第6章 JSP+JavaBean 技术

JSP 网页内嵌 Java 代码，可以实现网页的动态功能，但是，如果把实现复杂逻辑功能的代码也直接写在 JSP 网页内，大量的客户端 HTML 源码与服务端 Java 代码混合交集，大大降低程序的可阅读性，从而增加程序的维护难度。JSP 网页是以顺序过程的方式实现的，把实现复杂逻辑功能的代码也直接写在 JSP 网页内，这不符合面向对象及组件的设计思想，不能很好发挥 Java 语言面向对象的特色，程序结构性、重用性都差。

为了程序结构清晰，便于维护，应尽量减少 JSP 页内的嵌入代码。如果一个 JSP 页的实现较复杂，逻辑功能代码较多，就要考虑将 JSP 的页面功能与功能逻辑代码分离，把功能逻辑代码封装为 Java 类（称为 JavaBean），JSP 网页主要负责界面的输入、输出，在 JSP 页内调用 Java 类或实例的方法。这就是本章要讲的 JSP+JavaBean 技术。

使用 JSP+JavaBean 技术，可以实现 Web 应用程序的分层架构，图 6-1 是一种非常典型的层次架构，用 JSP 网页实现用户表示层（USL），通过 JavaBean 实现业务逻辑层的功能及连接其他组件、基础设施。由于本书篇幅限制及 JSP 定位，本书不深入讨论程序架构知识，请参考相关书籍。

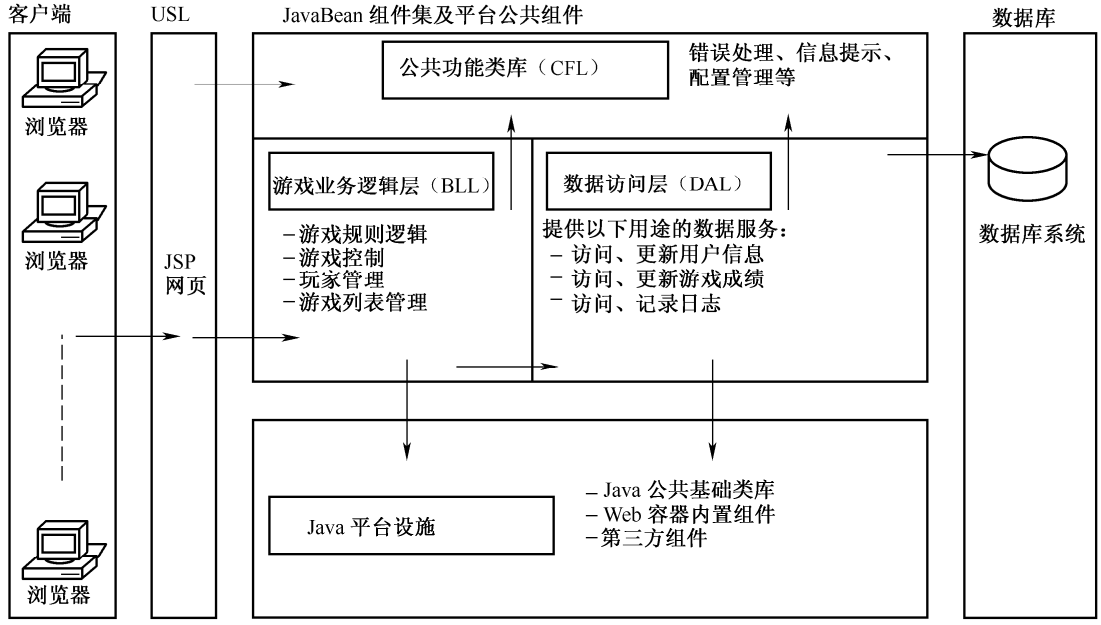

图 6-1　一种典型的基于 JSP+JavaBean 技术的层次架构

我们在前面的案例实践中，已经使用过 JSP+JavaBean 技术，例如：

- 为了能够多次重用连接数据库的代码，我们设计一个连接数据库类 dbConnect。
- 为了 JSP 网页及程序结构清晰，在设计用户信息页 userInfo.jsp 时，把用户信息的相关属性及检索、更新方法，封装到一个名为 userInfoBean 的 Java 类，在 userInfo.jsp 页中调用其对象方法。

下面进一步推进 99Game 案例进度，就如何完成实现游戏厅、游戏室、游戏桌等功能任务，介绍 JSP+JavaBean 技术的应用。

6.1 任务：游戏大厅、房间、牌桌

1. 游戏大厅

玩家登录后进入游戏大厅，游戏大厅预设多个不同级别的游戏房间，供玩家选择进入，它相当于选择游戏的总菜单，页面 gamehome.jsp 大致如图 6-2 所示。

图 6-2 游戏大厅

游戏房间的信息保存在数据库表 roominfo 中，允许管理员注册添加新的游戏房间。

2. 游戏房间

每个游戏房间内预置 20 个游戏桌，玩家可以单击游戏桌的空位加入游戏。页面要定时刷新，以反映游戏房间最新情况。游戏房间页（gameroom.jsp）大致如图 6-3 所示。

图 6-3　游戏房间

3．游戏牌桌

玩家在游戏房间页上单击某牌桌的空位，进入到此牌桌页（gamedesk.jsp），在牌桌页等其他玩家加入，或开始游戏，如图 6-4 所示。

图 6-4　进入游戏牌桌页，开始游戏

页面要定时刷新，以反映游戏最新情况，用户离开或关闭游戏牌桌页时，表示用户退出当前游戏。

6.2 方案：技术分析与实现要点

6.2.1 面向对象分析与设计

我们从案例的需求出发，识别出需求中的游戏大厅、游戏房间、游戏牌桌、游戏玩家等重要概念，进一步考虑它们的状态、数据及功能、动作，按照面向对象的设计思路，分别设计为游戏大厅（GameHome）、游戏房间（GameRoom）、游戏牌桌（GameDesk）、游戏玩家（GamePlayer）等 Java 程序类（JavaBean），并用类图形式勾勒它们的属性、行为及对象关系，如图 6-5 所示。

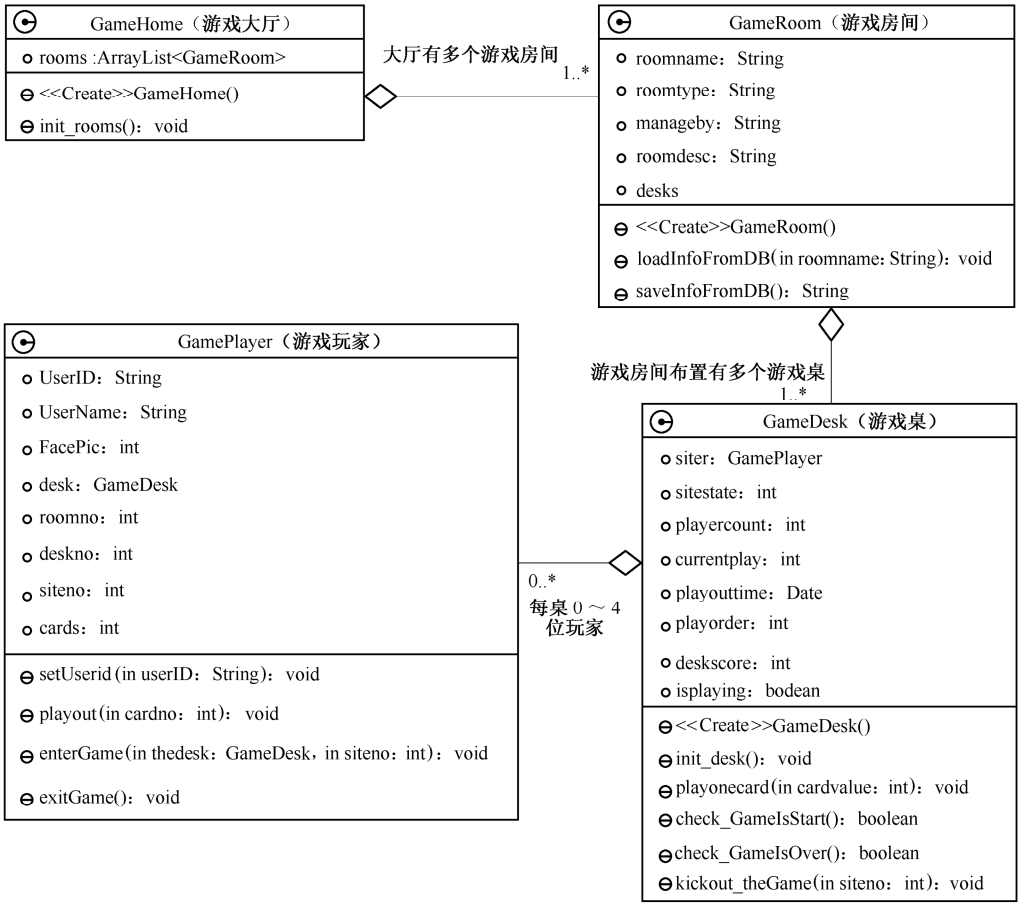

图 6-5 游戏主要类的类图

在游戏的对象模型中，GameHome 类中的属性 rooms，是元素个数可变的数组列表类型（ArrayList），表示游戏大厅里有多个游戏房间。GameRoom 类中的属性 desks，是 GameDesk 的数组，表达一个游戏房间布置多张游戏牌桌。GameGesk 类中的属性 GamePlayer[] siter = new

GamePlayer[4]，表示有 4 个游戏玩家座位，可以坐 0～4 位玩家。GamePlayer 类中的属性 Int[] cards，表示玩家的多张牌，属性 desk 用于从玩家对象导航找到玩家所在的牌桌对象。

我们再进一步对类的职责进行分工，把有关游戏整体的行为方法，如发牌开始游戏、出牌游戏规则、某座位玩家出局、终止游戏等，封装到 GameDesk 类；把有关玩家的操作游戏的行为处理方法，如加入游戏、出牌动作、玩家强退等，封装到 GamePlayer 类。

6.2.2 游戏对象模型的代码结构

根据上面的对象设计模型，我们可以推导出模型框架代码。阅读下面类的框架代码，进一步熟悉各主要类的状态数据及职责分工。

1．游戏大厅

```
public class GameHome
{
    //   大厅内有多个游戏室
    public ArrayList<GameRoom> rooms= new ArrayList<GameRoom>();
    public GameHome()//构造函数
    {
        //实例化时，调用 init_rooms()，进行初始化，设置房间
    }
    public void init_rooms()
    {
       //进行初始化，设置房间
    }
}
```

在游戏对象模型中，游戏大厅是对象间访问导航的开始点，访问导航线索：游戏大厅→游戏房间→游戏牌桌→游戏玩家，其对象的生存期是整个应用程序的生命周期。

2．游戏房间

```
public class GameRoom
{
    String roomname=""; //房间名
    String roomtype=""; //类型
    String manageby=""; //管理者
    String roomdesc=""; //描述
    //每个房间预设 20 张游戏桌
    GameDesk[] desks= new GameDesk[20];
    public GameRoom()
    {   //初始化，生成房间的所有游戏桌对象
        for(int i=0;i<desks.length;i++)
            desks[i]=new GameDesk();
    }
public void loadInfoFromDB(String rmname)
    {
    //从数据库表检索到此游戏房间的信息，如房间名、等级、描述等，设置属性
```

```
    }
    public String saveInfoFromDB()
    {
    //用户在网页上更改房间信息，如房间名、等级、描述等，存回数据库
    }
}
```

在游戏对象模型中，游戏房间是对游戏桌进行分类管理的一种手段，其本身不带有游戏逻辑功能，其对象依附于游戏大厅对象。

3．游戏牌桌

```
public class GameDesk //游戏桌，代表整个游戏
{
    public GamePlayer[] siter = new GamePlayer[4];//4 个位置的玩家
    public int[] sitestate = new int[4];//4 个位置的状态
    public int playercount;//玩家人数
    public int currentplay;//轮到的当前出牌者的位置号
    public java.util.Date playouttime;//出牌超时计时开始时间
    public int playorder;//出牌顺序:1--顺时针，-1--逆时针
    public int deskscore;//桌面牌分
    public boolean isplaying;//是否打牌状态

    public GameDesk()
    {
        init_desk();//初始化
    }
    //初始化游戏桌
    public void init_desk()
    {
        //初始化各成员属性
    }
    public void playonecard(int cardnum)
    {
       //实现出牌规则：出某种牌，游戏状态如何变化，桌面总分累计，
       //按规则设置下一出牌玩家，记忆下一玩家出牌超时计时的开始时间等
    }
    public boolean check_GameIsStart()
    {
        //检查玩家是否足四人，够 4 人就发牌，设置游戏开始标志
       //发牌算法：为每个玩家对象生成 10 个随机数，表示 10 张牌
    }
    public boolean check_GameIsOver()
    {
        //检查游戏是否结束:在游戏进行态中，只剩一个玩家时，游戏结束
        //游戏结束时计算各玩家得分，并存到数据库中，设置游戏结束标志
    }
    public void kickout_theGame(int siteno)
```

```
    {
       //某座位玩家出局，在相应的 sitestate[座位]中设置为第几位出局者
       //sitestate[座位]>0 表示对应座位玩家已出局，不能再出牌
    }
}
```

在游戏对象模型中，游戏桌是一个重要的概念，代表游戏，即封装具有游戏的属性与行为，是游戏的主要逻辑功能类，游戏的核心算法——出牌规则，就是它的一个方法。其属性表示着游戏的重要状态：打牌状态、桌面牌分、出牌顺序、出牌计时、位置状态、当前出牌座位等，其中，用 GamePlayer[] siter 表示座位上的玩家，规定如下：

```
Siter[座位号]==null          表示此座位空闲
Siter[座位号]==theplayer     表示玩家对象 theplayer 坐在此座位
```

还有，**int**[] sitestate=**new int**[4]，表示 4 个位置的出牌状态，规定如下：

sitestate[座位号]==0　　表示此座位没出局，可以出牌

sitestate[座位号]==2　　表示此座位已出局，不可以出牌，是第 2 位出局

4．游戏玩家

```
public class GamePlayer
{
   public String UserID;//用户 ID
   public String UserName;//用户姓名
   public int FacePic;//头像
   public GameDesk desk=null;//当前游戏所在桌对象，为 null 时表示没有游戏
   public int roomno,deskno,siteno;//用于记忆玩家所在的房号、桌号，座位号
   public int[] cards = new int[10];//手上的牌

   public void setUserid(String userID)
   {
       //设置玩家的 ID，同时从数据库检索玩家的详细信息：姓名、头像
   }
   //按游戏规则出牌、算分，是游戏规则的实现方法
   public synchronized void playout(int cardno)
   {
   //检查玩家可否打出第 cardno 张牌，
   //并调用 desk.playonecard(cardnum)，按规则算分、改变游戏状态
  }
   public void enterGame(GameDesk thedesk, int siteno)
   {
    //实现玩家加入某房某桌游戏，检查玩家是否足四人，够 4 人就发牌，开始游戏
    //thedesk.siter[siteno]=this，表示玩家坐上游戏桌的第 siteno 座位
   }
   public void exitGame() throws Exception
   {
       //强行退出游戏
   }
}
```

在游戏对象模型中，游戏玩家是另一个重要的概念，封装有玩家的状态属性与操作行为，主要负责玩家通过界面进行游戏操作的事件处理，有进入游戏（桌）、退出游戏（桌）、玩家出牌等。游戏玩家对象代表用户，其生命周期是整个 Session 过程，不管用户玩家是否坐在某桌游戏，游戏玩家对象在 Session 过程中始终存在。“游戏玩家坐在某桌游戏”用其属性 desk 指明。例如语句：

```
theplayer.desk=theroom.desks[i];//theplayer 玩家坐在 theroom 房间第 i 桌
```

6.2.3 JSP 界面页与 JavaBean 对象的关系设计

在 JSP+JavaBean 的程序结构布局中，我们尽可能贯彻 JSP、JavaBean 分工分层的思想：JSP 页负责界面显示及操作输入，主要的功能逻辑封装到 JavaBean。JSP 页对 JavaBean 对象的引用关系，以及对象间的导航关系，也需要重点规划、设计，如图 6-6 所示。

图 6-6　游戏 JSP 与 JavaBean 的关系设计

- 游戏大厅页 gamehome.jsp 中，通过如下动作：

```
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application"/>
```

设置引用游戏大厅对象，从游戏大厅的属性 rooms 取到所有的游戏房间信息，列表显示在网页上（见图 6-6）。

- 游戏房间页 gameroom.jsp 中，通过如下动作：

```
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application"/>
```

设置引用游戏大厅对象，从游戏大厅的属性 rooms[i]取到目标游戏房间信息，再通过游戏房间的属性 desks，导航取到此房间的所有游戏牌桌对象，显示在网页上（见图 6-6）。

- 游戏（桌）页 gamedesk.jsp 是玩家玩游戏（出牌）或观战的界面，网页上显示整个游戏的桌面状态（桌面分、座位上的玩家、玩家状态、当前出牌者等），以及当前玩家的手上牌值。在网页中设置引用对象（参看图 6-6）：

```
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application"/>
<jsp:useBean id="theplayer" class="mygame.GamePlayer" scope="session"/>
```

从 request 中取出传入的房间号、桌号等 URL 参数值，定位到游戏（牌桌）对象：

```
int roomno = Integer.parseInt(request.getParameter("room"));//房号
int deskno = Integer.parseInt(request.getParameter("desk"));//桌号
desk = gamehome.rooms.get(roomno).getDesks()[deskno];
```

如果玩家参与此游戏，玩家对象的 desk 属性引用所在游戏（牌桌）对象（参看上面设计的游戏对象模型），因此，也可以从游戏玩家对象 theplayer 属性 desk 取到当前玩家用户所在的牌桌对象。

6.2.4 使用 JavaBean 属性的 get/set 方法

如图 6-5 所示，在游戏房间 GameRoom 类中，为主要的属性设置 get/set 方法，例如，属性 roomname 的 get/set 方法：

```
public String getRoomname() {
        return roomname;
    }
    public void setRoomname(String roomname) {
        this.roomname = roomname;
    }
```

当用<jsp:usebean>引入 JavaBean 时，可以用<jsp:setProperty>把内置对象 request 中的参数的值传递给设置有 get/set 方法的属性。例如，在新建或更新游戏房间信息的网页 gameroominfo.jsp 中，如下设置会把请求对象 request 中的参数自动设置给 JavaBean 的同名属性（request 中的参数名与 JavaBean 的 setXXX 方法名的 XXX 部分比较，注意：request 中的参数名全部用小写字母，setXXX 方法名中的第一个 X 位置要大写，否则不会自动设置值，这可能是 JSP 的一个 Bug 吧）。

```
<jsp:useBean id="roominfo" class="mygame.GameRoom" scope="page">
  <jsp:setProperty name="roominfo" property="*" />
</jsp:useBean>
```

用这种方法取到 request 对象中的参数值，传给 JavaBean 中的相关属性，简单直观。属性的 get 方法常用来取出属性值，嵌入 JSP 页中，参考图 6-7 及下一节中的代码。

图 6-7 游戏房间信息页

6.3 实践：JSP+JavaBean 实现游戏

从对象模型分析与设计，我们可以推导出模型框架代码。在项目的 src/mygame 包文件夹下，创建游戏大厅类（GameHome）、游戏房间类（GameRoom）、游戏牌桌类（GameDesk）、

游戏玩家类（GamePlayer），并参照 6.2.2 节内容给出各 JavaBean 的框架性代码。

下面的实践步骤进一步按游戏需求编写功能类的具体实现代码，设计 JSP 网页。

6.3.1 实现（进入）游戏大厅

（1）按 6.2 节及图 6-5 的设计要求，在项目的 src/mygame 下，打开或创建游戏大厅类 GameHome，编写它的实现代码，如下。

GameHome.java:

```
package mygame;
import java.sql.*;
import java.util.*;
public class GameHome
{
    //   大厅内有多个游戏室
    public ArrayList<GameRoom> rooms= new ArrayList<GameRoom>();
    public GameHome() throws Exception
    {//实例化时，进行初始化，设置房间
        init_rooms();
    }
    public void init_rooms() throws Exception
    {
        rooms.clear();
        //初始化，从数据库中读取预设的游戏室信息
        Connection conn=dbConnect.getconntion();//连接到数据库
        try{
           Statement stmt=conn.createStatement();
           String sql="select * from roominfo";
           ResultSet rs = stmt.executeQuery(sql);
           while(rs.next())
           {  //每取一条记录信息，生成一个游戏室对象，加入到集合中
              GameRoom room = new GameRoom();
              room.setRoomname(rs.getString("RoomName"));
              room.setRoomtype(rs.getString("RoomType"));
              room.setManageby(rs.getString("ManageBy"));
              room.setRoomdesc(rs.getString("RoomDesc"));
              rooms.add(room);
           }
        }
        catch(Exception ex)
        {
           throw new Exception("从数据库表中读取游戏室信息时出错!");
        }
        finally{
           conn.close();//保证数据库连接的关闭
        }
    }
```

在 GameHome 的构造方法中调用 init_rooms()，实现在实例化游戏大厅时，从数据库载入各游戏房间的信息。

（2）设计游戏大厅页 gamehome.jsp：使用<jsp:useBean>指令，在 application 作用范围内设置游戏大厅实例，从游戏大厅实例的属性 rooms 取到所有的游戏房间信息，列表显示在网页上，源码如下。

gamehome.jsp：

```
<%@ page language="java" import="mygame.*" %>
<%@ page contentType="text/html;charset=GB2312" %>
<%@ include file="loginCheck.jsp" %>
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application"/>
<html>
  <head>
    <title>游戏大厅</title>
     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="cache-control" content="no-cache">
     <meta http-equiv="expires" content="0">
  </head>
  <body topmargin=0>
  <center>
  <img src="images/home.jpg" border="0" /><br>
  <div><font color="red"><%=currUser%></font>，欢迎来到 99 游戏大厅 
  [<A href="userInfo.jsp?cmd=load" target="_blank"><font color=red>个人信息
   </font></A>]  
  [<A href="userScore.jsp"><font color=red>历史战况</font></A>]</div>
  <table bgcolor="#ffffbb" width="520" ><tr><td>
<%
  for(int i=0;i<gamehome.rooms.size();i++)
  {
     GameRoom room = gamehome.rooms.get(i);
%>
    <A href="gameroom.jsp?no=<%=i%>">
       <img src="images/room<%=i%>.gif" border="0" width="150" height="50"/>
       <font size=2><%=room.getRoomname()%>: <%=room.getRoomdesc()%></font>
       </A><hr>
<%} %>
  </td></tr></table>
  <input type=button name=bn_newroom value="设置新游戏房间" onclick=
"new_room()">   
  <input type=button name=bn_exit value="离开大厅,退出" onclick="location=
      'logout.jsp'">
  </center>
  </body>
<script type="text/javascript">
<!--
function new_room()
```

```
{  //弹出房间信息页
   window.showModalDialog("gameroominfo.jsp",0,"dialogWidth=500px;
dialogHeight=350px;center=yes");
   window.location.reload();//刷新本页
}
//-→
</script>
</html>
```

在游戏大厅页面上，单击游戏房间链接，可以进入到相应的游戏房间页，单击“离开大厅, 退出”，退出登录，回到首页。

（3）设计游戏登录退出页 logout.jsp：通过作废当前 Session，撤销了原保存在 Session 中的登录信息，重定向到首页，源码如下。

logout.jsp：

```
<%@ page language="java"  pageEncoding="GBK"%>
<%
   session.invalidate();
   response.sendRedirect("99main.jsp");
%>
```

6.3.2 实现（进入）游戏房间

（1）按 6.2 节及图 6-5 的设计要求，在项目的 src/mygame 下，打开或创建游戏房间类 GameRoom，编写游戏房间类 GameRoom 的实现代码，如下。

GameRoom.java：

```
package mygame;
import java.sql.*;
public class GameRoom {
    String roomname=""; //房间名
    String roomtype=""; //类型
    String manageby=""; //管理者
    String roomdesc=""; //描述
    //每个房间预设 20 张游戏桌
    GameDesk[] desks= new GameDesk[20];
    public GameRoom()
    {   //初始化，生成房间的所有游戏桌对象
        for(int i=0;i<desks.length;i++)
            desks[i]=new GameDesk();
    }
    public void loadInfoFromDB(String rmname) throws Exception
    { //从数据库表检索到此游戏房间的信息
      Connection conn=dbConnect.getconntion();//连接到数据库
      try{
          String sql="select * from roominfo where RoomName='"+rmname+"'";
          Statement stmt=conn.createStatement();
```

```
            ResultSet rs = stmt.executeQuery(sql);
            if (rs.next())
            {
                roomname = rmname;
                roomtype=rs.getString("roomtype");
                manageby=rs.getString("manageby");
                roomdesc=rs.getString("roomdesc");
            }
        }
        catch(Exception ex)
        {
            throw new Exception("从数据库表检索信息时出错!");
        }
        finally{
            conn.close();//保证数据库连接的关闭
        }
    }
    public String saveInfoFromDB() throws Exception
    {
        if (roomname.length()==0)//房间名不能为空
            return "房间名不能为空";
        //保存信息到数据库表中:删掉旧信息，插入新信息
        Connection conn=dbConnect.getconntion();//连接到数据库
        conn.setAutoCommit(false);//设计事务为非自动提交
        String rtn_str;
        try{
            String sql="delete from roominfo where RoomName='"+roomname+"'";
            Statement stmt=conn.createStatement();
            stmt.executeUpdate(sql);
            stmt.close();
           //参数化 SQL
            sql = "insert into roominfo(RoomName,RoomType,ManageBy,RoomDesc)
values(?,?,?,?)";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, roomname);//设置参数值
            pstmt.setString(2, roomtype);
            pstmt.setString(3, manageby);
            pstmt.setString(4, roomdesc);
            pstmt.executeUpdate();

            conn.commit();//提交生效
            rtn_str="成功保存信息到数据库表!";
        }
        catch(Exception ex)
        {
            conn.rollback();//回滚恢复
```

```
            rtn_str="保存信息到数据库表时出错!";
        }
        finally{
            conn.close();//保证数据库连接的关闭
        }
        return rtn_str;
    }
}
```

（2）添加 GameRoom 类的属性 get/set 方法：右击项目中的 GameRoom 类，选择弹出菜单中的 Source→Generate Getters and Setters…，为所需的属性设置 get/set 方法，如图 6-8 所示。

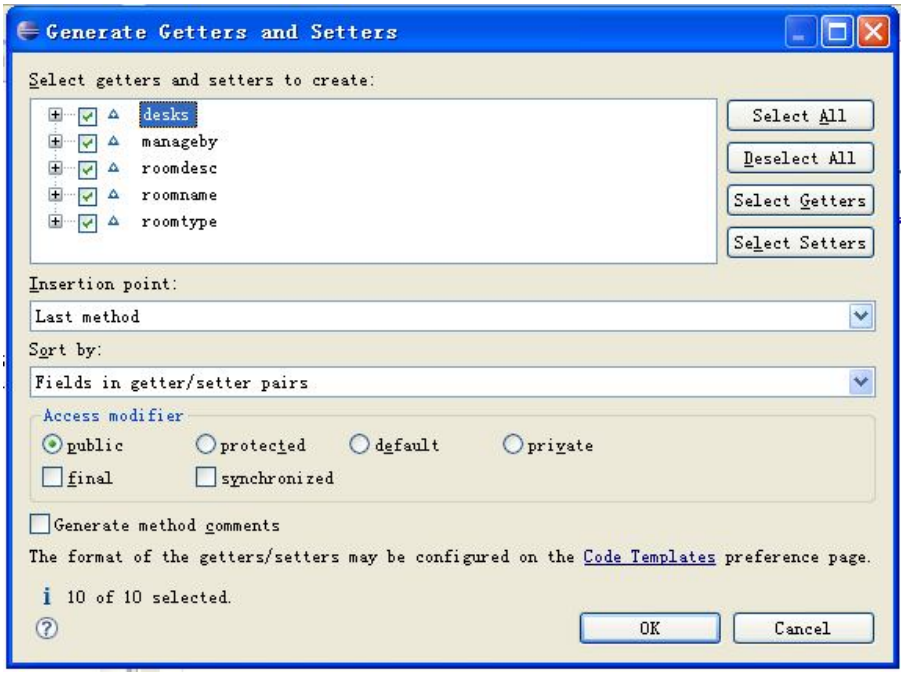

图 6-8　设置属性的 get/set 方法

在类 GameRoom 中自动生成的代码：

```
public String getRoomname() {
    return roomname;
}
public void setRoomname(String roomname) {
    this.roomname = roomname;
}
public String getRoomtype() {
    return roomtype;
}
public void setRoomtype(String roomtype) {
    this.roomtype = roomtype;
}
public String getManageby() {
```

```
    return manageby;
}
public void setManageby(String manageby) {
    this.manageby = manageby;
}
public String getRoomdesc() {
    return roomdesc;
}
public void setRoomdesc(String roomdesc) {
    this.roomdesc = roomdesc;
}
public GameDesk[] getDesks() {
    return desks;
}
public void setDesks(GameDesk[] desks) {
    this.desks = desks;
}
```

很多情况下，需要手工为属性设置 get/set 方法，并按需求编写其中的代码。

（3）为了查看或重设游戏房间信息，如房间名、等级说明、简介等，设计一个游戏房间信息页 gameroominfo.jsp，页中使用<jsp:useBean>指令，在 page 作用范围内设置 GameRoom 类的实例。源码如下：

gameroominfo.jsp：

```
<%@ page language="java" import="mygame.*" pageencoding="gbk"%>
<%@ include file="logincheck.jsp" %>
<% request.setcharacterencoding("gbk"); %>
<jsp:usebean id="roominfo" class="mygame.gameroom" scope="page">
  <jsp:setproperty name="roominfo" property="*" />
</jsp:usebean>
<jsp:usebean id="gamehome" class="mygame.gamehome" scope="application" />
 <%
   if (request.getparameter("load")!=null)
   {//通过 url 串的 load 参数传来游戏房间名，表示要从数据库读取对应房间的信息
      roominfo.loadinfofromdb(request.getparameter("load"));
   }
   else if (request.getparameter("save")!=null)
   {   //用户单击页面上的“保存”按钮控件，提交表单，保存信息到数据库
      string rtn_str = roominfo.saveinfofromdb();
      gamehome.init_rooms();//刷新游戏厅内的房间信息
 %>
      <script type="text/javascript">
        window.alert("<%=rtn_str%>");
      </script>
<%
   }
%>
```

```
<html>
  <head>
   <title>游戏房间信息</title>
     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="expires" content="0">
     <base target="_self">
  </head>
  <body>
 <form name="info" method="post" action="gameroominfo.jsp">
     <table align="center"><tr><td><font size=2>
     <h1><img src="images/roominfo.gif">游戏房间的信息</h1>
    房间名称：<input type=text size=28 name=roomname value="<%=roominfo.
getroomname()%>"><br>
    游戏等级：<input type=text size=28 name=roomtype value="<%=roominfo.
getroomtype()%>"><br>
    管理人员：<input type=text size=28 name=manageby value="<%=roominfo.
getmanageby()%>"><br>
    房间简介:<textarea rows=3 cols=50 name=roomdesc><%=roominfo.getroomdesc()%>
</textarea><br>

            
       
    <input type=submit name=save value=保存>          
   
    <input type=button name=exit value=退出 onclick="window.close()"> </font>
</td></tr></table>
 </form>
 </body>
</html>
```

页中，<% request.setCharacterEncoding("GBK"); %>用来将 request 中表单参数值的中文编码转为 GBK。

<jsp:setProperty name="roominfo" property="*" />，通过自动匹配 request 中参数名与 JavaBean 中的属性名，将 request 中参数值设置给 JavaBean 实例的同名属性（注意：request 中的参数名与 JavaBean 的 setXXX 方法名的 XXX 部分比较，request 中的参数名全部用小写字母，setXXX 方法名中的第一个 X 位置要大写，否则匹配不成功）。

通过判断名为 load 或 save 的 request 参数是否存在，来决定是从数据库装载信息，还是把信息保存到数据库。初始访问 gameroominfo.jsp 时，在 URL 串中设置“load=房间名”参数，从数据库装载信息，并显示在网页上。单击网页上名为“save”的按钮，提交表单中信息，把信息保存到数据库。

游戏房间的信息被修改后，需要刷新游戏大厅中的游戏房间列表中的信息，因此，用下面的<jsp:useBean>引用全局的游戏大厅对象，并在保存数据后重新初始化它。

```
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application">
```

（4）设计游戏房间页 gameroom.jsp：使用<jsp:useBean>指令，引用 application 作用范围内的游戏大厅实例，从游戏大厅实例的属性 rooms 中按编号取到当前游戏房间对象，显示房间

信息及各游戏桌在网页上，如图 6-9 所示。

图 6-9　游戏房间页

源码如下：

gameroom.jsp：

```
<%@ page language="java" import="mygame.*" %>
<%@ page contentType="text/html;charset=GB2312" %>
<%@ include file="loginCheck.jsp" %>
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application" />
<%
    //从游戏实例对象中，按编号取到当前游戏房间对象
    int no= Integer.parseInt(request.getParameter("no"));
    GameRoom room = gamehome.rooms.get(no);
 %>
<html>
<head>
    <title>游戏室</title>
     <meta http-equiv="pragma" content="no-cache">
     <meta http-equiv="cache-control" content="no-cache">
     <meta http-equiv="expires" content="0">
     <meta http-equiv="refresh" content="15" />
<script type="text/javascript">
<!--
function show_info()
{  //弹出房间信息页
   window.showModalDialog("gameroominfo.jsp?load=<%=room.getRoomname()%>",0,
```

```
"dialogWidth=500px;dialogHeight=350px;center=yes");
   window.location.reload();//刷新本页
}
//-→
</script>
</head>
<body topmargin=0 bgcolor="555555">
    <p align="center">
     <font size=3 color=red><%=room.getRoomname()%>: <%=room.getRoomdesc()%>
</font>
     <input type="button" name="load" value="房间信息" onclick="show_info()">
     <input  type="button"  name="exit"  value=" 退 出 "  onclick="location=
'gamehome.jsp'">
     <hr  color=red>
  <center><font size=2>
<table border="0" cellpadding="0" cellspacing="1" >
  <%
  GameDesk[] desk = room.getDesks();
  for(int i=0;i<desk.length;i++)
  {
  %>
    <% if (i%2==0){%><TR><%} %> <TD>
     <table bgcolor="#ffffee">
       <tr>
         <td width="100"></td>
         <td width="100" align="center">
         <%if (desk[i].siter[0]==null){//第一座位 %>
             <img src="images/chair.jpg" width="70" height="70" title="单击加
入" onclick="location='entergame.jsp?room=<%=no%>&desk=<%=i%>&site=0'">
         <%} else { %>
             <img  src="pic/<%=desk[i].siter[0].FacePic%>.gif"    width="70"
height="50" border=0><br>
             <font     size=2     color=green><%=desk[i].siter[0].UserName%>
[<%=desk[i].siter[0].UserID%>]</font>
         <%} %>
         </td>
         <td width="100"></td>
       </tr>
       <tr>
         <td width="100" align="center">
         <%if (desk[i].siter[3]==null){//第四座位 %>
             <img src="images/chair.jpg" width="70" height="70" title="单击加
入" onclick="location='entergame.jsp?room=<%=no%>&desk=<%=i%>&site=3'">
         <%} else { %>
             <img  src="pic/<%=desk[i].siter[3].FacePic%>.gif"    width="70"
height="50" border=0><br>
```

```
            <font     size=2     color=green><%=desk[i].siter[3].UserName%>
[<%=desk[i].siter[3].UserID%>]</font>
        <%} %>
        </td>
        <td width="100" align="center" background="images/desk.jpg">
           <Ahref="gamedesk.jsp?room=<%=no%>&desk=<%=i%>&observe=1"title="
单击进入观战"><font size=8 color=red><%=i%>号</font></A></td>
        <td width="100" align="center">
        <%if (desk[i].siter[1]==null){//第二座位 %>
            <img src="images/chair.jpg" width="70" height="70" title="单击加
入" onclick="location='entergame.jsp?room=<%=no%>&desk=<%=i%>&site=1'">
        <%} else { %>
            <img   src="pic/<%=desk[i].siter[1].FacePic%>.gif"   width="70"
height="50" border=0><br>
            <font     size=2     color=green><%=desk[i].siter[1].UserName%>
[<%=desk[i].siter[1].UserID%>]</font>
        <%} %>
        </td>
      </tr>
      <tr>
        <td width="100"><FONT face="宋体"></FONT></td>
       <td width="100" align="center">
        <%if (desk[i].siter[2]==null){//第三座位 %>
            <img src="images/chair.jpg" width="70" height="70" title="单击加
入" onclick="location='entergame.jsp?room=<%=no%>&desk=<%=i%>&site=2'">
        <%} else { %>
            <img  src="pic/<%=desk[i].siter[2].FacePic%>.gif"    width="70"
height="50" border=0><br>
            <font     size=2     color=green><%=desk[i].siter[2].UserName%>
[<%=desk[i].siter[2].UserID%>]</font>
        <%} %>
        </td>
        <td width="100"></td>
      </tr>
    </table>
    <TD><% if (i%2==0){%></TR><%} %>
 <%
  }
 %>
   </table>
  </body>
</html>
```

其中，<meta http-equiv="refresh" content="15" />设置了网页的自动刷新时间，使网页能够反映游戏房间的最近情况。

游戏的座位空时，显示一个椅子：<img src="images/chair.jpg" width="70" height="70" title="单击加入"onclick="location='entergame.jsp?room=<%=no%>&desk=<%=i%>&site=2'">，单击椅

子，转向 entergame.jsp 页，在 entergame.jsp 页中实现“在某房某桌某位加入游戏”的功能。

任意单击某桌游戏的桌面分，可以转入游戏（牌桌）网页，观战此桌游戏。

```
<A href="gamedesk.jsp?room=<%=no%>&desk=<%=i%>&observe=1" title="单击进入观战
"><font size=8 color=red><%=i%>号</font></A>
```

6.3.3 实现（坐上）玩家位置

把玩家的基本信息及行为封装为类 GamePlayer，用整个 Session（会话）作用范围内的一个 GamePlayer 实例（theplayer），来表示当前用户玩家。给其重要属性 desk 赋值：

```
theplayer.desk = desks[i];      //表示玩家在第 i 桌游戏
```

如果 theplayer.desk==null，表示玩家没有坐在游戏桌上，处于空闲。

（1）按 6.2 节及图 6-5 的设计要求，在项目的 src/mygame 下，打开或创建游戏玩家类 GamePlayer，编写游戏玩家类 GamePlayer 的实现代码。

GamePlayer.java：

```
package mygame;
import java.sql.*;
public class GamePlayer {
    public String UserID;//用户 ID
    public String UserName;//用户姓名
    public int FacePic;//头像
    public GameDesk desk=null;//当前游戏所在桌对象，为 null 时表示没有游戏
    public int roomno,deskno,siteno;//用于记忆玩家所在的房号、桌号、座号
    public int[] cards = new int[10];//手上的牌

    public GamePlayer() throws Exception
    {//实例化构造方法

    }
    public void setUserid(String userID) throws Exception
    {
        UserID = userID;
        //设置 ID 时，从数据库载入其信息
        Connection conn=dbConnect.getconntion();//连接到数据库
        Statement stmt=conn.createStatement();
        String sql="select * from userinfo where userid='"+userID+"'";
        ResultSet rs = stmt.executeQuery(sql);
        if(rs.next())
        {
           UserID=rs.getString("UserID");
           UserName=rs.getString("UserName");
           FacePic=rs.getInt("FacePic");
        }
        conn.close();//保证数据库连接的关闭
    }
```

```
    //按游戏规则出牌、算分，是游戏规则的实现方法
    public synchronized void playout(int cardno) throws Exception
    {//玩家打出第 cardno 张牌
       if (!desk.isplaying || cards[cardno]==100)//游戏结束或此张牌已出，不能再出
           return;
       int siteno = -1;//当前玩家座位，-1 表示没有坐在此桌
       for(int i=0;i<4;i++)//找到本玩家座位
         if (desk.siter[i]==this) siteno=i;
       if (desk.currentplay!=siteno)//如果现在不是轮到本玩家位置出牌
         return;
       int cardnum = cards[cardno];
       desk.playonecard(cardnum);//按规则改变游戏状态
       cards[cardno] = 100;//表示此张牌已出
    }

    //加入某房某桌游戏
    public void enterGame(GameDesk thedesk, int siteno) throws Exception
    {
        thedesk.siter[siteno]=this;//玩家坐上游戏桌的第 siteno 座位
        this.desk=thedesk; //设置玩家已在某个游戏桌的游戏中
        thedesk.playercount++;//玩家人数加 1
        thedesk.check_GameIsStart();//检查玩家是否足 4 人，够 4 人就发牌，开始游戏
    }

    //强行退出游戏
    public void exitGame() throws Exception
    {
        if (desk!=null)
        {
           int siteno = -1;//当前玩家座位，-1 表示没有坐在此桌
           for(int i=0;i<4;i++)//找到本玩家座位
             if (desk.siter[i]==this) siteno=i;

           if (siteno>=0)//本玩家在此桌上有座位
           {
           if (desk.isplaying)
              desk.kickout_theGame(siteno);//当前玩家出局，但游戏还没结束，还需计分
           else{
              desk.siter[siteno]=null;//完全离开
                 desk.playercount--;
           }
              desk=null;//退出游戏（桌），没有坐在游戏桌前
           }
        }
    }
}
```

（2）用户登录进入游戏系统，就成为了一名游戏玩家，因此，我们可以在登录验证通过后进入游戏大厅时，实例化 GamePlayer，生成游戏玩家对象，并设置其属性。在游戏大厅页 gamehome.jsp 开始部分，增加下面代码：

```
<jsp:useBean id="theplayer" class="mygame.GamePlayer" scope="session"/>
<jsp:setProperty name="theplayer" property="userid" value="<%=currUser%>"/>
```

<jsp:setProperty>的 name 要与<jsp:useBean>的 id 匹配，即上面是给实例对象 theplayer 的属性 userid 设置值，使其等于当前登录用户 ID。

使用<jsp:setProperty>设置 userid 属性值时，会调用 GamePlayer 类的 setUserid(..)方法，在方法 setUserid(..)中，连接到数据库，从用户信息表中检索出玩家的信息（参考 GamePlayer 类的代码）。

<jsp:setProperty name="theplayer" property="userid" …>中的 name 值要与<jsp:useBean>中的 id 相同，property 值是小写的 userid，对应 GamePlayer 类中的方法 setUserid(..)，方法名称中的 Userid 第一个字母大写。

（3）设计游戏加入页 entergame.jsp：在游戏房间页 gameroom.jsp 中，当用户单击游戏桌的空座位时，把房号、桌号、座位号传给 entergame.jsp 页，执行在此座位加入游戏的功能。entergame.jsp 的源码如下：

entergame.jsp：

```
<%@ page language="java" import="mygame.*" %>
<%@ page contentType="text/html;charset=GB2312" %>
<%@ include file="loginCheck.jsp" %>
<jsp:useBean id="theplayer" class="mygame.GamePlayer" scope="session"/>
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application" />
<html>
<head>
  <title>加入游戏</title>
  <meta http-equiv="Content-Type" content="text/html; charset=GB2312">
</head>
<body>
<%
    int roomno = Integer.parseInt(request.getParameter("room"));//房号
    int deskno = Integer.parseInt(request.getParameter("desk"));//桌号
    int siteno = Integer.parseInt(request.getParameter("site"));//座号
    if (theplayer.desk == null)
    {//玩家处于空闲状态，还没有坐下加入任何游戏
       GameDesk[] desks = gamehome.rooms.get(roomno).getDesks();
       if (desks[deskno].siter[siteno]==null)
       {//第 deskno 桌第 siteno 座位还空着，坐下加入
          theplayer.enterGame(desks[deskno],siteno);//坐上第 deskno 桌的第
                                                    siteno 位
          theplayer.roomno=roomno;//记忆房号
          theplayer.deskno=deskno;//记忆桌号
          theplayer.siteno=siteno;//记忆座号
```

```
            //转到游戏(桌)页

response.sendRedirect("gamedesk.jsp?room="+roomno+"&desk="+deskno);
      }
      else
      {//第 deskno 桌第 siteno 座位已被占，不可以坐下加入
%>
      <script type="text/javascript">
         window.alert("第<%=deskno%>桌第<%=siteno%>座位已被抢占！");
         window.location="gameroom.jsp?no=<%=roomno%>";//返回游戏房间页
      </script>
<%
      }
    }
   else
   {//玩家早已坐下加入某桌游戏，处于游戏中，不能再加入其他桌
  %>
      <script type="text/javascript">
         window.alert("你已在游戏中，不能再加入其他游戏！");
         window.location="gamedesk.jsp?room=<%=theplayer.roomno
%>&desk=<%=theplayer.deskno%>";//转到游戏桌页
      </script>
<%
   }
 %>
</body>
</html>
```

使用<jsp:useBean>指令，引用在 Session 作用范围内的游戏玩家对象，代表现实中的玩家用户概念。

程序逻辑：如果 theplayer.desk==**null**，表示玩家没有加入任何桌，从 request 参数中取到房号、桌号、座位号，如果 desks[deskno].siter[siteno]==**null**，表示座位还空着，可以坐下加入游戏，程序调用：

```
theplayer.enterGame(desks[deskno],siteno);//坐上第 deskno 桌的第 siteno 位
```

（4）设计退出游戏页 exitgame.jsp：在游戏（牌桌）页 gamedesk.jsp 上，当用户单击“强退”时，转到退出游戏页 exitgame.jsp，页中通过调用 theplayer.exitGame()，实现“玩家退出进行中的游戏（桌）”功能。退出后，theplayer.desk 的值变为 null。

exitgame.jsp：

```
<%@ page language="java" pageEncoding="GBK"%>
<jsp:useBean id="theplayer" class="mygame.GamePlayer" scope="session"/>
<%
   //退出游戏（桌），返回游戏大厅页
   theplayer.exitGame();
 %>
<script type="text/javascript">
<!--
```

```
    alert("你放弃了本桌游戏，强行退出了");
    location ="gamehome.jsp";
//-→
</script>
```

6.3.4 实现（坐在）游戏牌桌

把游戏的状态信息及行为封装为游戏（牌桌）类 GameDesk，每一个游戏牌桌的作用范围都是 application 范围。当把玩家对象实例的引用赋给游戏桌的某位置变量时，表示游戏桌的相应座位被玩家占着。例如，加入游戏时设置：

```
desks[i].siter[0] = theplayer; //第 i 桌的第 0 座位被玩家占着
```

在游戏结束时，初始化游戏桌，恢复设置游戏桌的各座位为空。

（1）按 6.2 节及图 6-5 的设计要求，在项目的 src/mygame 下，打开或创建游戏（牌桌）类 GameDesk，编写游戏（牌桌）类 GameDesk 的实现代码。

GameDesk.jave：

```
package mygame;
import java.sql.*;
import java.util.*;
public class GameDesk {
    public GamePlayer[] siter = new GamePlayer[4];//4 个位置的玩家
    public int[] sitestate = new int[4];//4 个位置的状态
    public int playercount;//玩家人数
    public int currentplay;//轮到的当前出牌者的位置号
    public java.util.Date playouttime;//出牌超时计时开始时间
    public int playorder;//出牌顺序:1--顺时针，-1--逆时针
    public int deskscore;//桌面牌分
    public boolean isplaying;//是否打牌状态

    public GameDesk()
    {
        init_desk();//初始化游戏桌
    }
    //初始化游戏桌
    public void init_desk()
    {
        playercount=0;//玩家人数
        currentplay = 0;//轮到的当前出牌者的位置号
        playorder = 1;//出牌顺序:1--顺时针，-1--逆时针
        deskscore =0;//桌面牌分
        isplaying = false;//是否打牌状态
        for(int i=0;i<4;i++){
            siter[i]=null;
            sitestate[i]=0;//位置状态:0-游戏中,1,2,3,4 表示第几位结束出牌(出局)
        }
```

```
}

//实现出牌规则的方法：出某种牌，游戏状态如何变化
public void playonecard(int cardnum)  throws Exception
{
   int siteno = currentplay;
   switch(cardnum) {
   case 4:
       deskscore -=10;//让中央总分减少 10
       currentplay = (4+currentplay+playorder)%4;//轮到下家出牌
       break;
   case 5:
       deskscore -=20;//让中央总分减少 20
       currentplay = (4+currentplay+playorder)%4;//轮到下家出牌
       break;
   case 10:
       playorder *=-1;//反转出牌顺序
       currentplay = (4+currentplay+playorder)%4;//轮到下家出牌
       break;
   case 11:
       currentplay = (4+currentplay+2)%4;//指定对家出牌
       break;
   case 12:
       deskscore =99;//总分立刻变为 99
       currentplay = (4+currentplay+playorder)%4;//轮到下家出牌
       break;
  default: //0,1,2,3,6,7,8,9
   deskscore += cardnum;
      currentplay = (4+currentplay+playorder)%4;//轮到下家出牌
      if (deskscore>99)
      {   //当前出牌玩家出局
       kickout_theGame(siteno);
       deskscore -= cardnum;//恢复桌分
      }
      break;
  }
   //如果轮到的位置玩家已出局，顺延到下一家
   while(sitestate[currentplay]>0)
   {
       currentplay = (4+currentplay+playorder)%4;//轮到下家出牌
   }
   playouttime = new java.util.Date();//下一玩家出牌计时开始时间
}

//检查玩家是否足 4 人，够 4 人就发牌，开始游戏
public boolean check_GameIsStart()
```

```
    {
        if (isplaying==false && playercount==4)
        {//够 4 人就发牌，开始游戏
            Random rand = new Random();
            for(int i=0;i<4;i++)//4 个玩家
                for(int j=0;j<10;j++)//每玩家 10 张牌
                    siter[i].cards[j]= rand.nextInt(12);//随机生成 0～12 整数
            isplaying = true;//游戏开始
        }
        currentplay = 0;//0 位置开始出牌
        playouttime = new java.util.Date();//下一玩家出牌计时开始时间

        return isplaying;
    }

    //检查游戏是否结束
    public boolean check_GameIsOver() throws Exception
    {
        if (isplaying==true && playercount==1)//还剩一个玩家
        {//游戏结束，记分
            Connection conn=dbConnect.getconntion();//连接到数据库
           PreparedStatement  pstmt=conn.prepareStatement("update  userinfo
set PlayNum=PlayNum+1,WinNum=WinNum+?,GameScore=GameScore+? where userid=?");
            for(int i=0;i<4;i++)//4 个玩家
            {
               pstmt.setInt(1, sitestate[i]==4?1:0);//赢次数
               pstmt.setInt(2,100*(sitestate[i]-2));//得分
               pstmt.setString(3,siter[i].UserID);//得分
               pstmt.executeUpdate();
            }
            conn.close();

            //玩家离开此游戏牌桌
            for(int i=0;i<4;i++){
              if (siter[i]!=null && siter[i].desk==this)
                    siter[i].desk=null;
            }
           init_desk();//初始化游戏桌
        }
        return !isplaying;
    }

    //某座位玩家出局
    public void kickout_theGame(int siteno) throws Exception
    {
      if (sitestate[siteno]>0) return;//早已出局
```

```
        //当前玩家出局,设置其座位为第几位出局者
        this.sitestate[siteno]= 5-playercount;//sitestate[i]>0:出局
        this.playercount -=1;//出牌人数减 1
        this.check_GameIsOver();//有人出局，检查游戏是否结束

        if (currentplay==siteno)
        {  //如果正好是轮到的此玩家出牌，但已出局，所以顺延到下一家
           while(this.sitestate[currentplay]>0)
           {
              currentplay = (4+currentplay+playorder)%4;//轮到下家出牌
           }
           playouttime = new java.util.Date();//下一玩家出牌计时开始时间
        }
     }
}
```

- 方法 playonecard(**int** cardnum)，实现游戏规则：

① 数字牌，玩家打出会增加中央总分

+1+2+3+5+6+7+8+9

② 魔法牌

0－跳过此玩家　　10－反转出牌顺序　　11－指定对家出牌

4－让中央总分减 10　　5－让总分减 2 0　　12－总分立刻变为 99

- 方法 kickout_theGame(座号)，实现“某座位玩家出局”的功能，sitestate[座号]>0 表示此位玩家已出局，其值表示是第几位出局，作为游戏得分的计算依据。
- 方法 check_GameIsStart()，检查游戏是否达到开始条件，玩家足 4 人就开始发牌（每玩家随机产生 10 张牌），游戏轮流出牌开始。
- 方法 check_GameIsOver()，每当有人出局，检查游戏是否达到结束条件，未出局玩家少于等于 1 人时，游戏结束，计算各玩家得分，保存到数据库，设置各玩家退出游戏（siter[i].desk = **null**），等待重新加入。

（2）设计并实现游戏（牌桌）页 gamedesk.jsp。游戏（牌桌）页的功能是：定位目标游戏（牌桌）对象，把游戏（牌桌）的状态信息显示在网页上（见图 6-4），并管理着各玩家的轮流出牌。当前出牌玩家单击某张牌，进行出牌操作时，转去执行出牌页程序。游戏（牌桌）页 gamedesk.jsp 的源码如下：

gamedesk.jsp：

```
<%@ page language="java" import="mygame.*" pageEncoding="GBK"%>
<%@ include file="loginCheck.jsp" %>
<jsp:useBean id="theplayer" class="mygame.GamePlayer" scope="session" />
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application" />
<html>
<head>
  <title>游戏（桌）界面</title>
  <meta http-equiv="Content-Type" content="text/html; charset=GB2312">
  <meta http-equiv="cache-control" content="no-cache">
```

```
   <meta http-equiv="expires" content="0">
   <meta http-equiv="refresh" content="5" />
</head>
<%
    int roomno = Integer.parseInt(request.getParameter("room"));//房号
    int deskno = Integer.parseInt(request.getParameter("desk"));//桌号
    GameDesk desk = gamehome.rooms.get(roomno).getDesks()[deskno];
    if (request.getParameter("observe")==null)
    { //非观战，是参战
        if (theplayer.desk==null)
        {//玩家已经退出游戏，在客户端显示“游戏结束”提示信息
%>
          <script type="text/javascript">
             if (confirm("游戏结束! 是否重新开始游戏? "))
                location="entergame.jsp?room=<%=theplayer.roomno  %>&desk=<%
=theplayer.deskno%>&site=<%=theplayer.siteno%>";
             else
                location="gamehome.jsp";
          </script>
<%
          return;
       }
    }
    //判断当前玩家是否在此桌，并找到当前玩家座位
    int siteno = -1;//当前玩家座位，-1 表示没有坐在此桌
    for(int i=0;i<4;i++)
       if (desk.siter[i]==theplayer)
          siteno=i;
 %>
<body bgColor="#000000" text="ffffff" topmargin=0>
    <table align="center">
      <tr height=70>
         <td><img src="images/logo.jpg"></td>
         <td colspan=2><img src="images/logo.gif">
             <input type=button name="bn1" value=返回 onclick="location=
                                    <'gamehome.jsp'"></td>
      </tr>
      <tr height=110>
         <td width="135"></td>
         <td width="135" align="center" <%=desk.sitestate[0]>0?"style=
                              'filter:gray'":""%>>
         <%if (desk.siter[0]==null){//第一座位 %>
             <img src="images/chair.jpg" width="70" height="70">
         <%} else { %>
            <%=siteno==0?"<font color=yellow><H>我</H></font>":"" %>
             <img src="pic/<%=desk.siter[0].FacePic%>.gif" width="70"
height="50" border=0><br>
```

```
            <font size=2 color=white>0:<%=desk.siter[0].UserName%> [<%=desk.
siter[0].UserID%>]</font>
            <br><%=desk.currentplay==0?"<img src='images/flag.gif'>":""%>
        <%} %>
        </td>
        <td width="135"></td>
      </tr>
      <tr height=140>
        <td   Width="135"   align="center"   <%=desk.sitestate[3]>0?"style=
'filter:gray'":""%>>
        <%if (desk.siter[3]==null){//第四座位 %>
            <img src="images/chair.jpg" width="70" height="70">
        <%} else { %>
           <%=siteno==3?"<font color=yellow><H>我</H></font>":"" %>
            <img src="pic/<%=desk.siter[3].FacePic%>.gif" width="70"height=
"50" border=0>
            <%=desk.currentplay==3?"<img src='images/flag.gif'>":"" %><br>
            <font size=2 color=white>3:<%=desk.siter[3].UserName%> [<%=desk.
siter[3].UserID%>]</font>
        <%} %>
        </td>
        <td width="135" align="center" background="images/<%=desk.playorder
==1?"o1":"o2"%>.gif">
<font size=8 color=yellow><%=desk.deskscore%></font></td>
       <td width="135" align="center" <%=desk.sitestate[1]>0?"style= 'filter:
gray'":""%>>
        <%if (desk.siter[1]==null){//第二座位 %>
            <img src="images/chair.jpg" width="70" height="70" >
        <%} else { %>
            <%=desk.currentplay==1?"<img src='images/flag.gif'>":"" %>
            <img src="pic/<%=desk.siter[1].FacePic%>.gif"  width="70" height
="50" border=0>
           <%=siteno==1?"<font color=yellow><H>我</H></font>":"" %><br>
            <font size=2 color=white>1:<%=desk.siter[1].UserName%> [<%=desk.
siter[1].UserID%>]</font>
        <%} %>
        </td>
      </tr>
      <tr height=110>
        <td width="135"><FONT face="宋体"></FONT></td>
       <td   width="135"   align="center"   <%=desk.sitestate[2]>0?"style=
'filter:gray'":""%>>
        <%if (desk.siter[2]==null){//第三座位 %>
            <img src="images/chair.jpg" width="70" height="70">
        <%} else { %>
            <%=desk.currentplay==2?"<img src='images/flag.gif'>":"" %><br>
           <%=siteno==2?"<font color=yellow><H>我</H></font>":"" %>
```

```
            <img src="pic/<%=desk.siter[2].FacePic%>.gif" width="70" height
="50" border=0><br>
            <font size=2 color=white>2:<%=desk.siter[2].UserName%> [<%=desk.
siter[2].UserID%>]</font>
        <%} %>
        </td>
        <td width="135"><br><br></td>
      </tr>
    </table>
<center>
  <table border=0 align="center">
   <tr>
<%
   if (desk.isplaying && siteno>=0)//玩家坐在本桌的第 siteno 位
   {
      for(int j=0;j<10;j++)
      {// 逐张显示玩家的牌
        if (desk.sitestate[siteno]>0)
          out.print("<td width=50 onclick=\"alert('你已出局，不能出牌！')\">");
        if (desk.currentplay==siteno && theplayer.cards[j]!=100)
          //轮到本玩家出牌，增加可出牌的链接
          out.print("<td width=50 onclick=\"playout('"+j+"')\">");
        else
          out.print("<td width=50 onclick=\"alert('还没轮到你出牌!! ')\">");
        out.print("<img src='images/"+theplayer.cards[j]+".gif'></td>");
      }

      if (!desk.isplaying)
        out.print("<td>游戏没开始！等待玩家</td>");
      else if (desk.currentplay==siteno)
        out.print("<td>轮到你出牌了!></td>");
      else if (siteno>=0 && desk.sitestate[siteno]>0)
        out.print("<td>你已经出局了!</td>");
      else
        out.print("<td>轮到第"+desk.currentplay+"位出牌了!</td>");
   }
 %>
 </tr></table>
 <% if (siteno>=0){//玩家坐在本桌的第 siteno 位%>
   <input type=button name=bn2 value="强退" onclick="location='exitgame.jsp'">
 <%} %>
</center>
</body>

<script type="text/javascript">
   var Isout=0;//每次只能出一张牌
   function playout(cardno)
```

```
    {
      if (Isout==0)
        location="playoutcard.jsp?cardno="+cardno;
      Isout =1;
    }
</script>
</HTML>
```

游戏（牌桌）页 gamedesk.jsp 是游戏程序的核心页，是 JSP+JavaBean 技术的典型一页，逻辑功能较复杂，然而，JSP 网页及 JavaBean 分工明确，程序层次清晰。客户端 JavaScript 脚本及 html 标记源码、服务端 JSP 代码、JavaBean 功能类，三者的分工与融合技术，是学习的重点。参看游戏界面（见图 6-4），沿着游戏过程逻辑，阅读网页程序。

- 页面定时刷新：

```
<meta http-equiv="refresh" content="5" />
```

- 定位到游戏（牌桌）对象：

```
desk = gamehome.rooms.get(roomno).getDesks()[deskno];
```

游戏所在的房号（roomno）、桌号（deskno）由 URL 传入。

- 如果不是观战（请求 URL 中没有参数 observe），是参战：

theplayer.desk==null 表示玩家参战的游戏已结束，已设置玩家离位，需要提示玩家“游戏已结束，是否继续加入游戏”。

- 查找玩家座位，判断用户是否坐在此游戏桌上，如果 siteno>=0，表示用户坐在此桌，在网页下方显示用户手上的牌。
- 显示各玩家的状态信息，例如第 0 座位：

```
<%=siteno==0?"<font color=yellow><H>我</H></font>":"" %>
<img  src="pic/<%=desk.siter[0].FacePic%>.gif"  width="70"  height="50"
border=0><br>
<font size=2 color=white>0:<%=desk.siter[0].UserName%> [<%=desk.siter[0].
UserID%>]</font>
<br><%=desk.currentplay==0?"<img src='images/flag.gif'>":""%>
```

第一语句：当 siteno==0 时，显示一个“我”字，表示坐在第 0 座位的玩家是用户自己。

第二、三语句：显示用户的头像、ID、姓名。

第四语句：如 desk.currentplay==0，显示一个标志图像，表示该 0 座位玩家出牌。

- 某玩家已出局时，其所在表格位置（<td>）的头像及文本变灰：

```
<td width="135" align="center" <%=desk.sitestate[3]>0?"style=
'filter:gray'":""%>>
```

- 在中央显示牌分及出牌顺序的箭头图片（o1.gif 或 o2.gif）：

```
〈td width="135" align="center" background="images/<%=desk.playorder
==1?"o1":"o2"%>.gif"〉
<font size=8 color=yellow><%=desk.deskscore%></font></td>
```

- 单击某张牌出牌时，转到出牌页：location="playoutcard.jsp?cardno="+cardno;。
- 单击“强退”按钮，转到退出游戏页：onclick="location='exitgame.jsp'"。

（3）实现玩家出牌页 playoutcard.jsp：玩家出牌页调用玩家对象的 playout(cardno)，在玩家对象的 playout(cardno)方法中，检查此次出牌的合法性，再调用游戏（牌桌）对象的方法

playout(int cardno)，实现按游戏规则出牌功能。

playoutcard.jsp：

```
<%@ page language="java" import="mygame.*" pageEncoding="GBK"%>
<jsp:useBean id="theplayer" class="mygame.GamePlayer" scope="session"/>
<%
   //出牌
   if (request.getParameter("cardno")!=null)
   {
      int cardno= Integer.parseInt(request.getParameter("cardno"));//第几张牌
      theplayer.playout(cardno);//出第 cardno 张牌
   }
   response.sendRedirect("gamedesk.jsp?room="+theplayer.roomno+"&desk="
+theplayer.deskno);
%>
```

6.3.5 实现（后台）定时维护

到此，游戏的用户操作及游戏处理逻辑已全部实现，但是，游戏系统还存在一个严重的缺陷：当某个游戏中的玩家网络断线、系统崩溃或恶意搁置时，其所在游戏将被定格在此玩家的出牌状态，进入垃圾等待时间。因此，必须对玩家“出牌拖延”设置超时机制，定时检查，在后台维护游戏。

（1）游戏的维护服务程序是以独立的线程在服务器端运行的，为此，我们把维护服务的逻辑封装为一个具有独立线程运行接口（Runnable）的类，代码如下：

GameService.jave：

```
package mygame;
import java.util.Date;
public class GameService implements Runnable {
    public boolean IsServiceStarted = false;//后台服务是否启动
    protected GameHome gamehome;//服务的目标——某游戏大厅
    static long delayinterval = 30;//出牌拖延超时时长(秒)
    static long checkinterval = 2;//后台检查时间间隔时长(秒)

    public GameService() throws Exception
    {//实例化时，进行初始化，设置房间
    }
public void run() {
        try{
          while(IsServiceStarted)
          {   //定时执行服务程序
              Thread.sleep(1000*checkinterval);
              do_service();
          }
        }
        catch(Exception ex){}
```

```
        IsServiceStarted=false;
    }

    //在后台服务工作内容:遍历检查所有游戏桌，检查是否有出牌超时等
    protected void do_service() throws Exception
    {
        GameRoom theroom;
        GameDesk thedesk;
        long delay;
        java.util.Date now = new Date();
        for(int i=0;i<gamehome.rooms.size();i++)
        {
            theroom = gamehome.rooms.get(i);
            for(int j=0;j<theroom.desks.length;j++)
            {
                thedesk = theroom.desks[j];
                if (thedesk.isplaying)//此桌游戏进行中
                {
                    //出牌计时数(秒)

    delay=(now.getTime()-thedesk.playouttime.getTime())/1000;
                    if (delay>delayinterval)//出牌拖延超时
                    {//当前出牌玩家出局
                       thedesk.kickout_theGame(thedesk.currentplay);
                         }
                }
            }
        }
    }

    //启动后台服务线程
    public void start_service(GameHome thegamehome) throws Exception
    {
        gamehome = thegamehome;
        if (!IsServiceStarted)
        {
            IsServiceStarted=true;
            new Thread(this).start();//开始线程,运行 run()
        }
    }

    //停止后台服务线程
    public void stop_service() throws Exception
    {
        IsServiceStarted = false;
    }
}
```

（2）启动游戏的维护服务程序：启动服务端后台程序的方法有多种，这里采用的方法是网页启动方式。在游戏大厅页 gamehome.jsp 初次被访问时，实例化 application 范围内的游戏大厅对象，实现游戏的整个对象模型，然后，启动游戏的维护服务程序。

游戏大厅页 gamehome.jsp 中的代码：

```
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application" />
```

改为：

```
<jsp:useBean id="gamehome" class="mygame.GameHome" scope="application" />
<jsp:useBean id="gameservice" class="mygame.GameService" scope="application">
<%//启动后台服务进程
      gameservice.start_service(gamehome);
%>
</jsp:useBean>
```

6.3.6 试运行、测试游戏

从 Windows 任务栏中依次打开 4 个独立的 IE 窗口，分别从 4 个 IE 窗口，访问游戏网站，登录游戏。4 个 IE 窗口在 Web 服务器上形成 4 个 Session，其上的登录用户是 4 个玩家，因此，可以在各 IE 窗口中操作坐到同一游戏桌，进行游戏。这样，我们就可以按功能点测试游戏，进一步设置断点，调试程序。

6.4 资料：JavaBean 知识与技术

JSP 作为一个很好的动态网站开发语言得到了越来越广泛的应用，在各类 JSP 应用程序中，JSP+JavaBean 的组合成为了一种事实上最常见的 JSP 程序的标准，就让我们来看看具体的 JSP 是如何与 JavaBean 结合在一起的吧。

6.4.1 JavaBean 简介

JavaBean 是描述 Java 的软件组件模型，有点类似于 Microsoft 的 COM 组件概念。在 Java 模型中，通过 JavaBean 可以无限扩充 Java 程序的功能，最重要的是 JavaBean 可以实现代码的重复利用，通过 JavaBean 的组合可以快速地生成新的应用程序，另外，对于程序的易维护性等也有很重大的意义。

JavaBean 传统的应用在于可视化的领域，如 AWT 下的应用。自从 JSP 诞生后，JavaBean 更多地应用在了非可视化领域，在服务器端应用方面表现出强大的生命力。

在 JSP 应用程序开发中，JavaBean 常用来封装事务逻辑、数据库操作等，可以很好地实现业务逻辑和前台程序（如 jsp 文件）的分离，使得系统具有更好的健壮性和灵活性。

一个简单的例子，比如说一个购物车程序，要实现购物车中添加一件商品这样的功能，就可以写一个购物车操作的 JavaBean，建立一个 public 的 AddItem 成员方法，前台 JSP 文件

里面直接调用这个方法来实现。如果后来又考虑添加商品的时候需要判断库存是否有货物，没有货物不得购买，在这个时候我们就可以直接修改 JavaBean 的 AddItem 方法，加入处理语句来实现，这样就完全不用修改前台 JSP 程序了。

当然，也可以把这些处理操作完全写在 JSP 程序中，不过这样的 JSP 页面可能就有成百上千行，光看代码就是一个头疼的事情，更不用说修改了。通过 JavaBean 可以很好地实现逻辑的封装、程序的易于维护等。使用 JSP 开发程序，一个很好的习惯就是多使用 JavaBean。

在创建非可视化 JavaBean 中，常用 get/set 这样的成员方法来处理属性（properties）。

下面是一个简单的 JavaBean。

MyJavaBean.java

```
import java.io.*;
public class MyJavaBean
{
private String FirstProperty = new String("");
public FirstJavaBean()
{
}
public String getFirstProperty() {
   return FirstProperty;
}
public void setFirstProperty(String value){
   FirstProperty = value;
}
public static void main(String[] args){
   System.out.println("My First JavaBean!");
}
}
```

这是一个很典型的 JavaBean 的代表，FirstProperty 是其中的一个属性（Property），外部通过 get/set 方法可以对这个属性进行操作。main 方法是为了测试程序用的，写 JavaBean 可以先不必加入到 JSP 程序中调用，而直接用 main 方法来进行调试，调试好以后就可以在 JSP 程序中调用了。

6.4.2 JavaBean 相关标签

在 JSP 中调用 JavaBean 有三个标准的标签，那就是<jsp:useBean>、<jsp:setProperty>以及<jsp:getProperty>。

- <jsp:useBean>标签

<jsp:useBean>可以定义一个具有一定生存范围以及一个唯一 id 的 JavaBean 的实例，这样 JSP 通过 id 来识别 JavaBean，也可以通过 id.method 类似的语句来操作 JavaBean。

在执行过程中，<jsp:useBean>首先会尝试寻找已经存在的具有相同 id 和 scope 值的 JavaBean 实例，如果没有就会自动创建一个新的实例。

其具体语法如下：

```
<jsp:useBean id="name"  scope="page|request|session|application"  typeSpec>
```

```
// useBean 标签体，常用于做初始化工作
</jsp:useBean>
```

其中，typeSpec 定义如下：

```
typeSpec ::=class="className" | class="className" type="typeName"
| type="typeName" class="className" | beanName="beanName" type="typeName"
| type="typeName" beanName="beanName" | type="typeName"
```

主要属性：

id：JavaBean 对象的唯一标志，代表了一个 JavaBean 对象的实例。它具有特定的存在范围（page|request|session|application）。在 JSP 中通过 id 来识别 JavaBean。

Scope：Javabean 对象的生存时间，可以是 page、request、session 和 application 中的一种。

Class：JavaBean 对象的 class 名字，特别注意大小写要完全一致。

Type：指定了脚本变量定义的类型，默认为脚本变量定义和 class 中的属性一致，一般我们都采用默认值。

- <jsp:setProperty>标签

<jsp:setProperty>标签用于设置 bean 的属性值。JSP 中调用的语法如下：

```
<jsp:setProperty name="beanName"  last_syntax />
```

其中，name 属性代表了已经存在的并且具有一定生存范围（scope）的 JavaBean 实例。last_syntax 代表的语法如下：

```
property="*" | property="propertyName" | property="propertyName"
param="parameterName" | property="propertyName" value="propertyValue"
```

主要属性：

Name：代表通过<jsp:useBean> 标签定义的 JavaBean 对象实例。

Property：这是个很重要的属性，代表了你想设置值的属性 property 名字。如果使用 property="*"，程序就会反复地查找当前所有 Request 参数，并且匹配 JavaBean 中相同名字的属性 property，并通过 JavaBean 中属性的 set 方法赋值 value 给这个属性。如果 value 属性为空，则不会修改 JavaBean 中的属性值。

Param：代表了页面请求的参数名字，<jsp:setProperty>标签不能同时使用 param 和 value。

Value：代表了赋给 Bean 的属性 property 的具体值。

- <jsp:getProperty>标签

<jsp:getProperty>标签用来取得 JavaBean 实例的属性值，并将它们转换为 java.lang.String，最后放置在隐含的 Out 对象中。

<jsp:getProperty>标签的语法如下：

```
<jsp:getProperty name="name" property="propertyName" />
```

其中，Name：代表了想要获得属性值的 Bean 的实例，Bean 实例必须在前面用<jsp:useBean>标签定义。Property：代表了想要获得值的那个 property 的名字。

6.4.3 JSP+JavaBean 例子

现在我们看看具体的 JSP+JavaBean 的例子——一个简单的计数器程序。

本例程共包含 3 个文件：JavaBean（counter.java 文件）、JSP（counter.jsp 文件）、counter1.jsp

文件，其中，counter.java 文件主要用来进行计数器的计数操作，counter.jsp 和 counter1.jsp 文件主要用来显示网页的计数。

counter.java：

```
package count;
public class counter {
   //初始化 JavaBean 的成员变量
   int count = 0;
   public counter()
   {
   }
   // 属性 Count 的 Get 方法
   public int getCount() {
   //计数操作，每一次请求都进行计数器加一
   count++;
   return this.count;
   }
   //属性 Count 的 Set 方法
   public void setCount(int count) {
      this.count = count;
   }
}
```

counter.jsp：

```
<html>
<head>
   <title> 计数器 </title>
      </head>
<body>
   <H1> Jsp+JavaBean 程序示例</H1>
      <!-初始化 counter 这个 Bean，实例为 bean0-→
   <jsp:useBean id="bean0" scope="application" class="count.counter" />
   <%
   //显示当前的属性 count 的值，使用 out.println 方法，后面将使用另一种方法
   out.println("The Counter is : " + bean0.getCount() + "<BR>");
   %>
   </body>
</html>
```

counter1.jsp:

```
<Html>
<head>
   <title> 计数器 </title>
      </head>
      <body>
   <h1> jsp+javabean 程序示例</h1>
      <!-初始化 counter 这个 bean，实例为 bean0-→
   <jsp:usebean id="bean0" scope="application" class="count.counter" />
```

```
    <!-- 使用 jsp:getproperty 标签得到 count 属性的值，也就是计数器的值-→
    the counter is : <jsp:getproperty name="bean0" property="count" /><br>
</body>
</html>
```

从这个例子我们不难看出 JSP 和 JavaBean 应用的一般操作方法，首先在 JSP 页面中要声明并初始化 JavaBean，这个 JavaBean 有一个唯一的 id 标志，还有一个生存范围 scope（设置为 application 是为了实现多个用户共享一个计数器的功能，如果要实现单个用户的计数功能，可以修改 scope 为 session），最后还要制定 JavaBean 的 class 来源 count.counter：

```
<jsp:useBean id="bean0" scope="application" class="count.counter" />
```

接着我们就可以使用 JavaBean 提供的 public 方法或者直接使用<jsp:getProperty>标签来得到 JavaBean 中属性的值：

```
out.println("The Counter is : " + bean0.getCount() + "<BR>");
```

或者

```
<jsp:getProperty name="bean0" property="count" />
```

6.5 研究：Session 事件的监听

到此，我们已实现的游戏程序还有一个缺陷：如果游戏玩家登录游戏，进入某游戏房间，单击某桌的空位坐下，但游戏一直没开始（如玩家不足），如果此后，玩家网络断线或恶意搁置、关闭浏览器，将引起 Session 超时而被清除，但是，此时的游戏逻辑中，玩家还坐在游戏桌的位置上，即游戏桌对象的位置属性变量还指向玩家对象，然而，玩家对象已随 Session 的清除而清除或悬空，游戏桌的状态因此而可能出现混乱，程序运行会出现不可预料的结果。

因此，需要监听 Session 的超时结束事件，编写处理方法，处理好善后工作。

任务：从网络、有关书籍、帮助文档等，搜集有关 Session 事件监听的技术资料，整理出实用的知识点及案例程序，并设计缺陷的解决方案，编写程序实现它。