情報システムとWeb技術＃4 — サーブレットによる伝統的Webアプリケーション

久野　靖*

2011.10.25

1　はじめに: 前回までにやったことと今回やること

この科目の名称が「情報システムと Web 技術」ということで、前半はさまざまなシステム分析技術、後半は Web アプリケーションの原理を実際に動かしながら検証する、ということでここまで進んで来ました。後半部分だけについて言うと各回は次の内容になります。

- ＃1 — Java による簡単なクライアントコード、サーバ側コード
- ＃2 — JDBC によるデータベースアクセス、GUI による 2 層アプリケーションと Web サーバ (のおもちゃ) による 3 層アプリケーション
- ＃3 — 汎用 Web サーバの役割、サーブレットとサーブレットコンテナ

なのですが、1 つのサーブレットで複数のページをサーブするあたりはやっぱり複雑になるので、このまま進んでこれに JDBC をくっつけて Web アプリにするという予定の話に進んでも、ごく一部の人しか読めないだろうという問題が明らかになりました。

そういうわけで、今回は自前の簡単なフレームワークを導入して、複数のページ遷移から成る Web アプリケーションをあらためて定式化通りに作成してみていただく、ということにします。

あと、適当なところで前回宿題としてお願いした、皆様固有の日記サイトの画面遷移と状態遷移の設計も見せて頂いて、もしその方がよければ前半で説明したフレームワークでそのサイトのひな型を作って見てもよいようにします。講義の方の題材は DWH(データウェアハウス) の資料を用意したのですが、時間のようすに応じて考えたいと想います。

2　簡単な Web アプリ用フレームワーク WebStates

Web アプリケーションは基本的には前回やったように、複数の画面の間を遷移していく、という形で設計できます。どのような遷移が分かりやすいかとか、そのような話は余裕があれば次回やるかも知れません。

さてここで、実際にサーブレットの立場に立ってみると、各リクエスト (GET または POST) に応じて次のような動作をすることになります。

(1) 提出されてきたデータに応じて処理を行う。

(2) 処理結果を返しつつ、次の画面を生成する。

*経営システム科学専攻

なお、サーブレットが最初に呼ばれた時は提出データは無いので、(1) の部分はありません。

こうして見ると、(1) は「前に表示していたページの処理の続き」であり、(2) は「次に表示するページの生成」ですから、サーブレットによる処理は状態遷移図で描いた各状態遷移に対応する、つまり「前の画面の状態」と「次の画面の状態」に半分ずつまたがったものになるわけです (図**??**)。これが Web プログラミングの分かりにくいところなわけです。

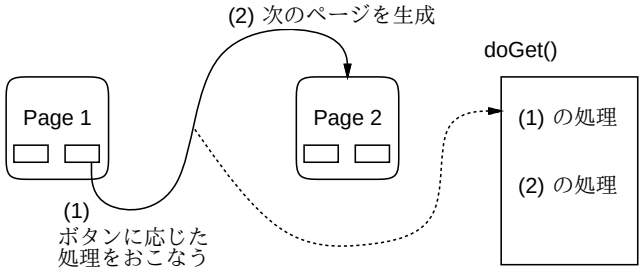

図 1: 状態遷移とサーブレット内の処理

また、(1) の中で「どの処理をするか」は状態に応じて決まって来ますが、さらに提出されてきたデータによって (たとえばどのボタンが押されたかによって) も決まって来ることになります。さらに、処理の内容によって (たとえばパスワードが OK かどうかによって) も、次の状態は変化します。

これらを手で書いているとごちゃごちゃになってわけが分からなくなるので、状態遷移をきっちり定式化して記述し、それを参照しながら動作するようなフレームワークを作りました。さらに、個々のページ表示や状態遷移に伴う処理を個別のメソッドとして定義するようにして、その規則に従っていれば読みやすく簡潔なコードになるようにします。ついでに、セッションに保存した値やフォームから提出されてくるパラメタの値も簡単に変数として参照できるようにしています (こういうサポートも読みやすさを増す効果があります)。

3 チュートリアル (1)

このフレームワークのコアの部分は、状態記述です。たとえば、cmd ボタンの値が left か right かによって One と Two という 2 つのページを行き来する記述は次のように書きます。

```
{{"One", "cmd:left:One", "cmd:right:Two"},
 {"Two", "cmd:left:Two", "cmd:right:One"}}
```

この記述によって、2 つのページを図 2 のように遷移させるわけです。

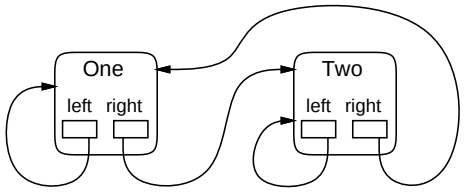

図 2: 2 つのボタンを持つページの状態遷移

それぞれのページは次の形をもつメソッドによって生成します。つまり、これらのメソッドは先に述べた (2) の「ページを生成する」部分だけの処理に専念すればいいわけです。

```
public void showOne() { ... }
public void showTwo() { ... }
```

実際にこのページ遷移を実装したものを見ていただきましょう (図 3)。次のように、状態遷移の記述、各ページ表示のメソッドと、あとそこから下請けとして呼び出す、2 つのボタンを持つフォームを生成するメソッドがあるだけです。なお、変数 pr にはブラウザに送られるストリームにつながった PrintWriter オブジェクトが格納されていて、これに出力することで自由に HTML を生成できます。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.lang.reflect.*;
import java.io.*;
import java.util.*;

public class Sample41 extends WebStates {
  // variables

  // initailzation
  public Sample41() {
    states = new String[][]{
      {"One", "cmd:left:One", "cmd:right:Two"},
      {"Two", "cmd:left:Two", "cmd:right:One"}};
  }

  // showXXX() --- print page for state XXX.
  public void showOne() {
    pr.println("<h1>One</h1>");
    putForm();
  }
  public void showTwo() {
    pr.println("<h1>Two</h1>");
    putForm();
  }

  // doXXX() --- do some operation; return state or next operation.

  // putXXX() --- print out specific HTML part
  void putForm() {
    pr.println("<div class=form>");
    pr.println("<form method=post action='#'>");
    pr.println("<p><button name=cmd value=left>Left</button>");
    pr.println("<button name=cmd value=right>Right</button></p>");
    pr.println("</div>");
  }
}
```

では実習ですが、まず前回自分が作ったコードを保存したい場合は WEB-INF ディレクトリごと別の名前に変更するなどしてください。

```
mv WEB-INF WEB-INF-OLD1   ←別名にして残すか、
rm -rf WEB-INF            ←いらなければ消す
```

コンパイルして動かす手順を再掲します。以下の操作を 1 回だけ実行してください。

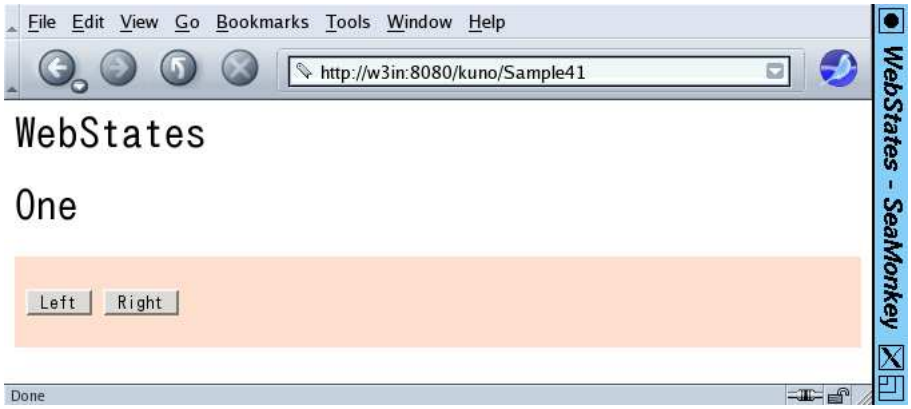

図 3: 2 つのボタンを持つページの画面

```
export CLASSPATH=.:/u1/kuno/work/servlet.jar  ←コンパイルに必要
cp -r /u1/kuno/work/WEB-INF WEB-INF           ←ファイルコピー
```

そして以下は修正のたびに実行します。

```
cd WEB-INF/classes                ← Java コードはここに置いてある
javac Sample41.java               ←コンパイル
cd                                ←元の位置に戻る
jar cvf ユーザ名.war WEB-INF
```

WAR ファイルの名前は自分の名前の後ろに「.war」をつけたものとしてください (他人と衝突しないため)。

次に、Tomcat に接続します。

```
http://w3in:8080/
```

Manager リンクをたどり (管理者は ID とパスワードは口頭で説明)、自分の WAR ファイルを deploy します。動かす時は次の URL を開くのでした。

```
http://w3in:8080/ユーザ名/Sample41
```

演習 1 上の例をそのまま動かせ。

演習 2 One から Five までの 5 つの状態を left/middle/right の 3 つのボタンで行き来するような状態遷移図を描きなさい。なるべく Five まで行くのに苦労するようなものがよい。次に、上の例題を変更してその状態遷移図を実現し、設計通りになっているか確認しなさい。また隣の人になるべく速く Five まで行く課題にチャレンジしてもらいなさい。

質問 1 上に説明した機能だけでは、何が足りないか分かりますか?

4 チュートリアル (2)

前記の機能だけでは、「押したボタンに応じて何らかの処理を行う」という最初の説明での (1) の部分がありませんね。さらに、その処理に応じて行き先の状態を変更することも必要です。

これらの機能を提供するため、実は状態遷移記述において「英大文字で開始される名前は状態」「小文字で開始される名前は処理のためのメソッド」という 2 種類のものが書けるようになっています (慣用的に、処理のためのメソッドは「doXXX()」という名前にしています)。そして、このメソッドは文

字列を返し、その文字列についても「大文字で始まる状態名」と「小文字で始まる処理名」のどちらかを返せます。これにより、いくつかの処理を順番に行って行って、最後に次の状態に遷移する、という形で処理が小分けに記述できるわけです。

そのほか、なるべく簡単にサーブレットの提供する機能を使うため、次のような機能が入れてあります。

- params という文字列配列に文字列変数の名前の並びを入れておくと、その変数にブラウザから送られて来たパラメタの値が自動的に入れられる。
- attrs という変数に文字列配列に文字列変数の名前の並びを入れておくと、その変数に同名のセッション情報として保持されていた値が自動的に入れられる (値を設定するときは setattr(名前, 値) により設定)。
- title という文字列変数にページタイトルを入れておくとそれが自動的に使われる。
- putNote(文字列) というメソッドでメッセージを出力できる。
- putHeadHTML()、putTailHTML() というメソッドを差し替えることで、HTML の冒頭部分と最後の部分を変更できる。

では、これらの機能を使って (全部ではありませんが)、前回やった簡単な BBS を再度作って見ます。今回の方がはるかに簡潔で分かりやすいですね?

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.lang.reflect.*;
import java.io.*;
import java.util.*;

public class Sample42 extends WebStates {
  // variables
  public String cmd, uname, pass, user, mesg;
  ArrayList<String> list = new ArrayList<String>();

  // initailzation
  public Sample42() {
    title = "Simple BBS";
    params = new String[]{"cmd", "uname", "pass", "mesg"};
    attrs = new String[]{"user"};
    states = new String[][]{
      {"Login", "cmd:login:doLogin"},
      {"View", "cmd:post:doPost", "cmd:logout:doLogout"}};
  }

  // showXXX() --- print page for state XXX.
  public void showLogin() {
    putLogin();
  }
  public void showView() {
    putMessages();
    putForm();
  }
```

```
  // doXXX() --- do some operation; return state or next operation.
  public String doLogin() {
    if(!uname.equals("") && pass.equals("hoge")) {
      putNote("welcome."); setattr("user", user = uname); return "View";
    } else {
      putNote("invalid loggin."); return "Login";
    }
  }
  public String doPost() {
    Calendar c = Calendar.getInstance();
    list.add(String.format("%d.%d %d:%02d %s: %s",
      c.get(Calendar.MONTH)+1, c.get(Calendar.DATE),
      c.get(Calendar.HOUR_OF_DAY), c.get(Calendar.MINUTE), user, mesg));
    return "View";
  }
  public String doLogout() {
    setattr("user", null); return "Login";
  }

  // putXXX() --- print out specific HTML part
  void putLogin() {
    pr.println("<div class=form>");
    pr.println("<form method=post action='#'>");
    pr.println("<p>Username: <input name=uname type=text size=12></p>");
    pr.println("<p>Password: <input name=pass type=password size=12></p>");
    pr.println("<p><button type=submit name=cmd value=login>Login</button></p>");
    pr.println("</div>");
  }
  void putMessages() {
    pr.println("<div class=main>");
    for(String s: list) { pr.printf("<p>%s</p>\n", s); }
    pr.println("</div>");
  }
  void putForm() {
    pr.println("<div class=form>");
    pr.println("<form method=post action='#'>");
    pr.printf("<p>Name: %s</p>\n", user);
    pr.println("<p>Mesg: <textarea name=mesg cols=40 rows=5></textarea></p>");
    pr.println("<p><button name=cmd value=post>Post</button>");
    pr.println("<button name=cmd value=view>View</button>");
    pr.println("<button name=cmd value=logout>Logout</button></p>");
    pr.println("</form></div>");
  }
}
```

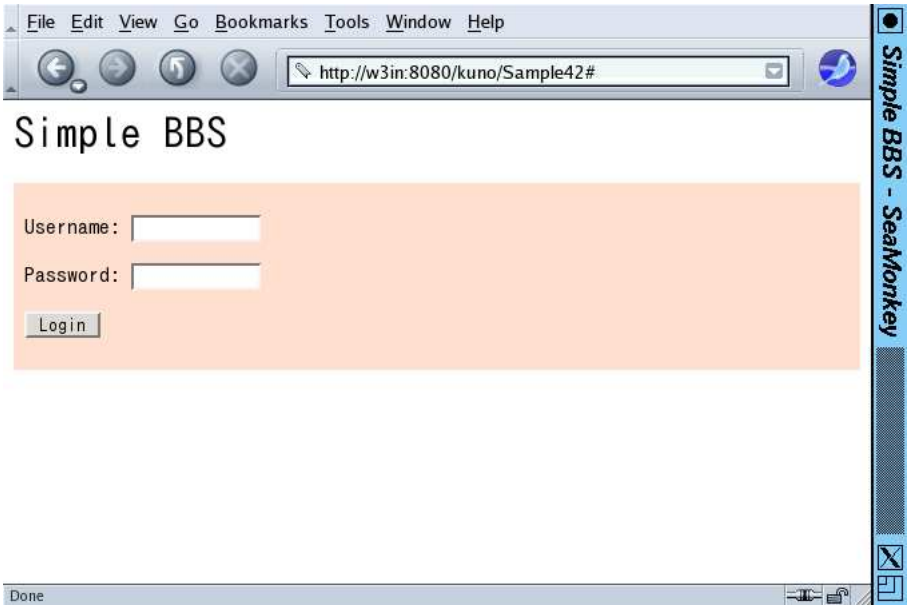

図 4: 掲示板のログイン画面

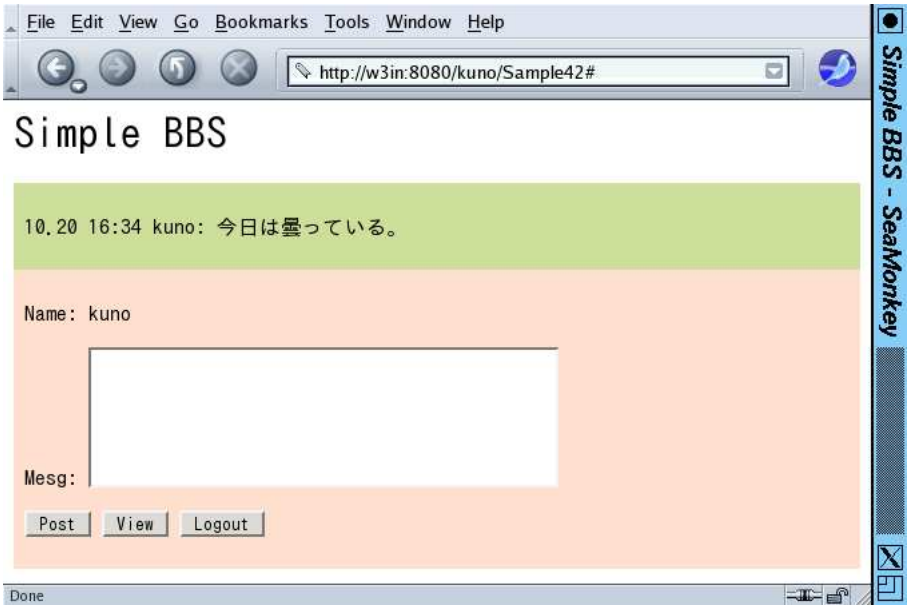

図 5: 掲示板のメッセージ表示画面

5　たね明かし: WebStates の実装

せっかくですから、クラス WebStates の実装をお見せしましょう。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.lang.reflect.*;
import java.io.*;
import java.util.*;

public class WebStates extends HttpServlet {
  HttpServletRequest req;
  HttpServletResponse res;
  HttpSession session;
  PrintWriter pr;
  boolean initialized, initfailed;
  HashMap<String,Method> mmap = new HashMap<String,Method>();
  HashMap<String,Integer> smap = new HashMap<String,Integer>();
  Class<? extends WebStates> cls = WebStates.class;
  String title = "WebStates";
  String[] params = {};
  String[] attrs = {};
  String[][] states = {};
```

ここまでが変数定義ですね。とにかく面倒な受け渡しはさぼることにしたので、変数だらけですが。

```
  public void init(ServletConfig conf) {
    initialized = false; initfailed = false;
  }
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
      throws IOException, ServletException {
    process(req, res);
  }
  protected void doPost(HttpServletRequest req, HttpServletResponse res)
      throws IOException, ServletException {
    process(req, res);
  }
```

ここまではサーブレットの標準的なメソッドです。そして次の process で処理本体を実行していますが、HTML の前半を書いて、それから状態遷移の記述に従った実行を行い、それが終わったらページ表示を行って、最後に HTML の終わりを書き出します。

```
  protected void process(HttpServletRequest req1, HttpServletResponse res1)
      throws IOException, ServletException {
    req = req1; res = res1;
    res.setContentType("text/html");
    res.setCharacterEncoding("iso-2022-jp");
    res.setHeader("Pragma", "no-cach");
    session = req.getSession();
    pr = res.getWriter();
```

```
    putHeadHTML();
    try {
      initialize();
      if(initfailed) { throw new Exception("init error."); }
      getvars();
      String state = getattr("state");
      if(getparam("debug").equals("y")) putNote("curr: " + state);
      if(state.equals("")) { state = states[0][0]; }
      int stateno = smap.get(state);
      for(int i = 1; i < states[stateno].length; ++i) {
        String[] a = states[stateno][i].split(":");
        if(a[0].equals("*") || getparam(a[0]).equals(a[1])) {
          state = a[2]; break;
        }
      }
      if(getparam("debug").equals("y")) putNote("proc: " + state);
      int limit = 0;
      while(!iscapital(state) && ++limit < 10) {
        state = (String)mmap.get(state).invoke(this, new Object[]{});
      }
      if(!iscapital(state) || !smap.containsKey(state)) {
        state = states[0][0];
      }
      if(getparam("debug").equals("y")) putNote("next: " + state);
      mmap.get("show"+state).invoke(this, new Object[]{});
      setattr("state", state);
    } catch(Exception ex) {
      putNote(ex.toString());
    }
    putTailHTML();
    pr.close();
  }
```

以下は役に立ちそうなメソッド群で、サブクラスから利用してもよいわけです。

```
  protected void setattr(String s, String t) {
    session.setAttribute(s, t);
  }
  protected String getattr(String s) {
    String t = (String)session.getAttribute(s); return t == null ? "" : t;
  }
  protected String getparam(String s) {
    String t = (String)req.getParameter(s); return t == null ? "" : t;
  }
  protected void getvars() throws Exception {
    for(String s: params) { cls.getField(s).set(this, getparam(s)); }
    for(String s: attrs) { cls.getField(s).set(this, getattr(s)); }
  }
  protected boolean iscapital(String s) {
```

```
    return s.length() > 0 && s.charAt(0) >= 'A' && s.charAt(0) <= 'Z';
  }
```

initialize() は最初に 1 回だけ実行するメソッドで、状態記述などが正しいかをチェックしています。これを process() から読んでいる理由は、エラーがあったときにメッセージをブラウザに表示するようにしているためです (でないとデバッグ方法がよく分からないですから)。

```
  protected void initialize() throws Exception {
    if(initfailed) { putNote("initialization error."); return; }
    if(initialized) { return; }
    cls = this.getClass();
    for(String s: params) {
      Field f = null;
      try {
        f = cls.getField(s);
      } catch(Exception ex) { }
      if(f == null || f.getType() != String.class) {
        putNote("undef/non-public/non-string variable: " + s); initfailed = true;
      }
    }
    for(String s: attrs) {
      Field f = null;
      try {
        f = cls.getField(s);
      } catch(Exception ex) { }
      if(f == null || f.getType() != String.class) {
        putNote("undef/non-public/non-string variable: " + s); initfailed = true;
      }
    }
    for(int i = 0; i < states.length; ++i) {
      if(states[i] == null || states[i].length < 1) {
        putNote("state desc wrong: "+i); initfailed = true; continue;
      }
      smap.put(states[i][0], i);
    }
    for(int i = 0; i < states.length; ++i) {
      if(states[i] == null || states[i].length < 1) { continue; }
      Method m = null;
      try {
        m = cls.getMethod("show"+states[i][0], new Class<?>[]{});
      } catch(Exception ex) { }
      if(m == null || m.getParameterTypes().length > 0) {
        putNote("invalid or undef method: show"+states[i][0]); initfailed = true;
      } else {
        mmap.put("show"+states[i][0], m);
      }
      for(int j = 1; j < states[i].length; ++j) {
        String[] a = states[i][j].split(":");
        if(a.length != 3 || a[2].length() == 0) {
```

```
            putNote("invalid transition: " + states[i][j]);
            initfailed = true; continue;
          }
          String n = a[2];
          if(iscapital(n)) {
            if(!smap.containsKey(n)) {
              putNote("undefined state: " + n); initfailed = true;
            }
          } else {
            try {
              m = null; m = cls.getMethod(n, new Class<?>[]{});
            } catch(Exception ex) { }
            if(m == null || m.getParameterTypes().length > 0) {
              putNote("invalid or undef method: " + n); initfailed = true;
            } else {
              mmap.put(n, m);
            }
          }
        }
      }
      initialized = true;
    }
```

最後に putNote()、putHeadHTML()、putTailHTML() があります。

```
    void putNote(String m) {
      pr.printf("<div class=note>%s</div>\n", m);
    }
    void putHeadHTML() {
      pr.printf("<html><head><title>%s</title><style type='text/css'>\n",title);
      pr.println(".note { padding: 1ex; background: rgb(200,255,120) }");
      pr.println(".form { padding: 1ex; background: rgb(255,220,200) }");
      pr.println(".main { padding: 1ex; background: rgb(200,220,155) }");
      pr.printf("</style></head><body><h1>%s</h1>\n", title);
    }
    void putTailHTML() {
      pr.println("</body></html>");
    }
  }
```

演習 3 先の例題をそのまま動かせ。動いたら次のような改造をしてみよ。

a. メッセージの表示順を「逆順」にするモードを追加する。
b. 特定の人の書き込みだけを見る (または除外する) モードを追加する。
c. その他、自分の面白いと思う機能を追加する。

演習 4 前回宿題だった、自分なりの Web アプリケーション設計の状態遷移と画面を、このフレームワークを使って作成しなさい。

A　付録: WebStates 参照マニュアル

A.1　概要

WebStates は状態遷移を持つ Web アプリケーション (のおもちゃ) を簡単に WebStates 書けるようにするためのフレームワークです。クラスが HttpServlet のサブクラスとして定義されており、フレームワーク利用者は WebStates のさらにサブクラスを作ってその中で変数を書き換えるなどの形でカスタマイズを行います。

変数を書き換えるカスタマイズは作成するクラスのコンストラクタ内で行うことを想定しています。このため、全体として作成するクラスは次のような構造を持つことになるでしょう (詳細は下記)。

```
public class WebApp extends WebStates {
  // variables
  public String パラメタやセッションデータを受け取る変数…

  // initialization
  public WebApp() {
    title = "ページタイトル";
    params = new String[]{"変数名", "変数名", …};
    attrs = new String[]{"変数名", "変数名", …};
    states = new String[][]{
      {状態 1 の記述…},
      {状態 2 の記述…},
      ....          }};
  }

  // showXXX() --- print page for stateXXX.
  public void showXXX() { ... }
  ...

  // doXXX() --- do some operation; return state or next operation.
  public String showXXX() { ... }
  ...

  // putXXX() --- print out specific HTML part
  void putXXX() { ... }
  ...
}
```

A.2　状態遷移記述

WebStates の土台となる考え方は、複数の画面がそれぞれ 1 つの状態に対応していて、処理に応じてそれぞれの状態の間を遷移してく、というものです。その状態と状態遷移の記述は変数 states に文字列の配列の配列という形で格納します。1 つの状態の記述は次のようになります。

```
{"状態名", "パラメタ:値:指定", "パラメタ:値:指定", …}
```

ここで、先頭の「状態名」は必ず英大文字で始まる必要がある。一番最初の状態を初期状態と呼び、アプリケーションはこの状態から始まる。また、何らかの問題があって状態が不定になった場合も初期状態に戻るようにしている。

以下の部分では、「パラメタ」はブラウザから送られて来るパラメタの名前を表し、「値」はそのパラメタの値がこの値と一致するときにこの項目が選ばれることを意味する。左から順番にマッチを見て行き、最初にマッチが見つかったらそこの「指定」が選ばれる。なお、「パラメタ」として「*」を指定することもでき、そのときは値に関係なく対応する指定が選ばれる (それより右側は調べない)。最後までマッチが無い場合のために、最後には必ず「"*:*:状態名」が置かれているものとして扱う。

「指定」も名前であり、それが大文字で始まる時は、単にその状態に移る。小文字で始まる場合は、その名前の public メソッド (上記の doXXX()) が定義されていなければならず、そのメソッドが呼び出される。このメソッドは文字列を返すので、その結果が再び「指定」として扱われる。ここで無限に繰り返しが始まる可能性があるが、一定回数メソッド呼び出し指定が選ばれたら打ち切られる。打ち切りがあった場合と、状態名 (大文字で始まる名前) が返されたがその状態が未定義の場合は、初期状態が指定されたものと見なされる。

A.3 ページ表示

各状態 XXX ごとに、その状態のページを表示するメソッドを showXXX() という形で定義する必要がある。これにより、それぞれの状態が固有の画面を持つという分かりやすさが実現される。ただし、showXXX() による表示の前に (doXXX() などの処理により) 別の内容を前の部分に出力することもできる。

putXXX() という名前のメソッドは慣例的に、showXXX() から下請けとして呼び出される、ページの各部分要素を出力するメソッドに用いている。

A.4 その他の変数

クラス WebStates で定義されていて、サブクラスから参照することを想定している変数には、次のものがある。

- req — HttpServletRequest オブジェクト。リクエスト情報を格納している。
- res — HttpServletResponse オブジェクト。レスポンス情報を格納している。
- pr — PrintWriter オブジェクト。ブラウザに送り返される文字列を書き出す場所。

サブクラスで書き換えることを想定している変数としては、まず上述の states が挙げられる (これを書き換えないと意味が無い)。

文字列変数 title には、アプリケーションの名称を入れておくことが想定されている (初期値は "WebStates"となっている)。これはページのタイトルとして使用される。

ブラウザから送られて来るフォームのパラメタと、セッションオブジェクトに保存しておく値とは、自動的に同名の文字列変数に取り出して来られるようになっている。ただしこれらの変数は public 指定になっている必要がある。

パラメタを取り出す場合は、文字列の配列型の変数 params に変数名のリストを格納しておけばよい。同様に、セッションに保存しておく値を取り出す場合は、文字列の配列型の変数 attrs に変数名のリストを格納しておけばよい。

A.5 その他のメソッド

クラス WebStates で定義されていて、サブクラスから呼び出すことが想定されているメソッドには次のものがある。

- putNote(String) — メッセージを画面に表示

- setattr(String, String) — 指定した名前の属性値をセッションに保持させる (次回に取り出せる)。

クラス WebStates で定義されていて、サブクラスで差し替えることが想定されているメソッドには次のものがある。

- putHeadHTML() — HTML の冒頭部分を出力
- putTailHTML() — HTML の末尾部分を出力

これらの具体的内容は以下に掲載しておく。

```
void putHeadHTML() {
  pr.printf("<html><head><title>%s</title><style type='text/css'>\n",title);
  pr.println(".note { padding: 1ex; background: rgb(200,255,120) }");
  pr.println(".form { padding: 1ex; background: rgb(255,220,200) }");
  pr.println(".main { padding: 1ex; background: rgb(200,220,155) }");
  pr.printf("</style></head><body><h1>%s</h1>\n", title);
}
void putTailHTML() {
  pr.println("</body></html>");
}
```