対戦ビンゴゲーム
ープログラムの説明ー

岐阜経済大学　経営学部 経営情報学科　井戸　伸彦

来歴：

0．0版　2003年1月17日

スライドの構成

はじめに

(1) どんなゲームか？
(2) サーブレット
(3) プログラムの基本事項
(4) クラスの構成
(5) ブラウザの画面（JSP）
(6) 全体のうごき
(7) データベース
(8) 環境
(9) ログ

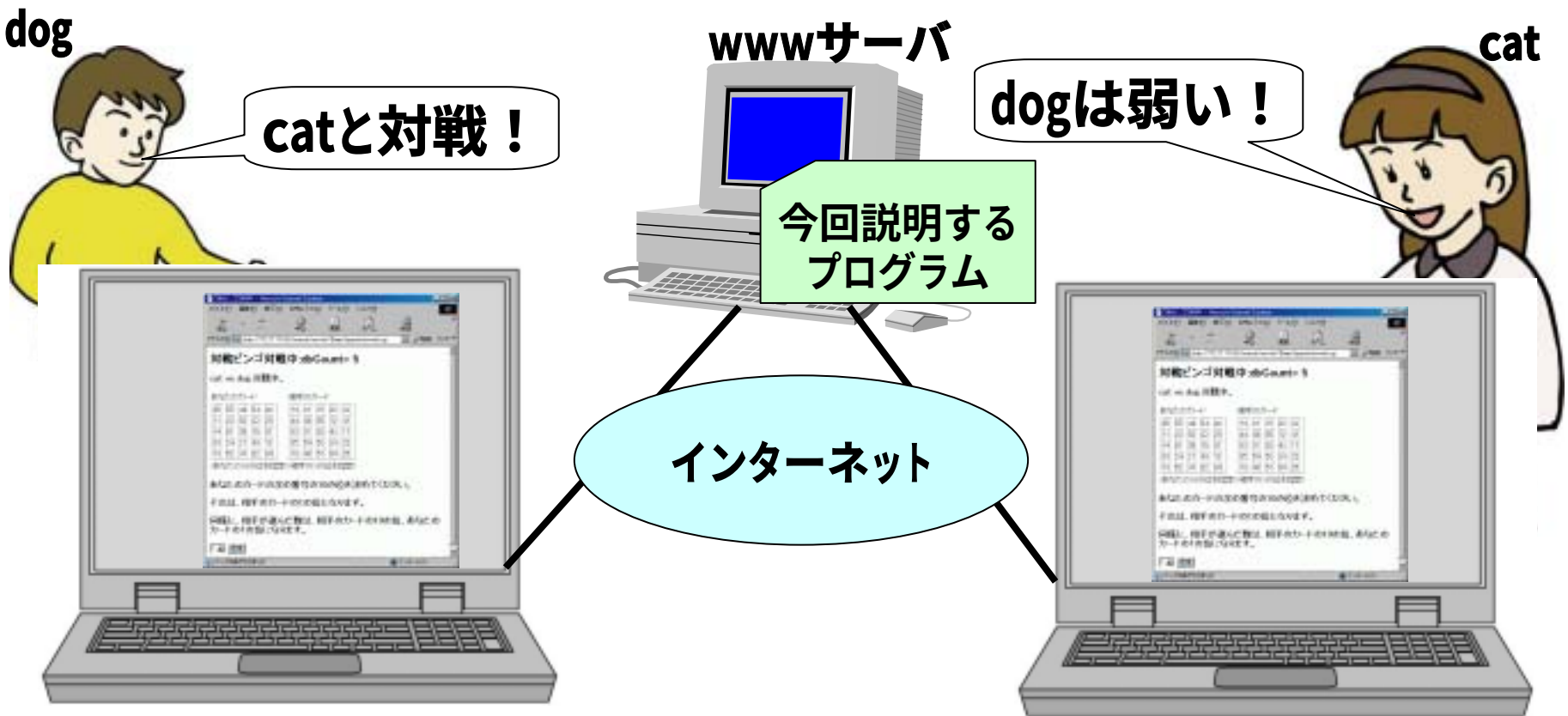
はじめに

- Web上の対戦ゲーム「対戦ビンゴ」のプログラムについて説明します。
- プログラムを理解するには、Javaの知識が必要な部分もあります。特に、クラスについては、別途説明します。
- JSPについては、スライド「ウィークエンド・シャッフルーJSP入門ー」にて勉強済みであることを前提とします。
- 説明は直感的な分かりやすさを重視し、厳密には不正確な言い方もしています。

(1．1) どのような構成のゲームか？

■Webでのオンライン対戦ゲーム

- 次のページにアクセスしてきた二人のユーザで競うゲームです。
- http://172.17.171.xx/tomcat/servlet/Bingo
- 同時に複数組がゲームを出来ますが、相手はプログラムで勝手に決めます。



(1．2) ゲームのルールは？

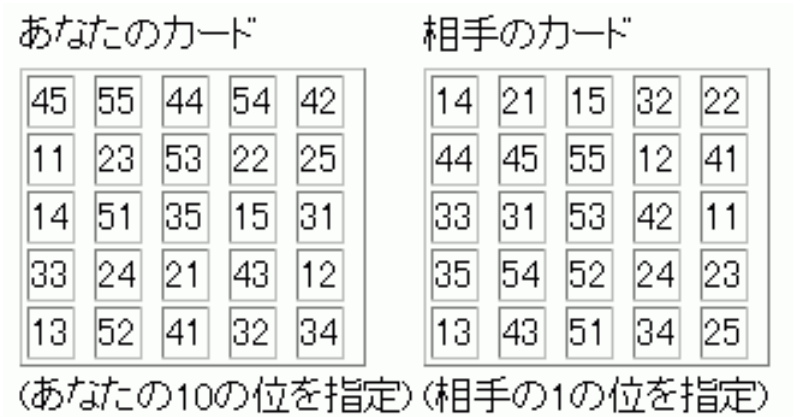
■自分と対戦相手とのBingoゲームのカードが画面に表示されます。

■カードに穴を開ける数字は、次の要領で決めます。

- 貴方の数字と対戦相手との数字は、逆になります。
  - 例えば、あなたが“31”ならば、相手は“13”です。
- 両者は、自分のカードの10の位の数字を決めます。
- 相手がどの数字を次に選ぶかは、わかりません。両者が決まってから、結果が表示されます。

■先にビンゴとなった方が勝ちです。

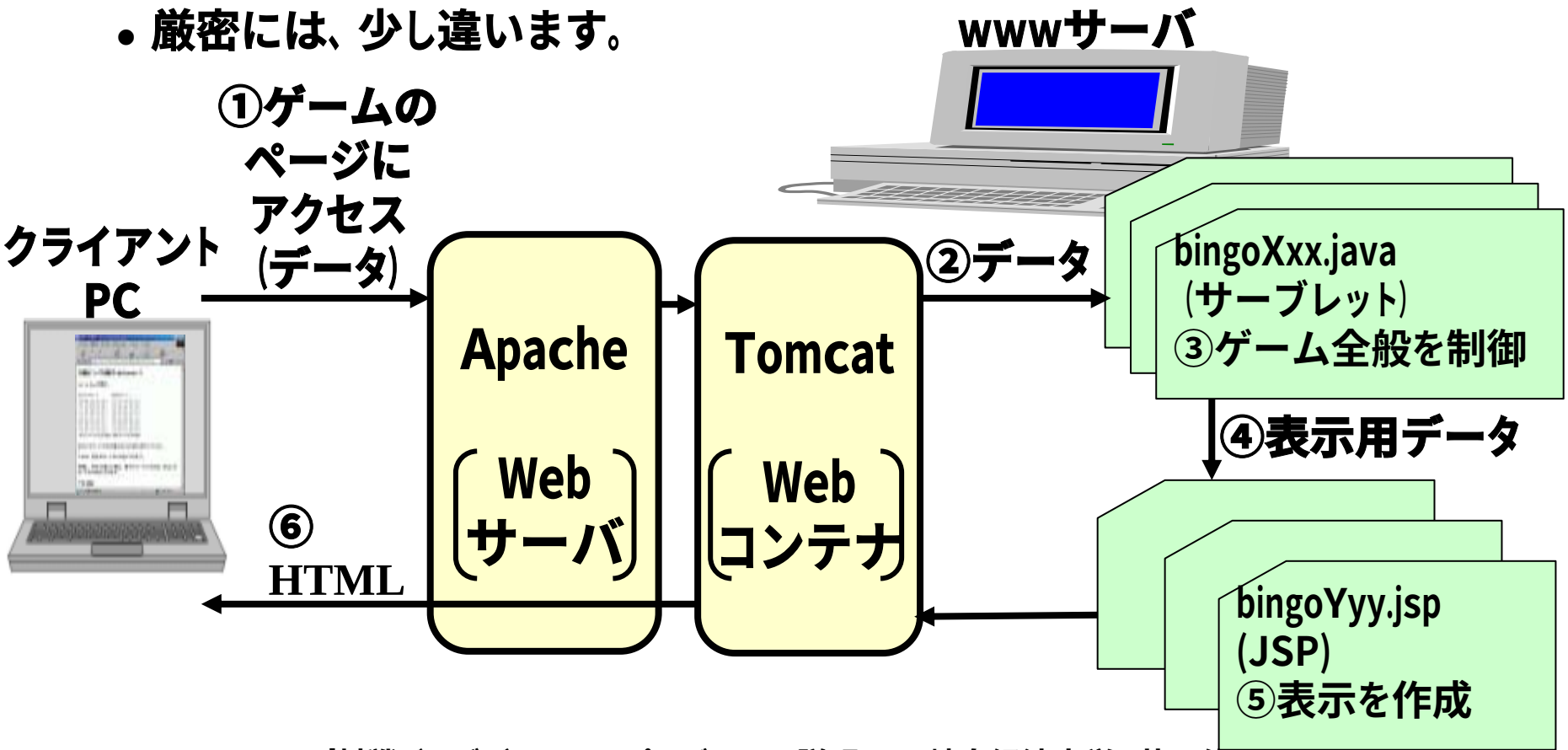
■相手が何を選んでくるかを読み合うことになります。



(2．1) サーブレットの配置

■サーバ上には、次の2種類のプログラムファイルがあります。
- Javaサーブレット：/home/ido/tomcat/WEB-INF/classes/bingoGame.javaなど。
- JSP：/home/ido/tomcat/bingo/bingoOnGame.jspなど。

■ファイルの関係のイメージは、次の通りです。
- 厳密には、少し違います。



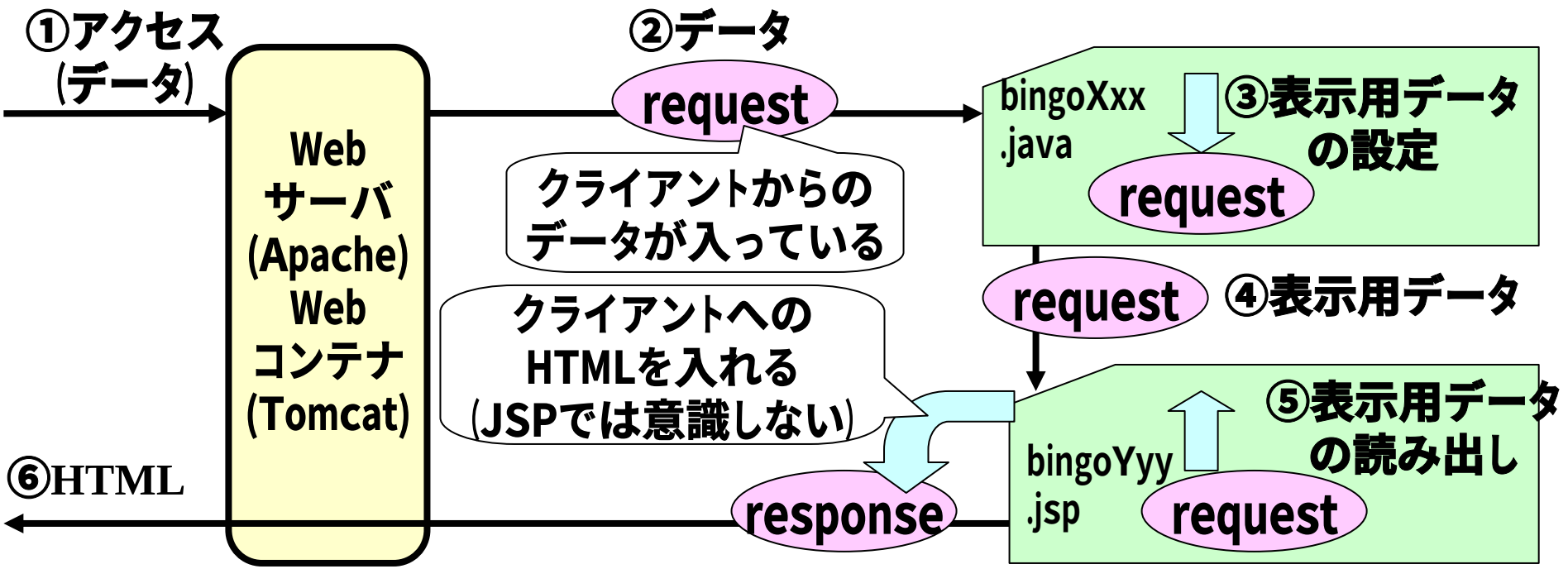
(2．2) サーブレットへのデータの引渡し

■HttpServletRequest

- クライアントからのデータは、HttpServletRequestクラスのインスタンスである“request”に入れてサーブレットに送られます。
- サーブレットからJSPへの表示用データも、“request”により設定します。

■HttpServletResponse

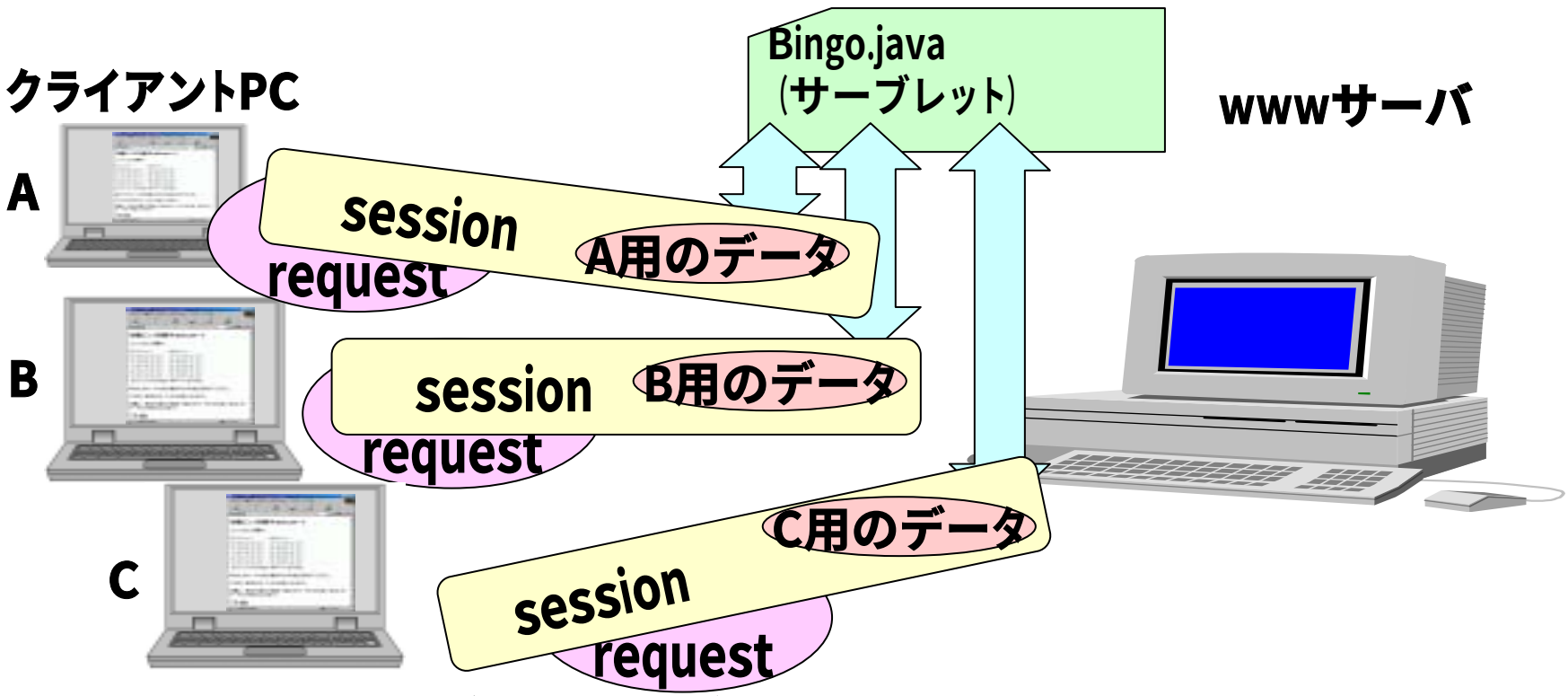
- JSPからTomcatへは、HttpServletResponseクラスである“response”にいれて送られますが、JSPでは特にこれを意識しません。



(2．3) セッション (Session)

■sessionの必要性

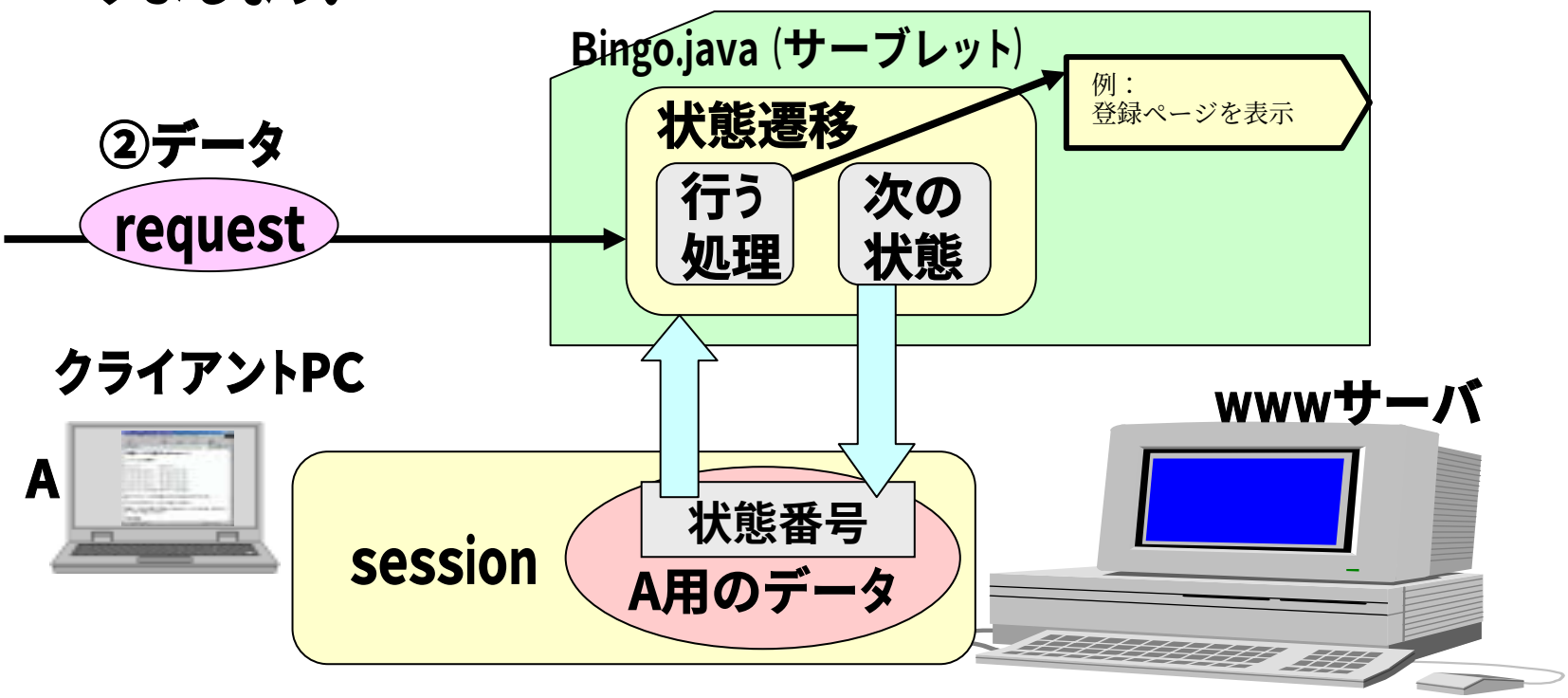
- このプログラムでは、複数台のクライアントPCからのアクセスを扱います。
- プログラムでは、「前にこんなアクセスをしたPCが、今度はこんなアクセスをしてきた」というようなことが分からなければなりません。
- クライアントPCごとのデータ置き場が、session (HttpSessionクラス) です。
- Session自体は、requestに入れてサーブレットに知らされます。



(3．1) プログラムの基本的構造：状態遷移

■プログラムは、状態遷移の考え方で制御されています。
- 状態とは、「今、どんな場面にあるか」を表すデータです。
- 送られてきたデータと現在の状態とにより、行う処理を決め、次の状態を決めます（状態が変わる（遷移する）わけです）。

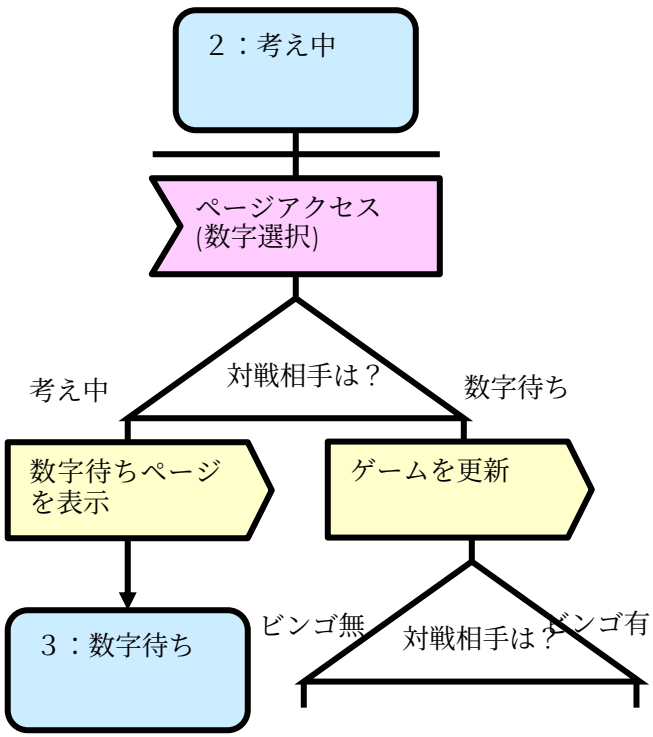
■プログラムの状態遷移は、別紙に示してあります。一緒に見てみましょう。

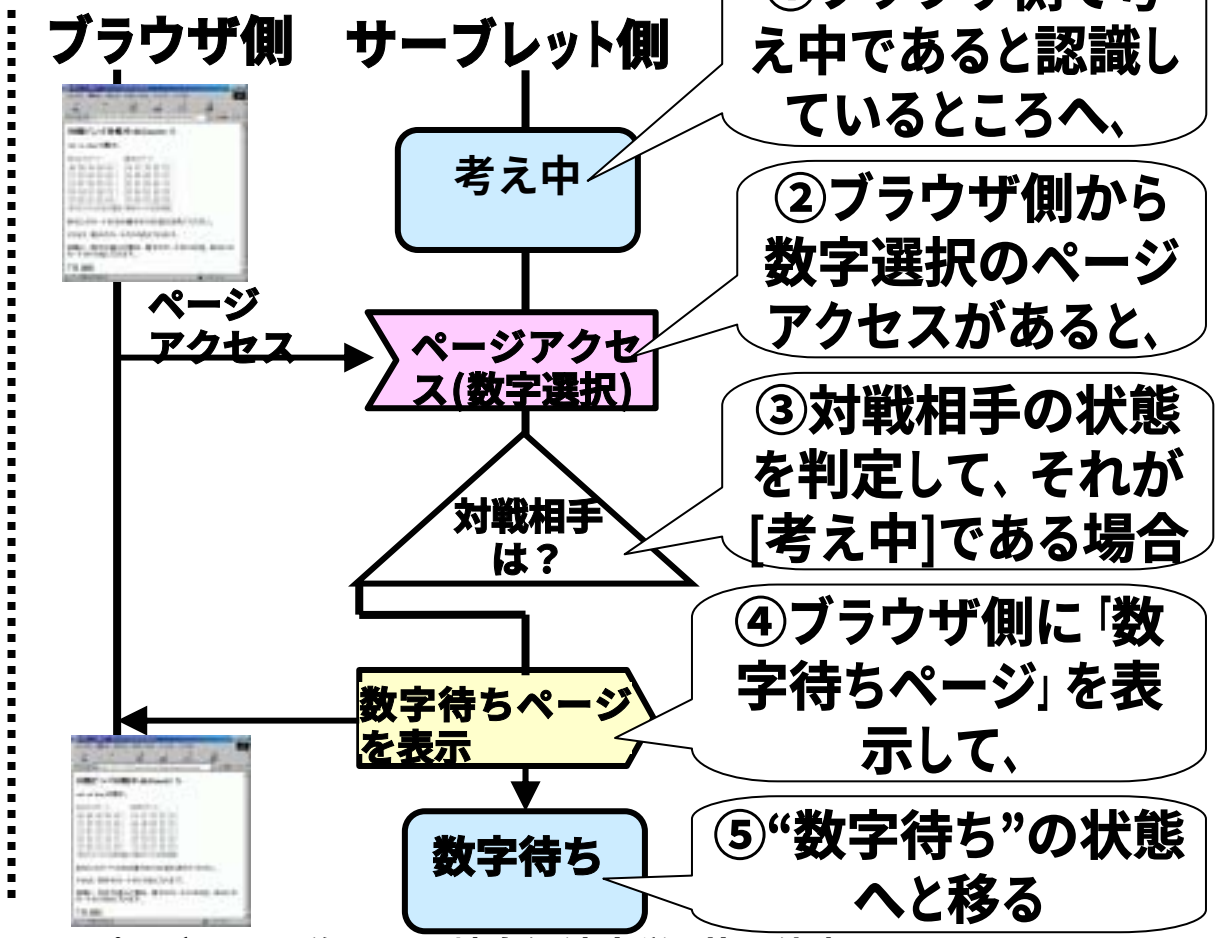


(3．2) 状態遷移図

■別紙の状態遷移図は、SDLと呼ぶ記法で記してあります。

■例として、左下に記した状態遷移図の部分が、何を示しているのかを、右下に示します。



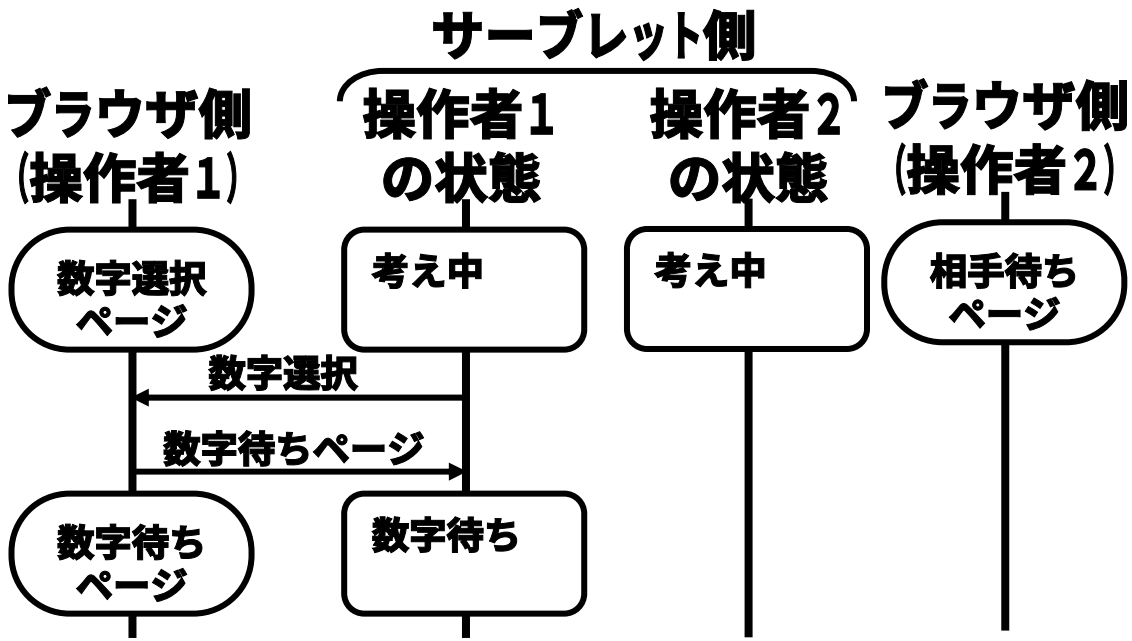


(3．3) シーケンス図

■先ほどとは別の別紙に、シーケンス図が示してあります。

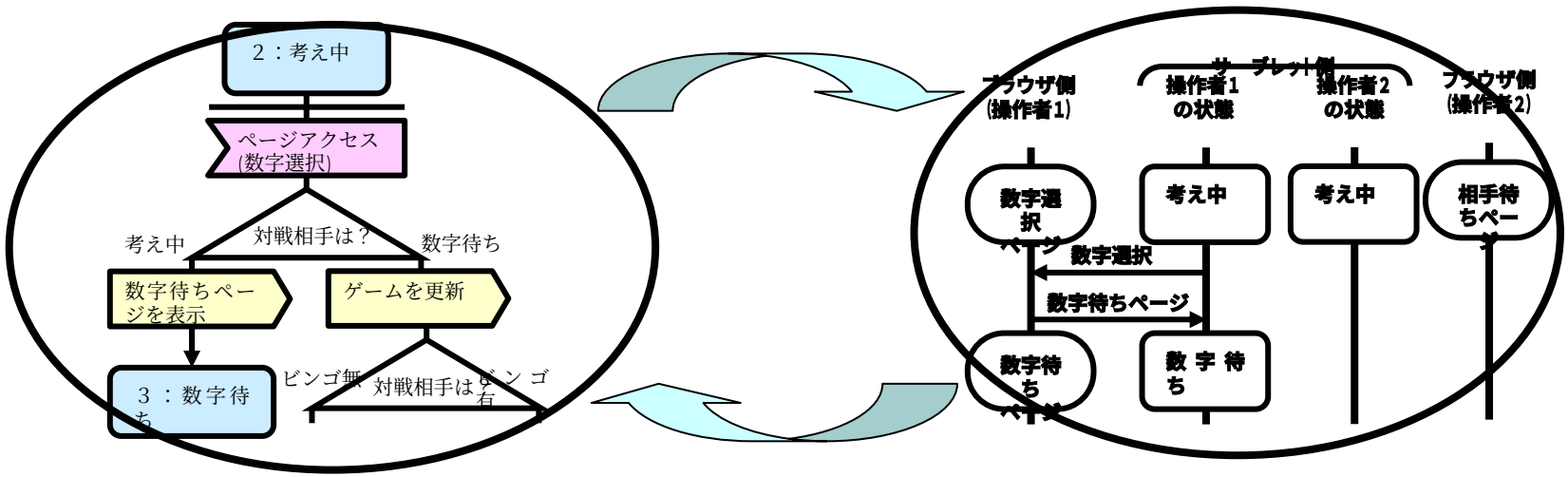
■状態遷移図では、状態を持つひとつの“もの”(=前のスライドではサーブレット側) に対してのみの動きをすべて表しますが、シーケンス図では、複数の要素も書き加えて、相互の動作関係を示します。但し、シーケンス図ではすべての場合ではなく、ある特定の場合についての相互動作関係を示します。

	状態遷移図	シーケンス図
示す対象	1つ	複数
示す場合	すべての場合	ある特定の場合



(3．4) 設計

■分散して配置され相互動作するソフトウェアの設計においては、状態遷移図とシーケンス図とを作成しながら設計を進めていくことが一般的です。一方を考えながら、他方を手直ししていく訳です。



■Bingoのプログラムでは、bingoGame.javaの中に状態遷移部分の記述があります。なるべく状態遷移図と対比がつきやすいような処理としています

(3．5) プログラムとの対応

■bingoGame.javaのファイル内のクラス、stateTransferを見ると、おおよそ次のようなコーディングになっています。

```
Switch(state)←① 状態を判定
    case STATE_xxxx :←② 現在の状態
        int op_state = opponent.getState();
        switch (op_state) {←④対戦相手の状態を判定
         case STATE_yyyy :←⑤ 対戦相手の現在の状態
            <⑥必要な処理を行い、>
            player.setState(STATE_zzzz);
                ↑⑦ 次の状態の設定状態
```

③対戦相手の
↓状態を取得

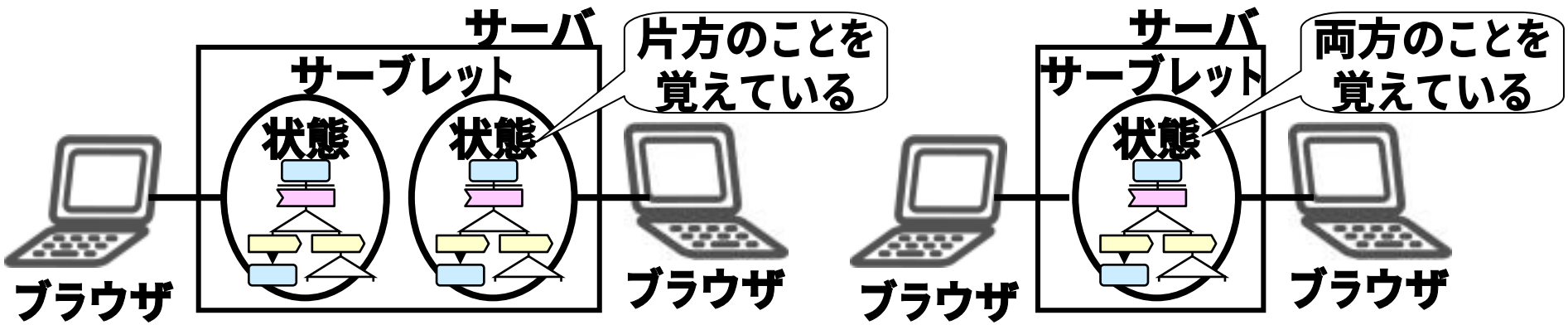
■プログラムの詳細については、後のスライド(スライド(6)以降)でみていきます。

(3．6) 状態を持つものを何とするか？

■ここまでには触れていませんでしたが、状態遷移図･シーケンス図を作成する前の段階に、設計における重要な課題があります。それは、「状態を持つもの何とするか」を決めることです。

■Bingoゲームの場合、次の2つの案が考えられます。

- (案1) 対戦者ごとに状態を持つ。　　(案2) ゲームごとに状態を持つ。



本プログラムは、(案1) を採っています。

■この決定は、残念ながら、必ず正解があるというものではなく、簡単ではありません。ここではこの議論に立ち入ることはせず、別稿にて考えることにします。

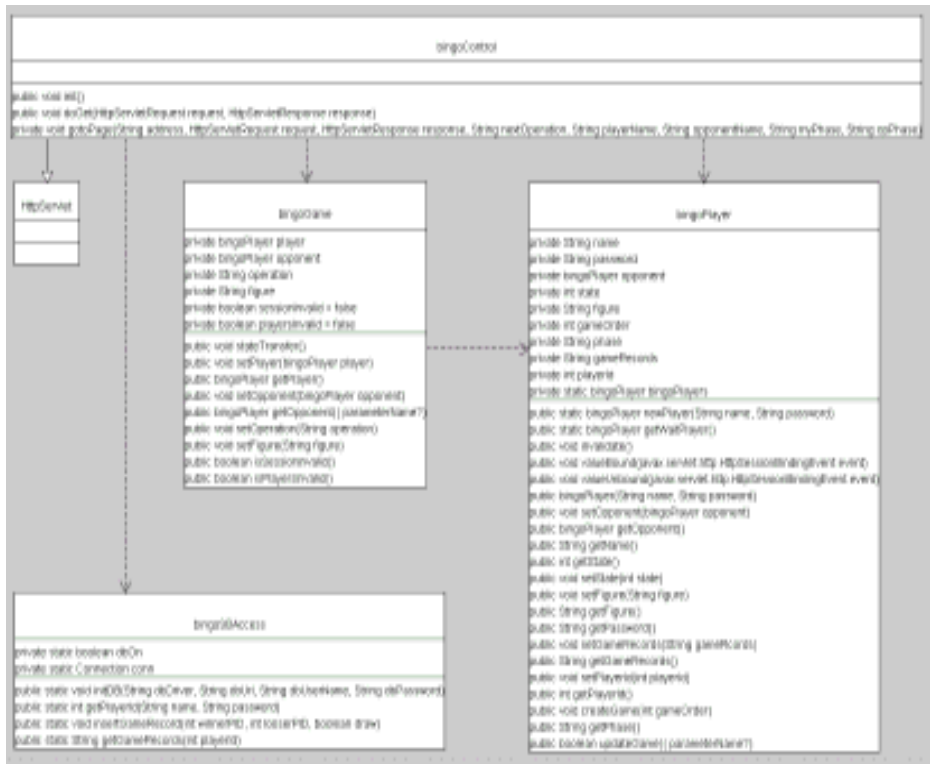
(4．1) UMLクラス図

■クラス

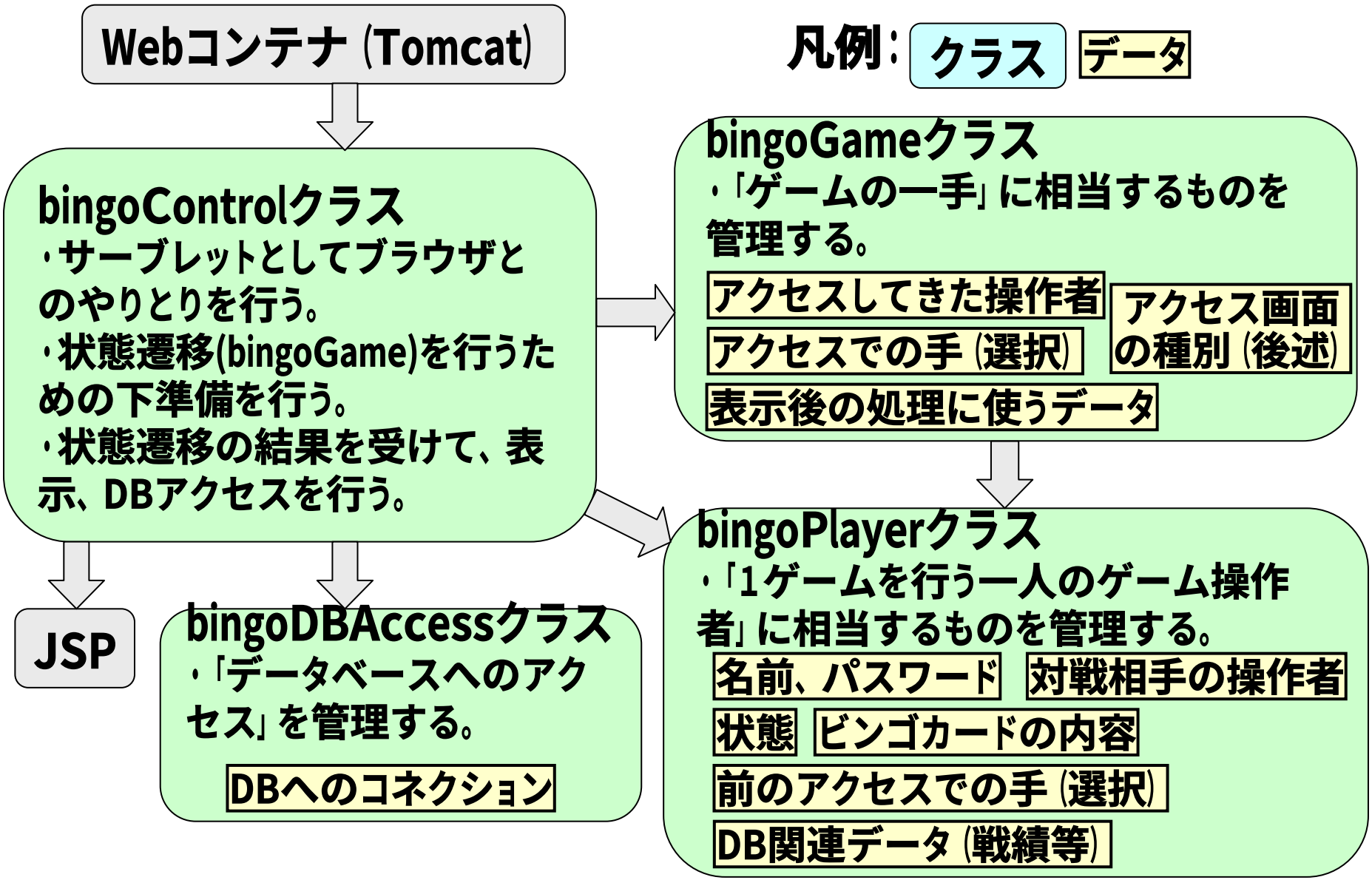
- オブジェクト指向については、別途説明します。
- ここでは、「クラスはプログラムのかたまり」と理解しておいてください。

■UML

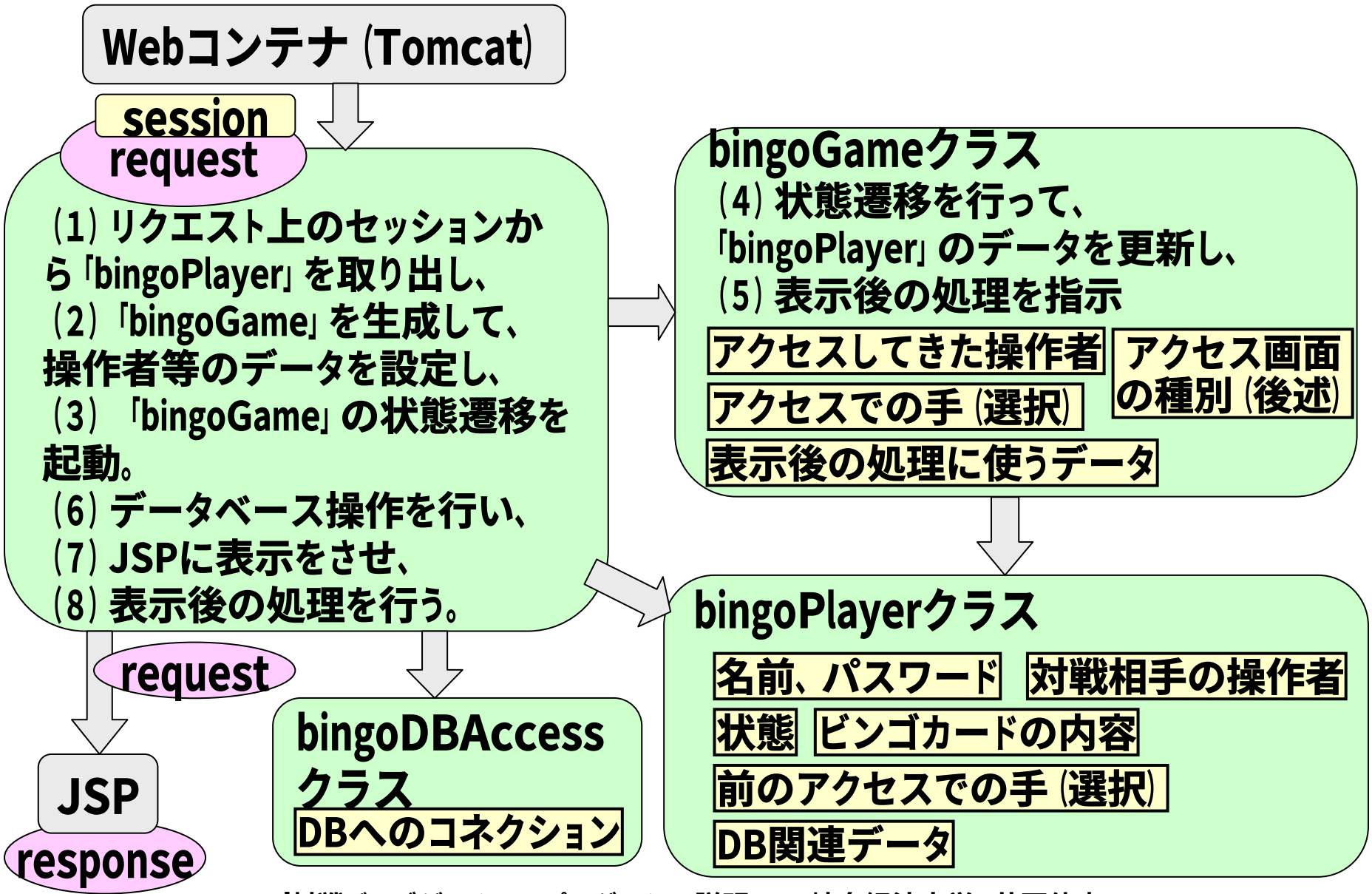
- UMLのクラス図で書くと、右のようなクラス構成になります。
- UMLについては別の機会に説明します。
- 次スライド以降では、直感的な説明を行います。



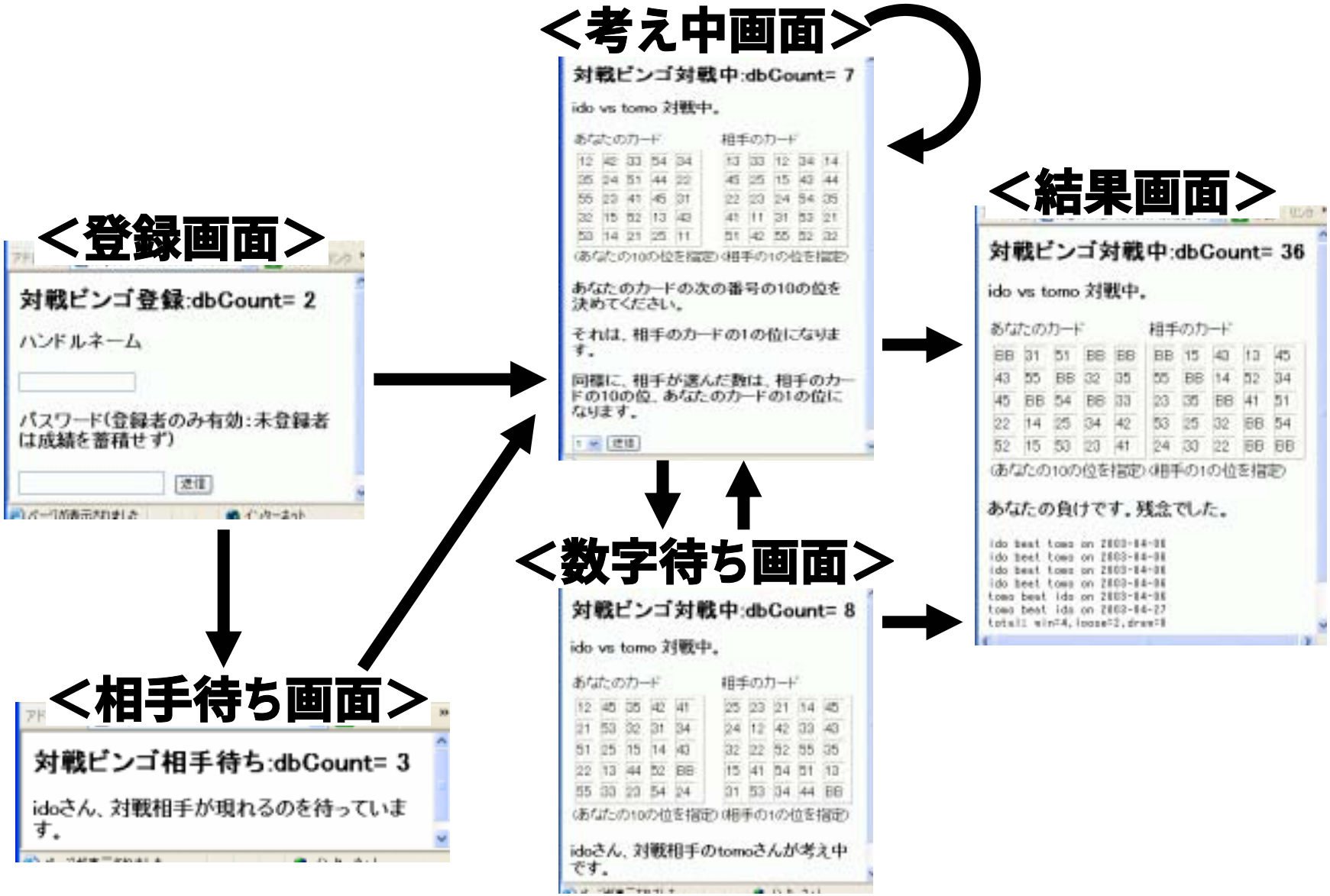
(4．2) クラスの構成


Webコンテナ (Tomcat)
凡例：
クラス
データ
bingoControlクラス
・サーブレットとしてブラウザとのやりとりを行う。
・状態遷移(bingoGame)を行うための下準備を行う。
・状態遷移の結果を受けて、表示、DBアクセスを行う。
bingoGameクラス
・「ゲームの一手」に相当するものを管理する。
アクセスしてきた操作者
アクセスでの手 (選択)
アクセス画面の種別 (後述)
表示後の処理に使うデータ
bingoPlayerクラス
・「1ゲームを行う一人のゲーム操作者」に相当するものを管理する。
名前、パスワード
対戦相手の操作者
状態
ビンゴカードの内容
前のアクセスでの手 (選択)
DB関連データ (戦績等)
JSP
bingoDBAccessクラス
・「データベースへのアクセス」を管理する。
DBへのコネクション

（4．3）一般的な動き


Webコンテナ（Tomcat）
session
request
（1）リクエスト上のセッションから「bingoPlayer」を取り出し、
（2）「bingoGame」を生成して、操作者等のデータを設定し、
（3）「bingoGame」の状態遷移を起動。
（6）データベース操作を行い、
（7）JSPに表示をさせ、
（8）表示後の処理を行う。
bingoGameクラス
（4）状態遷移を行って、「bingoPlayer」のデータを更新し、
（5）表示後の処理を指示
アクセスしてきた操作者
アクセス画面の種別（後述）
アクセスでの手（選択）
表示後の処理に使うデータ
bingoPlayerクラス
名前、パスワード
対戦相手の操作者
状態
ビンゴカードの内容
前のアクセスでの手（選択）
DB関連データ
request
JSP
response
bingoDBAccessクラス
DBへのコネクション

(5．1) 主な画面


＜考え中画面＞
＜登録画面＞
＜結果画面＞
＜数字待ち画面＞
＜相手待ち画面＞
対戦ビンゴ対戦中:dbCount= 7
ido vs tomo 対戦中。
あなたのカード
相手のカード
あなたのカードの次の番号の10の位を決めてください。
それは、相手のカードの1の位になります。
同様に、相手が選んだ数は、相手のカードの10の位、あなたのカードの1の位になります。
対戦ビンゴ登録:dbCount= 2
ハンドルネーム
パスワード(登録者のみ有効:未登録者は成績を蓄積せず)
対戦ビンゴ対戦中:dbCount= 36
あなたの負けです。残念でした。
対戦ビンゴ対戦中:dbCount= 8
idoさん、対戦相手のtomoさんが考え中です。
対戦ビンゴ相手待ち:dbCount= 3
idoさん、対戦相手が現れるのを待っています。

(5．2) 登録画面

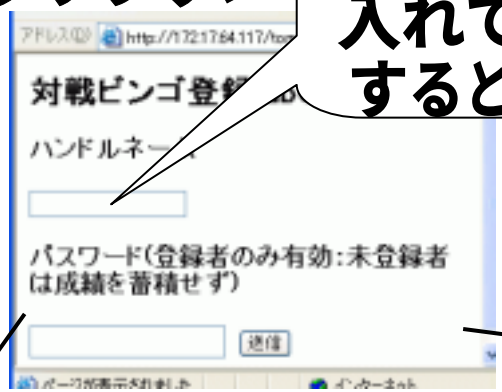
■表示：ＪＳＰファイル中のＨＴＭＬにて固定的に行います。

■送信：入力した名前とパスワードが、送信ボタンを押すことにより送信されます。

■受信：スライド(2.2)に記したとおり、bingoControlクラスでは、“request”から“name”と“password”とを読み出します。

＜ブラウザ＞

ここに”ido”と入れて送信すると。。。



request

送信

＜サーブレット＞

この変数に“ido”と読み出せる

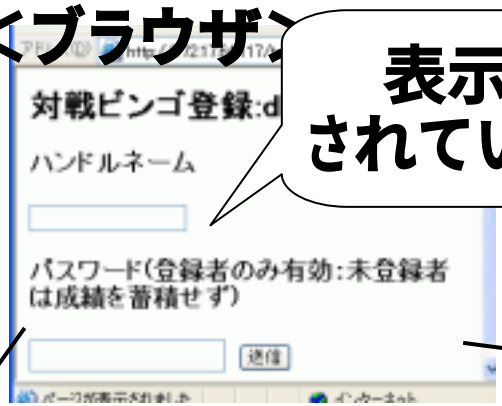
```
String PlayerName =
request.
getParameter(“yourName");
```

```
<input type="text" name="yourName"><br><br>
<input type="password" name="password">
<input type="hidden" name="operation" value="registrate">
```

(5．3) どの画面からのデータか？

- ゲームで使うブラウザの画面にて、“hidden”というフォームの要素があります (名前は“operation”です)。
- この要素は、画面上表示されていませんが、サーブレット側では、”value”に設定された値が読み出せます。
- 例えば、登録画面には、“registrate” (登録する) という値が入っているので、サーブレット側では送られてきたデータが、登録画面からであることを判定しています (スライド(4.2)の“アクセス画面の種別”がこの値になります)。

<ブラウザ>



表示はされていない

request

送信

<サーブレット>

この値が“registrate”なので、登録画面からのデータと分かる

```
String operation =
request.
getParameter("operation");
```

```
<input type="text" name="yourName"><br><br>
<input type="password" name="password">
<input type="hidden" name="operation" value="registrate">
```

(5．4) 相手待ち画面

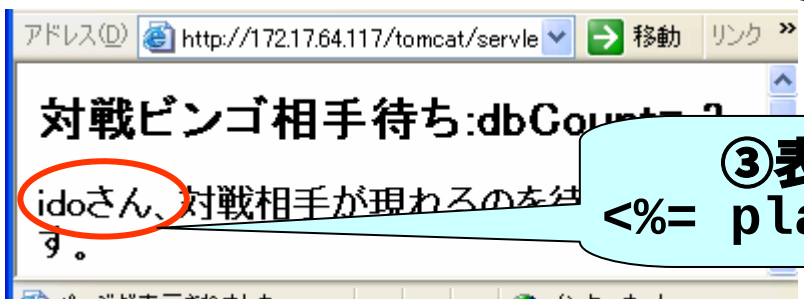
■表示：スライド(2.2)に記したとおり、サーブレット側で“request”に設定した、“player_name”の値を、ＪＳＰ側で読み取って表示します。

＜サーブレット＞①名前を書き込み、

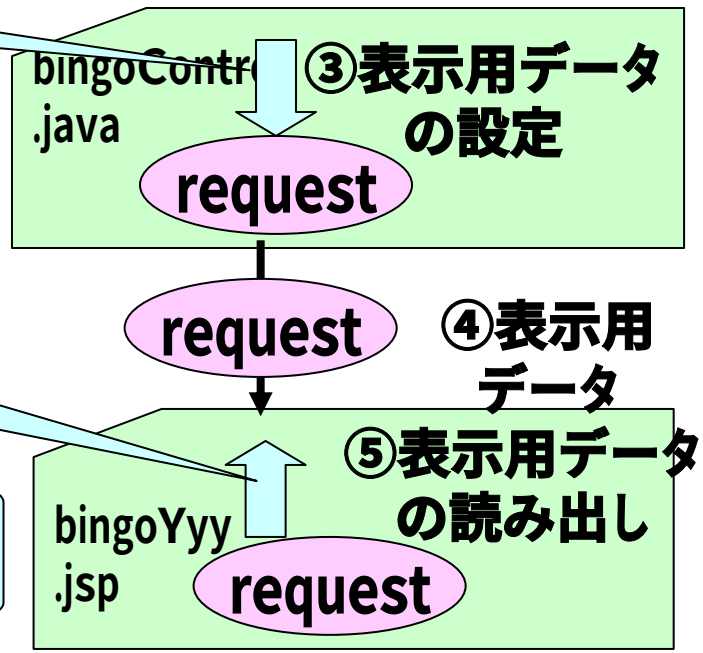
```
request.setAttribute("player_name"
,game.getPlayer().getName());
```

＜ＪＳＰ＞②名前を読み取って、

```
<% String player =
(String)request.getAttribute(
"player_name"); %>
```



③表示する。
<%= player %>さん、



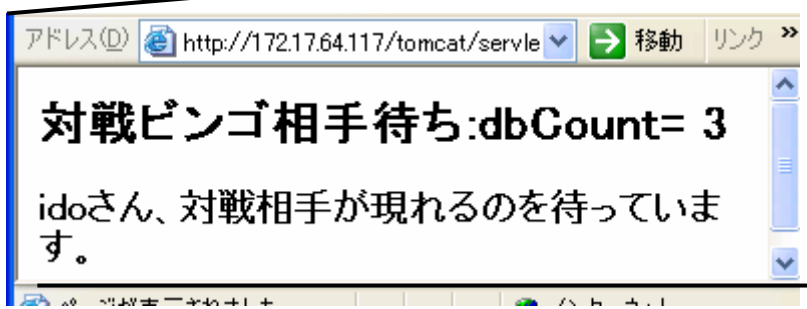
(5．5) 相手待ち画面の送信･受信

■送信

- 相手待ち画面に、送信ボタンはありません。
- 相手待ち画面のＨＴＭＬのヘッダには、「２０秒ごとに更新する」という指令にあたるメタ要素が含まれています。
- これによりブラウザは、何もなくても２０秒ごとに送信（＝ページの更新）を行います（「数字待ち画面」も同じです）。

■受信：スライド(5.3)に記した“operation”が受信されます。

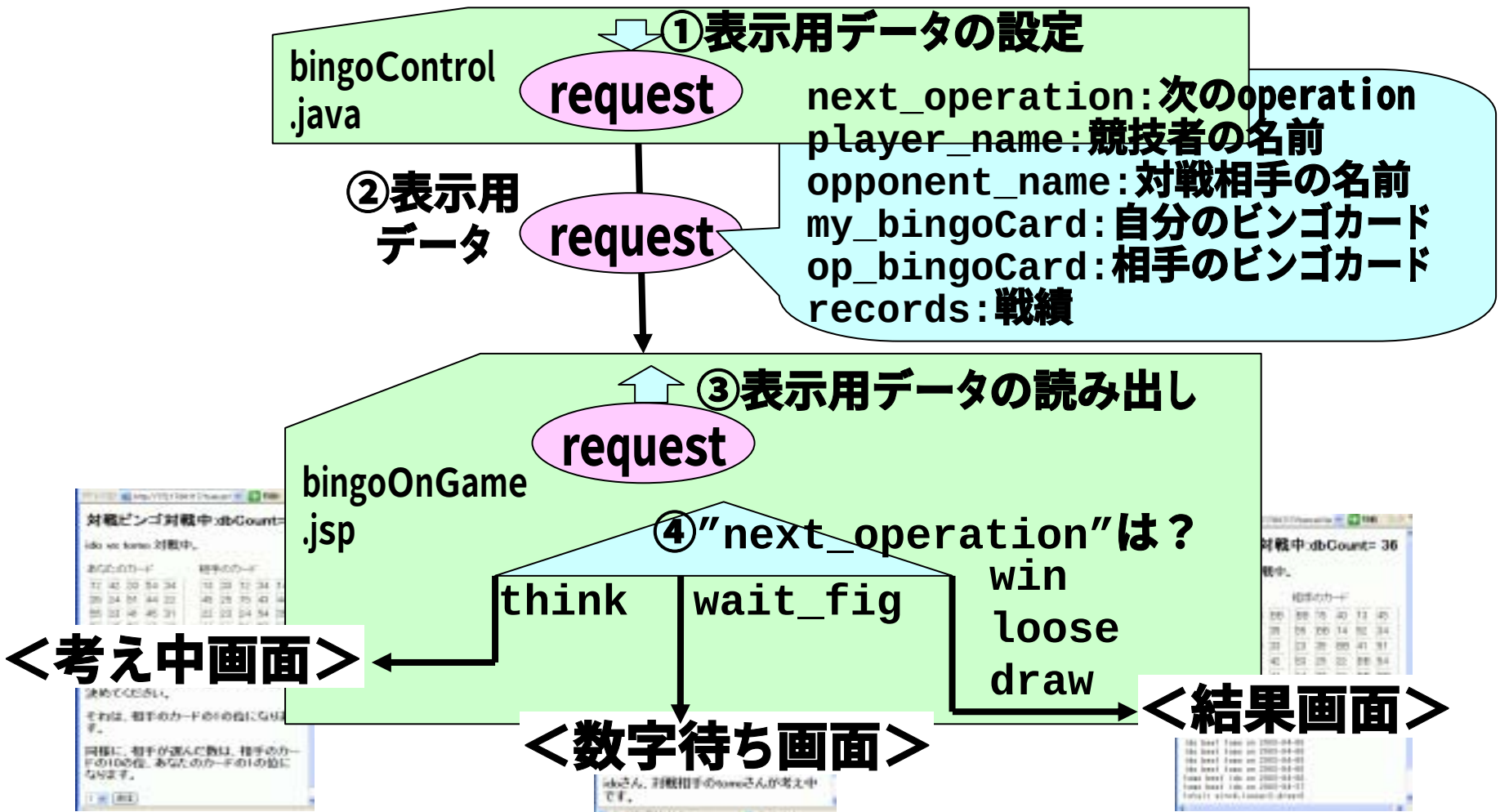
「２０秒ごとに“更新”する」という意味




```
<head>
<meta http-equiv="refresh"
content="20">
<title>対戦ビンゴ相手待ち</title>
</head>
```

(5．6) 考え中画面

■「考え中画面」、「数字待ち画面」、「結果画面」の3つは、同じＪＳＰファイルにて表示されています。



(5．7) 考え中画面　ー表示ー

■next_operation:次のoperation
- スライド(5.3)に記した、operationに設定します。

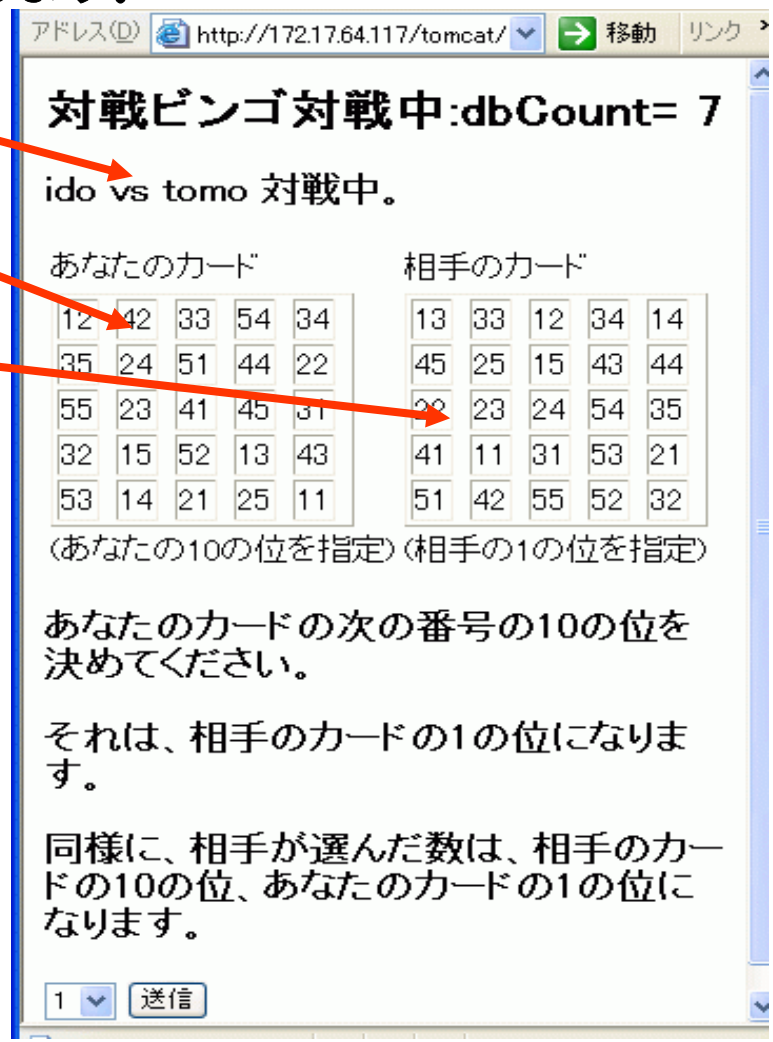
■player_name:競技者の名前
opponent_name:対戦相手の名前

■my_bingoCard:自分のビンゴカード
op_bingoCard:相手のビンゴカード
- つぎのようなデータです。
- 5 15 42 33 54 (中略) 14 21 25 11

5×5の升目であることを表す

升目に、”5 15 42 33 54 (中略) 14 21 25 11”と数字が並ぶことを表す

■records:戦績
- データベースから読み出した、過去の戦績です (後述) 。



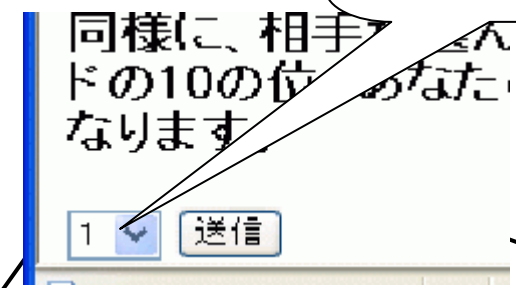
(5．8)考え中画面　ー送信･受信ー

■送信：フォームのselect要素により数字を選択し、送信ボタン押下することにより送信されます。

■受信：”request”から値を読み取ります。

<ブラウザ>

選択して送信すると。。。



request

送信

<サーブレット>

この変数に選択した番号の文字列が読み出せる

```
String figure =
(String)request.getPara
meter("figure");
```

<JSP>

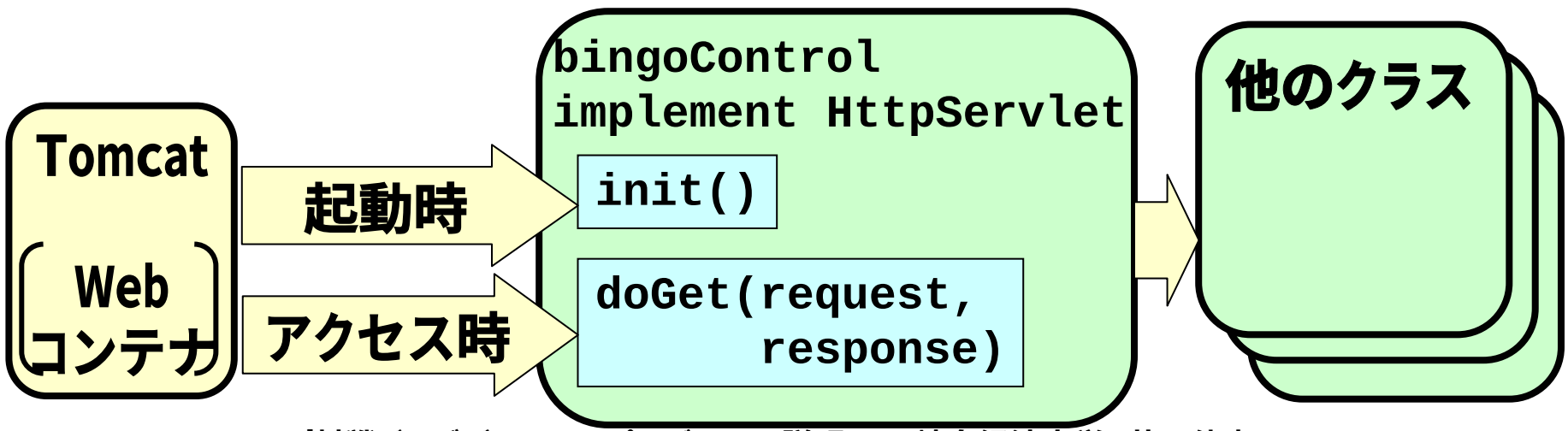
```
<select name="figure">
<% for(int i=0;i<
    gameOrder[0];i++){%>
<option value="<%= i+1 %>">
                  <%= i+1 %>
<% }%>
</select>
```

返信

```
<select name="figure">
<option value=“1">1
<option value=“2">2
<option value=“3">3
<option value=“4">4
<option value=“5">5
</select>
```

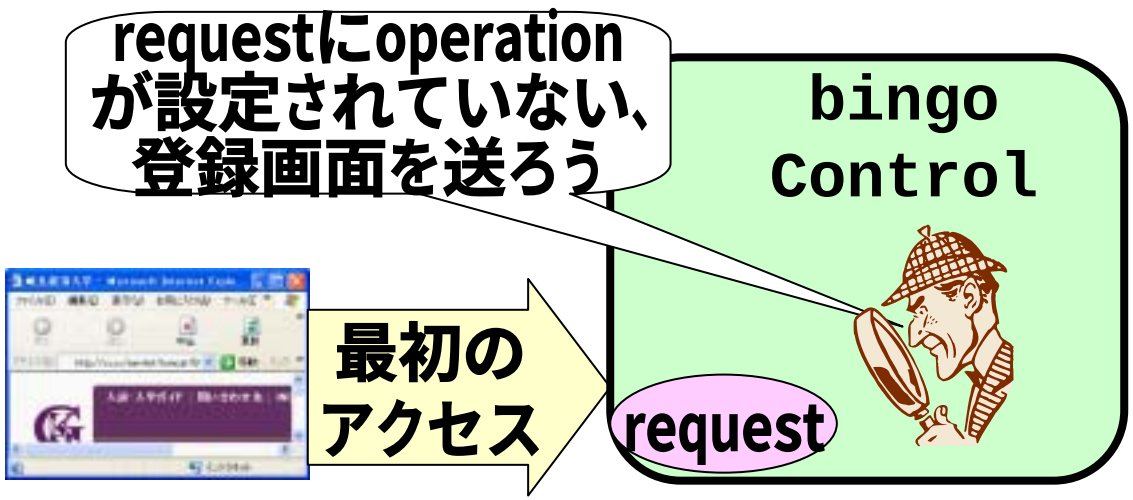
(6) 全体の動き

- tomcat (スライド(2.1)) は、クラス"bingoControl"だけを相手にしています(タイムアウト処理については後述)。これは、このクラスが、親クラス"HttpServlet"を継承しているためです。
- サーバが立ち上がって、tomcatが起動されたとき、クラス"bingoControl"のinitメソッドが呼ばれます。
- "http://xxx../tomcat/servlet/Bingo"にアクセスされた時、クラス"bingoControl"のdoGetメソッドが呼ばれます。



(6．1．1) 最初のアクセス

■最初に、“http://xxx../tomcat/servlet/Bingo”にアクセスされた時、スライド(5.3)に記した“operation”には値が入っていない(=null)はずですから、この場合は登録画面のＪＳＰ(bingoEntry.jsp)を起動して終了します。



(6．1．2) 登録画面によるアクセス

■クラス”bingoGame”のインスタンス”game”を生成(new)します。

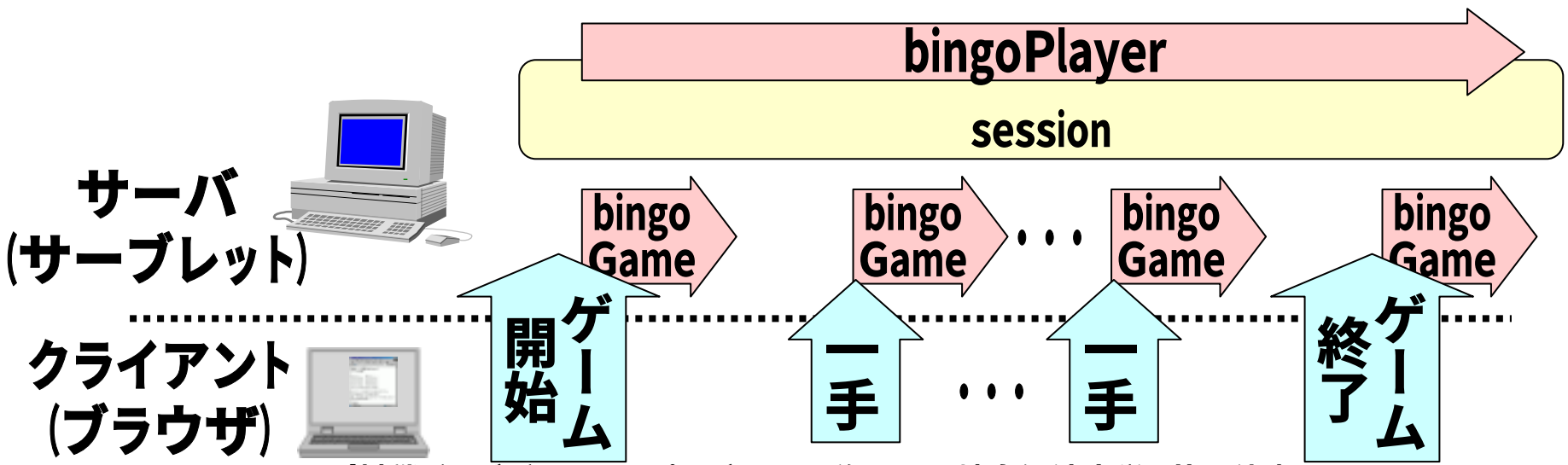
■セッション(requestより取り出したsession、スライド(2.3)参照)に蓄積しておいた、クラス”Bingoplayer”のインスタンス”player”を取り出します。

■登録画面からのアクセスの場合、セッションには何も蓄積されておらず、operationは”registrate”です。この場合に、クラス”Bingoplayer”の新しいインスタンス”player”を生成(new)し、これをセッションに保持させます。

（6．2）bingoPlayerとbingoGame

■クラス“bingoPlayer”は、「1ゲームを行う一人のゲーム操作者」に相当する“もの”（スライド(4.2)）ですから、そのインスタンスはゲームの開始から終了まで生きていて、セッションに保持されます。

■クラス“bingoGame”は、「ゲームの一手」に相当する“もの”（スライド(4.2)）ですから、アクセスを受けて始まる一連の処理の間だけ生きており、保持されません。



（6．3）bingoPlayerの生成

bingoPlayerクラス

```
private String name;
```
名前

生成時に設定

```
private String password;
```
パスワード

生成時に設定

```
private bingoPlayer opponent;
```
対戦相手の操作者

相手が見つかった時に設定

```
private int state;
```
状態

アクセスごとに設定・更新（最初は”STATE_NULL”）

```
private String figure;
```
前のアクセスでの手（選択）

相手がまだ数字選択をしていないときに設定

```
private int gameOrder;
```
ゲームの大きさ(ex.3×3、5×5)

拡張用：相手が見つかった時に一律“5”に設定

```
private String bingoCard;
```
ビンゴカードの内容

相手が見つかった時に設定、両者が数字選択するたびに更新

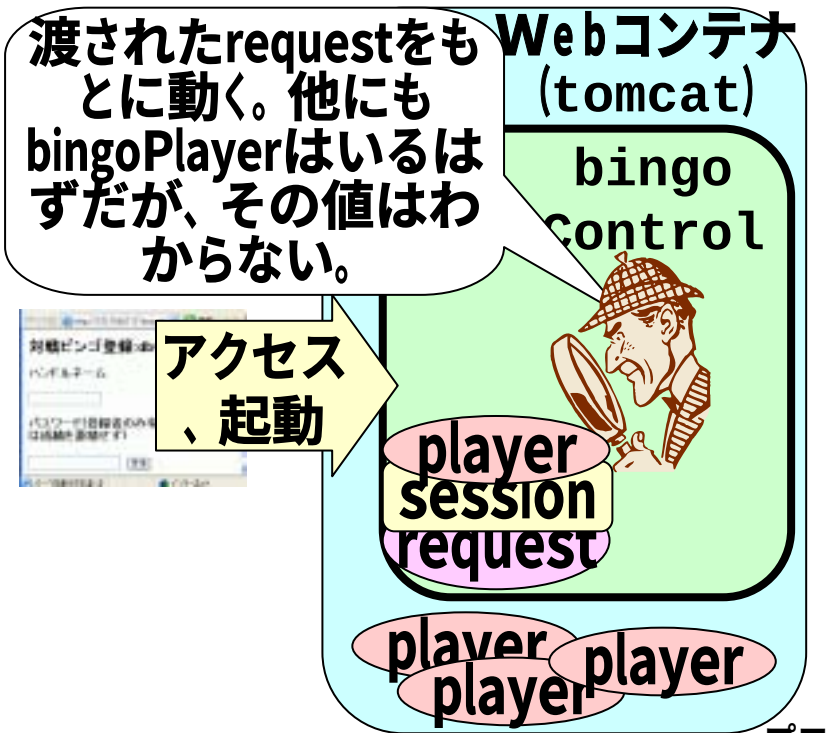
```
private String gameRecords;
```
DB関連データ（戦績）

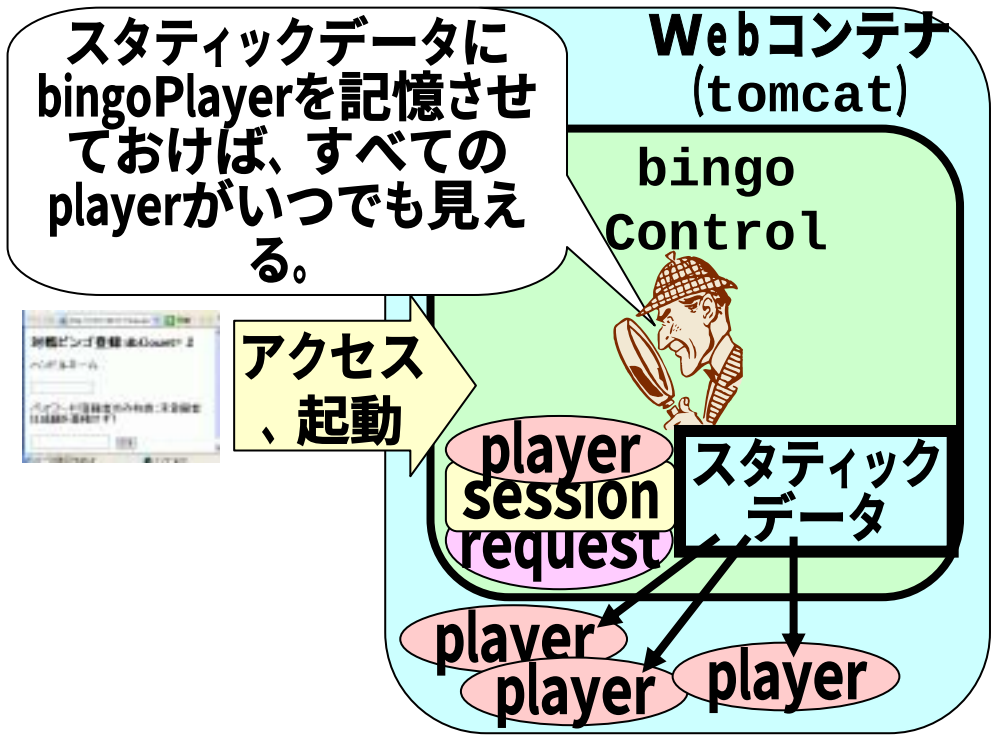
```
private int playerId;
```
DB関連データ（ＤＢ上の登録ＩＤ）

(6．4．1) スタティックデータ

■このビンゴゲームでは、同じ名前の競技者が戦うことを禁止しました(正確に言うと、同じ名前の競技者は、同時に1ゲームしかプレイできません)。

■この制限を設けるためには、スタティックデータを用いる必要があります。





（6．4．2）“bingoPlayers[]”

■bingoPlayerのインスタンスの一覧を保持するのが、クラス“bingoPlayer”の中のスタティックデータである、“bingoPlayers”です。

最初のアクセス

bingo Control

newPlayer

①新しいbingoPlayerを作ってくれ。

②空き(null)があって、同じ名前がいなければＯＫ。生成して、bingoPlayersに保存し、値を返すよ。

<bingoPlayers>

bingo Player

bingoPlayerのインスタンス1
bingoPlayerのインスタンス2
null⇒値を設定
bingoPlayerのインスタンスn
:
null

ゲーム終了

bingo Control

invalidate

①もうこのインスタンスは使わない。

②bingoPlayerから削除(=nullを設定)しておくよ。

<bingoPlayers>

bingo Player

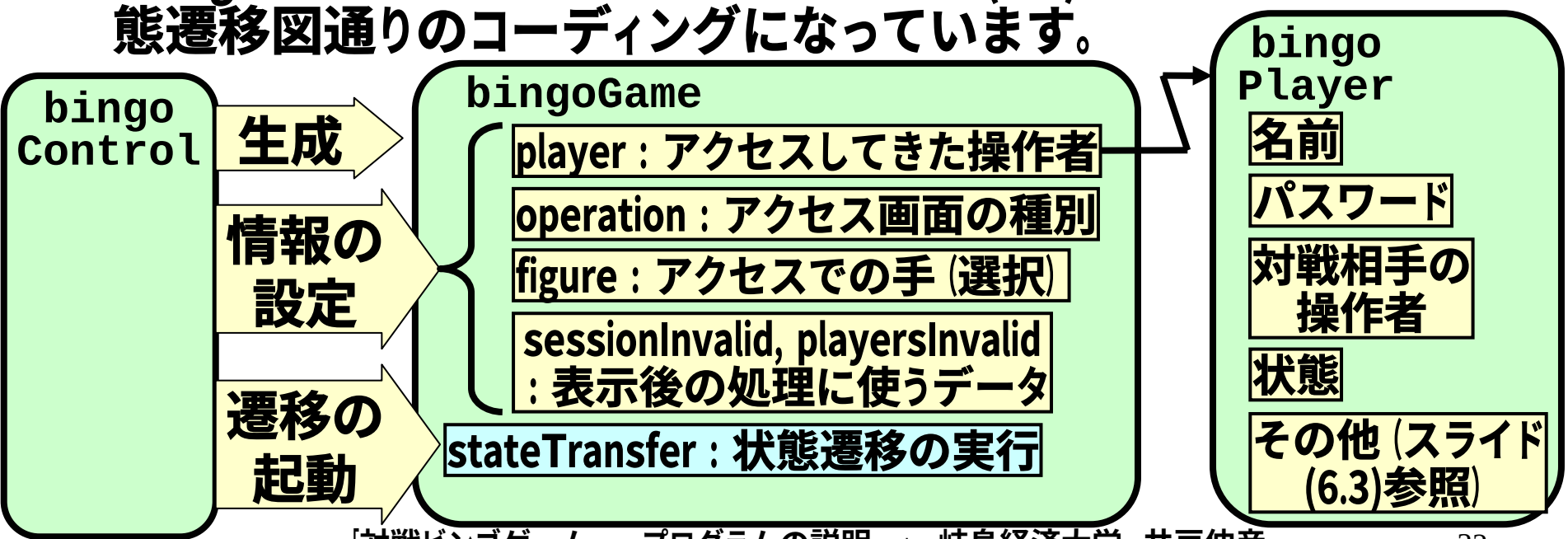
bingoPlayerのインスタンス1
bingoPlayerのインスタンス2
インスタンス⇒null
bingoPlayerのインスタンスn
:
null

(6．5) 状態遷移

■アクセスを受けた“bingoControl”では、次のように“bingoGame”の状態遷移を起動します。

- “bingoGame”をインスタンス生成する。
- “bingoGame”に、必要な情報(playerなど)を設定する。
  - “bingoPlayer”には、対戦相手や状態など、状態遷移に必要な情報が入っています(最初のアクセスの時は、bingoPlayerを生成します)。
- 状態遷移を起動する。

■“bingoGame”の状態遷移では、スライド(3.5)に記したように、状態遷移図通りのコーディングになっています。



(6．6) ゲームのロジック

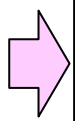
■ビンゴゲームのロジックは極めて簡単です。

- (1) 数字決め
  - ◆競技者によって選ばれた数字により、消去する2桁の数字を決める。
- (2) 検索･消去
  - ◆ビンゴカード中で2桁の数字を検索し、消去する。
- (3) ビンゴ確認
  - ◆消去されたマスが、縦横斜めに繋がっているかを確認する。

(1) 数字きめ
･自身の選択：5
･相手の選択：4
⇒ 数字54

(2) 検索･消去

45	BB	44	54	42
11	23	BB	BB	25
14	51	35	BB	31
33	BB	21	BB	12
13	52	41	BB	34



45	BB	44	BB	42
11	23	BB	BB	25
14	51	35	BB	31
33	BB	21	BB	12
13	52	41	BB	34

(3) ビンゴ判定

45	BB	44	BB	42
11	23	BB	BB	25
14	51	35	BB	31
33	BB	21	BB	12
13	52	41	BB	34

(7．1) データベースの利用

■次の2つの用途にデータベースを用いています。

- 競技者の管理
  - 競技者ID、名前、パスワードを管理します。
  - データベースに登録されていなくてもゲームは出来ますが、次項の成績は記録されません。
- 勝敗成績の管理
  - 決着のついたゲームごとに、ゲームID、日付、勝者、敗者、引き分けか否かを管理します。

■テーブル構成

<競技者 (Players) >

競技者ID (player_id)	名前 (name)	パスワード (passwd)
integer	character	character
1	ito	xxxxxx
2	suzuki	yyyyyy
:	:	:

<成績 (games) >

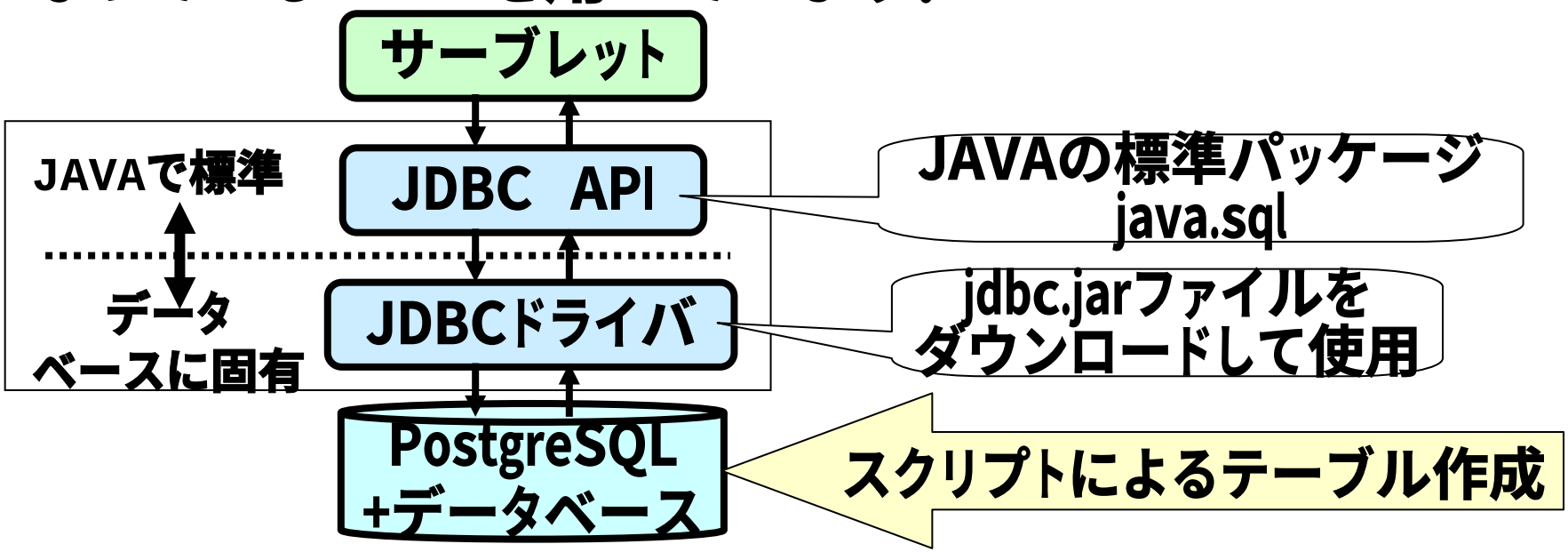
ゲームID (game_id)	日付 (date)	勝者 (winner)	敗者 (looser)	引分 (draw)
integer	date	integer	integer	boolean
1	2003.5.2	2	3	false
2	2003.5.3	3	2	true
:	:			

(7．2) データベースシステム

■本ゲームでは、フリーソフトウェアのデータベース、“PostgreSQL”を採用しています。

■テーブルの作成、およびユーザ登録は、スクリプトファイルを作成し、オフラインで行います。

■サーブレットからのアクセスは、事実上業界標準となっているJDBCを用いています。



(7．3) テーブル作成

■テーブル作成のスクリプトは次のとおりとなっています。

```
create table players
(player_id  serial       primary key,
 name       varchar(64) not null,
 passwd     varchar(64) not null unique);
create table games
(game_id    serial       primary key,
 date       date,
 winner     integer      not null references players,
 looser     integer      not null references players,
 draw       boolean      not null);
```

■gameテーブル中、“winner”と“looser”とは、playerテーブルの“player_id”を外部キーとしています。すなわち、playerテーブルの“player_id”に無い番号は、gameテーブルの“winner”と“looser”に設定出来ません。

(7．4) JDBCによるデータベース利用

■データベースを利用するための基本的なステップは、次の7つから成ります。

初期設定にて実施	①JDBCドライバをロードする。 ②データベースに接続するためのURLを定義する。 ③接続 (Connection) を確立する。
検索、更新にて実施	④Statementオブジェクトを作る。 ⑤クエリまたはアップデートを実行する。 ⑥結果を処理する。
システム停止時に実施	⑦接続をクローズする。

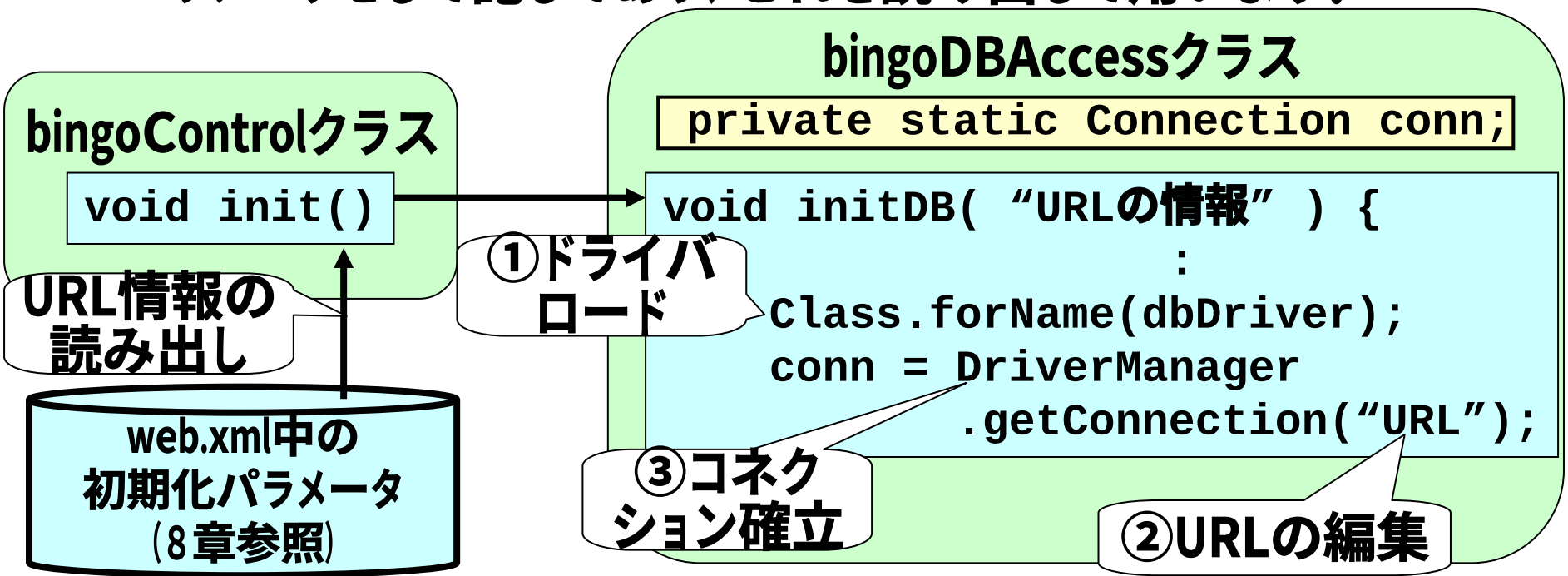
■クラス構成

- ビンゴゲームでは、“bingoDBAccess”クラスがデータベースアクセスをまとめて行います。

(7．4．1) 初期設定

■①から③のステップは、初期設定にて行います。

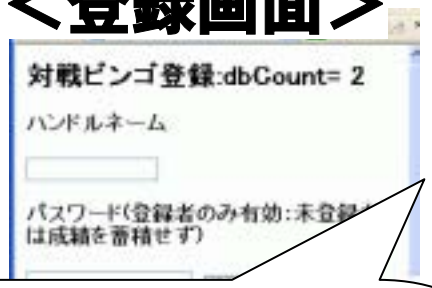
- bingoControlクラスの”initメソッド”(スライド(6)参照)から、“initDB”メソッドが呼ばれ、ここで最初の3つのステップを実行します。
- データベースのURLは、“web.xml”というファイルに初期化パラメータとして記してあり、これを読み出して用います。



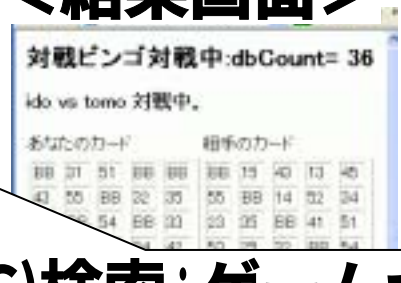
(7．4．2) 検索･更新

■検索･更新は、次の3つの場合に行います。

＜登録画面＞



＜結果画面＞



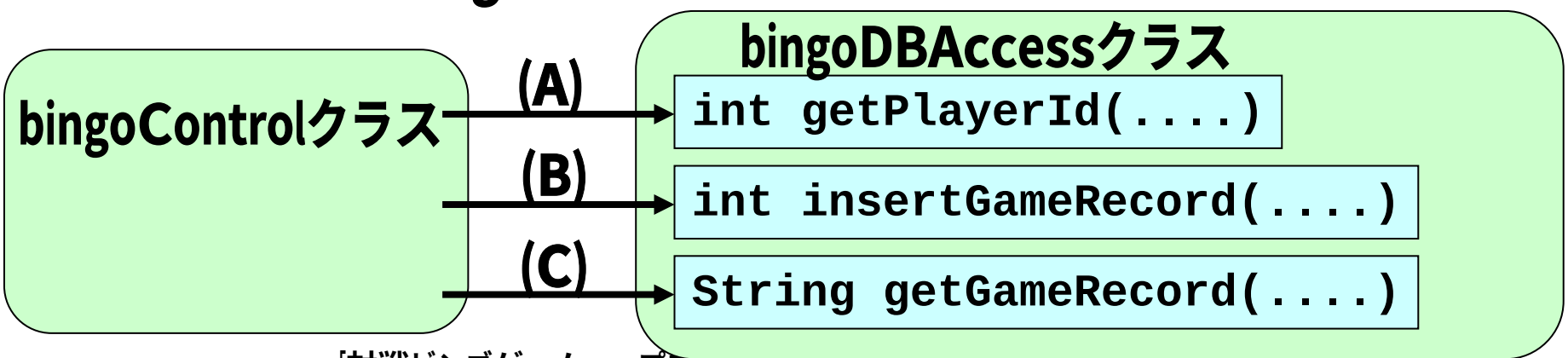
(A)検索：競技者が名前とパスワードを入力してきたとき。

・・・・対戦中・・・・

(B)更新：ゲームが終了して、その結果をデータベースに書き込むとき

(C)検索：ゲームが終了して、いままでの成績を競技者に示すとき

■それぞれ、bingoDBAccessの次のメソッドを用います。



(7．4．3) 検索の実施

■getPlayerIdメソッド、getGameRecordメソッドでは、次のように検索を行っています (更新もほぼ同様です)。

④Statementオブジェクトを作る。

- 文字列としてSQLコマンドを編集する。
- コネクションオブジェクトのcreateStatementメソッドを実行する。

⑤クエリまたはアップデートを実行する。

- statementオブジェクトのexecuteQueryメソッドを実行する。

⑥結果を処理する。

- 実行で得られた、ResultSetを読み出す。

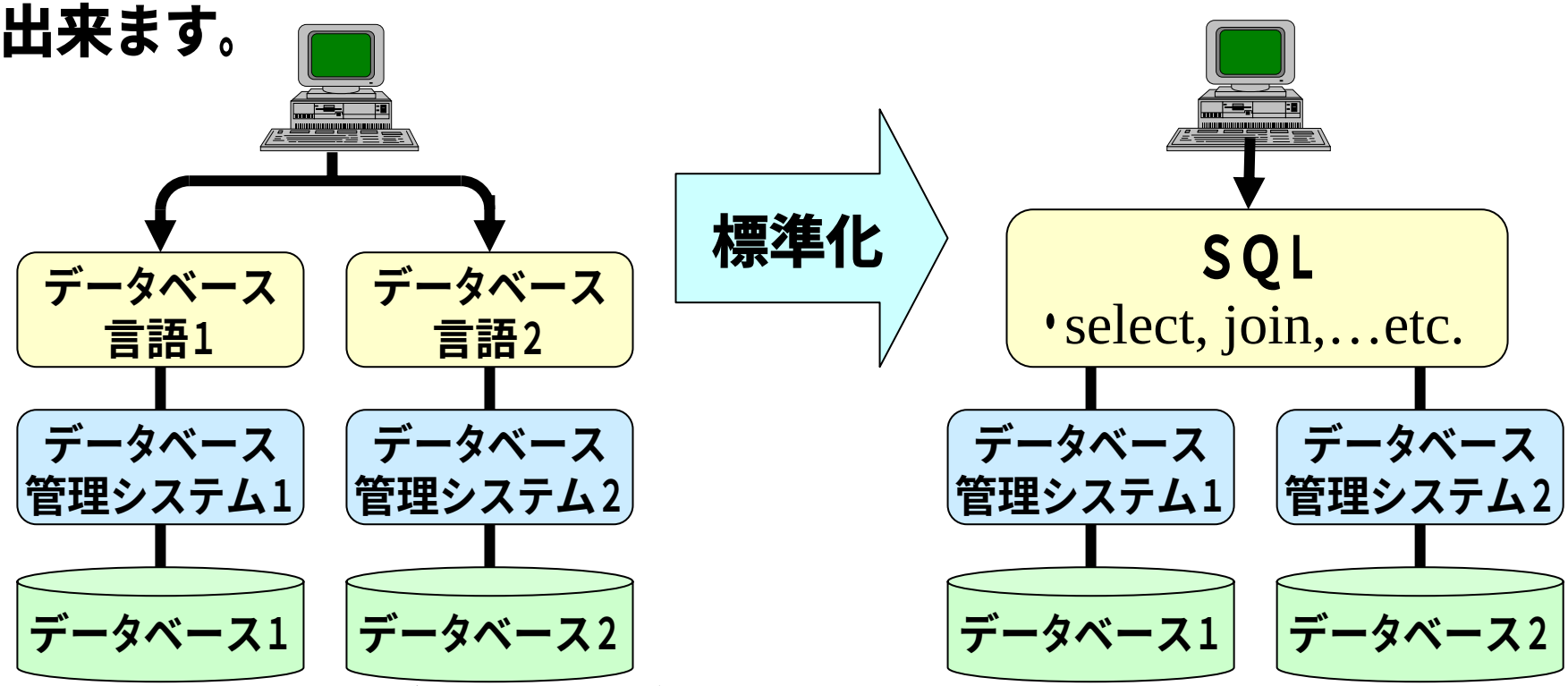
④
```
String selectSQL
   = "select player_id from players where name='"
     +name+"' and passwd='"+password+"'";
Statement stmt = conn.createStatement();
```
⑤
```
ResultSet rslt = stmt.executeQuery(selectSQL);
```
⑥
```
if(rslt.next()){
    int playerId = rslt.getInt(1);
```

(7．5．1) SQLコマンド

■データベースを扱うための、プログラミング言語データベース言語といいます。

■ＳＱＬは、データベース言語の標準化仕様です。

■標準化されているおかげで、さまざまなデータベース･データベース管理システムに対しても、同じプログラムを用いることが出来ます。



(7．5．2) SELECT句

■検索で用いるSELECT句は、SQL言語の文の中でも、最も頻繁に使用されものです。

■典型的な使い方を以下に示します。

SELCECT　M.伝票、M.商品コード、M.個数、S.商品名、S.単価

FROM　明細 M、商品 S

WHERE　M.商品コード＝S.商品コード

AND　S.単価＜800

これらの項目を取り出す。

これらのテーブルを元とし、明細はM、商品はSと表す。

明細の商品コードと商品の明細コードは一致。
かつ、商品の単価は800未満

＜明細 M＞

伝票	商品コード	個数
1101	101	22
1101	102	52
1105	201	67
1106	302	78

＜商品 S＞

商品コード	商品名	単価
101	のりせんべい	300
102	まめせんべい	500
201	ピーナッツ	600
302	おつまみ	1000

伝票	商品コード	個数	商品名	単価
1101	101	22	のりせんべい	300
1101	102	52	まめせんべい	500
1105	201	67	ピーナッツ	600

(8) 環境設定

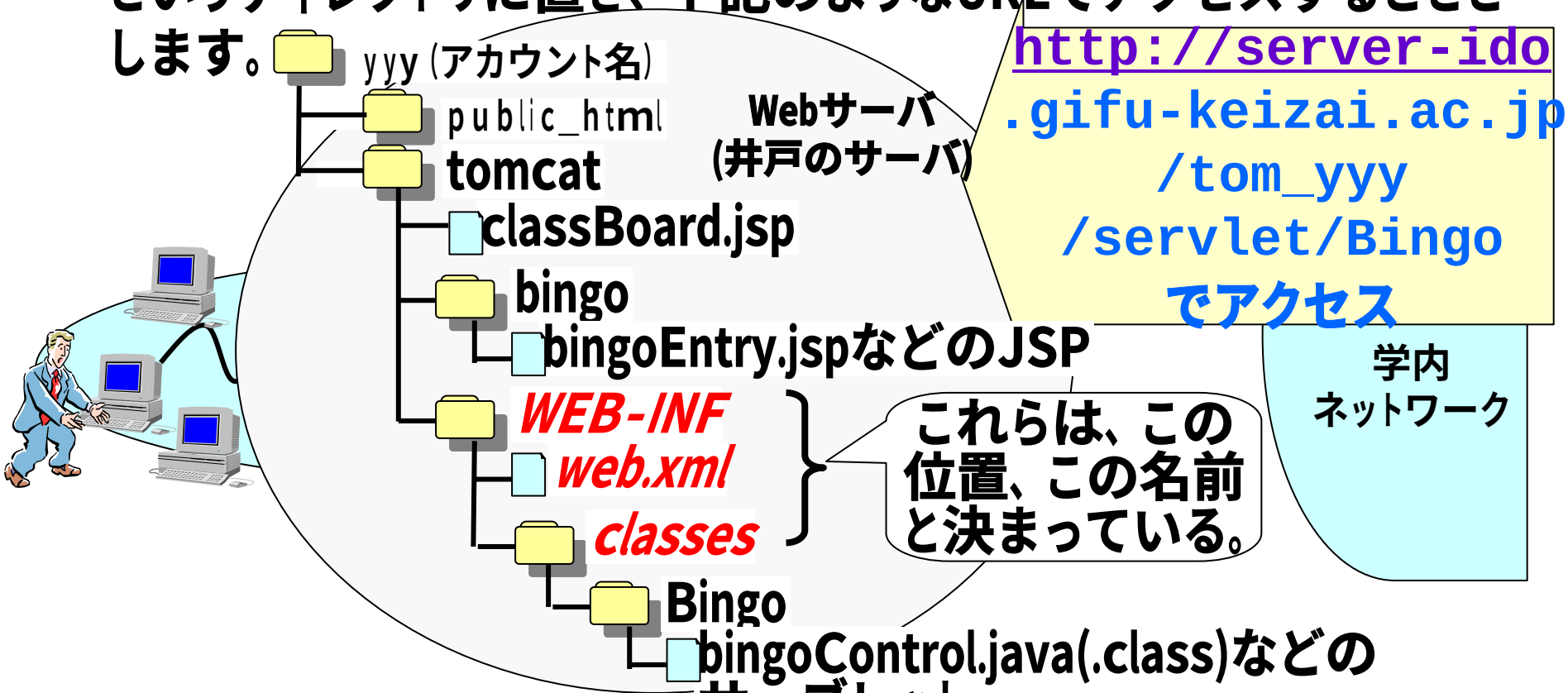
■ ビンゴゲームを動かすためには、次の環境設定が必要です。

1. アカウントの準備
   - 当たり前ですが、アカウントを準備します。このアカウントに対して、以下の環境を設定していきます。
2. ディレクトリの準備
   - Tomcatにより決められたディレクトリ構成配下に所定のファイルを置く必要があります。
3. システムファイルの設定
   - WebサーバであるApacheと、WebコンテナのTomcatにて、上記のアカウント・ディレクトリが使えるように設定を行います。

(8．1) ファイルの配置

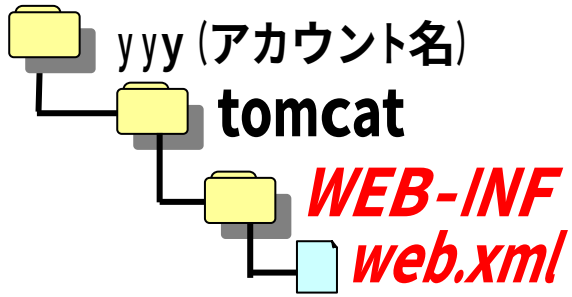
■サーブレットプログラムのファイルは、TOMCATで環境設定したディレクトリに置く必要があります。

■井戸のサーバでは、ゼミ生の各アカウント(=“yyy”)の“tomcat”というディレクトリに置き、下記のようなURLでアクセスすることとします。



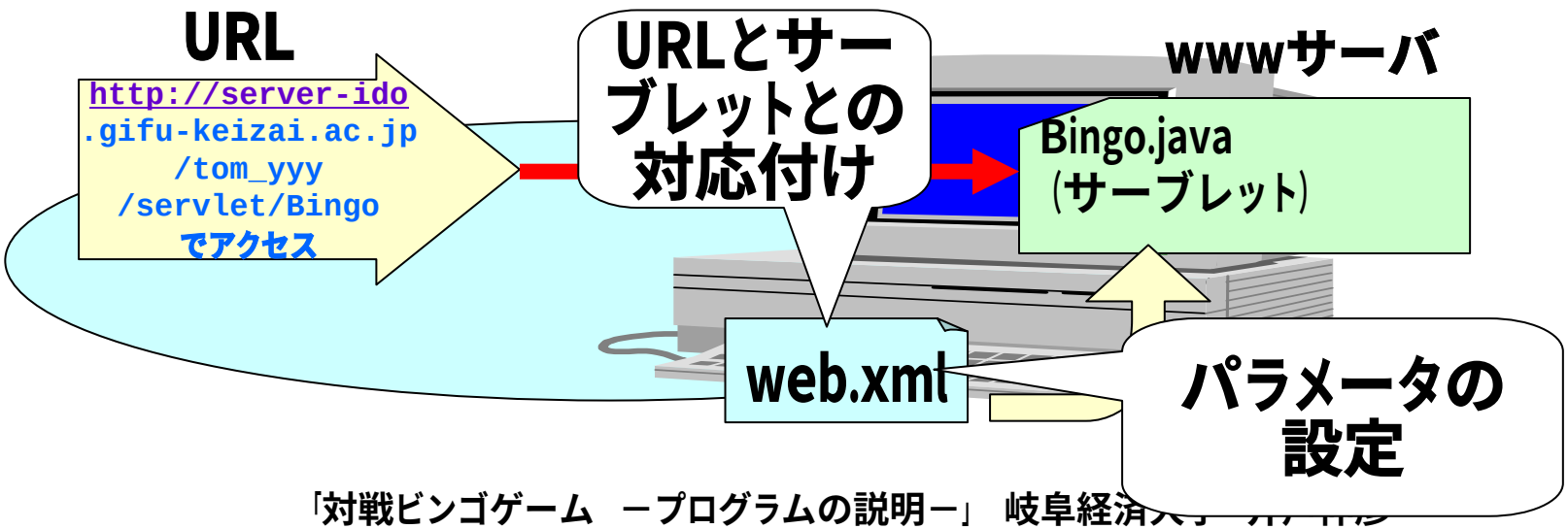
(8．2) web.xmlファイル

■web.xmlファイルは環境設定の
中心となるファイルです。



■ビンゴゲームのサーブレットでは、
web.xmlにより次のような設定がなされています。

- サイトにアクセスするURLとこれを受けて動作するサーブレットとの対応付け。
- サーブレットが動作する際のパラメータ設定



（8．2．1）web.xmlの形式

■web.xmlファイルは、XML文書として作成されています。

■XML文書の形式の指定

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE web-app *(View Source for full doctype...)*>

- ここでは、どのようなXML文書であるかを記述しています。

■要素名

<web-app>

　　中略

</web-app>

- 大雑把にいうと、この2つの間にWeb関連の設定を行うことが示されています。XMLは、常にこのような書き方をします。
- 以下の<servlet>、<serevlet-mapping>、<session-config>も同様に理解しておいてください。

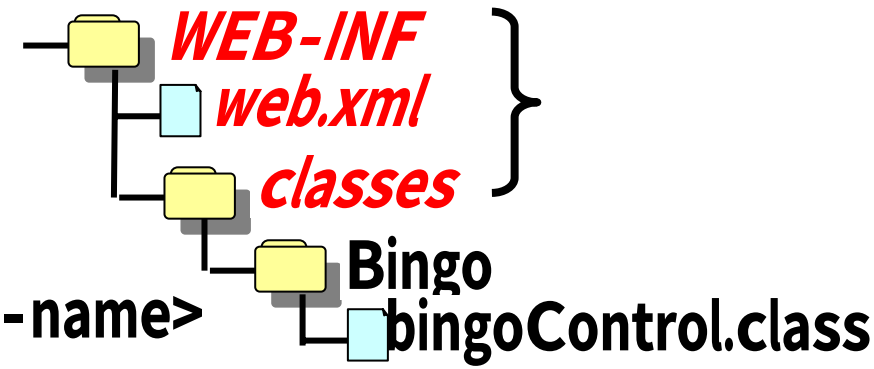
（8．2．2）URLとの対応付け

■サーブレットの命名

<servlet-name>Bingo</servlet-name>

<servlet-class>Bingo.bingoControl</servlet-class>

- ここでは、サーブレットのファイル“classes/Bingo/bingoControl.class”を、“Bingo”という名前で指すことが書かれています。



■名前とURLの対応

<servlet-mapping>

<servlet-name>Bingo</servlet-name>

<url-pattern>/tomcat/*</url-pattern>

</servlet-mapping>

- ここでは、先ほどBingoと命名したサーブレットへのURLを決めています。

(8．2．3) パラメタ設定

■パラメタ

<init-param>

<param-name>**logApi-init-file**</param-name>

<param-value>**WEB-INF/classes/Bingo/bingoLog.prop**</param-value>

</init-param>

- プログラムの中に直接パラメタを書くと、プログラムの移植や流用時に不都合が生じます。ここでは、プログラムに渡すパラメタを設定しています。(7) データベースへのURLもここに設定されています。

■スタート時の形態

<load-on-startup>**1**</load-on-startup>

- TOMCATが立ち上がったときに、サーブレットが直ぐに動ける状態になることを指定します。指定しないと、実際にURLにアクセスされてから、準備を整え始めます。

■タイムアウト

<session-config>

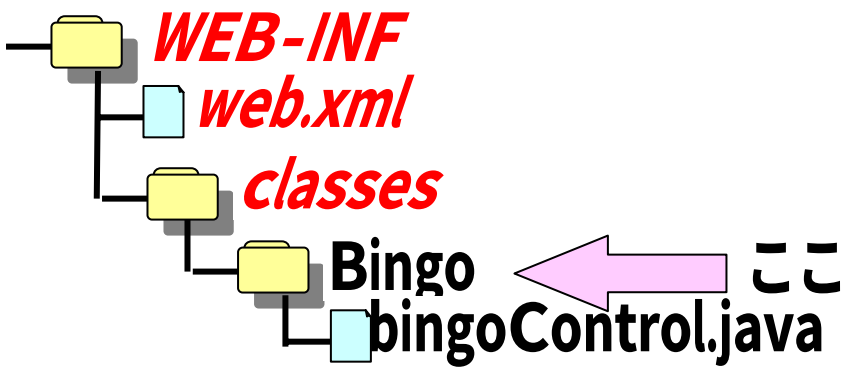
<session-timeout>**1**</session-timeout>

</session-config>

- TOMCATは、サーブレットにクライアントからアクセスがあってから、一定時間アクセスがないと、セッションを切ります。この時間 (分単位) を設定します。上記の値はデバッグ用に短くしたもので、デフォルトは３０です。

(8．3) コンパイル

■次のディレクトリにて行います。



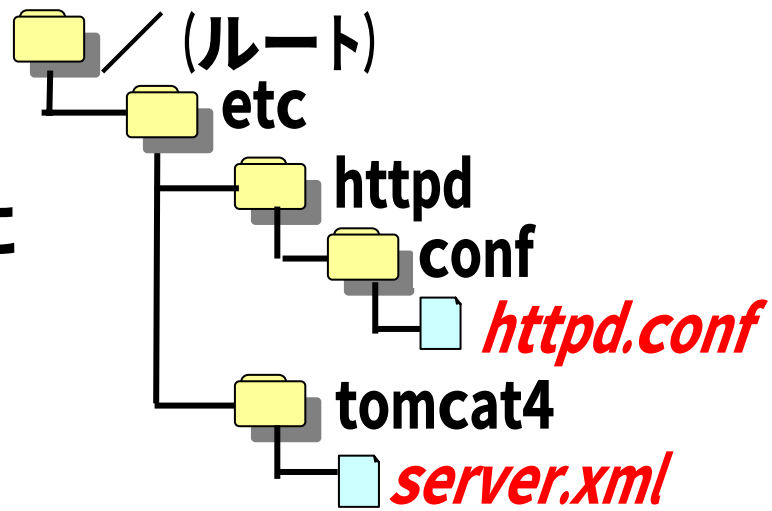
- `cd ~yyy/tomcat/WEB-INF/classes/Bingo`
- `javac Bingo/bingoControl.java`

■ビンゴのプログラムは、すべて“Bingo”というパッケージに含まれます。“Bingo”というディレクトリの配下にプログラムを置き、上記のようにコンパイルするのは、Javaでのパッケージのコンパイルの規則に則っています。

(8．4) システムの設定

■スライドユーザ(8.1)に記したユーザディレクトリが、サーブレット・JSPのディレクトリとして使えるようにするために、システムでは、次の2つのファイルに設定を行います。

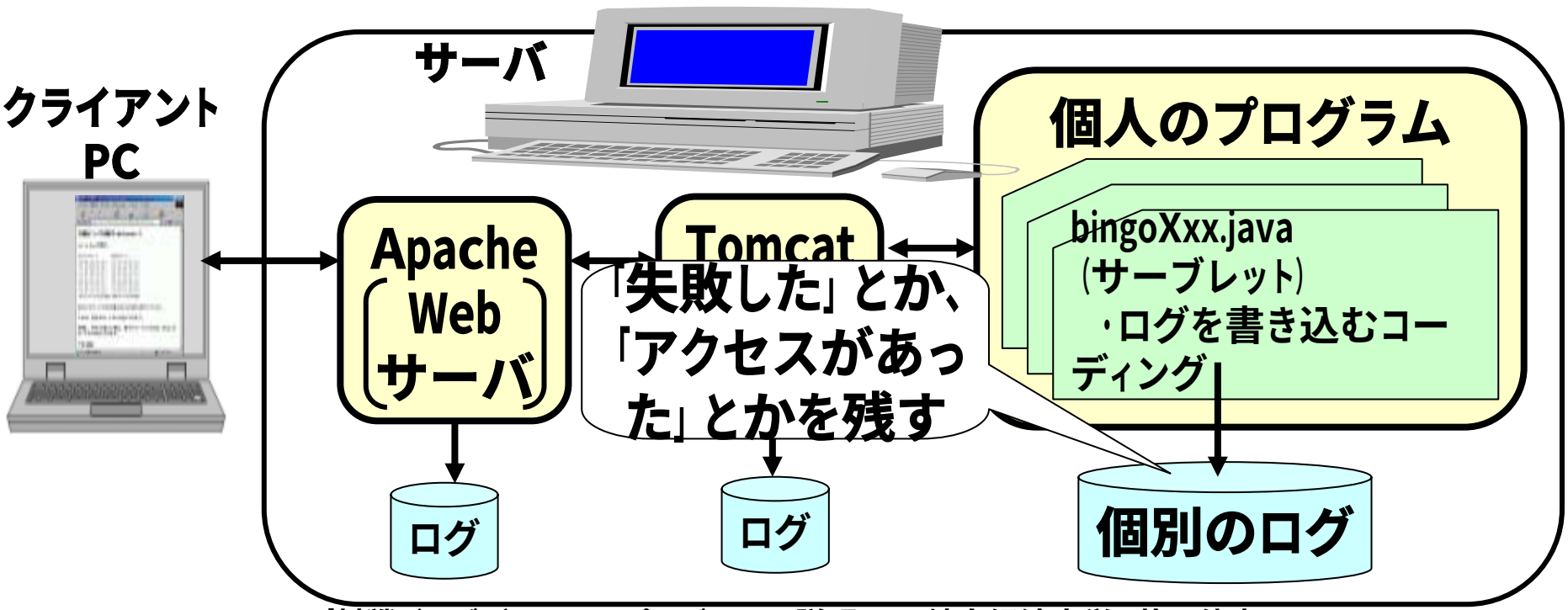
- /etc/httpd/conf/httpd.conf :
  指定したURLへの要求を受け取ると、
  TOMCATへ転送する。
- /etc/tomcat4/server.conf
  要求のあったURLを、ディレクトリに
  対応付ける。



(9．1) ログとは何か？

■ログとは、ソフトウェアが動作している時に、その動作上の履歴をファイルに残すことをいいます。

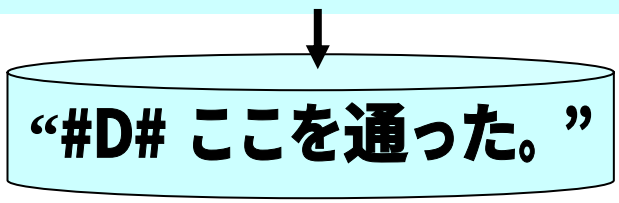
■ここでは、ApacheやTomcatが残すログではなく、我々が作るゲームプログラムが独自に残すログについて記します。



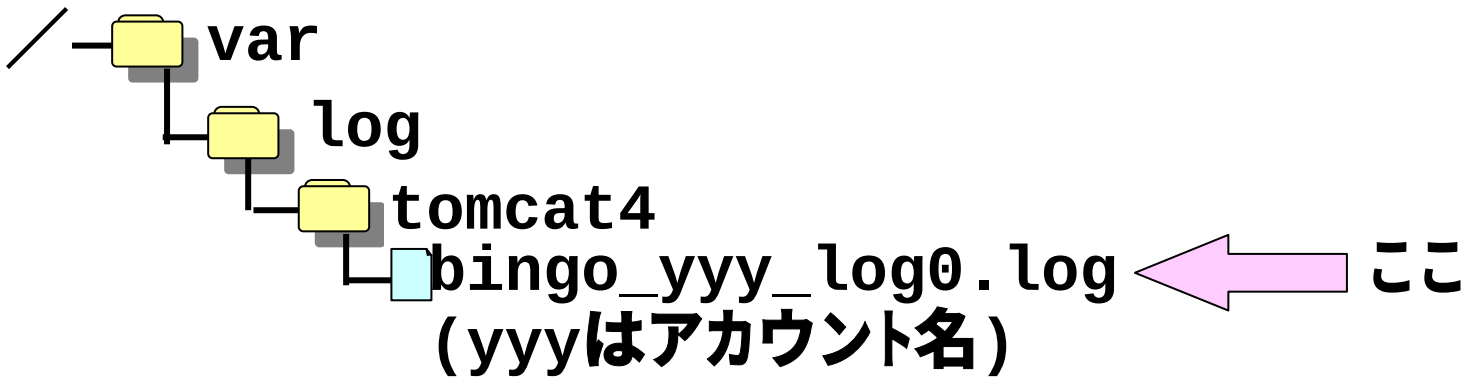
(9．2) ログの書き込み

■ログの書き込み自体は簡単です。プログラム中、次のようなラインは、ログの書き込みを行っています。

```
logger.fine(“#D# ここを通った。");
```



■ログのファイルは、次の場所に生成されています。

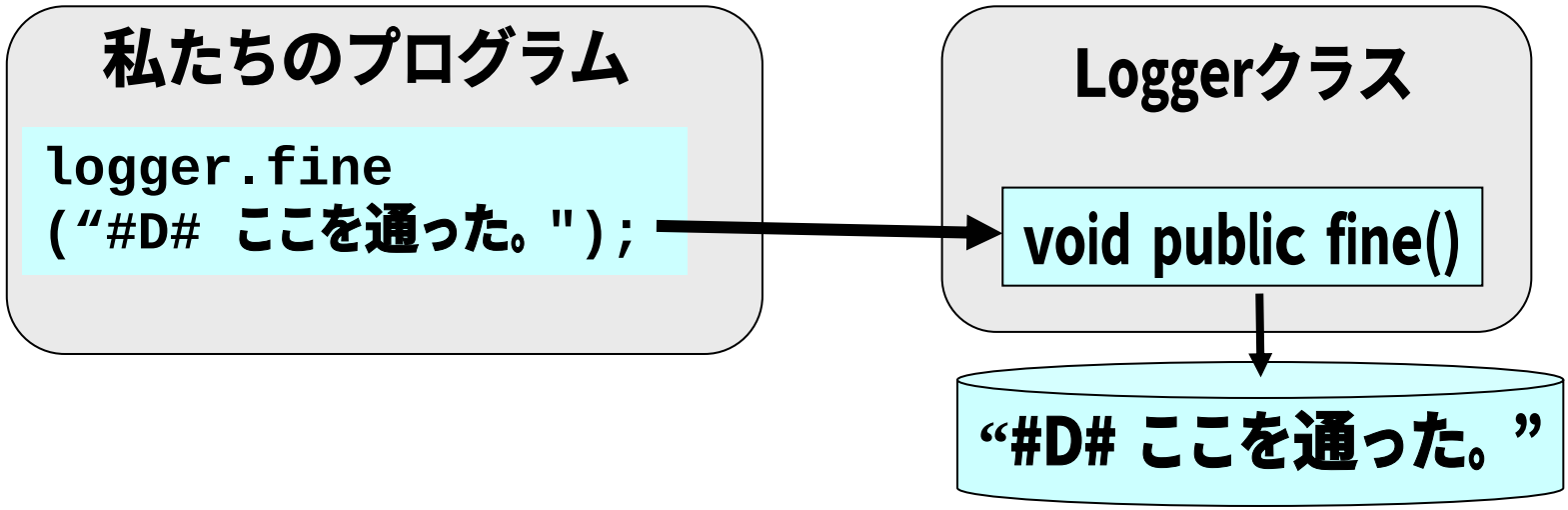


■次のようにコマンド入力してログの内容を見てください。

- `cd /var/log/tomcat4`
- `less bingo_yyy0.log`

(9．3) Loggerクラス

■前スライドのように簡単にログが取れるのは、Javaの標準クラスである“java.util.logging.Logger”を使っているからです。



■Loggerクラスは、次のようにインポートされることにより、利用可能となります。

- `import java.util.logging.Logger;`
- `import java.util.logging.Level;`
- `import java.util.logging.LogManager;`

（9．4）Loggerクラスの使い方

■まず、インスタンスを生成します。

- protected static Logger logger = Logger.getLogger(bingoControl.class.getName());
- 上記の下線部は、ビンゴクラスの名前になります。クラスの名前が同じであるならば、別ファイルで生成しても、実体は同じものになります。

■次に、生成したインスタンスに、どのように動作するかの指示を、設定ファイルによって知らせます（bingoControlでやっています）。

- `LogManager.getLogManager().readConfiguration(inputStream);`
- 上記の下線部は、ファイルの読み込み口となる“インプットストリーム”というクラスのインスタンスです。

(9．5) ファイル取得までの動作

■前スライドのように、Loggerクラス自体の使い方は簡単ですが、BingoControlでは、次のようにファイル取得を行っています。