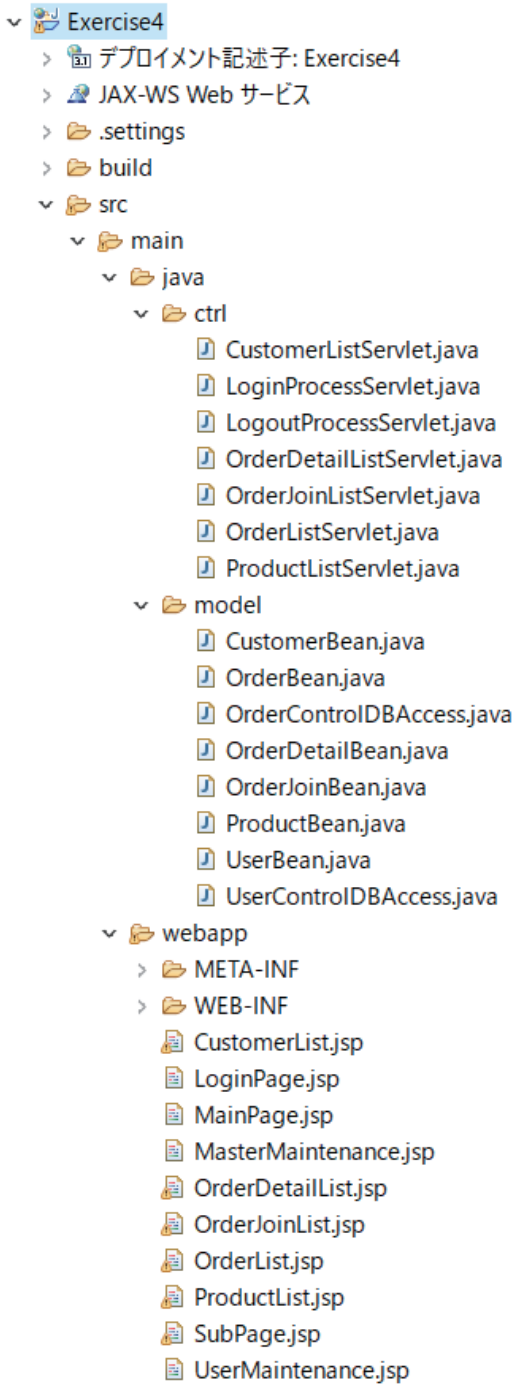
演習課題-4　レベル ☆☆☆

「顧客マスタ表」「商品マスタ表」「受注表」「受注明細表」をキー項目で結合した表を作成しなさい。表示する項目と順番は下の通りとする。なお，「受注ID」「明細ID」の優先順位で昇順に並べること。また，テーブルの項目に存在しない項目は何らかの方法で作成すること。消費税は８%とする。

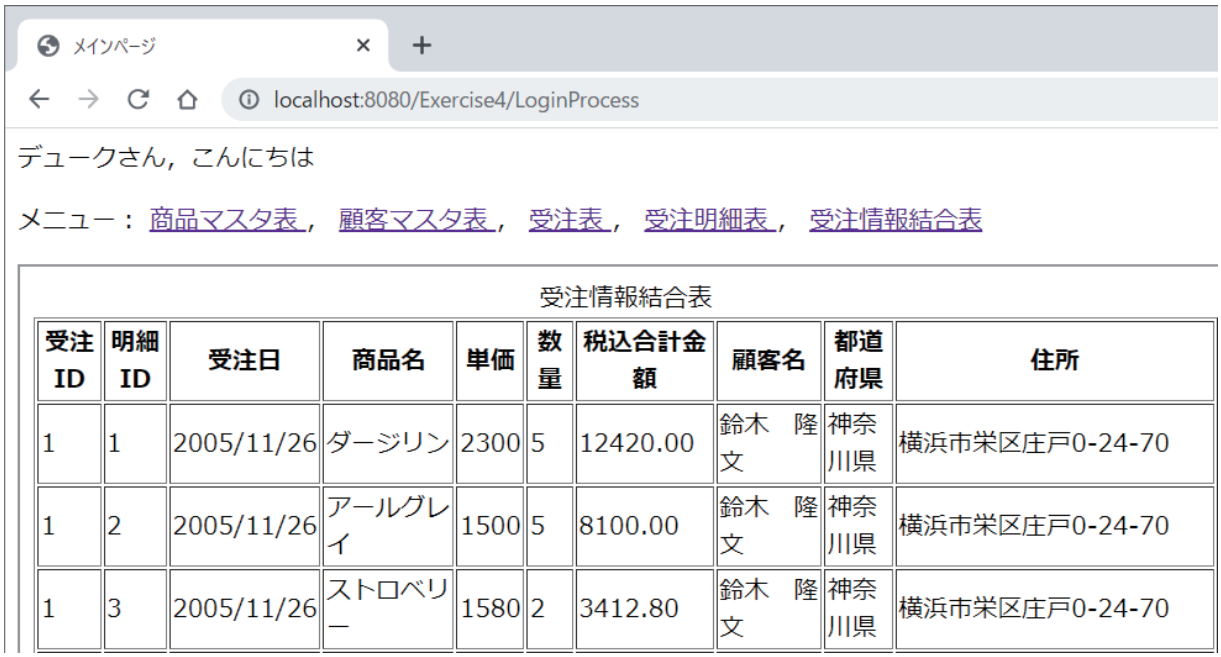
【表示項目】　「受注ID」「明細ID」「受注日」「商品名」「単価」「数量」「税込合計金額」「顧客名」「都道府県」「住所」

なお，表示については，利用者マスタに登録されているID・パスワード（例：ID「Duke」パスワード「java」）でログインした後の画面で，実行結果を参考にして表示すること。

［ファイルの保存場所］


Exercise4
デプロイメント記述子: Exercise4
JAX-WS Web サービス
.settings
build
src
main
java
ctrl
CustomerListServlet.java
LoginProcessServlet.java
LogoutProcessServlet.java
OrderDetailListServlet.java
OrderJoinListServlet.java
OrderListServlet.java
ProductListServlet.java
model
CustomerBean.java
OrderBean.java
OrderControlDBAccess.java
OrderDetailBean.java
OrderJoinBean.java
ProductBean.java
UserBean.java
UserControlDBAccess.java
webapp
META-INF
WEB-INF
CustomerList.jsp
LoginPage.jsp
MainPage.jsp
MasterMaintenance.jsp
OrderDetailList.jsp
OrderJoinList.jsp
OrderList.jsp
ProductList.jsp
SubPage.jsp
UserMaintenance.jsp

実行結果　(URL：http://localhost:8080/Exercise4/LoginPage.jsp)



受注情報結合表

受注ID	明細ID	受注日	商品名	単価	数量	税込合計金額	顧客名	都道府県	住所
1	1	2005/11/26	ダージリン	2300	5	12420.00	鈴木　隆文	神奈川県	横浜市栄区庄戸0-24-70
1	2	2005/11/26	アールグレイ	1500	5	8100.00	鈴木　隆文	神奈川県	横浜市栄区庄戸0-24-70
1	3	2005/11/26	ストロベリー	1580	2	3412.80	鈴木　隆文	神奈川県	横浜市栄区庄戸0-24-70

演習課題-5　レベル ☆☆

利用者マスタに登録されているID・パスワード（例：ID「Duke」パスワード「java」）でログインした後の画面で，「さん，こんにちは」と表示されている箇所のメッセージを，現在時刻に応じて変更する処理を作成しなさい。メッセージのJSPでの表示にはEL式を使用すること。

表示するメッセージの作成には“Message.properties”ファイルを必ず使用すること。なおMessage.propertyファイルは読み込みを行うJavaサーブレットと同じパッケージに配置すること。

ヒント　（ファイル名：Message.properties）

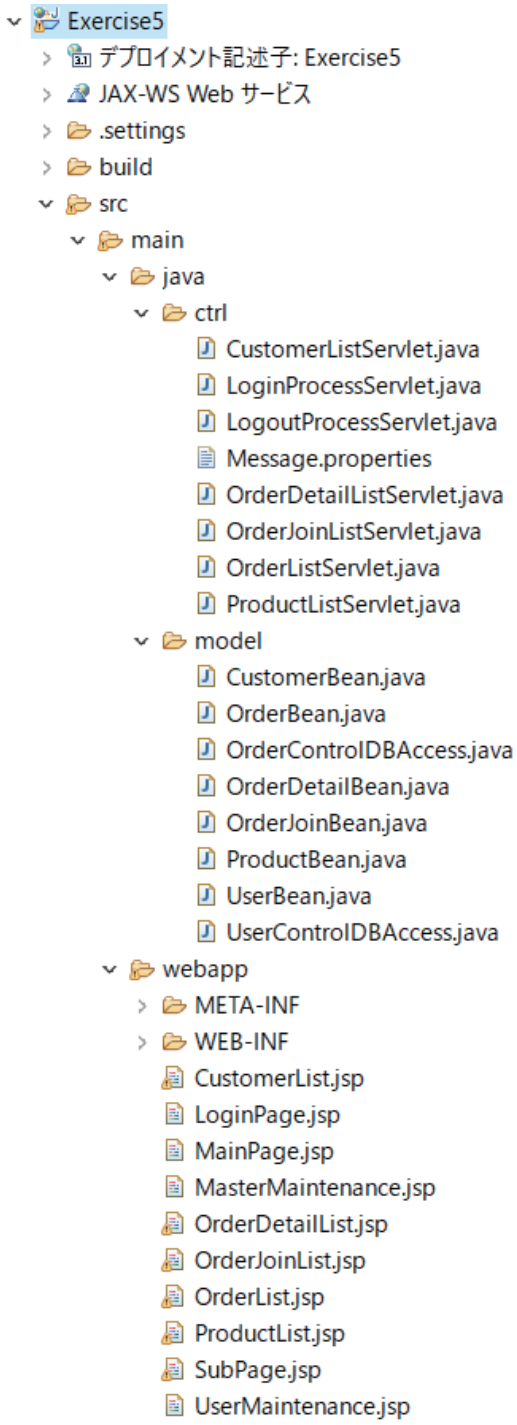
LateNight=さん，こんな深夜に本当にお疲れ様です。
EarlyMorning=さん，お早いですね。
Morning=さん，おはようございます。
AfterNoon=さん，午後もがんばっていきましょう。
Evening=さん，もう夕方です。もうひと頑張りですね。
Night=さん，今日も一日お疲れ様です。

時刻とメッセージの対応は次の通りとする。

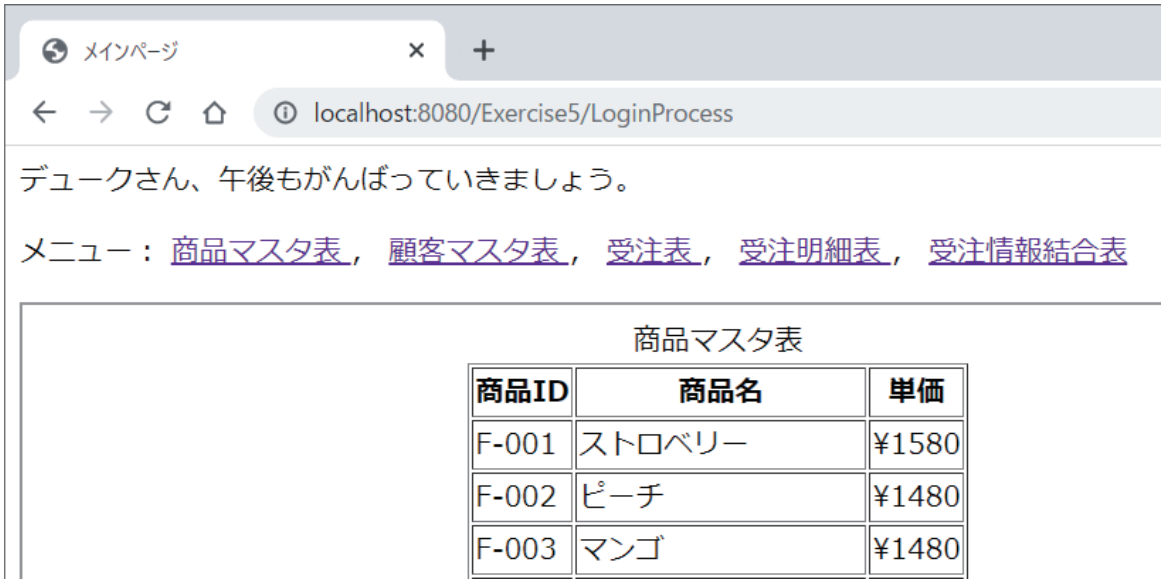
0:00～3:59　：　LateNight
4:00～7:59　：　EarlyMorning
8:00～11:59　：　Morning
12:00～15:59　：　AfterNoon
16:00～19:59　：　Evening
20:00～23:59　：　Night

なお，現在時刻の取得には“Date”オブジェクトを利用し，時刻のみを求めるには“SimpleFormat”クラスを利用する。それぞれの利用方法はAPIドキュメントやインターネット等で調べること。それ以外にも必要なオブジェクトやクラスがあれば，自分で調べて使用しても良い。

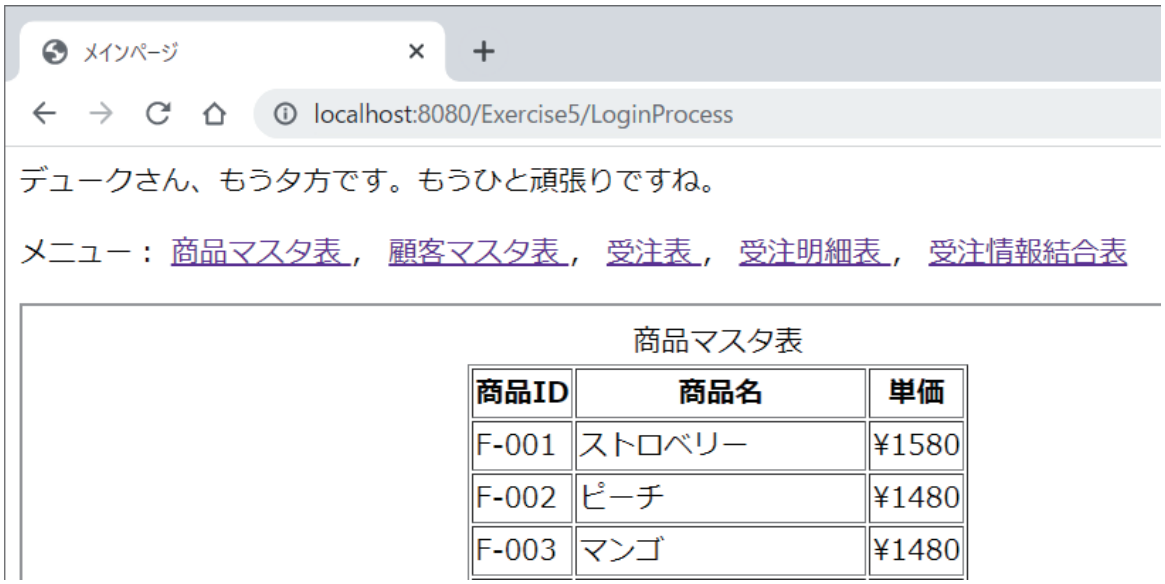
［ファイルの保存場所］


Exercise5
デプロイメント記述子: Exercise5
JAX-WS Web サービス
.settings
build
src
main
java
ctrl
CustomerListServlet.java
LoginProcessServlet.java
LogoutProcessServlet.java
Message.properties
OrderDetailListServlet.java
OrderJoinListServlet.java
OrderListServlet.java
ProductListServlet.java
model
CustomerBean.java
OrderBean.java
OrderControlDBAccess.java
OrderDetailBean.java
OrderJoinBean.java
ProductBean.java
UserBean.java
UserControlDBAccess.java
webapp
META-INF
WEB-INF
CustomerList.jsp
LoginPage.jsp
MainPage.jsp
MasterMaintenance.jsp
OrderDetailList.jsp
OrderJoinList.jsp
OrderList.jsp
ProductList.jsp
SubPage.jsp
UserMaintenance.jsp

実行結果　(URL：http://localhost:8080/Exercise5/LoginPage.jsp)



(12:00～15:59に実行した場合)



(16:00～19:59に実行した場合)

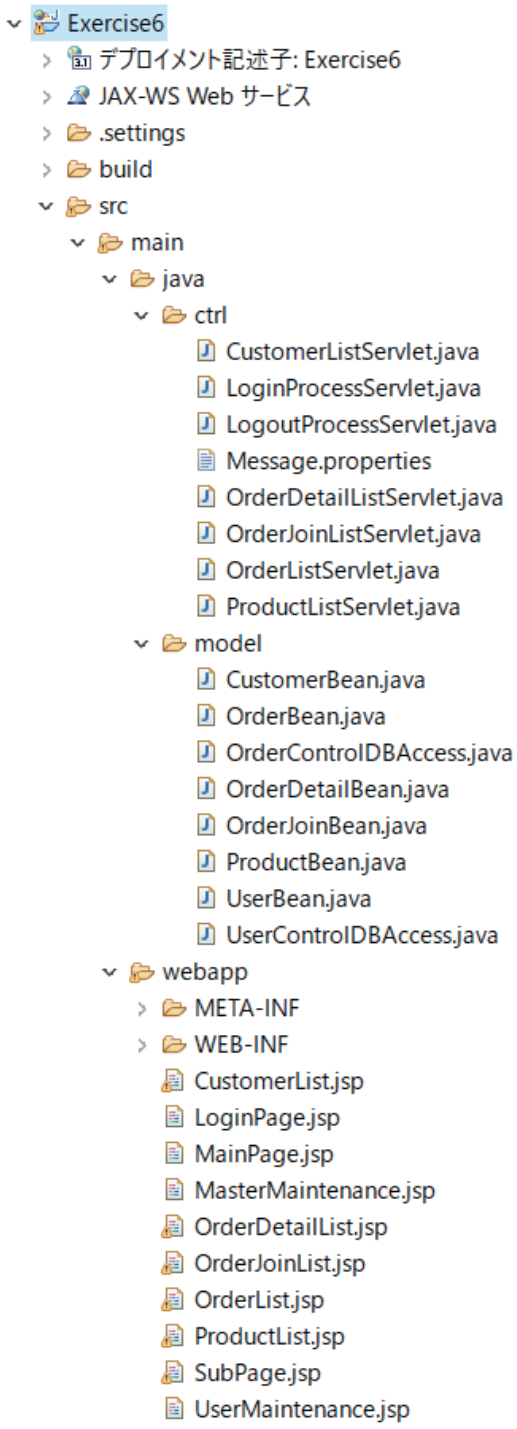
演習課題-6　レベル ☆☆☆

演習課題-5で作成したプログラムについて，「受注日」「商品名」「顧客名」「都道府県」の各情報で検索する処理を作成しなさい。検索条件を入力しなかった項目は，抽出条件の対象外とする（その項目については全件抽出される)。

ヒント　（例：WHERE条件を指定しながら、全件抽出されるSQL文）

```
SELECT　都道府県　FROM　顧客マスタ　WHERE　都道府県　LIKE　‘%’;
```

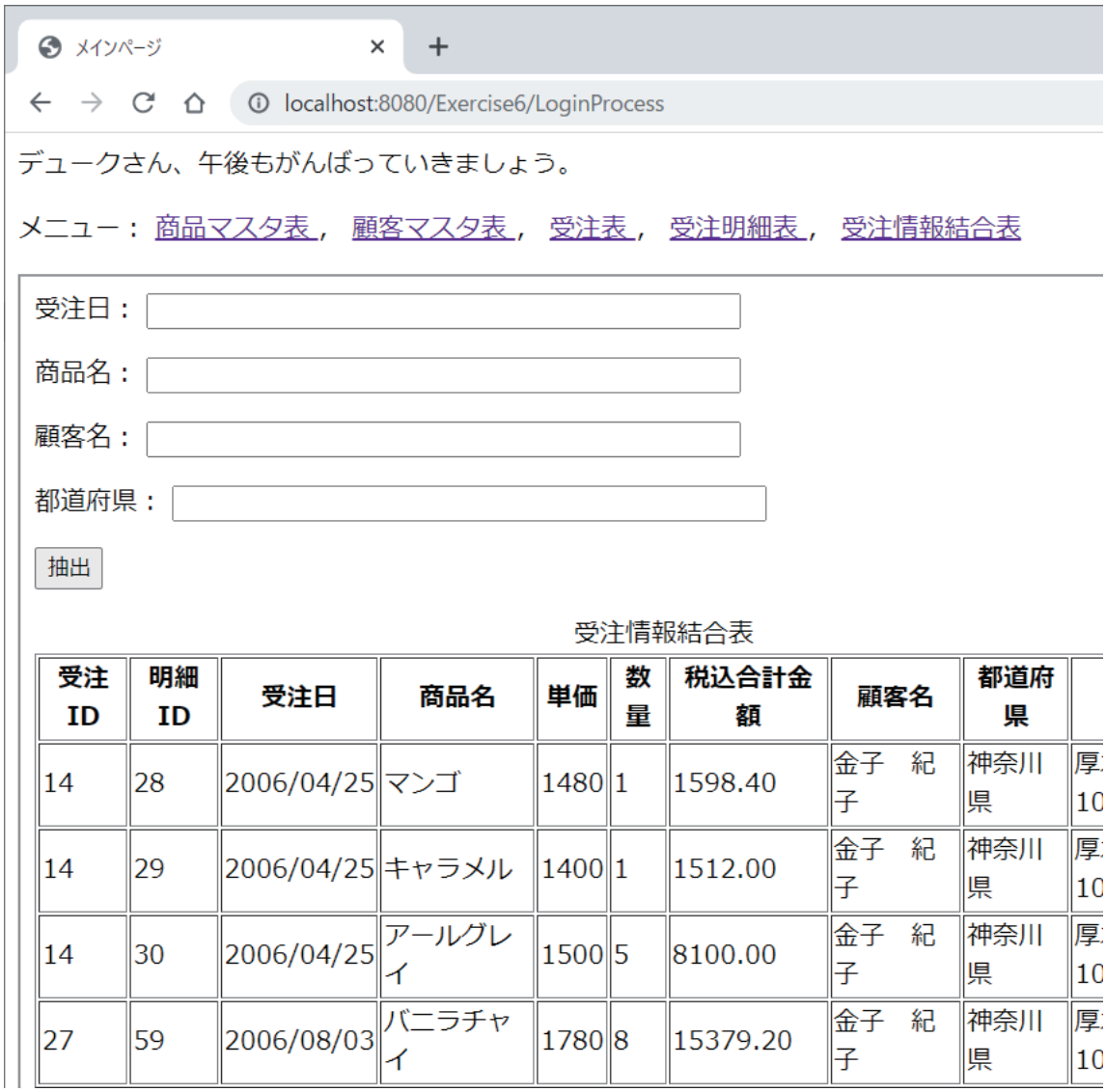
［ファイルの保存場所］


Exercise6
デプロイメント記述子: Exercise6
JAX-WS Web サービス
.settings
build
src
main
java
ctrl
CustomerListServlet.java
LoginProcessServlet.java
LogoutProcessServlet.java
Message.properties
OrderDetailListServlet.java
OrderJoinListServlet.java
OrderListServlet.java
ProductListServlet.java
model
CustomerBean.java
OrderBean.java
OrderControlDBAccess.java
OrderDetailBean.java
OrderJoinBean.java
ProductBean.java
UserBean.java
UserControlDBAccess.java
webapp
META-INF
WEB-INF
CustomerList.jsp
LoginPage.jsp
MainPage.jsp
MasterMaintenance.jsp
OrderDetailList.jsp
OrderJoinList.jsp
OrderList.jsp
ProductList.jsp
SubPage.jsp
UserMaintenance.jsp

実行結果　(URL：http://localhost:8080/Exercise6/LoginPage.jsp)

受注ID	明細ID	受注日	商品名	単価	数量	税込合計金額	顧客名	都道府県	
3	9	2005/12/22	ダージリン	2300	3	7452.00	小島　千里	東京都	目
8	20	2006/02/10	ダージリン	2300	6	14904.00	小林　和佳子	東京都	新
16	32	2006/05/17	ダージリン	2300	1	2484.00	宮　正澄	東京都	江
23	49	2006/07/11	ダージリン	2300	3	7452.00	佐藤　美佑	東京都	世 20
31	68	2006/08/30	ダージリン	2300	12	29808.00	宮　正澄	東京都	江

(商品名：ダージリン　都道府県：東京都　を入力して抽出した結果)



受注ID	明細ID	受注日	商品名	単価	数量	税込合計金額	顧客名	都道府県	
14	28	2006/04/25	マンゴ	1480	1	1598.40	金子　紀子	神奈川県	厚 10
14	29	2006/04/25	キャラメル	1400	1	1512.00	金子　紀子	神奈川県	厚 10
14	30	2006/04/25	アールグレイ	1500	5	8100.00	金子　紀子	神奈川県	厚 10
27	59	2006/08/03	バニラチャイ	1780	8	15379.20	金子　紀子	神奈川県	厚 10

（顧客名：金子　紀子　を入力して抽出した結果）

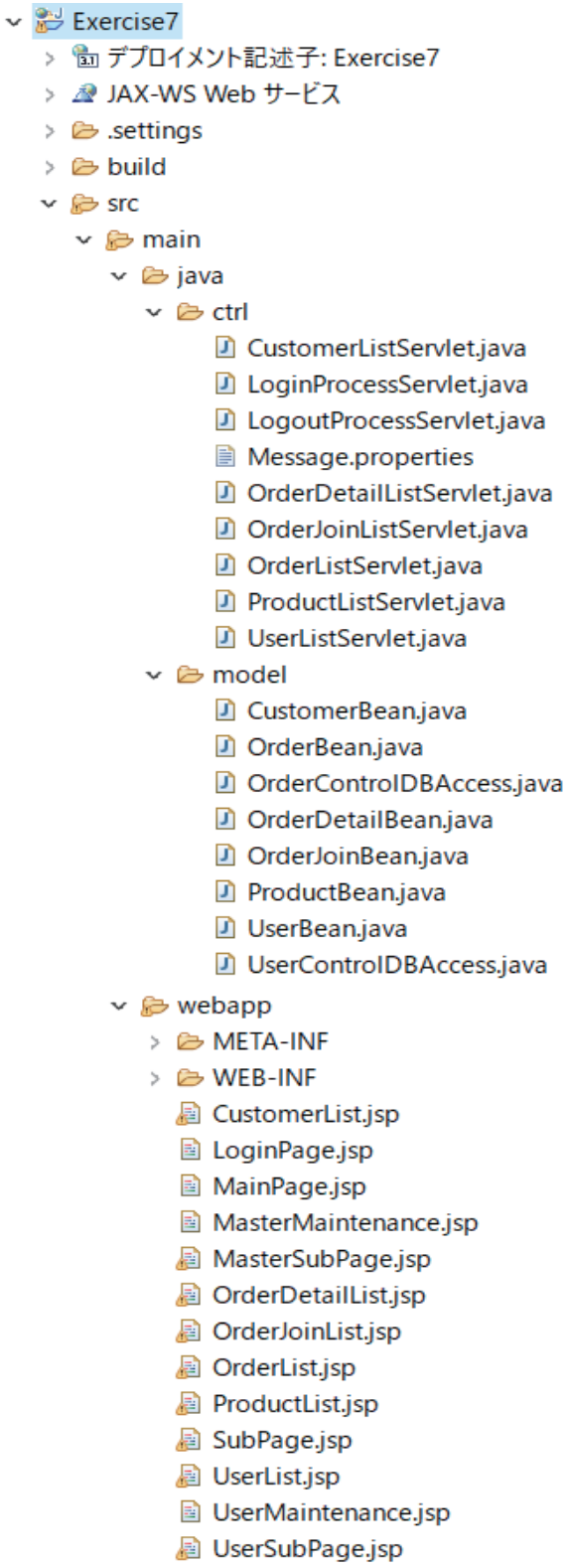
演習課題-7　レベル ☆☆☆☆

「顧客マスタ表」「商品マスタ表」「利用者マスタ表」の各表について，データ追加機能のプログラムを作成しなさい。ただし,「利用者マスタ表」の追加を行えるのは,「ID：admin　パスワード：jsp」でログインした場合,「顧客マスタ表」「商品マスタ表」のメンテナンスを行えるのは,「ID：master　パスワード：servlet」でログインした場合とする（演習課題-1で作成済み)。課題４で作成した顧客マスタの検索機能は削除しないこと。

ヒント　（更新SQLを実行させるプログラム（抜粋））

```
stmt.executeUpdate();  //更新SQL実行はexecuteUpdate。
```

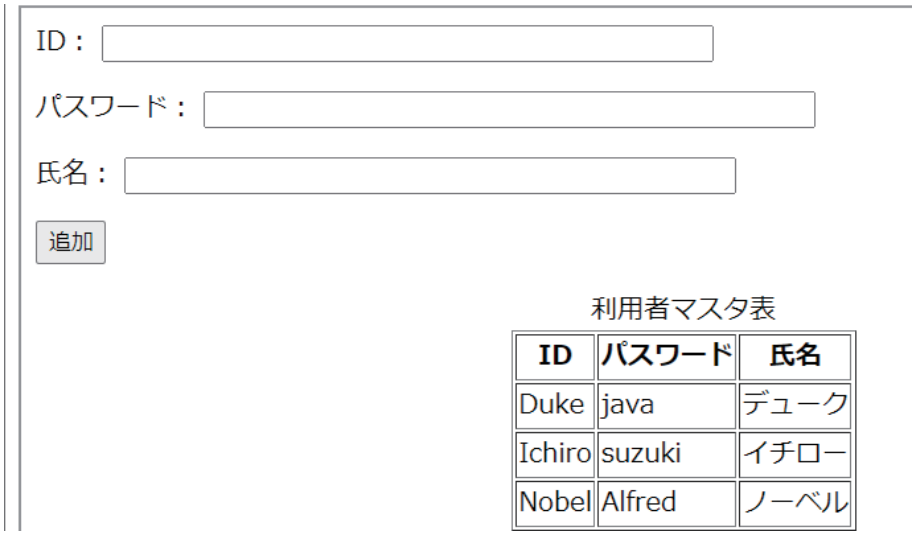
［ファイルの保存場所］


Exercise7
デプロイメント記述子: Exercise7
JAX-WS Web サービス
.settings
build
src
main
java
ctrl
CustomerListServlet.java
LoginProcessServlet.java
LogoutProcessServlet.java
Message.properties
OrderDetailListServlet.java
OrderJoinListServlet.java
OrderListServlet.java
ProductListServlet.java
UserListServlet.java
model
CustomerBean.java
OrderBean.java
OrderControlDBAccess.java
OrderDetailBean.java
OrderJoinBean.java
ProductBean.java
UserBean.java
UserControlDBAccess.java
webapp
META-INF
WEB-INF
CustomerList.jsp
LoginPage.jsp
MainPage.jsp
MasterMaintenance.jsp
MasterSubPage.jsp
OrderDetailList.jsp
OrderJoinList.jsp
OrderList.jsp
ProductList.jsp
SubPage.jsp
UserList.jsp
UserMaintenance.jsp
UserSubPage.jsp

実行結果　(URL：http://localhost:8080/Exercise7/LoginPage.jsp)

（ログイン画面で　ID：admin　パスワード：jsp　と入力してログイン後に，
追加情報を入力している状態）

↓



（実行結果）

都道府県指定抽出

都道府県：

抽出

顧客新規追加

顧客ID：21

顧客名：山田　太郎

フリガナ：ヤマダ　タロウ

郵便番号：5300001

都道府県：大阪府

住所：大阪市北区梅田1-1-100

電話番号：06-6341-0000

追加

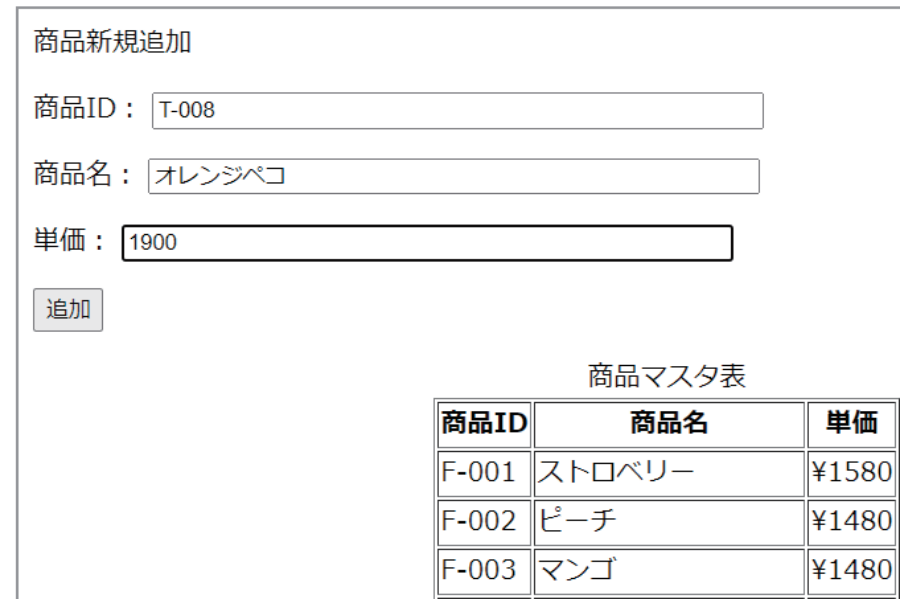
（ログイン画面でID：master　パスワード：servlet　でログインし，顧客マスタメンテナンスをクリックし，追加情報を入力している状態）

↓

20	金子　紀子	カネコ　ノリコ	2430035	神奈川県	厚木市愛甲036-10	045-333-0000
21	山田　太郎	ヤマダ　タロウ	5300001	大阪府	大阪市北区梅田1-1-100	06-6341-0000

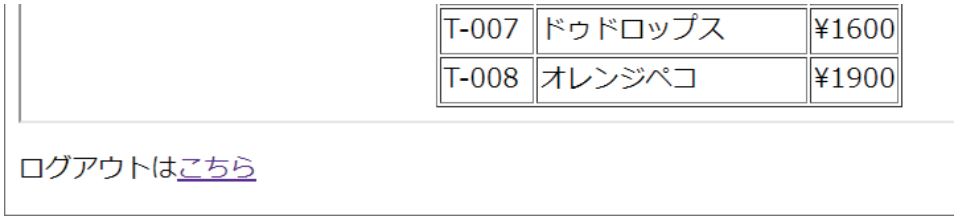
ログアウトはこちら

（実行結果）



（ログイン画面でID：master　パスワード：servlet　でログインし，商品マスタメンテナンスをクリックし，追加情報を入力している状態）

↓



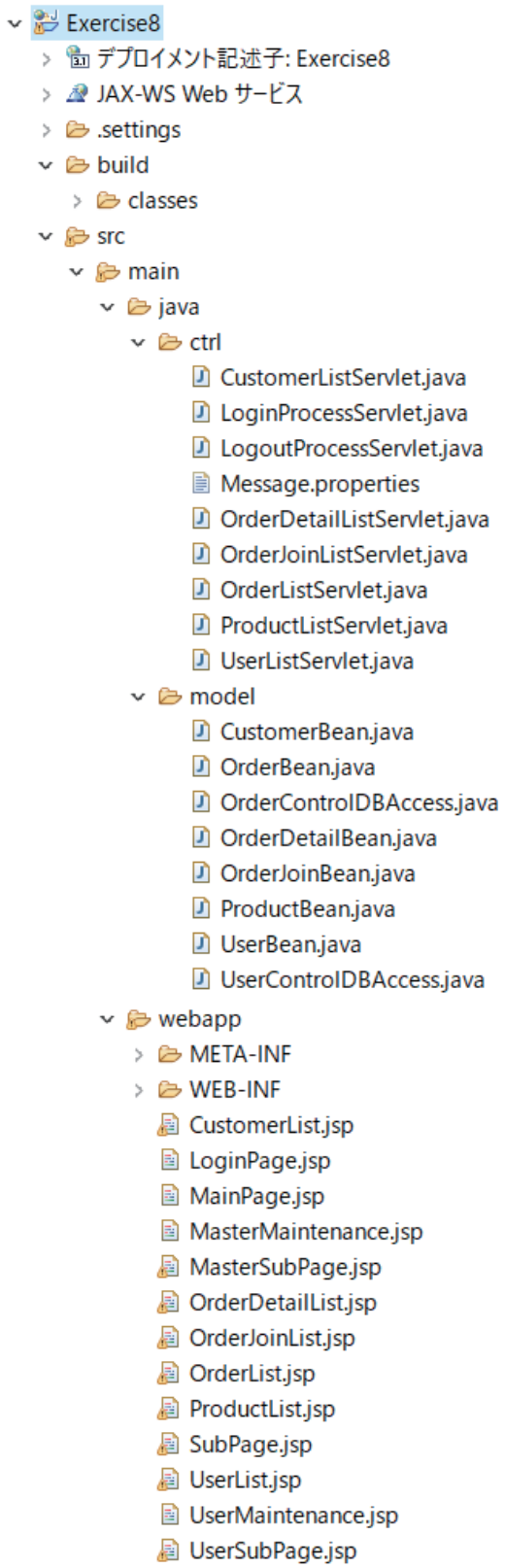
（実行結果）

演習課題-8　レベル ☆☆☆☆

「顧客マスタ表」「商品マスタ表」「利用者マスタ表」の各表について，データ削除機能のプログラムを作成しなさい。ただし,「利用者マスタ表」の削除を行えるのは,「ID：admin　パスワード：jsp」でログインした場合,「顧客マスタ表」「商品マスタ表」のメンテナンスを行えるのは,「ID：master　パスワード：servlet」でログインした場合とする（演習課題-1で作成済み)。演習課題-7で作成した追加機能は削除しないこと。

なお，削除するレコードは主キー項目のみ入力して指定すること。

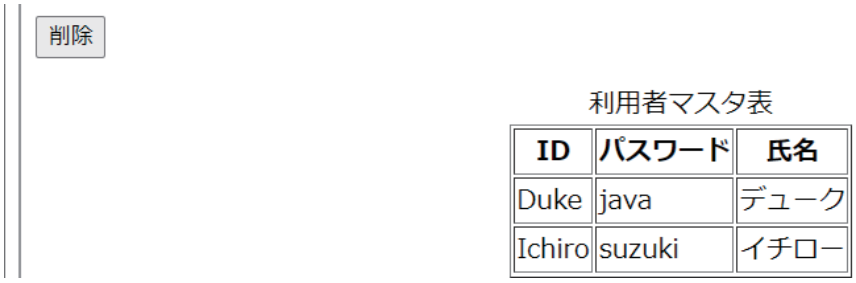
［ファイルの保存場所］


Exercise8
デプロイメント記述子: Exercise8
JAX-WS Web サービス
.settings
build
classes
src
main
java
ctrl
CustomerListServlet.java
LoginProcessServlet.java
LogoutProcessServlet.java
Message.properties
OrderDetailListServlet.java
OrderJoinListServlet.java
OrderListServlet.java
ProductListServlet.java
UserListServlet.java
model
CustomerBean.java
OrderBean.java
OrderControlDBAccess.java
OrderDetailBean.java
OrderJoinBean.java
ProductBean.java
UserBean.java
UserControlDBAccess.java
webapp
META-INF
WEB-INF
CustomerList.jsp
LoginPage.jsp
MainPage.jsp
MasterMaintenance.jsp
MasterSubPage.jsp
OrderDetailList.jsp
OrderJoinList.jsp
OrderList.jsp
ProductList.jsp
SubPage.jsp
UserList.jsp
UserMaintenance.jsp
UserSubPage.jsp

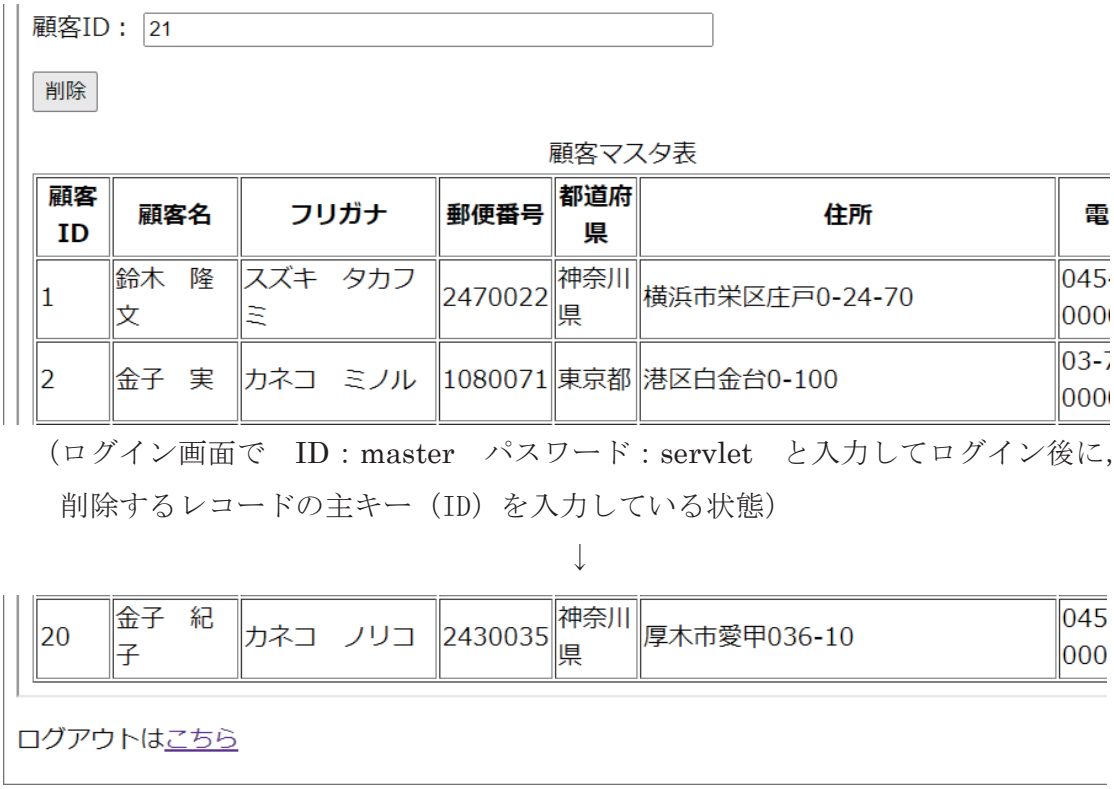
実行結果　(URL：http://localhost:8080/Exercise8/LoginPage.jsp)

（ログイン画面で　ID：admin　パスワード：jsp　と入力してログイン後に，
削除するレコードの主キー（ID）を入力している状態）

↓



（実行結果）

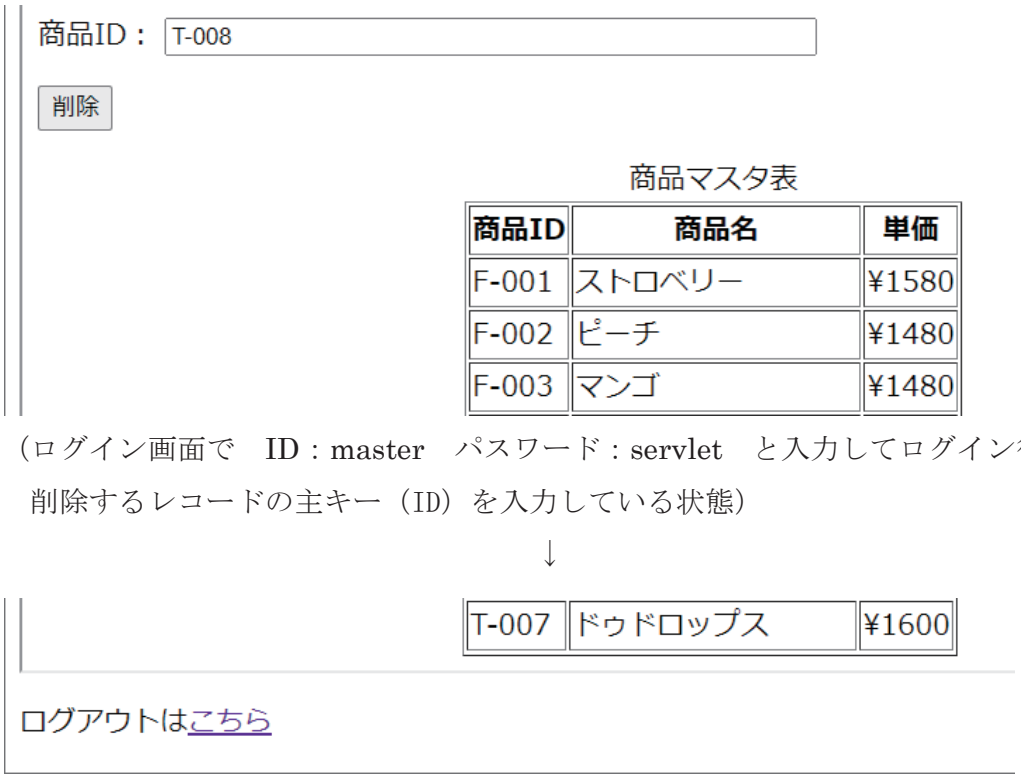


（ログイン画面で　ID：master　パスワード：servlet　と入力してログイン後に，削除するレコードの主キー（ID）を入力している状態）

↓

（実行結果）

注意：受注表に顧客IDが登録されているレコードは，データベースの“参照整合性”という仕組みのため，受注表に顧客IDがある状態で顧客マスタ表からレコードを削除することはできません。



（ログイン画面で　ID：master　パスワード：servlet　と入力してログイン後に，
削除するレコードの主キー（ID）を入力している状態）

↓

（実行結果）

注意：受注明細表に商品IDが登録されているレコードは，データベースの“参照整合性”という仕組みのため，受注表に顧客IDがある状態で顧客マスタ表からレコードを削除することはできません。

令和 6 年度文部科学省委託「専修学校による地域産業中核的人材養成」事業
IT 分野 DX 人材養成のモデルプログラム開発と実証事業

情報 DX エンジニア教材資料
JDBC プログラミング
サーブレット／ JSP

令和 7 年2月

一般社団法人全国専門学校情報教育協会
〒164-0003 東京都中野区東中野 1-57-8 辻沢ビル３F
電話：03-5332-5081 FAX.03-5332-5083