

第10章

手机网络游戏 (J2ME+Servlet 实现)

目前手机网络游戏的迅速崛起，让游戏运营商们又发现了一个新的发展“特区”，期待着手游能继网游之后成为新的经济增长点。手游为广大手机用户带来了一种全新的休闲方式，他们可以随时随地利用手游放松紧张的情绪，缓解学习、工作、生活中的种种压力。本章将介绍如何使用 Java 的 J2ME 技术开发一款以象棋为核心的手机网络游戏。通过阅读本章，可以学习到：

- J2ME 程序开发
- HTTP 协议的有状态连接
- J2ME 与 Tomcat 的通信
- 如何绘制游戏界面
- 游戏信息处理
- 使用 EclipseMe 工具

10.1 开发背景

加入 WTO 之后，随着 3G 时代的即将到来，中国手机网络游戏迅速崛起，成为继网游之后又一新的经济增长点。×××有限公司是一家手机游戏运营商，公司已经从网络版的手游运营中获取了大量的资金，现在总结了玩家的反馈信息，发现游戏玩家需要一些小型的休闲类游戏，例如纸牌、象棋、围棋和一些智力游戏等。现需要委托其他单位开发一个网络版的手游象棋游戏。

10.2 系统分析

10.2.1 需求分析

通过与×××有限公司的沟通和需求分析，要求系统具有以下功能。

- ☒ 系统操作简单，界面友好。
- ☒ 运行速度快，保证程序的稳定性。
- ☒ 支持多人操作。
- ☒ 支持移动网络连接。
- ☒ 以 HTTP 协议通信。

10.2.2 可行性分析

从手机游戏依托的技术平台来看，目前 Java 手机游戏保持很高的增长速度。手机游戏将是未来游戏市场的主要发展方向，只要拥有一部手机，就可以进入到全新的掌上游戏世界。游戏正成为无线增值服务的主力军，各游戏开发商早已洞察到这一点。通信公司 2.5G 和 3G 通信技术的发展，给手机用户带来了高达每秒 384Kbps 的移动带宽，将语音、图像、视频有序地结合起来，给手机用户带来了更加丰富多彩的多媒体娱乐服务。借此东风，各游戏开发商将进入全新的无线网络游戏时代，市场潜力巨大。

手机版象棋游戏采用 MIDP 1.0 开发，在低端手机上运行的同时，保证了高端手机的兼容性，其市场发展空间巨大。伴随着中国移动 GPRS 和中国联通 CDMA1X 数据业务的开展，手机游戏将是 3G 数据业务一个重要的应用领域。

10.3 系统设计

10.3.1 系统目标

根据需求分析的描述以及与用户的沟通，现制定系统实现目标如下。

- ☑ 界面设计简洁、友好、美观大方，保证直接上手便可游戏。
- ☑ 操作简单、快捷方便。
- ☑ 规则简单，方便游戏者进行游戏。
- ☑ 实现智能规则判断。
- ☑ 支持大型 Web 服务器，以 Http 协议通信。
- ☑ 向 PC 端开发靠拢，为以后 PC 与手机互联奠定基础。

10.3.2 系统功能结构

本系统的服务器端包括消息接收和桌面管理，功能结构如图 10.1 所示。客户端包括主窗体、游戏界面、消息处理等模块，其功能结构如图 10.2 所示。

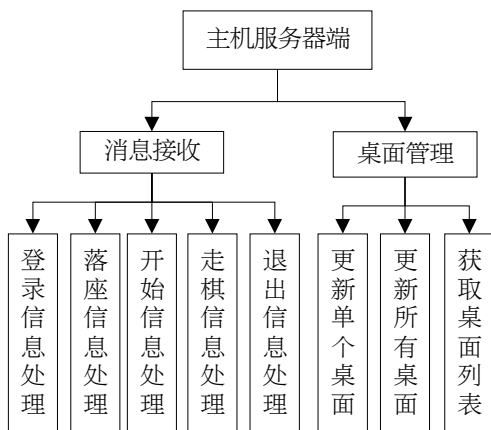


图 10.1 服务器端系统功能结构

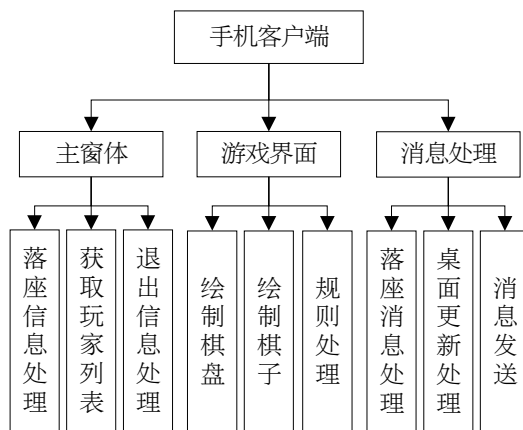


图 10.2 客户端系统功能结构

10.3.3 构建开发环境（根据语言的实际情况写）

在开发手机游戏时，分别使用了以下软、硬件环境。

- ☑ 硬件平台：
 - CPU: PIII 800GHz。
 - 内存: 256MB 以上。
 - 硬盘: 500MB 以上空间。
 - 显卡: 32MB 以上显存。

☑ 软件平台:

- 操作系统: Windows 2003 (SP1)。
- Java 开发包: JDK 1.6。
- J2ME 开发包: Wireless Toolkit 2.5.2 for CLDC。
- 分辨率: 最佳效果 1024×768 像素。
- 手机分辨率: 最佳效果为 240×292 像素。
- 开发工具: Eclipse 3.2+MyEclipse 5.1+EclipseMe 1.7.7。

10.3.4 系统预览

手机版象棋游戏的界面根据具体游戏进度而变换, 下面仅列出几个典型界面的预览, 其他页面参见光盘中的源程序。

游戏的开局界面如图 10.3 所示, 该界面是游戏的主界面, 包含游戏的规则算法、控制走棋、吃棋、选棋、退出、开始等操作。如图 10.4 所示是游戏进行到死局的界面效果, 在该界面中, 红棋已经无路可走, 它被对方的“炮”和“车”将死。



图 10.3 开局界面 (光盘\...\GameCanvas.java)

图 10.4 死局界面 (光盘\...\GameCanvas.java)

输棋的界面如图 10.5 所示, 该界面在玩家输棋的时候提示玩家“抱歉, 您失败了”。游戏胜利界面如图 10.6 所示, 该界面在玩家取得胜利的时候提示玩家“恭喜, 您获胜了”。



图 10.5 输棋界面 (光盘\...\GameCanvas.java)



图 10.6 胜利界面 (光盘\...\GameCanvas.java)

说明: 由于路径太长, 因此省略了部分路径, 省略的路径是“TM\10\xiangqi\src\com\lzw”。

10.3.5 文件夹组织结构

在进行系统开发之前, 需要规划文件夹组织结构。也就是说, 建立多个文件夹, 对各个功能模块进行划分, 实现统一管理。这样做的好处在于: 易于开发、管理和维护。开发本游戏时, 服务器文件夹组织结构和客户端文件夹组织结构分别如图 10.7 和图 10.8 所示。

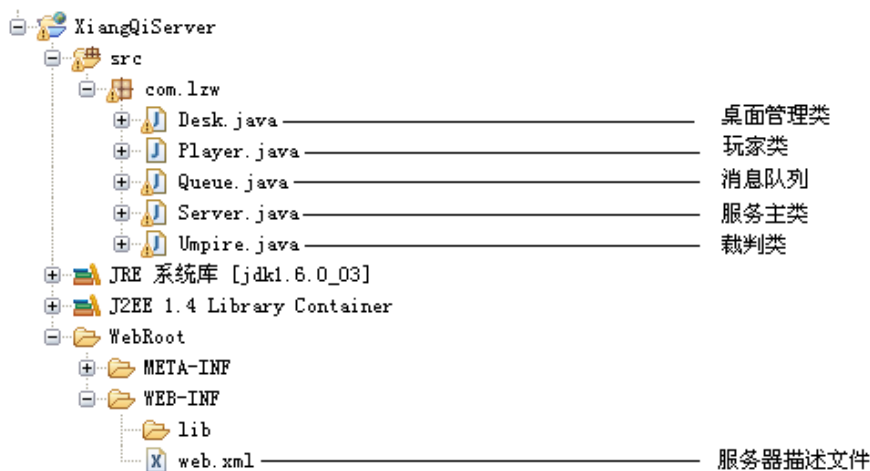


图 10.7 服务器端文件夹组织结构

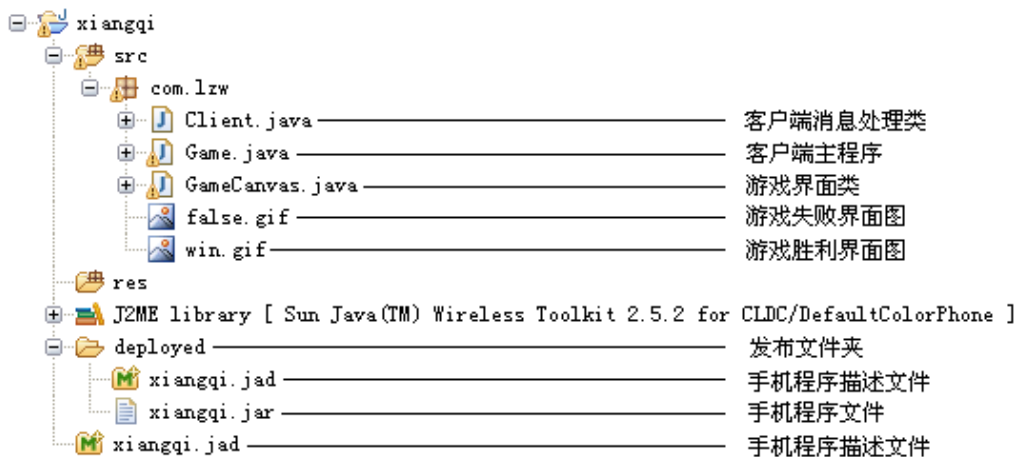


图 10.8 客户端文件夹组织结构

10.4 主程序设计

手游程序的主程序，必须继承 MIDlet 类，MIDlet 是一个抽象类，该类是手游应用程序的入口，这和 Java 应用程序中含有 main() 方法的主类相似。

Game 类继承了 MIDlet 类成为手游程序的主类，并实现了 CommandListener 接口处理相应的软键触发事件。该类在程序界面中创建了 List 列表，该列表包括游戏桌面信息和相应的操作（本实例主要是介绍游戏的开发，游戏桌面信息使用简单的字符信息“0”和“1”代替“空座”和“落座”信息）。简单的界面如图 10.9 所示。



图 10.9 简单桌面列表信息

创建主程序的步骤如下：

(1) 创建 **Game** 类，该类必须继承 **MIDlet** 类并实现 **CommandListener** 接口。在类中创建并初始化程序画布类（即游戏界面）、消息处理、桌面列表、软健按钮等对象。关键代码如下：

例程 01 代码位置：光盘\TM\10\xiangqi\src\com\lzw\Game.java

```
public class Game extends MIDlet implements CommandListener {
    private Client client;                                //声明客户端消息处理对象
    public static Display display;                       //声明静态的设备对象
    private GameCanvas canvas;                          //声明游戏画布对象
    private List playerList;                              //声明列表控件对象
    private int[][] desks;                                //声明桌面数组
    private int trySeat, tryDesk;                         //声明落座的桌号和位置变量
    private Command okCommand;                          //声明落座按钮控件对象
    public Game() {                                       //在构造方法中初始化所有类成员
        display = Display.getDisplay(this);
        playerList = new List("Online player", Choice.EXCLUSIVE);
        Command exitCommand = new Command("退出", Command.EXIT, 0);
        playerList.addCommand(exitCommand);
        okCommand = new Command("落座", Command.OK, 0);
        playerList.addCommand(okCommand);
        playerList.setCommandListener(this);
        display.setCurrent(playerList);
        client = new Client(this);
    }
    ...//省略部分代码
}
```

(2) 实现 **MIDlet** 类规定的 3 个声明周期方法，其中 **startApp()** 方法在手游程序被加载时，初始化程序需要的数据，**pauseApp()** 方法在手游程序被暂停或者被手机来电中断时暂存游戏数据，**destroyApp()** 方法在退出手游程序时负责清空程序数据，释放所有资源。这些方法是父类的抽象方法，即使不实现方法逻辑，也要编写相应的方法声明。关键代码如下：

例程 02 代码位置：光盘\TM\10\xiangqi\src\com\lzw\Game.java

```
protected void startApp() throws MIDletStateChangeException {
}
protected void pauseApp() {
}
protected void destroyApp(boolean p0) throws MIDletStateChangeException {    //在程序退出时，发送 exit 消息
    client.sendMessage("exit");
    display.setCurrent(null);
}
```

(3) 实现 `CommandListener` 接口中规定的 `commandAction()` 方法，该方法用于处理软键的触发事件。当玩家触发“退出”按钮时，该方法将使游戏退出并结束手游程序。当玩家触发“落座”按钮时，该方法将获取桌面列表中选择当前列表项，并分析该桌面的玩家信息，如果该桌面有空位置，则设置游戏玩家的桌号和座位号，并显示游戏界面。关键代码如下：

例程 03 代码位置：光盘\TM\10\xiangqi\src\com\lzw\Game.java

```
public void commandAction(Command c, Displayable s) {
    if (c.getCommandType() == Command.EXIT) {    //当玩家触发退出按钮时
        client.sendMessage("exit");
        try {
            destroyApp(false);
            notifyDestroyed();    //退出并结束手游程序
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else if (c == okCommand) {    //当玩家触发落座按钮时
        if (playerList.getSelectedIndex() >= 0) {
            try {
                String info = playerList.getString(playerList
                    .getSelectedIndex());    //获取桌面信息
                int index1 = info.indexOf("-");
                int d = Integer.parseInt(info.substring(0, index1));
                int index2 = info.indexOf("-", index1 + 1);
                int d1 = Integer.parseInt(info
                    .substring(index1 + 1, index2));
                int d2 = Integer.parseInt(info.substring(index2 + 1));
                if (d1 == 0 || d2 == 0) {    //如果该桌面有空的座位
                    if (d1 == 0)    //设置玩家座位号为 0
                        trySeat = 0;
                    else    //或者设置玩家座位号为 1
                        trySeat = 1;
                    tryDesk = d;
                    if (canvas == null)    //初始化游戏界面的画布对象
                        canvas = new GameCanvas(this, client);
                    else
                        canvas.init();    //调用游戏界面的 init() 方法
                    client.sendMessage("take," + d + "," + trySeat);    //调用 sendMessage() 方法发送落座消息
                }
            }
        }
    }
}
```



```

    }
    } catch (Exception exc) {
        System.out.println("Error parseInt");
        exc.printStackTrace();
    }
}
}
}
}

```

(4) 编写 `takeSeat()` 方法, 该方法用于处理玩家的落座信息。当程序接收到服务器的 `takeseat` 信息时, 将调用该方法完成程序界面的初始化、设置玩家座位、桌号, 并显示游戏界面。关键代码如下:

例程 04 代码位置: 光盘\TM\10\xiangqi\src\com\lzw\Game.java

```

public void takeSeat() {                                //处理落座信息的方法
    if (canvas == null)                                  //如果游戏界面对象未初始化
        canvas = new GameCanvas(this, client);          //初始化游戏界面对象
    else
        canvas.init();                                   //调用游戏界面的 init()方法
    canvas.setSeatPos(trySeat);                           //设置玩家座位
    canvas.setDeskIndex(tryDesk);                         //设置玩家桌号
    display.setCurrent(canvas);                          //显示游戏界面
}

```

(5) 编写 `updateDesk()` 方法, 该方法用于更新桌面列表的单个列表项, 也就是单个桌面的座位信息。当程序接收到服务器的 `updatedesk` 信息时, 将调用该方法解析 `updatedesk` 信息之后的桌面数据, 并更新到程序界面中。关键代码如下:

例程 05 代码位置: 光盘\TM\10\xiangqi\src\com\lzw\Game.java

```

public void updateDesk(String str) {                    //更新桌面
    int index1 = str.indexOf(",");                      //解析分隔符的位置
    int index2 = str.indexOf(":", index1 + 1);
    int index3 = str.indexOf(";", index2 + 1);
    try {                                                //截取桌面数据到 desks 数组中
        int d = Integer.parseInt(str.substring(index1 + 1, index2));
        desks[d][0] = Integer.parseInt(str.substring(index2 + 1, index3));
        desks[d][1] = Integer.parseInt(str.substring(index3 + 1));
        playerList.set(d, d + "-" + desks[d][0] + "-" + desks[d][1], null); //更新桌面列表
    } catch (Exception exc) {
    }
}

```

(6) 编写 `setDesks()` 方法, 该方法用于设置整个桌面列表。当程序接收到服务器的 `desks` 信息时 (该信息包括了所有桌面信息), `setDesks()` 方法必须解析这些信息并更新游戏主程序的所有桌面。关键代码如下:

例程 06 代码位置: 光盘\TM\10\xiangqi\src\com\lzw\Game.java

```

public void setDesks(String string) {
    for (int i = 0; i < playerList.size(); i++)          //清除原桌面列表

```

```

        playerList.delete(i);
        int index1, index2, index3, index4, index0;           //解析分隔符位置
        index1 = string.indexOf(",");
        index2 = string.indexOf(":", index1 + 1);
        int desknum = Integer.parseInt(string.substring(index1 + 1, index2)); //解析桌面编号
        desks = new int[desknum][4];                         //初始化桌面数组
        index0 = index2;
        int counter = 0;
        while (counter < desknum) {                           //解析服务器发送的桌面列表信息
            index1 = string.indexOf(",", index0 + 1);
            index4 = string.indexOf(":", index1 + 1);
            desks[counter][0] = Integer.parseInt(string.substring(index0 + 1, //将解析后的数据存入数组中
                index1));
            if (index4 > 0)
                desks[counter][1] = Integer.parseInt(string.substring(
                    index1 + 1, index4));
            else {
                string = string.trim();
                desks[counter][1] = Integer.parseInt(string
                    .substring(index1 + 1));
            }
            playerList.append(counter + "-" + desks[counter][0] + "-" //将数组中的数据更新到主程序界面
                + desks[counter][1], null);
            index0 = index4;
            counter++;
        }
    }

```

10.5 公共模块设计

本系统的服务器项目空间中，有部分程序是公用的，它们被多个模块甚至整个系统重复调用完成指定的业务逻辑，本节将这些公共的模块提出来进行单独介绍。

10.5.1 创建 Player 公共类

Player 类代表玩家对象。每个玩家都有不同的属性，这些属性用来表示唯一的玩家。在现实社会中，没有完全相同的两个人物，而游戏中也应如此，否则无法保证游戏的数据安全。要做到区分不同的玩家，就必须为每个玩家添加 IP 地址和端口属性。另外，玩家会坐在不同桌子的不同座位上、有不同颜色的棋子、独立的消息队列和开始状态等，这些都需要声明对应的属性并记录属性状态。关键代码如下：

例程 07 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Player.java

```

public class Player {
    private int ID = 1;           //玩家编号
    private String IP = "";       //玩家 IP 地址

```

```

private int PORT = 9999;           //玩家端口号
private Desk desk;                //玩家所在的桌面对象
public Queue data;               //消息队列
private String Color = "";        //玩家手中象棋的颜色
public boolean start = false;      //开始状态
public Player(String ip, int p){
    IP = ip;                       //初始化 IP
    PORT = p;                      //初始化端口
    desk = null;
    data = new Queue();            //初始化消息队列
}
public void setDesk(Desk d) {      //设置玩家桌号的方法
    desk = d;
}
public Desk getDesk() {            //获取玩家桌号的方法
    return desk;
}
public boolean equals(String ip) { //判断是否同一个 IP 玩家的方法
    if (IP.equals(ip))
        return true;
    else
        return false;
}
public boolean equals(Player p) { //判断是否同一个玩家对象的方法
    if (IP.equals(p.getIP()) && PORT == p.getPort())
        return true;
    else
        return false;
}
public String getIP() {            //获取 IP 的方法
    return IP;
}
public int getPort() {             //获取端口号的方法
    return PORT;
}
...//省略部分代码
public boolean isStart() {         //判断玩家是否开始游戏的方法
    return start;
}
public void init() {              //玩家的初始化方法
    start = false;                //设置未开始游戏状态
    data.clear();                 //清除消息队列
    IP="";
    PORT=0;
}
public void reset() {             //玩家复位方法
    Color = "";
    start = false;

```



```

    }
    public String getColor() {                                //获取玩家象棋颜色的方法
        return Color;
    }
    public void setColor(String color) {                       //设置玩家象棋颜色的方法
        Color = color;
    }
}

```

10.5.2 创建 Queue 公共类

编写消息队列公共类 `Queue`，该类负责存储服务器发送给玩家的消息，当手机客户端发出请求信息时，服务器会从该客户端对应的玩家对象的消息队列中获取消息，如果消息队列中没有消息，那么手机客户端必须等待服务器为其分配消息。

`Queue` 公共类遵循“先进先出”的存储结构，数据元素只能从队尾进入，从队首取出。在队列中，数据元素可以任意增减，但数据元素的次序不会改变。每当有数据元素从队列中被取出，后面的数据元素依次向前移动一位。所以，任何时候从队列中读到的都是队首的数据。关键代码如下：

例程 08 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Queue.java

```

public class Queue extends java.util.Vector {                //继承向量实现集合类
    public class EmptyQueueException extends java.lang.RuntimeException { //创建自定义异常类
        public EmptyQueueException() {
            super();                                           //调用父类的构造方法
        }
    }
    public Queue() {                                           //Queue 公共类的构造方法
        super();
    }
    ▪ public synchronized void push(Object x) {                //添加队列消息的方法
        super.addElement(x);
    }
    ▪ public synchronized Object pop() {                       //获取队列消息的方法
        /* 队列若为空，引发 EmptyQueueException 异常 */
        if (this.empty())
            throw new EmptyQueueException();                 //抛出异常的方法
        Object x = super.elementAt(0);                        //获取第一个消息
        super.removeElementAt(0);                             //从队列中移除第一个消息
        return x;
    }
    ▪ public synchronized Object front() {                    //读取队首消息的方法
        if (this.empty())
            throw new EmptyQueueException();
        return super.elementAt(0);
    }
}

```

```

    ▪ public boolean empty() {                                     //判断队列消息是否为空的方法
        return this.isEmpty();
    }

    ▪ public synchronized void clear() {                          //清除队列消息的方法
        super.removeAllElements();
    }

    ▪ public int search(Object x) {                                //搜索消息的方法
        return super.indexOf(x);
    }
}

```

■ 代码贴士

- **push(Object x)**：向队列插入一个值为 *x* 的元素。
- **pop()**：从队列中取出一个元素。
- **front()**：从队列中读一个元素，但队列保持不变。
- **empty()**：判断队列是否为空，空则返回 **true**。
- **clear()**：清空队列。
- **search(x)**：查找距队首最近的元素的位置，若不存在，返回 -1。

10.5.3 创建 Umpire 公共类

服务器端的 **Umpire** 公共类用于裁判游戏输赢、记录棋盘数据。在 **Desk** 公共类中将调用 **Umpire** 类的 **moveChess()** 方法更新棋盘数据的记录。另外该公共类还定义了其他方法来更改棋盘数据，例如 **init()** 方法可以初始化棋盘数据到开局状态。关键代码如下：

例程 09 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Umpire.java

```

public class Umpire {
    private int huihe;                                     //记录回合
    private int bigID;                                    //记录先手玩家的编号
    private int score;                                    //记录分数
    protected static int i, j;                            //棋盘数组的下标编号
    protected static int isRedWin = 1;                   //判断红色玩家的胜利记录
    protected static int isWhiteWin = 1;                 //判断白色玩家的胜利记录
    private int point[][];                                //声明棋盘记录数组
    public Umpire() {
        huihe = 0;                                        //初始化回合记录
        score = 0;                                       //初始化分数记录
        point = new int[][] { { 1, 2, 3, 4, 5, 6, 7, 8, 9 }, //初始化棋盘数据记录
                               { 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 10, 0, 0, 0, 0, 0, 11, 0 },
                               { 12, 0, 13, 0, 14, 0, 15, 0, 16 },
                               { 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                               { 28, 0, 29, 0, 30, 0, 31, 0, 32 },
                               { 0, 26, 0, 0, 0, 0, 27, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                               { 17, 18, 19, 20, 21, 22, 23, 24, 25 } };
    }
}

```

```

▪ public void checkWin() { //判断输赢的方法
    isRedWin = 0;
    isWhiteWin = 0;
    for (i = 0; i < 3; i++) { //遍历“将”的活动范围
        for (j = 0; j < 3; j++) {
            if (point[0 + i][3 + j] == 5) { //如果失去主将
                isRedWin++; //则另一方胜利
            }
        }
    }
    for (i = 0; i < 3; i++) { //遍历“帅”的活动范围
        for (j = 0; j < 3; j++) {
            if (point[7 + i][3 + j] == 21) { //如果失去主帅
                isWhiteWin++; //则对方胜利
            }
        }
    }
}

▪ public void moveChess(int selectedY, int selectedX, int n, int m) { //记录走棋的方法
    point[selectedY][selectedX] = point[n][m]; //更新棋盘指定位置的数据
    point[n][m] = 0; //移动棋子之后，将原位置清空
    checkWin(); //调用判断输赢的方法
}

public void logHuihe() { //记录回合数的方法
    huihe++;
}

public int getHuihe() { //获取回合数的方法
    return huihe;
}

▪ public void init() { //初始化方法，在开始新局游戏时调用
    huihe = 0; //回合归零
    score = 0; //分数归零
    isRedWin = 1; //初始化两方胜利记录
    isWhiteWin = 1;
    point = new int[][] { { 1, 2, 3, 4, 5, 6, 7, 8, 9 }, //初始化棋盘数据
        { 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 10, 0, 0, 0, 0, 0, 11, 0 },
        { 12, 0, 13, 0, 14, 0, 15, 0, 16 },
        { 0, 0, 0, 0, 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 28, 0, 29, 0, 30, 0, 31, 0, 32 },
        { 0, 26, 0, 0, 0, 0, 27, 0 }, { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 17, 18, 19, 20, 21, 22, 23, 24, 25 } };
    }
    ...//省略部分代码
}

```

■ 代码贴士

- checkWin()：该方法分别搜索游戏双方的主将，如果有一方失去主将，则另一方胜利。

moveChess(): 该方法将根据用户的操作更改棋子的位置, 然后调用 checkWin() 方法, 判断此次走棋是否导致另一方失败。

init(): 该方法用于初始化新一局游戏。它将执行清除胜负记录、回合数、复位棋盘操作。

注意: 在程序中所描述的白色象棋棋子在后期美工设计时, 为使界面美观, 被设计成黑色的棋子。

10.5.4 创建 Desk 公共类

服务器端的 Desk 公共类是游戏桌面的定义类, 该类包含了游戏桌面的编号、玩家数量、游戏回合局数等属性, 另外该类还包括 Umpire 对象、Player 对象、Server 对象等属性, 其中 Server 对象是服务器的主程序, 它负责信息的接收、发送和处理等业务。Desk 公共类的属性声明与初始化的关键代码如下:

例程 10 代码位置: 光盘\TM\10\XiangQiServer\src\com\lzw\Desk.java

```
public class Desk {
    private int ID;                //桌子 ID
    private Player[] players;      //玩家数组
    private int NUM = 2;           //玩家数量
    private Player banker = null;  //先手
    private int bankerID = 0;      //先手 ID
    private Umpire umpire;         //棋盘记录
    private int game = 0;          //游戏局数
    private int score = 0;         //分数
    Server server;                 //服务器信息处理对象
    public Desk() {
        game = 0;                 //初始化游戏局数
        server = new Server();     //初始化信息处理对象
        umpire = new Umpire();     //初始化棋盘记录对象
        players = new Player[NUM]; //初始化玩家数组
        banker = null;             //初始化先手
        bankerID = 0;              //初始化先手编号
        for (int i = 0; i < NUM; i++) {
            players[i] = null;     //初始化同一桌面的玩家
        }
    }
    public void init() {           //桌面的初始化方法
        banker = null;
        bankerID = 0;
        game = 0;
        umpire.init();             //初始化棋盘记录对象
    }
}
```

Desk 公共类将控制游戏的初始化、开始、重新开始、游戏结束、走棋等游戏中的业务逻辑, 这些业务逻辑有不同的方法实现。关键代码如下:



例程 11 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Desk.java

```

public void reset(){                                     //桌面复位方法
    umpire.init();
    for(int i=0;i<NUM;i++)
        players[i].reset();
}
public void start() {                                   //游戏开始方法
    players[bankerID].setColor("red");
    players[(bankerID+1)%NUM].setColor("white");
    sendMessage(players[bankerID], "color:"+players[bankerID].getColor());
    sendMessageToOther(players[bankerID], "color:"+players[(bankerID+1)%NUM].getColor());
    sendMessage(players[bankerID], "turn");
}
public void moveChess(String message) {                 //走棋方法
    int index0 = message.indexOf(";");
    int index1 = message.indexOf(":");
    int index2 = message.indexOf(",", index1 + 1);
    int index3 = message.indexOf(",", index2 + 1);
    int index4 = message.indexOf(",", index3 + 1);
    int seat = Integer.parseInt(message.substring(index0 + 1, index1));
    int selectedY = Integer.parseInt(message.substring(index1 + 1, index2));
    int selectedX = Integer.parseInt(message.substring(index2 + 1, index3));
    int n = Integer.parseInt(message.substring(index3 + 1, index4));
    int m = Integer.parseInt(message.substring(index4 + 1));
    umpire.moveChess(selectedY, selectedX, n, m);        //调用 Umpire 类的 moveChess()方法
    sendMessageToOther(players[seat], message);
    if (Umpire.isWhiteWin == 0) {                       //判断输赢
        sendMessage(players[bankerID], "win,you");
        sendMessageToOther(players[bankerID], "win");
        bankerID = (seat + 1) % NUM;
        reset();
    } else if (Umpire.isRedWin == 0) {                   //判断输赢
        sendMessage(players[(bankerID+1)%NUM], "win,you");
        sendMessageToOther(players[(bankerID+1)%NUM], "win");
        bankerID = (seat + 1) % NUM;
        reset();
    }
}
}

```

在 Desk 公共类中必须定义控制和获取玩家信息的方法，例如判断玩家是否坐满，也就是说桌面上是否有两个玩家来共同参与游戏；另外，还需要设置并获取玩家位置等控制玩家的操作。关键代码如下：

例程 12 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Desk.java

```

public boolean isReady() {                             //本桌是否坐满，并且都开始
    for (int i = 0; i < NUM; i++) {
        if (players[i] == null)
            return false;
    }
}

```



```

        else if (!players[i].isStart())           //是否开始游戏
            return false;
    }
    umpire.init();                               //调用 Umpire 类的 init()方法
    return true;
}
public boolean isEmpty(int pos) {                //座号 pos 是否空
    if (pos >= NUM)
        return false;
    return players[pos] == null;
}
public int getPlayerSeat(Player p)              //返回玩家座位
{
    for (int i = 0; i < NUM; i++) {
        if (players[i] == null)
            continue;
        if (players[i].equals(p))                //判断指定玩家和座位上的玩家是否为同一个玩家
            return i;
    }
    return -1;
}
public void setPlayer(int pos, Player n) {       //设定玩家 n 坐在 pos 座位上
    if (pos >= NUM)
        return;
    players[pos] = n;
}
public void removePlayer(Player p) {            //移除玩家 p
    for (int i = 0; i < NUM; i++) {
        if (players[i] == null)
            continue;
        else if (players[i].equals(p))
            players[i] = null;
    }
}
}

```

Desk 公共类在管理游戏进度、规则、走棋记录等操作时，需要向玩家发送通知信息，这些信息需要分别发送给所有玩家、单个玩家和除自己以外的其他玩家。这就需要编写不同的消息发送方法。关键代码如下：

例程 13 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Desk.java

```

public void sendMessageToAll(String mes) {       //发送信息到所有玩家
    for (int i = 0; i < NUM; i++)
        if (players[i] != null)
            sendMessage(players[i], mes);
}
public void sendMessageToOther(Player player, String message) { //发送信息到其他玩家
    for (int i = 0; i < NUM; i++) {
        if (players[i] != null && !players[i].equals(player)) //判断接收信息的玩家是否是自己

```

```
        sendMessage(players[i], message);
    }
}

public void sendMessage(Player p, String m) {           //发送信息到单个玩家的方法
    server.sendMessage(p, m);
}

public void sendBankerInfo() {                          //发送象棋先手信息
    sendMessageToAll("bankerInfo:" + bankerID);
}
}
```

10.6 游戏模块设计

10.6.1 游戏模块概述

手机的游戏模块包括界面设计、规则算法、按键处理等。对于手机游戏来说，界面要根据游戏所针对的客户群体来设计，只有使客户群体接受游戏界面，游戏产品才能拥有更好的卖点。例如，针对白领人士，需要制作适合 30 岁以上的，集稳重、智慧、精干于一体的有些类似办公的界面，界面要简洁，不需要太多的颜色装饰。规则算法是游戏公平和完整性的保障。按键处理负责游戏中的具体操作，例如走棋、选棋等。游戏模块的界面如图 10.10 所示。

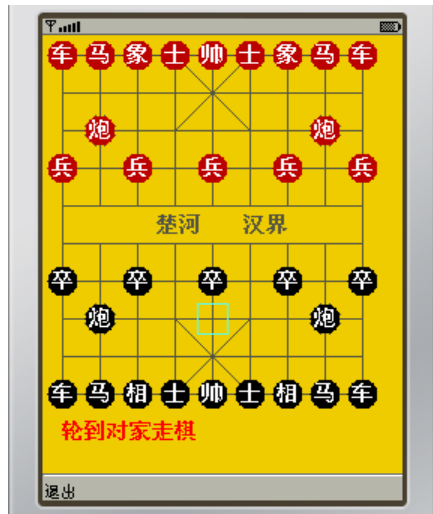


图 10.10 游戏界面

10.6.2 游戏模块技术分析

Canvas 是 MIDP 提供的低级用户界面类。和高级用户界面相比，Canvas 拥有更大的灵活性。由于 Canvas 不提供任何现成的可视组件，所以要在 Canvas 上显示图形或者文本，都必须通过 Graphics 类

绘制出来。如果使用高级用户界面，不但界面死板无法更改，而且也缺少相应的灵活性，不能用于开发游戏。因此，使用低级用户界面，开发人员可以获得完全的界面控制能力，并能精确地控制每一个像素的位置，在游戏开发中这是必不可少的。Canvas 类定义的常用方法如表 10.1 所示。

表 10.1 Canvas 类定义的常用方法

方法名称	说 明
paint	绘图方法，用于绘制游戏界面
repaint	重新执行绘图方法
isDoubleBuffered	判断是否支持双缓存技术
getKeyCode	获取指定动作对应的按键代码

10.6.3 棋盘绘制模块实现过程

(1) 游戏界面中最重要的就是棋盘界面，包括棋盘和棋子的大小、颜色、棋子布局位置等多个属性，所以定义棋盘绘制模块的属性是首要任务。创建 GameCanvas 类，该类必须继承 Canvas 类成为游戏界面的画板，游戏中的所有事务，都是在这个画板上绘制的。另外，该类还需要实现 CommandListener 接口来处理软键按钮的事件处理，在该类中声明游戏中需要的各种属性。关键代码如下：

例程 14 代码位置：光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```
public class GameCanvas extends Canvas implements CommandListener {
    protected Game game; //游戏主类对象
    String color = "";
    protected int rightSpace; //屏幕右侧预留的空间
    protected int x; //棋盘输出的坐标
    private boolean myTurn = false;
    protected int gridWidth; //每个棋格的边长
    protected int mapWidth, canvasW; //棋盘的宽度和画布的宽度
    protected int a, b, c, d;
    protected int chessR; //棋子的半径
    private int desknum = -1; //桌子序号
    private int seatPos = -1; //座位序号
    private boolean banker = false;
    protected int selectedX, selectedY; //选择框在棋盘格局上的 x,y 位置
    protected static int i, j;
    protected int m, n, p; //记录开始的选择框位置和在数组中的位置
    protected String q; //记住 word[selectedX][selectedY]
    protected int guard, guard1, guard2, g, g1; //标记 FIRE 被按了多少次
    protected static int turnWho; //表示该谁走了
    protected static int isRedWin; //红棋胜利
    protected static int isWhiteWin; //白棋胜利
    private Client client; //消息管理对象
}
```

```

protected Command exitCmd, start, ok;           //软键按钮
private int point[][];                          //棋子位置数组
protected String[][] chess;                    //棋子名称数组
private int chessSelColor;                      //选择棋子的颜色
private int backColor;                         //棋盘背景色
private int charColor;                         //棋子汉字颜色
private int lineColor;                         //棋盘线的颜色
private int borderColor;                      //楚河汉界的颜色
private int selBorderColor;                   //选择棋子的边框色
private int blackChees;                       //黑棋颜色
private int redChess;                         //红棋颜色
...//省略部分代码
}

```

(2) 在构造方法中初始化游戏界面中的属性。因为构造方法用于创建类的实例对象，所以在构造方法中初始化游戏界面中的所有属性，可以保证游戏界面对象被使用之前是已经被初始化的。关键代码如下。

例程 15 代码位置：光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```

public GameCanvas(Game game, Client client) {    //构造函数
    this.game = game;                          //初始化游戏主类对象
    this.client = client;                      //初始化信息管理对象
    chessSelColor = 0x188312;                  //初始化选择棋子的颜色
    backColor = 0xEECD05;                     //初始化背景色
    charColor = 0xFFFFF;                     //初始化汉字颜色
    lineColor = 0x5A5743;                     //初始化棋盘线的颜色
    borderColor = 0x5A5743;                   //初始化楚河汉界的颜色
    selBorderColor = 0x50FAFC;                 //初始化选择棋子的边框色
    blackChees = 0x000000;                    //初始化黑棋颜色
    redChess = 0xBF0404;                      //初始化红棋颜色
    rightSpace = getWidth() / 6;              //初始化棋盘右侧预留空间
    x = rightSpace * 1 / 3;
    canvasW = getWidth() - rightSpace;        //初始化游戏界面的画板宽度
    mapWidth = canvasW - canvasW % 8;         //初始化棋盘宽度
    gridWidth = mapWidth / 8;                 //初始化棋子的网格宽度
    a = gridWidth * 2 / 5;
    b = gridWidth / 8;
    c = gridWidth - a;
    d = gridWidth - b;
    chessR = gridWidth * 2 / 5;               //初始化棋子的半径
    selectedX = 4;                            //初始化选择框的 X 轴位置
    selectedY = 7;                            //初始化选择框的 Y 轴位置
    guard = 0;
    guard1 = selectedX;
    guard2 = selectedY;
    m = guard1;
    n = guard2;
}

```

```

    chess = new String[10][9];           //初始化棋子数组
    turnWho = 1;                         //初始化先手
    initChess();                         //调用初始化棋子数组的方法
    exitCmd = new Command("退出", Command.EXIT, 0); //初始化退出软键的按钮
    start = new Command("开始", Command.OK, 1);   //初始化开始软键的按钮
    addCommand(start);                     //添加开始软键
    addCommand(exitCmd);                  //添加退出软键
    setCommandListener(this);             //设置软键命令监听器为自身对象
}

```

(3) 编写初始化棋子数组的 `initChess()` 方法, 该方法需要初始化棋子位置的数组 `point`, 该数组记录了每个棋子在棋盘上的摆放位置, 该位置是按照象棋棋子的摆放规则排列的, 在绘制棋子的时候需要根据该数组中的定位信息确定不同的棋子名称。`chess` 是一个二维数组, 它可以存放棋盘中对应的棋子名称。在定义好棋子布局的数组之后, 需要马上初始化每个棋子对应的名称, 如车、马、象、士、将等。关键代码如下:

例程 16 代码位置: 光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```

public void initChess() {
    point = new int[][] {
        { 1, 2, 3, 4, 5, 6, 7, 8, 9 },           //初始化棋盘第一行棋子
        { 0, 0, 0, 0, 0, 0, 0, 0, 0 },           //初始化棋盘第二行棋子
        { 0, 10, 0, 0, 0, 0, 0, 11, 0 },          //初始化棋盘第三行棋子
        { 12, 0, 13, 0, 14, 0, 15, 0, 16 },       //初始化棋盘第四行棋子
        { 0, 0, 0, 0, 0, 0, 0, 0, 0 },           //初始化棋盘第五行棋子
        { 0, 0, 0, 0, 0, 0, 0, 0, 0 },           //初始化棋盘第六行棋子
        { 28, 0, 29, 0, 30, 0, 31, 0, 32 },       //初始化棋盘第七行棋子
        { 0, 26, 0, 0, 0, 0, 0, 27, 0 },          //初始化棋盘第八行棋子
        { 0, 0, 0, 0, 0, 0, 0, 0, 0 },           //初始化棋盘第九行棋子
        { 17, 18, 19, 20, 21, 22, 23, 24, 25 };  //初始化棋盘第十行棋子
    };

    chess = new String[][] {
        {"车", "马", "象", "士", "将", "士", "象", "马", "车"}, //初始化每行棋子的名称
        {"空", "空", "空", "空", "空", "空", "空", "空", "空"}, //如果棋盘对应的位置没有棋子
        {"空", "炮", "空", "空", "空", "空", "空", "炮", "空"}, //将显示“空”
        {"兵", "空", "兵", "空", "兵", "空", "兵", "空", "兵"},
        {"空", "空", "空", "空", "空", "空", "空", "空", "空"},
        {"空", "空", "空", "空", "空", "空", "空", "空", "空"},
        {"卒", "空", "卒", "空", "卒", "空", "卒", "空", "卒"},
        {"空", "炮", "空", "空", "空", "空", "空", "炮", "空"},
        {"空", "空", "空", "空", "空", "空", "空", "空", "空"},
        {"车", "马", "相", "士", "帅", "士", "相", "马", "车"},
    };
}

```

■ 代码贴士

- point: 这是一个整数类型的二维数组, 用于记录每个棋子的位置。这个二维数组对棋盘进行了平面的抽象。

- chess：这是一个字符串类型的二维数组，记录了每个棋子的名称，如将、马、炮等。

(4) 编写 `paintChessboardUp()` 方法，该方法用于绘制棋盘的上半部网格线，其中棋子“炮”的十字线画法比较复杂，它需要分别绘制左上方、右上方、左下方、右下方的横线和竖线。执行该方法后，游戏界面的棋盘效果如图 10.11 所示。

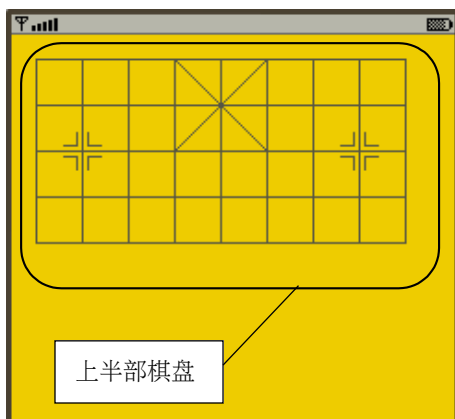


图 10.11 绘制上半部棋盘的效果

`paintChessboardUp()` 方法的关键代码如下：

例程 17 代码位置：光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```
protected void paintChessboardUp(Graphics g) { //画上半部棋盘
    for (int k = 0; k < 4; k++) {
        for (int l = 0; l < 8; l++) {
            g.setColor(lineColor);
            g.drawRect(x + l * gridWidth, x + k * gridWidth, gridWidth,
                gridWidth);
        }
    }
    g.setColor(lineColor); //设置画棋盘线的颜色
    g.drawLine(x + 3 * gridWidth, x, x + 5 * gridWidth, x + 2 * gridWidth);
    g.drawLine(x + 5 * gridWidth, x, x + 3 * gridWidth, x + 2 * gridWidth);
    { //画左上方的炮
        g.drawLine(x + d, x + gridWidth + c, x + d, x + gridWidth + d); //左上竖
        g.drawLine(x + c, x + gridWidth + d, x + d, x + gridWidth + d); //左上横
        g.drawLine(x + d + 2 * b, x + gridWidth + c, x + d + 2 * b, x + gridWidth + d); //右上竖
        g.drawLine(x + gridWidth + b, x + gridWidth + d, x + gridWidth + a, x + gridWidth + d); //右上横
        g.drawLine(x + d, x + 2 * gridWidth + b, x + d, x + 2 * gridWidth + a); //左下竖
        g.drawLine(x + c, x + gridWidth + d + 2 * b, x + d, x + gridWidth + d + 2 * b); //左下横
        g.drawLine(x + d + 2 * b, x + 2 * gridWidth + b, x + d + 2 * b, x + 2 * gridWidth + a); //右下竖
        g.drawLine(x + gridWidth + b, x + gridWidth + d + 2 * b, x
            + gridWidth + a, x + gridWidth + d + 2 * b); //右下横
    }
    { //画右上方的炮
        g.drawLine(x + d + 6 * gridWidth, x + gridWidth + c, x + d + 6 * gridWidth, x + gridWidth + d);
```

```

g.drawLine(x + c + 6 * gridWidth, x + gridWidth + d, x + d + 6 * gridWidth, x + gridWidth + d);
g.drawLine(x + d + 2 * b + 6 * gridWidth, x + gridWidth + c, x + d
+ 2 * b + 6 * gridWidth, x + gridWidth + 13 + 9);
g.drawLine(x + gridWidth + b + 6 * gridWidth, x + gridWidth + d, x
+ gridWidth + a + 6 * gridWidth, x + gridWidth + d);
g.drawLine(x + d + 6 * gridWidth, x + 2 * gridWidth + b, x + d + 6
* gridWidth, x + 2 * gridWidth + a);
g.drawLine(x + c + 6 * gridWidth, x + gridWidth + d + 2 * b, x + d
+ 6 * gridWidth, x + gridWidth + d + 2 * b);
g.drawLine(x + d + 2 * b + 6 * gridWidth, x + 2 * gridWidth + b, x
+ d + 2 * b + 6 * gridWidth, x + 2 * gridWidth + a);
g.drawLine(x + gridWidth + b + 6 * gridWidth, x + gridWidth + d + 2
* b, x + gridWidth + a + 6 * gridWidth, x + gridWidth + d + 2 * b);
}
}

```

(5) 编写 `paintRiver()` 方法, 该方法将绘制棋盘上的“楚河汉界”。象棋的棋盘以“楚河汉界”分割为上、下两部分, 每个部分分别摆放对战玩家的棋子。绘制“楚河汉界”后的棋盘效果如图 10.12 所示。

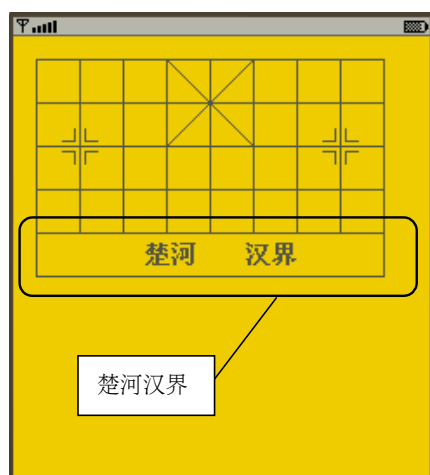


图 10.12 绘制楚河汉界后的棋盘界面

`paintRiver()` 方法的关键代码如下:

例程 18 代码位置: 光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```

protected void paintRiver(Graphics g)//画河
{
    g.setColor(lineColor); //设置楚河汉界的颜色
    g.drawRect(x, x + 4 * gridWidth, mapWidth, gridWidth);
    g.setFont(Font.getFont(Font.FACE_PROPORTIONAL, Font.STYLE_BOLD,
        Font.SIZE_LARGE)); //设置字体
    g.setColor(borderColor);
    g.drawString("楚河    汉界", getWidth() / 2, x + 4 * gridWidth
        + gridWidth * 3 / 4, Graphics.HCENTER | Graphics.BASELINE); //绘制楚河汉界
}

```

(6) 编写 `paintChessboardDown()` 方法, 该方法用于绘制棋盘的下半部网格线。它和绘制棋盘上半部分的方法相似, 不过绘制的位置不同。执行该方法后, 游戏界面的棋盘效果如图 10.13 所示。

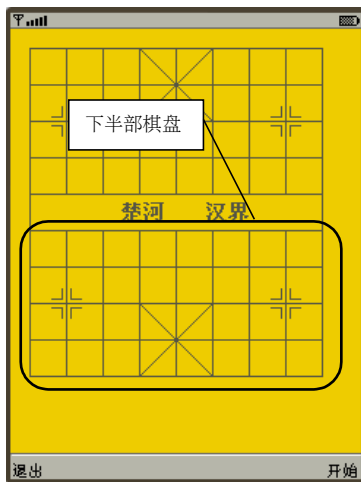


图 10.13 绘制棋盘下半部后的界面

关键代码如下:

例程 19 代码位置: 光盘\TM\10\xiangqi\src\com\lwz\GameCanvas.java

```
protected void paintChessboardDown(Graphics g) { //画下半部棋盘
    for (int q = 0; q < 4; q++) {
        for (int w = 0; w < 8; w++) {
            g.setColor(lineColor); //设置线的颜色
            g.drawRect(x + w * gridWidth, x + (q + 5) * gridWidth, gridWidth, gridWidth); //绘制棋盘
        }
    }
    g.setColor(lineColor);
    g.drawLine(x + 3 * gridWidth, x + 7 * gridWidth, x + 5 * gridWidth, x + 9 * gridWidth);
    g.drawLine(x + 5 * gridWidth, x + 7 * gridWidth, x + 3 * gridWidth, x + 9 * gridWidth);
    { //画左上方的炮
        g.drawLine(x + d, x + 6 * gridWidth + c, x + d, x + 6 * gridWidth + d); //左上竖
        g.drawLine(x + c, x + 6 * gridWidth + d, x + d, x + 6 * gridWidth + d); //左上横
        g.drawLine(x + d + 2 * b, x + 6 * gridWidth + c, x + d + 2 * b, x + 6 * gridWidth + d); //右上竖
        g.drawLine(x + gridWidth + b, x + 6 * gridWidth + d, x + gridWidth + a, x + 6 * gridWidth + d); //右上横
        g.drawLine(x + d, x + 7 * gridWidth + b, x + d, x + 7 * gridWidth + a); //左下竖
        g.drawLine(x + c, x + 6 * gridWidth + d + 2 * b, x + d, x + 6 * gridWidth + d + 2 * b); //左下横
        g.drawLine(x + d + 2 * b, x + 7 * gridWidth + b, x + d + 2 * b, x + 7 * gridWidth + a); //右下竖
        g.drawLine(x + gridWidth + b, x + 6 * gridWidth + d + 2 * b, x + gridWidth + a, x + 6 * gridWidth + d + 2 * b); //右下横
    }
    { //画右上方的炮
        g.drawLine(x + d + 6 * gridWidth, x + 6 * gridWidth + c, x + d + 6 * gridWidth, x + 6 * gridWidth + d); //绘制方法与左上方的炮相同
        g.drawLine(x + c + 6 * gridWidth, x + 6 * gridWidth + d, x + d + 6 * gridWidth, x + 6 * gridWidth + d);
    }
}
```



```

g.drawLine(x + d + 2 * b + 6 * gridWidth, x + 6 * gridWidth + c, x
+ d + 2 * b + 6 * gridWidth, x + 6 * gridWidth + d);
g.drawLine(x + gridWidth + b + 6 * gridWidth,
x + 6 * gridWidth + d, x + gridWidth + a + 6 * gridWidth, x
+ 6 * gridWidth + d);
g.drawLine(x + d + 6 * gridWidth, x + 7 * gridWidth + b, x + d + 6
* gridWidth, x + 7 * gridWidth + a);
g.drawLine(x + c + 6 * gridWidth, x + 6 * gridWidth + d + 2 * b, x
+ d + 6 * gridWidth, x + 6 * gridWidth + d + 2 * b);
g.drawLine(x + d + 2 * b + 6 * gridWidth, x + 7 * gridWidth + b, x
+ d + 2 * b + 6 * gridWidth, x + 7 * gridWidth + a);
g.drawLine(x + gridWidth + b + 6 * gridWidth, x + 6 * gridWidth + d
+ 2 * b, x + gridWidth + a + 6 * gridWidth, x + 6
* gridWidth + d + 2 * b);
}
}

```

(7) 编写 `paintAllChess()` 方法, 该方法用于绘制棋子和棋子上的汉字。在绘制棋子之前需要分别设置棋子和棋子上汉字的颜色, 并设置相应的字体样式。绘制棋子之后的界面效果如图 10.14 所示。

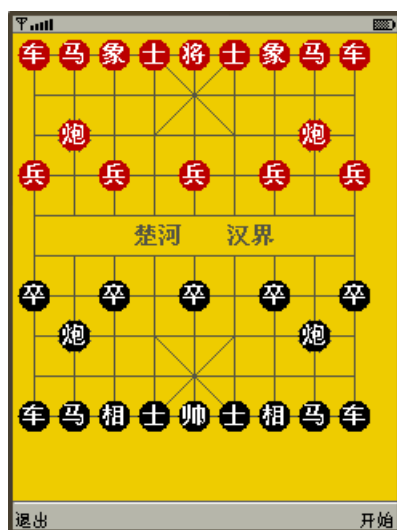


图 10.14 绘制棋子后的棋盘界面

关键代码如下:

例程 20 代码位置: 光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```

protected void paintAllChess(Graphics g) { //画出所有棋子
    if (point == null)
        return;
    for (i = 0; i < 10; i++) { //遍历棋盘数组
        for (j = 0; j < 9; j++) {
            if (point[i][j] != 0) {
                if (point[i][j] < 17) { //根据棋子的位置
                    g.setColor(redChess); //设置棋子是红色
                } else {

```

```

        g.setColor(blackChees);           //还是黑色
    }
    g.fillArc(x - chessR + j * gridWidth, x - chessR + i
              * gridWidth, 2 * chessR, 2 * chessR, 0, 360); //绘制棋子的圆
    g.setColor(charColor);                 //设置棋子汉字的颜色
    g.setFont(Font.getFont(Font.FACE_PROPORTIONAL,           //设置字体
                           Font.STYLE_BOLD, Font.SIZE_LARGE));
    g.drawString(chess[i][j], x + j * gridWidth, x + chessR + i //绘制棋子上的汉字
                 * gridWidth, Graphics.HCENTER | Graphics.BOTTOM);
    }
    }
}

```

代码贴士

- fillArc()：该方法用于绘制弧形的绘图操作，在本方法中用于绘制棋子的圆形。
- setColor()：该方法用于设置绘图对象的当前颜色。
- setFont()：该方法用于设置绘图对象的当前字体。
- drawString()：该方法用于根据指定颜色、字体和字符串在绘图面板上（棋盘上）绘制字符串。

(8) 编写 paintSelectedChess()方法，该方法用于绘制选择的棋子。当一个棋子被选中时，该方法将改变棋子的颜色和字体样式，使该棋子有被选中的动作。关键代码如下：

例程 21 代码位置：光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```

protected void paintSelectedChess(Graphics g) {           //画选择的棋子
    m = guard1;
    n = guard2; //再重新单独输出一个棋子
    g.setColor(chessSelColor);
    g.fillArc(x - chessR + guard1 * gridWidth, x - chessR + guard2
              * gridWidth, 2 * chessR, 2 * chessR, 0, 360);
    g.setColor(charColor);
    g.setFont(Font.getFont(Font.FACE_MONOSPACE, Font.STYLE_BOLD,
                           Font.SIZE_LARGE));
    g.drawString(chess[guard2][guard1], x + guard1 * gridWidth, x + chessR
                 + guard2 * gridWidth, Graphics.HCENTER | Graphics.BOTTOM);
}

```

(9) 编写 paintSelected()方法，该方法用于绘制选择棋子用的选择框。玩家可以通过上、下、左、右 4 个方向键改变选择框的位置。paintSelected()方法的关键代码如下：

例程 22 代码位置：光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```

protected void paintSelected(Graphics g) {               //画选择框
    g.setColor(selBorderColor);                          //设置选择框的颜色
    g.drawRect(x - chessR + selectedX * gridWidth, x - chessR + selectedY //绘制选择框
              * gridWidth, 2 * chessR, 2 * chessR);
}

```

(10) 重写父类的 paint()方法，该方法被手游主程序调用，用来绘制界面。之前定义的所有绘制

棋盘的方法都必须在该方法中运行，最后还需要调用 `whoTurn()` 方法通知该走棋的玩家继续进行游戏。关键代码如下：

例程 23 代码位置：光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```
protected void paint(Graphics g) {
    g.setColor(backColor);
    g.fillRect(0, 0, getWidth(), getHeight());
    paintChessboardUp(g);           //调用画上半部棋盘的方法
    paintRiver(g);                 //调用画河的方法
    paintChessboardDown(g);        //调用画下半部棋盘的方法
    paintAllChess(g);              //调用画棋子的方法
    if (guard % 2 == 1) {
        paintSelectedChess(g);     //调用画选择的棋子的方法
    }
    paintSelected(g);
    whoTurn(g);
}
```

10.6.4 按键处理模块实现过程

游戏的按键处理模块用于监听游戏动作，例如上、下、左、右 4 个方向的控制、棋子的选择、移动、游戏的开始、退出等都需要由手机按键来控制。实现手机按键处理模块的步骤如下：

(1) `GameCanvas` 类已经实现了 `CommandListener` 接口，它可以监听手机的按键信息，在该类中编写 `commandAction()` 方法，当玩家选择相应的软键按钮，程序就会调用该方法处理开始和退出等软键按钮的事件。关键代码如下：

例程 24 代码位置：光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```
public void commandAction(Command c, Displayable d) {           //软键按钮事件处理
    if (c == exitCmd) {                                         //如果选择了“退出”软键
        (new Thread() {
            public void run() {
                client.sendMessage("exitgame");                //向服务器发送退出信息
            }
        }).start();
        seatPos = -1;
        desknum = -1;
        Game.display.setCurrent(game.getPlayerList());          //显示桌面列表界面
    } else if (c == ok) {                                        //如果选择了 ok 软键
        reset();                                                  //重新设置游戏参数
        Game.display.setCurrent(this);                            //显示游戏界面
    } else if (c == start) {                                     //如果选择了“开始”软键
        banker = false;
        client.sendMessage("start");                             //向服务器发送开始信息
        this.removeCommand(start);                               //移除“开始”软键
    }
}
```

```

    }
}

```

(2) 重写父类的 `keyPressed()` 方法，在该方法中处理游戏的按键事件，例如上、下、左、右按键用于移动选择棋子的选择框。关键代码如下：

例程 25 代码位置：光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```

protected void keyPressed(int keyCode) {           //处理游戏按键
    int action = getGameAction(keyCode);           //获取按键代码
    if (myTurn) {                                   //判断是否轮到自己走棋
        if (action == Canvas.LEFT) {               //如果按下左键
            selectedX = (--selectedX + 8 + 1) % (8 + 1); //选择框的位置向左移动
        } else if (action == Canvas.RIGHT) {         //如果按下右键
            selectedX = (++selectedX) % (8 + 1);      //选择框的位置向右移动
        } else if (action == Canvas.UP) {            //如果按下上键
            selectedY = (--selectedY + 9 + 1) % (9 + 1); //选择框的位置向上移动
        } else if (action == Canvas.DOWN) {          //如果按下下键
            selectedY = (++selectedY) % (9 + 1);      //选择框的位置向下移动
        } else if (action == Canvas.FIRE) {          //如果按下 FIRE 键
            guard = guard + 1;                       //改变 FIRE 键的状态
            if (guard % 2 == 1) {                     //如果第一次按下 FIRE 键
                if (point[selectedY][selectedX] != 0) {
                    guard1 = selectedX;               //改变 guard1 和 guard2 的值
                    guard2 = selectedY;               //paintSelectedChess()方法会
                                                        //根据这两个变量绘制选棋
                }
            }
            if (guard % 2 == 0) {                     //如果是第二次按下 FIRE 键
                if (point[selectedY][selectedX] != point[n][m]) { //判断选择的棋子不是自身
                    if ((point[n][m] == 1) | (point[n][m] == 9)
                        | (point[n][m] == 17) | (point[n][m] == 25)) { //当选定的棋子是车的时候
                        if (point[selectedY][selectedX] == 0) {       //如果走棋位置是空的位置
                            theRuleOfChe(m, n, selectedX, selectedY); //执行车的走棋规则
                        } else {                                       //否则
                            if ((point[selectedY][selectedX] / 17) != (point[n][m] / 17)) {
                                theRuleOfChe(m, n, selectedX, selectedY); //吃掉目标棋子
                            }
                        }
                    }
                }
                if ((point[n][m] == 2) | (point[n][m] == 8)
                    | (point[n][m] == 18) | (point[n][m] == 24)) { //如果选定的棋子是马
                    ...//执行马的走棋规则
                }
                if ((point[n][m] == 10) | (point[n][m] == 11)
                    | (point[n][m] == 26) | (point[n][m] == 27)) { //如果选定的棋子是炮
                    ...//执行炮的走棋规则
                }
                if ((point[n][m] == 3) | (point[n][m] == 7)

```

```

        | (point[n][m] == 19) | (point[n][m] == 23)) {    //如果选定的棋子是相
        ...//执行相的走棋规则
    }
    if ((point[n][m] == 4) | (point[n][m] == 6)
        | (point[n][m] == 20) | (point[n][m] == 22)) {    //如果选定的棋子是士
        ...//执行士的走棋规则
    }
    if ((point[n][m] > 11 & point[n][m] < 17)) {    //如果选定的棋子是卒
        ...//执行兵或卒的走棋规则
    }
    }
    }
    }
    }
    repaint();    //重新绘制游戏界面
}

```

■ 代码贴士

- Canvas.LEFT：该常量对应于手机键盘的左方向键。
- Canvas.RIGHT：该常量对应于手机键盘的右方向键。
- Canvas.UP：该常量对应于手机键盘的上方向键。
- Canvas.DOWN：该常量对应于手机键盘的下方向键。
- Canvas.FIRE：该常量对应于手机键盘的选择键。

(3) 编写 changTwoChessNum()方法，该方法用于改变两个位置的棋子。当玩家移动自己的棋子或者吃掉对方棋子的时候会调用该方法，更新棋盘的界面。关键代码如下：

例程 26 代码位置：光盘\TM\10\xiangqi\src\com\lzw\GameCanvas.java

```

protected void changTwoChessNum(int m, int n, int selectedX, int selectedY,
    boolean send) {    //改变两个格子的值
    if (send) {    //判断是否需要发送信息
        if ((color.equals("red") && point[n][m] < 17)    //如果是红色棋子的玩家
            || (color.equals("white") && point[n][m] >= 17)) {
            if (!banker) {    //如果是先手
                client.sendMessage("move;" + seatPos + ":" + selectedY
                    + "," + selectedX + "," + n + "," + m);    //发送走棋信息
            } else {    //否则
                client.sendMessage("move;" + seatPos + ":"
                    + (9 - selectedY) + "," + (8 - selectedX) + ","
                    + (9 - n) + "," + (8 - m));    //发送非先手的走棋信息
            }
            myTurn = false;    //结束走棋回合
            p = point[selectedY][selectedX];    //调换两个棋子的位置
            point[selectedY][selectedX] = point[n][m];
            point[n][m] = 0;
            q = chess[selectedY][selectedX];    //将目标位置清空
        }
    }
}

```

```

        chess[selectedY][selectedX] = chess[n][m];
        chess[n][m] = "空";
    }
    } else {
        myTurn = false;
        p = point[selectedY][selectedX];
        point[selectedY][selectedX] = point[n][m];
        point[n][m] = 0;
        q = chess[selectedY][selectedX];
        chess[selectedY][selectedX] = chess[n][m];
        chess[n][m] = "空";
    }
}

```

10.6.5 游戏信息处理模块实现过程

为保持游戏的同步，游戏客户端和服务端必须频繁地通信并处理不同的游戏信息。例如一个玩家执行了走棋操作，那么其走棋信息必须通知服务器，服务器将记录棋盘的当前棋局，并通知其他玩家棋局的变更信息，这样其他的手机客户端也会知道该玩家的走棋信息。这样就完成了一个走棋的回合。这个回合的过程需要根据接收服务器的信息来判断和执行，这就需要编写相应的信息处理方法。

游戏界面中编写了 `receiveMessage()` 方法，该方法可以实现服务器信息的解析和业务处理。关键代码如下：

例程 27 代码位置：光盘\TM\10\xiangqi\src\com\lwz\GameCanvas.java

```

public void receiveMessage(String str) { //TODO Auto-generated method
    //stub
    if (str.startsWith("move")) {
        int index0 = str.indexOf(";");
        int index1 = str.indexOf(":");
        int index2 = str.indexOf(",", index1 + 1);
        int index3 = str.indexOf(",", index2 + 1);
        int index4 = str.indexOf(",", index3 + 1);
        int seat = Integer.parseInt(str.substring(index0 + 1, index1));
        int selectedY = Integer.parseInt(str.substring(index1 + 1, index2));
        int selectedX = Integer.parseInt(str.substring(index2 + 1, index3));
        int n = Integer.parseInt(str.substring(index3 + 1, index4));
        int m = Integer.parseInt(str.substring(index4 + 1));
        if (seat != seatPos) {
            if (banker) {
                changTwoChessNum(8 - m, 9 - n, 8 - selectedX,
                    9 - selectedY, false);
                myTurn = true;
            } else {
                changTwoChessNum(m, n, selectedX, selectedY, false);
                myTurn = true;
            }
        }
    }
}

```

```

        }
        repaint(); //重新绘制游戏界面
    }
} else if (str.startsWith("color")) {
    int index = str.indexOf(":");
    color = str.substring(index + 1);
    if (color.equals("white")) { //初始化白旗棋盘
        point = new int[][] { //初始化棋子数组
            { 1, 2, 3, 4, 5, 6, 7, 8, 9 },
            { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 10, 0, 0, 0, 0, 0, 11, 0 },
            { 12, 0, 13, 0, 14, 0, 15, 0, 16 },
            { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 28, 0, 29, 0, 30, 0, 31, 0, 32 },
            { 0, 26, 0, 0, 0, 0, 0, 27, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 17, 18, 19, 20, 21, 22, 23, 24, 25 } };
    } else if (color.equals("red")) { //初始化红旗棋盘
        point = new int[][] { //初始化棋子数组
            { 17, 18, 19, 20, 21, 22, 23, 24, 25 },
            { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 26, 0, 0, 0, 0, 0, 27, 0 },
            { 28, 0, 29, 0, 30, 0, 31, 0, 32 },
            { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 12, 0, 13, 0, 14, 0, 15, 0, 16 },
            { 0, 10, 0, 0, 0, 0, 0, 11, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            { 1, 2, 3, 4, 5, 6, 7, 8, 9 } };
    }
    repaint(); //重新绘制游戏界面
} else if (str.startsWith("turn")) { //解析回合信息
    myTurn = true; //获得走棋权利
    banker = true;
    repaint(); //重新绘制游戏界面
} else if (str.startsWith("win")) { //解析获胜信息
    checkWin(str); //处理获胜业务
} else if (str.startsWith("exitgame")) { //解析退出游戏信息
    game.initialize(); //初始化游戏数据
    Game.display.setCurrent(game.getPlayerList()); //显示桌面列表界面
}
}
}

```

10.6.6 客户端信息处理模块概述

客户端信息处理模块主要负责与服务器的连接、接收服务器传送的信息并调用相关方法、向服务器



发送信息等。因为 HTTP 协议属于无状态连接协议，所以服务器无法知道下一次请求是由谁发出的，因此需要在这里增加一个线程与服务器保持连接。服务器会通过客户端的 IP 地址和 Port 端口区分不同的玩家。服务器端有相应的 **JavaBean** 类用来保存每个客户的信息，在客户端发送下一次请求时，把数据发给客户端。这样就简单解决了 HTTP 无状态的难题。

游戏中所有的操作都用于更新棋盘的界面，例如走棋、吃子、开始和结束游戏等。这些操作都需要通过信息处理模块来通知服务器，服务器会根据操作判断游戏的进度、输赢状况并通知其他玩家关于本次操作的棋盘更新信息。

(1) 创建 **Client** 类，该类需要实现 **Runnable** 接口成为一个线程，并且在 **run()** 方法中定义无限循环。无限循环的线程和服务器保持联系，不断地接收和发送信息。在接收到服务器的信息时，需要根据信息更新游戏界面和处理相应的业务逻辑。关键代码如下：

例程 28 代码位置：光盘\TM\10\xiangqi\src\com\lzw\Client.java

```
public class Client implements Runnable {
    private Game midlet;                //游戏主程序的引用
    private HttpConnection dc;          //HTTP 连接对象
    private String port = "port=-1";    //端口变量
    public Client(Game mid) {           //在构造方法中初始化
        midlet = mid;
        sendMessage("register");
        System.out.println("Success register!");
        new Thread(this).start();       //并启动信息处理线程
    }
    String urlReceive;
    String urlSend;
    public void run() {
        while (true) {
            StringBuffer strbuf = new StringBuffer();
            urlReceive = "http://localhost:8080/XiangQiServer/XiangQi?message=hello&"
                + port;                  //定义接收服务器信息的 URL
            try {
                Thread.sleep(1000);     //使线程休眠 1 秒钟
                dc = (HttpConnection) Connector.open(urlReceive, Connector.READ);
                InputStream is = null;
                is = dc.openInputStream();
                int len = (int) dc.getLength();
                DataInputStream dis = new DataInputStream(is);
                if (len > 0) {
                    byte[] data = new byte[len];
                    dis.readFully(data);
                    for (int i = 0; i < data.length; i++) {
                        strbuf.append((char) data[i]);
                    }
                }
                String s = strbuf.toString().trim();
                if (s != null && !s.equals("")) {
                    if (s.startsWith("desks")) {
                        //接收服务器返回的信息
                    }
                }
            }
        }
    }
}
```



```

        midlet.setDesks(s); //处理桌面信息
    } else if (s.startsWith("takeseat")) {
        midlet.takeSeat(); //处理落座信息
    } else if (s.startsWith("updatedesk")) {
        midlet.updateDesk(s); //处理更新桌面的信息
    } else {
        if (midlet.getCanvas() != null) //由主程序处理其他信息
            midlet.getCanvas().receiveMessage(s);
    }
    ...//省略部分代码
}
} catch (Exception ex) {
    ex.printStackTrace();
}
}
}

```

(2) 编写 `sendMessage()` 方法, 该方法用于发送客户端的游戏信息。它是手机客户端和服务端通信的桥梁, 在按键处理模块中, 几乎所有操作都需要调用该方法向服务器发送信息, 例如游戏的开始和结束信息。`sendMessage()` 方法的关键代码如下:

例程 29 代码位置: 光盘\TM\10\xiangqi\src\com\lzw\Client.java

```

public void sendMessage(String message) {
    System.out.println("send message:" + message);
    try {
        urlSend = "http://localhost:8080/XiangQiServer/XiangQi?message=" //连接服务器的 URL, 其中包括
            + message + "&" + port; //将要发送的信息 message
        HttpURLConnection dc = (HttpURLConnection) Connector.open(urlSend,
            Connector.READ); //创建 HTTP 连接对象
        InputStream is = null;
        is = dc.openInputStream(); //获取输入流
        is.close();
        dc.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}
}

```

10.7 服务器模块设计

10.7.1 服务器模块概述

服务器模块负责所有玩家的游戏通信和业务处理, 例如 J2ME (手机客户端) 的消息保存、玩家登录、落座、退出、走棋等消息的处理; 根据请求消息调用 `Desk` 类对应的方法; 当客户端发送下一次请求



时将保存的消息返回给客户端；在玩家登录后向玩家发送房间已经更新的桌面信息等。

10.7.2 服务器模块技术分析

实际的服务器模块是一个 `HttpServlet` 类，它是一种服务器端的 Java Web 应用程序，可以动态地生成 Web 页面，处理客户端发送的 HTTP 请求，并返回该请求的相应信息。它可以处理 `doGet()`、`doPost()` 和 `server()` 等方法。这些方法的说明如表 10.2 所示。

表 10.2 `HttpServlet` 类定义的主要方法

名 称	说 明
<code>doGet()</code>	处理 HTTP GET 方法提交的请求，这种请求方法将参数暴露在 URL 中
<code>doPost()</code>	处理 HTTP POST 方法提交的请求，这种请求方法将参数隐藏
<code>doDelete()</code>	处理 HTTP DELETE 请求，该请求用于删除服务器端的文件
<code>doHead()</code>	处理 HTTP HEAD 请求，该请求用于从服务器请求数据
<code>doOption()</code>	处理 HTTP OPTION 请求，该请求将查询服务器支持的请求方法
<code>doPut()</code>	处理 HTTP PUT 请求，该请求将向服务器上传数据或者文件
<code>doTrace()</code>	处理 HTTP TRACE 请求，该请求用来调试 Web 程序

10.7.3 服务器模块实现过程

(1) 创建 `Server` 类，该类必须继承 `HttpServlet` 类，实现处理 HTTP 请求的能力，在类中定义玩家列表、游戏桌面数组、玩家计数器等属性，然后在 `init()` 方法中初始化它们。关键代码如下：

例程 30 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```
public class Server extends HttpServlet {
    private Hashtable players = new Hashtable();           //玩家列表
    private int NUM = 2;                                   //每桌的玩家数量
    private int DESKNUM = 10;                              //桌子数量
    private Desk[] desks = new Desk[DESKNUM];             //桌面数组
    private int counter = 1;                               //玩家计数器
    private Player player;                                 //玩家对象
    PrintWriter out;                                       //输出流
    /*
    * 初始化 对 players(Hashtable)、desks(所有桌子)初始化
    */
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        players.clear();                                   //清除玩家的消息队列
        for (int i = 0; i < DESKNUM; i++) {                //初始化所有桌面对象
            desks[i] = new Desk();
            desks[i].setID(i);
        }
    }
}
```

```

    }
}
...//省略部分代码
}

```

(2) 重写父类的 `doGet()` 方法, 该方法可以接收客户端发送的 HTTP 请求。在该方法中分别解析不同的请求信息, 并调用相应的业务方法处理客户请求。关键代码如下:

例程 31 代码位置: 光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```

/*
 * 接受信息主方法 接受各种信息, 然后调用相应的处理方法
 */
public void doGet(HttpServletRequest request, HttpServletResponse response) //接受信息
    throws ServletException, IOException {
    out = response.getWriter();
    String message = request.getParameter("message");
    String port = request.getParameter("port");
    port = port.trim();
    if (!message.startsWith("hello"))
        System.out.println(message);
    if (message.startsWith("register")) { //登录信息
        denglu(request); //调用 denglu()方法
        return;
    }
    if (message.startsWith("take")) { //落座信息
        luozuo(message, player); //调用 luozuo()方法
        return;
    } else if (message.startsWith("start")) { //开始信息
        tryStart(player); //调用 tryStart()方法
        return;
    } else if (message.startsWith("move")) { //走棋信息
        Desk desk = player.getDesk();
        desk.moveChess(message); //调用 moveChess()方法
    } else if (message.startsWith("hello")) { //读取信息
        if (!player.data.empty()) //如果该玩家的消息队列非空
            out.println((String) player.data.pop()); //发送消息队列最顶端的信息
        else //否则
            out.println("noData"); //通知客户端没有消息
        return;
    } else if (message.startsWith("exitgame")) { //退出信息
        tryExit(players, player, request); //调用 tryExit()方法
        return;
    } else if (message.startsWith("exit")) { //退出信息
        tryExit(players, player, request); //调用 tryExit()方法
        return;
    }
}
}

```

■ 代码贴士

- denglu(): 该方法用于处理玩家登录的业务逻辑。
- luozuo(): 该方法用于处理玩家落座的业务逻辑。
- tryStart(): 该方法用于处理玩家开始信息的业务逻辑。
- moveChess(): 该方法用于记录玩家的走棋信息。
- out.println(): 该方法用于发送信息。该方法在信息发送方法中也是这样调用的。
- tryExit(): 该方法用于处理玩家退出游戏的业务逻辑。

(3) 编写 sendMessage()方法, 该方法用于发送信息到指定玩家的客户端, 也就是手机上。这个方法将被其他方法甚至整个服务器模块调用, 向客户端发送不同的信息。关键代码如下:

例程 32 代码位置: 光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```
public void sendMessage(Player p, String str) {           //发送信息, 等待用户读取
    p.data.push(new String(str));                         //将信息压入玩家的消息队列
}
```

(4) 编写 updateClientsDesk()方法, 该方法在服务器接收到玩家落座时被调用, 它将更新整个主程序的桌面列表。关键代码如下:

例程 33 代码位置: 光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```
public void updateClientsDesk(int deskid) {               //更新所有客户桌面
    for (Enumeration en = players.elements(); en.hasMoreElements();) { //遍历玩家列表
        Player player = (Player) en.nextElement();
        if (player != null)                                //如果玩家对象非空
            updateDesk(player, deskid);                   //更新该玩家所在的桌面信息
    }
}
```

(5) 编写 updateDesk()方法, 该方法用于更新单个桌面的座位信息。它根据不同玩家和桌号参数更新指定桌面的座位信息。关键代码如下:

例程 34 代码位置: 光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```
public void updateDesk(Player isa, int deskid) {          //更新单个桌面
    String message = "updatedesk," + deskid;             //定义更新桌面的信息
    String str = "";
    for (int i = 0; i < desks[deskid].getPlayersCounter(); i++) { //遍历每个桌面上的玩家对象
        if (i == 0) {                                     //如果是 1 号桌面
            if (desks[deskid].isEmpty(i))                 //如果该桌面为空
                str = "0";                                 //以 0 填充
            else                                            //否则
                str = "1";                                 //以 1 填充
        } else {                                          //如果是其他桌面
            if (desks[deskid].isEmpty(i))
                str = "0";
            else
                str = "1";
        }
    }
}
```

```

        str = str + ",0";
    else
        str = str + ",1";
    }
}
message = message + ":" + str;
sendMessage(isa, message);
}

```

//在原信息基础上添加座位信息
//发送信息到玩家消息队列

(6) 编写 `sendDeskList()` 方法, 该方法在玩家登录服务器时被 `denglu()` 方法调用, 用于发送服务器的整个桌面列表。当手机客户端的游戏刚刚启动时, 桌面列表是空的, 所以需要获取所有的桌面信息。在以后变更桌面列表时, 只执行更新方法就可以了。 `sendDeskList()` 方法的关键代码如下:

例程 35 代码位置: 光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```

public void sendDeskList(Player player) {
    String message = "desks," + DESKNUM;
    for (int i = 0; i < DESKNUM; i++) {
        String str = "";
        for (int j = 0; j < desks[i].getPlayersCounter(); j++) {
            if (j == 0) {
                if (desks[i].isEmpty(j))
                    str = "0";
                else
                    str = "1";
            } else {
                if (desks[i].isEmpty(j))
                    str = str + ",0";
                else
                    str = str + ",1";
            }
        }
        message = message + ":" + str;
    }
    sendMessage(player, message);
}

```

//获得桌面列表
//定义桌面列表信息
//遍历所有桌面
//遍历所有玩家对象
//判断桌面是否为空
//生成新的信息字符串
//追加信息字符串
//发送信息到登录玩家

(7) 编写 `denglu()` 方法, 该方法用于注册玩家到服务器, 注册成功的玩家将被存入玩家列表中。当服务器接收到 `register` 信息时, 将会解析玩家的 IP 地址和 Port 端口号, 根据玩家的信息生成 `player` 对象并存入玩家列表中, 同时调用 `sendDeskList()` 方法发送桌面列表到手机客户端。关键代码如下:

例程 36 代码位置: 光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```

public void denglu(HttpServletRequest request) {
    Player player = new Player(request.getRemoteAddr(), request
        .getRemotePort());
    player.setID(counter);
    counter++;
    players.put(request.getRemoteAddr() + ":" + request.getRemotePort(),

```

//处理登录信息
//根据玩家 IP 和端口创建玩家对象
//设置玩家编号
//玩家数量累加

```

        player);                                //将玩家对象存入玩家列表
        sendMessage(player, "port=" + request.getRemotePort()); //发送玩家端口信息
        sendDeskList(player);                    //发送桌面列表信息
    }

```

(8) 编写 `luozuo()` 方法，它在服务器接收到手机客户端的 `take` 信息时被调用，主要用于处理玩家的落座信息。该方法解析落座信息之后，将玩家安排到指定的桌面，然后更新桌面，并通知玩家落座的位置。关键代码如下：

例程 37 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```

public void luozuo(String message, Player player) { //处理落座信息
    try {
        int index1 = message.indexOf(","); //解析落座信息
        int index2 = message.indexOf(",", index1 + 1);
        int dindex = Integer.parseInt(message.substring(index1 + 1, index2));
        int pindex = Integer.parseInt(message.substring(index2 + 1));
        if (dindex < DESKNUM && dindex >= 0) { //判断解析的桌号是否超出范围
            if (desks[dindex].isEmpty(pindex)) { //如果该桌号有空位置
                desks[dindex].setPlayer(pindex, player); //将玩家安排到该桌号
                player.setDesk(desks[dindex]);
                sendMessage(player, "takeseat"); //发送信息通知玩家落座位置
                updateClientsDesk(dindex); //更新桌面
            }
        }
    } catch (Exception exc) {
    }
}

```

(9) 编写 `tryStart()` 方法，它在服务器接收到 `start` 信息时将调用，主要用于处理游戏的开始信息。该方法将判断桌面上的所有玩家是否都发送了开始信息，以此决定是否开始游戏。关键代码如下：

例程 38 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```

public void tryStart(Player player) { //处理开始信息
    player.start();
    Desk d1 = player.getDesk(); //获取玩家所在的桌面
    if (d1.isReady()) { //判断桌面是否准备好
        d1.start(); //开始该桌面的游戏
    }
}

```

(10) 编写 `tryExit()` 方法处理退出游戏时发送的信息，该方法在服务器接收到 `exit` 信息时被调用。它将从桌面中移除该玩家，同时在玩家列表中也删除该玩家对象。关键代码如下：

例程 39 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```

public void tryExit(Hashtable players, Player player,
    HttpServletRequest request) {
    Desk de = player.getDesk();
    player.init();
}

```

```

de.removePlayer(player);
player=null;
players.remove(request.getRemoteHost() + ":" + request.getRemotePort());
updateClientsDesk(de.getID());
}

```

10.7.4 单元测试

服务器模块主要以信息处理为核心，当服务器接收到不同的信息时，必须解析相应的信息并处理该信息所要执行的业务逻辑，然后将执行结果发送回客户端。在这个信息处理的流程中，对单个信息进行调试的时候，经常会出现其他信息处理的错误，导致程序开发人员无法判断错误位置是其他信息处理模块，还是当前调试的信息处理模块。

为解决此类调试问题，可以在 `Server` 类的 `doGet()` 方法中输出服务器所接收的信息，例如注册、登录、走棋等信息。

在手机客户端所发送的众多信息中，`hello` 信息是保持通信的信息，它在无限循环的线程中不停地询问服务器是否有该客户端的信息。手机客户端每秒都会发送一次 `hello` 信息，而该信息对于调试没有多大用处，所以在输出服务器接收的信息时，要限制不输出 `hello` 信息，以保持调试信息的整洁，从而方便程序员对调试信息的分析。关键代码如下：

例程 40 代码位置：光盘\TM\10\XiangQiServer\src\com\lzw\Server.java

```

public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    out = response.getWriter();
    String message = request.getParameter("message");           //获取信息参数值
    String port = request.getParameter("port");                 //获取端口信息
    port = port.trim();
    if (!message.startsWith("hello"))                           //如果不是 hello 信息
        System.out.println(message);                           //输出该信息到控制台
    ...//省略部分代码
}

```

经过测试，进行象棋游戏的双方从注册到开始游戏，服务器所接收的信息如图 10.15 所示。

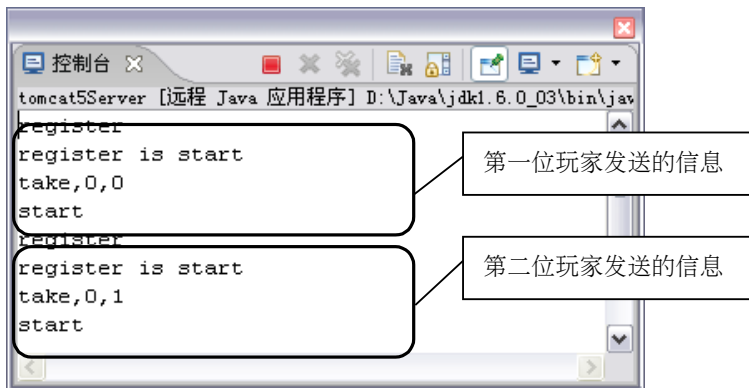


图 10.15 服务器在控制台输出的调试信息

10.8 发布与运行

在完成所有代码的编写以后，接下来就是程序的发布与运行。下面分别介绍服务器端与客户端工程的发布和运行。

10.8.1 服务器端的发布与运行

服务器模块是一个 Web 工程，它不像应用程序那样直接运行，而需要发布到 Tomcat 服务器中。启动 Tomcat 服务器后，Tomcat 服务器再加载服务器应用。然后客户端运行时，会访问服务器资源。下面介绍 Eclipse 中 Web 工程的发布和运行。

1. 服务器的发布

服务器模块编写结束之后，需要发布到 Tomcat 服务器中才能得到应用。下面简单介绍一下工程的发布步骤。

(1) 在工具栏上单击  按钮，在弹出的 Project Deployments 对话框中的 Project 下拉列表框中选择服务器工程 XiangQiServer 选项，然后单击 Add 按钮，如图 10.16 所示。

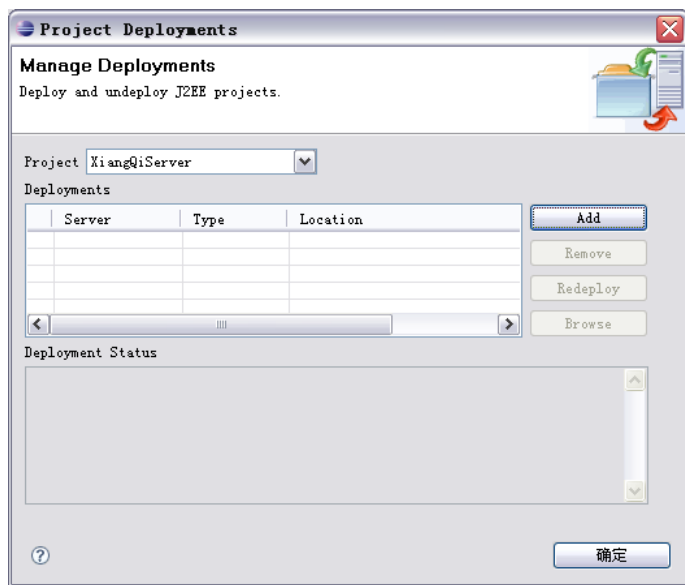


图 10.16 工程发布窗口

(2) 在弹出的 New Deployment 对话框中，在 Server 下拉列表框中选择 Tomcat5 选项，单击“完成”按钮，如图 10.17 所示。

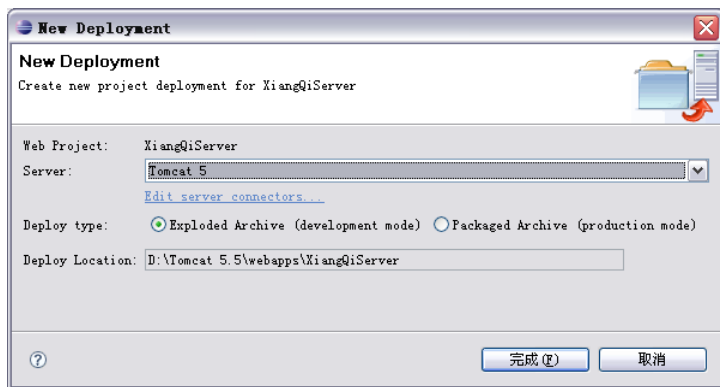


图 10.17 建立发布选项

(3) 在 Project Deployments 对话框的 Deployments 区域中会有一个发布项, 如图 10.18 所示。单击 Redeploy 按钮可以重新发布。到此服务器的工程发布结束。

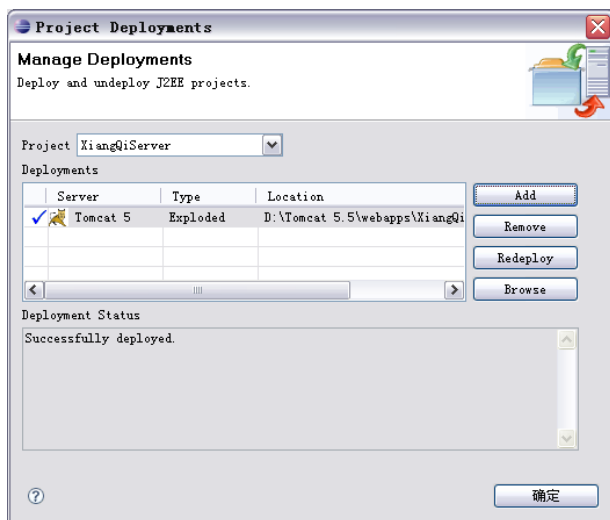


图 10.18 重新发布工程

2. 服务器的运行



在工具栏上单击  按钮右侧的下拉箭头, 选择 Tomcat 5/Start 命令, 运行服务器。服务器在启动过程中会在控制台输出调试信息, 如图 10.19 所示。



图 10.19 Tomcat 在控制器的启动信息

10.8.2 客户端的运行


服务器启动之后，就可以运行客户端来测试游戏了。客户端的设置比较简单。客户端的运行步骤如下：

(1) 单击工具栏上的  按钮右侧的下拉箭头，在弹出的菜单中选择 **Run** 命令。

(2) 在弹出的“运行”窗口中 **Wireless Toolkit Emulator** 节点上单击鼠标右键，选择“新建”命令；在窗口右侧的“名称”文本框中输入运行项的名称，如 **Game**；在 **Project** 文本框中输入或单击右侧的 **Browse** 按钮，选择“xiangqi”项目；在 **Executable** 区域中选中 **Midlet** 单选按钮并在其后的文本框中输入游戏的主程序“com.lzw.Game”，如图 10.20 所示。单击“应用”按钮。



图 10.20 “运行”窗口

(3) 最后单击工具栏上的  按钮运行程序，依次运行 2 个客户端后，便可进行游戏的测试。

10.9 开发技巧与难点分析

在开发手机网络游戏项目的过程中，对使用 HTTP 通信协议实现游戏通信出现过开发难点。该难点是因为 HTTP 协议是无状态的，每次客户端对服务器发送请求，服务器对该请求应答之后，就无法再与客户端通信，因为这个连接已经关闭了，所以服务器无法知道下一次请求是由谁发出的。

解决该问题的办法是使客户端添加一个信息发送的线程，该线程会无限循环，不停地向服务器发送请求，以维持与服务器的连接，而服务器端使用相应的消息队列保存每个玩家的信息，在客户端发送下一次请求时，把数据发给客户端，这样就简单解决了 HTTP 无状态的难题。客户端的关键代码如下：

例程 41 代码位置：光盘\TM\10\xiangqi\src\com\lzw\Client.java

```
public class Client implements Runnable {  
    ...//省略部分代码
```

```

public void run() {
    while (true) {
        StringBuffer strbuf = new StringBuffer();
        urlReceive = "http://localhost:8080/XiangQiServer/XiangQi?message=hello&"
            + port; //在请求中使用 hello 信息
        try { //与服务器保持通信
            Thread.sleep(1000); //使线程休眠 1 秒钟
            dc = (HttpConnection) Connector.open(urlReceive, Connector.READ); //访问服务器
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
}

```

10.10 使用 EclipseMe 工具编写手机程序

本章使用 EclipseMe 插件开发了手机游戏的客户端，EclipseMe 是 Eclipse 的插件，它扩展了 Eclipse，提供了开发、调试、运行和发布手机应用程序的能力。下面介绍一下 EclipseMe 插件的安装和使用该插件开发手机程序的步骤：

10.10.1 EclipseMe 的安装

EclipseMe 是开发 J2ME 程序的免费 Eclipse 插件，可到 <http://eclipseme.org/> 下载 zip 包。安装步骤如下：

(1) 在 Eclipse 中选择“帮助”，“软件更新”/“查找并安装”命令，在弹出的“安装/更新”对话框中选中“搜索要安装的新功能部件”单选按钮，如图 10.21 所示。单击“下一步”按钮。

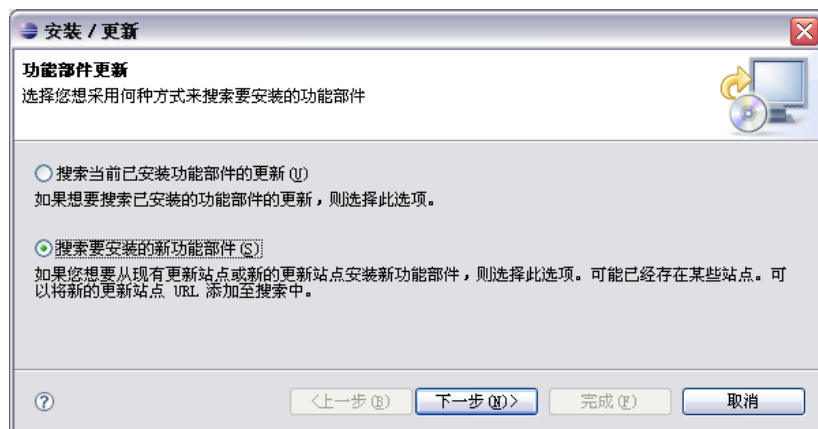


图 10.21 “安装/更新”对话框

(2) 在弹出的“安装”对话框中，单击“新建已归档的站点”按钮，在弹出的对话框中选择下载的 EclipseMe 的 zip 安装包，单击“打开”按钮，“安装”对话框中会添加 EclipseMe 的安装复选框，如图 10.22 所示。单击“完成”按钮，根据提示进行安装即可。

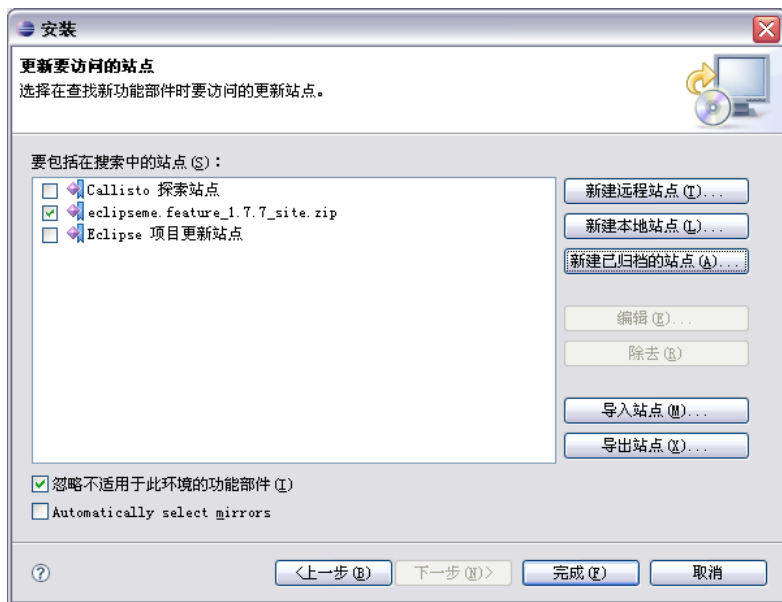


图 10.22 “安装”对话框

10.10.2 配置 EclipseMe 插件

安装了 EclipseMe 插件之后，必须为其配置驱动管理，也就是为插件添加手机虚拟设备。具体步骤如下：

(1) 选择“窗口”/“首选项”命令，在弹出的“首选项”对话框中，依次选择 J2ME/Device Management 节点，如图 10.23 所示。单击右侧的 Import 按钮。

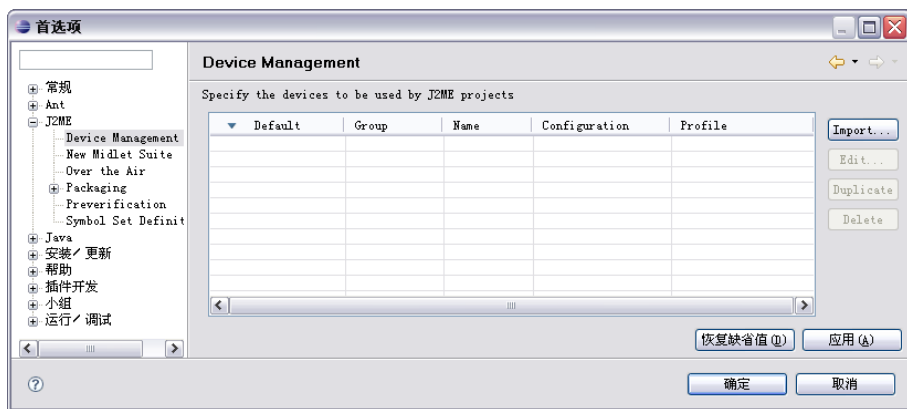


图 10.23 “首选项”对话框

(2) 在弹出的 Import Devices 对话框中, 单击 Browse 按钮, 选择 J2ME 的安装文件夹, 例如 “D:\Java\WTK2.5.2”, 然后单击 Refresh 按钮, 在 Devices 表格中会出现多个驱动的复选框, 选择所有驱动, 如图 10.24 所示。单击“完成”按钮。

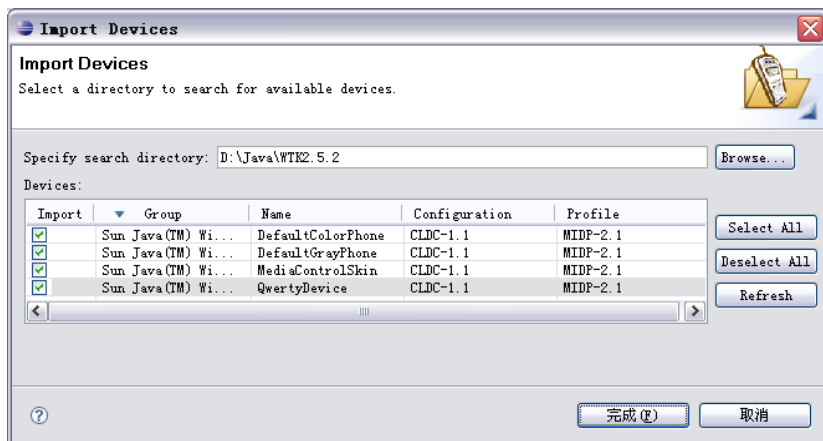


图 10.24 Import Devices 对话框

(3) 返回图 10.23 所示的“首选项”对话框, 在设备驱动的表格中选择 name 列为 DefaultColorPhone 的复选框, 单击“确定”按钮。这样默认驱动设备就是彩色屏幕的虚拟手机了。

10.10.3 创建 J2ME 项目

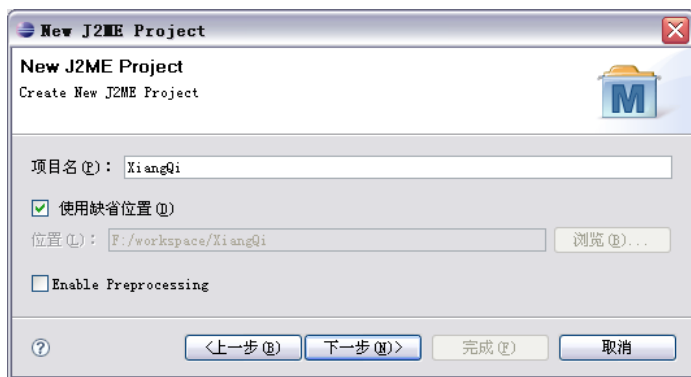
使用 EclipseMe 编写手机应用程序, 必须先创建 J2ME 项目, 程序源代码的编写、编译、程序资源管理等都存放在 J2ME 项目中。下面将介绍如何创建 J2ME 项目。

(1) 启动 Eclipse, 选择“文件”/“新建”/“项目”命令, 在弹出的“新建项目”对话框中依次选择 J2ME/J2ME Midlet Suite 节点, 如图 10.25 所示。单击“下一步”按钮。



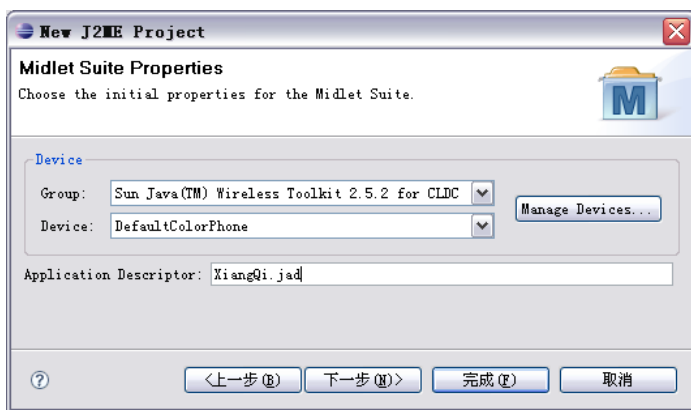
图 10.25 “新建项目”对话框

(2) 在弹出的 New J2ME Project 对话框的“项目名”文本框中输入新建 J2ME 项目的名称，例如“XiangQi”，如图 10.26 所示。单击“下一步”按钮。



10.26 New J2ME Project 对话框 (1)

(3) 在弹出的对话框中采用默认设置，其中 Device 下拉列表框中显示的是之前设置的默认虚拟设备，如图 10.27 所示。单击“完成”按钮。



10.27 New J2ME Project 对话框 (2)

现在读者可以编写自己的 J2ME 应用程序或者手机游戏了。

10.11 本章小结

本章运用软件工程的设计思想，通过一个完整的手机网络游戏为读者详细讲解了一个 J2ME 项目的开发流程。通过本章的学习，读者可以了解 J2ME 移动应用程序的开发流程、Canvas 绘图面板的使用、事件监听等技术。另外，本章还介绍了如何利用 EclipseMe 工具开发 J2ME 项目，希望对读者日后的程序开发有所帮助。

