第5章

源码下载

推箱子游戏

5.1 推箱子游戏介绍

经典的推箱子游戏是一个来自日本的古老游戏，目的是训练玩家的逻辑思维能力。在一个狭小的仓库中，要求把箱子放到指定的位置，但一不小心就会出现箱子无法移动或者通道被堵住的情况，所以需要巧妙地利用有限的空间和通道合理地安排移动的次序和位置，这样才能顺利地完成任务。

推箱子游戏的规则如下：

游戏运行载入相应的地图，屏幕中出现一个推箱子的工人，其周围是围墙、人可以走的通道、几个可以移动的箱子和箱子放置的目的地。玩家通过按上、下、左、右键控制工人推箱子，当把箱子都推到了目的地之后出现过关信息，并显示下一关。如果推错，玩家可以右击撤销上次的移动，还可以按空格键重新玩这一关，直到过完全部关卡。

本章开发推箱子游戏，该游戏的效果如图 5-1 所示。

图 5-1 推箱子游戏界面

游戏中的图片资源如图 5-2 所示。

图 5-2 推箱子游戏的图片资源

其中，pic1 为墙；pic2 为箱子；pic3 为在目的地的箱子；pic4 为目的地；pic5 为向下的人；pic6 为向左的人；pic7 为向右的人；pic8 为向上的人；pic9 为通道；pic10 为站在目的地向下的人；pic11 为站在目的地向左的人；pic12 为站在目的地向右的人；pic13 为站在目的地向上的人。

5.2 程序设计的思路

首先确定开发难点。对工人的操作很简单，就是向 4 个方向移动，工人移动，箱子也移动，所以按键的处理也比较简单。当箱子到达目的地时，就会产生游戏过关事件，需要逻辑判断。仔细想一下，所有事件都发生在一张地图中，这张地图包括箱子的初始位置、箱子最终放置的位置和围墙障碍等。每一关地图都要更换，这些位置也要改变，所以每一关的地图数据是最关键的，它决定了每一关的不同场景和物体的位置。下面重点分析一下地图。

可以把地图想象成一个网格，每个格子就是工人每次移动的步长(这里为 30 像素)，也是箱子移动的距离，这样问题就简化多了。首先设计一个 mapRow×mapColumn 的二维数组 map，按照这样的框架来思考。对于格子的两个屏幕像素坐标(x、y)，可以由二维数组下标(i、j)换算。

换算公式为 leftX ＋ j×30，leftY ＋ i×30

每个格子的状态值分别用枚举类型值：

```
//定义一些常量,对应地图的元素
final byte WALL = 1,BOX = 2,BOXONEND = 3,END = 4,MANDOWN = 5,
MANLEFT = 6,MANRIGHT = 7,MANUP = 8,GRASS = 9,
MANDOWNONEND = 10,MANLEFTONEND = 11,
MANRIGHTONEND = 12,MANUPONEND = 13;
```

其中，WALL(1)代表墙，BOX(2)代表箱子，BOXONEND(3)代表放到目的地的箱子，END(4)代表目的地，MANDOWN(5)代表向下的人，MANLEFT(6)代表向左的人，MANRIGHT(7)代表向右的人，MANUP(8)代表向上的人，GRASS(9)代表通道，MANDOWNONEND(10)代表站在目的地向下的人，MANLEFTONEND(11)代表站在目的地向左的人，MANRIGHTONEND(12)代表站在目的地向右的人，MANUPONEND(13)代表站在目的地向上的人。

原始地图中格子的状态值数组采用相应的整数形式存储。

在玩家通过键盘控制工人推箱子的过程中，需要按游戏规则判断是否响应该按键指示。下面分析工人将会遇到什么情况，以便归纳出所有的规则和对应算法。为了描述方便，假设工人移动趋势方向为向右，其他方向的原理与之相同。p1、p2 分别代表工人移动趋势方向前的两个方格，如图 5-3 所示。

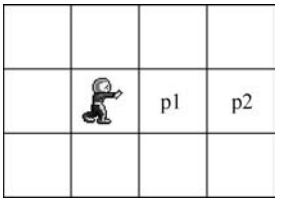

图 5-3　初始位置

1. 前方 p1 是围墙

```
如果工人前方是围墙(即阻挡工人的路线)
{
    退出规则判断,布局不做任何改变
}
```

2. 前方 p1 是通道(GRASS)或目的地(END)

```
如果工人前方是通道或目的地
{
    工人可以进到 p1 方格,修改相关位置格子的状态值
}
```

3. 前方 p1 是箱子

在前面的两种情况中，只要根据前方 p1 处的物体就可以判断出工人是否可以移动，而在第 3 种情况中(如图 5-4 所示)，需要根据箱子前方 p2 处的物体才能判断出工人是否可以移动。此时有以下几种可能：

(1) p1 处为箱子(BOX)或者放到目的地的箱子(BOXONEND)，p2 处为通道(GRASS)；工人可以进到 p1 方格；p2 方格的状态为箱子。修改相关位置格子的状态值。

(2) p1 处为箱子(BOX)或者放到目的地的箱子(BOXONEND)，p2 处为目的地(END)；工人可以进到 p1 方格；p2 方格的状态为放置好的箱子。修改相关位置格子的状态值。

(3) p1 处为箱子(BOX)，p2 处为墙(WALL)。退出规则判断，布局不做任何改变。

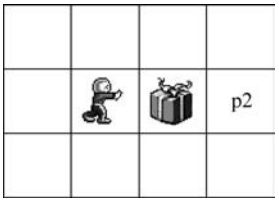

图 5-4　前方 p1 是箱子

综合前面的分析，可以设计出整个游戏的实现流程。

整个游戏的源文件说明如下。

- GameFrame.java：游戏界面视图。

- Map.java：封装游戏当前状态。
- MapFactory.java：提供地图数据。

5.3 程序设计的步骤

5.3.1 设计地图数据类

地图数据类(MapFactory)保存所有关卡的原始地图数据，每一关的数据为一个二维数组，所以此处 map 是三维数组。

```
import java.io.InputStream;
public class MapFactory {
      static byte map[][][] = {
              {
              { 0,0,1,1,1,0,0,0 },
              { 0,0,1,4,1,0,0,0 },
              { 0,0,1,9,1,1,1,1 },
              { 1,1,1,2,9,2,4,1 },
              { 1,4,9,2,5,1,1,1 },
              { 1,1,1,1,2,1,0,0 },
              { 0,0,0,1,4,1,0,0 },
              { 0,0,0,1,1,1,0,0 }
              },
              {
              { 1,1,1,1,1,0,0,0,0 },
              { 1,9,9,5,1,0,0,0,0 },
              { 1,9,2,2,1,0,1,1,1 },
              { 1,9,2,9,1,0,1,4,1 },
              { 1,1,1,9,1,1,1,4,1 },
              { 0,1,1,9,9,9,9,4,1 },
              { 0,1,9,9,9,1,9,9,1 },
              { 0,1,9,9,9,1,1,1,1 },
              { 0,1,1,1,1,1,0,0,0 }
              },
             …//省略其余关卡数据
    };
    static int count = map.length;
    public static byte[][] getMap(int grade)
    {byte temp[][];
        if(grade >= 0 && grade < count)
            temp = map[grade];
        else
            temp = map[0];
        int row = temp.length;
        int column = temp[0].length;
        byte[][] result = new byte[row][column];
        for(int i = 0;i < row;i++)
            for(int j = 0;j < column;j++)
                result[i][j] = temp[i][j];
```

```
            return result;
        }
        public static int getCount()
        {
            return count;
        }
    }
```

5.3.2　设计地图类

由于每移动一步，都需要保存当前的游戏状态，所以此处定义地图类（Map），保存人的位置和游戏地图的当前状态。若撤销移动，恢复地图时通过该类获取人的位置、地图的当前状态和关卡数。

```
public class Map {
    int manX = 0;
    int manY = 0;
    byte map[][];
    int grade;
    //此构造方法用于撤销操作
    //撤销操作只需要人的位置和地图的当前状态
    public Map(int manX,int manY,byte[][]map)
    {
        this.manX = manX;
        this.manY = manY;
        int row = map.length;
        int column = map[0].length;
        byte temp[][] = new byte[row][column];
        for(int i = 0;i < row;i++)
            for(int j = 0;j < column;j++)
                temp[i][j] = map[i][j];
        this.map = temp;
    }

    //此构造方法用于保存操作
    //恢复地图时需要人的位置、地图的当前状态和关卡数(关卡切换时此为基数)
    public Map(int manX,int manY,byte[][] map,int grade)
    {
        this(manX,manY,map);
        this.grade = grade;
    }

    public int getManX() {
        return manX;
    }
    public int getManY() {
        return manY;
    }
    public byte[][] getMap() {
        return map;
```

```
    }
    public int getGrade() {
        return grade;
    }
}
```

5.3.3 设计游戏面板类

游戏面板类(GameFrame)完成游戏界面的刷新显示以及相应的鼠标、键盘事件。

```
//推箱子游戏带音乐版
//右击——悔棋功能
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.util.ArrayList;
import javax.sound.midi.*;
import javax.swing.* ;
public class GameFrame extends JFrame implements ActionListener,MouseListener,KeyListener{
    //主面板类
    private int grade = 0;
    //row、column 记录人的行号、列号
    //leftX、leftY 记录左上角图片的位置,避免图片从(0,0)坐标开始
    private int row = 7,column = 7,leftX = 0,leftY = 0;
    //记录地图的行/列数
    private int mapRow = 0,mapColumn = 0;
    //width、height 记录屏幕的大小
    private int width = 0,height = 0;
    private boolean acceptKey = true;
    //程序所用到的图片
    private Image pic[] = null;
    private byte[][] map = null;
    private ArrayList list = new ArrayList();
    Sound sound;
```

关于格子状态值的常量对应地图的元素。

```
final byte WALL = 1,BOX = 2,BOXONEND = 3,END = 4,MANDOWN = 5,
MANLEFT = 6,MANRIGHT = 7,MANUP = 8,GRASS = 9,
MANDOWNONEND = 10,MANLEFTONEND = 11,
MANRIGHTONEND = 12,MANUPONEND = 13;
```

在构造方法 GameFrame()中,调用 initMap()初始化本关 grade 游戏地图,清空悔步信息列表 list,同时播放 MIDI 游戏背景音乐。

```
public GameFrame() {
    super("推箱子游戏带音乐版");
    setSize(600,600);
    setVisible(true);
```

```
    setResizable(false);
    setLocation(300,20);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container cont = getContentPane();
    cont.setLayout(null);
    cont.setBackground(Color.black);
    //初始 13 张图片
    getPic();
    width = this.getWidth();
    height = this.getHeight();
    this.setFocusable(true);
    initMap();                    //初始化本关 grade 游戏地图,清空悔步信息列表 list
    this.addKeyListener(this);
    this.addMouseListener(this);
    sound = new Sound();
    sound.loadSound();            //播放 MIDI 游戏背景音乐
}
```

在 initMap()方法中调用 getMapSizeAndPosition()和 getManPosition()。

```
public void initMap() {
    map = getMap(grade);
    list.clear();
    getMapSizeAndPosition();
    getManPosition();
}
```

getManPosition()获取工人当前位置(row,column):

```
public void getManPosition() {
    for (int i = 0; i < map.length; i++)
        for (int j = 0; j < map[0].length; j++)
            if (map[i][j] == MANDOWN || map[i][j] == MANDOWNONEND
            || map[i][j] == MANUP || map[i][j] == MANUPONEND
            || map[i][j] == MANLEFT || map[i][j] == MANLEFTONEND
            || map[i][j] == MANRIGHT || map[i][j] == MANRIGHTONEND)
            {
                row = i;
                column = j;
                break;
            }
}
```

getMapSizeAndPosition()获取游戏区域大小及显示游戏的左上角位置(leftX,leftY):

```
private void getMapSizeAndPosition() {
    //TODO Auto - generated method stub
    mapRow = map.length;
    mapColumn = map[0].length;
    leftX = (width - map[0].length * 30) / 2;
    leftY = (height - map.length * 30) / 2;
```

```
        System.out.println(leftX);
        System.out.println(leftY);
        System.out.println(mapRow);
        System.out.println(mapColumn);
    }
```

getPic()加载要显示的图片：

```
    public void getPic() {
        pic = new Image[14];
        for(int i = 0; i < 13; i++)
        {
            pic[i] = Toolkit.getDefaultToolkit().getImage("images\\pic" + i + ".jpg");
        }
    }
```

grassOrEnd(byte man)判断人所在位置是通道 GRASS 还是目的地 END：

```
    public byte grassOrEnd(byte man) {
        byte result = GRASS;
        if (man == MANDOWNONEND || man == MANLEFTONEND || man == MANRIGHTONEND || man == MANUPONEND)
            result = END;
        return result;
    }
```

游戏逻辑主要有人物移动。人物可以向 4 个方向移动，这里以向上移动作为例子(向下、向左、向右移动类似)进行介绍。推箱子游戏的特殊性决定了人物移动涉及 3 个位置，即人当前位置、人将要移动到的位置和箱子要到达的位置。

向上移动涉及的 3 个位置为人当前位置(map[row][column])、人的上一步位置 p1(map[row－1][column])和人的上上一步位置 p2(map[row－2][column])。

首先判断 p1 处的图像类型，这里的图像类型可能为 BOX、BOXONEND、WALL、GRASS 和 END。

(1) 如果前方 p1 是围墙 WALL，因为不能移动，所以不用处理直接返回即可。

(2) 如果 p1 为 BOX、BOXONEND，则判断 p2 处的图像类型，这里需要处理的图像类型为 END、GRASS，如果 p2 的图像类型是这两种图像，则进行以下处理。

保存当前整个游戏的地图信息到 ArrayList 类型的 list 中，用于撤销动作。

```
    Map currMap = new Map(row,column,map);
    list.add(currMap);
```

判断 p2 处是否为目的地。如果是目的地，则 p2 方格状态为放置好的箱子，否则为箱子。

```
    byte boxTemp = map[row - 2][column] == END?BOXONEND: BOX;
    map[row - 2][column] = boxTemp;
```

人往上走一步，需判断 p1 处是 BOX 还是 BOXONEND。如果是 BOX，则 p1 方格状态改为 MANUP(人在 p1 位置)，否则为 MANUPONEND(人在目的地)。

```
byte manTemp = map[row - 1][column] == BOX ? MANUP: MANUPONEND;
map[row - 1][column] = manTemp;
```

人刚才站的地方变成 GRASS 或者 END。

```
map[row][column] = grassOrEnd(map[row][column])
```

修改人的位置在 map 数组中的行坐标(row--)。

(3) 如果 p1 的图像类型为 GRASS 或者 END，则只需要做以下处理。

保存当前整个游戏的地图信息到 ArrayList 类型的 list 中，用于撤销动作。

```
Map currMap = new Map(row,column,map);
list.add(currMap);
```

判断 p1 处是否为目的地。如果是目的地，则 p1 方格状态改为 MANUPONEND(人在目的地)，否则 p1 方格状态改为 MANUP(人在 p1 位置)。

```
byte temp = map[row - 1][column] == END? MANUPONEND: MANUP;
map[row - 1][column] = temp;
```

人刚才站的地方变成 GRASS 或者 END。

```
map[row][column] = grassOrEnd(map[row][column])
```

修改人的位置在 map 数组中的行坐标(row--)。

具体向上移动的代码如下：

```
private void moveUp() {  //向上
    //上一步 p1 为 WALL
    if (map[row - 1][column] == WALL)
        return;
    //上一步 p1 为 BOX、BOXONEND,需考虑 p2
    if (map[row - 1][column] == BOX || map[row - 1][column] == BOXONEND) {
        //上上一步 p2 为 END、GRASS,则向上一步,其他不用处理
        if (map[row - 2][column] == END || map[row - 2][column] == GRASS) {
            Map currMap = new Map(row,column,map);
            list.add(currMap);
            byte boxTemp = map[row - 2][column] == END?BOXONEND: BOX;
            byte manTemp = map[row - 1][column] == BOX?MANUP:MANUPONEND;
            //箱子变成 boxTemp,箱子往上一步
            map[row - 2][column] = boxTemp;
            //人变成 mapTemp,人往上走一步
            map[row - 1][column] = manTemp;
            //人刚才站的地方变成 GRASS 或者 END
            map[row][column] = grassOrEnd(map[row][column]);
```

```
            //人离开后修改人的坐标
            row--;
        }
    } else {
        //上一步 p1 为 GRASS、END,无须考虑 p2,其他情况不用处理
        if (map[row - 1][column] == GRASS || map[row - 1][column] == END) {
            Map currMap = new Map(row,column,map);
            list.add(currMap);
            byte temp = map[row - 1][column] == END?MANUPONEND:MANUP;
            //人变成 temp,人往上走一步
            map[row - 1][column] = temp;
            //人刚才站的地方变成 GRASS 或者 END
            map[row][column] = grassOrEnd(map[row][column]);
            //人离开后修改人的坐标
            row--;
        }
    }
}
```

向下、向左、向右移动与向上移动类似,这里不再赘述。

```
private void moveDown() {                    //向下
    …
}
private void moveLeft() {                    //向左
    //左一步 p1 为 WALL
    if (map[row][column - 1] == WALL)
        return;
    //左一步 p1 为 BOX、BOXONEND,需考虑 p2
    if (map[row][column - 1] == BOX || map[row][column - 1] == BOXONEND) {
        //左左一步 p2 为 END、GRASS,则向左一步,其他不用处理
        if (map[row][column - 2] == END || map[row][column - 2] == GRASS) {
            Map currMap = new Map(row,column,map);
            list.add(currMap);
            byte boxTemp = map[row][column - 2] == END ? BOXONEND: BOX;
            byte manTemp = map[row][column - 1] == BOX ? MANLEFT: MANLEFTONEND;
            //箱子变成 boxTemp,箱子往左一步
            map[row][column - 2] = boxTemp;
            //人变成 manTemp,人往左走一步
            map[row][column - 1] = manTemp;
            //人刚才站的地方变成 GRASS 或者 END
            map[row][column] = grassOrEnd(map[row][column]);
            column--;
        }
    } else {
        //左一步 p1 为 GRASS、END,无须考虑 p2,其他情况不用处理
        if (map[row][column - 1] == GRASS || map[row][column - 1] == END) {
            Map currMap = new Map(row,column,map);
            list.add(currMap);
            byte temp = map[row][column - 1] == END ? MANLEFTONEND: MANLEFT;
```

```
            //人变成 temp,人往左走一步
            map[row][column - 1] = temp;
            //人刚才站的地方变成 GRASS 或者 END
            map[row][column] = grassOrEnd(map[row][column]);
            column -- ;

        }
    }
}
private void moveRight() { //向右
    …
}
```

isFinished()验证是否过关。如果有目的地 END 值或人在目的地,则表示没有过关。

```
public boolean isFinished() {
    for (int i = 0; i < mapRow; i++)
        for (int j = 0; j < mapColumn; j++)
            if (map[i][j] == END || map[i][j] == MANDOWNONEND
               || map[i][j] == MANUPONEND || map[i][j] == MANLEFTONEND
                    || map[i][j] == MANRIGHTONEND)
                return false;
    return true;
}
```

paint(Graphics g)绘制整个游戏区域图形:

```
public void paint(Graphics g)     //绘图
{
    for (int i = 0; i < mapRow; i++)
        for (int j = 0; j < mapColumn; j++) {
            //绘出地图,i 代表行数,j 代表列数
            if (map[i][j] != 0)
                g.drawImage(pic[map[i][j]],leftX + j * 30,leftY + i * 30,this);
        }
    g.setColor(Color.RED);
    g.setFont(new Font("楷体_2312",Font.BOLD,30));
    g.drawString("现在是第",150,140);
    g.drawString(String.valueOf(grade + 1),310,140);
    g.drawString("关",360,140);
}
```

getManX()、getManY()返回人的位置:

```
public int getManX() {
    return row;
}
public int getManY() {
    return column;
}
```

getGrade()返回当前关卡数：

```
public int getGrade() {
    return grade;
}
```

getMap(int grade)返回当前关的地图信息：

```
public byte[][] getMap(int grade) {
    return MapFactory.getMap(grade);
}
```

DisplayToast(String str)显示提示信息对话框：

```
/* 显示提示信息对话框 */
public void DisplayToast(String str) {
    JOptionPane.showMessageDialog(null,str,"提示",
            JOptionPane.ERROR_MESSAGE);
}
```

undo()撤销移动操作：

```
public void undo() {
    if (acceptKey) {
        //撤销
        if (list.size() > 0) {
            //若要撤销,必须走过
            Map priorMap = (Map) list.get(list.size() - 1);
            map = priorMap.getMap();
            row = priorMap.getManX();
            column = priorMap.getManY();
            repaint();
            list.remove(list.size() - 1);
        } else
            DisplayToast("不能再撤销!");
    } else {
        DisplayToast("此关已完成,不能撤销!");
    }
}
```

nextGrade()实现下一关的初始化及调用 repaint()显示游戏界面：

```
public void nextGrade() {
    if (grade >= MapFactory.getCount() - 1) {
        DisplayToast("恭喜你完成所有关卡!");
        acceptKey = false;
    } else {
        grade++;
        initMap();
        repaint();
```

```
            acceptKey = true;
        }
    }
```

priorGrade()实现上一关的初始化及调用 repaint()显示游戏界面：

```
public void priorGrade() {
    grade -- ;
    acceptKey = true;
    if (grade < 0)
        grade = 0;
    initMap();
    repaint();
}
```

在键盘事件 keyPressed 中根据用户的按键分别调用向 4 个方向移动的方法。

```
public void keyPressed(KeyEvent e)                    //键盘事件
{
    if(e.getKeyCode() == KeyEvent.VK_UP){
        //向上
        moveUp();}
    if(e.getKeyCode() == KeyEvent.VK_DOWN){
        //向下
        moveDown();
        }
    if(e.getKeyCode() == KeyEvent.VK_LEFT){    //向左
        moveLeft();
    }
    if(e.getKeyCode() == KeyEvent.VK_RIGHT){   //向右
        moveRight();
    }
    repaint();
    if (isFinished()) {
        //禁用按键
        acceptKey = false;
        if(grade == 10){JOptionPane.showMessageDialog(this,"恭喜通过最后一关");}
        else
        {
            //提示进入下一关
            String msg = "恭喜您通过第" + grade + "关!!!\n 是否要进入下一关?";
            int type = JOptionPane.YES_NO_OPTION;
            String title = "过关";
            int choice = 0;
            choice = JOptionPane.showConfirmDialog(null,msg,title,type);
            if(choice == 1)System.exit(0);
            else if(choice == 0)
            {
                //进入下一关
                acceptKey = true;
                nextGrade();
```

```
            }
        }
    }
}
public void actionPerformed(ActionEvent arg0) {
    //TODO Auto-generated method stub
}
public void keyReleased(KeyEvent arg0) {
    //TODO Auto-generated method stub
}
public void keyTyped(KeyEvent arg0) {
    //TODO Auto-generated method stub
}
```

鼠标事件的相关代码如下：

```
public void mouseClicked(MouseEvent e) {
    //TODO Auto-generated method stub
    if (e.getButton() == MouseEvent.BUTTON3) //右击撤销移动
    {
         undo();                              //撤销移动
    }
}
```

main()方法(程序入口)实现一个 GameFrame 窗口：

```
    public static void main(String[] args)
    {
        new GameFrame();
    }
}
```

5.3.4 设计播放背景音乐类

设计播放背景音乐类(Sound),用于播放背景音乐。

```
import javax.sound.midi.*;
import java.io.File;
class Sound                          //播放背景音乐类
{
    String path = new String("musics\\");
    String file = new String("nor.mid");
    Sequence seq;
    Sequencer midi;
    boolean sign;
    void loadSound()
    {
        try {
            seq = MidiSystem.getSequence(new File(path + file));
            midi = MidiSystem.getSequencer();
            midi.open();
```

```
                midi.setSequence(seq);
                midi.start();
                midi.setLoopCount(Sequencer.LOOP_CONTINUOUSLY);
            }
            catch (Exception ex) {ex.printStackTrace();}
            sign = true;
        }
        void mystop(){midi.stop();midi.close();sign = false;}
        boolean isplay(){return sign;}
        void setMusic(String e){file = e;}
    }
```

游戏动作的音效通常比较短小，其播放时间最多只有一两秒钟，而游戏背景音乐需要持续比较长的时间，少则十几秒，多则几分钟甚至更长时间。Java 支持的背景音乐文件格式主要有 CD、MP3 和 MIDI。这里使用的是 MIDI 格式音乐。

MIDI(Musical Instrument Digital Interface，乐器数字接口)是 20 世纪 80 年代初为解决电声乐器之间的通信问题而提出的。MIDI 传输的不是声音信号，而是音符、控制参数等指令，它指示 MIDI 设备要做什么、怎么做，例如演奏哪个音符、多大音量等。它们被统一表示成 MIDI 消息(MIDI Message)。

Java 提供了专门的包来处理和播放 MIDI 音乐，包名为 javax.sound.midi，其中包括与 MIDI 相关的各个类及其方法。读取 MIDI 文件信息的主要步骤如下。

(1) 打开 MIDI 文件：

```
Sequence sequence = MidiSystem.getSequence(new File(filename));
```

(2) 建立音频序列：

```
Sequencer sequencer = MidiSystem.getSequencer();
```

(3) 打开音频序列：

```
sequencer.open();
```

在读取 MIDI 音频序列并初始化音频序列器之后，接下来便可以播放 MIDI 音乐了。播放 MIDI 音乐的步骤如下。

(1) 读取即将播放的音频序列：

```
sequencer.setSequence(sequence);
```

(2) 播放音频序列：

```
sequencer.start();
```

如果要循环播放 MIDI 音乐，则可以使用 Sequencer 的 isRunning()方法进行判断，若该方法的返回值为 false，说明音乐已经播放完毕，此时可以再次调用 start()方法重新开始播放。