Tarena
达内科技

arena eaching ystem

JAVA经典项目集锦

Java企业应用及互联网
高级工程师培训课程

达内集团教学研发部 编著

目　录

飞机大战

1. 问题

Shoot 游戏是一款十分有趣的射击类小游戏，流畅的画面，高难度的挑战。游戏中，玩家驾驶英雄机，在空中进行战斗。点击并移动自己的英雄机，发射炮弹，打掉敌飞机以及蜜蜂，来获得分数和奖励，打掉一架敌飞机赢得 5 分，打掉一只蜜蜂赢得 1 条命或是获得 20 次双倍火力，如果撞上敌飞机或小蜜蜂，将减少命、双倍火力清零。每撞到一次蜜蜂或是敌飞机命减 1，当命数为 0 时，则游戏结束。初始界面如图-1 所示：

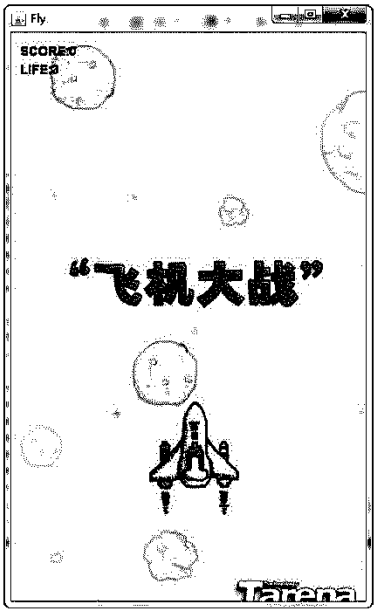

图 - 1

从图-1 可以看出，默认分数为 0，默认 3 条命，请看如图-2 所示具体介绍。

图 – 2

玩家在如图-1 所示的界面的任意位置，按下鼠标左键，开始游戏。界面效果如图-3 所示：

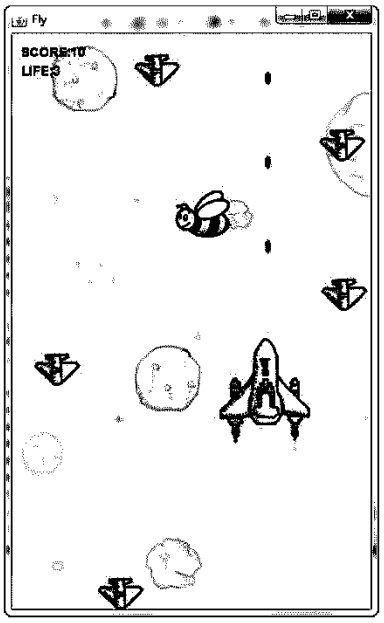

图 – 3

开始游戏后，天空中不断有敌飞机和蜜蜂出现，英雄机发射子弹打掉敌飞机和蜜蜂以获取分数、增命或是双倍火力。如果英雄机与飞机或蜜蜂发生碰撞则减少命并且双倍火力清零，直至寿命为 0 则游戏结束。界面效果如图-4 所示：

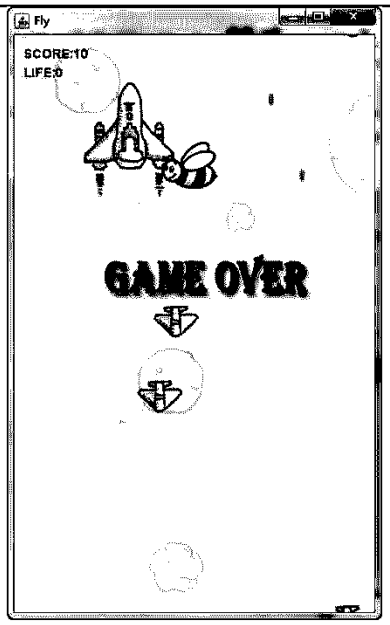

图 – 4

此时点击鼠标左键，可以重新进入开始状态。

另外，在游戏进行过程中，鼠标离开游戏界面，游戏将进入暂停状态，界面效果如图-5所示：

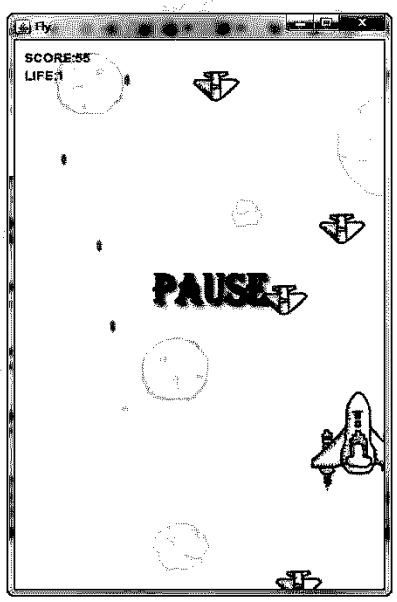

图 – 5

当鼠标再次移入界面时，游戏将继续进行。

2. 方案

软件的开发过程如下：

1．需求(软件功能的文字描述)

2．业务需求分析：找对象，以及对象之间的关系。本项目中对象如下所示：

```
ShootGame
  |-- 英雄机 Hero
  |-- 敌飞机 Airplane
  |-- 蜜蜂 Bee
  |-- 子弹 Bullet
```

3．软件概要设计

数据建模：使用一个数据模型，描述对象的关系。使用绘图坐标系作为参考模型，英雄机、敌飞机、蜜蜂、子弹都是矩形区域，如图-6 所示：

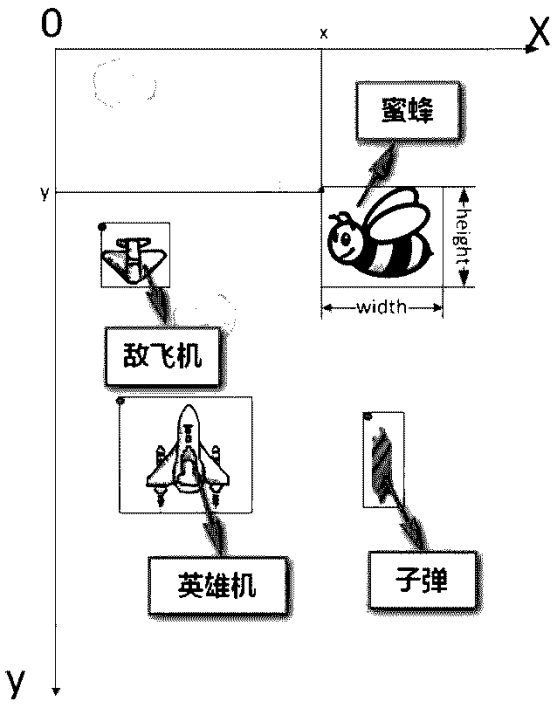

图 - 6

图-6 中以蜜蜂为例，标识出了数据 x、y、width 以及 height 所表示的位置。英雄机、敌飞机、子弹与蜜蜂的这四个属性类似的。

4．类的设计

本案例中的类、及类之间的关系如图-7 所示：

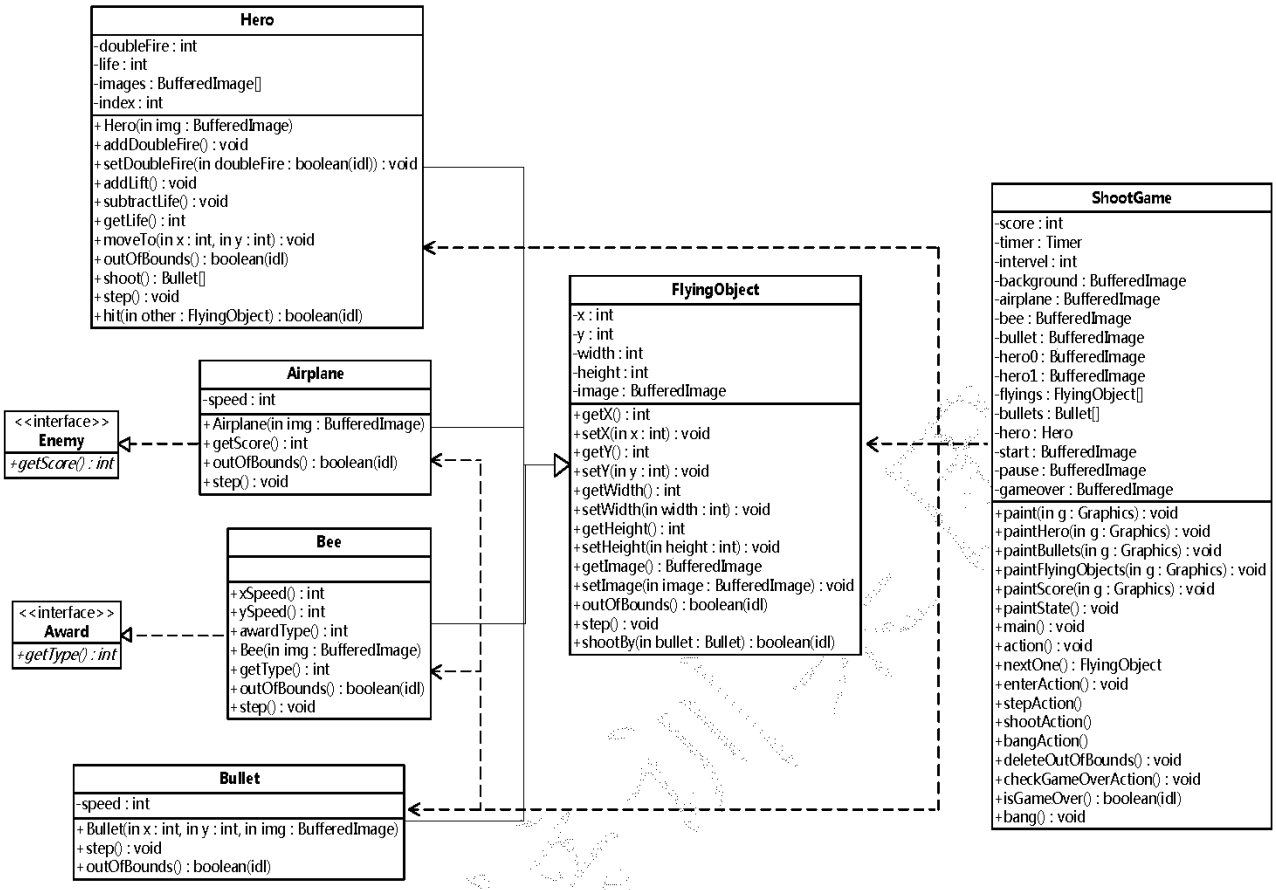

图 - 7

3. 步骤

实现此案例需要按照如下步骤进行。

步骤一：新建工程和包

首先，新建名为 Shoot 的 Java 工程； 然后，在工程下的 src 目录下新建包 com.tarena.shoot； 最后，将该工程所需的图片拷贝到该包下，工程结构如图-8 所示：

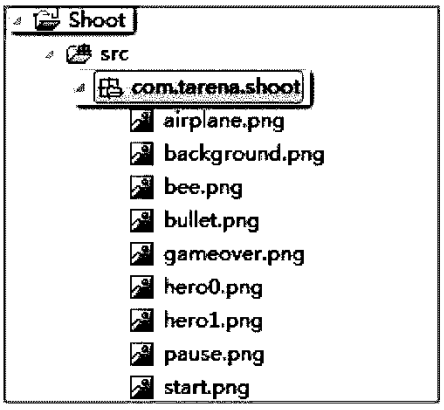

图- 8

在项目中，包的名字一般为公司域名倒过来，再加上项目名称，即为包名。如图-8 中的包名为 com.tarena.shoot，其中，com.tarena 是达内公司的域名倒过来，shoot 为本项目的名称。

步骤二：创建抽象父类 FlyingObject

由图-6 可以分析出英雄机、敌飞机、子弹以及蜜蜂都有 x、y、width 以及 height 属性，因此，将这些属性抽象到父类 FlyingObject 中。另外，它们在界面上都以图片的形式显示，因此在父类 FlyingObject 中，添加 image 属性，表示它们的贴图，并提供上述 5 个属性的 getter 和 setter 方法，FlyingObject 类的代码如下所示：

```
package com.tarena.shoot;
import java.awt.image.BufferedImage;

public abstract class FlyingObject {
 protected int x;    //x 坐标
 protected int y;    //y 坐标
 protected int width;    //宽
 protected int height;   //高
 protected BufferedImage image;   //图片
 public int getX() {
     return x;
 }

 public void setX(int x) {
     this.x = x;
 }

 public int getY() {
     return y;
 }

 public void setY(int y) {
     this.y = y;
 }

 public int getWidth() {
     return width;
 }

 public void setWidth(int width) {
     this.width = width;
 }

 public int getHeight() {
     return height;
 }

 public void setHeight(int height) {
     this.height = height;
 }

 public BufferedImage getImage() {
     return image;
 }

 public void setImage(BufferedImage image) {
     this.image = image;
 }
}
```

步骤三：创建接口 Enemy，实现该接口的类为敌人

创建接口 Enemy，表示敌人。如果子弹击中敌飞机，英雄机可以获取分数，因此，在 Enemy 接口中提供获取分数的方法，代码如下所示：

```
/**
 * 敌人，可以有分数
 */
public interface Enemy {
 /** 敌人的分数  */
 int getScore();
}
```

步骤四：创建接口 Award，实现该接口的类表示奖励

创建接口 Award，表示奖励。如果子弹击中了蜜蜂，英雄机可以获取奖励。奖励有两种形式，分别是双倍火力或增命，因此，提供获取的奖励类型的方法，代码如下所示：

```
package com.tarena.shoot;
/**
 * 奖励
 */
public interface Award {
 int DOUBLE_FIRE = 0;  //双倍火力
 int LIFE = 1;   //1 条命
 /** 获得奖励类型(上面的 0 或 1) */
 int getType();
}
```

上述代码中，如果奖励类型为 0，则表示奖励双倍火力；如果奖励类型为 1，则表示奖励 1 条命。

步骤五：新建类 Airplane,表示敌飞机

新建类 Airplane ,表示敌飞机。敌飞机属于飞行物 ,因此 ,继承自 FlyingObject 类；敌飞机也属于敌人，因此，需要实现 Enemy 接口。敌飞机可以向下移动，因此有移动的速度作为属性，代码如下所示：

```
package com.tarena.shoot;

import java.util.Random;

import com.tarena.shoot.ShootGame;

/**
 * 敌飞机：是飞行物，也是敌人
 */
public class Airplane extends FlyingObject implements Enemy {
 private int speed = 2;

 public int getScore() {
     return 0;
 }
}
```

步骤六：新建类 Bee，表示蜜蜂

新建类 Bee，表示蜜蜂。蜜蜂属于飞行物，因此，继承自 FlyingObject 类；击中蜜蜂可以获得奖励，因此，需要实现 Award 接口，并且有奖励类型作为属性。蜜蜂可以左右移动、向下移动，因此有移动的速度作为属性，代码如下所示：

```
package com.tarena.shoot;

/** 蜜蜂 */
public class Bee extends FlyingObject implements Award{
 private int xSpeed = 1;   //x 坐标移动速度
 private int ySpeed = 2;   //y 坐标移动速度
 private int awardType;    //奖励类型
 public int getType() {
     return 0;
 }
}
```

步骤七：新建类 Bullet，表示子弹

新建类 Bullet，表示子弹。子弹属于飞行物，因此，继承自 FlyingObject 类；子弹可以向上移动，因此有移动的速度作为属性，代码如下所示：

```
package com.tarena.shoot;

/**
 * 子弹类:是飞行物
 */
public class Bullet extends FlyingObject {
 private int speed = 3;  //移动的速度
}
```

步骤八：新建类 Hero，表示英雄机

新建类 Hero，表示英雄机。英雄机属于飞行物，因此，继承自 FlyingObject 类；英雄机发出子弹，击中蜜蜂可以获取双倍火力或增命，因此，将双倍火力的子弹数量和命的数量作为该类的属性，代码如下所示：

```
package com.tarena.shoot;

import java.awt.image.BufferedImage;

/**
 * 英雄机:是飞行物
 */
public class Hero extends FlyingObject{
 protected BufferedImage[] images = {};
 protected int index = 0;

 private int doubleFire;
 private int life;
}
```

上述代码中，还有 images 属性和 index 属性，其中 images 属性表示 Hero 的贴图，Hero 的贴图由两张图片组成，因此使用数组类型；index 属性是使两张图片进行交替显示的计数。

步骤九：新建类 ShootGame，加载图片

新建类 ShootGame ,该类继承自 JPanel ,在该类中 ,使用静态常量定义面板的宽和高，并使用 ImageIO 的 read 方法加载图片，代码如下所示：

```
package com.tarena.shoot;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;
import javax.swing.JPanel;

public class ShootGame extends JPanel {
 public static final int WIDTH = 400; // 面板宽
 public static final int HEIGHT = 654; // 面板高

 public static BufferedImage background;
 public static BufferedImage start;
 public static BufferedImage airplane;
 public static BufferedImage bee;
 public static BufferedImage bullet;
 public static BufferedImage hero0;
 public static BufferedImage hero1;
 public static BufferedImage pause;
 public static BufferedImage gameover;

 static {// 静态代码块
     try {
         background = ImageIO.read(ShootGame.class
                 .getResource("background.png"));
         airplane = ImageIO
                 .read(ShootGame.class.getResource("airplane.png"));
         bee = ImageIO.read(ShootGame.class.getResource("bee.png"));
         bullet = ImageIO.read(ShootGame.class.getResource("bullet.png"));
         hero0 = ImageIO.read(ShootGame.class.getResource("hero0.png"));
         hero1 = ImageIO.read(ShootGame.class.getResource("hero1.png"));
         pause = ImageIO.read(ShootGame.class.getResource("pause.png"));
         gameover = ImageIO
                 .read(ShootGame.class.getResource("gameover.png"));
         start = ImageIO
                 .read(ShootGame.class.getResource("start.png"));
     } catch (Exception e) {
         e.printStackTrace();
     }
 }
}
```

步骤十：为 Bee 类添加构造方法，初始化属性

在 Bee 类中添加构造方法，将属性进行初始化，请看图-9。

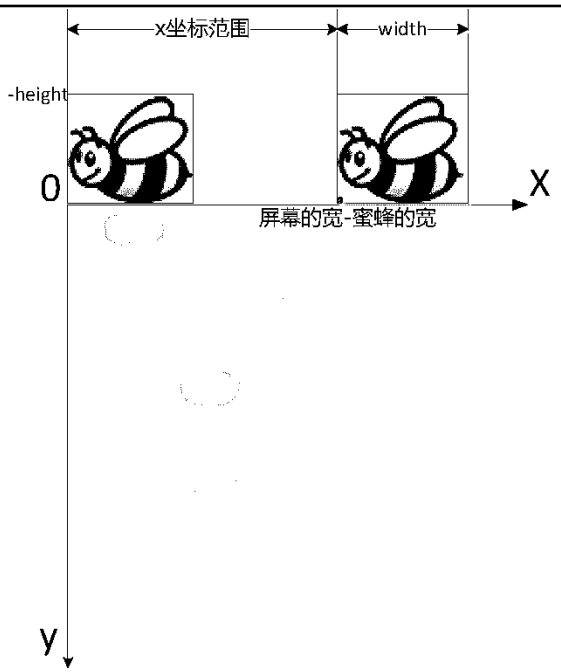

图 - 9

从图-9 可以看出，image 属性初始化为 ShootGame 类加载的图片；width 初始化为图片的宽度、height 初始化为图片的高度、x 坐标的范围为 0 到(屏幕的宽度 - 蜜蜂的宽度)，因此，x 坐标初始化为这个范围的随机数；y 坐标初始化为蜜蜂的负高度；奖励的类型为 2 以内的随机数，即为 0 或者 1，代码如下所示：

```
public Bee(){
    this.image = ShootGame.bee;
    width = image.getWidth();
    height = image.getHeight();
    y = -height;
    Random rand = new Random();
    x = rand.nextInt(ShootGame.WIDTH - width);
    awardType = rand.nextInt(2);
}
```

步骤十一：为 Airplane 类添加构造方法，初始化属性

在 Airplane 类中添加构造方法，将属性进行初始化， Airplane 的初始化与 Bee 类似，代码如下所示：

```
public Airplane(){
    this.image = ShootGame.airplane;
    width = image.getWidth();
    height = image.getHeight();
    y = -height;
    x = (int)(Math.random()*(ShootGame.WIDTH - width));
}
```

步骤十二：为 Bullet 类添加构造方法，初始化属性

在 Bullet 类中添加构造方法，将属性进行初始化，代码如下所示：

```
package com.tarena.shoot;
/**
 * 子弹类:是飞行物
 */
public class Bullet extends FlyingObject {
 private int speed = 3;  //移动的速度
 public Bullet(int x,int y){
     this.x = x;
     this.y = y;
     this.image = ShootGame.bullet;
 }
}
```

步骤十三：为 Hero 类添加构造方法，初始化属性

在 Hero 类中添加构造方法，将属性进行初始化，英雄机的出场位置如图-10 所示：

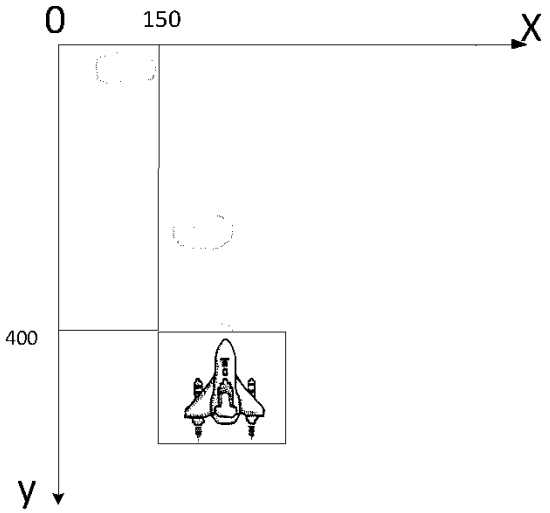

图 - 10

代码如下所示：

```
public Hero() {
    life = 3;
    doubleFire = 0;
    this.image = ShootGame.hero0;
    images = new BufferedImage[]{ShootGame.hero0, ShootGame.hero1};
    width = image.getWidth();
    height = image.getHeight();
    x = 150;
    y = 400;
}
```

步骤十四：编写 main 方法

在 ShootGame 类中添加 main 方法，在该方法中设置窗口的大小、居中、点击窗口的右上角“X”关闭窗口以及设置窗口可见，代码如下所示：

```
public static void main(String[] args) {
    JFrame frame = new JFrame("Fly");
    ShootGame game = new ShootGame(); // 面板对象
```

```
        frame.add(game); // 将面板添加到 JFrame 中
        frame.setSize(WIDTH, HEIGHT); // 大小
        frame.setAlwaysOnTop(true); // 其总在最上
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 默认关闭操作
        frame.setLocationRelativeTo(null); // 设置窗体初始位置
        frame.setVisible(true); // 尽快调用 paint
    }
```

运行 ShootGame 类，运行效果如图-11 所示：

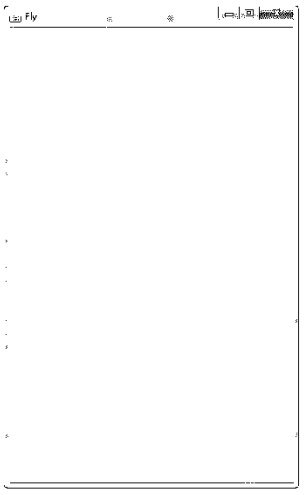

图 - 11

步骤十五：绘制界面

绘制界面的过程如下：

1. 在 ShootGame 类中，添加 FlyingObject[]类型的属性 flyings，用于存储射击游戏中的所有敌飞机和蜜蜂；
2. 在 ShootGame 类中，添加 Bullet[]类型的属性 bullets，用于存储射击游戏中的所有的子弹；
3. 在 ShootGame 类中，添加 Hero 类型的属性 hero，表示英雄机；
4. 在 ShootGame 类中，添加 paintHero 方法、paintBullets 方法、paintFlyingObjects 方法，分别用于实现在面板上画英雄机、子弹、敌飞机、蜜蜂；并重写 paint 方法，在该方法中调用上述三个方法；
5. 在 ShootGame 类中，添加构造初始化属性 flyings、bullets 以及 hero；
6. 重构 Airplane 类和 Bee 类，设置固定的 x、y 坐标位置，以便显示查看。

ShootGame 类中添加的代码如下所示：

```
package com.tarena.shoot;

import java.awt.Graphics;
import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;
```

```
public class ShootGame extends JPanel {
    public static final int WIDTH = 400; // 面板宽
    public static final int HEIGHT = 654; // 面板高

    public static BufferedImage background;
    public static BufferedImage start;
    public static BufferedImage airplane;
    public static BufferedImage bee;
    public static BufferedImage bullet;
    public static BufferedImage hero0;
    public static BufferedImage hero1;
    public static BufferedImage pause;
    public static BufferedImage gameover;

    private FlyingObject[] flyings = {}; // 敌机数组
    private Bullet[] bullets = {}; // 子弹数组
    private Hero hero = new Hero(); // 英雄机

    public ShootGame(){
        //初始化一只蜜蜂一架飞机
        flyings=new FlyingObject[2];
        flyings[0]=new Airplane();
        flyings[1]=new Bee();
        //初始化一颗子弹
        bullets=new Bullet[1];
        bullets[0]=new Bullet(200,350);
    }

    static {// 静态代码块
        ... ...
    }

    @Override
    public void paint(Graphics g) {
        g.drawImage(background, 0, 0, null); // 画背景图
        paintHero(g); // 画英雄机
        paintBullets(g); // 画子弹
        paintFlyingObjects(g); // 画飞行物
    }

    /** 画英雄机 */
    public void paintHero(Graphics g) {
        g.drawImage(hero.getImage(), hero.getX(), hero.getY(), null);
    }

    /** 画子弹 */
    public void paintBullets(Graphics g) {
        for (int i = 0; i < bullets.length; i++) {
            Bullet b = bullets[i];
            g.drawImage(b.getImage(), b.getX() , b.getY(),null);
        }
    }

    /** 画飞行物 */
    public void paintFlyingObjects(Graphics g) {
        for (int i = 0; i < flyings.length; i++) {
            FlyingObject f = flyings[i];
```

```
            g.drawImage(f.getImage(), f.getX(), f.getY(), null);
        }
    }

    public static void main(String[] args) {
        ... ...
    }
}
```

Bee 类修改的代码如下所示：

```
package com.tarena.shoot;

import java.util.Random;

/** 蜜蜂 */
public class Bee extends FlyingObject implements Award{
    private int xSpeed = 1;   //x 坐标移动速度
    private int ySpeed = 2;   //y 坐标移动速度
    private int awardType;    //奖励类型
    public Bee(){
        this.image = ShootGame.bee;
        width = image.getWidth();
        height = image.getHeight();

        //y = -height;

        Random rand = new Random();

        //x = rand.nextInt(ShootGame.WIDTH - width);
        x=100;
        y=200;

        awardType = rand.nextInt(2);
    }
    public int getType() {
        return 0;
    }
}
```

Airplane 修改的代码如下所示：

```
package com.tarena.shoot;

import java.util.Random;

import com.tarena.shoot.ShootGame;

/**
 * 敌飞机：是飞行物，也是敌人
 */
public class Airplane extends FlyingObject implements Enemy {
    private int speed = 2;

    /** 初始化数据 */
    public Airplane(){
        this.image = ShootGame.airplane;
        width = image.getWidth();
```

```
        height = image.getHeight();

        //y = -height;
        //x = (int)(Math.random()*(ShootGame.WIDTH - width));
        y=100;
        x=100;

    }

    public int getScore() {
        return 0;
    }
}
```

步骤十六：运行

运行 ShootGame 类，显示的界面效果如图-12 所示：

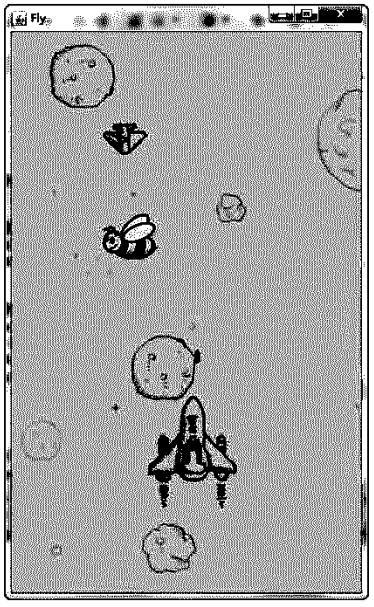

图 - 12

从图-12 可以发现，在界面上显示了英雄机、敌飞机、蜜蜂以及子弹。

步骤十七：实现英雄机、敌飞机、蜜蜂、子弹的移动

1．由于英雄机、敌飞机、蜜蜂以及子弹都是可以移动的，因此在 FlyingObject 类中添加抽象方法 step，声明飞行物移动一步的方法，代码如下所示：

```
/**
 * 飞行物移动一步
 */
public abstract void step();
```

3．在 Airplane 类中，实现父类 FlyingObject 的 step 方法，实现的代码如下所示：

```
@Override
public void step() {   //移动
    y += speed;
```

```
}
```

4．在 Bee 类中，实现父类 FlyingObject 的 step 方法，实现的代码如下所示：

```
@Override
public void step() {      //可斜飞
    x += xSpeed;
    y += ySpeed;
    if(x > ShootGame.WIDTH-width){
        xSpeed = -1;
    }
    if(x < 0){
        xSpeed = 1;
    }
}
```

由于蜜蜂可以左右移动，因此，当移动到屏幕的最右端时，使其向左移动；当移动到屏幕的最左端时，使其向右移动。蜜蜂左右移出屏幕的临界状态如图-13 所示：

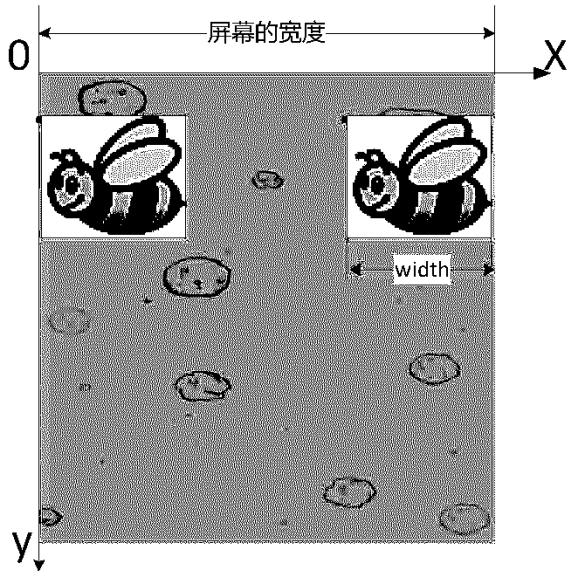

图 – 13

从图-13 可以看出，当蜜蜂的 x 坐标小于 0 或大于屏幕的宽度 – 蜜蜂的宽度时，蜜蜂移出游戏界面。

5．在 Bullet 类中，实现父类 FlyingObject 的 step 方法，实现的代码如下所示：

```
@Override
public void step(){   //移动方法
    y-=speed;
}
```

6．在 Hero 类中，实现父类 FlyingObject 的 step 方法，实现的代码如下所示：

```
@Override
public void step() {
    if(images.length>0){
```

```
            image = images[index++/10%images.length];
        }
    }
```

英雄机的 step 方法，实现了图片的更换，有动画效果。

7．在 ShootGame 类中，添加 nextOne 方法，该方法用于随机产生蜜蜂和敌飞机，代码如下所示：

```
/**
 * 随机生成飞行物
 *
 * @return 飞行物对象
 */
public static FlyingObject nextOne() {
    Random random = new Random();
    int type = random.nextInt(20); // [0,19)
    if (type==0) {
        return new Bee();
    }else{
        return new Airplane();
    }
}
```

从代码中可以发现，产生蜜蜂的几率会小一些，只有当产生的随机数为 0 时，才产生蜜蜂。

8．在 ShootGame 类中，添加 enterAction 方法，该方法用于实现每调用 40 次该方法，将随机生成的一个蜜蜂或是敌飞机放入 flying 数组中，代码如下所示：

```
int flyEnteredIndex = 0; // 飞行物入场计数
/** 飞行物入场 */
public void enterAction() {
    flyEnteredIndex++;
    if (flyEnteredIndex % 40 == 0) { // 400毫秒--10*40
        FlyingObject obj = nextOne(); // 随机生成一个飞行物
        flyings = Arrays.copyOf(flyings, flyings.length + 1);扩容
        flyings[flyings.length - 1] = obj;//放到最后一位
    }
}
```

9．在 ShootGame 类中，添加 stepAction 方法，该方法用于实现所有飞行物的移动，代码如下所示：

```
public void stepAction() {
    /** 飞行物走一步 */
    for (int i = 0; i < flyings.length; i++) {
        FlyingObject f = flyings[i];
        f.step();
    }

    /** 子弹走一步 */
    for (int i = 0; i < bullets.length; i++) {
        Bullet b = bullets[i];
```

```
            b.step();
        }
        hero.step();
    }
```

10. 在 ShootGame 类中，添加如下两个属性：

```
private Timer timer; // 定时器
private int intervel = 1000/100; // 时间间隔(毫秒)
```

另外，添加 action 方法，该方法使用 Timer 实现每隔 10 毫秒入场一个飞机或是蜜蜂，并使所有飞行物移动一步，最后重绘页面，代码如下所示：

```
public void action() { // 启动执行代码
    timer = new Timer(); // 主流程控制
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
                enterAction(); // 飞行物入场
                stepAction(); // 走一步
            repaint(); // 重绘，调用paint()方法
        }
    }, intervel, intervel);
}
```

11.在 main 方法中调用 action 方法，代码如下所示：

```
public static void main(String[] args) {
    JFrame frame = new JFrame("Fly");
    ShootGame game = new ShootGame(); // 面板对象
    frame.add(game); // 将面板添加到 JFrame 中
    frame.setSize(WIDTH, HEIGHT); // 大小
    frame.setAlwaysOnTop(true); // 其总在最上
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 默认关闭操作
    frame.setLocationRelativeTo(null); // 设置窗体初始位置
    frame.setVisible(true); // 尽快调用paint

    game.action(); // 启动执行

}
```

12.重构 Airplane 类的构造方法，代码如下所示：

```
/** 初始化数据 */
public Airplane(){
    this.image = ShootGame.airplane;
    width = image.getWidth();
    height = image.getHeight();
```

```
        y = -height;
        x = (int)(Math.random()*(ShootGame.WIDTH - width));
//          y=100;
//          x=100;

    }
```

13.重构 Bee 类的构造方法，代码如下所示：

```
public Bee(){
    this.image = ShootGame.bee;
    width = image.getWidth();
    height = image.getHeight();

    y = -height;

    Random rand = new Random();

    x = rand.nextInt(ShootGame.WIDTH - width);
//          x=100;
//          y=200;

    awardType = rand.nextInt(2);
}
```

此时，运行 ShootGame 类，会发现敌飞机一直向下移动、子弹一直向上移动、蜜蜂是斜着飞的、英雄机的尾翼是有动画效果的。

步骤十八：实现英雄机发射子弹

1.在 Hero 类中添加 shoot 方法，实现发射子弹，英雄机发射子弹的位置如图-14 所示：

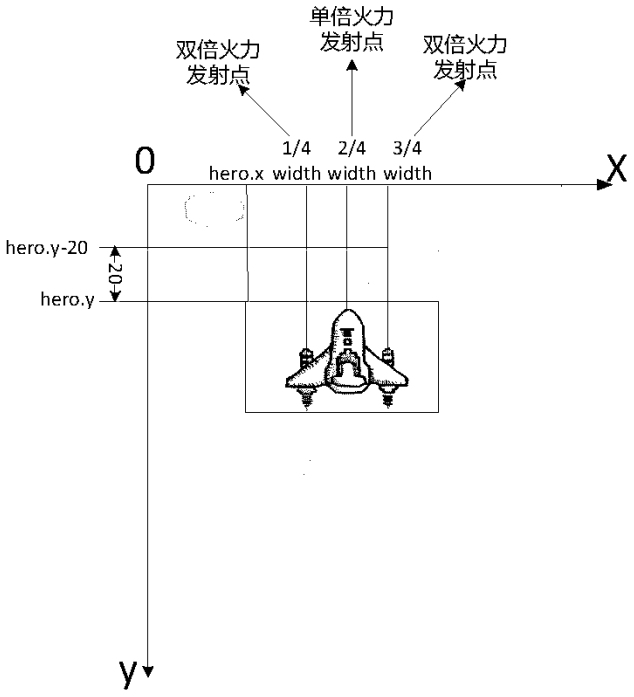

图 - 14

从图-14 可以看出，将英雄机的宽度分成了四分，在 1/2 处发出的子弹是单倍火力的发射点；在 1/4 和 3/4 处发出的子弹是双倍火力的发射点，shoot 方法的代码如下所示：

```
public Bullet[] shoot() { // 发射子弹
    int xStep = width / 4;
    int yStep = 20;
    if (doubleFire>0) {
        Bullet[] bullets = new Bullet[2];
        bullets[0] = new Bullet(x + xStep, y - yStep);
        bullets[1] = new Bullet(x + 3 * xStep, y - yStep);
        doubleFire -= 2;
        return bullets;
    } else { // 单倍
        Bullet[] bullets = new Bullet[1];
        // y-yStep(子弹到飞机的位置)
        bullets[0] = new Bullet(x + 2 * xStep, y - yStep);
        return bullets;
    }
}
```

2.在 ShootGame 类中添加 shootAction 方法，实现每调用 30 次该方法发射一次子弹，并将发射的子弹存储到 bullets 数组中，shootAction 方法的代码如下所示：

```
int shootIndex = 0; // 射击计数
/** 射击 */
public void shootAction() {
    shootIndex++;
    if (shootIndex % 30 == 0) { // 100 毫秒发一颗
        Bullet[] bs = hero.shoot(); // 英雄打出子弹
        bullets = Arrays.copyOf(bullets, bullets.length + bs.length); // 扩
容
        System.arraycopy(bs, 0, bullets, bullets.length - bs.length,
                bs.length); // 追加数组
    }
}
```

3．在 ShootGame 类中的 action 方法调用 shootAction 方法，代码如下所示：

```
public void action() { // 启动执行代码
    timer = new Timer(); // 主流程控制
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
                enterAction(); // 飞行物入场
                stepAction(); // 走一步

                shootAction(); // 射击

                repaint(); // 重绘，调用 paint()方法
        }
    }, intervel, intervel);
}
```

4．重构 ShootGame 类的构造方法，将其中的代码注释掉，注释的代码如下所示：

```
public ShootGame(){
    //初始化一只蜜蜂一架飞机

//        flyings=new FlyingObject[2];
//        flyings[0]=new Airplane();
//        flyings[1]=new Bee();

    //初始化一颗子弹

//        bullets=new Bullet[1];
//        bullets[0]=new Bullet(200,350);

}
```

此时，运行 ShootGame 类，会发现界面上实现了连续发射子弹。

步骤十九：添加鼠标移动事件处理 ，当鼠标移动时，英雄机跟随着移动

1．在 Hero 类中，添加 moveTo 方法，该方法有两个参数 x、y，分别表示鼠标的 x 坐标位置和 y 坐标位置，如图-15 中的红点位置表示鼠标所在的位置，英雄机的中心点。

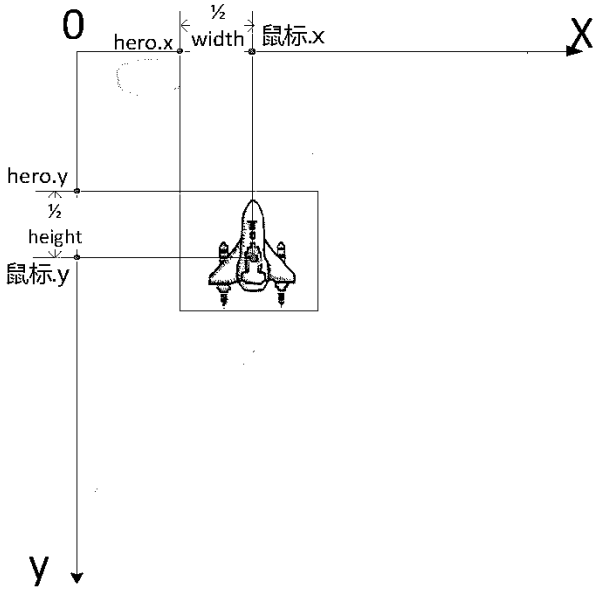

图 - 15

从图-15 可以看出，要实现英雄机跟随着移动而移动，那么英雄机的坐标算法如下：

```
hero.x=鼠标的 x 坐标-width/2；
hero.y=鼠标的 y 坐标-height/2
```

moveTo 方法的代码如下所示：

```
/**
```

```
     * 当前物体移动了一下，相对距离， x,y 鼠标位置
     */
    public void moveTo(int x, int y) {
        this.x = x - width / 2;
        this.y = y - height / 2;
    }
```

2.在 ShootGame 类的 action 方法，添加鼠标的移动事件处理，代码如下所示：

```
public void action() { // 启动执行代码


    // 鼠标监听事件
    MouseAdapter l = new MouseAdapter() {
        @Override
        public void mouseMoved(MouseEvent e) { // 鼠标移动
                int x = e.getX();
                int y = e.getY();
                hero.moveTo(x, y);
        }
    };
    this.addMouseMotionListener(l); // 处理鼠标滑动操作


    timer = new Timer(); // 主流程控制
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
                enterAction(); // 飞行物入场
                stepAction(); // 走一步
                shootAction(); // 射击
                bangAction();
                repaint(); // 重绘，调用paint()方法
        }
    }, intervel, intervel);
}
```

步骤二十：实现子弹打敌飞机和蜜蜂

1．由于蜜蜂和敌飞机都可以被子弹击中，因此在 FlyingObject 类中添加 shootBy 方法，该方法的参数为子弹类型。图-16 以蜜蜂为例，说明了被子弹击中的算法。

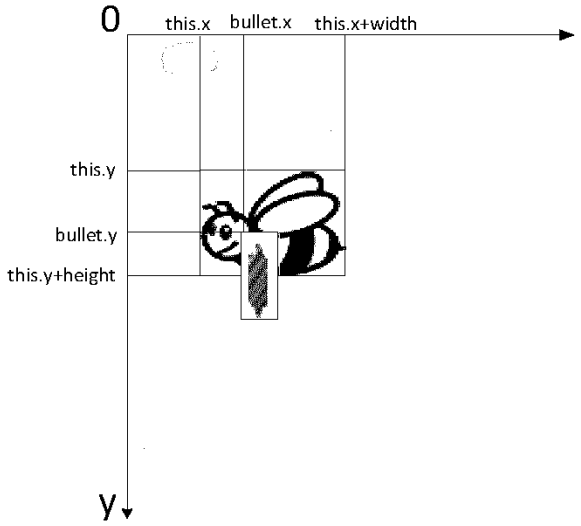

图 - 16

由图-16 可以看出，当子弹的 x 坐标在蜜蜂的 x 与 x+width 之间，并且子弹的 y 坐标在蜜蜂的 y 与 y+height 之间时，子弹击中了蜜蜂，即：

```
bee.x<bullet.x<bee.x+width
&&
bee.y<bullet.y<bee.y+height
```

在代码中把蜜蜂换成 this 就可以了，shootBy 方法代码如下所示：

```
/**
 * 检查当前飞行物体是否被子弹(x,y)击(shoot)中，
 * true 表示击中，飞行物可以被击中
 * @param Bullet 子弹对象
 * @return true 表示被击中了
 */
public boolean shootBy(Bullet bullet){
    int x = bullet.x;  //子弹横坐标
    int y = bullet.y;  //子弹纵坐标
    return this.x<x && x<this.x+width && this.y<y && y<this.y+height;
}
```

2.当英雄机击中蜜蜂时，可以获取奖励，增命或是获得双倍火力，因此在 Hero 类中添加 addDoubleFire 实现获取双倍火力；添加 addLife 方法增命，代码如下所示：

```
public void addDoubleFire(){
    doubleFire += 40;
}

public void addLife() { // 增命
    life++;
}
```

3.在 Airplane 类中，实现 getScore 方法，每击中一架敌飞机获得 5 分，getScore

方法的代码如下所示：

```
public int getScore() {
    return 5;
}
```

4．在 Bee 类中，实现 getType 方法，获取奖励的类型，getType 的代码如下所示：

```
public int getType() {
    return awardType;
}
```

5．在 ShootGame 类中添加属性 score，用于记录得分，代码如下所示：

```
private int score = 0; // 得分
```

6．在 ShootGame 类中，添加 bangAction 方法和 bang 方法，这两个方法实现了子弹与飞行物（蜜蜂或敌飞机）的碰撞检测，详细过程如下：

1） 循环遍历存储所有的子弹数组 bullets；

2）在上述循环中，再次使用循环，遍历存储所有飞行物(蜜蜂或敌飞机)的数组 flyings，在该循环中判断当前子弹是否击中某个飞行物(蜜蜂或敌飞机)，如果击中则退出该循环，记录被击中的飞行物在 flyings 数组中的索引 index；

3）在 flyings 数据中找到该飞行物，并将其移除；

4）判断该飞行物的类型是 Enemy 还是 Award，如果是 Enemy 类型，则获取加分；如果是 Award 类型，则获取奖励；

5）获取奖励的类型，如果奖励的类型为 DOUBLE_FIRE，则获得 20 次双倍火力；如果奖励的类型为 LIFE，则增命，代码如下所示：

```
/** 子弹与飞行物碰撞检测 */
public void bangAction() {
    for (int i = 0; i < bullets.length; i++) { // 遍历所有子弹
        Bullet b = bullets[i];
        bang(b);
    }
}
/** 子弹和飞行物之间的碰撞检查 */
public void bang(Bullet bullet) {
    int index = -1; // 击中的飞行物索引
    for (int i = 0; i < flyings.length; i++) {
        FlyingObject obj = flyings[i];
        if (obj.shootBy(bullet)) { // 判断是否击中
            index = i; // 记录被击中的飞行物的索引
            break;
        }
    }
    if (index != -1) { // 有击中的飞行物
        FlyingObject one = flyings[index]; // 记录被击中的飞行物
```

```
            FlyingObject temp = flyings[index]; // 被击中的飞行物与最后一个飞行物交
换
            flyings[index] = flyings[flyings.length - 1];
            flyings[flyings.length - 1] = temp;
            // 删除最后一个飞行物(即被击中的)
            flyings = Arrays.copyOf(flyings, flyings.length - 1);

            // 检查 one 的类型 如果是敌人，就算分
            if (one instanceof Enemy) { // 检查类型，是敌人，则加分
                Enemy e = (Enemy) one; // 强制类型转换
                score += e.getScore(); // 加分
            }
            if (one instanceof Award) { // 若为奖励，设置奖励
                Award a = (Award) one;
                int type = a.getType(); // 获取奖励类型
                switch (type) {
                case Award.DOUBLE FIRE:
                    hero.addDoubleFire(); // 设置双倍火力
                    break;
                case Award.LIFE:
                    hero.addLife(); // 设置加命
                    break;
                }
            }
        }
    }
```

7．在 Action 方法中调用 bangAction 方法，代码如下所示：

```
public void action() { // 启动执行代码
    timer = new Timer(); // 主流程控制
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
                enterAction(); // 飞行物入场
                stepAction(); // 走一步
                shootAction(); // 射击

                bangAction();

                repaint(); // 重绘，调用 paint()方法
        }
    }, intervel, intervel);
}
```

步骤二十一：实现画分数和命数

1.在 Hero 类中，添加 getLife 方法，该方法用于获取英雄机的命数，代码如下所示：

```
public int getLife() {
    return life;
}
```

2.在 ShootGame 类中，添加 paintScore 方法，该方法用于画分数和命数，代码如下所示：

```
/** 画分数 */
public void paintScore(Graphics g) {
    int x = 10;
    int y = 25;
    Font font = new Font(Font.SANS_SERIF,Font.BOLD, 14);
    g.setColor(new Color(0x3A3B3B));
    g.setFont(font); // 设置字体
    g.drawString("SCORE:" + score, x, y); // 画分数
    y+=20;
    g.drawString("LIFE:" + hero.getLife(), x, y);
}
```

3.在 ShootGame 类的 paint 方法中，调用 paintScore 方法，代码如下所示：

```
@Override
public void paint(Graphics g) {
    g.drawImage(background, 0, 0, null); // 画背景图
    paintHero(g); // 画英雄机
    paintBullets(g); // 画子弹
    paintFlyingObjects(g); // 画飞行物

    paintScore(g);//画分数

}
```

步骤二十二：删除越界飞行物（蜜蜂和敌飞机）和子弹

1.由于蜜蜂、敌飞机、子弹都可能出现越界现象，因此，在 FlyingObject 类中添加抽象方法 outOfBounds，根据子类不同实现具体的越界算法，代码如下所示：

```
/**
 * 检查是否出界
 * @param width  边界宽
 * @param height 边界高
 * @return true 出界与否
 */
public abstract boolean outOfBounds();
```

2.在 Bee 类中，实现父类 FlyingObject 的越界判断方法，蜜蜂是向下飞行的，下越界的临界状态如图-17 所示：

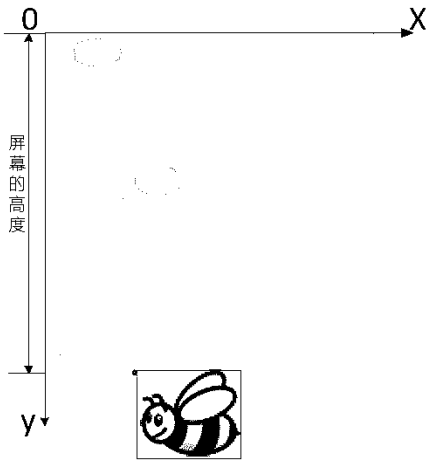

图 - 17

由图-17 可以看出，当蜜蜂的 y 坐标大于屏幕的高度时，蜜蜂超出了边界 ，代码如下所示：

```
@Override
public boolean outOfBounds() {
    return y>ShootGame.HEIGHT;
}
```

3．在 Airplane 类中，实现父类 FlyingObject 的越界判断方法，敌飞机上下越界的临界状态与蜜蜂相同，代码如下所示：

```
@Override
public  boolean outOfBounds() {   //越界处理
    return y>ShootGame.HEIGHT;
}
```

4．在 Bullet 类中，实现父类 FlyingObject 的越界判断方法，子弹是向上运动的，子弹上越界的临界状态如图-18 所示：

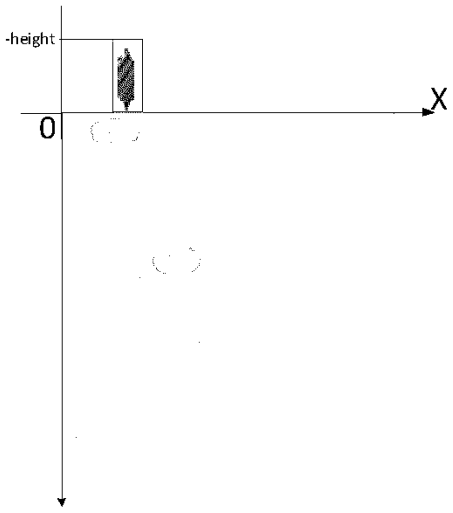

图 - 18

由图-18 可以看出，当子弹的 y 坐标小于子弹的负高度时，子弹超出了边界 ，代码如下所示：

```
@Override
public boolean outOfBounds() {
    return y<-height;
}
```

5.在 ShootGame 类中添加 outOfBoundsAction 方法 ,该方法用于删除越界飞行物(蜜蜂和敌飞机) 及子弹，详细实现过程如下：

1) 创建 FlyingObject 类型的数组 flyingLives，用于存储所有活着的飞行物 (蜜蜂和敌飞机)，即没有越界的飞行物 (蜜蜂和敌飞机)；

2) 循环遍历存储所有飞行物 (蜜蜂和敌飞机) 的数组 flyings，并判断每一个飞行物 (蜜蜂或敌飞机) 是否越界，将没有越界的飞行物 (蜜蜂或敌飞机) 放入 flyingLives 数组中存储，并记录不越界飞行物的个数 index；

3) 将 flyingLives 数组中的元素，复制到 flyings 数组中，并重新指定 flying 数组的长度为 index；

4) 删除子弹与删除飞行物 (蜜蜂和敌飞机) 的过程类似。

outOfBoundsAction 方法的代码如下所示：

```
/** 删除越界飞行物及子弹 */
public void outOfBoundsAction() {
    int index = 0;
    // 存储活着的飞行物
    FlyingObject[] flyingLives = new FlyingObject[flyings.length];
    for (int i = 0; i < flyings.length; i++) {
        FlyingObject f = flyings[i];
        if (!f.outOfBounds()) {
            flyingLives[index++] = f; // 不越界的留着
        }
    }
    flyings = Arrays.copyOf(flyingLives, index); // 将不越界的飞行物都留着

    index = 0; // 重置为 0
    Bullet[] bulletLives = new Bullet[bullets.length];
    for (int i = 0; i < bullets.length; i++) {
        Bullet b = bullets[i];
        if (!b.outOfBounds()) {
            bulletLives[index++] = b;
        }
    }
    bullets = Arrays.copyOf(bulletLives, index); // 将不越界的子弹留着
}
```

6.在 ShootAction 类的 action 方法中调用 outOfBoundsAction，代码如下所示：

```
public void action() { // 启动执行代码
    // 鼠标监听事件
    MouseAdapter l = new MouseAdapter() {
        @Override
```

```
        public void mouseMoved(MouseEvent e) { // 鼠标移动
                int x = e.getX();
                int y = e.getY();
                hero.moveTo(x, y);
        }
    };
    this.addMouseMotionListener(l); // 处理鼠标滑动操作

    timer = new Timer(); // 主流程控制
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
                enterAction(); // 飞行物入场
                stepAction(); // 走一步
                shootAction(); // 射击
                bangAction();

                outOfBoundsAction(); // 删除越界飞行物及子弹

                repaint(); // 重绘，调用paint()方法
        }
    }, intervel, intervel);
}
```

步骤二十三：判断英雄机是否与飞行物（蜜蜂和敌飞机）碰撞

1.当英雄机与飞行物（蜜蜂和敌飞机）发生碰撞时，需要减少命的数量以及将双倍火力清零，因此，在 Hero 类中添加 subtractLife 方法，用于实现减命；添加 setDoubleFire 用于重新设置双倍火力的值，代码如下所示：

```
public void subtractLife() { // 减命
    life--;
}
public void setDoubleFire(int doubleFire) {
    this.doubleFire = doubleFire;
}
```

2. 在 Hero 类中添加 hit 方法用于英雄机与飞行物（蜜蜂和敌飞机）的碰撞检测，图-19 以蜜蜂为例说明了碰撞算法。

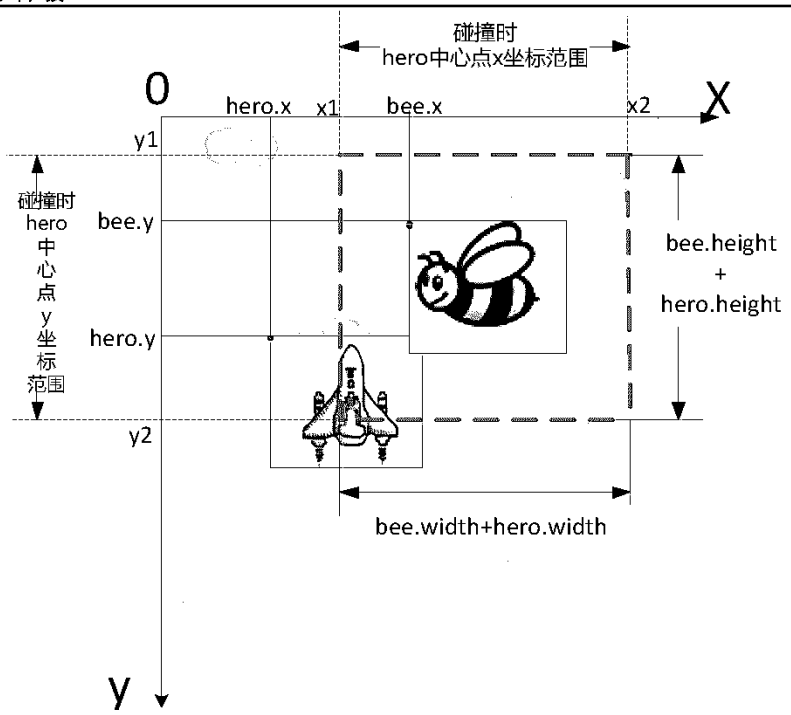

图 - 19

由图-19 可以看出，x1、x2 以及 y1、y2 的坐标算法如下：

```
x1=bee.x-1/2hero.width
x2=bee.x+1/2hero.width+bee.width

y1=bee.y-1/2hero.height
y2=bee.y+1/2hero.height+bee.height
```

英雄机中心点的 x、y 坐标算法如下：

```
hero.中心点 x=hero.x+1/2hero.width
hero.中心点 y=hero.y+1/2hero.height
```

当满足如下条件时，英雄机与蜜蜂发生碰撞：

```
x1<Hero.中心点 x<x2
y1<Hero.中心点 y<y2
```

hit 方法的实现代码如下所示：

```
public boolean hit(FlyingObject other) { // 碰撞算法
    int x1 = other.x - this.width / 2;
    int x2 = other.x + other.width + this.width / 2;
    int y1 = other.y - this.height / 2;
    int y2 = other.y + other.height + this.height / 2;
    return this.x + this.width / 2 > x1 && this.x + this.width / 2 < x2
           && this.y + this.height / 2 > y1
           && this.y + this.width / 2 < y2;
```

```
}
```

3．在 ShootGame 类中，添加 isGameOver 方法，该方法用于判断游戏是否结束，方法实现的详细过程如下：

1） 循环遍历存储所有飞行物（蜜蜂和敌飞机）的数组 flyings；

2） 在循环中，判断英雄机是否与某个飞行物（蜜蜂和敌飞机）发生碰撞，如果发生碰撞，则减命、双倍火力清零，并记录被撞飞行物在 flyings 数组中的索引 index，该索引默认为-1，即没有发生碰撞；

3）判断 index 是否为-1，如果不为-1，将该索引位置的元素从 flyings 数组中清除；

4）判断命数是否小于等于 0，并返回比较结果。

```
/** 检查游戏是否结束 */
public boolean isGameOver() {
    for (int i = 0; i < flyings.length; i++) {
       int index = -1;
        FlyingObject obj = flyings[i];
        if (hero.hit(obj)) { // 检查英雄机与飞行物是否碰撞
            hero.subtractLife();
            hero.setDoubleFire(0);
            index = i;
        }
       if(index!=-1){
        FlyingObject t = flyings[index];
        flyings[index] = flyings[flyings.length-1];
        flyings[flyings.length-1] = t;
        flyings = Arrays.copyOf(flyings, flyings.length-1);
       }
    }

    return  hero.getLife() <= 0;
}
```

步骤二十四：实现游戏的开始、运行、暂停以及结束

游戏分为四种状态，分别为 START、RUNNING、PAUSE、GAME_OVER，表示游戏开始状态、运行状态、暂停状态以及游戏结束状态。

首先介绍一下鼠标事件对状态的影响，当执行鼠标点击事件时，会对游戏中的 START 状态、GAME_OVER 状态产生影响。如果点击鼠标时为 START 状态，则将游戏的状态设置为 RUNNING，即点击鼠标游戏进入运行状态；如果点击鼠标时为 GAME_OVER 状态，则将 flyings 数组、bullets 数组、hero 对象、score 变量设置为初始状态，并将状态设置为 START 状态。代码如下：

```
flyings = new FlyingObject[0];
bullets = new Bullet[0];
hero = new Hero();
score = 0;
state = START;
```

当鼠标执行移动事件时，判断状态是否 RUNNING 状态，如果为 RUNNING，则执行英雄机跟随鼠标移动的方法。

当鼠标执行进入事件时，判断状态是否 PAUSE，如果为 PAUSE 状态，则状态更改为 RUNNING。

当鼠标执行退出事件时，判断状态是否 GAME_OVER 状态，如果不为 GAME_OVER 状态，则状态更改为 PAUSE。

然后，当游戏状态为 RUNNING 状态时，执行飞行物入场、所有飞行物走一步、射击、子弹打飞行物、删除越界飞行物及子弹、检查游戏结束这一系列动作，代码如下：

```
if (state == RUNNING) {
    enterAction(); // 飞行物入场
    stepAction(); // 走一步
    shootAction(); // 射击
    bangAction(); // 子弹打飞行物
    outOfBoundsAction(); // 删除越界飞行物及子弹
    checkGameOverAction(); // 检查游戏结束
}
```

最后，如果判断游戏已经结束，那么将游戏状态设置为 GAME_OVER，代码如下：

```
/** 检查游戏结束 */
public void checkGameOverAction() {
    if (isGameOver()) {
        state = GAME_OVER; // 改变状态
    }
}
```

具体实现步骤如下：

1.在 ShootGame 类中添加以下属性和常量，代码如下所示：

```
private int state;
public static final int START = 0;
public static final int RUNNING = 1;
public static final int PAUSE = 2;
public static final int GAME_OVER = 3;
```

2. 在 ShootGame 类中添加 checkGameOverAction 方法，该方法用于判断游戏是否已经结束，如果已经结束，则将游戏状态设置为 GAME_OVER，代码如下所示：

```
/** 检查游戏结束 */
public void checkGameOverAction() {
    if (isGameOver()) {
        state = GAME_OVER; // 改变状态
    }
}
```

3.修改 ShootGame 类的 action 方法，添加鼠标点击、移入、退出等操作的状态处理，代码如下所示：

```
public void action() { // 启动执行代码
    // 鼠标监听事件
    MouseAdapter l = new MouseAdapter() {
        @Override
        public void mouseMoved(MouseEvent e) { // 鼠标移动

            if (state == RUNNING) { // 运行时移动英雄机

                int x = e.getX();
                int y = e.getY();
                hero.moveTo(x, y);

            }

        }

        @Override
        public void mouseEntered(MouseEvent e) { // 鼠标进入
            if (state == PAUSE) { // 暂停时运行
                state = RUNNING;
            }
        }

        @Override
        public void mouseExited(MouseEvent e) { // 鼠标退出
            if (state != GAME_OVER) {
                state = PAUSE; // 游戏未结束，则设置其为暂停
            }
        }

        @Override
        public void mouseClicked(MouseEvent e) { // 鼠标点击
            switch (state) {
            case START:
                state = RUNNING;
                break;
            case GAME_OVER: // 游戏结束，清理现场
                flyings = new FlyingObject[0];
                bullets = new Bullet[0];
                hero = new Hero();
                score = 0;
                state = START;
                break;
            }
        }

    };

    this.addMouseListener(l); // 处理鼠标点击操作

    this.addMouseMotionListener(l); // 处理鼠标滑动操作

    timer = new Timer(); // 主流程控制
    timer.schedule(new TimerTask() {
        @Override
```

```
            public void run() {
                if (state == RUNNING) {
                    enterAction(); // 飞行物入场
                    stepAction(); // 走一步
                    shootAction(); // 射击
                    bangAction(); // 子弹打飞行物
                    outOfBoundsAction(); // 删除越界飞行物及子弹

                    checkGameOverAction(); // 检查游戏结束

                }
                repaint(); // 重绘，调用paint()方法
            }
        }, intervel, intervel);
    }
```

4．在 ShootGame 类中添加 paintState 方法，画出 START、PAUSE 以及 GAME_OVER 状态显示的图片，代码如下所示：

```
/** 画游戏状态 */
public void paintState(Graphics g) {
    switch (state) {
    case START:
        g.drawImage(start, 0, 0, null);
        break;
    case PAUSE:
        g.drawImage(pause, 0, 0, null);
        break;
    case GAME OVER:
        g.drawImage(gameover, 0, 0, null);
        break;
    }
}
```

5．在 ShootGame 类中 paint 方法中，调用 paintState 方法，代码如下所示：

```
@Override
public void paint(Graphics g) {
    g.drawImage(background, 0, 0, null); // 画背景图
    paintHero(g); // 画英雄机
    paintBullets(g); // 画子弹
    paintFlyingObjects(g); // 画飞行物
    paintScore(g); // 画分数

    paintState(g); // 画游戏状态

}
```

4. 完整代码

Airplane 类的完整代码如下所示：

```
package com.tarena.shoot;

import com.tarena.shoot.ShootGame;

/**
 * 敌飞机：是飞行物，也是敌人
 */
public class Airplane extends FlyingObject implements Enemy {
 private int speed = 2;

 /** 初始化数据 */
 public Airplane(){
     this.image = ShootGame.airplane;
     width = image.getWidth();
     height = image.getHeight();
     y = -height;
     x = (int)(Math.random()*(ShootGame.WIDTH - width));
//       y=100;
//       x=100;
 }

 public int getScore() {
     return 5;
 }

 @Override
 public void step() {   //移动
     y += speed;
 }

 @Override
 public  boolean outOfBounds() {   //越界处理
     return y>ShootGame.HEIGHT;
 }

}
```

Award 类的完整代码如下所示：

```
package com.tarena.shoot;
/**
 * 奖励
 */
public interface Award {
 int DOUBLE_FIRE = 0;  //双倍火力
 int LIFE = 1;   //1 条命
 /** 获得奖励类型(上面的 0 或 1) */
 int getType();
}
```

Bee 类的完整代码如下所示：

```
package com.tarena.shoot;

import java.util.Random;


/** 蜜蜂 */
public class Bee extends FlyingObject implements Award{
 private int xSpeed = 1;   //x 坐标移动速度
 private int ySpeed = 2;   //y 坐标移动速度
 private int awardType;    //奖励类型
```

```
    public Bee(){
        this.image = ShootGame.bee;
        width = image.getWidth();
        height = image.getHeight();
        y = -height;
        Random rand = new Random();
        x = rand.nextInt(ShootGame.WIDTH - width);
//      x=100;
//      y=200;
        awardType = rand.nextInt(2);
    }
    public int getType() {
        return awardType;
    }
    @Override
    public void step() {      //可斜飞
        x += xSpeed;
        y += ySpeed;
        if(x > ShootGame.WIDTH-width){
            xSpeed = -1;
        }
        if(x < 0){
            xSpeed = 1;
        }
    }
    @Override
    public boolean outOfBounds() {
        return y>ShootGame.HEIGHT;
    }

}
```

Bullet 类的完整代码如下所示：

```
package com.tarena.shoot;
/**
 * 子弹类:是飞行物
 */
public class Bullet extends FlyingObject {
    private int speed = 3;  //移动的速度
    public Bullet(int x,int y){
        this.x = x;
        this.y = y;
        this.image = ShootGame.bullet;
    }
    @Override
    public void step(){   //移动方法
        y-=speed;
    }

    @Override
    public boolean outOfBounds() {
        return y<-height;
    }
}
```

Enemy 类的完整代码如下所示：

```
package com.tarena.shoot;

/**
 * 敌人，可以有分数
 */
```

```
public interface Enemy {
 /** 敌人的分数  */
 int getScore();
}
```

FlyingObject 类的完整代码如下所示：

```
package com.tarena.shoot;

import java.awt.image.BufferedImage;

public abstract class FlyingObject {
 protected int x;    //x 坐标
 protected int y;    //y 坐标
 protected int width;    //宽
 protected int height;   //高
 protected BufferedImage image;   //图片

 public int getX() {
     return x;
 }

 public void setX(int x) {
     this.x = x;
 }

 public int getY() {
     return y;
 }

 public void setY(int y) {
     this.y = y;
 }

 public int getWidth() {
     return width;
 }

 public void setWidth(int width) {
     this.width = width;
 }

 public int getHeight() {
     return height;
 }

 public void setHeight(int height) {
     this.height = height;
 }

 public BufferedImage getImage() {
     return image;
 }

 public void setImage(BufferedImage image) {
     this.image = image;
 }
 /**
  * 飞行物移动一步
  */
 public abstract void step();

 /**
  * 检查当前飞行物体是否被子弹(x,y)击(shoot)中,
```

```
     * true 表示击中，飞行物可以被击中
     * @param Bullet 子弹对象
     * @return true 表示被击中了
     */
    public boolean shootBy(Bullet bullet){
        int x = bullet.x;  //子弹横坐标
        int y = bullet.y;  //子弹纵坐标
        return this.x<x && x<this.x+width && this.y<y && y<this.y+height;
    }

    /**
     * 检查是否出界
     * @param width  边界宽
     * @param height 边界高
     * @return true 出界与否
     */
    public abstract boolean outOfBounds();

}
```

Hero 类的完整代码如下所示：

```
package com.tarena.shoot;

import java.awt.image.BufferedImage;

/**
 * 英雄机:是飞行物
 */
public class Hero extends FlyingObject {
    protected BufferedImage[] images = {};
    protected int index = 0;

    private int doubleFire;
    private int life;

    public Hero() {
        life = 3;
        doubleFire = 0;
        this.image = ShootGame.hero0;
        images = new BufferedImage[] { ShootGame.hero0, ShootGame.hero1 };
        width = image.getWidth();
        height = image.getHeight();
        x = 150;
        y = 400;
    }

    @Override
    public void step() {
        if (images.length > 0) {
            image = images[index++ / 10 % images.length];
        }
    }

    public Bullet[] shoot() { // 发射子弹
        int xStep = width / 4;
        int yStep = 20;
        if (doubleFire > 0) {
            Bullet[] bullets = new Bullet[2];
            bullets[0] = new Bullet(x + xStep, y - yStep);
            bullets[1] = new Bullet(x + 3 * xStep, y - yStep);
            doubleFire -= 2;
            return bullets;
        } else { // 单倍
```

```
            Bullet[] bullets = new Bullet[1];
            // y-yStep(子弹距飞机的位置)
            bullets[0] = new Bullet(x + 2 * xStep, y - yStep);
            return bullets;
        }
    }

    public void addDoubleFire() {
        doubleFire += 40;
    }

    public void setDoubleFire(int doubleFire) {
        this.doubleFire = doubleFire;
    }

    public void addLife() { // 增命
        life++;
    }

    public void subtractLife() { // 减命
        life--;
    }

    public int getLife() {
        return life;
    }
    /**
     * 当前物体移动了一下，相对距离， x,y 鼠标位置
     */
    public void moveTo(int x, int y) {
        this.x = x - width / 2;
        this.y = y - height / 2;
    }

    @Override
    public boolean outOfBounds() {
        return false;
    }

    public boolean hit(FlyingObject other) { // 碰撞算法

        int x1 = other.x - this.width / 2;
        int x2 = other.x + other.width + this.width / 2;
        int y1 = other.y - this.height / 2;
        int y2 = other.y + other.height + this.height / 2;
        return this.x + this.width / 2 > x1 && this.x + this.width / 2 < x2
                && this.y + this.height / 2 > y1
                && this.y + this.width / 2 < y2;
    }

}
```

ShootGame 类的完整代码如下所示：

```
package com.tarena.shoot;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.util.Arrays;
import java.util.Random;
import java.util.Timer;
```

```
import java.util.TimerTask;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class ShootGame extends JPanel {
    public static final int WIDTH = 400; // 面板宽
    public static final int HEIGHT = 654; // 面板高

    /** 游戏的当前状态: START RUNNING PAUSE GAME OVER */
    private int state;
    public static final int START = 0;
    public static final int RUNNING = 1;
    public static final int PAUSE = 2;
    public static final int GAME_OVER = 3;

    private int score = 0; // 得分
    private Timer timer; // 定时器
    private int intervel = 1000 / 100; // 时间间隔(毫秒)

    public static BufferedImage background;
    public static BufferedImage start;
    public static BufferedImage airplane;
    public static BufferedImage bee;
    public static BufferedImage bullet;
    public static BufferedImage hero0;
    public static BufferedImage hero1;
    public static BufferedImage pause;
    public static BufferedImage gameover;

    private FlyingObject[] flyings = {}; // 敌机数组
    private Bullet[] bullets = {}; // 子弹数组
    private Hero hero = new Hero(); // 英雄机

    public ShootGame() {
        // 初始化一只蜜蜂一架飞机
        // flyings=new FlyingObject[2];
        // flyings[0]=new Airplane();
        // flyings[1]=new Bee();
        // 初始化一颗子弹
        // bullets=new Bullet[1];
        // bullets[0]=new Bullet(200,350);
    }

    static {// 静态代码块
        try {
            background = ImageIO.read(ShootGame.class
                    .getResource("background.png"));
            airplane = ImageIO
                    .read(ShootGame.class.getResource("airplane.png"));
            bee = ImageIO.read(ShootGame.class.getResource("bee.png"));
            bullet = ImageIO.read(ShootGame.class.getResource("bullet.png"));
            hero0 = ImageIO.read(ShootGame.class.getResource("hero0.png"));
            hero1 = ImageIO.read(ShootGame.class.getResource("hero1.png"));
            pause = ImageIO.read(ShootGame.class.getResource("pause.png"));
            gameover = ImageIO
                    .read(ShootGame.class.getResource("gameover.png"));
            start = ImageIO.read(ShootGame.class.getResource("start.png"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void paint(Graphics g) {
        g.drawImage(background, 0, 0, null); // 画背景图
```

```
        paintHero(g); // 画英雄机
        paintBullets(g); // 画子弹
        paintFlyingObjects(g); // 画飞行物
        paintScore(g);
        paintState(g); // 画游戏状态
    }

    /** 画英雄机 */
    public void paintHero(Graphics g) {
        g.drawImage(hero.getImage(), hero.getX(), hero.getY(), null);
    }

    /** 画子弹 */
    public void paintBullets(Graphics g) {
        for (int i = 0; i < bullets.length; i++) {
            Bullet b = bullets[i];
            g.drawImage(b.getImage(), b.getX(), b.getY(), null);
        }
    }

    /** 画飞行物 */
    public void paintFlyingObjects(Graphics g) {
        for (int i = 0; i < flyings.length; i++) {
            FlyingObject f = flyings[i];
            g.drawImage(f.getImage(), f.getX(), f.getY(), null);
        }
    }

    /** 画分数 */
    public void paintScore(Graphics g) {
        int x = 10;
        int y = 25;
        Font font = new Font(Font.SANS SERIF, Font.BOLD, 14);
        g.setColor(new Color(0x3A3B3B));
        g.setFont(font); // 设置字体
        g.drawString("SCORE:" + score, x, y); // 画分数
        y += 20;
        g.drawString("LIFE:" + hero.getLife(), x, y);
    }

    /** 画游戏状态 */
    public void paintState(Graphics g) {
        switch (state) {
        case START:
            g.drawImage(start, 0, 0, null);
            break;
        case PAUSE:
            g.drawImage(pause, 0, 0, null);
            break;
        case GAME OVER:
            g.drawImage(gameover, 0, 0, null);
            break;
        }
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Fly");
        ShootGame game = new ShootGame(); // 面板对象
        frame.add(game); // 将面板添加到 JFrame 中
        frame.setSize(WIDTH, HEIGHT); // 大小
        frame.setAlwaysOnTop(true); // 其总在最上
        frame.setDefaultCloseOperation(JFrame.EXIT ON CLOSE); // 默认关闭操作
        frame.setLocationRelativeTo(null); // 设置窗体初始位置
        frame.setVisible(true); // 尽快调用 paint
        game.action(); // 启动执行
    }
```

```
public void action() { // 启动执行代码
    // 鼠标监听事件
    MouseAdapter l = new MouseAdapter() {
        @Override
        public void mouseMoved(MouseEvent e) { // 鼠标移动
            if (state == RUNNING) { // 运行时移动英雄机
                int x = e.getX();
                int y = e.getY();
                hero.moveTo(x, y);
            }
        }
        @Override
        public void mouseEntered(MouseEvent e) { // 鼠标进入
            if (state == PAUSE) { // 暂停时运行
                state = RUNNING;
            }
        }
        @Override
        public void mouseExited(MouseEvent e) { // 鼠标退出
            if (state != GAME_OVER) {
                state = PAUSE; // 游戏未结束，则设置其为暂停
            }
        }
        @Override
        public void mouseClicked(MouseEvent e) { // 鼠标点击
            switch (state) {
            case START:
                state = RUNNING;
                break;
            case GAME_OVER: // 游戏结束，清理现场
                flyings = new FlyingObject[0];
                bullets = new Bullet[0];
                hero = new Hero();
                score = 0;
                state = START;
                break;
            }
        }
    };
    this.addMouseListener(l); // 处理鼠标点击操作
    this.addMouseMotionListener(l); // 处理鼠标滑动操作

    timer = new Timer(); // 主流程控制
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
            if (state == RUNNING) {
                enterAction(); // 飞行物入场
                stepAction(); // 走一步
                shootAction(); // 射击
                bangAction(); // 子弹打飞行物
                outOfBoundsAction(); // 删除越界飞行物及子弹
                checkGameOverAction(); // 检查游戏结束
            }
            repaint(); // 重绘，调用paint()方法
        }

    }, intervel, intervel);
}


/**
 * 随机生成飞行物
 *
```

```
     * @return 飞行物对象
     */
    public static FlyingObject nextOne() {
        Random random = new Random();
        int type = random.nextInt(20); // [0,19)
        if (type == 0) {
            return new Bee();
        } else {
            return new Airplane();
        }
    }

    int flyEnteredIndex = 0; // 飞行物入场计数

    /** 飞行物入场 */
    public void enterAction() {
        flyEnteredIndex++;
        if (flyEnteredIndex % 40 == 0) { // 400毫秒--10*40
            FlyingObject obj = nextOne(); // 随机生成一个飞行物
            flyings = Arrays.copyOf(flyings, flyings.length + 1);
            flyings[flyings.length - 1] = obj;
        }
    }

    public void stepAction() {
        /** 飞行物走一步 */
        for (int i = 0; i < flyings.length; i++) {
            FlyingObject f = flyings[i];
            f.step();
        }

        /** 子弹走一步 */
        for (int i = 0; i < bullets.length; i++) {
            Bullet b = bullets[i];
            b.step();
        }
        hero.step();
    }

    int shootIndex = 0; // 射击计数

    /** 射击 */
    public void shootAction() {
        shootIndex++;
        if (shootIndex % 30 == 0) { // 100毫秒发一颗
            Bullet[] bs = hero.shoot(); // 英雄打出子弹
            bullets = Arrays.copyOf(bullets, bullets.length + bs.length); // 扩
容
            System.arraycopy(bs, 0, bullets, bullets.length - bs.length,
                    bs.length); // 追加数组
        }
    }

    /** 子弹与飞行物碰撞检测 */
    public void bangAction() {
        for (int i = 0; i < bullets.length; i++) { // 遍历所有子弹
            Bullet b = bullets[i];
            bang(b);
        }
    }

    /** 子弹和飞行物之间的碰撞检查 */
    public void bang(Bullet bullet) {
        int index = -1; // 击中的飞行物索引
        for (int i = 0; i < flyings.length; i++) {
```

```
            FlyingObject obj = flyings[i];
            if (obj.shootBy(bullet)) { // 判断是否击中
                index = i; // 记录被击中的飞行物的索引
                break;
            }
        }
        if (index != -1) { // 有击中的飞行物
            FlyingObject one = flyings[index]; // 记录被击中的飞行物

            FlyingObject temp = flyings[index]; // 被击中的飞行物与最后一个飞行物交
换
            flyings[index] = flyings[flyings.length - 1];
            flyings[flyings.length - 1] = temp;

            flyings = Arrays.copyOf(flyings, flyings.length - 1); // 删除最后一
个飞行物(即被击中的)

            // 检查 one 的类型 如果是敌人, 就算分
            if (one instanceof Enemy) { // 检查类型, 是敌人, 则加分
                Enemy e = (Enemy) one; // 强制类型转换
                score += e.getScore(); // 加分
            }
            if (one instanceof Award) { // 若为奖励, 设置奖励
                Award a = (Award) one;
                int type = a.getType(); // 获取奖励类型
                switch (type) {
                case Award.DOUBLE_FIRE:
                    hero.addDoubleFire(); // 设置双倍火力
                    break;
                case Award.LIFE:
                    hero.addLife(); // 设置加命
                    break;
                }
            }
        }
    }

    /** 删除越界飞行物及子弹 */
    public void outOfBoundsAction() {
        int index = 0;
        FlyingObject[] flyingLives = new FlyingObject[flyings.length]; // 活着
的飞行物
        for (int i = 0; i < flyings.length; i++) {
            FlyingObject f = flyings[i];
            if (!f.outOfBounds()) {
                flyingLives[index++] = f; // 不越界的留着
            }
        }
        flyings = Arrays.copyOf(flyingLives, index); // 将不越界的飞行物都留着

        index = 0; // 重置为 0
        Bullet[] bulletLives = new Bullet[bullets.length];
        for (int i = 0; i < bullets.length; i++) {
            Bullet b = bullets[i];
            if (!b.outOfBounds()) {
                bulletLives[index++] = b;
            }
        }
        bullets = Arrays.copyOf(bulletLives, index); // 将不越界的子弹留着
    }

    /** 检查游戏结束 */
    public void checkGameOverAction() {
        if (isGameOver()) {
```

```
            state = GAME_OVER; // 改变状态
        }
    }

    /** 检查游戏是否结束 */
    public boolean isGameOver() {
        for (int i = 0; i < flyings.length; i++) {
            int index = -1;
            FlyingObject obj = flyings[i];
            if (hero.hit(obj)) { // 检查英雄机与飞行物是否碰撞
                hero.subtractLife();
                hero.setDoubleFire(0);
                index = i;
            }
            if (index != -1) {
                FlyingObject t = flyings[index];
                flyings[index] = flyings[flyings.length - 1];
                flyings[flyings.length - 1] = t;
                flyings = Arrays.copyOf(flyings, flyings.length - 1);
            }
        }
        return hero.getLife() <= 0;
    }
}
```

飞扬的小鸟

1. 问题

这款游戏的起源是越南独立开发者开发的手机游戏，短时间竟占领了全球各大 App Store 免费排行榜首位。游戏中，玩家控制一只小鸟飞过一个个柱子的间隙。飞得越远分数越高，看玩家能使小鸟在空中坚持多久。初始界面如图-1 所示：

图 - 1

玩家在如图-1 所示的界面的任意位置，按下鼠标左键，开始游戏。

游戏开始以后，玩家需要不断控制点击屏幕的频率来调节小鸟的飞行高度和降落速度，让小鸟顺利的通过画面右端的柱子间隙。如果玩家不小心擦碰到了柱子或掉落到地面上，则游戏宣告结束。如图-2 所示：

图 – 2

如图-2 所示的左上角显示了用户的得分，每通过一个柱子的间隙得 1 分。另外，此时玩家可以在如图-2 所示的界面的任意位置，按下鼠标左键，重新开始游戏。

2. 方案

软件的开发过程如下：

1．需求(软件功能的文字描述)

2．业务需求分析：找对象，以及对象之间的关系。本项目中对象如下所示：

```
BirdGame
   |-- 地面 Ground
   |-- 小鸟 Bird
   |-- 多个柱子 Column
```

3．软件概要设计：计算机只能按顺序“处理数据”。

数据建模：使用一个数据模型，描述对象的关系。使用绘图坐标系作为参考模型，鸟是正方形区域；柱子是长方形区域，中间有间隙；地面是矩形。 对象是结构化的“数据”，是一组有关系的数据。

4．类的设计

本案例中的类，及类之间的关系如图-3 所示：

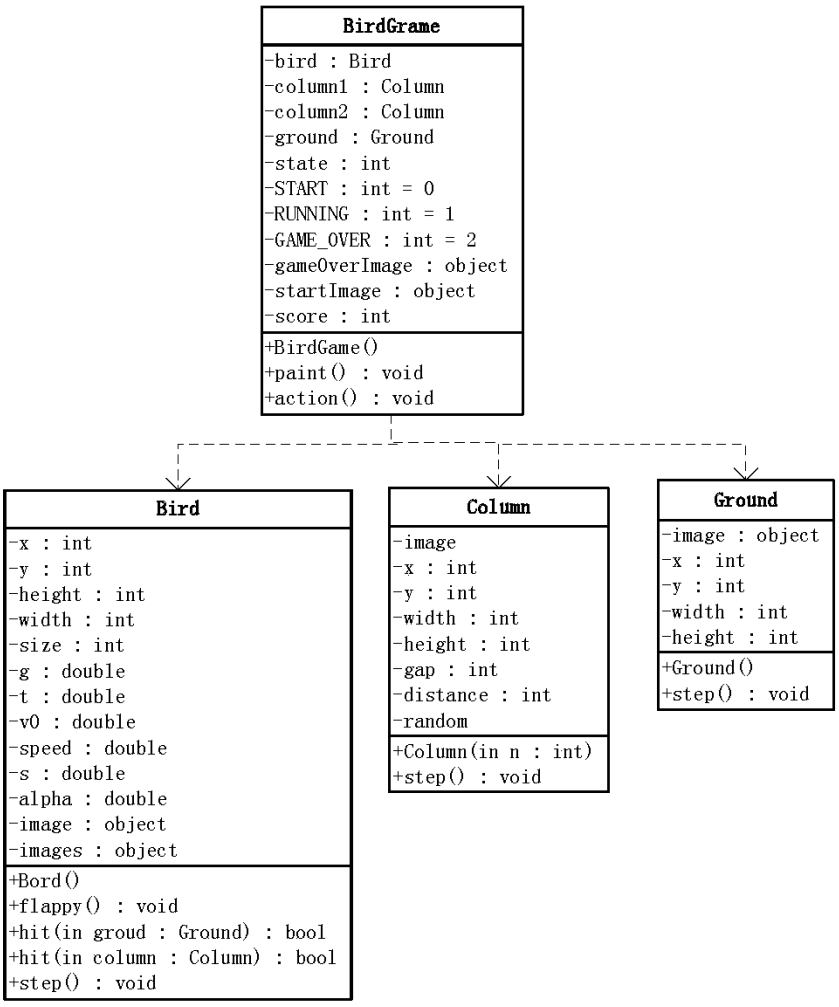

图 - 3

3. 步骤

实现此案例需要按照如下步骤进行。

步骤一：新建工程和包

新建名为 BirdGrame 的 Java 工程；然后，在工程下的 src 目录下新建包 com.tarena.bird； 最后，将该工程所需的图片拷贝到该包下，工程结构如图-4 所示：

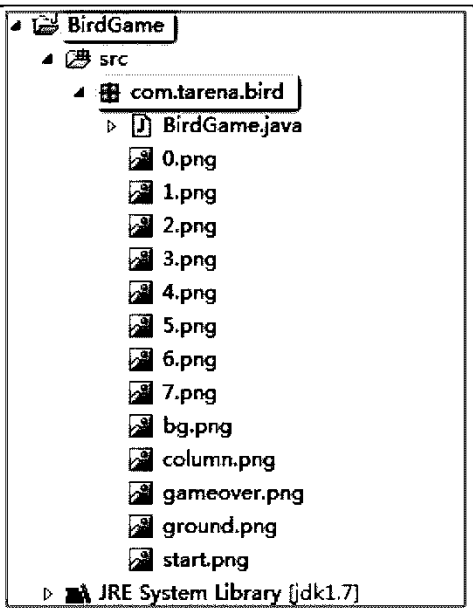

图- 4

在项目中，包的名字一般为公司域名倒过来，再加上项目名称，即为包名。如图-4 中中的包名为 com.tarena.bird，其中，com.tarena 是达内公司的域名倒过来，bird 为本项目的名称。

步骤二：构建工程的结构

首先，新建类 BirdGame，该类由 public 修饰并且该类继承自 JPanel 类，扩展 Java AWT 的面板功能；然后，新建 Groud 类，表示游戏中的地面；新建 Column 表示游戏中的柱子；新建 Bird 表示游戏中的鸟，游戏中的各个对象如图-5 所示：

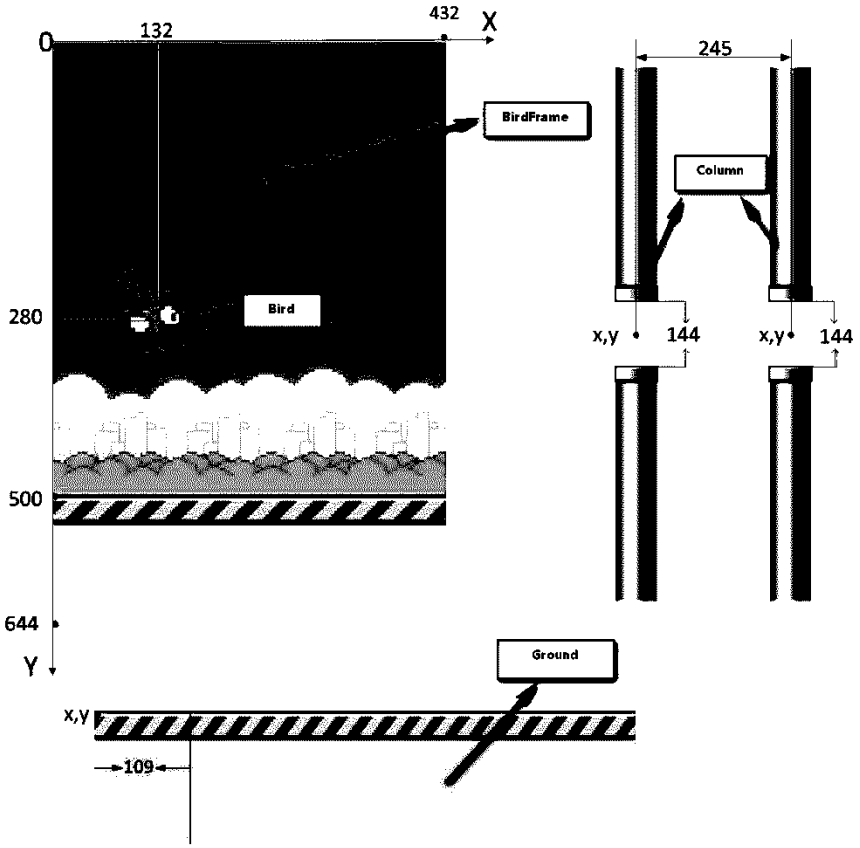

图 - 5

代码如下所示：

```
package com.tarena.bird;

import javax.swing.JPanel;

public class BirdGame extends JPanel {
 /** 启动软件的方法 */
 public static void main(String[] args) throws Exception {

 }
}
/** 地面 */
class Ground {

}
/** 柱子类型，x,y是柱子的中心点的位置 */
class Column {

}
/** 鸟类型，x,y是鸟类型中心的位置 */
class Bird {

}
```

步骤三：为 BirdGame 类添加属性

添加的代码如图-6 所示：

```
public class BirdGame extends JPanel {
    Bird bird;
    Column column1, column2;
    Ground ground;
    BufferedImage background;
    /** 启动软件的方法 */
    public static void main(String[] args) throws Exception {
    }
}
```

图 - 6

步骤四：为 Ground 类添加属性

Ground 类中，各个属性表示的含义如图-7 所示：

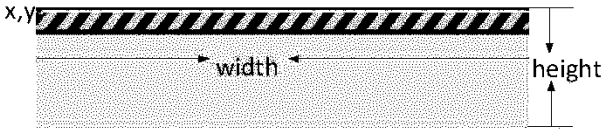

图 - 7

在 Ground 类中添加的代码如图-8 所示：

```
class Ground {
    BufferedImage image;
    int x, y;
    int width;
    int height;
}
```

图 - 8

另外，从图-8 中可以看出，还有一个 image 属性，该属性表示为 Ground 类的贴图。

步骤五：为 Column 类添加属性

Column 类中，各个属性表示的含义如图-9 所示：

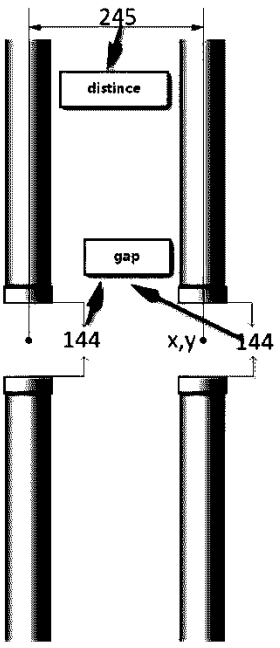

图 - 9

在 Column 类中添加的代码如图-10 所示：

```
/** 柱子类型，x,y是柱子的中心点的位置 */
class Column {
    BufferedImage image;
    int x, y;
    int width, height;
    /** 柱子中间的缝隙 */
    int gap;
    int distance;// 距离，两个柱子之间的距离
}
```

图 - 10

另外，从图-10 中可以看出，还有一个 image 属性，该属性表示为 Column 类的贴图。

步骤六：为 Bird 类添加属性

代码如图-11 所示：

```
/** 鸟类型，x,y是鸟类型中心的位置 */
class Bird {
    BufferedImage image;
    int x, y;
    int width, height;
    int size;// 鸟的大小，用于碰撞检测
}
```

图 - 11

Bird 的属性 image，表示为 Bird 类的贴图；x，y 表示鸟类型的中心位置；width，height 指的是鸟图片的宽和高；本案例中，小鸟的大小用一个正方形区域来表示，size 即表示该正方形的边长，用于后续的碰撞检测。

步骤七：为 BirdGrame 类添加构造方法，初始化属性

代码如图-12 所示：

```
public class BirdGame extends JPanel {
    Bird bird;
    Column column1, column2;
    Ground ground;
    BufferedImage background;

    /** 初始化 BirdGame 的属性变量 */
    public BirdGame() throws Exception {
        bird = new Bird();
        column1 = new Column(1);
        column2 = new Column(2);
        ground = new Ground();
        background = ImageIO.read(getClass().getResource("bg.png"));
    }

    /** 启动软件的方法 */
    public static void main(String[] args) throws Exception {
    }
}
```

图 - 12

从图-12 中，可以看出实例化 Column 类的时候，出现了编译错误。这是因为，还没有为 Column 类编写带一个参数的构造器。

步骤八：为 Column 类添加构造方法，初始化属性

代码如图-13 所示：

```
/** 柱子类型，x,y是柱子的中心点的位置 */
class Column {
    BufferedImage image;
    int x, y;
    int width, height;
    /** 柱子中间的缝隙 */
    int gap;
    int distance;// 距离，两个柱子之间的距离

    Random random = new Random();
    /** 构造器：初始化数据，n代表第几个柱子 */
    public Column(int n) throws Exception {
        image = ImageIO.read(getClass().getResource("column.png"));
        width = image.getWidth();
        height = image.getHeight();
        gap = 144;
        distance = 245;
        x = 550 + (n - 1) * distance;
        y = random.nextInt(218) + 132;
    }
}
```

图 - 13

下面介绍一下 x、y 坐标的计算方式，请看图-14。

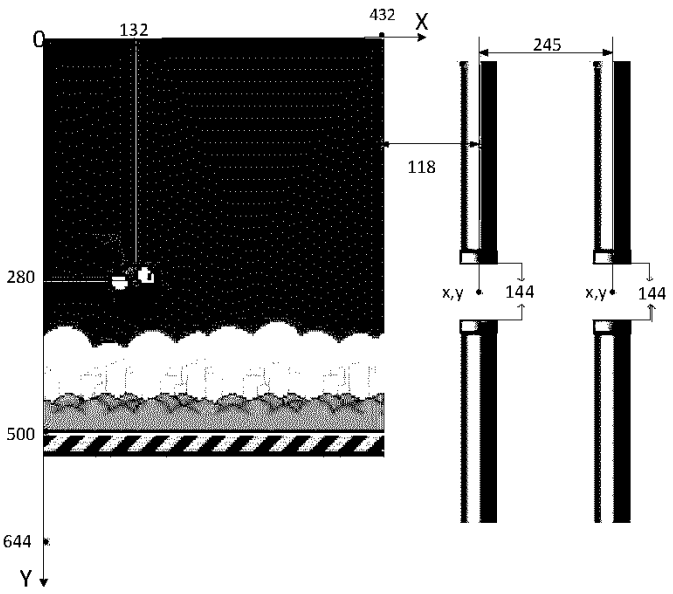

图 - 14

从图-14 可以看出，第一个柱子的 x 坐标 x1=432+118=550，第二根柱子的 x 坐标 x2=550+245，因此，柱子的 x 坐标可以总结为：x=550+(n-1)*245，其中 n 表示第几根柱子，例如：1，2。

y 坐标的计算，请看图-15。

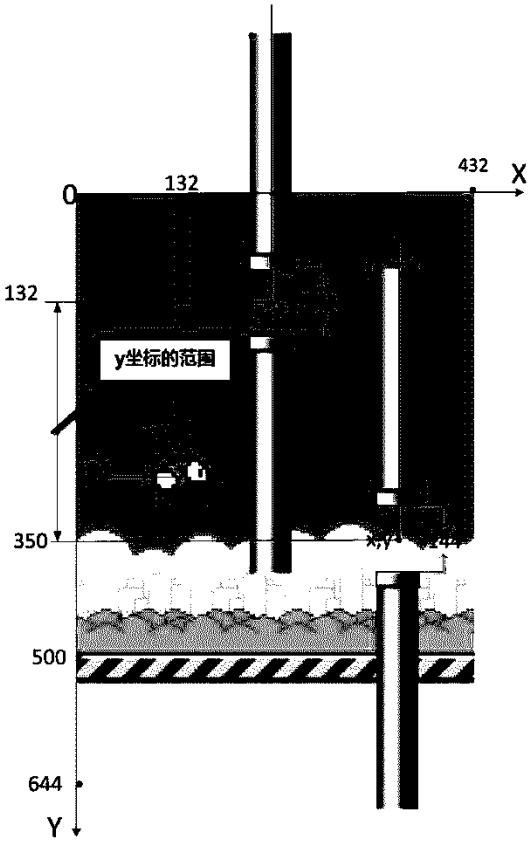

图 - 15

另外 从图-15 可以看出 柱子 y 坐标范围规定为 132 到 350 之间的随机数。使用 Random 类的 nextInt 方法生产[0，218）之间的随机数，再加上 132，因此计算出来的范围即为 132 到 350 之间。代码如下所示：

```
y = random.nextInt(218) + 132;
```

步骤九：为 Ground 类添加构造方法，初始化属性

代码如图-16 所示：

```
/** 地面 */
class Ground {
    BufferedImage image;
    int x, y;
    int width;
    int height;

    public Ground() throws Exception {
        image = ImageIO.read(getClass().getResource("ground.png"));
        width = image.getWidth();
        height = image.getHeight();
        x = 0;
        y = 500;
    }
}
```

图 - 16

地面初始的坐标位置如图-17 所示：

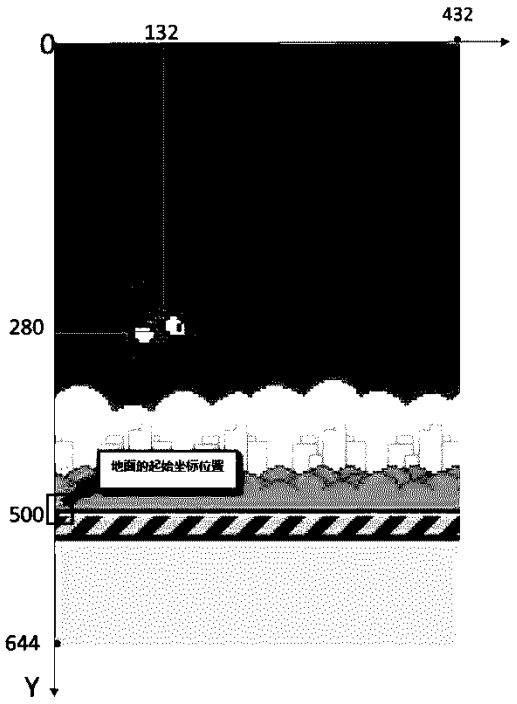

图 - 17

步骤十：为 Bird 类添加构造方法，初始化属性

代码如图-18 所示：

```
/** 鸟类型, x,y是鸟类型中心的位置 */
class Bird {
    BufferedImage image;
    int x, y;
    int width, height;
    int size;// 鸟的大小，用于碰撞检测

    public Bird() throws Exception {
        image = ImageIO.read(getClass().getResource("0.png"));
        width = image.getWidth();
        height = image.getHeight();
        x = 132;
        y = 280;
        size = 40;
    }
}
```

图 - 18

鸟的初始坐标位置如图-19 所示：

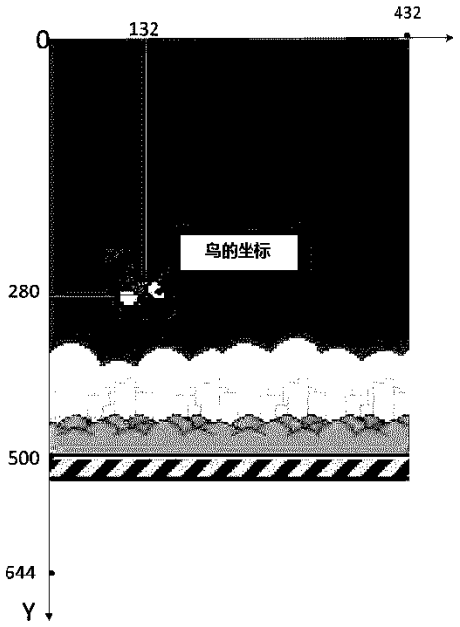

图 - 19

步骤十一：编写 main 方法

在 main 方法中，设置窗口的大小、居中、点击窗口的右上角“X”关闭窗口以及设置窗口可见，代码如图-20 所示：

```
public class BirdGame extends JPanel {
    Bird bird;
    Column column1, column2;
    Ground ground;
    BufferedImage background;

    /** 初始化 BirdGame 的属性变量 */
    public BirdGame() throws Exception {
        bird = new Bird();
        column1 = new Column(1);
        column2 = new Column(2);
        ground = new Ground();
        background = ImageIO.read(getClass().getResource("bg.png"));
    }

    /** 启动软件的方法 */
    public static void main(String[] args) throws Exception {
        JFrame frame = new JFrame();
        BirdGame game = new BirdGame();
        frame.add(game);
        frame.setSize(440, 670);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

图 - 20

步骤十二：绘制界面

在 BirdGame 类中重写 paint 方法，绘制界面。首先绘制地面，代码如图-21 所示：

```
/** "重写(修改)"paint方法实现绘制 */
public void paint(Graphics g) {
    g.drawImage(background, 0, 0, null);
}
```

图 - 21

使用 Graphics 的 drawImage 方法，绘制背景界面时，绘制的起始坐标点为（0，0）。然后，绘制两个柱子，代码如图-22 所示：

```
/** "重写(修改)"paint方法实现绘制 */
public void paint(Graphics g) {
    g.drawImage(background, 0, 0, null);
    g.drawImage(column1.image, column1.x - column1.width / 2, column1.y
            - column1.height / 2, null);
    g.drawImage(column2.image, column2.x - column2.width / 2, column2.y
            - column2.height / 2, null);
}
```

图 - 22

绘制两个柱子时，绘制的起始坐标与 Column 类的属性 x、y 的值不同，需要做一些更改，请看图-23。

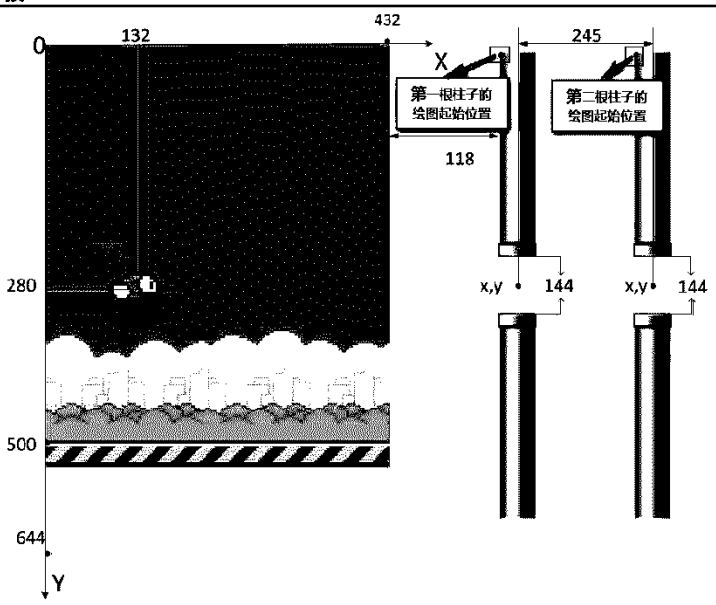

图 - 23

从图-23 可以看出，绘图起始位置 x=柱子.x-1/2 柱子.width；绘图起始位置 y=柱子.y-1/2 柱子.height。

接着，绘制地面，代码如图-24 所示：

```
/** "重写(修改)"paint方法实现绘制 */
public void paint(Graphics g) {
    g.drawImage(background, 0, 0, null);
    g.drawImage(column1.image, column1.x - column1.width / 2, column1.y
            - column1.height / 2, null);
    g.drawImage(column2.image, column2.x - column2.width / 2, column2.y
            - column2.height / 2, null);
    g.drawImage(ground.image, ground.x, ground.y, null);
}
```

图 - 24

最后，绘制小鸟，代码如图-25 所示：

```
/** "重写(修改)"paint方法实现绘制 */
public void paint(Graphics g) {
    g.drawImage(background, 0, 0, null);
    g.drawImage(column1.image, column1.x - column1.width / 2, column1.y
            - column1.height / 2, null);
    g.drawImage(column2.image, column2.x - column2.width / 2, column2.y
            - column2.height / 2, null);
    g.drawImage(ground.image, ground.x, ground.y, null);
    g.drawImage(bird.image, bird.x - bird.width / 2, bird.y - bird.height
            / 2, null);
}
```

图 - 25

从图-25 可以看出，绘制小鸟的时候，起始坐标位置也需要换算一下，原理请参考柱子的绘制原理。

步骤十三：运行

运行 BirdFrame 类，显示的界面效果如图-26 所示：

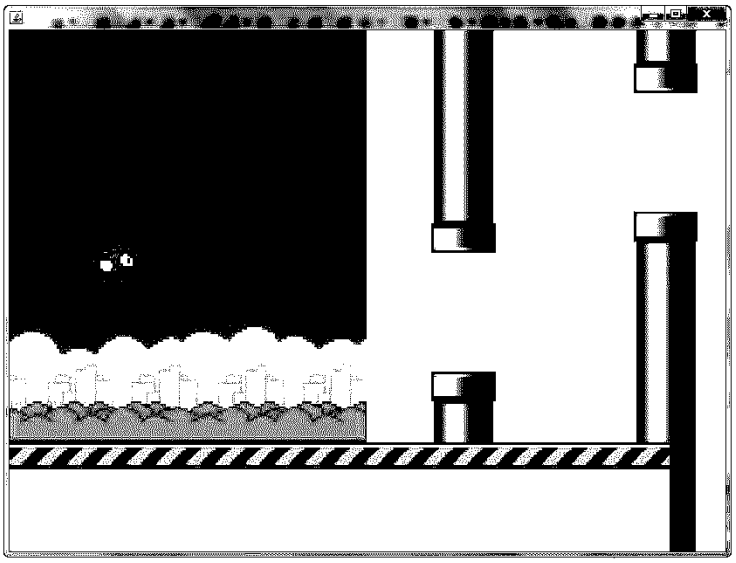

图 - 26

步骤十四：实现地面的运动

首先，在 Ground 类中添加 step 方法，代码如图-27 所示：

```
/** 地面 */
class Ground {
    BufferedImage image;
    int x, y;
    int width;
    int height;

    public Ground() throws Exception {
        image = ImageIO.read(getClass().getResource("ground.png"));
        width = image.getWidth();
        height = image.getHeight();
        x = 0;
        y = 500;
    }
    // 地面的类体中,添加方法，地面移动一步
    public void step() {
        x--;
        if (x == -109) {
            x = 0;
        }
    }
}
```

图 - 27

step 方法的作用为使地面移动一步，如果移动到 x 坐标为-109，则将 x 坐标设置为 0，形成滚动的效果。地面的坐标位置如图-28 所示：

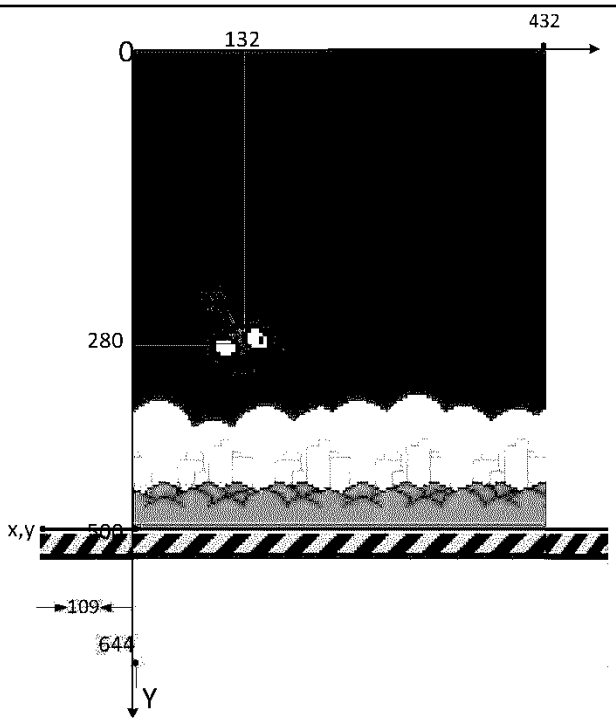

图 - 28

然后，在 BirdGame 类中添加 action()方法，在方法中利用死循环，在该循环中做以下操作来实现，地面的运动：

1）修改地面的位置，调用 Ground 类的 step 方法来实现。

2）重新绘制界面，调用 JPanel 的 repaint 方法来实现。

3）休眠 1/30 秒，即设置屏幕的刷新率为 1 秒 30 次，调用 Thread 类的 sleep 方法来实现。

在 BirdFrame 中添加的代码如图-29 所示：

```
// BirdGame中添加方法action()
public void action() throws Exception {
    while (true) {
        ground.step();
        repaint();
        Thread.sleep(1000 / 30);
    }
}

/** 启动软件的方法 */
public static void main(String[] args) throws Exception {
    JFrame frame = new JFrame();
    BirdGame game = new BirdGame();
    frame.add(game);
    frame.setSize(440, 670);
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
    game.action();
}
```

图 - 29

步骤十五：实现柱子的移动

首先，要实现柱子的移动，要计算出将柱子移动回出场位置的坐标，请看图-30。

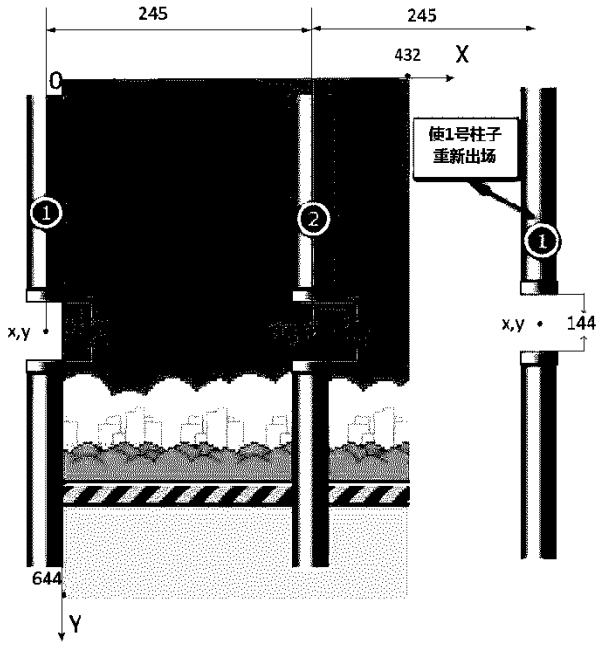

图 - 30

从图-30 可以看出，移回出场位置的 1 号柱子的 x 坐标为 distance * 2 - width / 2，y 坐标的范围不变，依然是 132 到 350 之间的随机数。在 Colunm 类中添加的代码如图-31 所示：

```
// 在Column中添加方法 step，在action调用此方法
public void step() {
    x--;
    if (x == -width / 2) {
        x = distance * 2 - width / 2;
        y = random.nextInt(218) + 132;
    }
}
```

图 - 31

另外，从图-30 可以看出，当柱子的 x 坐标为-width/2 时，柱子移动出界面。

然后，在 BirdGame 类的 action 方法，添加对 step 方法的调用，代码如图-32 所示：

```
// BirdGame中添加方法action()
public void action() throws Exception {
    while (true) {
        ground.step();
        column1.step();
        column2.step();
        repaint();
        Thread.sleep(1000 / 30);
    }
}
```

图 - 32

步骤十六：实现鸟的运动

首先，实现上抛运行。计算经过时间t的位移s的计算公式，请看图-33。

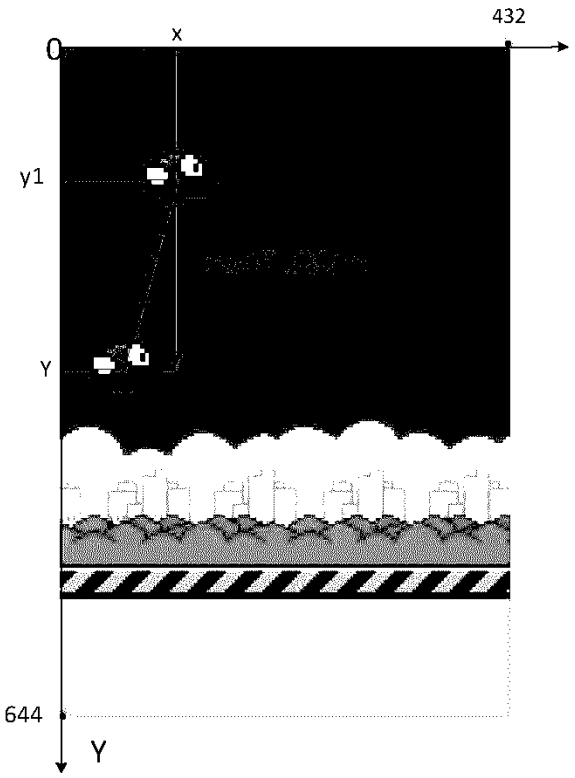

图 – 33

经过时间t的位置s的公式为：

```
s = v0 * t + g * t * t / 2;
```

从图-33 可以看出，经过时间 t 后小鸟的纵坐标位置 y1 的计算公式为：

```
y1 = y - s;
```

经过时间 t 后小鸟的运行速度 v 为：

```
v = v0 - g * t;
```

此时，在 Bird 类添加的代码如下所示：

```
/** 鸟类型，x,y是鸟类型中心的位置 */
class Bird {
 BufferedImage image;
 int x, y;
 int width, height;
 int size;// 鸟的大小，用于碰撞检测
```

```
// 在 Bird 类中增加属性，用于计算鸟的位置
double g;// 重力加速度
double t;// 两次位置的间隔时间
double v0;// 初始上抛速度
double speed;// 是当前的上抛速度
double s;// 是经过时间 t 以后的位移
double alpha;// 是鸟的倾角 弧度单位

public Bird() throws Exception {
    image = ImageIO.read(getClass().getResource("0.png"));
    width = image.getWidth();
    height = image.getHeight();
    x = 132;
    y = 280;
    size = 40;

    g = 4;
    v0 = 20;
    t = 0.25;
    speed = v0;
    s = 0;
    alpha = 0;

}
// 在 Bird 中添加鸟的移动方法

public void step() {
    double v0 = speed;
    s = v0 * t + g * t * t / 2;// 计算上抛运动位移
    y = y - (int) s;// 计算鸟的坐标位置
    double v = v0 - g * t;// 计算下次的速度
    speed = v;
}

}
```

然后，实现鸟的动画处理。使用数组，存储要显示的图片，共存储 8 张图片，鸟每运动一次换一张图片显示，以达到鸟的动画效果。此时，Bird 类中添加的代码如下所示：

```
/** 鸟类型, x,y 是鸟类型中心的位置 */
class Bird {
 BufferedImage image;
 int x, y;
 int width, height;
 int size;// 鸟的大小，用于碰撞检测

 // 在 Bird 类中增加属性，用于计算鸟的位置
 double g;// 重力加速度
 double t;// 两次位置的间隔时间
 double v0;// 初始上抛速度
 double speed;// 是当前的上抛速度
 double s;// 是经过时间 t 以后的位移
```

```
double alpha;// 是鸟的倾角 弧度单位

// 定义一组（数组）图片，是鸟的动画帧
BufferedImage[] images;
// 是动画帧数组元素的下标位置
int index;

public Bird() throws Exception {
    image = ImageIO.read(getClass().getResource("0.png"));
    width = image.getWidth();
    height = image.getHeight();
    x = 132;
    y = 280;
    size = 40;
    g = 4;
    v0 = 20;
    t = 0.25;
    speed = v0;
    s = 0;
    alpha = 0;
    // 创建数组,创建 8 个元素的数组
    // 是 8 个空位置，没有图片对象，
    // 8 个位置的序号: 0 1 2 3 4 5 6 7

    images = new BufferedImage[8];
    for (int i = 0; i < 8; i++) {
        // i = 0 1 2 3 4 5 6 7
        images[i] = ImageIO.read(getClass().getResource(i + ".png"));
    }
    index = 0;

}

// 在 Bird 中添加飞翔(fly)的代码
public void fly() {
    index++;
    image = images[ ( index/12 )  % 8];
}

// 在 Bird 中添加鸟的移动方法
public void step() {
    double v0 = speed;
    s = v0 * t + g * t * t / 2;// 计算上抛运动位移
    y = y - (int) s;// 计算鸟的坐标位置
    double v = v0 - g * t;// 计算下次的速度
    speed = v;
}
}
```

从上述代码中可以看出，每循环到第 8 张图片，则将 index 的值设置为 0，即从第 1 张图片再次开始显示。

另外，fly 方法实现了鸟的飞翔。一个数与 8 进行取余数，可以获取到 0 到 7 之间的余数，正好是 images 中图片的下标。index/12 的目的是使鸟的运动速度放慢，即切换照片

的频率放慢。

接着，在 BirdFrame 类的 action 方法中，调用 Bird 的 fly 方法和 step 方法。运行后，会发现鸟可以动起来了。代码如图-34 所示：

```
// BirdGame中添加方法action()
public void action() throws Exception {
    while (true) {
        ground.step();
        column1.step();
        column2.step();
        bird.step();
        bird.fly();
        repaint();
        Thread.sleep(1000 / 30);
    }
}
```

图 - 34

最后，计算倾角。请看图-35，了解什么是倾角。

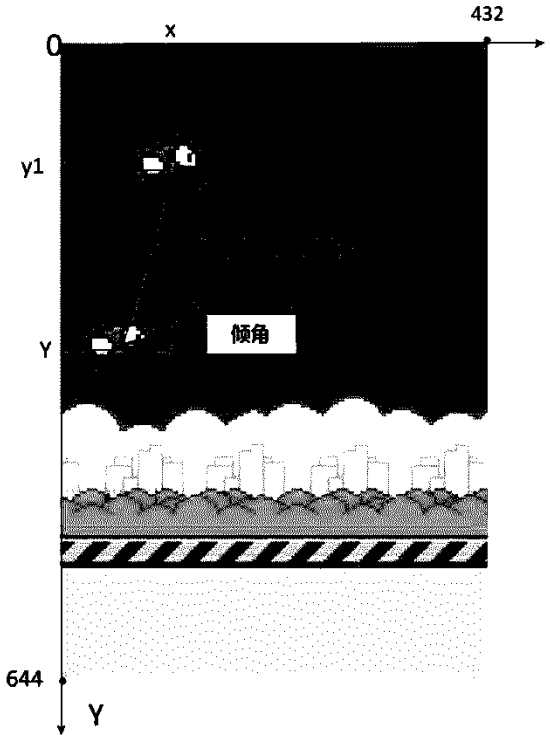

图 - 35

假设水平移动的位移为固定数 8，那么倾角=反正切 s/8。在 Bird 类添加的代码如下所示：

```
/** 鸟类型, x,y是鸟类型中心的位置 */
class Bird {
BufferedImage image;
int x, y;
int width, height;
```

```
    int size;// 鸟的大小，用于碰撞检测

    // 在 Bird 类中增加属性，用于计算鸟的位置
    double g;// 重力加速度
    double t;// 两次位置的间隔时间
    double v0;// 初始上抛速度
    double speed;// 是当前的上抛速度
    double s;// 是经过时间 t 以后的位移
    double alpha;// 是鸟的倾角 弧度单位
    // 在 Bird 类中定义
    // 定义一组（数组）图片，是鸟的动画帧
    BufferedImage[] images;
    // 是动画帧数组元素的下标位置
    int index;
    public Bird() throws Exception {
        image = ImageIO.read(getClass().getResource("0.png"));
        width = image.getWidth();
        height = image.getHeight();
        x = 132;
        y = 280;
        size = 40;
        g = 4;
        v0 = 20;
        t = 0.25;
        speed = v0;
        s = 0;
        alpha = 0;
        // 创建数组,创建 8 个元素的数组
        // 是 8 个空位置，没有图片对象，
        // 8 个位置的序号: 0 1 2 3 4 5 6 7
        images = new BufferedImage[8];
        for (int i = 0; i < 8; i++) {
            // i = 0 1 2 3 4 5 6 7
            images[i] = ImageIO.read(getClass().getResource(i + ".png"));
        }
        index = 0;
    }
    // 在 Bird 中添加鸟的移动方法
    public void step() {
        double v0 = speed;
        s = v0 * t + g * t * t / 2;// 计算上抛运动位移
        y = y - (int) s;// 计算鸟的坐标位置
        double v = v0 - g * t;// 计算下次的速度
        speed = v;

        // 调用 Java API 提供的反正切函数，计算倾角
        alpha = Math.atan(s / 8);

    }
}
```

上述代码中，调用 Java API 提供的反正切方法 *atan*，计算了倾角。

接着，在 BirdFrame 类的 paint 中，为达到鸟的飞翔效果，使用 Java API，使小鸟旋转，代码如图-36 所示：

```
/** "重写(修改)"paint方法实现绘制 */
public void paint(Graphics g) {
    g.drawImage(background, 0, 0, null);
    g.drawImage(column1.image, column1.x - column1.width / 2, column1.y
            - column1.height / 2, null);
    g.drawImage(column2.image, column2.x - column2.width / 2, column2.y
            - column2.height / 2, null);
    g.drawImage(ground.image, ground.x, ground.y, null);
    // 旋转(rotate)绘图坐标系，是API方法
    Graphics2D g2 = (Graphics2D) g;
    g2.rotate(-bird.alpha, bird.x, bird.y);
    g.drawImage(bird.image, bird.x - bird.width / 2, bird.y - bird.height
            / 2, null);
    g2.rotate(bird.alpha, bird.x, bird.y);
}
```

图 – 36

步骤十七：实现鼠标事件

要实现鼠标的按下事件，步骤如下：

1．使用“匿名内部类”声明事件"处理方法"。即在 BirdFrame 类的 action 方法声明事件处理方法；

2．首先，在 Bird 中添加 flappy 方法，以实现鸟的飞扬。然后，在处理方法中，调用 Bird 类的 flappy 方法；

3．注册事件监听。即在 BridFrame 类的 action 方法中，注册事件监听。

在 BirdFrame 类的添加的代码如图-37 所示：

```
public void action() throws Exception {
    MouseListener l = new MouseAdapter() {
        // 鼠标按下
        public void mousePressed(MouseEvent e) {
            // 鸟向上飞扬
            bird.flappy();
        }
    };
    // 将l挂接到当前的面板（game）上
    addMouseListener(l);

    while (true) {
        column1.step();
        column2.step();
        bird.step();// 上下移动
        bird.fly();// 挥动翅膀
        ground.step();// 地面移动
        repaint();
        Thread.sleep(1000 / 30);
    }
}
```

图 – 37

在 Bird 类添加的代码如图-38 所示：

```
public void flappy() {
    // 重新设置初始速度，重新向上飞
    speed = v0;
}
```

图 - 38

从图-38 可以看出，实现 flappy 方法，只需将速度设置为初始速度 v0 即可。

步骤十八：实现记分

要实现记分，需要进行以下操作：

1. 在 BirdFrame 类中增加属性 score，用于记分，score 初始化为 0。
2. 在 BirdFrame 类的 action 方法的主循环中，添加判分逻辑。当柱子的 x 坐标与鸟的 x 坐标重合时，则加一分。
3. 在 BirdFrame 类的 paint 方法中将分数画出来。

在 BirdFrame 类中增加的代码如下所示：

```
public class BirdGame extends JPanel {
Bird bird;
Column column1, column2;
Ground ground;
BufferedImage background;

// 分数
int score;

/** 初始化 BirdGame 的属性变量 */
public BirdGame() throws Exception {
    bird = new Bird();
    column1 = new Column(1);
    column2 = new Column(2);
    ground = new Ground();
    background = ImageIO.read(getClass().getResource("bg.png"));
}

/** "重写(修改)"paint 方法实现绘制 */
public void paint(Graphics g) {
    g.drawImage(background, 0, 0, null);
    g.drawImage(column1.image, column1.x - column1.width / 2, column1.y
            - column1.height / 2, null);
    g.drawImage(column2.image, column2.x - column2.width / 2, column2.y
            - column2.height / 2, null);
    g.drawImage(ground.image, ground.x, ground.y, null);
    // 旋转(rotate)绘图坐标系，是 API 方法
    Graphics2D g2 = (Graphics2D) g;
    g2.rotate(-bird.alpha, bird.x, bird.y);
    g.drawImage(bird.image, bird.x - bird.width / 2, bird.y - bird.height
            / 2, null);
    g2.rotate(bird.alpha, bird.x, bird.y);

    // 在 paint 方法中添加绘制分数的算法
    Font f = new Font(Font.SANS_SERIF, Font.BOLD, 40);
    g.setFont(f);
    g.drawString("" + score, 40, 60);
    g.setColor(Color.WHITE);
```

```
        g.drawString("" + score, 40 - 3, 60 - 3);

    }

    // BirdGame 中添加方法 action()
    public void action() throws Exception {
        MouseListener l = new MouseAdapter() {
            //鼠标按下
            public void mousePressed(MouseEvent e) {
               bird.flappy();
            }
           };
        // 将 l 挂接到当前的面板（game）上
        addMouseListener(l);

        while (true) {
            ground.step();
            column1.step();
            column2.step();
            bird.step();
            bird.fly();


            // 计分逻辑
            if (bird.x == column1.x || bird.x == column2.x) {
                score++;
            }


            repaint();
            Thread.sleep(1000 / 30);
        }
    }

    /** 启动软件的方法 */
    public static void main(String[] args) throws Exception {
        JFrame frame = new JFrame();
        BirdGame game = new BirdGame();
        frame.add(game);
        frame.setSize(440, 670);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        game.action();
    }
}
```

步骤十九：实现鸟的碰撞检测

1．实现鸟碰撞地面。

1）为 BirdGame 增加 boolean 变量 gameOver，该变量用于标识游戏是否结束，如果 gameOver 的值为 true，则表示游戏结束了；如果 gameOver 的值为 false，则表示游戏还没有结束。gameOver 的默认值为 false。

2)当检测到碰到地面时 ,即 gameOver 的值为 true。在主循环中检测是否碰到地面。在 Bird 类中增加方法 hit(Ground)检测鸟是否碰地面。当鸟的 y 坐标减去 size/2 大于等于地面的 y 坐标时，则与地面发生碰撞。

3） 修改 BirdFrame 类的 action 方法中的主循环，当游戏没有结束时，执行鸟和柱子的移动。也就是说，如果游戏结束了，鸟和柱子就不移动了。

4）在 BirdFrame 类的 paint 方法添加当 gameOver 的值为 true 时，显示游戏结束界面。

在 BirdFrame 类添加的代码如下所示：

```
public class BirdGame extends JPanel {
 Bird bird;
 Column column1, column2;
 Ground ground;
 BufferedImage background;

 boolean gameOver;
 BufferedImage gameOverImage;

 // 分数
 int score;

 /** 初始化 BirdGame 的属性变量 */
 public BirdGame() throws Exception {

     gameOver = false;
     gameOverImage = ImageIO.read(getClass().getResource("gameover.png"));

     bird = new Bird();
     column1 = new Column(1);
     column2 = new Column(2);
     ground = new Ground();
     background = ImageIO.read(getClass().getResource("bg.png"));
 }

 /** "重写(修改)"paint 方法实现绘制 */
 public void paint(Graphics g) {
     g.drawImage(background, 0, 0, null);
     g.drawImage(column1.image, column1.x - column1.width / 2, column1.y
             - column1.height / 2, null);
     g.drawImage(column2.image, column2.x - column2.width / 2, column2.y
             - column2.height / 2, null);
     g.drawImage(ground.image, ground.x, ground.y, null);
     // 旋转(rotate)绘图坐标系，是 API 方法
     Graphics2D g2 = (Graphics2D) g;
     g2.rotate(-bird.alpha, bird.x, bird.y);
     g.drawImage(bird.image, bird.x - bird.width / 2, bird.y - bird.height
             / 2, null);
     g2.rotate(bird.alpha, bird.x, bird.y);

     // 在 paint 方法中添加绘制分数的算法
     Font f = new Font(Font.SANS_SERIF, Font.BOLD, 40);
     g.setFont(f);
     g.drawString("" + score, 40, 60);
     g.setColor(Color.WHITE);
     g.drawString("" + score, 40 - 3, 60 - 3);

     if (gameOver) {
         g.drawImage(gameOverImage, 0, 0, null);
     }

 }

 // BirdGame 中添加方法 action()
```

```
public void action() throws Exception {
    MouseListener l = new MouseAdapter() {
        //鼠标按下
        public void mousePressed(MouseEvent e) {
            bird.flappy();
        }
    };
    // 将 l 挂接到当前的面板（game）上
    addMouseListener(l);
    while (true) {

        if (!gameOver) {
            ground.step();
            column1.step();
            column2.step();
            bird.step();
            bird.fly();
        }

        if (bird.hit(ground)) {
            gameOver = true;
        }


        // 计分逻辑
        if (bird.x == column1.x || bird.x == column2.x) {
            score++;
        }
        repaint();
        Thread.sleep(1000 / 30);
    }
}

/** 启动软件的方法 */
public static void main(String[] args) throws Exception {
    JFrame frame = new JFrame();
    BirdGame game = new BirdGame();
    frame.add(game);
    frame.setSize(440, 670);
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT ON CLOSE);
    frame.setVisible(true);
    game.action();
}
}
```

在 Bird 类添加的碰撞方法的代码如图-39 所示：

```
public boolean hit(Ground ground) {
    boolean hit = y + size / 2 > ground.y;
    if (hit) {
        //将鸟放置的地面上
        y = ground.y - size / 2;
        //使鸟落地面时，有摔倒的效果
        alpha = -3.14159265358979323 / 2;
    }
    return hit;
}
```

图 - 39

2．实现鸟碰撞柱子。请看如-40。

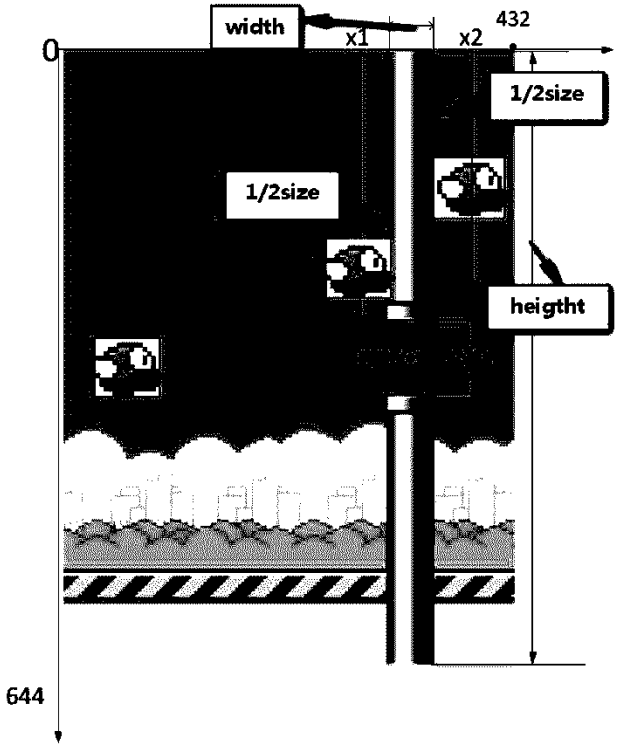

图 - 40

从图-40 可以看出，小鸟与柱子发生碰撞的 x 坐标计算如下：

```
x1=column.x-width/2-size/2
x2=column.x+width/2+size/2
```

当鸟的 x 坐标大于 x1（x>x1）并且小于 x2（x<x2）时与柱子发生碰撞。但是如果在缝隙中，则不发生碰撞，请看如图-41。

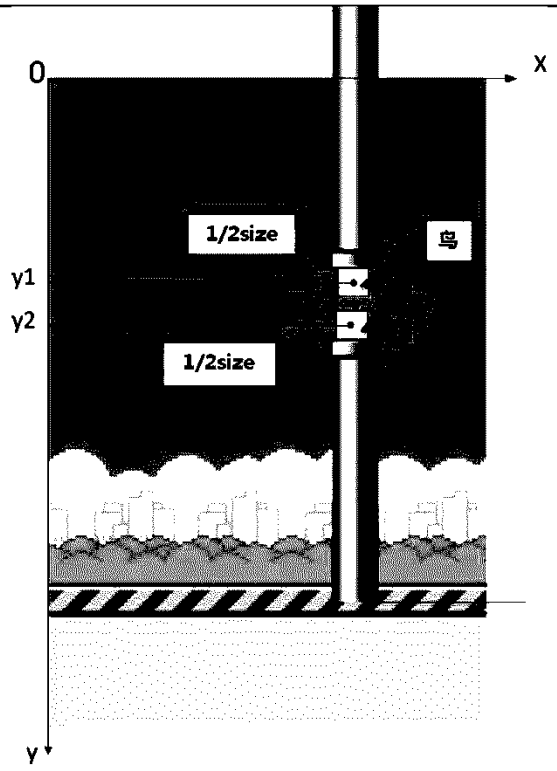

图 - 41

从图-41 可以总结出，小鸟在柱子的缝隙中时 y 坐标计算如下：

```
y1=column.y-gap/2+size/2
y2=column.y+gap/2-size/2
```

当鸟的 y 坐标大于 y1（y>y1）并且小于 y2（y<y2）时，鸟在缝隙中。

在 Bird 类添加 hit（Column）方法，用于判断鸟是否碰撞柱子，代码如图-42 所示：

```
// 检测当前的鸟是否撞到柱子
public boolean hit(Column column) {
    // 先检测是否在柱子的范围以内
    if (x > column.x - column.width / 2 - size / 2
            && x < column.x + column.width / 2 + size / 2) {
        // 检测是否在缝隙中
        if (y > column.y - column.gap / 2 + size / 2
                && y < column.y + column.gap / 2 - size / 2) {
            return false;
        }
        return true;
    }
    return false;
}
```

图 - 42

修改 BirdFrame 类的 action 方法，修改的代码如图-43 所示：

```
public void action() throws Exception {
    MouseListener l = new MouseAdapter() {

        public void mousePressed(MouseEvent e) {
            bird.flappy();
        }
    };
    // 将l挂接到当前的面板（game）上
    addMouseListener(l);
    while (true) {
        if (!gameOver) {
            ground.step();
            column1.step();
            column2.step();
            bird.step();
            bird.fly();
        }
        if ((bird.hit(ground) || bird.hit(column1) || bird.hit(column2))) {
            gameOver = true;
        }
        // 计分逻辑
        if (bird.x == column1.x || bird.x == column2.x) {
            score++;
        }
        repaint();
        Thread.sleep(1000 / 30);
    }
}
```

图 – 43

完成上述代码，即完成了判断鸟的碰撞功能。

步骤二十：实现游戏的开始、结束以及重新开始

首先，把游戏分成三种状态，分别是 start、running 以及 game_over，start 状态的界面效果如图-44 所示：

图 – 44

当游戏为 start 状态时，小鸟的翅膀是挥舞的、地面是移动的。即在该状态时，调用的 Bird 的 fly 方法以及 Ground 的 step 方法。

running 状态的界面效果如图-45 所示：

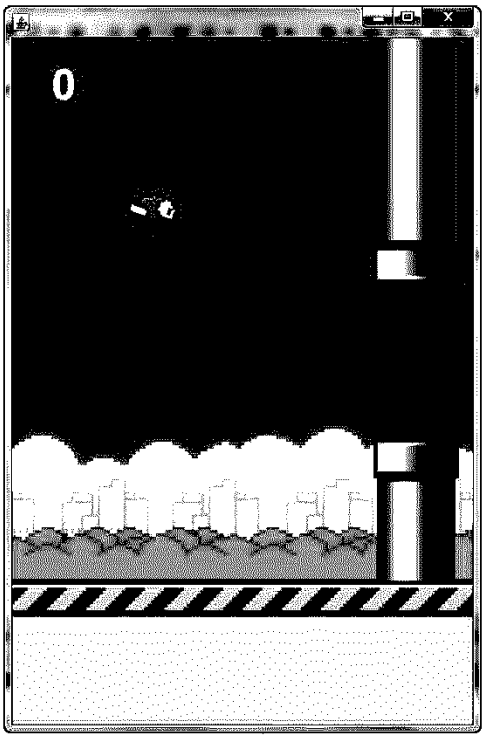

图 - 45

当游戏为 running 状态时，小鸟的翅膀是挥舞的、小鸟是上下移动的、地面是移动的、柱子是移动的。即在该状态时，调用的 Bird 的 fly 方法、Bird 的 step 方法、Ground 的 step 方法、Column 的 step 方法。并在这个状态中进行记分以及判断是否撞到柱子。

当游戏为 game_over 状态时的界面效果如图-46 所示：

图 - 46

当游戏为 game_over 状态时，只是显示的游戏结束的图片。

下面介绍一下当鼠标按下时，各个状态是如何进行转换的。

首先，当鼠标按下时如果当前游戏的状态为 game_over，则将下列信息进行重新初始化，并把状态更新为 start，即实现了重新开始功能。

```
column1 = new Column(1);
column2 = new Column(2);
bird = new Bird();
score = 0;
state = START;
```

然后，当鼠标按下时如果当前游戏的状态为 start，则将状态更新为 running。

```
state = RUNNING;
```

最后，当鼠标按下时如果当前游戏的状态为 running，则使小鸟飞扬，即调用 Bird 类的 flappy 方法。

完成游戏的开始、结束以及重新开始功能，在 BirdFrame 类中添加和修改的代码如下所示：

```
public class BirdGame extends JPanel {
 Bird bird;
 Column column1, column2;
 Ground ground;
 BufferedImage background;
```

```
//boolean gameOver;

/** 游戏状态 */
int state;
public static final int START = 0;
public static final int RUNNING = 1;
public static final int GAME_OVER = 2;

BufferedImage startImage;
BufferedImage gameOverImage;

// 分数
int score;

/** 初始化 BirdGame 的属性变量 */
public BirdGame() throws Exception {

    state = START;
    //gameOver = false;
    startImage = ImageIO.read(getClass().getResource("start.png"));

    gameOverImage = ImageIO.read(getClass().getResource("gameover.png"));
    bird = new Bird();
    column1 = new Column(1);
    column2 = new Column(2);
    ground = new Ground();
    background = ImageIO.read(getClass().getResource("bg.png"));
}

/** "重写(修改)"paint 方法实现绘制 */
public void paint(Graphics g) {
    g.drawImage(background, 0, 0, null);
    g.drawImage(column1.image, column1.x - column1.width / 2, column1.y
            - column1.height / 2, null);
    g.drawImage(column2.image, column2.x - column2.width / 2, column2.y
            - column2.height / 2, null);
    g.drawImage(ground.image, ground.x, ground.y, null);
    // 旋转(rotate)绘图坐标系，是 API 方法
    Graphics2D g2 = (Graphics2D) g;
    g2.rotate(-bird.alpha, bird.x, bird.y);
    g.drawImage(bird.image, bird.x - bird.width / 2, bird.y - bird.height
            / 2, null);
    g2.rotate(bird.alpha, bird.x, bird.y);

    // 在 paint 方法中添加绘制分数的算法
    Font f = new Font(Font.SANS_SERIF, Font.BOLD, 40);
    g.setFont(f);
    g.drawString("" + score, 40, 60);
    g.setColor(Color.WHITE);
    g.drawString("" + score, 40 - 3, 60 - 3);

    //if (gameOver) {
    //  g.drawImage(gameOverImage, 0, 0, null);
    //}

    // 在 paint 方法中添加显示游戏结束状态代码
    switch (state) {
    case GAME_OVER:
        g.drawImage(gameOverImage, 0, 0, null);
        break;
    case START:
```

```
            g.drawImage(startImage, 0, 0, null);
            break;
        }

    }

    // BirdGame 中添加方法 action()
    public void action() throws Exception {
        MouseListener l = new MouseAdapter() {
            public void mousePressed(MouseEvent e) {

                //bird.flappy();
                try {
                    switch (state) {
                    case GAME_OVER:
                        column1 = new Column(1);
                        column2 = new Column(2);
                        bird = new Bird();
                        score = 0;
                        state = START;
                        break;
                    case START:
                        state = RUNNING;
                    case RUNNING:
                        // 鸟向上飞扬
                        bird.flappy();
                    }
                } catch (Exception ex) {
                    ex.printStackTrace();
                }

            }
        };
        // 将 l 挂接到当前的面板 ( game ) 上
        addMouseListener(l);
        while (true) {

//          if (!gameOver) {
//              ground.step();
//              column1.step();
//              column2.step();
//              bird.step();
//              bird.fly();
//          }
//          if ((bird.hit(ground) || bird.hit(column1) || bird.hit(column2))) {
//              gameOver = true;
//          }
//          // 计分逻辑
//          if (bird.x == column1.x || bird.x == column2.x) {
//              score++;
//          }
            switch (state) {
            case START:
                bird.fly();
                ground.step();
                break;
            case RUNNING:
                column1.step();
                column2.step();
                bird.step();// 上下移动
                bird.fly();// 挥动翅膀
```

```
                ground.step();// 地面移动
                // 计分逻辑
                if (bird.x == column1.x || bird.x == column2.x) {
                    score++;
                }
                // 如果鸟撞上地面游戏就结束了
                if (bird.hit(ground) || bird.hit(column1) || bird.hit(column2)) {
                    state = GAME_OVER;
                }
                break;
            }

            repaint();
            Thread.sleep(1000 / 30);
        }
    }
```

4. 完整代码

本案例中，文件 BirdGame 的完整代码如下所示：

```
package com.tarena.bird;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.BufferedImage;
import java.util.Random;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class BirdGame extends JPanel {
 Bird bird;
 Column column1, column2;
 Ground ground;
 BufferedImage background;
 /** 游戏状态 */
 int state;
 public static final int START = 0;
 public static final int RUNNING = 1;
 public static final int GAME OVER = 2;
 BufferedImage gameOverImage;
 BufferedImage startImage;
 // 分数
 int score;

 /** 初始化 BirdGame 的属性变量 */
 public BirdGame() throws Exception {
     state = START;
     startImage = ImageIO.read(getClass().getResource("start.png"));
     gameOverImage = ImageIO.read(getClass().getResource("gameover.png"));
     score = 0;
     bird = new Bird();
     column1 = new Column(1);
     column2 = new Column(2);
```

```
        ground = new Ground();
        background = ImageIO.read(getClass().getResource("bg.png"));
    }

    /** "重写(修改)"paint 方法实现绘制 */
    public void paint(Graphics g) {
        g.drawImage(background, 0, 0, null);
        g.drawImage(column1.image, column1.x - column1.width / 2, column1.y
                - column1.height / 2, null);
        g.drawImage(column2.image, column2.x - column2.width / 2, column2.y
                - column2.height / 2, null);
        // 在 paint 方法中添加绘制分数的算法
        Font f = new Font(Font.SANS_SERIF, Font.BOLD, 40);
        g.setFont(f);
        g.drawString("" + score, 40, 60);
        g.setColor(Color.WHITE);
        g.drawString("" + score, 40 - 3, 60 - 3);

        g.drawImage(ground.image, ground.x, ground.y, null);
        // 旋转(rotate)绘图坐标系，是 API 方法
        Graphics2D g2 = (Graphics2D) g;
        g2.rotate(-bird.alpha, bird.x, bird.y);
        g.drawImage(bird.image, bird.x - bird.width / 2, bird.y - bird.height
                / 2, null);
        g2.rotate(bird.alpha, bird.x, bird.y);
        // 在 paint 方法中添加显示游戏结束状态代码
        switch (state) {
        case GAME_OVER:
            g.drawImage(gameOverImage, 0, 0, null);
            break;
        case START:
            g.drawImage(startImage, 0, 0, null);
            break;
        }

        // 添加调试的方框
        // g.drawRect(bird.x-bird.size/2,
        // bird.y-bird.size/2,
        // bird.size, bird.size);
        // g.drawRect(column1.x-column1.width/2,
        // column1.y-column1.height/2,
        // column1.width,
        // column1.height/2-column1.gap/2);
        // g.drawRect(column1.x-column1.width/2,
        // column1.y+column1.gap/2,
        // column1.width,
        // column1.height/2-column1.gap/2);
    }// paint 方法的结束
        // BirdGame 中添加方法 action()

    public void action() throws Exception {
        MouseListener l = new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                try {
                    switch (state) {
                    case GAME_OVER:
                        column1 = new Column(1);
                        column2 = new Column(2);
                        bird = new Bird();
                        score = 0;
                        state = START;
                        break;
                    case START:
                        state = RUNNING;
                    case RUNNING:
                        // 鸟向上飞扬
                        bird.flappy();
```

```
                    }
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            }
        };
        // 将 l 挂接到当前的面板（game）上
        addMouseListener(l);

        while (true) {
            switch (state) {
            case START:
                bird.fly();
                ground.step();
                break;
            case RUNNING:
                column1.step();
                column2.step();
                bird.step();// 上下移动
                bird.fly();// 挥动翅膀
                ground.step();// 地面移动
                // 计分逻辑
                if (bird.x == column1.x || bird.x == column2.x) {
                    score++;
                }
                // 如果鸟撞上地面游戏就结束了
                if (bird.hit(ground) || bird.hit(column1) || bird.hit(column2))
{
                    state = GAME_OVER;
                }
                break;
            }
            repaint();
            Thread.sleep(1000 / 60);
        }
    }

    /** 启动软件的方法 */
    public static void main(String[] args) throws Exception {
        JFrame frame = new JFrame();
        BirdGame game = new BirdGame();
        frame.add(game);
        frame.setSize(440, 670);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        game.action();
    }
    }

    /** 地面 */
    class Ground {
     BufferedImage image;
     int x, y;
     int width;
     int height;

     public Ground() throws Exception {
         image = ImageIO.read(getClass().getResource("ground.png"));
         width = image.getWidth();
         height = image.getHeight();
         x = 0;
         y = 500;
     }// 地面的构造器结束

    // 地面的类体中,添加方法，地面移动一步
```

```
    public void step() {
        x--;
        if (x == -109) {
            x = 0;
        }
    }
}// 地面类的结束

/** 柱子类型，x,y是柱子的中心点的位置 */
class Column {
    BufferedImage image;
    int x, y;
    int width, height;
    /** 柱子中间的缝隙 */
    int gap;
    int distance;// 距离，两个柱子之间的距离
    Random random = new Random();

    /** 构造器：初始化数据，n代表第几个柱子 */
    public Column(int n) throws Exception {
        image = ImageIO.read(getClass().getResource("column.png"));
        width = image.getWidth();
        height = image.getHeight();
        gap = 144;
        distance = 245;
        x = 550 + (n - 1) * distance;
        y = random.nextInt(218) + 132;
    }

    // 在Column中添加方法 step，在action调用此方法
    public void step() {
        x--;
        if (x == -width / 2) {
            x = distance * 2 - width / 2;
            y = random.nextInt(218) + 132;
        }
    }
}// Column类的结束

/** 鸟类型，x,y是鸟类型中心的位置 */
class Bird {
    BufferedImage image;
    int x, y;
    int width, height;
    int size;// 鸟的大小，用于碰撞检测

    // 在Bird类中增加属性，用于计算鸟的位置
    double g;// 重力加速度
    double t;// 两次位置的间隔时间
    double v0;// 初始上抛速度
    double speed;// 是当前的上抛速度
    double s;// 是经过时间t以后的位移
    double alpha;// 是鸟的倾角 弧度单位
    // 在Bird类中定义
    // 定义一组（数组）图片，是鸟的动画帧
    BufferedImage[] images;
    // 是动画帧数组元素的下标位置
    int index;

    public Bird() throws Exception {
        image = ImageIO.read(getClass().getResource("0.png"));
        width = image.getWidth();
        height = image.getHeight();
        x = 132;
        y = 280;
```

```
        size = 40;
        g = 4;
        v0 = 20;
        t = 0.25;
        speed = v0;
        s = 0;
        alpha = 0;
        // 创建数组,创建 8 个元素的数组
        // 是 8 个空位置，没有图片对象，
        // 8 个位置的序号：0 1 2 3 4 5 6 7
        images = new BufferedImage[8];
        for (int i = 0; i < 8; i++) {
            // i = 0 1 2 3 4 5 6 7
            images[i] = ImageIO.read(getClass().getResource(i + ".png"));
        }
        index = 0;
    }

    // 在 Bird 中添加飞翔(fly)的代码
    public void fly() {
        index++;
        image = images[(index / 12) % 8];
    }

    // 在 Bird 中添加鸟的移动方法
    public void step() {
        double v0 = speed;
        s = v0 * t + g * t * t / 2;// 计算上抛运动位移
        y = y - (int) s;// 计算鸟的坐标位置
        double v = v0 - g * t;// 计算下次的速度
        speed = v;
        // if(y>=500){//如果到达地面，就重新抛起
        // y = 280;
        // speed = 35;
        // }
        // 调用 Java API 提供的反正切函数，计算倾角
        alpha = Math.atan(s / 8);
    }

    // 在 Bird 中添加方法
    public void flappy() {
        // 重新设置初始速度，重新向上飞
        speed = v0;
    }

    // 在鸟中添加方法 hit
    // 检测当前鸟是否碰到地面 ground
    // 如果返回 true 表示发生碰撞
    // 否则返回 false 表示没有碰撞
    public boolean hit(Ground ground) {
        boolean hit = y + size / 2 > ground.y;
        if (hit) {
            y = ground.y - size / 2;
            alpha = -3.14159265358979323 / 2;
        }
        return hit;
    }

    // 检测当前的鸟是否撞到柱子
    public boolean hit(Column column) {
        // 先检测是否在柱子的范围以内
        if (x > column.x - column.width / 2 - size / 2
                && x < column.x + column.width / 2 + size / 2) {
            // 检测是否在缝隙中
            if (y > column.y - column.gap / 2 + size / 2
```

```
                    && y < column.y + column.gap / 2 - size / 2) {
                return false;
            }
            return true;
        }
        return false;
    }
}
```

俄罗斯方块

1. 问题

Tetris 俄罗斯方块（Tetris，俄文：Тетрис）是一款风靡全球的电视游戏机和掌上游戏机游戏，它由俄罗斯人阿列克谢·帕基特诺夫发明，故得此名。俄罗斯方块的基本规则是移动、旋转和摆放游戏自动输出的各种方块，使之排列成完整的一行或多行并且消除得分。由于上手简单、老少皆宜，从而家喻户晓，风靡世界。其界面效果如图 - 1 所示：

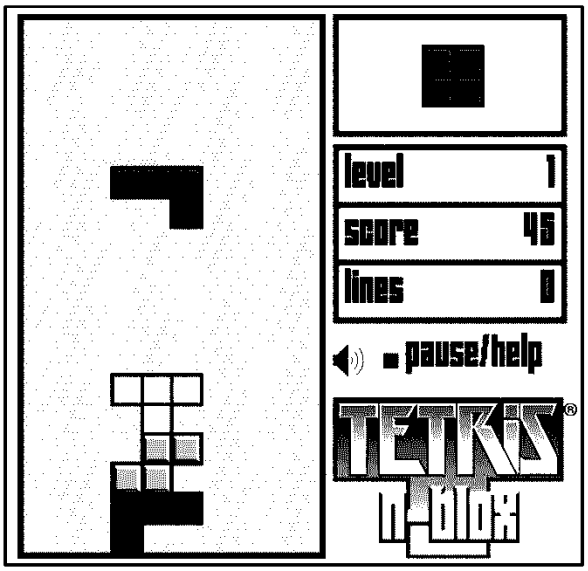

图 - 1

Tetris 游戏的基本规则为：

1 一个用于摆放正方形格子的平面虚拟场地（Wall），其标准大小：行宽为 10，列高为 20，以格子为单位计算，宽为 10 个格子，高为 20 个格子；

2 一组由 4 个小型正方形组成的规则图形（Tetromino），共有 7 种，分别以 S、Z、L、J、I、O、T 这 7 个字母的形状来命名，如图 - 2 所示：

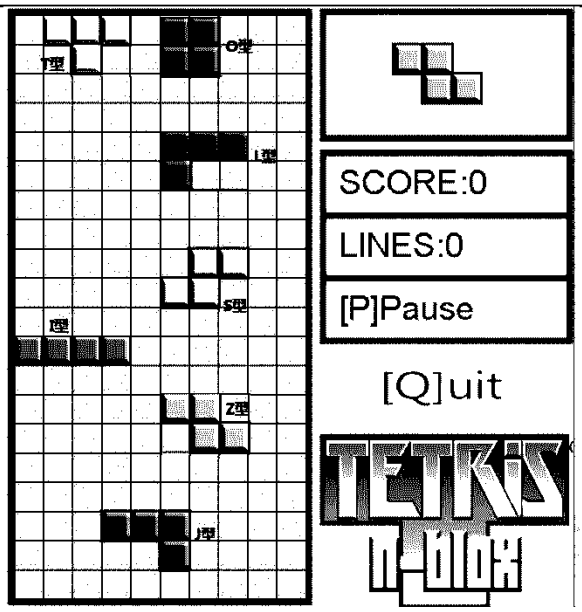

图 - 2

Tetris 游戏的逻辑为：

1. 玩家操作有：旋转方块；以格子为单位左右移动方块；让方块加速落下。

2. 方块移到区域最下方或是着地到其他方块上无法移动时，就会固定在该处，而新的方块出现在区域上方开始落下。

3. 当区域中某一行横向格子全部由方块填满，则该列会消失并成为玩家的得分。同时删除的行数越多，得分指数上升。

4. 当固定的方块堆到区域最上方而无法消除层数时，则游戏结束。

5. 一般来说，游戏还会提示下一个要落下的方块，熟练的玩家会考虑到下一个方块，评估要如何进行。由于游戏能不断进行下去，对商用游戏不太理想，所以一般还会随着游戏的进行而加速提高难度。

6. 预先设置的随机发生器不断地输出单个方块到场地顶部。

2. 方案

软件的开发过程如下：

1）需求(软件功能的文字描述)

2）需求分析(找对象)

3）概要设计

3.1）数学模型

3.2）类的设计

4）详细功能的设计

4.1）数据初始化

4.2）界面绘制

4.3）左右移动功能设计

4.4）下落功能设计

Tetris 项目开发过程如下：

1．业务需求分析

本案例中的问题部分就是我们的需求。

2．业务分析

根据图-2 进行分析，找出有哪些业务对象。可以看出窗口上有如下内容：

窗口

```
 |-- tetris(俄罗斯方块)
      |-- score 累计分数
      |-- lines 销毁的行数
      |-- wall(墙 20 行 X10 列)
      |      |-- 20 行 X10 列的 Cell
      |-- tetromino 正在下落的（4 格方块，有 7 种形态）
      |      |-- 4 个 Cell
      |-- nextOne 下一个准备下落的方块
            |-- 4 个 Cell
```

3．类设计

要想创建对象，就要有类。找出对象中的数据，根据对象定义类。本项目中，Cell 类（格子）、Tetromino 类（四格方块）、Tetris 类（俄罗斯方块）这几个类的数据情况如下：

```
Cell 类
 |-- int row
 |-- int col
 |-- image 贴图

Tetromino 类
 |--Cell[] cells
            |-- Cell * 4

Tetris 类
 |-- Cell[][] wall = 20*10
 |-- Tetromino tetromino 正在下落的方块
 |-- Tetromino nextOne 下一个方块
 |-- int lines
 |-- int score
```

本项目中类的属性和方法，以及类之间的如图-3 所示：

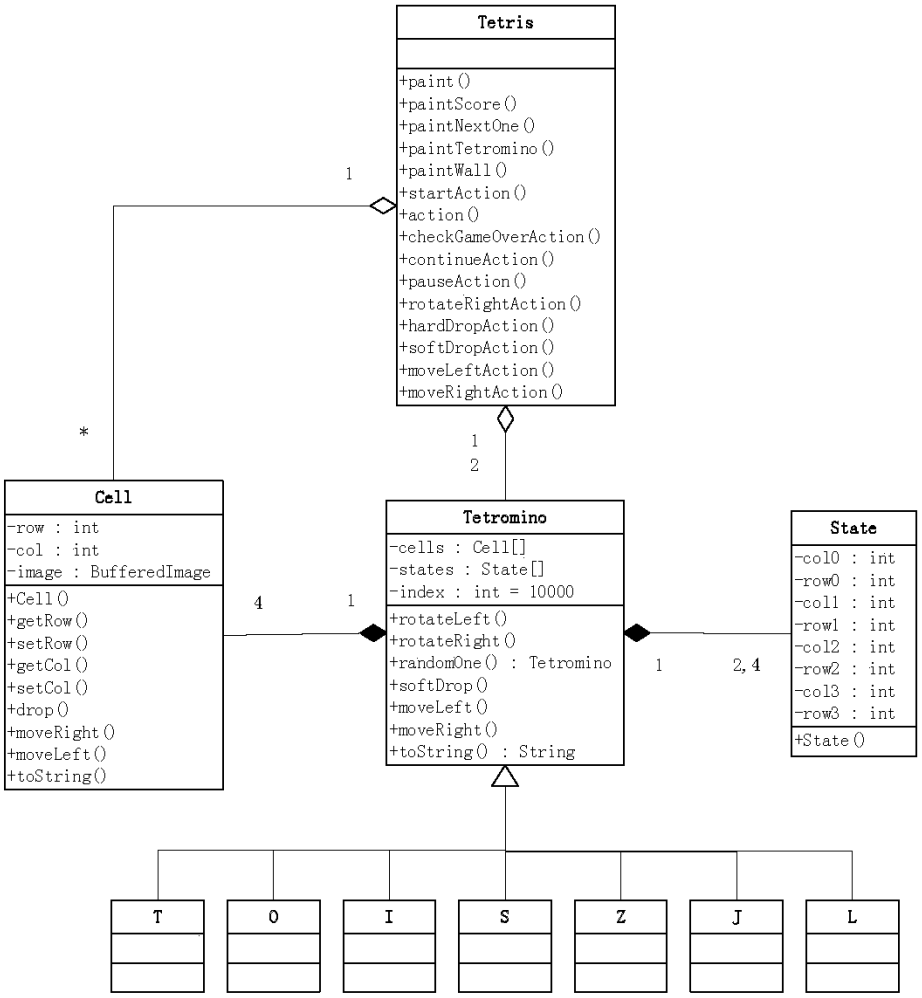

图-3

在后续的步骤中，详细介绍了这些类中的方法的实现，请参考步骤。

4.编码

编码就是我们所说的 coding 的过程，下面的步骤，就是本项目的全部编码过程。

3. 步骤

实现此案例需要按照如下步骤进行。

步骤一：新建工程和包

新建名为 TETRIS 的工程 ,然后 ,在工程下的 src 目录下新建包 com.tarena.tetris，工程结构如图-4 所示：

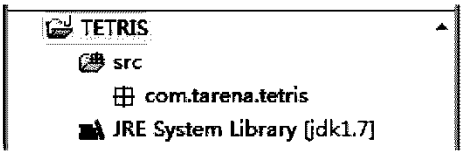

图- 4

在项目中，包的名字一般为公司域名倒过来，再加上项目名称，即为包名。如图-4 中中的包名为 com.tarena.tetris，其中，com.tarena 是达内公司的域名倒过来，tetris 为本项目的名称。

步骤二：新建 Cell 类

在 com.tarena.tetris 包下新建名为 Cell 的类，代码如下所示：

```
package com.tarena.tetris;

import java.awt.image.BufferedImage;

/**
 * 格子
 */
public class Cell {
 private int row;//格子的行
 private int col;//格子的列
 private BufferedImage image;//格子的贴图

 public Cell(int row, int col, BufferedImage image) {
     super();
     this.row = row;
     this.col = col;
     this.image = image;
 }

 public int getRow() {
     return row;
 }

 public void setRow(int row) {
     this.row = row;
 }

 public int getCol() {
     return col;
 }

 public void setCol(int col) {
     this.col = col;
 }

 public BufferedImage getImage() {
     return image;
 }

 public void setImage(BufferedImage image) {
     this.image = image;
 }

 public void drop(){
     row++;
 }

 public void moveRight(){
     col++;
 }

 public void moveLeft(){
     col--;
 }
```

```
    @Override//重写
    public String toString() {
        return row+","+col;
    }

}
```

步骤三：新建 Tetromino,表示四格方块

在 com.tarena.tetris 包下新建名为 Tetromino 的类，用于表示四个格子的方块，代码如下所示：

```
package com.tarena.tetris;

/**
 * 4 格方块
 */
public abstract class Tetromino {
 protected Cell[] cells = new Cell[4];
}
```

上述代码中的 cells 属性，来存放四个格子，即一个方块。

步骤四：新建 Tetris 类，表示俄罗斯方块

在 com.tarena.tetris 包下新建名为 Tetris 的类，用于构建俄罗斯方块项目的整体流程，代码如下所示：

```
package com.tarena.tetris;
/**
 * 俄罗斯方块
 */
public class Tetris {
 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块

 public static final int ROWS = 20;// 背景墙的行数
 public static final int COLS = 10;// 背景墙的列数

 public static void main(String[] args) {

 }
}
```

Tetris 类介绍：

```
Tetris
    |-- Cell[][] wall = 20 行*10 列  表示墙
    |-- Tetromino tetromino 表示正在下落的方块
    |-- Tetromino nextOne 表示下一个方块
```

|-- int lines 表示销毁的行数

|-- int score 表示游戏得分

步骤五：构建 Tetromino 类的结构

构建 Tetromino 类，在该类中添加方块下落方法 softDrop，左移方法 moveLeft，右移方法 moveRight 以及 toString 方法。另外，Tetromino 类的同一文件中，新建类 T、I、S、Z、O、L、J，他们都为 Tetromino 类的子类，用于表示各种形状的方块。代码如下所示：

```
package com.tarena.tetris;
/**
 * 4 格方块
 */
public abstract class Tetromino {
 protected Cell[] cells = new Cell[4];

 /** 工厂方法，随机产生 4 格方块 */
 public static Tetromino randomOne() {
     return null;
 }

 public void softDrop() {

 }

 public void moveLeft() {

 }

 public void moveRight() {

 }

 public String toString() {
     return null;
 }

}


class T extends Tetromino {
}

class I extends Tetromino {
}

class S extends Tetromino {
}

class Z extends Tetromino {
}

class O extends Tetromino {
}

class L extends Tetromino {
}
```

```
class J extends Tetromino {
}
```

步骤六：重构 Tetromino 类，实现上一步定义的方法

重构 Tetromino 类，实现方块下落方法 softDrop，左移方法 moveLeft，右移方法 moveRight、随机获取 7 种形状中的一种的 randomOne 方法以及 toString 方法，代码如下所示：

```
package com.tarena.tetris;

import java.util.Arrays;
import java.util.Random;

/**
 * 4 格方块
 */
public abstract class Tetromino {
 protected Cell[] cells = new Cell[4];

 /** 工厂方法，随机产生 4 格方块 */
 public static Tetromino randomOne() {

     Random random = new Random();
     int type = random.nextInt(7);
     switch (type) {
     case 0:
         return new T();
     case 1:
         return new I();
     case 2:
         return new S();
     case 3:
         return new J();
     case 4:
         return new L();
     case 5:
         return new Z();
     case 6:
         return new O();
     }

     return null;
 }
/**
 * 方块下落一个格子
 */
 public void softDrop() {

     for (int i = 0; i < cells.length; i++) {
         cells[i].drop();
     }

 }
/**
 * 方块左移一个格子
 */
 public void moveLeft() {
```

```
        for (int i = 0; i < cells.length; i++) {
            cells[i].moveLeft();
        }

    }
    /**
     * 方块右移一个格子
     */
    public void moveRight() {

        for (int i = 0; i < cells.length; i++) {
            cells[i].moveRight();
        }

    }
    /**
     * 覆盖 toString 方法，显示方块中每个格子的行列信息
     */
    public String toString() {

         return Arrays.toString(cells);

    }
}
/**
 * T 形方块
 */
class T extends Tetromino {
}
/**
 * S 形方块
 */
class I extends Tetromino {
}
/**
 * T 形方块
 */
class S extends Tetromino {
}
/**
 * Z 形方块
 */
class Z extends Tetromino {
}
/**
 * O 形方块
 */
class O extends Tetromino {
}
/**
 * L 形方块
 */
class L extends Tetromino {
}
/**
 * J 形方块
 */
class J extends Tetromino {
}
```

步骤七：重构 Tetromino 类，为各个子类添加构造方法

重构 Tetromino 类，为各个子类添加构造方法，代码如下所示：

```
package com.tarena.tetris;

import java.util.Arrays;
import java.util.Random;

/**
 * 4 格方块
 */
public abstract class Tetromino {
 protected Cell[] cells = new Cell[4];

 /** 工厂方法，随机产生 4 格方块 */
 public static Tetromino randomOne() {
     ... ...
 }

 /**
  * 方块下落一个格子
  */
 public void softDrop() {
     ... ...
 }

 /**
  * 方块左移一个格子
  */
 public void moveLeft() {
     ... ...
 }

 /**
  * 方块右移一个格子
  */
 public void moveRight() {
     ... ...
 }

 /**
  * 覆盖 toString 方法，显示方块中每个格子的行列信息
  */
 public String toString() {
     return Arrays.toString(cells);
 }
}

/**
 * T 形方块
 */
class T extends Tetromino {

 public T() {
     cells[0] = new Cell(0, 4, null);
     cells[1] = new Cell(0, 3, null);
     cells[2] = new Cell(0, 5, null);
     cells[3] = new Cell(1, 4, null);
 }
```

```
}

/**
 * I 形方块
 */
class I extends Tetromino {

public I() {
    cells[0] = new Cell(0, 4, null);
    cells[1] = new Cell(0, 3, null);
    cells[2] = new Cell(0, 5, null);
    cells[3] = new Cell(0, 6, null);
}

}

/**
 * S 形方块
 */
class S extends Tetromino {

public S() {
    cells[0] = new Cell(1, 4, null);
    cells[1] = new Cell(1, 3, null);
    cells[2] = new Cell(0, 4, null);
    cells[3] = new Cell(0, 5, null);
    states = new State[] { new State(0, 0, 0, -1, -1, 0, -1, 1),
            new State(0, 0, -1, 0, 0, 1, 1, 1) };
}

}

/**
 * Z 形方块
 */
class Z extends Tetromino {

public Z() {
    cells[0] = new Cell(1, 4, null);
    cells[1] = new Cell(0, 3, null);
    cells[2] = new Cell(0, 4, null);
    cells[3] = new Cell(1, 5, null);
}

}

/**
 * O 形方块
 */
class O extends Tetromino {

public O() {
    cells[0] = new Cell(0, 4, null);
    cells[1] = new Cell(0, 5, null);
    cells[2] = new Cell(1, 4, null);
    cells[3] = new Cell(1, 5, null);
}

}
```

```
/**
 * L形方块
 */
class L extends Tetromino {

public L() {
    cells[0] = new Cell(0, 4, null);
    cells[1] = new Cell(0, 3, null);
    cells[2] = new Cell(0, 5, null);
    cells[3] = new Cell(1, 3, null);
}

}

/**
 * J形方块
 */
class J extends Tetromino {

public J() {
    cells[0] = new Cell(0, 4, null);
    cells[1] = new Cell(0, 3, null);
    cells[2] = new Cell(0, 5, null);
    cells[3] = new Cell(1, 5, null);
}

}
```

以上代码对各种形状进行初始化，初始化时，cell 数组的第一个元素为后期旋转时的旋转轴位置。上述代码中，S、Z 两种形状，为了旋转时美观的考虑，cell[0]的坐标为（1，4），其余形状的 cell[0]坐标都为（0，4），在此请注意区别。

步骤八：重构 Tetris 类，构建界面

首先， 将 Tetris 类继承自 JPanel 类； 然后，在 Tetris 类的 main 方法中，使用 JFrame 创建框体，并将 Tetris 类所在的面板放置到窗体上，代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;

import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块

 public static final int ROWS = 20;// 背景墙的行数
```

```
public static final int COLS = 10;// 背景墙的列数

public static void main(String[] args) {

        JFrame frame = new JFrame();
        // 在加载 Tetris 类的时候，会执行静态代码块
        // 静态代码块,装载了图片素材，为图片对象
        Tetris tetris = new Tetris();
        //将面板的颜色设置为蓝色，用于测试
        tetris.setBackground(new Color(0x0000ff));
        frame.add(tetris);
        frame.setSize(530, 580);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);//"尽快"显示框口

}
}
```

上述代码中，设置了面板的背景色，以便运行后看到面板放到窗体上的效果。

运行上述代码，显示界面如图-5 所示：

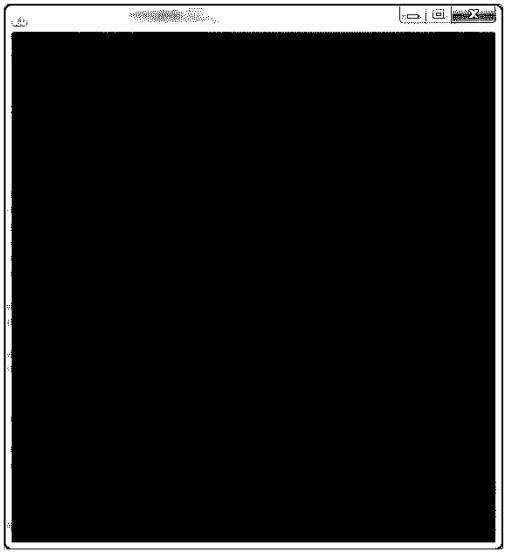

图- 5

步骤九：重构 Tetris 类和 Tetromino 类，加载图片

首先，在 Tetris 类中的 static 块中，使用 ImageIO 类的 read 方法加载图片，在此，请注意将图片文件放置在 com.tarena.tetris 包下。Tetris 类代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;
```

```
/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块
 /** 背景图片 */

 private static BufferedImage background;
 public static BufferedImage T;
 public static BufferedImage S;
 public static BufferedImage I;
 public static BufferedImage L;
 public static BufferedImage J;
 public static BufferedImage O;
 public static BufferedImage Z;

 public static final int ROWS = 20;// 背景墙的行数
 public static final int COLS = 10;// 背景墙的列数
 // 将图片素材，复制到 com.tarena.tetris 包中.
 /** 使用静态代码块加载静态的图片 */

 static {
     try {
         // Tetris.class 的同一个包中找 "tetris.png"
         background = ImageIO.read(Tetris.class.getResource("tetris.png"));
         T = ImageIO.read(Tetris.class.getResource("T.png"));
         I = ImageIO.read(Tetris.class.getResource("I.png"));
         S = ImageIO.read(Tetris.class.getResource("S.png"));
         Z = ImageIO.read(Tetris.class.getResource("Z.png"));
         J = ImageIO.read(Tetris.class.getResource("J.png"));
         L = ImageIO.read(Tetris.class.getResource("L.png"));
         O = ImageIO.read(Tetris.class.getResource("O.png"));
     } catch (Exception e) {
         e.printStackTrace();
     }
 }

 public static void main(String[] args) {
         ... ...
 }
}
```

然后，更改各种形状中方块的图片，Tetromino 类代码如下所示：

```
package com.tarena.tetris;
package com.tarena.tetris;

import java.util.Arrays;
import java.util.Random;

/**
 * 4 格方块
```

```
 */
public abstract class Tetromino {
 protected Cell[] cells = new Cell[4];

 /** 工厂方法，随机产生 4 格方块 */
 public static Tetromino randomOne() {
     ... ...
 }

 /**
  * 方块下落一个格子
  */
 public void softDrop() {
     ... ...
 }

 /**
  * 方块左移一个格子
  */
 public void moveLeft() {
     ... ...
 }

 /**
  * 方块右移一个格子
  */
 public void moveRight() {
     ... ...
 }

 /**
  * 覆盖 toString 方法，显示方块中每个格子的行列信息
  */
 public String toString() {
     return Arrays.toString(cells);
 }
}

/**
 * T 形方块
 */
class T extends Tetromino {
 public T() {

     cells[0] = new Cell(0, 4, Tetris.T);
     cells[1] = new Cell(0, 3, Tetris.T);
     cells[2] = new Cell(0, 5, Tetris.T);
     cells[3] = new Cell(1, 4, Tetris.T);

 }
}

/**
 * I 形方块
 */
class I extends Tetromino {
 public I() {

     cells[0] = new Cell(0, 4, Tetris.I);
     cells[1] = new Cell(0, 3, Tetris.I);
     cells[2] = new Cell(0, 5, Tetris.I);
     cells[3] = new Cell(0, 6, Tetris.I);
```

```
    }
}

/**
 * S 形方块
 */
class S extends Tetromino {
public S() {

    cells[0] = new Cell(1, 4, Tetris.S);
    cells[1] = new Cell(1, 3, Tetris.S);
    cells[2] = new Cell(0, 4, Tetris.S);
    cells[3] = new Cell(0, 5, Tetris.S);

}
}

/**
 * Z 形方块
 */
class Z extends Tetromino {
public Z() {

    cells[0] = new Cell(1, 4, Tetris.Z);
    cells[1] = new Cell(0, 3, Tetris.Z);
    cells[2] = new Cell(0, 4, Tetris.Z);
    cells[3] = new Cell(1, 5, Tetris.Z);

}
}

/**
 * O 形方块
 */
class O extends Tetromino {
public O() {

    cells[0] = new Cell(0, 4, Tetris.O);
    cells[1] = new Cell(0, 5, Tetris.O);
    cells[2] = new Cell(1, 4, Tetris.O);
    cells[3] = new Cell(1, 5, Tetris.O);

}
}

/**
 * L 形方块
 */
class L extends Tetromino {
public L() {

    cells[0] = new Cell(0, 4, Tetris.L);
    cells[1] = new Cell(0, 3, Tetris.L);
    cells[2] = new Cell(0, 5, Tetris.L);
    cells[3] = new Cell(1, 3, Tetris.L);

}
}

/**
```

```
 * J形方块
 */
class J extends Tetromino {
 public J() {

     cells[0] = new Cell(0, 4, Tetris.J);
     cells[1] = new Cell(0, 3, Tetris.J);
     cells[2] = new Cell(0, 5, Tetris.J);
     cells[3] = new Cell(1, 5, Tetris.J);

 }
}
```

步骤十：测试加载图片是否成功

首先，在Tetris类中，添加画墙的方法paintWall；

然后，在该类中，重写JPanel类的paint方法，在该方法中，加载背景图片、坐标系平移15个像素 、调用paintWall方法；

第三，在Tetris类中添加action方法，在该方法中，初始化数组wall，将墙的第二行第二列设置为T行方块中的一个格子（2，2），也就是说墙的第二行第二列有东西了，不再是空白的；

最后，重构Tetris类的main方法，添加对action方法的调用，代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块
 /** 背景图片 */
 private static BufferedImage background;
 public static BufferedImage T;
 public static BufferedImage S;
 public static BufferedImage I;
 public static BufferedImage L;
 public static BufferedImage J;
 public static BufferedImage O;
 public static BufferedImage Z;
 public static final int ROWS = 20;// 背景墙的行数
 public static final int COLS = 10;// 背景墙的列数
 // 将图片素材，复制到 com.tarena.tetris 包中.
```

```
/** 使用静态代码块加载静态的图片 */
static {
    ... ...
}
/**
 * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
 **/

@Override
public void paint(Graphics g) {
    // 画背景，画墙，画正在下落的方块 画下一个方块...
    g.drawImage(background, 0, 0, null);
    g.translate(15, 15);// 坐标系平移
    paintWall(g);// 画墙

}

/** 在 Tetris 添加启动方法 action() */

public void action() {
    wall = new Cell[ROWS][COLS];
     wall[2][2] = new Cell(2,2, T);
}
public static final int CELL_SIZE = 26;

/** 画墙 */

private void paintWall(Graphics g) {
    for (int row = 0; row < wall.length; row++) {
        Cell[] line = wall[row];
        // line 代表墙上的每一行
        for (int col = 0; col < line.length; col++) {
            Cell cell = line[col];
            // cell 代表墙上的每个格子
            int x = col * CELL_SIZE;
            int y = row * CELL_SIZE;
            if (cell == null) {
                 g.drawRect(x,y,CELL_SIZE,CELL_SIZE);

            } else {
                g.drawImage(cell.getImage(), x - 1, y - 1, null);

            }
            //g.drawString(row+","+col, x,y+CELL_SIZE);
        }
    }
}

public static void main(String[] args) {
        JFrame frame = new JFrame();
        // 在加载 Tetris 类的时候，会执行静态代码块
        // 静态代码块,装载了图片素材，为图片对象
        Tetris tetris = new Tetris();
        //将面板的颜色设置为蓝色，用于测试
        tetris.setBackground(new Color(0x0000ff));
        frame.add(tetris);
        frame.setSize(530, 580);
```

```
            frame.setLocationRelativeTo(null);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);// 在显示窗口时候,会"尽快"的调用 paint()方法绘制
界面

            tetris.action();

    }
}
```

上述代码中，在 paintWall 方法中，通过循环判断墙上的每一个区域是否有格子，如果 Cell 类的对象 cell 为 null，则说明没有格子，如果不为 null，则说明有格子。如果有格子，获取格子图片贴图到墙的相应位置。另外，在绘制图片时，出现 x-1,y-1 这样的代码，是为了不使两个格子黑色边框重叠，否则，美观效果不好。

运行上述代码，界面显示效果如图-6 所示：

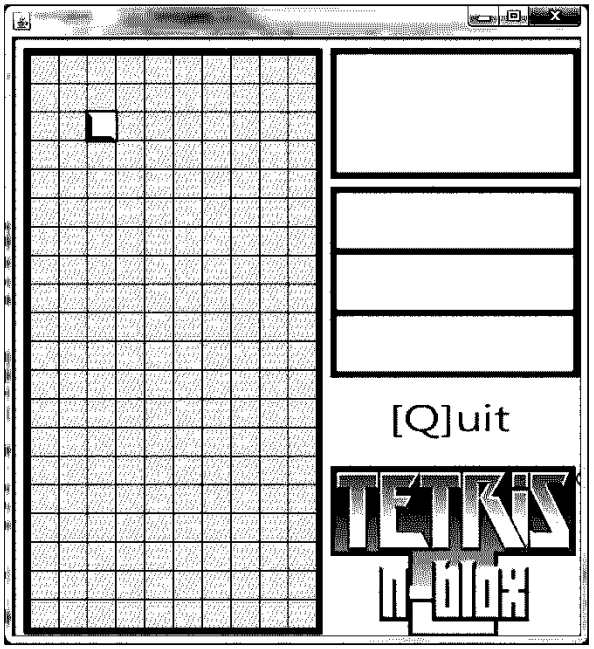

图- 6

在图-6 中，墙的第二行第二列显示了一个黄色格子。并且背景色图片也加载成功了。

步骤十一：绘制正在下落的方块

重构 Tetris 类，在类中添加方法 paintTetromino， 该方法实现绘制正在下落的方块。首先，将 row，col 坐标转换 drawImage 方法需要的 x，y 坐标，然后进行贴图，代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
```

```
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块
 /** 背景图片 */
 private static BufferedImage background;
 public static BufferedImage T;
 public static BufferedImage S;
 public static BufferedImage I;
 public static BufferedImage L;
 public static BufferedImage J;
 public static BufferedImage O;
 public static BufferedImage Z;
 public static final int ROWS = 20;// 背景墙的行数
 public static final int COLS = 10;// 背景墙的列数
 // 将图片素材，复制到 com.tarena.tetris 包中.
 /** 使用静态代码块加载静态的图片 */
 static {
     ... ...
 }
 /**
  * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
  **/
 @Override
 public void paint(Graphics g) {
     ... ...
 }
 /** 在 Tetris 添加启动方法 action() */
 public void action() {
     ... ...
 }

 /**
  * 绘制正在下落的方块
  * @param g
  */
 public void paintTetromino(Graphics g) {
     if (tetromino == null) {
         return;
     }
     // 将每个格子的 row,col 换算为 x,y 然后贴图
     Cell[] cells = tetromino.cells;
     for (int i = 0; i < cells.length; i++) {
         // i = 0 1 2 3
         Cell cell = cells[i];
         // cell 每个格子
         int x = cell.getCol() * CELL_SIZE;
         int y = cell.getRow() * CELL_SIZE;
         g.drawImage(cell.getImage(), x - 1, y - 1, null);
     }
 }
```

```
public static final int CELL_SIZE = 26;
/** 画墙 */
private void paintWall(Graphics g) {
    ... ...
}
public static void main(String[] args) {
        ... ...
}
}
```

步骤十二：将正在下落的方块显示在界面上

重构 Tetris 类中的 action 方法，初始化 tetromino 属性，代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
private int score;// 分数
private int lines;// 销毁的行数
private Cell[][] wall;// 背景墙
private Tetromino tetromino;// 正在下落的四格方块
private Tetromino nextOne;// 下一个四格方块
/** 背景图片 */
private static BufferedImage background;
public static BufferedImage T;
public static BufferedImage S;
public static BufferedImage I;
public static BufferedImage L;
public static BufferedImage J;
public static BufferedImage O;
public static BufferedImage Z;
public static final int ROWS = 20;// 背景墙的行数
public static final int COLS = 10;// 背景墙的列数
// 将图片素材，复制到 com.tarena.tetris 包中.
/** 使用静态代码块加载静态的图片 */
static {
    ... ...
}
/**
 * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
 **/
@Override
public void paint(Graphics g) {
    ... ...
}
/** 在 Tetris 添加启动方法 action() */
public void action() {
    wall = new Cell[ROWS][COLS];
```

```
        wall[2][2] = new Cell(2,2, T);

        tetromino= Tetromino.randomOne();

    }
    /**
     * 绘制正在下落的方块
     * @param g
     */
    public void paintTetromino(Graphics g) {
        ... ...
    }

    public static final int CELL_SIZE = 26;
    /** 画墙 */
    private void paintWall(Graphics g) {
        ... ...
    }
    public static void main(String[] args) {
            ... ...
    }
}
```

运行上述代码，界面效果如图-7 所示：

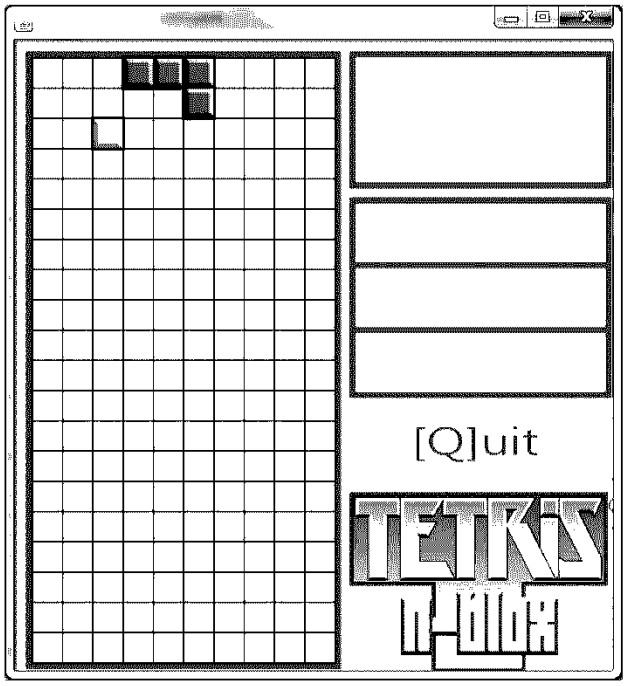

图- 7

图-7 中显示了 J 形方块，也可能是其它 6 种形状，因为，我们是通过随机的方式获取的方块。

步骤十三：绘制下一个要下落的方块

绘制下一个要下落的方块的过程，与绘制正在下落的方块的过程相同，代码如下所示：

```
package com.tarena.tetris;
```

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块
 /** 背景图片 */
 private static BufferedImage background;
 public static BufferedImage T;
 public static BufferedImage S;
 public static BufferedImage I;
 public static BufferedImage L;
 public static BufferedImage J;
 public static BufferedImage O;
 public static BufferedImage Z;
 public static final int ROWS = 20;// 背景墙的行数
 public static final int COLS = 10;// 背景墙的列数
 // 将图片素材，复制到 com.tarena.tetris 包中.
 /** 使用静态代码块加载静态的图片 */
 static {
     ... ...
 }
 /**
  * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
  **/
 @Override
 public void paint(Graphics g) {
     // 画背景，画墙，画正在下落的方块 画下一个方块...
     g.drawImage(background, 0, 0, null);
     g.translate(15, 15);// 坐标系平移
     paintWall(g);// 画墙
     paintTetromino(g);//绘制正在下落的方块

     paintNextOne(g);


 }
 /** 在 Tetris 添加启动方法 action() */
 public void action() {
     wall = new Cell[ROWS][COLS];
     wall[2][2] = new Cell(2,2, T);
     tetromino= Tetromino.randomOne();

     nextOne=Tetromino.randomOne();

 }
```

```
    /**
     * 绘制下一个要下落的方块
     * @param g
     */

    private void paintNextOne(Graphics g) {
        if (nextOne == null) {
            return;
        }
        // 将每个格子的 row,col 换算为 x,y 然后贴图
        Cell[] cells = nextOne.cells;
        for (int i = 0; i < cells.length; i++) {
            // i = 0 1 2 3
            Cell cell = cells[i];
            // cell 每个格子
            int x = (cell.getCol() + 10) * CELL_SIZE;
            int y = (cell.getRow() + 1) * CELL_SIZE;
            g.drawImage(cell.getImage(), x - 1, y - 1, null);
        }
    }


    /**
     * 绘制正在下落的方块
     * @param g
     */
    public void paintTetromino(Graphics g) {
        ... ...
    }

    public static final int CELL_SIZE = 26;
    /** 画墙 */
    private void paintWall(Graphics g) {
        ... ...
    }
    public static void main(String[] args) {
            ... ...
    }
}
```

运行上述代码，界面显示效果如图-8 所示：

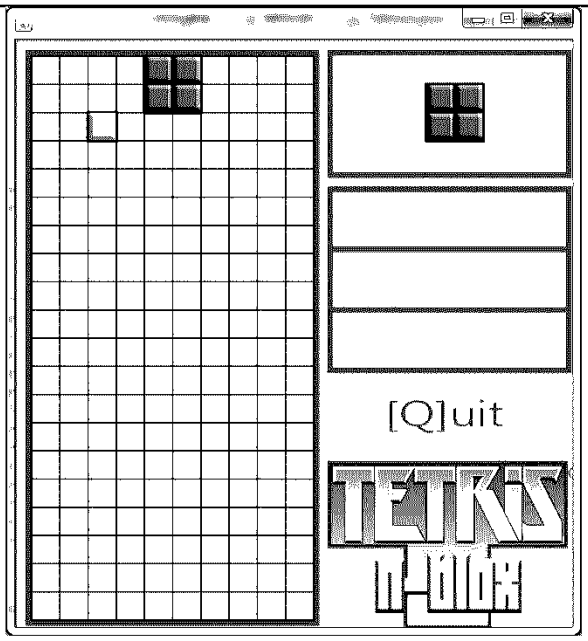

图- 8

在图-8 中，右侧区域显示了 0 行方块，即为下一次要出现在墙的方块。

步骤十四：处理键盘按下事件

重构 Tetris 类的 action 方法，添加处理键盘事件的方法。在按下键盘的向下键时执行下落方法、左方向键时执行左移方法、右方向键时执行右移方法，代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块
 /** 背景图片 */
 private static BufferedImage background;
 public static BufferedImage T;
 public static BufferedImage S;
 public static BufferedImage I;
 public static BufferedImage L;
 public static BufferedImage J;
 public static BufferedImage O;
```

```
public static BufferedImage Z;
public static final int ROWS = 20;// 背景墙的行数
public static final int COLS = 10;// 背景墙的列数
// 将图片素材，复制到 com.tarena.tetris 包中.
/** 使用静态代码块加载静态的图片 */
static {
    ... ...
}
/**
 * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
 **/
@Override
public void paint(Graphics g) {
    ... ...
}
/** 在 Tetris 添加启动方法 action() */
public void action() {
    wall = new Cell[ROWS][COLS];
    //wall[2][2] = new Cell(2,2, T);
    tetromino= Tetromino.randomOne();
    nextOne=Tetromino.randomOne();

    // 处理键盘按下事件，在按下按键时候执行下落方法


    KeyAdapter l = new KeyAdapter() {
        @Override
        // key 按键 Pressed 按下了
        public void keyPressed(KeyEvent e) {
            int key = e.getKeyCode();// [c]

            switch (key) {
            case KeyEvent.VK_DOWN:
                tetromino.softDrop();
                break;
            case KeyEvent.VK_RIGHT:
                tetromino.moveRight();
                break;
            case KeyEvent.VK_LEFT:
                tetromino.moveLeft();
                break;
            }
            repaint();// 再画一次!
        }
    };


    // 下落流程：监听键盘事件->如果下箭头按下->
    // 执行下落算法 tetromino.softDrop()->
    // 修改每个格子对象的数据->调用 repaint()->
    // 尽快调用 paint()->paint 方法会根据当前的数据
    // 重新绘制界面 -> 看到移动以后的方块了
    // 绑定事件到当前面板


    this.requestFocus();
    this.addKeyListener(l);


}
/**
```

```
     * 绘制下一个要下落的方块
     * @param g
     */
    private void paintNextOne(Graphics g) {
        ... ...
    }

    /**
     * 绘制正在下落的方块
     * @param g
     */
    public void paintTetromino(Graphics g) {
        ... ...
    }

    public static final int CELL_SIZE = 26;
    /** 画墙 */
    private void paintWall(Graphics g) {
        ... ...
    }
    public static void main(String[] args) {
            ... ...
    }
}
```

运行上述代码，显示界面后，按下键盘上的向下键、左方向键和右方向键，会发现方块实现了下落、左移和右移。

步骤十五：检查当前正在下落的方块左右移动时是否出界 、是否重合

在 Tetris 类中，新建 moveRightAction、moveLeftAction、outOfBounds 以及 coincide 方法，其中，outOfBounds 实现是否左右出界，coincide 实现是否重合。

outOfBounds 方法实现的逻辑为：如果方块中，有一个格子的列小于 0 或者列大于等于 10，则说明该方块左右出界了。

coincide 方法的实现逻辑为：循环获取正在下落的方块中，每一个格子的 row、col 坐标，如果对应墙上的 wall[row][col]该位置是否为 null，如果不为 null，则说明墙上的该位置有格子存在，也就是我们说的出现了重合现象。代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
    private int score;// 分数
    private int lines;// 销毁的行数
    private Cell[][] wall;// 背景墙
```

```
private Tetromino tetromino;// 正在下落的四格方块
private Tetromino nextOne;// 下一个四格方块
/** 背景图片 */
private static BufferedImage background;
public static BufferedImage T;
public static BufferedImage S;
public static BufferedImage I;
public static BufferedImage L;
public static BufferedImage J;
public static BufferedImage O;
public static BufferedImage Z;
public static final int ROWS = 20;// 背景墙的行数
public static final int COLS = 10;// 背景墙的列数
// 将图片素材，复制到 com.tarena.tetris 包中.
/** 使用静态代码块加载静态的图片 */
static {
    ... ...
}
/**
 * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
 **/
@Override
public void paint(Graphics g) {
    ... ...
}
/** 在 Tetris 添加启动方法 action() */
public void action() {
    wall = new Cell[ROWS][COLS];
    //wall[2][2] = new Cell(2,2, T);
    tetromino= Tetromino.randomOne();
    nextOne=Tetromino.randomOne();

    // 处理键盘按下事件，在按下按键时候执行下落方法
    KeyAdapter l = new KeyAdapter() {
        @Override
        // key 按键 Pressed 按下了
        public void keyPressed(KeyEvent e) {
            int key = e.getKeyCode();// [c]

            switch (key) {
            case KeyEvent.VK_DOWN:
                tetromino.softDrop();
                break;
            case KeyEvent.VK_RIGHT:

                moveRightAction();


                break;
            case KeyEvent.VK_LEFT:

                moveLeftAction();


                break;
            }
            repaint();// 再画一次!
        }
    };
    // 下落流程：监听键盘事件->如果下箭头按下->
```

```
        // 执行下落算法 tetromino.softDrop()->
        // 修改每个格子对象的数据->调用 repaint()->
        // 尽快调用 paint()->paint 方法会根据当前的数据
        // 重新绘制界面 -> 看到移动以后的方块了
        // 绑定事件到当前面板
        this.requestFocus();
        this.addKeyListener(l);
    }
    /**
     * 绘制下一个要下落的方块
     * @param g
     */
    private void paintNextOne(Graphics g) {
        ... ...
    }

    /**
     * 绘制正在下落的方块
     * @param g
     */
    public void paintTetromino(Graphics g) {
        ... ...
    }

    public static final int CELL_SIZE = 26;
    /** 画墙 */
    private void paintWall(Graphics g) {
        ... ...
    }
    /** 检查当前正在下落的方块是否出界了 */

    private boolean outOfBounds() {
        Cell[] cells = tetromino.cells;
        for (int i = 0; i < cells.length; i++) {
            Cell cell = cells[i];
            int col = cell.getCol();
            if (col < 0 || col >= COLS) {
                return true;
            }
        }
        return false;
    }

    /** 检查正在下落的方块是否与墙上的砖块重叠 */

    private boolean coincide() {
        Cell[] cells = tetromino.cells;
        for (int i = 0; i < cells.length; i++) {
            Cell cell = cells[i];
            int row = cell.getRow();
            int col = cell.getCol();
            // 如果墙的 row,col 位置上有格子,就重叠了!
            if (row >= 0 && row < ROWS && col >= 0 && col <= COLS
                    && wall[row][col] != null) {
                return true;// 重叠
            }
        }
        return false;
    }
```

```
/** 在 Tetris 类上添加方法，向右移动的流程控制 */

public void moveRightAction() {
    // 尝试先向右移动，如果发现超出了边界，就
    // 向左移动，修正回来.
    tetromino.moveRight();// coincide 重叠
    if (outOfBounds() || coincide()) {
        tetromino.moveLeft();
    }
}
public void moveLeftAction() {
    tetromino.moveLeft();
    if (outOfBounds() || coincide()) {
        tetromino.moveRight();
    }
}

public static void main(String[] args) {
        ... ...
}
}
```

运行上述代码，会发现左右移动不会出现越界的现象，并且移动方块不会和界面上的黄色方块重叠。

步骤十六： 实现下落流程

处理下落的流程为：

1）判断方块能否继续下落，如果能下落，则下落一步。

2）如果方块不能下落，则落到墙上。

3）落到墙面上以后，销毁充满的行，并且计算得分。

4）检查游戏是否结束。

解决方案：

1） 创建 canDrop 方法判断，判断方块能否继续下落。方块不能继续下落的条件有两个，一是当前下落的方块中的某一个格子的行坐标，与最后一行相等；二是当前下落的方块中的某一个格子的 row，col 坐标，与其对应的 wall[row+1][col]不为 null，则说明这个格子对应的下一行，在墙已经存在格子了。其余情况方块都是可以下落的。

2） 创建 landIntoWall 方法，实现方块着落在墙上。实现该方法首先构建循环，然后，在循环中，获取正在下落方块的每一个格子，格子的行列坐标用(row,col)表示，最后将格子放置到墙对应的位置 wall[row][col]。

3） 构建 destoryLines 方法，在该方法中，循环墙上的每一行，如果墙上的某一行对应的都有格子，说明该行是满的（构建 fullCells 方法实现进行判断）。如果这一行是满的，则销毁这一行。请看图-9。

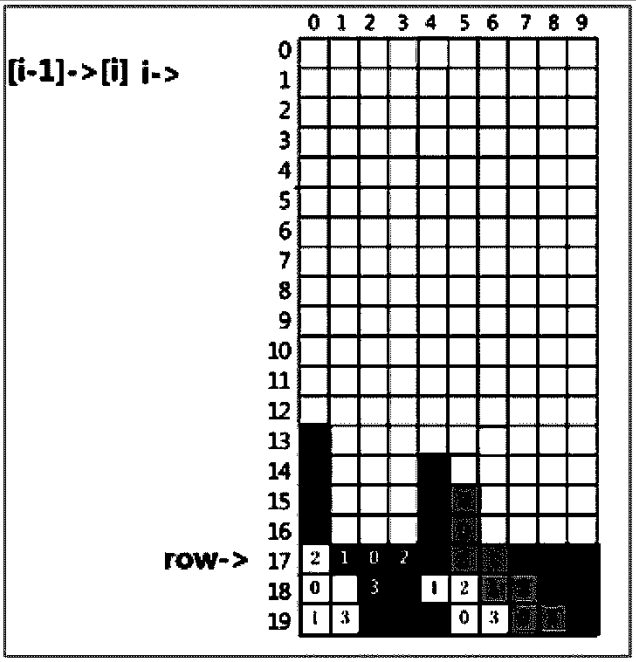

图 - 9

在此，构建 destoryRow 方法，实现销毁一行，在该方法中，从 row 循环到 1（row 为满的一行），在循环中，使用 System 类的 arraycopy 方法将（row-1）复制到 row 行，如图-8 所示：最后，使用 Arrays 的 fill 方法将墙的第 0 行的所有格子填充为 null。在 destoryLines 方法的最后，计算得分。

4）检查游戏是否结束的方法在后续介绍。代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.util.Arrays;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块
 /** 背景图片 */
 private static BufferedImage background;
 public static BufferedImage T;
 public static BufferedImage S;
```

```
public static BufferedImage I;
public static BufferedImage L;
public static BufferedImage J;
public static BufferedImage O;
public static BufferedImage Z;
public static final int ROWS = 20;// 背景墙的行数
public static final int COLS = 10;// 背景墙的列数
// 将图片素材，复制到 com.tarena.tetris 包中.
/** 使用静态代码块加载静态的图片 */
static {
    ... ...
}
/**
 * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
 **/
@Override
public void paint(Graphics g) {
    ... ...
}
/** 在 Tetris 添加启动方法 action() */
public void action() {
    ... ...
}
/**
 * 绘制下一个要下落的方块
 * @param g
 */
private void paintNextOne(Graphics g) {
    ... ...
}

/**
 * 绘制正在下落的方块
 * @param g
 */
public void paintTetromino(Graphics g) {
    ... ...
}

public static final int CELL_SIZE = 26;
/** 画墙 */
private void paintWall(Graphics g) {
    ... ...
}
/** 检查当前正在下落的方块是否出界了 */
private boolean outOfBounds() {
    ... ...
}
/** 检查正在下落的方块是否与墙上的砖块重叠 */
private boolean coincide() {
    ... ...
}
/** 在 Tetris 类上添加方法，向右移动的流程控制 */
public void moveRightAction() {
    ... ...
}
public void moveLeftAction() {
    ... ...
}
```

```
/** 下落流程控制 */
public void softDropAction() {
    if (canDrop()) {
        tetromino.softDrop();
    } else {
        landIntoWall();
        destoryLines();
        tetromino = nextOne;
        nextOne = Tetromino.randomOne();
    }
}

private static int[] scoreTable = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4

private void destoryLines() {
    int lines = 0;
    for (int row = 0; row < wall.length; row++) {
        if (fullCells(row)) {
            deleteRow(row);
            lines++;
        }
    }
    this.score += scoreTable[lines];
    this.lines += lines;
}

private void deleteRow(int row) {
    for (int i = row; i >= 1; i--) {
        System.arraycopy(wall[i - 1], 0, wall[i], 0, COLS);
    }
    Arrays.fill(wall[0], null);
}

/**
 * 检查当前行的每个格子,是否是满的，如果满了则返回 true，否则返回 false
 */

private boolean fullCells(int row) {
    Cell[] line = wall[row];
    for (Cell cell : line) {
        if (cell == null) {// 如果有 null 返回 false 否则返回 true
            return false;
        }
    }
    return true;
}

private void landIntoWall() {
    Cell[] cells = tetromino.cells;
    for (int i = 0; i < cells.length; i++) {
        Cell cell = cells[i];
        int row = cell.getRow();
        int col = cell.getCol();
        wall[row][col] = cell;
    }
}

/** 检查当前的方块是否能够下落，返回 true 能够下落 */
```

```
    private boolean canDrop() {
        Cell[] cells = tetromino.cells;
        for (int i = 0; i < cells.length; i++) {
            Cell cell = cells[i];
            int row = cell.getRow();
            if (row == ROWS - 1) {
                return false;
            }
        }
        for (Cell cell : cells) {// Java 5 以后可以使用
            int row = cell.getRow() + 1;
            int col = cell.getCol();
            if (row >= 0 && row < ROWS && col >= 0 && col <= COLS
                    && wall[row][col] != null) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
            ... ...
    }
}
```

运行上述代码，发现可以实现行的销毁，另外，可以将 Tetromino 类的 randomOne 方法的实现都改成 O 形方块，这样界面上生成的方块就都为 O 行方块，可以方便的测试销毁行的操作。

步骤十七：硬下落流程

硬下落的意思为方块下落到不能下落为止；当不能下落时，让方块着落，然后销毁满的行，并将该流程绑定到空格键上，代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.util.Arrays;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
    private int score;// 分数
    private int lines;// 销毁的行数
    private Cell[][] wall;// 背景墙
    private Tetromino tetromino;// 正在下落的四格方块
    private Tetromino nextOne;// 下一个四格方块
    /** 背景图片 */
    private static BufferedImage background;
```

```
public static BufferedImage T;
public static BufferedImage S;
public static BufferedImage I;
public static BufferedImage L;
public static BufferedImage J;
public static BufferedImage O;
public static BufferedImage Z;
public static final int ROWS = 20;// 背景墙的行数
public static final int COLS = 10;// 背景墙的列数
// 将图片素材，复制到 com.tarena.tetris 包中.
/** 使用静态代码块加载静态的图片 */
static {
    ... ...
}

/**
 * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
 **/
@Override
public void paint(Graphics g) {
    ... ...
}

/** 在 Tetris 添加启动方法 action() */
public void action() {
    wall = new Cell[ROWS][COLS];
    wall[2][2] = new Cell(2, 2, T);
    tetromino = Tetromino.randomOne();
    nextOne = Tetromino.randomOne();

    // 处理键盘按下事件，在按下按键时候执行下落方法
    KeyAdapter l = new KeyAdapter() {
        @Override
        // key 按键 Pressed按下了
        public void keyPressed(KeyEvent e) {
            int key = e.getKeyCode();// [c]

            switch (key) {
            case KeyEvent.VK_DOWN:
                softDropAction();
                break;
            case KeyEvent.VK_RIGHT:
                moveRightAction();
                break;
            case KeyEvent.VK_LEFT:
                moveLeftAction();
                break;


            case KeyEvent.VK_SPACE:
                hardDropAction();
                break;


            }
            repaint();// 再画一次!
        }
    };
    // 下落流程：监听键盘事件->如果下箭头按下->
    // 执行下落算法 tetromino.softDrop()->
    // 修改每个格子对象的数据->调用 repaint()->
    // 尽快调用 paint()->paint 方法会根据当前的数据
```

```
        // 重新绘制界面 -> 看到移动以后的方块了

        // 绑定事件到当前面板
        this.requestFocus();
        this.addKeyListener(l);
    }

    /**
     * 绘制下一个要下落的方块
     *
     * @param g
     */
    private void paintNextOne(Graphics g) {
        ... ...
    }

    /**
     * 绘制正在下落的方块
     *
     * @param g
     */
    public void paintTetromino(Graphics g) {
        ... ...
    }

    public static final int CELL_SIZE = 26;

    /** 画墙 */
    private void paintWall(Graphics g) {
        ... ...
    }

    /** 检查当前正在下落的方块是否出界了 */
    private boolean outOfBounds() {
        ... ...
    }

    /** 检查正在下落的方块是否与墙上的砖块重叠 */
    private boolean coincide() {
        ... ...
    }

    /** 在 Tetris 类上添加方法，向右移动的流程控制 */
    public void moveRightAction() {
        ... ...
    }

    public void moveLeftAction() {
        ... ...
    }

    /** 下落流程控制 */
    public void softDropAction() {
        ... ...
    }

    private static int[] scoreTable = { 0, 1, 10, 50, 100 };

    // 0 1 2 3 4
    private void destoryLines() {
        ... ...
    }
```

```
    private void deleteRow(int row) {
        ... ...
    }

    /**
     * 检查当前行的每个格子,是否是满的，如果满了则返回 true，否则返回 false
     */
    private boolean fullCells(int row) {
        ... ...
    }

    private void landIntoWall() {
        ... ...
    }

    /** 检查当前的方块是否能够下落，返回 true 能够下落 */
    private boolean canDrop() {
        ... ...
    }

    /**
     * 硬下落流程， 下落到不能下落为止 绑定到 空格(VK_SPACE)事件上
     * */

    public void hardDropAction() {
        while (canDrop()) {
            tetromino.softDrop();
        }
        landIntoWall();
        destoryLines();
        tetromino = nextOne;
        nextOne = Tetromino.randomOne();
    }

    public static void main(String[] args) {
        ... ...
    }
}
```

步骤十八：绘制分数

在 Tetris 类中，构建 paintScore 方法，在该方法中使用 Java API 提供的方法将分数绘制到界面上，代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.util.Arrays;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
```

```
public class Tetris extends JPanel {
private int score;// 分数
private int lines;// 销毁的行数
private Cell[][] wall;// 背景墙
private Tetromino tetromino;// 正在下落的四格方块
private Tetromino nextOne;// 下一个四格方块
/** 背景图片 */
private static BufferedImage background;
public static BufferedImage T;
public static BufferedImage S;
public static BufferedImage I;
public static BufferedImage L;
public static BufferedImage J;
public static BufferedImage O;
public static BufferedImage Z;
public static final int ROWS = 20;// 背景墙的行数
public static final int COLS = 10;// 背景墙的列数
// 将图片素材，复制到 com.tarena.tetris 包中.
/** 使用静态代码块加载静态的图片 */
static {
    ... ...
}

/**
 * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
 **/
@Override
public void paint(Graphics g) {
    // 画背景，画墙，画正在下落的方块 画下一个方块...
    g.drawImage(background, 0, 0, null);
    g.translate(15, 15);// 坐标系平移
    paintWall(g);// 画墙
    paintTetromino(g);// 绘制正在下落的方块
    paintNextOne(g);//绘制下一个要下落的方块

    paintScore(g);//绘制分数

}

/** 在 Tetris 添加启动方法 action() */
public void action() {
    ... ...
}

public static final int FONT_COLOR = 0x667799;
public static final int FONT_SIZE = 30;
   /**
    * 绘制分数
    * @param g
    */
private void paintScore(Graphics g) {
    int x = 290;
    int y = 160;
    g.setColor(new Color(FONT_COLOR));
    Font font = g.getFont();// 取得 g 当前字体
```

```
        font = new Font(font.getName(), font.getStyle(), FONT_SIZE);
        g.setFont(font);// 更改了 g 的字体
        String str = "SCORE:" + score;
        g.drawString(str, x, y);
        y += 56;
        str = "LINES:" + lines;
        g.drawString(str, x, y);
    }

    /**
     * 绘制下一个要下落的方块
     *
     * @param g
     */
    private void paintNextOne(Graphics g) {
        ... ...
    }

    /**
     * 绘制正在下落的方块
     *
     * @param g
     */
    public void paintTetromino(Graphics g) {
        ... ...
    }

    public static final int CELL_SIZE = 26;

    /** 画墙 */
    private void paintWall(Graphics g) {
        ... ...
    }

    /** 检查当前正在下落的方块是否出界了 */
    private boolean outOfBounds() {
        ... ...
    }

    /** 检查正在下落的方块是否与墙上的砖块重叠 */
    private boolean coincide() {
        ... ...
    }

    /** 在 Tetris 类上添加方法，向右移动的流程控制 */
    public void moveRightAction() {
        ... ...
    }

    public void moveLeftAction() {
        ... ...
    }

    /** 下落流程控制 */
    public void softDropAction() {
        ... ...
    }

    private static int[] scoreTable = { 0, 1, 10, 50, 100 };

    // 0 1 2 3 4
    private void destoryLines() {
        ... ...
```

```
    }

    private void deleteRow(int row) {
        ... ...
    }

    /**
     * 检查当前行的每个格子,是否是满的，如果满了则返回 true，否则返回 false
     */
    private boolean fullCells(int row) {
        ... ...
    }

    private void landIntoWall() {
        ... ...
    }

    /** 检查当前的方块是否能够下落，返回 true 能够下落 */
    private boolean canDrop() {
        ... ...
    }

    /**
     * 硬下落流程，下落到不能下落为止 绑定到 空格(VK_SPACE)事件上
     * */
    public void hardDropAction() {
        ... ...
    }

    public static void main(String[] args) {
        ... ...
    }
}
```

运行上述代码，界面显示效果如图-10 所示：

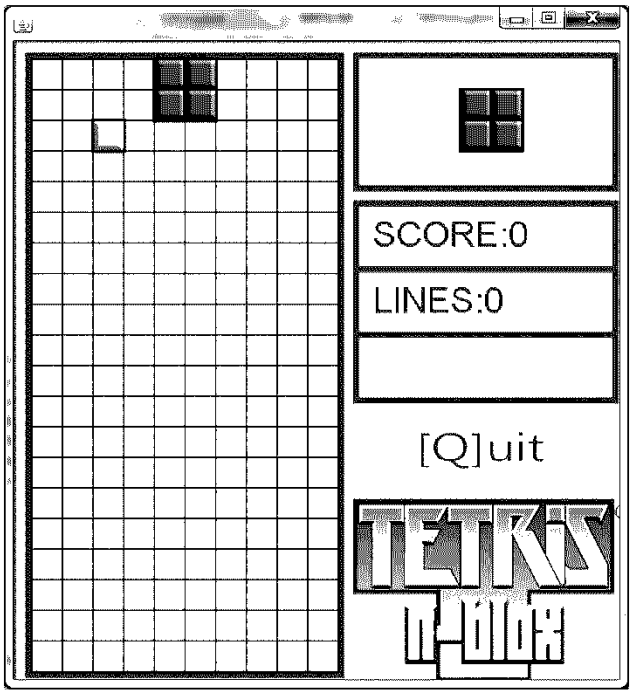

图- 10

观察图-10会发现，在图的右侧出现了“SCORE：0”和“LINES：0”字样的文字。

步骤十九：旋转流程控制

各种形状的方块都可以实现旋转，有的方块的旋转状态有两种，有的方块的旋转状态为四种，下面首先以T形为例能介绍旋转算法。T形方块旋转后有四种状态，如图-11所示：

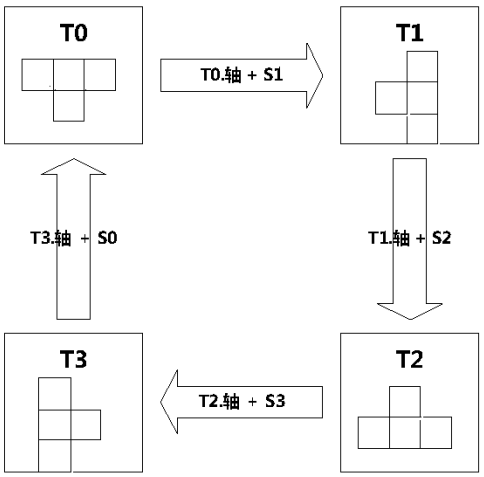

图 - 11

在图-11中，显示了T形方块进行旋转后的四种状态。并且在旋转过程中，展示了如何从一种状态到另一种状态，例如：从T0到T1 ，坐标换算的公式为T0.轴+S1，那么这里的T0.轴和S1分别指什么呢？请看图-12 ,该图中的cells即为Tetromino类的cells属性。

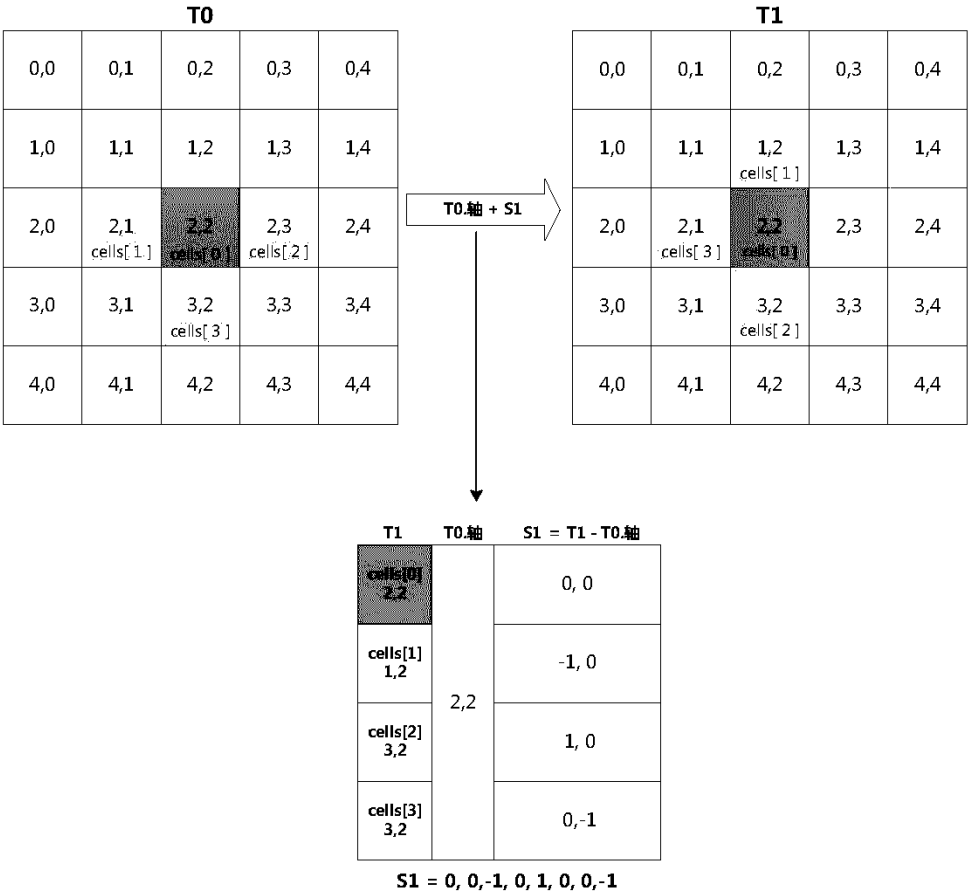

图 – 12

观察图-12 可以看出，T0.轴指的是 cells[0]，其在图中的行列坐标为（2，2）。另外，计算 S1 的公式为：S1 = T1 – T0.轴，其中，T1 为旋转后的 T 形方块。以 cells[0]为轴，从 T0 到 T1 坐标变换为：

```
cells [ 0 ].row = cells [ 0 ].row + (0)
cells [ 0 ].col = cells [ 0 ].col + (0)
cells [ 1 ].row = cells [ 0 ].row + (-1)
cells [ 1 ].col = cells [ 0 ].col + (0)
cells [ 2 ].row = cells [ 0 ].row + (1)
cells [ 2 ].col = cells [ 0 ].col + (0)
cells [ 3 ].row = cells [ 0 ].row + (0)
cells [ 3 ].col = cells [ 0 ].col + (-1)
```

以上代码中，等号前面的代码即为 T1 状态的四个格子的行列坐标；等号后边的代码即为“T0.轴+S1”。

根据以上分析，可以得出从 T0 旋转到 T1，变换坐标用的八个值为：

```
0, 0, -1, 0, 1, 0, 0, -1
```

以上八个值即为 S1。

以此类推，从 T1 旋转到 T2、T2 旋转到 T3、T3 旋转到 T0 的原理和 T0 旋转到 T1 相同，

如图-13 所示：

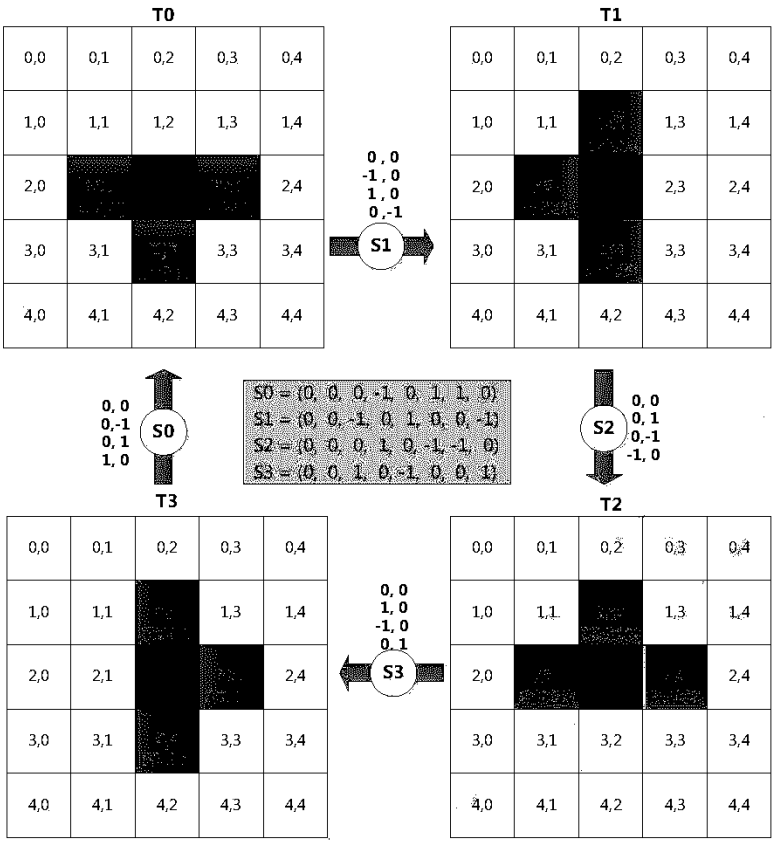

图 – 13

L 形旋转过程坐标变换如图-14 所示：

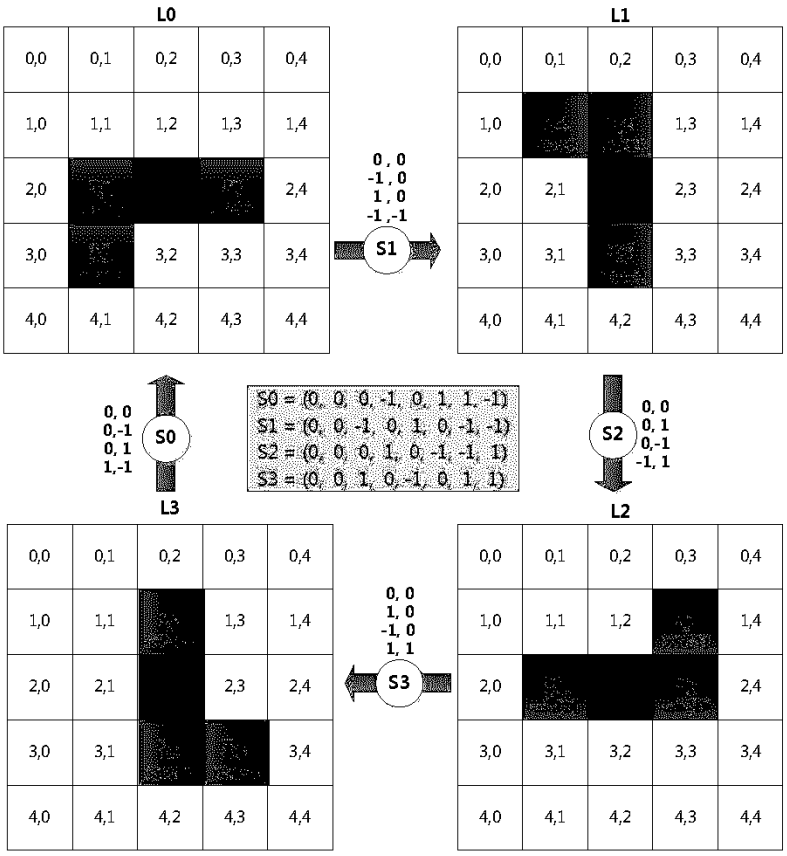

图 – 14

J 形旋转过程坐标变换如图-15 所示：

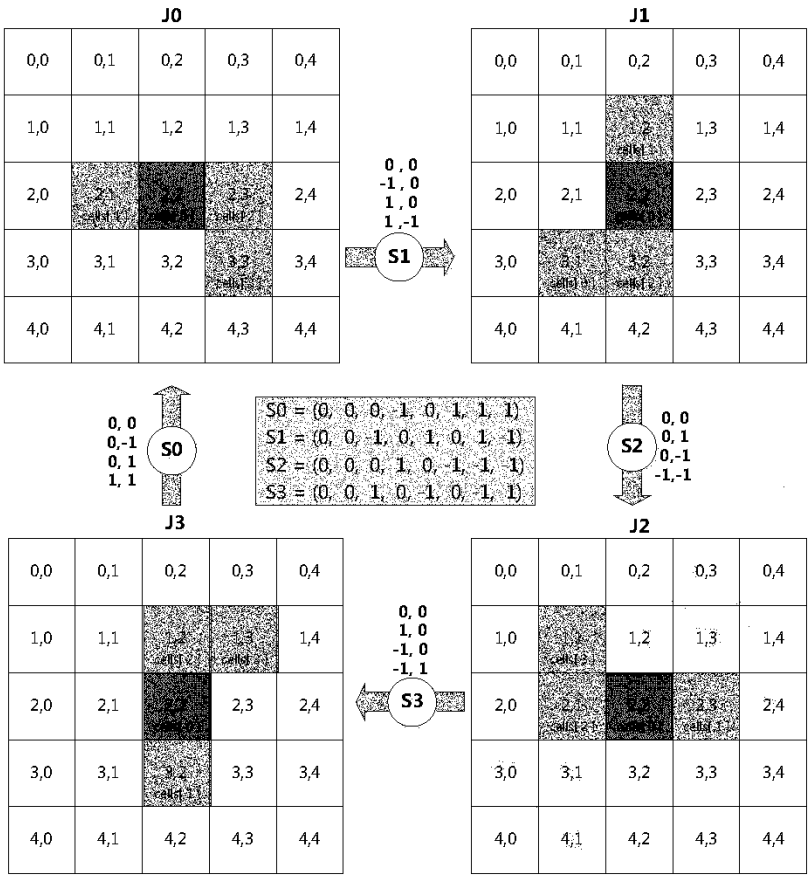

图 – 15

I 形旋转过程坐标变换如图-16 所示：

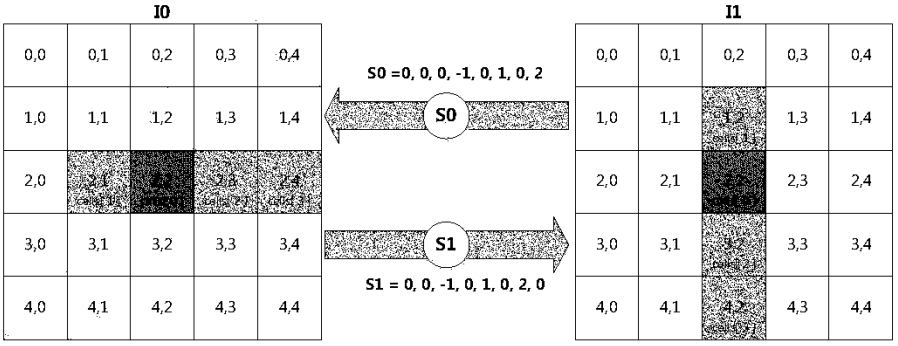

图 – 16

O 形旋转过程坐标变换如图-17 所示：

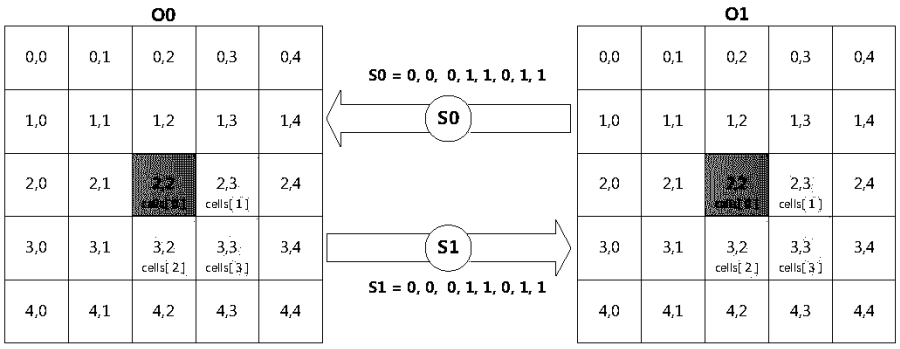

图 – 17

S 形旋转过程坐标变换如图-18 所示：

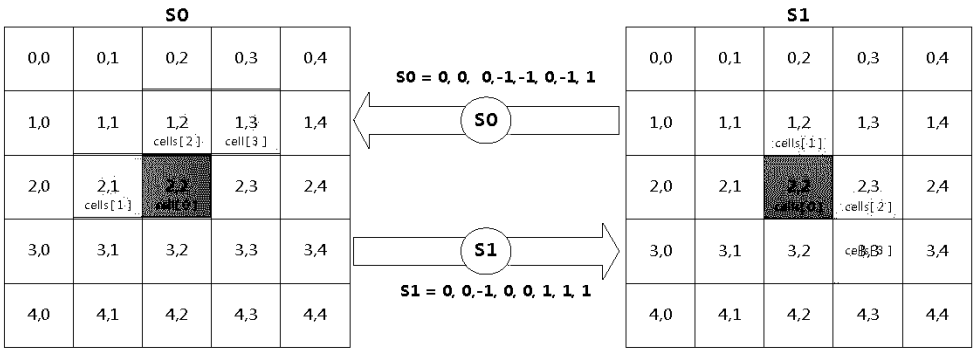

图 – 18

Z 形旋转过程坐标变换如图-19 所示：

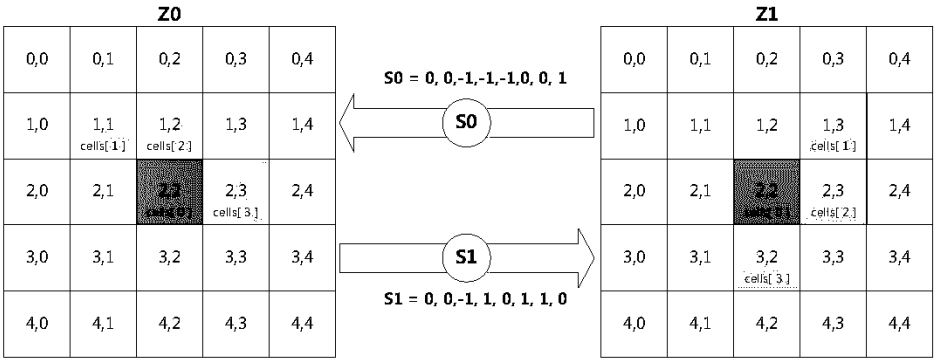

图 – 19

实现方块旋转的代码如下所示：

```
package com.tarena.tetris;

import java.util.Arrays;
import java.util.Random;

/**
 * 4 格方块
 */
public abstract class Tetromino {
 protected Cell[] cells = new Cell[4];
```

```
/** 旋转状态 */

protected State[] states;

/** 旋转状态的序号 states[index%states.length] */

protected int index = 10000;

/** 内部类 */

protected class State {
    int row0, col0, row1, col1, row2, col2, row3, col3;

    public State(int row0, int col0, int row1, int col1, int row2,
            int col2, int row3, int col3) {
        this.row0 = row0;
        this.col0 = col0;
        this.row1 = row1;
        this.col1 = col1;
        this.row2 = row2;
        this.col2 = col2;
        this.row3 = row3;
        this.col3 = col3;
    }
}

/** 向右转 */

public void rotateRight() {
    // 取得变换的下个数据状态 states[n]
    // 取得当前轴的 row, col
    // 旋转以后的数据=(row,col) + states[n]
    index++;
    State s = states[index % states.length];
    Cell o = cells[0];
    int row = o.getRow();
    int col = o.getCol();
    cells[1].setRow(row + s.row1);
    cells[1].setCol(col + s.col1);
    cells[2].setRow(row + s.row2);
    cells[2].setCol(col + s.col2);
    cells[3].setRow(row + s.row3);
    cells[3].setCol(col + s.col3);
}

/** 工厂方法，随机产生 4 格方块 */
public static Tetromino randomOne() {
    ... ...
}

/**
 * 方块下落一个格子
 */
public void softDrop() {
    ... ...
}
```

```
    /**
     * 方块左移一个格子
     */
    public void moveLeft() {
        ... ...
    }

    /**
     * 方块右移一个格子
     */
    public void moveRight() {
        ... ...
    }

    /**
     * 覆盖 toString 方法，显示方块中每个格子的行列信息
     */
    public String toString() {
        return Arrays.toString(cells);
    }
}

/**
 * T 形方块
 */
class T extends Tetromino {
    public T() {
        cells[0] = new Cell(0, 4, Tetris.T);
        cells[1] = new Cell(0, 3, Tetris.T);
        cells[2] = new Cell(0, 5, Tetris.T);
        cells[3] = new Cell(1, 4, Tetris.T);

        states = new State[4];
        states[0] = new State(0, 0, 0, -1, 0, 1, 1, 0);
        states[1] = new State(0, 0, -1, 0, 1, 0, 0, -1);
        states[2] = new State(0, 0, 0, 1, 0, -1, -1, 0);
        states[3] = new State(0, 0, 1, 0, -1, 0, 0, 1);

    }
}

/**
 * I 形方块
 */
class I extends Tetromino {
    public I() {
        cells[0] = new Cell(0, 4, Tetris.I);
        cells[1] = new Cell(0, 3, Tetris.I);
        cells[2] = new Cell(0, 5, Tetris.I);
        cells[3] = new Cell(0, 6, Tetris.I);

        states = new State[] { new State(0, 0, 0, -1, 0, 1, 0, 2),
                new State(0, 0, -1, 0, 1, 0, 2, 0) };

    }
}

/**
 * S 形方块
 */
class S extends Tetromino {
    public S() {
```

```
        cells[0] = new Cell(1, 4, null);
        cells[1] = new Cell(1, 3, null);
        cells[2] = new Cell(0, 4, null);
        cells[3] = new Cell(0, 5, null);

        states = new State[] { new State(0, 0, 0, -1, -1, 0, -1, 1),
                new State(0, 0, -1, 0, 0, 1, 1, 1) };

    }
}

/**
 * Z 形方块
 */
class Z extends Tetromino {
    public Z() {
        cells[0] = new Cell(1, 4, Tetris.Z);
        cells[1] = new Cell(0, 3, Tetris.Z);
        cells[2] = new Cell(0, 4, Tetris.Z);
        cells[3] = new Cell(1, 5, Tetris.Z);

        states = new State[] { new State(0, 0, -1, -1, -1, 0, 0, 1),
                new State(0, 0, -1, 1, 0, 1, 1, 0) };

    }
}

/**
 * O 形方块
 */
class O extends Tetromino {
    public O() {
        cells[0] = new Cell(0, 4, Tetris.O);
        cells[1] = new Cell(0, 5, Tetris.O);
        cells[2] = new Cell(1, 4, Tetris.O);
        cells[3] = new Cell(1, 5, Tetris.O);

        states = new State[] { new State(0, 0, 0, 1, 1, 0, 1, 1),
                new State(0, 0, 0, 1, 1, 0, 1, 1) };

    }
}

/**
 * L 形方块
 */
class L extends Tetromino {
    public L() {
        cells[0] = new Cell(0, 4, Tetris.L);
        cells[1] = new Cell(0, 3, Tetris.L);
        cells[2] = new Cell(0, 5, Tetris.L);
        cells[3] = new Cell(1, 3, Tetris.L);

        states = new State[] { new State(0, 0, 0, 1, 0, -1, -1, 1),
                new State(0, 0, 1, 0, -1, 0, 1, 1),
                new State(0, 0, 0, -1, 0, 1, 1, -1),
                new State(0, 0, -1, 0, 1, 0, -1, -1) };

    }
}
```

```
/**
 * J形方块
 */
class J extends Tetromino {
public J() {
    cells[0] = new Cell(0, 4, Tetris.J);
    cells[1] = new Cell(0, 3, Tetris.J);
    cells[2] = new Cell(0, 5, Tetris.J);
    cells[3] = new Cell(1, 5, Tetris.J);


    states = new State[] { new State(0, 0, 0, -1, 0, 1, 1, 1),
            new State(0, 0, -1, 0, 1, 0, 1, -1),
            new State(0, 0, 0, 1, 0, -1, -1, -1),
            new State(0, 0, 1, 0, -1, 0, -1, 1) };

}
}
```

重构 Tetris 类的代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.util.Arrays;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
private int score;// 分数
private int lines;// 销毁的行数
private Cell[][] wall;// 背景墙
private Tetromino tetromino;// 正在下落的四格方块
private Tetromino nextOne;// 下一个四格方块
/** 背景图片 */
private static BufferedImage background;
public static BufferedImage T;
public static BufferedImage S;
public static BufferedImage I;
public static BufferedImage L;
public static BufferedImage J;
public static BufferedImage O;
public static BufferedImage Z;
public static final int ROWS = 20;// 背景墙的行数
public static final int COLS = 10;// 背景墙的列数
// 将图片素材，复制到 com.tarena.tetris 包中.
/** 使用静态代码块加载静态的图片 */
static {
```

```
    ... ...
}

/**
 * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
 **/
@Override
public void paint(Graphics g) {
    ... ...
}

/** 在 Tetris 添加启动方法 action() */
public void action() {
    wall = new Cell[ROWS][COLS];
    wall[2][2] = new Cell(2, 2, T);
    tetromino = Tetromino.randomOne();
    nextOne = Tetromino.randomOne();

    // 处理键盘按下事件，在按下按键时候执行下落方法
    KeyAdapter l = new KeyAdapter() {
        @Override
        // key 按键 Pressed按下了
        public void keyPressed(KeyEvent e) {
            int key = e.getKeyCode();// [c]

            switch (key) {
            case KeyEvent.VK_DOWN:
                softDropAction();
                break;
            case KeyEvent.VK_RIGHT:
                moveRightAction();
                break;
            case KeyEvent.VK_LEFT:
                moveLeftAction();
                break;
            case KeyEvent.VK_SPACE:
                hardDropAction();
                break;


            case KeyEvent.VK_UP:
                rotateRightAction();
                break;


            }
            repaint();// 再画一次!
        }
    };
    // 下落流程：监听键盘事件->如果下箭头按下->
    // 执行下落算法 tetromino.softDrop()->
    // 修改每个格子对象的数据->调用 repaint()->
    // 尽快调用 paint()->paint 方法会根据当前的数据
    // 重新绘制界面 -> 看到移动以后的方块了

    // 绑定事件到当前面板
    this.requestFocus();
    this.addKeyListener(l);

}

public static final int FONT_COLOR = 0x667799;
public static final int FONT_SIZE = 30;
```

```
    /**
     * 绘制分数
     * @param g
     */
    private void paintScore(Graphics g) {
        ... ...
    }

    /**
     * 绘制下一个要下落的方块
     *
     * @param g
     */
    private void paintNextOne(Graphics g) {
        ... ...
    }

    /**
     * 绘制正在下落的方块
     *
     * @param g
     */
    public void paintTetromino(Graphics g) {
        ... ...
    }

    public static final int CELL_SIZE = 26;

    /** 画墙 */
    private void paintWall(Graphics g) {
        ... ...
    }

    /** 检查当前正在下落的方块是否出界了 */
    private boolean outOfBounds() {
        ... ...
    }

    /** 检查正在下落的方块是否与墙上的砖块重叠 */
    private boolean coincide() {
        ... ...
    }

    /** 在 Tetris 类上添加方法，向右移动的流程控制 */
    public void moveRightAction() {
        ... ...
    }

    public void moveLeftAction() {
        ... ...
    }

    /** 下落流程控制 */
    public void softDropAction() {
        ... ...
    }

    private static int[] scoreTable = { 0, 1, 10, 50, 100 };

    // 0 1 2 3 4
    private void destoryLines() {
        ... ...
    }
```

```
    private void deleteRow(int row) {
        ... ...
    }

    /**
     * 检查当前行的每个格子,是否是满的，如果满了则返回 true，否则返回 false
     */
    private boolean fullCells(int row) {
        ... ...
    }

    private void landIntoWall() {
        ... ...
    }

    /** 检查当前的方块是否能够下落，返回 true 能够下落 */
    private boolean canDrop() {
        ... ...
    }

    /**
     * 硬下落流程，下落到不能下落为止 绑定到 空格(VK_SPACE)事件上
     * */
    public void hardDropAction() {
        ... ...
    }

    /** 在 Tetris 类中添加 旋转流程控制方法 */

    public void rotateRightAction() {
        tetromino.rotateRight();
        if (outOfBounds() || coincide()) {
            tetromino.rotateLeft();
        }
    }

    public static void main(String[] args) {
        ... ...
    }
}
```

步骤二十：实现自由下落

在 action 方法中添加定时计划任务，实现自由下落功能，代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.util.Arrays;
import java.util.Timer;
import java.util.TimerTask;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
```

```
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块
 /** 背景图片 */
 private static BufferedImage background;
 public static BufferedImage T;
 public static BufferedImage S;
 public static BufferedImage I;
 public static BufferedImage L;
 public static BufferedImage J;
 public static BufferedImage O;
 public static BufferedImage Z;
 public static final int ROWS = 20;// 背景墙的行数
 public static final int COLS = 10;// 背景墙的列数

 /** 在 Tetris 类中增加定时器 */
 private Timer timer;


 // 将图片素材，复制到 com.tarena.tetris 包中.
 /** 使用静态代码块加载静态的图片 */
 static {
     ...  ...
 }

 /**
  * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
  **/
 @Override
 public void paint(Graphics g) {
     ...  ...
 }

 /** 在 Tetris 添加启动方法 action() */
 public void action() {
     wall = new Cell[ROWS][COLS];
     wall[2][2] = new Cell(2, 2, T);
     tetromino = Tetromino.randomOne();
     nextOne = Tetromino.randomOne();

     // 处理键盘按下事件，在按下按键时候执行下落方法
     KeyAdapter l = new KeyAdapter() {
         @Override
         // key 按键 Pressed按下了
         public void keyPressed(KeyEvent e) {
             int key = e.getKeyCode();// [c]

             switch (key) {
             case KeyEvent.VK_DOWN:
                 softDropAction();
                 break;
             case KeyEvent.VK_RIGHT:
                 moveRightAction();
```

```
                    break;
                case KeyEvent.VK_LEFT:
                    moveLeftAction();
                    break;
                case KeyEvent.VK_SPACE:
                    hardDropAction();
                    break;
                case KeyEvent.VK_UP:
                    rotateRightAction();
                    break;
                }
                repaint();// 再画一次!
            }
        };
        // 下落流程：监听键盘事件->如果下箭头按下->
        // 执行下落算法 tetromino.softDrop()->
        // 修改每个格子对象的数据->调用 repaint()->
        // 尽快调用 paint()->paint 方法会根据当前的数据
        // 重新绘制界面 -> 看到移动以后的方块了

        // 绑定事件到当前面板
        this.requestFocus();
        this.addKeyListener(l);

        // 在 Action 方法中添加,定时计划任务
        timer = new Timer();
        timer.schedule(new TimerTask() {
            public void run() {
                softDropAction();
                repaint();
            }
        }, 10, 10);

    }

    public static final int FONT_COLOR = 0x667799;
    public static final int FONT_SIZE = 30;
    /**
     * 绘制分数
     * @param g
     */
    private void paintScore(Graphics g) {
        ... ...
    }

    /**
     * 绘制下一个要下落的方块
     *
     * @param g
     */
    private void paintNextOne(Graphics g) {
        ... ...
    }

    /**
     * 绘制正在下落的方块
     *
     * @param g
     */
    public void paintTetromino(Graphics g) {
        ... ...
```

```
}

public static final int CELL_SIZE = 26;

/** 画墙 */
private void paintWall(Graphics g) {
    ... ...
}

/** 检查当前正在下落的方块是否出界了 */
private boolean outOfBounds() {
    ... ...
}

/** 检查正在下落的方块是否与墙上的砖块重叠 */
private boolean coincide() {
    ... ...
}

/** 在 Tetris 类上添加方法，向右移动的流程控制 */
public void moveRightAction() {
    ... ...
}

public void moveLeftAction() {
    ... ...
}

/** 下落流程控制 */
public void softDropAction() {
    ... ...
}

private static int[] scoreTable = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4
private void destoryLines() {
    ... ...
}

private void deleteRow(int row) {
    ... ...
}

/**
 * 检查当前行的每个格子,是否是满的，如果满了则返回 true，否则返回 false
 */
private boolean fullCells(int row) {
    ... ...
}

private void landIntoWall() {
    ... ...
}

/** 检查当前的方块是否能够下落，返回 true 能够下落 */
private boolean canDrop() {
    ... ...
}

/**
 * 硬下落流程， 下落到不能下落为止 绑定到 空格(VK_SPACE)事件上
 * */
public void hardDropAction() {
```

```
    ...  ***
}

/** 在 Tetris 类中添加 旋转流程控制方法 */
public void rotateRightAction() {
    ...  ***
}

public static void main(String[] args) {
    ...  ***
}
}
```

步骤二十二：开始流程控制、暂停流程控制、继续流程控制以及结束流程控制

本案例，游戏分为三种状态：Running、Pause、GameOver，这三种状态之间的相互切换如图-20 所示：

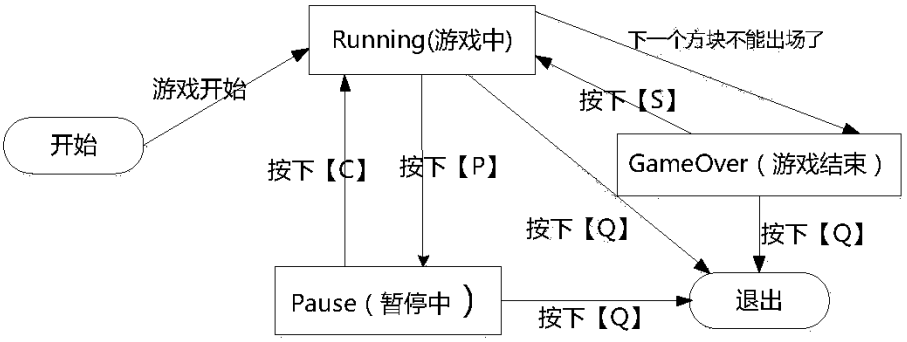

图 – 20

从图-20 可以看出，游戏开始首先进入到 Running 状态， 在 Running 状态时按下键【P】则可以进入 Pause 状态； 在 Pause 状态按下键【C】则转换到 Running 状态； 在 Running 状态如果下一个方块不能出场了则进入 GameOver 状态； 在 GameOver 状态按下键【S】则转换到 Running 状态；在 Running 状态、Pause 状态、GameOver 状态时，按下键【Q】则，退出系统。图-21 展示了本案例中各个键的用途。

图 – 21

各个状态之间的转换的实现过程如下：

1．在 Tetris 中添加状态管理的属性 state，同时增加三个常量 RUNNING、PAUSE、GAME_OVER，值分别为 0、1、2。

2．在定时器中，如果为 Running 状态，则执行下落流程。在此过程中，发现游戏结束了，则将状态更新为 GameOver。

3．键盘事件中，优先处理按键【Q】，退出系统。如果按下的为【P】键，则将状态转换为 Pause 状态；如果为 Pause 状态，则只能响应按键【C】，继续游戏，并将游戏状态更新为 Running；当游戏状态为 GameOver 时，可以响应按键【S】，重新开始游戏，并将游戏状态更新为 Running。

4．在下落流程中，如果游戏结束了，则将游戏状态更新 GameOver。

代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.util.Arrays;
import java.util.Timer;
import java.util.TimerTask;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {

    /** 游戏的当前状态: RUNNING PAUSE GAME_OVER */
    private int state;
    public static final int RUNNING = 0;
    public static final int PAUSE = 1;
    public static final int GAME_OVER = 2;


    private int score;// 分数
    private int lines;// 销毁的行数
    private Cell[][] wall;// 背景墙
    private Tetromino tetromino;// 正在下落的四格方块
    private Tetromino nextOne;// 下一个四格方块
    /** 背景图片 */
    private static BufferedImage background;

    private static BufferedImage gameOver;
    private static BufferedImage pause;


    public static BufferedImage T;
    public static BufferedImage S;
```

```
public static BufferedImage I;
public static BufferedImage L;
public static BufferedImage J;
public static BufferedImage O;
public static BufferedImage Z;
public static final int ROWS = 20;// 背景墙的行数
public static final int COLS = 10;// 背景墙的列数

/** 在 Tetris 类中增加定时器 */
private Timer timer;

// 将图片素材，复制到 com.tarena.tetris 包中.
/** 使用静态代码块加载静态的图片 */
static {
    try {
        // Tetris.class 的同一个包中找 "tetris.png"
        background = ImageIO.read(Tetris.class.getResource("tetris.png"));

        gameOver = ImageIO.read(Tetris.class.getResource("game-over.png"));
        pause = ImageIO.read(Tetris.class.getResource("pause.png"));

        T = ImageIO.read(Tetris.class.getResource("T.png"));
        I = ImageIO.read(Tetris.class.getResource("I.png"));
        S = ImageIO.read(Tetris.class.getResource("S.png"));
        Z = ImageIO.read(Tetris.class.getResource("Z.png"));
        J = ImageIO.read(Tetris.class.getResource("J.png"));
        L = ImageIO.read(Tetris.class.getResource("L.png"));
        O = ImageIO.read(Tetris.class.getResource("O.png"));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * JPanel paint() paint画 重写 paint() 修改原有的绘制方法
 **/
@Override
public void paint(Graphics g) {
    // 画背景，画墙，画正在下落的方块 画下一个方块...
    g.drawImage(background, 0, 0, null);
    g.translate(15, 15);// 坐标系平移
    paintWall(g);// 画墙
    paintTetromino(g);// 绘制正在下落的方块
    paintNextOne(g);//绘制下一个要下落的方块
    paintScore(g);//绘制分数

    paintState(g);// 绘制游戏的状态

}

private void paintState(Graphics g) {
    switch (state) {
    case PAUSE:
        g.drawImage(pause, -15, -15, null);
        break;
    case GAME_OVER:
        g.drawImage(gameOver, -15, -15, null);
```

```
            break;
        }
    }

    /** 在 Tetris 添加启动方法 action() */
    public void action() {
        wall = new Cell[ROWS][COLS];
        //wall[2][2] = new Cell(2, 2, T);
        tetromino = Tetromino.randomOne();
        nextOne = Tetromino.randomOne();

        state = RUNNING;

        // 处理键盘按下事件，在按下按键时候执行下落方法
        KeyAdapter l = new KeyAdapter() {
            @Override
            // key 按键 Pressed 按下了

            public void keyPressed(KeyEvent e) {
                int key = e.getKeyCode();
                switch (state) {
                case GAME_OVER:
                    processGameoverKey(key);
                    break;
                case PAUSE:
                    processPauseKey(key);
                    break;
                case RUNNING:
                    processRunningKey(key);
                }
                repaint();
            }

        };
        // 下落流程：监听键盘事件->如果下箭头按下->
        // 执行下落算法 tetromino.softDrop()->
        // 修改每个格子对象的数据->调用 repaint()->
        // 尽快调用 paint()->paint 方法会根据当前的数据
        // 重新绘制界面 -> 看到移动以后的方块了

        // 绑定事件到当前面板
        this.requestFocus();
        this.addKeyListener(l);

        // 在 Action 方法中添加,定时计划任务
        timer = new Timer();

        timer.schedule(new TimerTask() {
            public void run() {
                if (state == RUNNING) {
                    softDropAction();
                }
                repaint();
            }
        }, 10, 10);

    }
```

```
private void processPauseKey(int key) {
    switch (key) {
    case KeyEvent.VK_Q:
        System.exit(0);
        break;
    case KeyEvent.VK_C:
        state = RUNNING;
        break;
    }
}

protected void processRunningKey(int key) {
    switch (key) {
    case KeyEvent.VK_Q:
        System.exit(0);
        break;
    case KeyEvent.VK_RIGHT:
        Tetris.this.moveRightAction();
        break;
    case KeyEvent.VK_LEFT:
        Tetris.this.moveLeftAction();
        break;
    case KeyEvent.VK_DOWN:
        softDropAction();
        break;
    case KeyEvent.VK_SPACE:
        hardDropAction();
        break;
    case KeyEvent.VK_UP:
        rotateRightAction();
        break;
    case KeyEvent.VK_P:
        state = PAUSE;
        break;
    }
}

protected void processGameoverKey(int key) {
    switch (key) {
    case KeyEvent.VK_Q:
        System.exit(0);
        break;
    case KeyEvent.VK_S:
        /** 游戏重新开始 */
        this.lines = 0;
        this.score = 0;
        this.wall = new Cell[ROWS][COLS];
        this.tetromino = Tetromino.randomOne();
        this.nextOne = Tetromino.randomOne();
        this.state = RUNNING;
        this.index = 0;
        break;
    }
}


public static final int FONT_COLOR = 0x667799;
public static final int FONT_SIZE = 30;
   /**
    * 绘制分数
    * @param g
    */
private void paintScore(Graphics g) {
    ...  ...
}
```

```
/**
 * 绘制下一个要下落的方块
 *
 * @param g
 */
private void paintNextOne(Graphics g) {
    ... ...
}

/**
 * 绘制正在下落的方块
 *
 * @param g
 */
public void paintTetromino(Graphics g) {
    ... ...
}

public static final int CELL_SIZE = 26;

/** 画墙 */
private void paintWall(Graphics g) {
    ... ...
}

/** 检查当前正在下落的方块是否出界了 */
private boolean outOfBounds() {
    ... ...
}

/** 检查正在下落的方块是否与墙上的砖块重叠 */
private boolean coincide() {
    ... ...
}

/** 在 Tetris 类上添加方法，向右移动的流程控制 */
public void moveRightAction() {
    ... ...
}

public void moveLeftAction() {
    ... ...
}

/** 下落流程控制 */
public void softDropAction() {
    if (canDrop()) {
        tetromino.softDrop();
    } else {
        landIntoWall();
        destoryLines();

        if (isGameOver()) {
          state = GAME_OVER;
        } else {
          tetromino = nextOne;
          nextOne = Tetromino.randomOne();
        }

    }
}

private static int[] scoreTable = { 0, 1, 10, 50, 100 };
```

```
// 0 1 2 3 4
private void destoryLines() {
    ... ...
}

private void deleteRow(int row) {
    ... ...
}

/**
 * 检查当前行的每个格子,是否是满的，如果满了则返回 true，否则返回 false
 */
private boolean fullCells(int row) {
    ... ...
}

private void landIntoWall() {
    ... ...
}

/** 检查当前的方块是否能够下落，返回 true 能够下落 */
private boolean canDrop() {
    ... ...
}

/**
 * 硬下落流程，下落到不能下落为止 绑定到 空格(VK_SPACE)事件上
 * */
public void hardDropAction() {
    while (canDrop()) {
        tetromino.softDrop();
    }
    landIntoWall();
    destoryLines();

    if (isGameOver()) {
        state = GAME_OVER;
    } else {
        tetromino = nextOne;
        nextOne = Tetromino.randomOne();
    }

}

/** 在 Tetris 类中添加 旋转流程控制方法 */
public void rotateRightAction() {
    ... ...
}


/** 检查游戏是否结束 */
private boolean isGameOver() {
    // 如果下一个方块没有出场位置了，则游戏结束
    // 就是：下一个出场方块每个格子行列对应的
    // 墙上位置如果有格子，就游戏结束
    Cell[] cells = nextOne.cells;
    for (Cell cell : cells) {
        int row = cell.getRow();
        int col = cell.getCol();
        if (wall[row][col] != null) {
            return true;
```

```
            }
        }
        return false;
    }

    public static void main(String[] args) {
        ... ...
    }
}
```

步骤二十三：处理下落的级别

设置游戏中方块下落的初始速度为 40，当销毁掉的行数为 100 行时，则速度 speed 加快 1。游戏级别初始值为 1，随着下落速度的加快，级别增加。当定时器运行 speed 次时，方块下落一步。每次暂停在开始以后，定时器执行的次数从新开始计数。实现代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.util.Arrays;
import java.util.Timer;
import java.util.TimerTask;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
 /** 游戏的当前状态: RUNNING PAUSE GAME_OVER */
 private int state;
 public static final int RUNNING = 0;
 public static final int PAUSE = 1;
 public static final int GAME OVER = 2;

 private int score;// 分数
 private int lines;// 销毁的行数
 private Cell[][] wall;// 背景墙
 private Tetromino tetromino;// 正在下落的四格方块
 private Tetromino nextOne;// 下一个四格方块
 /** 背景图片 */
 private static BufferedImage background;
 private static BufferedImage gameOver;
 private static BufferedImage pause;
 public static BufferedImage T;
 public static BufferedImage S;
 public static BufferedImage I;
 public static BufferedImage L;
 public static BufferedImage J;
 public static BufferedImage O;
 public static BufferedImage Z;
 public static final int ROWS = 20;// 背景墙的行数
```

```
public static final int COLS = 10;// 背景墙的列数

/** 在 Tetris 类中增加定时器 */
private Timer timer;

/** 速度 */
private int speed;
/** 难度级别 */
private int level;
/** 下落计数器 当 index%speed==0 时候下落一次 */
private int index;

// 将图片素材，复制到 com.tarena.tetris 包中.
/** 使用静态代码块加载静态的图片 */
static {
    ... ...
}

/**
 * JPanel paint() paint 画 重写 paint() 修改原有的绘制方法
 **/
@Override
public void paint(Graphics g) {
    ... ...
}

private void paintState(Graphics g) {
    ... ...
}

/** 在 Tetris 添加启动方法 action() */
public void action() {
    wall = new Cell[ROWS][COLS];
    //wall[2][2] = new Cell(2, 2, T);
    tetromino = Tetromino.randomOne();
    nextOne = Tetromino.randomOne();
    state = RUNNING;
    // 处理键盘按下事件，在按下按键时候执行下落方法
    KeyAdapter l = new KeyAdapter() {
        @Override
        // key 按键 Pressed 按下了
        public void keyPressed(KeyEvent e) {
            int key = e.getKeyCode();
            switch (state) {
            case GAME_OVER:
                processGameoverKey(key);
                break;
            case PAUSE:
                processPauseKey(key);
                break;
            case RUNNING:
                processRunningKey(key);
            }
            repaint();
        }
    };
    // 下落流程：监听键盘事件->如果下箭头按下->
    // 执行下落算法 tetromino.softDrop()->
    // 修改每个格子对象的数据->调用 repaint()->
```

```
        // 尽快调用 paint()->paint 方法会根据当前的数据
        // 重新绘制界面 -> 看到移动以后的方块了

        // 绑定事件到当前面板
        this.requestFocus();
        this.addKeyListener(l);

        // 在 Action 方法中添加,定时计划任务
        timer = new Timer();

        timer.schedule(new TimerTask() {
            public void run() {
                speed = 40 - (lines / 100);
                speed = speed < 1 ? 1 : speed;
                level = 41 - speed;
                if (state == RUNNING && index % speed == 0) {
                    softDropAction();
                }
                index++;
                repaint();
            }
        }, 10, 10);

    }
    private void processPauseKey(int key) {
        switch (key) {
        case KeyEvent.VK_Q:
            System.exit(0);
            break;
        case KeyEvent.VK_C:

            index = 0;

            state = RUNNING;
            break;
        }
    }

    protected void processRunningKey(int key) {
        ...  ...
    }

    protected void processGameoverKey(int key) {
        ...  ...
    }

    public static final int FONT COLOR = 0x667799;
    public static final int FONT SIZE = 30;
       /**
        * 绘制分数
        * @param g
        */
    private void paintScore(Graphics g) {
        int x = 290;
        int y = 160;
        g.setColor(new Color(FONT_COLOR));
        Font font = g.getFont();// 取得 g 当前字体
        font = new Font(font.getName(), font.getStyle(), FONT SIZE);
        g.setFont(font);// 更改了 g 的字体
        String str = "SCORE:" + score;
        g.drawString(str, x, y);
```

```
        y += 56;
        str = "LINES:" + lines;
        g.drawString(str, x, y);

        y += 56;
        g.drawString("LEVEL:" + level, x, y);

    }

    /**
     * 绘制下一个要下落的方块
     *
     * @param g
     */
    private void paintNextOne(Graphics g) {
        ... ...
    }

    /**
     * 绘制正在下落的方块
     *
     * @param g
     */
    public void paintTetromino(Graphics g) {
        ... ...
    }

    public static final int CELL_SIZE = 26;

    /** 画墙 */
    private void paintWall(Graphics g) {
        ... ...
    }

    /** 检查当前正在下落的方块是否出界了 */
    private boolean outOfBounds() {
        ... ...
    }

    /** 检查正在下落的方块是否与墙上的砖块重叠 */
    private boolean coincide() {
        ... ...
    }

    /** 在 Tetris 类上添加方法，向右移动的流程控制 */
    public void moveRightAction() {
        ... ...
    }

    public void moveLeftAction() {
        ... ...
    }

    /** 下落流程控制 */
    public void softDropAction() {
        ... ...
    }

    private static int[] scoreTable = { 0, 1, 10, 50, 100 };

    // 0 1 2 3 4
    private void destoryLines() {
        ... ...
```

```
    }

    private void deleteRow(int row) {
        ... ...
    }

    /**
     * 检查当前行的每个格子,是否是满的，如果满了则返回 true，否则返回 false
     */
    private boolean fullCells(int row) {
        ... ...
    }

    private void landIntoWall() {
        ... ...
    }

    /** 检查当前的方块是否能够下落，返回 true 能够下落 */
    private boolean canDrop() {
        ... ...
    }

    /**
     * 硬下落流程，下落到不能下落为止 绑定到 空格(VK_SPACE)事件上
     * */
    public void hardDropAction() {
        ... ...
    }

    /** 在 Tetris 类中添加 旋转流程控制方法 */
    public void rotateRightAction() {
        ... ...
    }

    /** 检查游戏是否结束 */
    private boolean isGameOver() {
        ... ...
    }

    public static void main(String[] args) {
        ... ...
    }
}
```

4. 完整代码

本案例中，Cell 类的完整代码如下所示：

```
package com.tarena.tetris;

import java.awt.image.BufferedImage;

/**
 * 格子
 */
public class Cell {
    private int row;//格子的行
    private int col;//格子的列
    private BufferedImage image;//格子的贴图

    public Cell(int row, int col, BufferedImage image) {
```

```
        super();
        this.row = row;
        this.col = col;
        this.image = image;
    }

    public int getRow() {
        return row;
    }

    public void setRow(int row) {
        this.row = row;
    }

    public int getCol() {
        return col;
    }

    public void setCol(int col) {
        this.col = col;
    }

    public BufferedImage getImage() {
        return image;
    }

    public void setImage(BufferedImage image) {
        this.image = image;
    }

    public void drop(){
        row++;
    }

    public void moveRight(){
        col++;
    }

    public void moveLeft(){
        col--;
    }

    @Override//重写
    public String toString() {
        return row+","+col;
    }

    }
```

Tetromino类的完整代码如下所示：

```
package com.tarena.tetris;

import java.util.Arrays;
import java.util.Random;

/**
 * 4格方块
 */
public abstract class Tetromino {
 protected Cell[] cells = new Cell[4];

 /** 旋转状态 */
 protected State[] states;
```

```
    /** 旋转状态的序号 states[index%states.length] */
    protected int index = 10000;

    /** 内部类 */
    protected class State {
        int row0, col0, row1, col1, row2, col2, row3, col3;

        public State(int row0, int col0, int row1, int col1, int row2,
                int col2, int row3, int col3) {
            this.row0 = row0;
            this.col0 = col0;
            this.row1 = row1;
            this.col1 = col1;
            this.row2 = row2;
            this.col2 = col2;
            this.row3 = row3;
            this.col3 = col3;
        }
    }

    /** 向右转 */
    public void rotateRight() {
        // 取得变换的下个数据状态 states[n]
        // 取得当前轴的 row, col
        // 旋转以后的数据=(row,col) + states[n]
        index++;
        State s = states[index % states.length];
        Cell o = cells[0];
        int row = o.getRow();
        int col = o.getCol();
        cells[1].setRow(row + s.row1);
        cells[1].setCol(col + s.col1);
        cells[2].setRow(row + s.row2);
        cells[2].setCol(col + s.col2);
        cells[3].setRow(row + s.row3);
        cells[3].setCol(col + s.col3);
    }

    public void rotateLeft() {
        index--;
        State s = states[index % states.length];
        Cell o = cells[0];
        int row = o.getRow();
        int col = o.getCol();
        cells[1].setRow(row + s.row1);
        cells[1].setCol(col + s.col1);
        cells[2].setRow(row + s.row2);
        cells[2].setCol(col + s.col2);
        cells[3].setRow(row + s.row3);
        cells[3].setCol(col + s.col3);
    }

    /** 工厂方法, 随机产生 4 格方块 */
    public static Tetromino randomOne() {
        Random random = new Random();
        int type = random.nextInt(7);
        switch (type) {
        case 0:
            return new O();
        case 1:
            return new S();
        case 2:
            return new Z();
        case 3:
            return new J();
        case 4:
            return new L();
```

```
        case 5:
            return new I();
        case 6:
            return new T();
        }
        return null;
    }

    /**
     * 方块下落一个格子
     */
    public void softDrop() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].drop();
        }
    }

    /**
     * 方块左移一个格子
     */
    public void moveLeft() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].moveLeft();
        }
    }

    /**
     * 方块右移一个格子
     */
    public void moveRight() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].moveRight();
        }
    }

    /**
     * 覆盖 toString 方法，显示方块中每个格子的行列信息
     */
    public String toString() {
        return Arrays.toString(cells);
    }
}


/**
 * T 形方块
 */
class T extends Tetromino {
    public T() {
        cells[0] = new Cell(0, 4, Tetris.T);
        cells[1] = new Cell(0, 3, Tetris.T);
        cells[2] = new Cell(0, 5, Tetris.T);
        cells[3] = new Cell(1, 4, Tetris.T);

        states = new State[4];
        states[0] = new State(0, 0, 0, -1, 0, 1, 1, 0);
        states[1] = new State(0, 0, -1, 0, 1, 0, 0, -1);
        states[2] = new State(0, 0, 0, 1, 0, -1, -1, 0);
        states[3] = new State(0, 0, 1, 0, -1, 0, 0, 1);
    }
}

/**
 * I 形方块
 */
class I extends Tetromino {
```

```
    public I() {
        cells[0] = new Cell(0, 4, Tetris.I);
        cells[1] = new Cell(0, 3, Tetris.I);
        cells[2] = new Cell(0, 5, Tetris.I);
        cells[3] = new Cell(0, 6, Tetris.I);

        states = new State[] { new State(0, 0, 0, -1, 0, 1, 0, 2),
                new State(0, 0, -1, 0, 1, 0, 2, 0) };
    }
    }

    /**
     * S 形方块
     */
    class S extends Tetromino {
    public S() {
        cells[0] = new Cell(1, 4, Tetris.S);
        cells[1] = new Cell(1, 3, Tetris.S);
        cells[2] = new Cell(0, 4, Tetris.S);
        cells[3] = new Cell(0, 5, Tetris.S);
        states = new State[] { new State(0, 0, 0, -1, -1, 0, -1, 1),
                new State(0, 0, -1, 0, 0, 1, 1, 1) };
    }
    }

    /**
     * Z 形方块
     */
    class Z extends Tetromino {
    public Z() {
        cells[0] = new Cell(1, 4, Tetris.Z);
        cells[1] = new Cell(0, 3, Tetris.Z);
        cells[2] = new Cell(0, 4, Tetris.Z);
        cells[3] = new Cell(1, 5, Tetris.Z);

        states = new State[] { new State(0, 0, -1, -1, -1, 0, 0, 1),
                new State(0, 0, -1, 1, 0, 1, 1, 0) };
    }
    }

    /**
     * O 形方块
     */
    class O extends Tetromino {
    public O() {
        cells[0] = new Cell(0, 4, Tetris.O);
        cells[1] = new Cell(0, 5, Tetris.O);
        cells[2] = new Cell(1, 4, Tetris.O);
        cells[3] = new Cell(1, 5, Tetris.O);

        states = new State[] { new State(0, 0, 0, 1, 1, 0, 1, 1),
                new State(0, 0, 0, 1, 1, 0, 1, 1) };
    }
    }

    /**
     * L 形方块
     */
    class L extends Tetromino {
    public L() {
        cells[0] = new Cell(0, 4, Tetris.L);
        cells[1] = new Cell(0, 3, Tetris.L);
        cells[2] = new Cell(0, 5, Tetris.L);
        cells[3] = new Cell(1, 3, Tetris.L);
```

```
        states = new State[] { new State(0, 0, 0, 1, 0, -1, -1, 1),
                new State(0, 0, 1, 0, -1, 0, 1, 1),
                new State(0, 0, 0, -1, 0, 1, 1, -1),
                new State(0, 0, -1, 0, 1, 0, -1, -1) };
    }
}

/**
 * J形方块
 */
class J extends Tetromino {
    public J() {
        cells[0] = new Cell(0, 4, Tetris.J);
        cells[1] = new Cell(0, 3, Tetris.J);
        cells[2] = new Cell(0, 5, Tetris.J);
        cells[3] = new Cell(1, 5, Tetris.J);

        states = new State[] { new State(0, 0, 0, -1, 0, 1, 1, 1),
                new State(0, 0, -1, 0, 1, 0, 1, -1),
                new State(0, 0, 0, 1, 0, -1, -1, -1),
                new State(0, 0, 1, 0, -1, 0, -1, 1) };
    }
}
```

Tetris类的完整代码如下所示：

```
package com.tarena.tetris;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.util.Arrays;
import java.util.Timer;
import java.util.TimerTask;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * 俄罗斯方块
 */
public class Tetris extends JPanel {
    /** 游戏的当前状态: RUNNING PAUSE GAME_OVER */
    private int state;
    public static final int RUNNING = 0;
    public static final int PAUSE = 1;
    public static final int GAME_OVER = 2;

    private int score;// 分数
    private int lines;// 销毁的行数
    private Cell[][] wall;// 背景墙
    private Tetromino tetromino;// 正在下落的四格方块
    private Tetromino nextOne;// 下一个四格方块
    /** 背景图片 */
    private static BufferedImage background;
    private static BufferedImage gameOver;
    private static BufferedImage pause;
    public static BufferedImage T;
    public static BufferedImage S;
```

```
    public static BufferedImage I;
    public static BufferedImage L;
    public static BufferedImage J;
    public static BufferedImage O;
    public static BufferedImage Z;
    public static final int ROWS = 20;// 背景墙的行数
    public static final int COLS = 10;// 背景墙的列数

    /** 在 Tetris 类中增加定时器 */
    private Timer timer;

    /** 速度 */
    private int speed;
    /** 难度级别 */
    private int level;
    /** 下落计数器 当 index%speed==0 时候下落一次 */
    private int index;

    // 将图片素材, 复制到 com.tarena.tetris 包中.
    /** 使用静态代码块加载静态的图片 */
    static {
        try {
            // Tetris.class 的同一个包中找 "tetris.png"
            background = ImageIO.read(Tetris.class.getResource("tetris.png"));
            gameOver                                                          =
ImageIO.read(Tetris.class.getResource("game-over.png"));
            pause = ImageIO.read(Tetris.class.getResource("pause.png"));
            T = ImageIO.read(Tetris.class.getResource("T.png"));
            I = ImageIO.read(Tetris.class.getResource("I.png"));
            S = ImageIO.read(Tetris.class.getResource("S.png"));
            Z = ImageIO.read(Tetris.class.getResource("Z.png"));
            J = ImageIO.read(Tetris.class.getResource("J.png"));
            L = ImageIO.read(Tetris.class.getResource("L.png"));
            O = ImageIO.read(Tetris.class.getResource("O.png"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * JPanel paint() paint 画 重写 paint() 修改原有的绘制方法
     **/
    @Override
    public void paint(Graphics g) {
        // 画背景, 画墙, 画正在下落的方块 画下一个方块...
        g.drawImage(background, 0, 0, null);
        g.translate(15, 15);// 坐标系平移
        paintWall(g);// 画墙
        paintTetromino(g);// 绘制正在下落的方块
        paintNextOne(g);//绘制下一个要下落的方块
        paintScore(g);//绘制分数
        paintState(g);// 绘制游戏的状态
    }

    private void paintState(Graphics g) {
        switch (state) {
        case PAUSE:
            g.drawImage(pause, -15, -15, null);
            break;
        case GAME_OVER:
            g.drawImage(gameOver, -15, -15, null);
            break;
        }
    }
```

```
/** 在 Tetris 添加启动方法 action() */
public void action() {
    wall = new Cell[ROWS][COLS];
    //wall[2][2] = new Cell(2, 2, T);
    tetromino = Tetromino.randomOne();
    nextOne = Tetromino.randomOne();
    state = RUNNING;
    // 处理键盘按下事件, 在按下按键时候执行下落方法
    KeyAdapter l = new KeyAdapter() {
        @Override
        // key 按键 Pressed 按下了
        public void keyPressed(KeyEvent e) {
            int key = e.getKeyCode();
            switch (state) {
            case GAME_OVER:
                processGameoverKey(key);
                break;
            case PAUSE:
                processPauseKey(key);
                break;
            case RUNNING:
                processRunningKey(key);
            }
            repaint();
        }
    };
    // 下落流程：监听键盘事件->如果下箭头按下->
    // 执行下落算法 tetromino.softDrop()->
    // 修改每个格子对象的数据->调用 repaint()->
    // 尽快调用 paint()->paint 方法会根据当前的数据
    // 重新绘制界面 -> 看到移动以后的方块了

    // 绑定事件到当前面板
    this.requestFocus();
    this.addKeyListener(l);

    // 在 Action 方法中添加,定时计划任务
    timer = new Timer();
    timer.schedule(new TimerTask() {
        public void run() {
            speed = 40 - (lines / 100);
            speed = speed < 1 ? 1 : speed;
            level = 41 - speed;
            if (state == RUNNING && index % speed == 0) {
                softDropAction();
            }
            index++;
            repaint();
        }
    }, 10, 10);
}
private void processPauseKey(int key) {
    switch (key) {
    case KeyEvent.VK_Q:
        System.exit(0);
        break;
    case KeyEvent.VK C:
        index = 0;
        state = RUNNING;
        break;
    }
}

protected void processRunningKey(int key) {
    switch (key) {
    case KeyEvent.VK_Q:
```

```
            System.exit(0);
            break;
        case KeyEvent.VK_RIGHT:
            Tetris.this.moveRightAction();
            break;
        case KeyEvent.VK LEFT:
            Tetris.this.moveLeftAction();
            break;
        case KeyEvent.VK_DOWN:
            softDropAction();
            break;
        case KeyEvent.VK SPACE:
            hardDropAction();
            break;
        case KeyEvent.VK UP:
            rotateRightAction();
            break;
        case KeyEvent.VK_P:
            state = PAUSE;
            break;
        }
    }

    protected void processGameoverKey(int key) {
        switch (key) {
        case KeyEvent.VK Q:
            System.exit(0);
            break;
        case KeyEvent.VK_S:
            /** 游戏重新开始 */
            this.lines = 0;
            this.score = 0;
            this.wall = new Cell[ROWS][COLS];
            this.tetromino = Tetromino.randomOne();
            this.nextOne = Tetromino.randomOne();
            this.state = RUNNING;
            this.index = 0;
            break;
        }
    }

    public static final int FONT_COLOR = 0x667799;
    public static final int FONT_SIZE = 30;
       /**
        * 绘制分数
        * @param g
        */
    private void paintScore(Graphics g) {
        int x = 290;
        int y = 160;
        g.setColor(new Color(FONT_COLOR));
        Font font = g.getFont();// 取得 g 当前字体
        font = new Font(font.getName(), font.getStyle(), FONT_SIZE);
        g.setFont(font);// 更改了 g 的字体
        String str = "SCORE:" + score;
        g.drawString(str, x, y);
        y += 56;
        str = "LINES:" + lines;
        g.drawString(str, x, y);

        y += 56;
        g.drawString("LEVEL:" + level, x, y);

    }


    /**
```

```
     * 绘制下一个要下落的方块
     *
     * @param g
     */
    private void paintNextOne(Graphics g) {
        if (nextOne == null) {
            return;
        }
        // 将每个格子的 row,col 换算为 x,y 然后贴图
        Cell[] cells = nextOne.cells;
        for (int i = 0; i < cells.length; i++) {
            // i = 0 1 2 3
            Cell cell = cells[i];
            // cell 每个格子
            int x = (cell.getCol() + 10) * CELL_SIZE;
            int y = (cell.getRow() + 1) * CELL_SIZE;
            g.drawImage(cell.getImage(), x - 1, y - 1, null);
        }
    }

    /**
     * 绘制正在下落的方块
     *
     * @param g
     */
    public void paintTetromino(Graphics g) {
        if (tetromino == null) {
            return;
        }
        // 将每个格子的 row,col 换算为 x,y 然后贴图
        Cell[] cells = tetromino.cells;
        for (int i = 0; i < cells.length; i++) {
            // i = 0 1 2 3
            Cell cell = cells[i];
            // cell 每个格子
            int x = cell.getCol() * CELL_SIZE;
            int y = cell.getRow() * CELL_SIZE;
            g.drawImage(cell.getImage(), x - 1, y - 1, null);
        }
    }

    public static final int CELL_SIZE = 26;

    /** 画墙 */
    private void paintWall(Graphics g) {
        for (int row = 0; row < wall.length; row++) {
            Cell[] line = wall[row];
            // line 代表墙上的每一行
            for (int col = 0; col < line.length; col++) {
                Cell cell = line[col];
                // cell 代表墙上的每个格子
                int x = col * CELL_SIZE;
                int y = row * CELL_SIZE;
                if (cell == null) {
                    g.drawRect(x, y, CELL SIZE, CELL SIZE);
                } else {
                    g.drawImage(cell.getImage(), x - 1, y - 1, null);
                }
                // g.drawString(row+","+col, x,y+CELL_SIZE);
            }
        }
    }

    /** 检查当前正在下落的方块是否出界了 */
    private boolean outOfBounds() {
        Cell[] cells = tetromino.cells;
```

```
        for (int i = 0; i < cells.length; i++) {
            Cell cell = cells[i];
            int col = cell.getCol();
            if (col < 0 || col >= COLS) {
                return true;
            }
        }
        return false;
    }

    /** 检查正在下落的方块是否与墙上的砖块重叠 */
    private boolean coincide() {
        Cell[] cells = tetromino.cells;
        for (int i = 0; i < cells.length; i++) {
            Cell cell = cells[i];
            int row = cell.getRow();
            int col = cell.getCol();
            // 如果墙的 row,col 位置上有格子,就重叠了!
            if (row >= 0 && row < ROWS && col >= 0 && col <= COLS
                    && wall[row][col] != null) {
                return true;// 重叠
            }
        }
        return false;
    }

    /** 在 Tetris 类上添加方法, 向右移动的流程控制 */
    public void moveRightAction() {
        // 尝试先向右移动, 如果发现超出了边界, 就
        // 向左移动, 修正回来.
        tetromino.moveRight();// coincide 重叠
        if (outOfBounds() || coincide()) {
            tetromino.moveLeft();
        }
    }

    public void moveLeftAction() {
        tetromino.moveLeft();
        if (outOfBounds() || coincide()) {
            tetromino.moveRight();
        }
    }

    /** 下落流程控制 */
    public void softDropAction() {
        if (canDrop()) {
            tetromino.softDrop();
        } else {
            landIntoWall();
            destoryLines();
            if (isGameOver()) {
                state = GAME_OVER;
            } else {
                tetromino = nextOne;
                nextOne = Tetromino.randomOne();
            }
        }
    }

    private static int[] scoreTable = { 0, 1, 10, 50, 100 };

    // 0 1 2 3 4
    private void destoryLines() {
        int lines = 0;
        for (int row = 0; row < wall.length; row++) {
            if (fullCells(row)) {
                deleteRow(row);
```

```
                lines++;
            }
        }
        this.score += scoreTable[lines];
        this.lines += lines;
    }

    private void deleteRow(int row) {
        for (int i = row; i >= 1; i--) {
            System.arraycopy(wall[i - 1], 0, wall[i], 0, COLS);
        }
        Arrays.fill(wall[0], null);
    }

    /**
     * 检查当前行的每个格子,是否是满的，如果满了则返回 true，否则返回 false
     */
    private boolean fullCells(int row) {
        Cell[] line = wall[row];
        for (Cell cell : line) {
            if (cell == null) {// 如果有 null 返回 false 否则返回 true
                return false;
            }
        }
        return true;
    }

    private void landIntoWall() {
        Cell[] cells = tetromino.cells;
        for (int i = 0; i < cells.length; i++) {
            Cell cell = cells[i];
            int row = cell.getRow();
            int col = cell.getCol();
            wall[row][col] = cell;
        }
    }

    /** 检查当前的方块是否能够下落，返回 true 能够下落 */
    private boolean canDrop() {
        Cell[] cells = tetromino.cells;
        for (int i = 0; i < cells.length; i++) {
            Cell cell = cells[i];
            int row = cell.getRow();
            if (row == ROWS - 1) {
                return false;
            }
        }
        for (Cell cell : cells) {// Java 5 以后可以使用
            int row = cell.getRow() + 1;
            int col = cell.getCol();
            if (row >= 0 && row < ROWS && col >= 0 && col <= COLS
                    && wall[row][col] != null) {
                return false;
            }
        }
        return true;
    }

    /**
     * 硬下落流程，下落到不能下落为止 绑定到 空格(VK_SPACE)事件上
     * */
    public void hardDropAction() {
        while (canDrop()) {
            tetromino.softDrop();
        }
        landIntoWall();
        destoryLines();
```

```
        if (isGameOver()) {
            state = GAME_OVER;
        } else {
            tetromino = nextOne;
            nextOne = Tetromino.randomOne();
        }
    }

    /** 在 Tetris 类中添加 旋转流程控制方法 */
    public void rotateRightAction() {
        tetromino.rotateRight();
        if (outOfBounds() || coincide()) {
            tetromino.rotateLeft();
        }
    }

    /** 检查游戏是否结束 */
    private boolean isGameOver() {
        // 如果下一个方块没有出场位置了，则游戏结束
        // 就是：下一个出场方块每个格子行列对应的
        // 墙上位置如果有格子，就游戏结束
        Cell[] cells = nextOne.cells;
        for (Cell cell : cells) {
            int row = cell.getRow();
            int col = cell.getCol();
            if (wall[row][col] != null) {
                return true;
            }
        }
        return false;
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        // 在加载 Tetris 类的时候，会执行静态代码块
        // 静态代码块,装载了图片素材，为图片对象
        Tetris tetris = new Tetris();
        // 将面板的颜色设置为蓝色，用于测试
        tetris.setBackground(new Color(0x0000ff));
        frame.add(tetris);
        frame.setSize(530, 580);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);// 在显示窗口时候,会"尽快"的调用 paint()方法绘制界面
        tetris.action();
    }
}
```

数据挖掘系统（DMS）

1. 问题

一．DMS 项目中用到的术语定义如下：

1．DMS：数据挖掘系统（Data Mining System）。

2．用户：需要使用 Unix 实验室出租业务的客户，用户通过电信公司提供的账号和密码登录指定的 Unix 机器，并使用 Unix 服务。

3．OS 账号：用户登录所使用的账号，即 OS 账号，也是 Unix 服务器的系统区分登录用户的唯一标识。登录日志文件中记载的也是 OS 账号及其登录信息。

4．终端机器： 用户通过一台本地终端机器来访问我们的 Unix 机器，并使用 Unix 服务，该本地终端就是我们所说的终端机器。在我们的程序中要记录这一终端的 IP，作为向 OS 账号所代表的用户收费的依据之一。

5．登录日志文件：在 Unix 机器中自动记录所有 OS 账号登入/登出日志的文件，该文件为：/var/adm/wtmpx。DMS 程序需要提取该文件的数据并匹配出某 OS 账号登入/登出的记录，以作为将来的收费依据。

6．用户登入记录：登录日志文件中记录的该用户的 OS 账号登入 Unix 系统的日志记录，该日志记录包含登录的 OS 账号、进程 ID、登入 Unix 的时刻、登录的终端机器 IP 等信息。

7．用户登出记录：登录日志文件中记录的该用户的 OS 账号登出 Unix 系统的日志记录，该日志记录包含登录的 OS 账号、进程 ID、登出 Unix 的时刻，登录的终端机器 IP 等信息。

8．登录会话记录：用户使用 OS 账号登入和登出的记录是分开的两条记录，这两条记录的 OS 账号和进程 ID 相同。需要将登入记录和登出记录按照对应关系匹配起来，形成一条完整的登录信息，并计算出该账号本次登录的时长。登录记录中包含以下信息：登录的 OS 账号、进程 ID、登入时刻、登出时刻、登录时长和登录终端机器 IP。

9．DMS 采集端：DMS 采集端作为 T-DMS 项目的一个组成部分，其主要任务是采集、解析、匹配登录数据，并将匹配成功的数据发送到 DMS 的服务器端，同时将匹配不成功的数据保存为未匹配的日志文件。

10．DMS 服务器端：DMS 服务器端用于接收 DMS 采集端发送来的日志数据，保存为文

本文件的同时，提供数据的浏览监控功能。

11. 归档日志文件：服务器端的文本文件，文件名为 server.txt，用于记录成功发送到服务器端的登录会话数据，以备于数据归档保存。

二．DMS 项目背景如下：

在电信的业务中，有一种 Unix 实验室出租业务。只要用户向电信运营商申请一个 Unix 账号（OS 账号），就可以远程登录 Unix 实验室，以使用 Unix 系统。任何用户登录电信运营商提供的 Unix 实验室的 Unix 系统时， Unix 系统都会记录该 OS 账号的登入和登出信息，这些信息都保存在 Unix 的系统日志文件中。

用户使用电信运营商提供的 Unix 实验室的服务需要缴纳一定的费用。因此，电信运营商需要一套系统，将用户登录实验室的时间长度数据采集起来，以作为对用户的收费依据。

数据采集程序采集到登录数据之后，首先需要对数据进行解析和匹配等处理，然后需要将数据发送到服务器端保存。服务器端得到数据之后，需要保存数据，同时提供数据的浏览等功能。

整个系统的结构组成如图 - 1 所示：

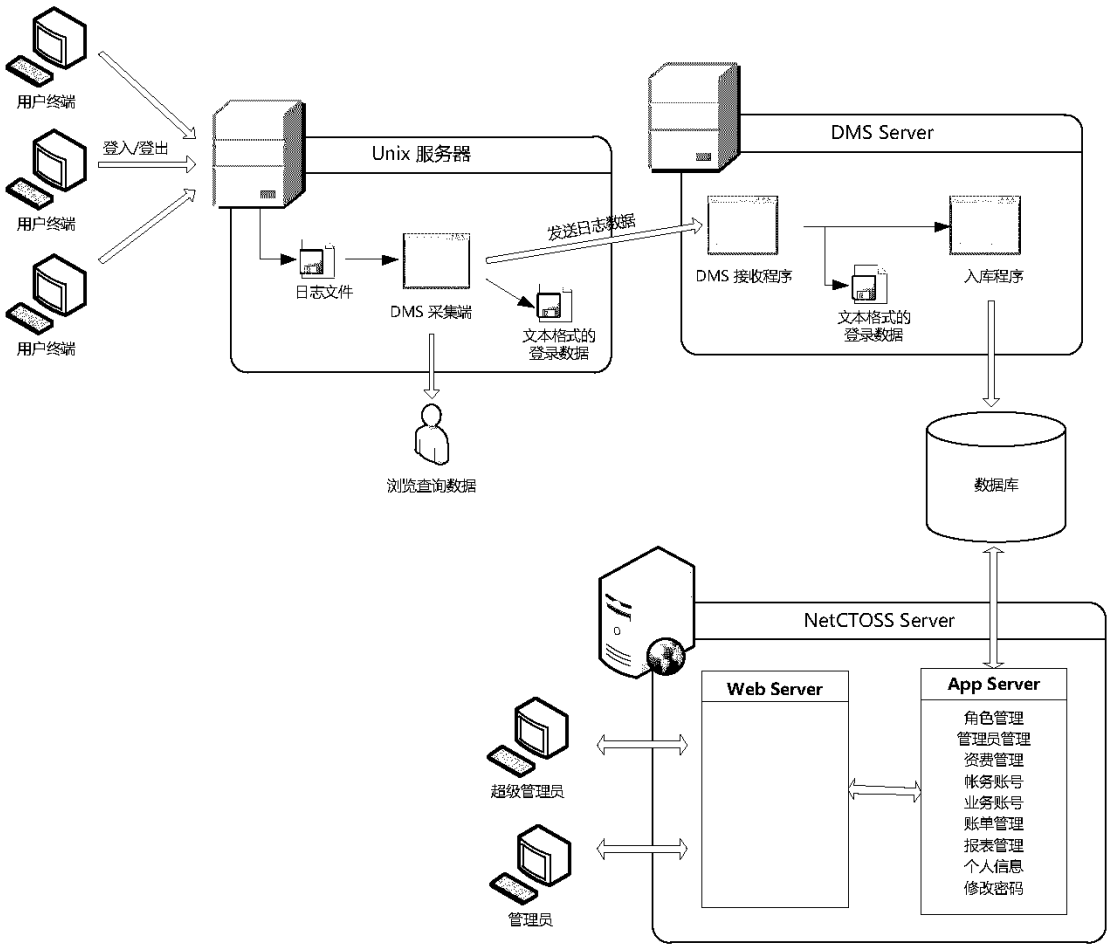

图 - 1

由图 - 1 可以看出，DMS 系统是一个基于“客户端-服务器”架构的数据挖掘系统。该系统由 DMS 采集端和 DMS 服务器端两大部分组成。

DMS 采集端的主要任务是采集、解析、匹配和保存并发送登录数据。DMS 采集端程序通过读取 Unix 系统日志文件中记录的所有 OS 账号的登入/登出的原始记录，解析找出成对的登入/登出记录，从而得到 OS 账号登录实验室的登入时刻、登出时刻、登录时间长度等数据。为了便于查询和作为电信收费依据，DMS 采集端程序需要将解析并匹配处理好的数据以可阅读的文本形式发送到 DMS 服务器端保存，对于没有匹配成功的数据则需要保存为采集端的文本文件作为数据备份。

DMS 服务器端通过接收程序接收到日志数据后，首先需要将日志数据备份保存为文件数据，并需要提供 UI 用户界面实现数据的浏览和监控等功能。另外，DMS 服务器还可以通过入库程序将日志数据存入数据库。

日志数据存入数据库后，可以通过 NetCTOSS 系统来实现查询和管理功能。管理员登录 NetCTOSS 系统后，可以进行用户的账号管理、账单管理以及报表数据查询等操作。

三．DMS 项目的业务描述如下：

DMS 项目作为 T-DMS 整个项目的一个组成部分，是一个基于“客户端-服务器”架构的数据挖掘系统，由采集端和服务器端两部分组成。DMS 系统的详细结构组成如图 - 2 所示：

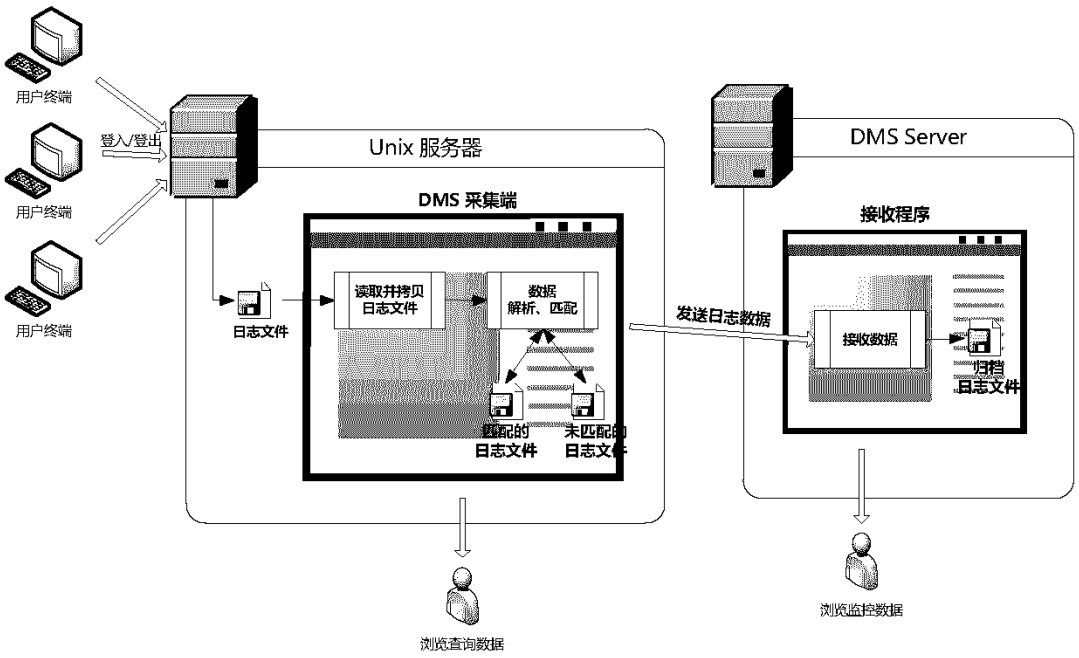

图 - 2

由图 - 2 可以看出，DMS 的采集端程序将定时读取日志文件，然后拷贝本次读取成功的数据存为文件，并进行日志数据的解析和匹配。对于匹配成功的数据，系统需要数据发送到服务器端。对于匹配不成功的数据 则也需要保存为采集端的文本文件(未匹配的日志文件)，用于数据的备份，留待与下次提取解析后的日志数据合并后再进行匹配等操作。另外，DMS 采集端程序还需要提供 UI 界面供用户实现数据的浏览和查询。

DMS 的服务器端的接收程序启动后，将实时接收 DMS 采集端发来的日志数据，并保存为服务器端的文本文件（归档日志文件），同时还提供 UI 界面以实现数据的监控功能。

四．DMS 采集端功能描述如下：

1．定期读取日志文件，得到最新的日志数据（从上一次读取后的位置开始读取）；

2．解析其中的原始记录，提取其中的必需信息；

3．实现数据匹配；

4．将匹配成功的数据提交到服务器；

5．未匹配成功的数据，留作下次解析匹配操作。

DMS 采集端进行读取、解析匹配以及发送操作后，均需要在界面上显示当前数据操作的结果，界面交互效果如图 - 3 所示：

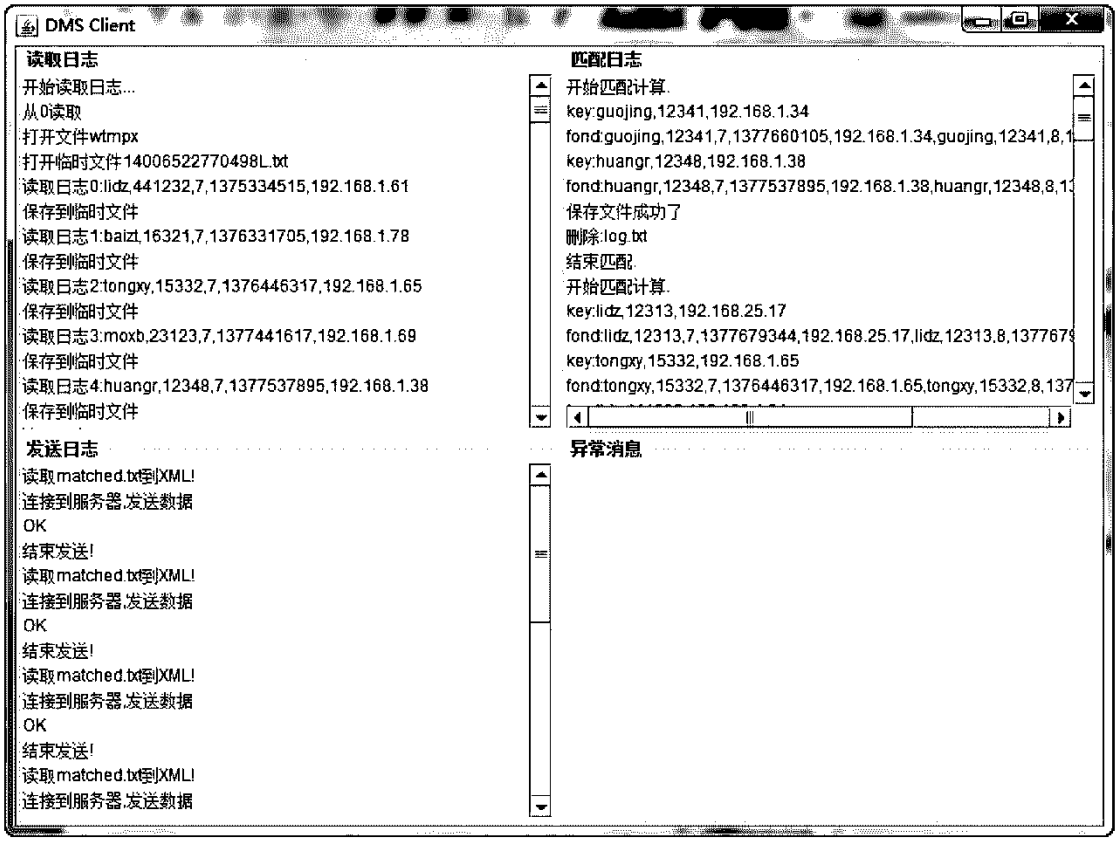

图 - 3

图 - 3 所示界面为 DMS 系统采集端的 UI 界面。程序运行后，将在界面上显示每次数据采集操作中所采集的读取日志、匹配日志、发送日志以及异常信息，以实现数据的显示和监控。

五．DMS 项目服务器端功能描述如下：

1．持续接收采集端发来的登录数据，并将数据保存到归档日志文件 server.txt 中；

2．显示数据接收以及操作的结果，以实现数据监控；

DMS 项目服务器端的系统功能如图 - 4 所示：

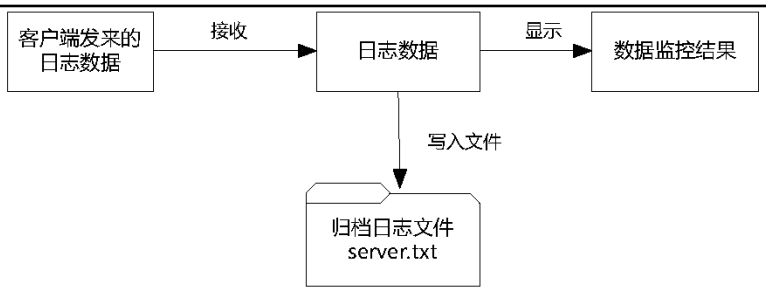

图 - 4

由图 - 4 可以看出，服务器端程序运行后，将持续接收采集端发来的日志数据，并将日志数据保存到归档日志文件 server.txt 中，同时在界面上显示当前数据操作的结果。

服务器端界面所显示的信息分为四类：网络连接日志、客户服务日志、保存日志和异常消息，以实现数据的监控。其界面效果如图 - 5 所示：

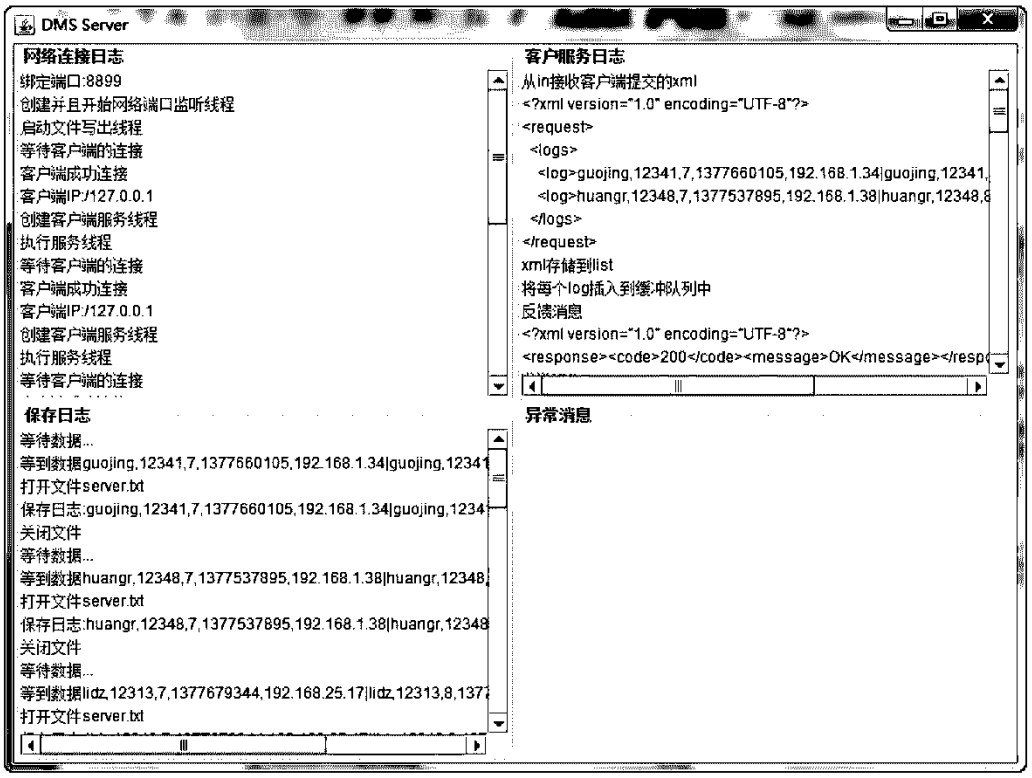

图 - 5

六：DMS 项目的数据描述如下：

用户登录 Unix 系统的日志记录文件在 Unix 服务器的/var/adm 目录下，文件名是 wtmpx。日志记录文件是一个二进制文件，该文件中包含多条登录记录，每条登录记录包含 372 字节。wtmpx 日志文件的数据内容形如图 - 6 所示：

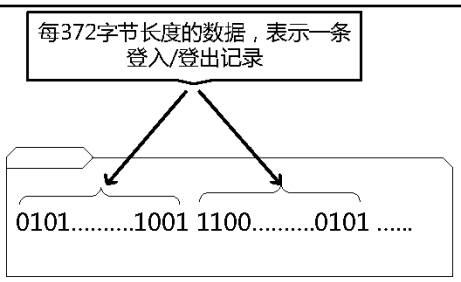

图 - 6

由图 - 6 可以看出，日志文件中所包含的登录记录总数为：日志文件大小/372。

每条登录记录包含 372 字节的数据，每条记录（每 372 字节）中所包含的数据格式如表-1 所示：

表 - 1 登录数据的格式说明

位置范围	字节长度	含义
000-031	32	/* UID，OS 账号，即用户登录用的用户名*/
032-035	4	/* inittab id */
036-067	32	/* device name (console， lnxx) */
068-071	4	/* 进程 ID*/
072-073	2	/* 登录类型 7-登入，8 登出 */
074-075	2	/* process termination*/
076-077	2	/* exit status*/
	2	/* 这是 C 数据类型补齐产生的空位*/
080-083	4	/* 登录时刻*/
084-087	4	/* and microseconds */
088-091	4	/* session ID， used for windowing */
092-111	20	/* reserved for future use */
112-113	2	/* significant length of ut_host */
114-371	258	/* 登录终端 IP*/

由表 - 1 可见，数据采集系统需要从日志文件 wtmpx 中，按照表 - 1 所述的格式对用户的登录记录数据进行采集。每条登录记录中必须采集如下信息：

- 用户登录的 OS 账号
- 用户登录的进程 ID
- 用户登录/登出的时间

- 用户登录期间的在线时间
- 用户的终端 IP

这些需要采集的数据信息在 372 字节长度的数据中的位置如表 - 2 所示：

表 - 2 需要采集的数据格式说明

位置范围	字节长度	数据含义
000-031	32	用户名，匹配同一次登录会话的必须数据之一，为文本类型数据（String）
068-071	4	进程 ID，匹配同一次登录会话的必须数据之二，为整数类型数据（int）
072-073	2	登录类型，值在 1-8 之间，但只处理 7 与 8 两种情况：7 为登入，8 为登出；为整数类型数据（short）
080-083	4	登入或者登出时刻，为整数类型数据（int）
114-371	258	用户登录的终端 IP，为文本类型数据（String）

对于每条登录记录，只需要采集表 - 2 中所列出的位置上的数据即可。

在每 372 字节长度的数据中，需要采集的数据的位置和数据内容形如图 - 7 所示：

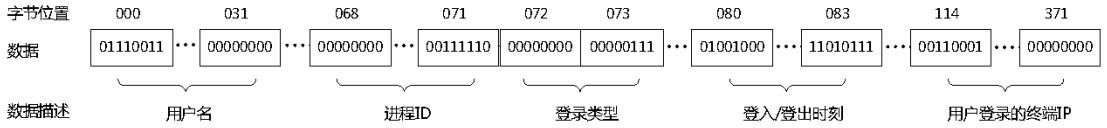

图 - 7

对于每条登录数据，需要根据登录类型来判断操作类型。登录类型数据的取值为 1 - 8，DMS 系统只需要处理登录类型的值为 7 与 8 两种情况。其中，7 为登入，8 为登出，其他操作类型的日志数据不用考虑。

日志文件 wtmpx 中的原始记录是按照时间的先后排序好的数据，如果依次提取 wtmpx 文件中每 372 字节长度的数据，并采集表 - 2 和图 - 7 中所示位置的必需数据，判断登录类型为 7 或者 8 之后，加以解析以形成可阅读的数据，所形成的记录形如表 - 3 所示：

表 - 3 采集并解析后的数据格式示例

用户名	进程 ID	登录类型	登入/登出时刻	用户登录的终端 IP
as080705	45123	7	2012-12-19 16:53:20	192.168.1.48
as080713	12351	8	2012-12-19 16:53:21	192.168.25.51
b090150	12356	7	2012-12-19 16:53:26	192.168.25.56
as080705	45123	8	2012-12-19 16:53:35	192.168.1.48
b090150	12356	8	2012-12-19 16:53:37	192.168.25.56
as080706	12472	7	2012-12-19 16:53:37	192.168.25.75

用户登录时，需要使用 OS 账号进行登录，每次登录会产生一个进程 ID。同一个用户

在登出后可重新登录，但是两次登录所产生的进程 ID 不会相同。如果两条记录的用户登录名和进程 ID 相同，则为同一次登录会话。对于同一次会话的两条登录数据，需要根据登录类型来判断操作类型：7 为登入，8 为登出，其他类型的操作不用考虑。

DMS 采集端程序每次从登录日志文件中所提取到的数据，有成对匹配的登入记录和登出记录，也会有单条的登入或者登出记录。

由表 - 3 中的数据可以看出，此表的 6 条示例数据中，有两对匹配的登录会话记录，共涉及到 2 个OS 账号(as080705和b090150)；另外，还有一条登入记录(as080706)，即截至此次提取数据时，尚未有登出的记录；还有一条登出记录（as080713），表示该 OS 账号在上次提取数据的时间段登入，却在本次提取数据的时间段登出。

因为 DMS 采集端程序所提取到的登录日志数据中，会有单条的登入或者登出的数据，因此，在进行数据匹配之前，需要先将本次所提取到的日志数据与未匹配的日志文件中的登录数据进行合并。

未匹配的日志文件是指上次操作后所遗留下来的数据，只包含登入数据，用于与后续的数据进行合并。上次匹配操作后剩余的历史数据形如表 - 4 所示：

表 - 4 历史数据示例

用户名	进程 ID	登录类型	登入/登出时刻	用户登录的终端 IP
as080713	12351	7	2012-12-19 16:53:01	192.168.25.51

由此可见，需要将表 - 3 中的数据（本次提取的数据）和表 - 4 中的数据（历史数据）合并起来再进行后续操作。

将表 - 3 和表 - 4 中的示例数据合并，合并后的数据形如表 - 5 所示：

表 - 5 合并历史数据后的数据示例

用户名	进程 ID	登录类型	登入/登出时刻	用户登录的终端 IP
as080713	12351	7	2012-12-19 16:53:01	192.168.25.51
as080705	45123	7	2012-12-19 16:53:20	192.168.1.48
as080713	12351	8	2012-12-19 16:53:21	192.168.25.51
b090150	12356	7	2012-12-19 16:53:26	192.168.25.56
as080705	45123	8	2012-12-19 16:53:35	192.168.1.48
b090150	12356	8	2012-12-19 16:53:37	192.168.25.56
as080706	12472	7	2012-12-19 16:53:37	192.168.25.75

注：表 - 5 中的第一行（阴影行）为历史数据。

对于表 - 5 中的合并后的数据按照登入/登出进行匹配，匹配后的结果形如表 - 6 所示：

表 - 6 匹配后的数据示例

用户名	进程 ID	登入时间	登出时间	登录时长（秒）	终端 IP

as080713	12351	2012-12-19 16:53:01	2012-12-19 16:53:21	20	192.168.25.51
as080705	45123	2012-12-19 16:53:20	2012-12-19 16:53:35	15	192.168.1.48
b090150	12356	2012-12-19 16:53:26	2012-12-19 16:53:37	11	192.168.25.56
as080706	12472	2012-12-19 16:53:37	无	无	192.168.25.75

由表 - 6 可见，对于合并后的登录数据进行匹配时，对于可以成功匹配的登入和登出信息，直接形成一次完整的登录会话的记录；对于只有登出的记录，则需要在历史记录中查找并匹配其登入记录，并形成一次完整的登录会话的记录；对于只有登入的记录，表示用户依然在线，则记载为历史登入数据，留待下次提取、解析并合并后进行匹配。

成功发送到 DMS 服务器端的日志数据，由服务器端程序保存为归档日志文件 server.txt。该文件的内容格式如图 - 8 所示：

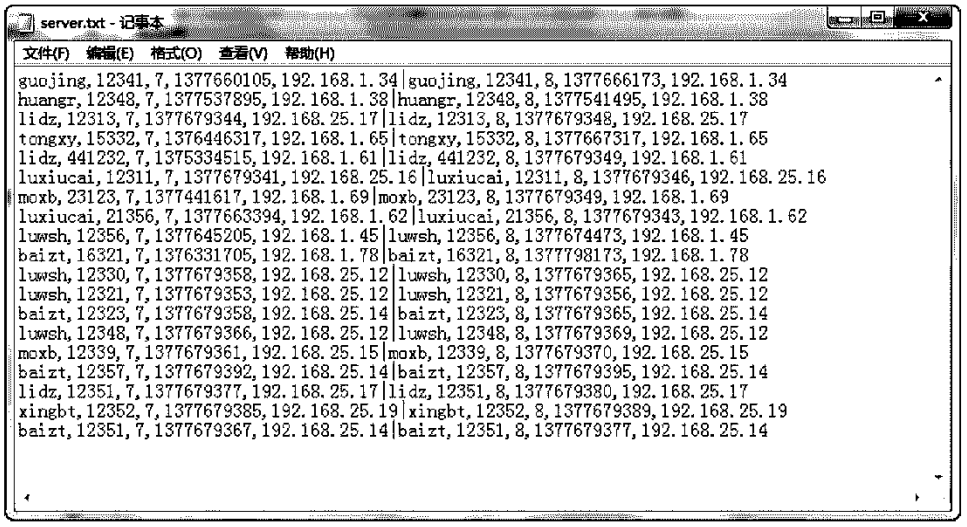

图 - 8

2. 方案

一．DMS 详细功能分析如下：

DMS 项目是一个基于“客户端-服务器”架构的数据挖掘系统， 由 DMS 采集端和 DMS 服务器端两部分组成。

DMS 项目采集端的主要功能为：

1. 定时读取日志文件 wtmpx（从上一次读取后的位置开始读取），并解析其中的原始记录，提取其中的必需信息存储到文件 log.txt 中；
2. 将 log.txt 文件中的记录匹配为完整的登录会话记录(一次登录会话包含一次登入记录和一次登出记录) 并存储到 match.txt 中；如果有未匹配的日志记录，则将这些记录存储到 login.txt 文件中。
3. 匹配时，如果存在未匹配的日志文件 login.txt，则将解析后的登录日志文件 (log.txt)中的数据和未匹配的日志文件（login.txt）中的数据进行合并；
4. 在匹配时，还需判断是否存在 matched.txt 文件，如果存在，则说明存在配成功

的数据，没有发送到服务器，因此不进行匹配。

5．匹配数据成功后，将 log.txt 文件删除；

6．向服务器发送数据。将 matched.txt 文件中的数据发送到服务器，发送成功后，将 matched.txt 文件删除。并在界面显示当前操作的结果，并进入下一次提取、解析和匹配等操作。如果发送数据不成功，也需要在界面显示当前操作的结果，然后直接进入下一次提取、解析和匹配等操作。

DMS 项目采集端的系统功能如图 - 9 所示：

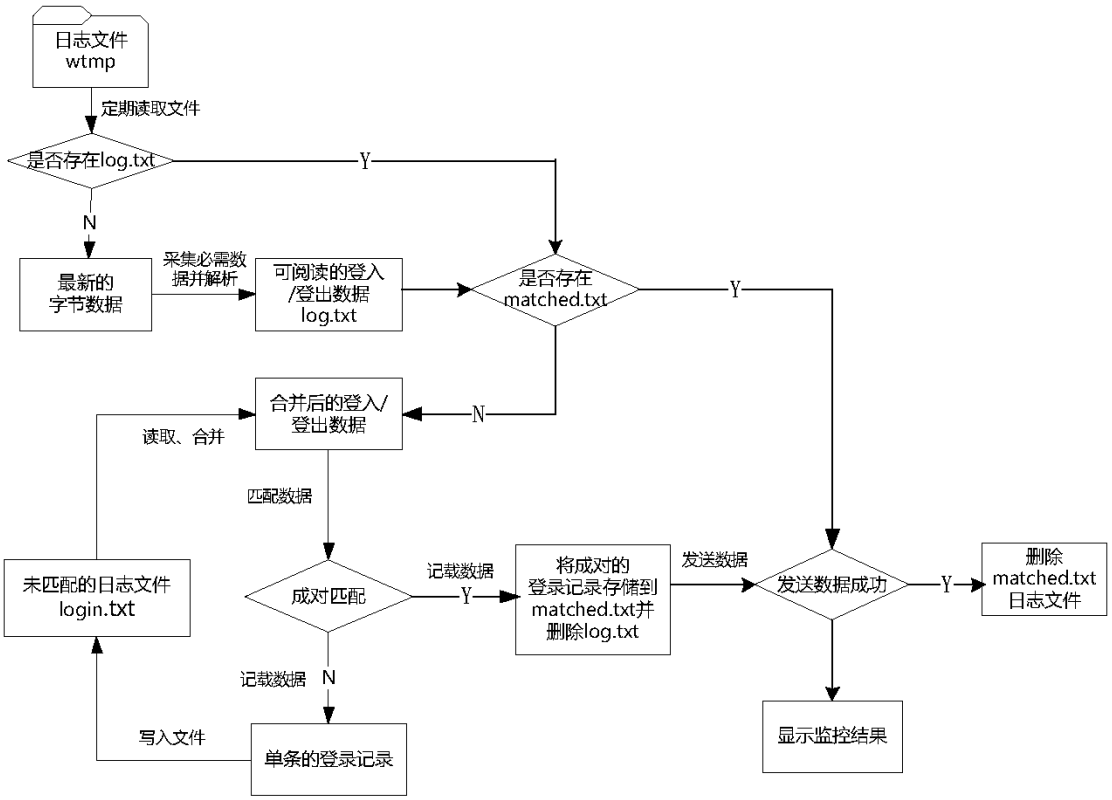

图 - 9

二．DMS 项目服务器端的主要功能如下：

1．持续接收 DMS 采集端发来的登录数据，并将数据保存到归档日志文件 server.txt 中；

2．显示数据接收以及操作的结果，以实现数据监控。

DMS 项目服务器端的系统功能如图 - 10 所示：

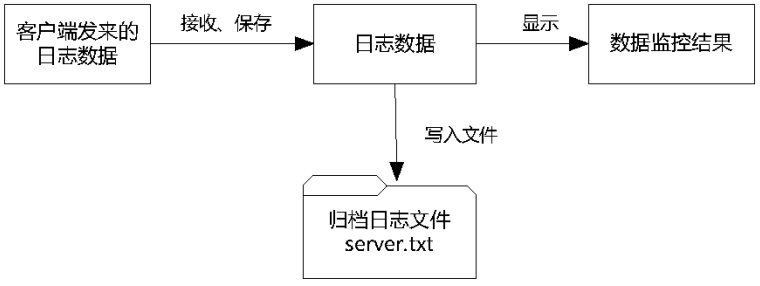

图 - 10

三．DMS 项目数据分析如下：

日志文件 wtmpx 是实时动态增长的，每 372 字节的数据代表一条登入/登出记录。因此，DMS 采集端程序从日志文件中读取数据时，只需要读取从上次读取到本次读取之间所产生的最新的日志数据。即，采集端程序需要从上次读取的末尾位置开始读取，以每 372 字节的数据为单位进行读取，直到接近文件末尾。DMS 采集端程序读取日志文件的过程如图 - 11 所示：

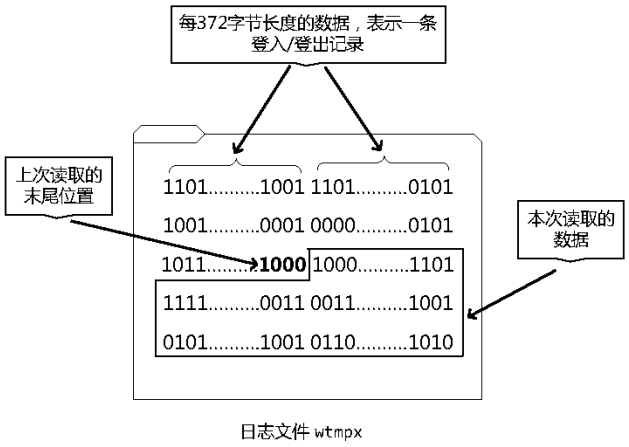

图 - 11

图 - 11 中红色字体部分表示上次读取的末尾位置，则本次读取从此开始。

DMS 采集端程序从日志文件 wtmpx 中得到数据后，将数据进行解析存储到文件 log.txt 中。该文件中包含多条登入/登出记录，每条记录（每 372 字节）中需要采集的信息有：

- 登录的 OS 账号
- 登录的进程 ID
- 登录/登出的时间
- 登录期间的在线时间
- 终端 IP

这些需要采集的数据信息在每 372 长度的字节中的位置如表 - 7 所示：

表 - 7 字节数组中需要采集的数据位置说明

位置范围	长度	数据含义
000-031	**32**	用户名，为文本类型数据（String）
068-071	**4**	进程 ID，为整数类型数据（int）
072-073	**2**	登录类型，值在 1-8 之间，但只处理 7 与 8 两种情况：7 为登入，8 为登出；为整数类型数据（short）
080-083	**4**	登入或者登出时刻，为整数类型数据（int）
114-371	**258**	用户登录的终端 IP，为文本类型数据（String）

每 372 长度的登入/登出数据中需要采集的数据如图 - 12 所示：

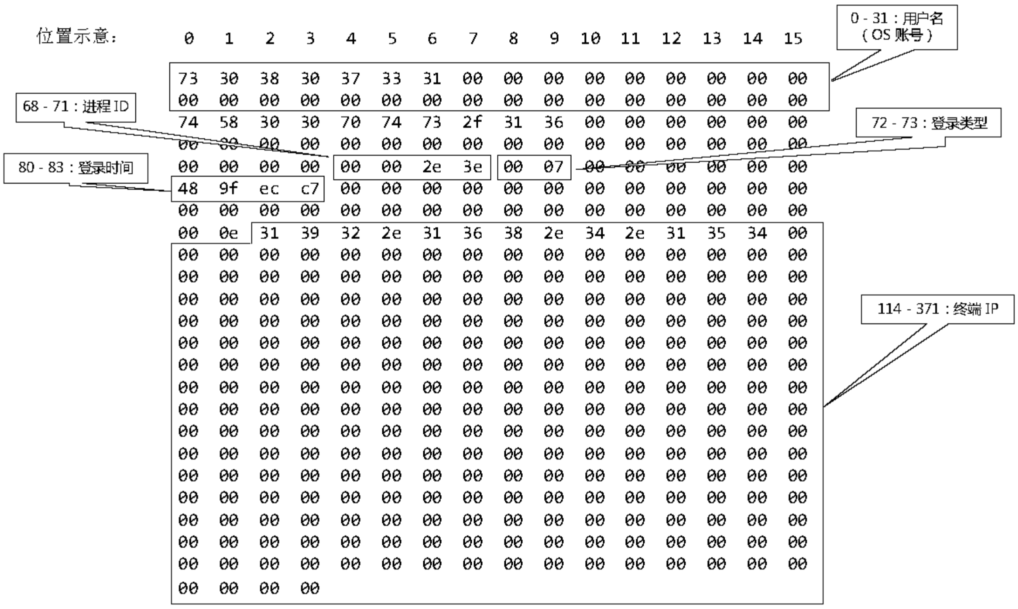

图 - 12

提取日志文件 wtmpx 中每 372 字节长度的数据，并采集表 - 7 和图 - 12 中所示位置的必需数据后，加以解析以形成可阅读的数据后保存为对象 Log，最终保存为 Log 对象的数据列表，所形成的列表中的记录形如图 - 13 所示：

Log对象的列表

序号	user	pid	type	time	ip
0	s080468	13003	8	1353484306	192.168.25.25
1	ws080246	13058	7	1353484468	192.168.25.30
2	as080714	13060	7	1353484692	192.168.25.57
3	ws080246	13058	8	1353484695	192.168.25.30
4	a080704	13105	7	1353484887	192.168.25.33
5	as080714	13060	8	1353485024	192.168.25.57
6	c090207	13015	7	1353485032	192.168.25.83

图 - 13

由图 - 13 可以看出，将每一条登录数据中的 OS 账号值存入 Log 对象的 user 属性，进程 ID 值存入 pid 属性，登录类型值存入 type 属性，登录时间值存入 time 属性，用户终端 IP 值存入属性 ip。

如果两条记录的用户登录名和进程 ID 相同，则为同一次登录会话。对于同一次会话的两条登录数据，需要根据登录类型来判断操作类型：7 为登入，8 为登出，其他类型的操作不用考虑。

因此，图 - 13 中序号为 1 和序号为 3 的两条记录可以匹配为一对，序号为 2 和序号为

5 的两条记录可以匹配为一对。

序号为 4 和 6 的记录为单独的登入记录，则表明 OS 账号 a080704 和 c090207 所对应的登出记录将出现在后续提取的日志记录之中。

序号为 0 的登录记录为单独的登出记录，则表明 OS 账号 s080468 的登入记录存在于在以前所提取的历史记录中。

正因为可能会有单条的登入或者登出数据，因此，DMS 采集端程序在解析完数据后，需要将本次得到的登录数据与上一次解析后留下的未匹配的日志数据合并，以实现数据匹配。

每次匹配后所遗留的历史登入记录保存为未匹配的日志文件 login.txt，假设该文件的内容格式如图 - 14 所示：

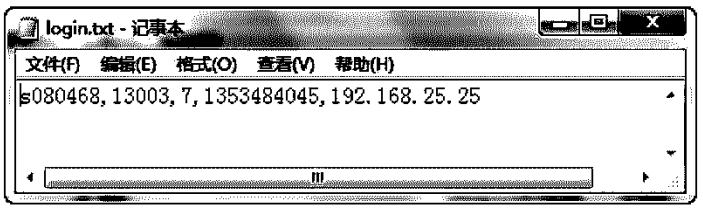

图 - 14

需要从 login.txt 文件中提取并解析历史登入数据，并转换为 Log 对象的数据列表，形如图 - 15 所示：

login.txt文件中读取的历史登入数据（Log 对象的列表）

序号	user	pid	type	time	ip
1	**s080468**	**13003**	**7**	**1353484045**	**192.168.25.25**

图 - 15

将图 - 13 所示的数据与图 - 15 所示的历史登入数据合并后，得到的数据记录形如图 - 16 所示：

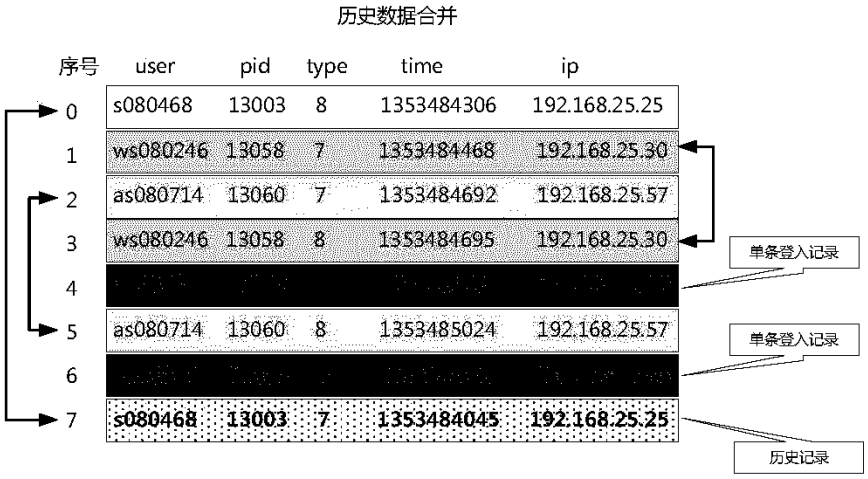

图 - 16

得到如图 - 16 中所示的合并后的数据后，DMS 采集端系统程序则需要进一步对登录

记录进行匹配，将每个 OS 账号的一次登入和登出信息进行匹配，以形成一次完整的登录会话的记录。

对图 - 16 中的示例数据按照登入/登出进行匹配，每匹配成功的一对记录存入对象 LogPair，匹配成功的所有对象则存入 LogPair 对象的列表作为匹配结果；而所有没有匹配成功的登入数据则存入 Log 的列表作为未匹配的登入记录。

对图 - 16 所示的数据进行匹配，并分别保存匹配结果和历史数据的过程形如图 - 17 所示：

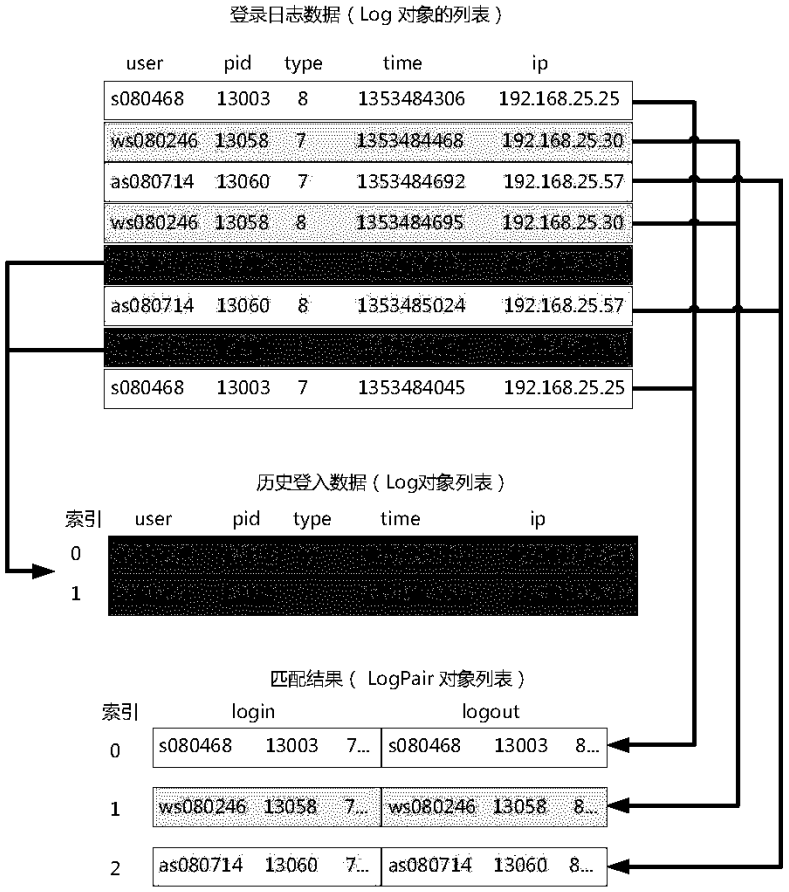

图 - 17

由图 - 17 可以看出，DMS 采集端系统将没有匹配成功的登入数据对象，记载到 Log 对象的列表中，并将该数据记载为未匹配的日志文件 login.txt，如图 - 18 所示：

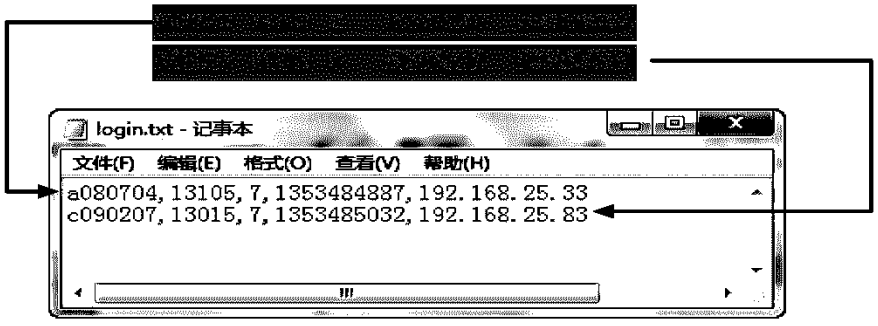

图 - 18

对于图 - 18 中匹配成功后的数据对象，则记录到 LogPair 对象的列表中，准备将数据发送到服务器端。

四．DMS 系统采集端的类图如图 - 19 所示

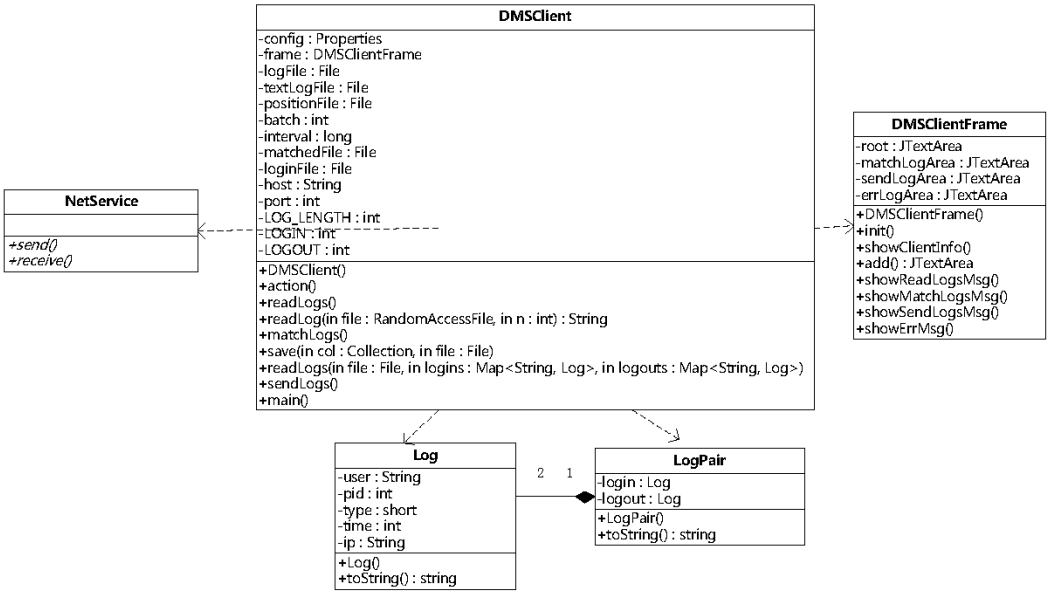

图 - 19

五．DMS 系统采集端的类图如图 - 20 所示

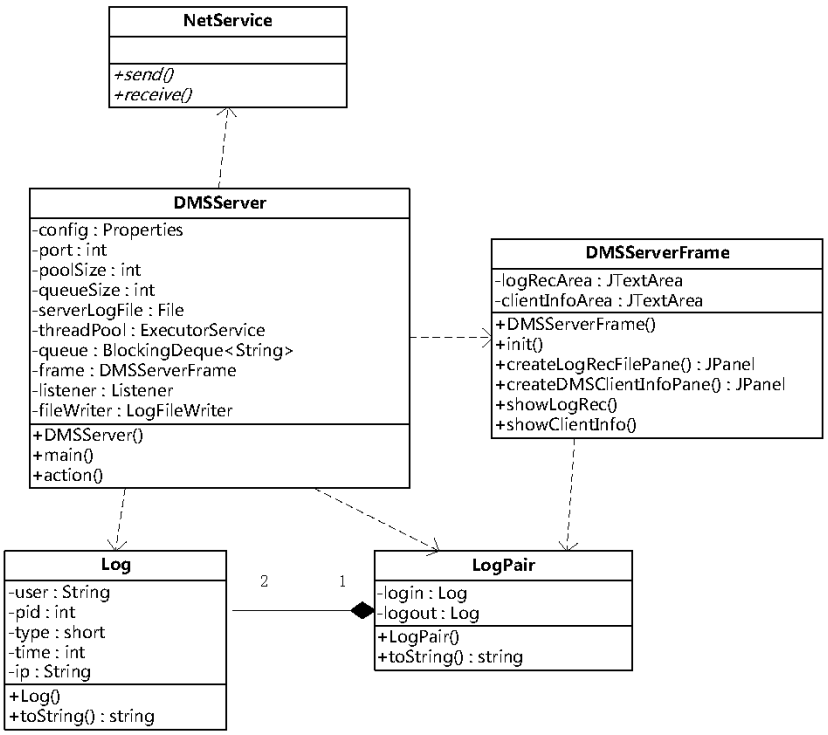

图 - 20

六．类功能描述如下：

1. 类 Log:日志数据业务类，代表从日志中解析出来的日志记录，每条日志记录解析为一个 Log 对象。
2. 类 LogPair："日志记录对" 业务对象，包含一条登录记录和一条登出记录。
3. 类 DMSClient :采集客户端核心业务程序。主要用于获取日志数据、解析日志数据，匹配日志数据以及向采集服务器发送日志数据。
4. 类 DMSClientFrame：采集客户端界面，显示对采集端执行情况的监控。
5. 类 DMSServer：采集服务器接收程序。接收到数据以后存储到文件中。
6. 类 DMSServerFrame：采集服务器界面，显示对服务器端执行情况的监控。
7. 类 NetService：XML 发送和接收协议。

3. 步骤

实现此案例需要按照如下步骤进行。

步骤一：创建工程

创建名为 DMS 的 Java 工程， 并将 wtmpx 文件拷贝到 DMS 工程下，工程结构图如图 - 21 所示：

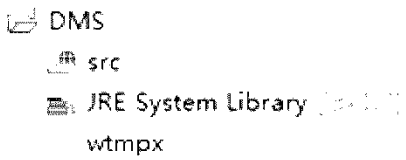

图 - 21

另外，wtmpx 文件是原始日志文件，可以从老师处获取到。

步骤二：构建 DMS 项目的采集客户端结构

首先，在项目 DMS 的 src 目录下新建包 com.tarena.dms；然后，在该包下新建类 DMSClient；最后，在该类中构建 DMS 采集客户端的结构，代码如下所示：

```
package com.tarena.dms;

import java.io.File;

public class DMSClient {

 /** 二进制日志文件 */
 private File logFile;
 /** 文本日志文件 */
 private File textLogFile;
 /** 存储指针位置文件 */
 private File positionFile;
 private int batch;// 每批读取的数量
 /** 主循环间隔时间 */
 private long interval;

 /**软件启动的方法*/
```

```
public void action() {
    while (true) {
        readLogs();
        matchLogs();
        sendLogs();
        try {
            Thread.sleep(interval);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
/**读取日志的方法*/
public void readLogs() {
}
/**匹配日志的方法*/
public void matchLogs() {
}
/**向服务器发送日志的*/
public void sendLogs() {
}
public static void main(String[] args) {
    DMSClient client = new DMSClient();
    client.action();// 启动方法
}
}
```

步骤三：将属性实例化对象

将上一步定义的属性，进行实例化，代码如下所示：

```
package com.tarena.dms;

import java.io.File;

public class DMSClient {

/** 二进制日志文件 */
private File logFile;
/** 文本日志文件 */
private File textLogFile;
/** 存储指针位置文件 */
private File positionFile;
private int batch;// 每批读取的数量
/** 主循环间隔时间 */
private long interval;


public DMSClient(){
    logFile = new File("wtmpx");
    textLogFile = new File("log.txt");
    positionFile = new File("position.txt");
    batch=10;
    interval=5000;
}

/**软件启动的方法*/
public void action() {
    while (true) {
```

```
            readLogs();
            matchLogs();
            sendLogs();
            try {
                Thread.sleep(interval);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    /**读取日志的方法*/
    public void readLogs() {
    }
    /**匹配日志的方法*/
    public void matchLogs() {
    }
    /**向服务器发送日志的*/
    public void sendLogs() {
    }

    public static void main(String[] args) {
        DMSClient client = new DMSClient();
        client.action();// 启动方法
    }
}
```

步骤四：实现 readLogs 方法

readLogs 方法实现成批的读取原始日志文件（wtmpx）到文本日志文件（log.txt）中，实现该方法的详细步骤如下：

1．检查 log.txt 是否存在。如果存在，则直接返回，不对 wtmpx 文件的日志信息进行读取；否则，打开文件 wtmpx 进行读取；

2．在读取 wtmpx 文件中的内容之前，首先判断 position.txt 文件是否存在，如果不存在则从 wtmpx 文件的开始进行读取；

3．如果 position.txt 文件存在，则检查是否有新的日志记录产生，如果有则进行读取；

4．连续读取一批日志记录到临时文件；

5．保存下次的读取 wtmpx 文件的位置 position；

6．将临时文件改名为 log.txt。

实现 readLogs 方法的代码如下所示：

```
package com.tarena.dms;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.util.Scanner;

public class DMSClient {

    /** 二进制日志文件 */
    private File logFile;
```

```
/** 文本日志文件 */
private File textLogFile;
/** 存储指针位置文件 */
private File positionFile;
private int batch;// 每批读取的数量
/** 主循环间隔时间 */
private long interval;

public static final int LOG_LENGTH = 372;


public DMSClient() {
    logFile = new File("wtmpx");
    textLogFile = new File("log.txt");
    positionFile = new File("position.txt");
    batch = 10;
    interval = 5000;
}

public void action() {
    while (true) {
        readLogs();
        matchLogs();
        sendLogs();
        try {
            Thread.sleep(interval);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}


public void readLogs() {
    if (!logFile.exists()) {
         System.out.println( "没有输入日志文件"+logFile);
        return;
    }
    if (textLogFile.exists()) {
        return;
    }
    // 读取位置，是起始读取行数
    int position;// 第一行的位置
    //  读取上次的读取位置(没有的从开始位置读取)
    try {
        // positionFile 是文本文件，保存一个整数
        Scanner in = new Scanner(positionFile);
        // "10" -> 10
        position = in.nextInt();
        in.close();
    } catch (FileNotFoundException e) {
        position = 0;
    }
    //  根据上次的读取位置和文件长度判断，是否有新数据
    if (position * LOG_LENGTH == logFile.length()) {
        return;
    }
    //  从上次读取位置开始，批量读取到 临时文件
    // 如果读取的文件末尾就不读取了
    RandomAccessFile in = null;
    PrintWriter out = null;
```

```
        try {
            in = new RandomAccessFile(logFile, "r");
            File temp = new File(System.nanoTime() + "L.txt");
            out = new PrintWriter(temp, "utf-8");
            int i; // 10 //position = 0
            for (i = 0; i < batch; i++) {
                int n = i + position;
                String log = readLog(in, n);
                if (log == null) {
                    break;
                }
                out.println(log); // 保存临时文件
            } // ? i=10
            position = position + i;// 10
            out.close();// 文件关闭以后才能改名
            // 将临时文件改名到 textLogFile
            if (temp.renameTo(textLogFile)) {
                // 保存下次读取位置
                PrintWriter pw = new PrintWriter(positionFile);
                pw.println(position);
                pw.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
            return;
        } finally {
            try {
                if (in != null)
                    in.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    /** 读取日志方法 */
    public String readLog(RandomAccessFile file, int n) throws IOException {
        long start = (long) LOG_LENGTH * n;
        // 读取登录用户名 0
        file.seek(start + 0);
        byte[] buf = new byte[32];
        int c = file.read(buf);
        // 检测是否读取到文件的末尾
        if (c == -1) {
            return null;
        }
        String name = new String(buf, "iso8859-1").trim();
        // 读取登录进程号码 68
        file.seek(start + 68);
        int pid = file.readInt();
        // 读取登录类型 72
        file.seek(start + 72);
        short type = file.readShort();
        // 读取时间
        file.seek(start + 80);
        int time = file.readInt();
        // 读取 IP
        file.seek(start + 114);
        buf = new byte[258];
        file.read(buf);
```

```
        String ip = new String(buf, "iso8859-1").trim();
        return name + "," + pid + "," + type + "," + time + "," + ip;
    }

    public void matchLogs() {
    }

    public void sendLogs() {
    }
    public static void main(String[] args) {
        DMSClient client = new DMSClient();
        client.action();// 启动方法
    }
}
```

步骤五：测试 readLogs 方法

首先，在工程中添加 JUnit4 的支持；然后添加测试方法 testReadLogs，代码如下所示：

```
package com.tarena.dms;

import org.junit.Test;

public class TestCase {
    @Test
    public void testReadLogs() {
        // 对日志读取方法进行单独测试。
        DMSClient client = new DMSClient();
        client.readLogs();
    }
}
```

运行测试方法 testReadLogs 类，刷新 DMS 工程，会发现工程中多了两个文件，一个是 log.txt、另一个是 position.txt，此时的工程结构如图 - 22 所示：

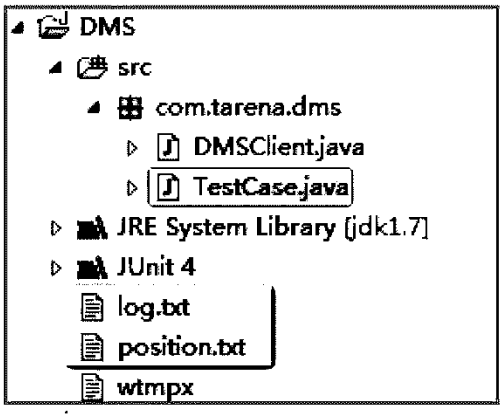

图 - 22

打开如图 - 22 中所示的 log.txt 会发现其中有 10 条日志记录，该文件内容如如图 - 23 所示：

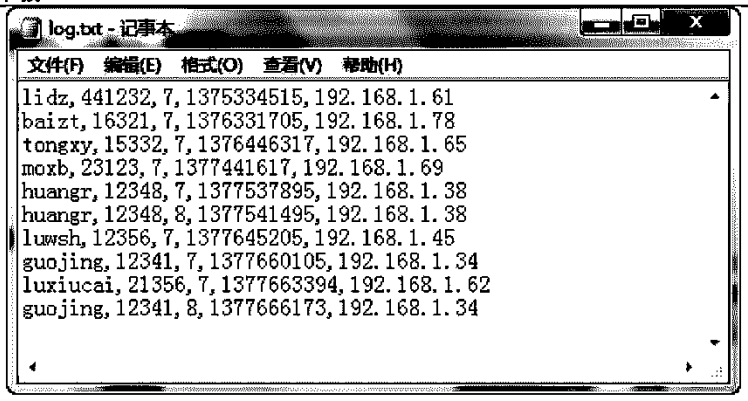

图 - 23

从 log.txt 的文件内容可以看出，读取到 10 条日志记录。以第一条日志记录为例说明各项表示的意义。其中“baizt”为登录的 OS 账号、“16321”为登录的进程 ID、“7”表示为登录日志记录、“1376331705”为登入期间的在线时间、“192.168.1.78”为终端 IP。

打开图 - 23 所示的 position.txt 文件会发现其中内容为 10。

步骤六：编写采集客户端界面

客户端界面的结构如下：

```
DMSClient Frame 设计规划
  |-- root JPanel 格子布局 2x2
      |-- JScrollPane 滚动面板
      |   |--readLogArea JTextArea 多行文本控件
      |-- JScrollPane 滚动面板
      |   |--matchLogArea JTextArea 多行文本控件
      |-- JScrollPane 滚动面板
      |   |--sendLogArea JTextArea 多行文本控件
      |-- JScrollPane 滚动面板
      |   |--errLogArea JTextArea 多行文本控件
```

编写客户端界面的代码如下所示：

```
package com.tarena.dms;

import java.awt.GridLayout;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.TitledBorder;

/**
 * DMS 客户端界面
 *
 */
public class DMSClientFrame extends JFrame {
 private JPanel root;
 private JTextArea readLogArea;
```

```
private JTextArea matchLogArea;
private JTextArea sendLogArea;
private JTextArea errLogArea;

public DMSClientFrame() {
    // 初始化界面
    setTitle("DMS Client");
    setSize(800, 600);
    setLocationRelativeTo(null);
    // 控件的大小和位置成为控件的布局位置
    // GridLayout 表格布局，将面板设置为
    // 等宽等高的格子，如：(2,2) 两行两列 4 个
    // 区域，添加控件，控件的大小和位置就
    // 自动的出现在这个区域
    root = new JPanel(new GridLayout(2, 2));
    add(root);
    readLogArea = add("读取日志");
    matchLogArea = add("匹配日志");
    sendLogArea = add("发送日志");
    errLogArea = add("异常消息");
}

/**
 * 生产一个显示区域 在当前 root 面板上添加一个显示消息的区域
 *
 * @param title
 *            是边框的标题
 */
public JTextArea add(String title) {
    JTextArea area = new JTextArea();
    // JTextArea 必须放置在 JScrollPane 中
    JScrollPane pane = new JScrollPane(area);
    // 将滚动面板添加到 窗口的主面板上
    root.add(pane);
    // 给滚动面板增加 带有标题的边框
    pane.setBorder(new TitledBorder(title));
    // 禁止 area 编辑功能
    area.setEditable(false);
    return area;
}

public static void main(String[] args) {
    // 界面的测试方法，不是软件启动方法
    DMSClientFrame frame = new DMSClientFrame();
    frame.setDefaultCloseOperation(EXIT_ON_CLOSE);
    frame.setVisible(true);
}
}
```

运行 DMSClientFrame 类，界面效果如图 - 24 所示：

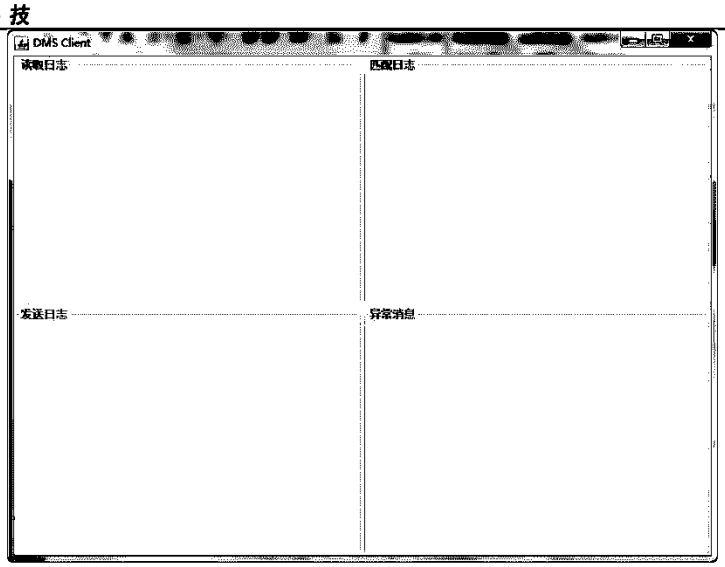

图 - 24

步骤七：在客户端界面上显示读取日志过程中的监控信息

首先，修改 DMSClientFrame 类，添加 showReadLogsMsg 方法，代码如图-25 所示：

```
/** 在DMSClientFrame 上添加方法, 显示
 * 读取日志的消息 */
public void showReadLogsMsg(String msg) {
    //在这个界面的readLogArea区域增加消息
    this.readLogArea.append(msg+"\n");
    //Area 区域
}
```

图 - 25

然后，修改 DMSClient 类，添加监控信息的代码，添加的代码如下所示：

```
package com.tarena.dms;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.util.Scanner;

import javax.swing.JFrame;

public class DMSClient {

/** 软件界面 */
private DMSClientFrame frame;

/** 二进制日志文件 */
private File logFile;
/** 文本日志文件 */
private File textLogFile;
/** 存储指针位置文件 */
```

```
private File positionFile;
private int batch;// 每批读取的数量
/** 主循环间隔时间 */
private long interval;

public static final int LOG_LENGTH = 372;

public DMSClient() {

    frame = new DMSClientFrame();

    logFile = new File("wtmpx");
    textLogFile = new File("log.txt");
    positionFile = new File("position.txt");
    batch = 10;
    interval = 5000;
}

public void action() {

    // 软件起动时候显示界面.
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);

    while (true) {
        readLogs();
        matchLogs();
        sendLogs();
        try {
            Thread.sleep(interval);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public void readLogs() {

    frame.showReadLogsMsg("开始读取日志...");

    if (!logFile.exists()) {
        // System.out.println(
        // "没有输入日志文件"+logFile);
        // 在界面上显示读取日志的消息

        frame.showReadLogsMsg("没有输入日志文件" + logFile);

        return;
    }
    if (textLogFile.exists()) {

        frame.showReadLogsMsg("输出文件已经存在,本次不读取了!");

        return;
    }
    // 读取位置，是起始读取行数
    int position;// 第一行的位置
```

```
// 读取上次的读取位置(没有的从开始位置读取)
try {
    // positionFile 是文本文件，保存一个整数
    Scanner in = new Scanner(positionFile);
    // "10" -> 10
    position = in.nextInt();
    in.close();

    frame.showReadLogsMsg("从文件读取位置:" + position);

} catch (FileNotFoundException e) {

    frame.showReadLogsMsg("从 0 读取");

    position = 0;
}
// 根据上次的读取位置和文件长度判断，是否有新数据
if (position * LOG_LENGTH == logFile.length()) {

    frame.showReadLogsMsg("没有新 log 产生，本次读取结束!");

    return;
}
// 从上次读取位置开始，批量读取到 临时文件
// 如果读取的文件末尾就不读取了
RandomAccessFile in = null;
PrintWriter out = null;
try {
    in = new RandomAccessFile(logFile, "r");

    frame.showReadLogsMsg("打开文件" + logFile);

    File temp = new File(System.nanoTime() + "L.txt");
    out = new PrintWriter(temp, "utf-8");

    frame.showReadLogsMsg("打开临时文件" + temp);

    int i; // 10 //position = 0
    for (i = 0; i < batch; i++) {
        int n = i + position;
        String log = readLog(in, n);

        frame.showReadLogsMsg("读取日志" + n + ":" + log);

        if (log == null) {
            break;
        }
        out.println(log); // 保存临时文件

        frame.showReadLogsMsg("保存到临时文件");

    } // ? i=10
    position = position + i;// 10
```

```
            frame.showReadLogsMsg("下次读取位置" + position);
            out.close();// 文件关闭以后才能改名
            // 将临时文件改名到 textLogFile
            if (temp.renameTo(textLogFile)) {

                frame.showReadLogsMsg("临时文件改名:" + textLogFile);

                // 保存下次读取位置
                PrintWriter pw = new PrintWriter(positionFile);
                pw.println(position);
                pw.close();

                frame.showReadLogsMsg("保存下次读取位置:" + positionFile);

            } else {

                frame.showReadLogsMsg("临时文件改名失败!");

            }
        } catch (IOException e) {
            e.printStackTrace();

            frame.showReadLogsMsg("读取日志失败!" + e);

            return;
        } finally {
            try {
                if (in != null)
                    in.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        frame.showReadLogsMsg("结束读取!");

    }

    /** 读取日志方法 */
    public String readLog(RandomAccessFile file, int n) throws IOException {
        ... ...
    }

    public void matchLogs() {
        ... ...
    }

    public void sendLogs() {
        ... ...
    }
    public static void main(String[] args) {
        DMSClient client = new DMSClient();
        client.action();// 启动方法
    }
}
```

最后，运行 DMSClient 类，界面显示效果如图 - 26 所示：

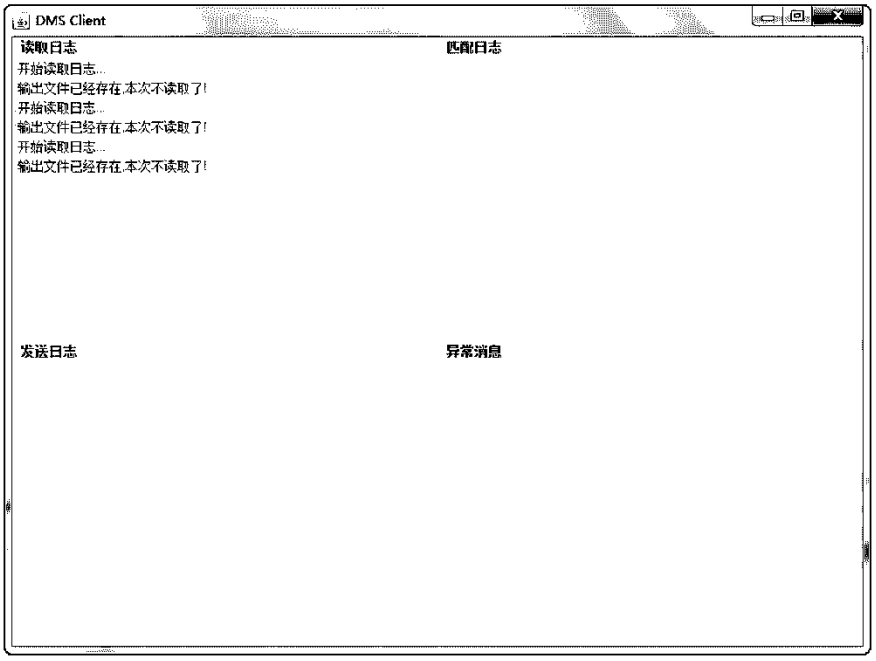

图 - 26

步骤八：定义显示异常信息、显示发送日志信息、显示匹配日志信息的方法

在 DMSClientFrame 类中，添加 showErrMgs 方法，用于显示异常信息；添加 showSendLogsMsg 方法，用于显示发送日志信息；定义是 showMatchLogsMsg 方法，用于显示匹配日志信息，代码如图-27 所示：

```
/** 显示匹配日志消息 */
public void showMatchLogsMsg(String msg){
    this.matchLogArea.append(msg+"\n");
}
/** 显示发送日志消息 */
public void showSendLogsMsg(String msg){
    this.sendLogArea.append(msg+"\n");
}
/** 在界面显示异常消息 */
public void showErrMsg(Exception e) {
    //StringWriter == StringBuilder
    //  writer(char)   append(char)
    //建立一个字符串缓冲区
    //StringWriter 是字符节点流，是字符缓冲区
    StringWriter buf = new StringWriter();
    PrintWriter out = new PrintWriter(buf);
    //将异常跟踪消息写到缓冲区
    e.printStackTrace(out);//PrintWriter
    out.close();
    //读取缓冲区中的数据，在界面显示出来
    String msg = buf.toString();
    this.errLogArea.append(msg+"\n");
}
```

图 - 27

在 DMSClient 类的 readLogs 方法的异常处理添加显示异常信息的代码 ,代码如图-28 所示：

```
} catch (IOException e) {
    e.printStackTrace();
    frame.showReadLogsMsg("读取日志失败!" + e);
    frame.showErrMsg(e);
    return;
} finally {
    try {
        if (in != null)
            in.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

图 - 28

步骤九：加载属性文件

新建名为 client.properties 的属性文件，在该文件中存储客户端可能需要经常改变的信息，文件内容如下所示：

```
# client.properties
server.ip=localhost
# server.ip=192.168.100.45
server.port=8899
log.file=wtmpx
#log.file=/var/adm/wtmpx
position.file=position.txt
text.log.file=log.txt
batch=10
interval=5000
matched.log.file=matched.txt
login.log.file=login.txt
```

在 DMSClient 类的构造方法中，使用 Properties 类的 load 方法加载属性文件的信息，代码如下所示：

```
public class DMSClient {
 /** 软件界面 */
 private DMSClientFrame frame;
 /** 二进制日志文件 */
 private File logFile;
 /** 文本日志文件 */
 private File textLogFile;
 /** 存储指针位置文件 */
 private File positionFile;
 private int batch;// 每批读取的数量
 /** 主循环间隔时间 */
 private long interval;

 /** 读取配置文件 */
 private Properties config;
```

```
    /** 匹配对的日志文件 */
    private File matchedFile;
    /** 单条登录日志 */
    private File loginFile;
    /** 服务器 IP */
    private String host;
    /** 服务器端口号 */
    private int port;

    public static final int LOG_LENGTH = 372;

    public DMSClient() {

        frame = new DMSClientFrame();
        config = new Properties();
        try {
            FileInputStream in = new FileInputStream("client.properties");
            // 读取配置文件，到内存的散列表
            config.load(in);
        } catch (Exception e) {
            frame.showErrMsg(e);
            // 配置文件读取异常，就不能再执行软件了
            throw new RuntimeException(e);
        }
        // 切记：log.file 这个 key 在配置文件中存在！
        logFile = new File(config.getProperty("log.file"));
        textLogFile = new File(config.getProperty("text.log.file"));
        positionFile = new File(config.getProperty("position.file"));
        batch = Integer.parseInt(config.getProperty("batch"));
        interval = Long.parseLong(config.getProperty("interval"));
        matchedFile = new File(config.getProperty("matched.log.file"));
        loginFile = new File(config.getProperty("login.log.file"));
        host = config.getProperty("server.ip");
        port = Integer.parseInt(config.getProperty("server.port"));

    }

    public void action() {
        ... ...
    }

    public void readLogs() {
        ... ...
    }

    /** 读取日志方法 */
    public String readLog(RandomAccessFile file, int n) throws IOException {
       ... ...
    }

    public void matchLogs() {
    }

    public void sendLogs() {
    }
    public static void main(String[] args) {
        DMSClient client = new DMSClient();
        client.action();// 启动方法
    }
}
```

步骤十：实现 matchLogs 方法

matchLogs 方法实现匹配登入登出记录，即匹配读取 log.txt 文件的内容， 生成 matched.txt 文件，该方法的详细实现过程如下：

1．检查是否有输入日志文件 log.txt，如果没有则返回；

2．检查是否有输出日志文件 matched.txt，如果有则返回。存在 matched.txt 文件说明存在日志数据没有成功向服务器发送，因此不向 matched.txt 中写数据；

3．如果 login.txt 文件存在，读取 login.txt 文件中日志记入到 Map 集合 logins 和 logouts。logins 用于存放登入记录、logout 用于存放登出记录；

4．读取 log.txt 文件中的数据到 Map 集合 logins 和 logouts；

5．迭代 Map 集合 logouts，获取到该集合的每一个的 key，并根据该 key 到 Map 集合 logins 中查找对应的登入记录，如果找到对应的登入记录，则形成登录对 ，存放到 List 集合 matched<LogPair>中；

6．存放记录对成功后，将集合 logins 的该登录记录删除；

7．将 mathced 集合中的记录对保存到文件 matched.txt；

8．将 logins 集合中没有匹配的单条登录记录保存到 login.txt 文件中；

9．删除文件 log.txt。

编写代码过程如下：

首先，在 com.tarena.dms 包下新建类 Log，该类用于表示日志对象，一条日志记录对应一个 Log 对象，代码如下所示：

```
package com.tarena.dms;

import java.io.Serializable;
/**
 * 日志对象
 */
public class Log implements Serializable {
 private String user;
 private int pid;
 private short type;
 private int time;
 private String ip;

 public Log() {
 }

 public Log(String user, int pid, short type, int time, String ip) {
     super();
     this.user = user;
     this.pid = pid;
     this.type = type;
     this.time = time;
     this.ip = ip;
 }
 /**
  *  lidz,441232,7,1375334515,192.168.1.61
  */
 public Log(String str){
     String[] data=str.split(",");
     user = data[0];
     pid = Integer.parseInt(data[1]);
     type = Short.parseShort(data[2]);
```

```
        time = Integer.parseInt(data[3]);
        ip = data[4];
    }

    @Override
    public String toString() {
        return user+","+pid+","+type+","+time+","+ip;
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public int getPid() {
        return pid;
    }

    public void setPid(int pid) {
        this.pid = pid;
    }

    public short getType() {
        return type;
    }

    public void setType(short type) {
        this.type = type;
    }

    public int getTime() {
        return time;
    }

    public void setTime(int time) {
        this.time = time;
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }
}
```

然后，在 com.tarena.dms 包下新建类 LogPair，该类用于表示登录日志对，一对匹配的登入登出日志记录对应一个 LogPair 对象，代码如下所示：

```
package com.tarena.dms;

import java.io.Serializable;
/**
 * 登录日志对
 */
public class LogPair implements Serializable {
 private Log login;
 private Log logout;

 public LogPair() {
 }
```

```
    public LogPair(Log login, Log logout) {
        this.login = login;
        this.logout = logout;
    }

    public LogPair(String str){
        String[] data = str.split("\\|");
        login = new Log(data[0]);
        logout = new Log(data[1]);
    }

    @Override
    public String toString() {
        return login+"|"+logout;
        //login.toString()+"|"+logout.toString();
    }

    public Log getLogin() {
        return login;
    }

    public void setLogin(Log login) {
        this.login = login;
    }

    public Log getLogout() {
        return logout;
    }

    public void setLogout(Log logout) {
        this.logout = logout;
    }
}
```

最后，DMSClient 类的实现代码如下：

```
package com.tarena.dms;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Scanner;
import java.util.Set;

import javax.swing.JFrame;

import org.dom4j.Document;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;
```

```
public class DMSClient {
/** 软件界面 */
private DMSClientFrame frame;
/** 二进制日志文件 */
private File logFile;
/** 文本日志文件 */
private File textLogFile;
/** 存储指针位置文件 */
private File positionFile;
private int batch;// 每批读取的数量
/** 主循环间隔时间 */
private long interval;
/** 读取配置文件 */
private Properties config;
/** 匹配对的日志文件 */
private File matchedFile;
/** 单条登录日志 */
private File loginFile;
/** 服务器 IP */
private String host;
/** 服务器端口号 */
private int port;
public static final int LOG_LENGTH = 372;

private static final short LOGIN = 7;
private static final short LOGOUT = 8;


/** 初始化成员变量 */
public DMSClient() {
    ... ...
}
public void action() {
    ... ...
}
public void readLogs() {
    ... ...
}
/** 读取日志方法 */
public String readLog(RandomAccessFile file, int n) throws IOException {
    ... ...
}

public void matchLogs() {
    frame.showMatchLogsMsg("开始匹配计算.");
    if (!textLogFile.exists()) {
        frame.showMatchLogsMsg("没有文件:" + textLogFile);
        return;
    }
    if (matchedFile.exists()) {
        frame.showMatchLogsMsg("还存在匹配的日志，没有发送");
        return;
    }
    // 读取日志
    Map<String, Log> logins = new HashMap<String, Log>();
    Map<String, Log> logouts = new HashMap<String, Log>();
    try {
        readLogs(loginFile, logins, logouts);
```

```
            readLogs(textLogFile, logins, logouts);
        } catch (IOException e) {
            frame.showErrMsg(e);
            return;
        }
        List<LogPair> matched = new ArrayList<LogPair>();
        Set<String> keySet = logouts.keySet();
        for (String key : keySet) {
            frame.showMatchLogsMsg("key:" + key);
            Log login = logins.remove(key);
            if (login != null) {
                Log logout = logouts.get(key);
                frame.showMatchLogsMsg("fond:" + login + "," + logout);
                matched.add(new LogPair(login, logout));
            }
        }
        // 匹配结束保存文件
        File tempL = new File(System.nanoTime() + "L.txt");// logins
        File tempM = new File(System.nanoTime() + "m.txt");// matched
        try {
            save(matched, tempM);
            save(logins.values(), tempL);
            frame.showMatchLogsMsg("保存文件成功了");
            if (tempM.renameTo(matchedFile)) {
                loginFile.delete();
                if (tempL.renameTo(loginFile)) {
                    textLogFile.delete();
                    frame.showMatchLogsMsg("删除:" + textLogFile);
                    frame.showMatchLogsMsg("结束匹配.");
                }
            }
        } catch (Exception e) {
            frame.showErrMsg(e);
            return;
        }
    }

    private void save(Collection col, File file) throws IOException {
        PrintWriter out = new PrintWriter(file);
        for (Object obj : col) {
            out.println(obj);// obj.toString()
        }
        out.close();
    }

    private void readLogs(File file, Map<String, Log> logins,
            Map<String, Log> logouts) throws IOException {
        if (!file.exists()) {
            return;
        }
        BufferedReader in = new BufferedReader(new InputStreamReader(
                new BufferedInputStream(new FileInputStream(file))));
        String str;
        while ((str = in.readLine()) != null) {
            Log log = new Log(str);
            String key = log.getUser() + "," + log.getPid() + "," + log.getIp();
            if (log.getType() == LOGIN) {
                logins.put(key, log);
            } else if (log.getType() == LOGOUT) {
                logouts.put(key, log);
            }
        }
        in.close();
    }
```

```
    public static void main(String[] args) {
        DMSClient client = new DMSClient();
        client.action();// 启动方法
    }
}
```

步骤十一：测试 matchLogs 方法

首先，在 TestCase 类中添加 testMatchLogs 方法，该方法的代码如图-29 所示：

```
@Test
public void testMatchLogs() {
    // 对日志读取方法进行单独测试.
    DMSClient client = new DMSClient();
    client.matchLogs();
}
```

图 - 29

运行 testMatchLogs 类，刷新工程，此时的工程结构如图-30 所示：

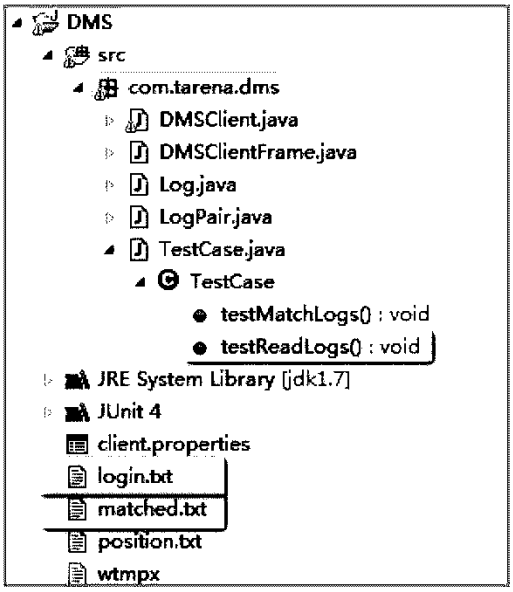

图 - 30

由图 - 30 可以看出，在 DMS 工程下多了两个文件，一个文件是 login.txt、另一个文件是 matched.txt。login.txt 文件中有 6 条登录日志记录，如图 - 31 所示：

图 - 31

matched.txt 文件中有两条日志对，如图 - 32 所示：

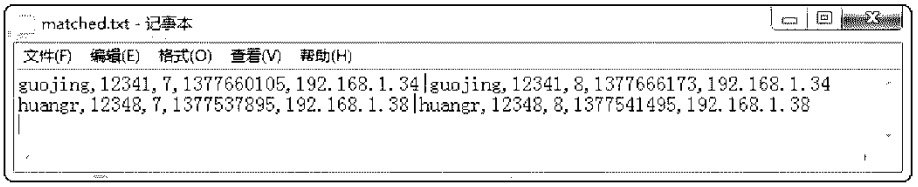

图 - 32

步骤十二：编写发送 XML 格式数据的网络协议

首先，新建类 NetService，该类负责 XML 数据的发送和接收。发送 XML 数据的协议报文设计为：type、length、value，分别表示发送数据的类型、要发送的数据长度、要发送的 XML 数据，其长度与 length 保持一致。

然后，在 NetService 类中，新建 send 方法，用于发送 XML 数据，该方法的实现过程如下：

1. 将 XML Document 写入到内存的缓冲区中，得到全部的字节数据，同时得到数据的长度；
2. 发送 type 和 length，此两项数据共 50 个字节，不足则凑足 50 个字节；
3. 将 XML 的全部字节数据发送到输出流中。

最后，在 NetService 类中，新建 receive 方法，用于接收 XML 数据，该方法的实现过程如下：

1. 先接收 type 和 length；
2. 如果 type 不是 "XML"，则抛出异常；
3. 根据长度，接收 value，接收发送过来的 XML 数据，从流中接收 length 个 byte 到数组缓冲区；
4. 将数组缓冲区中的数据，解析为 XML Document。

NetService 类的代码如下所示：

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Arrays;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.io.OutputFormat;
import org.dom4j.io.SAXReader;
import org.dom4j.io.XMLWriter;

public class NetService {
 /**
  * 将 doc 对象通过网络流 out 发送到服务器
  *
  * @param doc
  *          xml 对象
  * @param out
```

```
     *            网络输出流
     */
    public static void send(Document doc, OutputStream out)
                                throws IOException {
        // 先将 doc 转换为 bytes
        ByteArrayOutputStream buf = new ByteArrayOutputStream();
        OutputFormat fmt = OutputFormat.createPrettyPrint();
        XMLWriter writer = new XMLWriter(buf, fmt);
        writer.write(doc);
        byte[] bytes = buf.toByteArray();
        // 发送 50 个 bytes: type,length
        String str = "xml," + bytes.length;
        byte[] tl = str.getBytes("utf-8");
        tl = Arrays.copyOf(tl, 50);
        out.write(tl);
        // 发送 bytes
        out.write(bytes);
        out.flush();
    }

    /**
     * 从网络流 in 中接收 XML Doc
     *
     * @param in
     *            网络输入流
     * @return XML Doc
     */
    public static Document receive(InputStream in) throws IOException,
            DocumentException {
        // 先接收 header (type + length)
        byte[] bytes = new byte[50];
        in.read(bytes);
        // 解析 Header 得到 length
        String header = new String(bytes, "utf-8").trim();
        String[] data = header.split(",");
        String type = data[0];
        if (!"XML".equals(type)) {
            throw new DocumentException("不是 XML 文件格式");
        }
        int length = Integer.parseInt(data[1]);
        // 再根据长度接收 bytes
        bytes = new byte[length];
        in.read(bytes);
        // 最后将 bytes 解析为 Document
        SAXReader reader = new SAXReader();
        Document doc = reader.read(new ByteArrayInputStream(bytes));
        return doc;
    }
}
```

步骤十三：测试网络协议的正确性

在 TestCase 中添加测试方法 testSend，在该方法中，使用文件输入流和文件输出流替换网络流进行测试，向文件 demo.txt 中发送和接受数据，代码如图-33 所示：

```
@Test
public void testSend() throws Exception {
    // 利用文件流对网络协议进行测试
    Document doc = DocumentHelper.createDocument();
    Element root = doc.addElement("test");
    root.addText("Hello World!");
    // 使用文件流 替换网络流
    FileOutputStream out = new FileOutputStream("demo.txt");
    NetService.send(doc, out);
    out.close();

    FileInputStream in = new FileInputStream("demo.txt");
    Document doc2 = NetService.receive(in);
    in.close();
    System.out.println(doc2.asXML());
}
```

图 - 33

运行 testSend 方法，控制台输出结果如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<test>Hello World!</test>
```

demo.txt 文件的内容如图-34 所示：

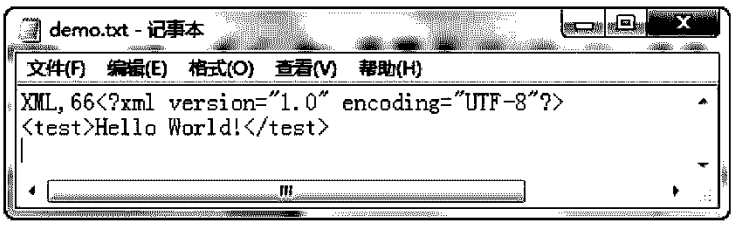

图 - 34

步骤十四：实现 sendLogs 方法

sendLogs 方法实现向服务器发送数据，该方法的详细实现过程如下：

1．检查是否存在被发送的数据文件 matched.txt，如果不存在该文件，则返回；

2．如果存在 matched.txt 数据文件，则读取其中数据到 XML Document 中；

3．连接到服务器，向服务器发送 XML 数据；

4．等待服务器响应的结果，响应结果为 XML 形式， 如果响应结果为 200 表示发送成功，则删除 matched.txt 文件；

5．如果返回结果为 500，则表示发送失败，不删除 matched.txt 文件；

6．如果在上述过程中发生异常，也不能删除 matched.txt 文件。

采集客户端向服务器发送的数据格式如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <logs>
    <log>日志对</log>
```

```
    <log>日志对</log>
    <log>日志对</log>
  </logs>
</request>
```

服务器向客户端发送响应的数据格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <code>200</code>
  <message>OK</message>
</response>
```

其中 code 为 200 表示发送成功；code 为 500 表示发送失败。

sendLogs 方法的代码实现如下所示：

```
package com.tarena.dms;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Scanner;
import java.util.Set;

import javax.swing.JFrame;

import org.dom4j.Document;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;

public class DMSClient {
 /** 软件界面 */
 private DMSClientFrame frame;
 /** 二进制日志文件 */
 private File logFile;
 /** 文本日志文件 */
 private File textLogFile;
 /** 存储指针位置文件 */
 private File positionFile;
 private int batch;// 每批读取的数量
 /** 主循环间隔时间 */
 private long interval;
 /** 读取配置文件 */
```

```
private Properties config;
/** 匹配对的日志文件 */
private File matchedFile;
/** 单条登录日志 */
private File loginFile;
/** 服务器 IP */
private String host;
/** 服务器端口号 */
private int port;

public static final int LOG_LENGTH = 372;

private static final short LOGIN = 7;
private static final short LOGOUT = 8;

public DMSClient() {
    ... ...
}

public void action() {
    ... ...
}

public void readLogs() {
    ... ...
}

/** 读取日志方法 */
public String readLog(RandomAccessFile file, int n) throws IOException {
    ... ...
}

private void save(Collection col, File file) throws IOException {
    ... ...
}

private void readLogs(File file, Map<String, Log> logins,
        Map<String, Log> logouts) throws IOException {
    ... ...
}

public void sendLogs() {

    if (!matchedFile.exists()) {
        frame.showSendLogsMsg("没输入数据文件!" + matchedFile);
        return;
    }
    frame.showSendLogsMsg("读取" + matchedFile + "到 XML!");
    Document doc = DocumentHelper.createDocument();
    Element root = doc.addElement("request");
    Element logs = root.addElement("logs");
    BufferedReader in = null;
    try {
        in = new BufferedReader(new InputStreamReader(new FileInputStream(
                matchedFile), "utf-8"));
        String str;
        while ((str = in.readLine()) != null) {
            logs.addElement("log").addText(str);
        }
    } catch (IOException e) {
        frame.showErrMsg(e);
        frame.showSendLogsMsg(e.getMessage());
        return;
```

```
        } finally {
            try {
                if (in != null)
                    in.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        frame.showSendLogsMsg("连接到服务器,发送数据");
        Socket s = null;
        try {
            s = new Socket(host, port);
            OutputStream out = s.getOutputStream();
            NetService.send(doc, out);
            InputStream netIn = s.getInputStream();
            Document response = NetService.receive(netIn);
            root = response.getRootElement();
            String code = root.elementTextTrim("code");
            String msg = root.elementTextTrim("message");
            frame.showSendLogsMsg(msg);
            if (code.equals("200")) {
                matchedFile.delete();
            }
            frame.showSendLogsMsg("结束发送!");
        } catch (Exception e) {
            frame.showErrMsg(e);
            frame.showSendLogsMsg(e.getMessage());
        } finally {
            try {
                if (s != null)
                    s.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

    }
    public static void main(String[] args) {
        DMSClient client = new DMSClient();
        client.action();// 启动方法
    }
}
```

步骤十五：构建服务器端界面

服务器端界面与采集客户端界面类似，代码如下所示：

```
package com.tarena.dms;

import java.awt.GridLayout;
import java.io.PrintWriter;
import java.io.StringWriter;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.TitledBorder;

public class DMSServerFrame extends JFrame {
    private JPanel root;
    private JTextArea netLogArea;
```

```
private JTextArea clientLogArea;
private JTextArea saveLogArea;
private JTextArea errLogArea;
public DMSServerFrame() {
    //初始化界面
    setTitle("DMS Server");
    setSize(800, 600);
    setLocationRelativeTo(null);
    root = new JPanel(new GridLayout(2,2));
    add(root);
    netLogArea = add("网络连接日志");
    clientLogArea = add("客户服务日志");
    saveLogArea = add("保存日志");
    errLogArea = add("异常消息");
}

/**
 * 生产一个显示区域
 * 在当前 root 面板上添加一个显示消息的区域
 * @param title 是边框的标题
 */
public JTextArea add(String title){
    JTextArea area = new JTextArea();
    //JTextArea 必须放置在 JScrollPane 中
    JScrollPane pane = new JScrollPane(area);
    //将滚动面板添加到 窗口的主面板上
    root.add(pane);
    //给滚动面板增加 带有标题的边框
    pane.setBorder(new TitledBorder(title));
    //禁止 area 编辑功能
    area.setEditable(false);
    return area;
}

public static void main(String[] args) {
    //界面的测试方法，不是软件启动方法
    DMSServerFrame frame =
        new DMSServerFrame();
    frame.setDefaultCloseOperation(
            EXIT_ON_CLOSE);
    frame.setVisible(true);
}
// 在 DMSServerFrame 中添加
/** 显示网络连接消息 */
public void showNetMsg(String msg){
    this.netLogArea.append(msg+"\n");
}
/** 接收数据情况的消息 */
public void showClientMsg(String msg){
    this.clientLogArea.append(msg+"\n");
}
/** 保存文件的消息 */
public void showSaveMsg(String msg){
    this.saveLogArea.append(msg+"\n");
}
/** 接收错误消息 */
public void showErrMsg(Exception e) {
    StringWriter buf = new StringWriter();
    PrintWriter out=new PrintWriter(buf);
```

```
            e.printStackTrace(out);
            out.close();
            String msg = buf.toString();
            this.errLogArea.append(msg+"\n");
        }
    }
```

步骤十六：构建服务器端实现框架

新建 DMSServer 类，在该类中添加属性、方法以及内部类，如下所示：

```
package com.tarena.dms;

import java.io.File;
import java.net.ServerSocket;
import java.util.Properties;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ExecutorService;


public class DMSServer {
    /** 读取服务端配置文件的 */
    private Properties config;
    /** Tcp 服务监听端口 */
    private int port;
    /** 客户服务线程池线程数量 */
    private int poolSize;
    /** 缓冲队列大小 */
    private int queueSize;
    /** 接收数据的保存文件 */
    private File serverLogFile;
    /** 客户端服务线程池 */
    private ExecutorService threadPool;
    /** 数据接收缓冲队列 */
    private BlockingQueue<String> queue;
    /** 服务端服务端口 */
    private ServerSocket ss;
    /** 图形界面 */
    private DMSServerFrame frame;

    /** 网络监听线程引用 */
    private Listener listener;
    /** 写文件线程引用 */
    private LogFileWriter fileWriter;
    public DMSServer() {
    }

    /** 网络服务监听线程 */
    class Listener {
    }
    /** 客户服务线程，接收日志写入到缓冲队列 */
    class LogReceiver{
    }
    /** 缓冲队列写出到文件的线程*/
    class LogFileWriter {
    }
    /** 软件启动方法 */
    public void action(){
```

```
    }
    public static void main(String[] args) {
        DMSServer server = new DMSServer();
        server.action();
    }
}
```

步骤十七：架构建服务器端的属性文件

新建属性文件 server.properties，该文件的内容如下所示：

```
# server.properties
server.port=8899
server.log.file=server.txt
pool.size=200
queue.size=1000
```

步骤十八：对 DMSServer 类的属性进行初始化

在 DMSDMSServer 类的构造方法中，初始化属性，代码如图-35 所示：

```
public DMSServer() {
    frame = new DMSServerFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocationRelativeTo(null);
    config = new Properties();
    try {
        config.load(new FileInputStream("server.properties"));
    } catch (Exception e) {
        e.printStackTrace();
        frame.showErrMsg(e);
    }
    port = Integer.parseInt(config.getProperty("server.port"));
    poolSize = Integer.parseInt(config.getProperty("pool.size"));
    queueSize = Integer.parseInt(config.getProperty("queue.size"));
    serverLogFile = new File(config.getProperty("server.log.file"));
    threadPool = Executors.newFixedThreadPool(poolSize);
    queue = new LinkedBlockingQueue<String>(queueSize);
}
```

图 – 35

步骤十九：实现客户服务线程，接收日志写入到缓冲队列

实现客户服务线程 LogReceiver，该线程用于接收日志信息写入缓冲队列，其流程如下：

1．从输入流接收客户端提交的 XML 数据；

2．将日志信息写入到集合 List 中；

3．将 List 集合中的日志存放到缓冲队列中；

4．如果日志信息能正常存到缓冲队列中，则向客户端返回信息“200”、“OK”；

5．如果缓冲队列满了，5 秒内数据没有成功插入，则将所有 List 中的数据撤回并反馈信息为“500”、“Error”；

6．创建反馈的 XML 信息，并以 XML 形式发送反馈信息；

7．关闭 socket，断开连接。

LogReceiver 类的代码如下所示：

```
/** 客户服务线程，接收日志写入到缓冲队列 */
class LogReceiver implements Runnable {
    Socket socket;
    public LogReceiver(Socket socket) {
        this.socket = socket;
    }
    public void run() {
        try {
            frame.showClientMsg("从 in 接收客户端提交的 xml");
            InputStream in = socket.getInputStream();
            Document request = NetService.receive(in);
            frame.showClientMsg(request.asXML());

            frame.showClientMsg("xml 存储到 list");
            List<String> list = new ArrayList<String>();
            String xpath = "/request/logs/log";
            List<Element> elements = request.selectNodes(xpath);
            for (Element e : elements) {
                String log = e.getTextTrim();
                list.add(log);
            }
            frame.showClientMsg("将每个 log 插入到缓冲队列中");
            String code = "200";
            String msg = "OK";
            for (String log : list) {
                // 将 log 向 queue 插入，如果 5 秒之内
                // 还没有插入成功，返回 false
                boolean added = queue.offer(log, 5, TimeUnit.SECONDS);
                if (!added) {
                    frame.showClientMsg("队列满插入失败！撤回插入数据");
                    // 从 queue 中删除包含在 list 中的数据
                    queue.removeAll(list);
                    code = "500";
                    msg = "队列满插入失败";
                    break;
                }
            }
            frame.showClientMsg("反馈消息");
            Document response = DocumentHelper.createDocument();
            Element root = response.addElement("response");
            root.addElement("code").addText(code);
            root.addElement("message").addText(msg);
            frame.showClientMsg(response.asXML());
            OutputStream out = socket.getOutputStream();
            frame.showClientMsg("发送 XML");
            NetService.send(response, out);
            socket.close();
        } catch (Exception e) {
            frame.showErrMsg(e);
        }
    }
}
```

步骤二十：实现缓冲队列写出到文件的线程

在内部类 LogFileWriter，实现缓冲队列写出到文件的线程。实现的策略为如果缓冲队列中有数据，打开文件 server.txt 将数据尽快写入该文件，直到队列为空则关闭文件，

进入等待接收数据的状态，LogFileWriter 类的实现代码如下所示：

```
class LogFileWriter extends Thread{
    public void run() {
        while(true){
            try{
                frame.showSaveMsg("等待数据...");
                String log = queue.take();
                frame.showSaveMsg("等到数据"+log);
                PrintWriter out = new PrintWriter(
                    new  FileWriter(serverLogFile,true));
                frame.showSaveMsg("打开文件"+
                        serverLogFile);
                do{
                    frame.showSaveMsg("保存日志:"+log);
                    out.println(log);
                    log=queue.poll();
                }while(log!=null);
                frame.showSaveMsg("关闭文件");
                out.close();
            }catch(Exception e){
                frame.showErrMsg(e);
            }
        }
    }
}
```

步骤二十一：实现网络服务监听线程

在内部类 Listener 中，实现网络服务监听线程。在该线程中，等待客户端的连接，当有客户端进行连接时，则执行接收数据的线程，代码如下所示：

```
/** 网络服务监听线程 */
class Listener extends Thread {
    public void run() {
        while (true) {
            try {
                frame.showNetMsg("等待客户端的连接");
                Socket s = ss.accept();
                frame.showNetMsg("客户端成功连接");
                String ip = s.getInetAddress().toString();
                frame.showNetMsg("客户端 IP:" + ip);
                frame.showNetMsg("创建客户端服务线程");
                LogReceiver receiver = new LogReceiver(s);
                frame.showNetMsg("执行服务线程");
                threadPool.execute(receiver);
            } catch (Exception e) {
                frame.showErrMsg(e);
                e.printStackTrace();
            }
        }
    }
}
```

步骤二十二：实现软件启动方法

在 action 方法中实现软件的启动。首先，显示服务器端界面； 然后启动网络监听线程；最后启动文件写出线程，代码如下所示：

```
/** 软件启动方法 */
public void action() {
    frame.setVisible(true);
    try {
        frame.showNetMsg("绑定端口:" + port);
        // 绑定网络端口
        ss = new ServerSocket(port);
        // 创建并且开始网络端口监听线程
        listener = new Listener();
        listener.start();
        frame.showNetMsg("创建并且开始网，络端口监听线程");
        // 启动文件写出线程
        fileWriter = new LogFileWriter();
        fileWriter.start();
        frame.showNetMsg("启动文件写出线程");
    } catch (Exception e) {
        frame.showErrMsg(e);
        e.printStackTrace();
    }
}
```

步骤二十三：运行

首先，运行服务器端 DMSServer，启动界面如图-36 所示：

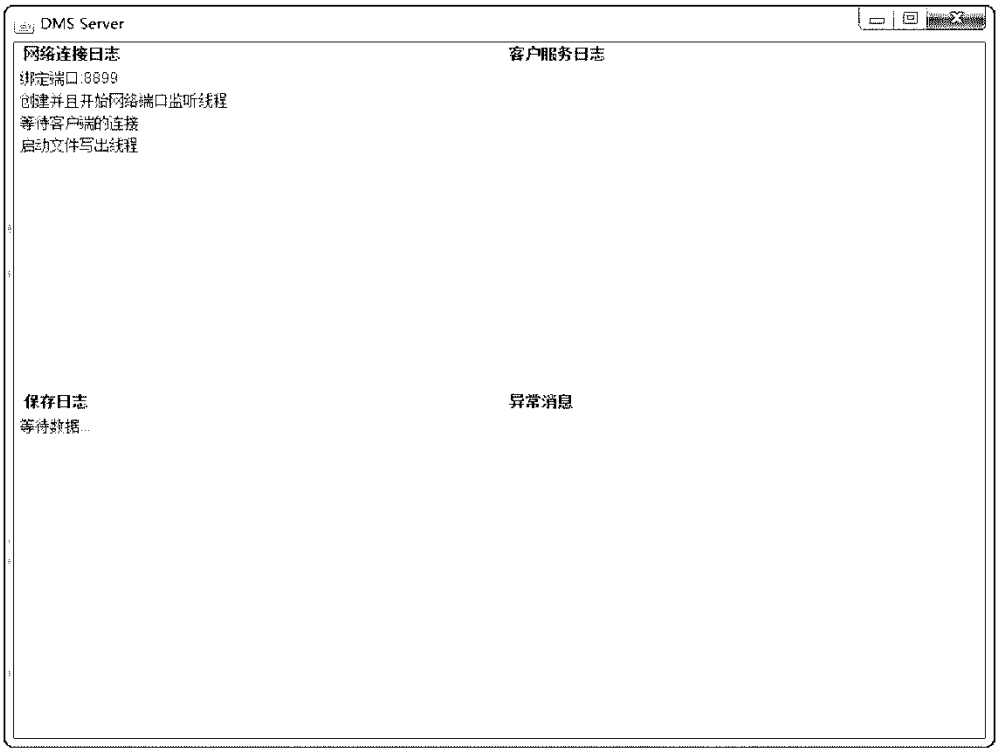

图 - 36

然后，开启一个客户端连接服务器。运行 DMSClient 类，采集客户端界面如图-37 所

示：

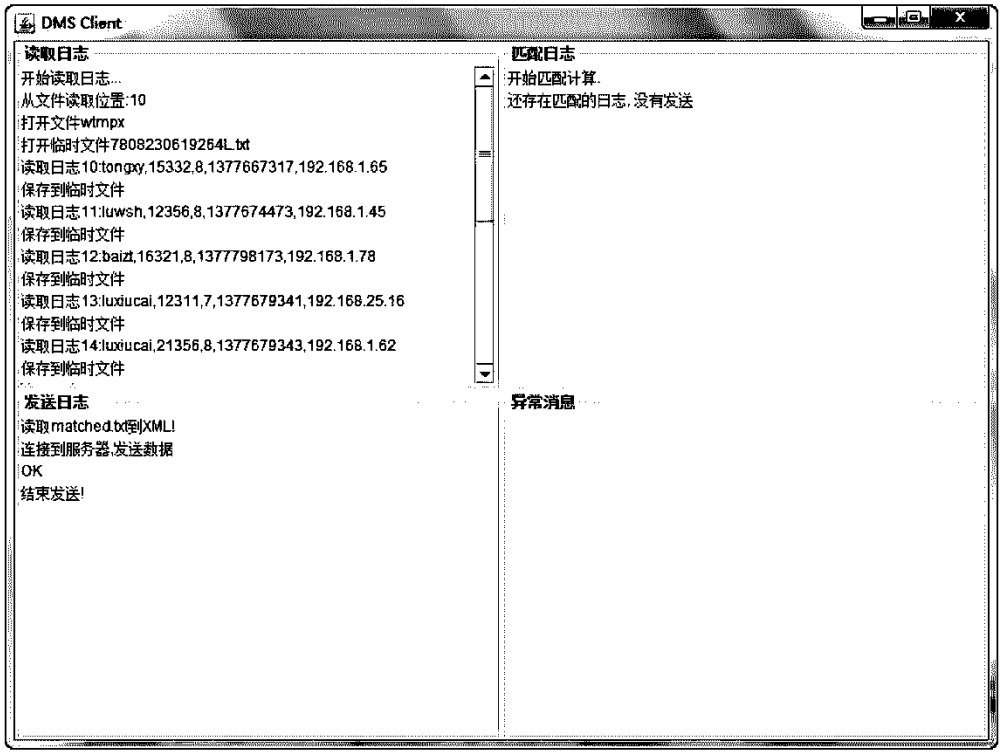

图 - 37

服务器端的界面如图-38 所示：

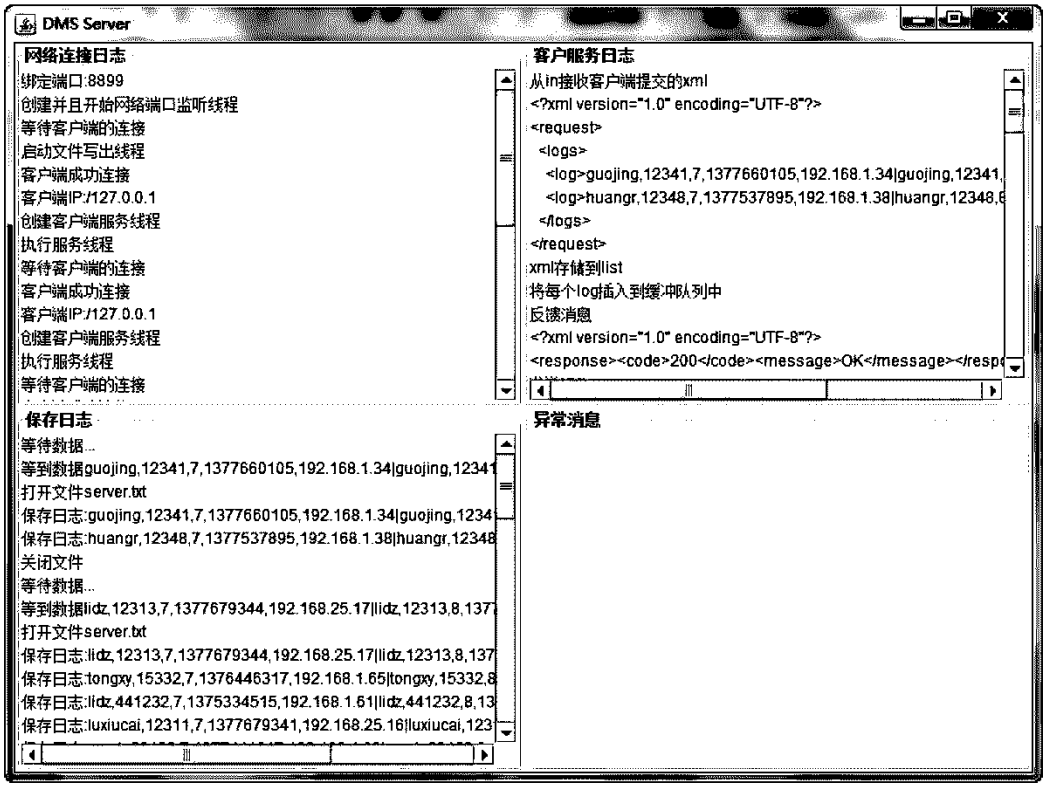

图 - 38

4. 完整代码

本案例中，DMSClient 类的完整代码如下所示：

```
package com.tarena.dms;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Scanner;
import java.util.Set;

import javax.swing.JFrame;

import org.dom4j.Document;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;

public class DMSClient {
 /** 软件界面 */
 private DMSClientFrame frame;
 /** 二进制日志文件 */
 private File logFile;
 /** 文本日志文件 */
 private File textLogFile;
 /** 存储指针位置文件 */
 private File positionFile;
 private int batch;// 每批读取的数量
 /** 主循环间隔时间 */
 private long interval;
 /** 读取配置文件 */
 private Properties config;
 /** 匹配对的日志文件 */
 private File matchedFile;
 /** 单条登录日志 */
 private File loginFile;
 /** 服务器 IP */
 private String host;
 /** 服务器端口号 */
 private int port;

 public static final int LOG LENGTH = 372;

 private static final short LOGIN = 7;
 private static final short LOGOUT = 8;

 public DMSClient() {
     frame = new DMSClientFrame();
```

```
        config = new Properties();
        try {
            FileInputStream in = new FileInputStream("client.properties");
            // 读取配置文件，到内存的散列表
            config.load(in);
        } catch (Exception e) {
            frame.showErrMsg(e);
            // 配置文件读取异常，就不能再执行软件了
            throw new RuntimeException(e);
        }
        // 切记：log.file 这个 key 在配置文件中存在！
        logFile = new File(config.getProperty("log.file"));
        textLogFile = new File(config.getProperty("text.log.file"));
        positionFile = new File(config.getProperty("position.file"));
        batch = Integer.parseInt(config.getProperty("batch"));
        interval = Long.parseLong(config.getProperty("interval"));
        matchedFile = new File(config.getProperty("matched.log.file"));
        loginFile = new File(config.getProperty("login.log.file"));
        host = config.getProperty("server.ip");
        port = Integer.parseInt(config.getProperty("server.port"));
    }

    public void action() {
        // 软件起动时候显示界面.
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        while (true) {
            readLogs();
            matchLogs();
            sendLogs();
            try {
                Thread.sleep(interval);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void readLogs() {
        frame.showReadLogsMsg("开始读取日志...");
        if (!logFile.exists()) {
            // System.out.println(
            // "没有输入日志文件"+logFile);
            // 在界面上显示读取日志的消息
            frame.showReadLogsMsg("没有输入日志文件" + logFile);
            return;
        }
        if (textLogFile.exists()) {
            frame.showReadLogsMsg("输出文件已经存在," + "本次不读取了！");
            return;
        }
        // 读取位置，是起始读取行数
        int position;// 第一行的位置
        //  读取上次的读取位置(没有的从开始位置读取)
        try {
            // positionFile 是文本文件，保存一个整数
            Scanner in = new Scanner(positionFile);
            // "10" -> 10
            position = in.nextInt();
            in.close();
            frame.showReadLogsMsg("从文件读取位置:" + position);
        } catch (FileNotFoundException e) {
            frame.showReadLogsMsg("从 0 读取");
            position = 0;
        }
```

```
        //  根据上次的读取位置和文件长度判断, 是否有新数据
        if (position * LOG LENGTH == logFile.length()) {
            frame.showReadLogsMsg("没有新 log 产生" + "本次读取结束");
            return;
        }
        //  从上次读取位置开始, 批量读取到 临时文件
        // 如果读取的文件末尾就不读取了
        RandomAccessFile in = null;
        PrintWriter out = null;
        try {
            in = new RandomAccessFile(logFile, "r");
            frame.showReadLogsMsg("打开文件" + logFile);
            File temp = new File(System.nanoTime() + "L.txt");
            out = new PrintWriter(temp, "utf-8");
            frame.showReadLogsMsg("打开临时文件" + temp);
            int i; // 10 //position = 0
            for (i = 0; i < batch; i++) {
                int n = i + position;
                String log = readLog(in, n);
                frame.showReadLogsMsg("读取日志" + n + ":" + log);
                if (log == null) {
                    break;
                }
                out.println(log); // 保存临时文件
                frame.showReadLogsMsg("保存到临时文件");
            } // ? i=10
            position = position + i;// 10
            frame.showReadLogsMsg("下次读取位置" + position);
            out.close();// 文件关闭以后才能改名
            // 将临时文件改名到 textLogFile
            if (temp.renameTo(textLogFile)) {
                frame.showReadLogsMsg("临时文件改名:" + textLogFile);
                // 保存下次读取位置
                PrintWriter pw = new PrintWriter(positionFile);
                pw.println(position);
                pw.close();
                frame.showReadLogsMsg("保存下次读取位置:" + positionFile);
            } else {
                frame.showReadLogsMsg("临时文件改名失败!");
            }
        } catch (IOException e) {
            e.printStackTrace();
            frame.showReadLogsMsg("读取日志失败!" + e);
            frame.showErrMsg(e);
            return;
        } finally {
            try {
                if (in != null)
                    in.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        frame.showReadLogsMsg("结束读取!");
    }

    /** 读取日志方法 */
    public String readLog(RandomAccessFile file, int n) throws IOException {
        long start = (long) LOG LENGTH * n;
        // 读取登录用户名 0
        file.seek(start + 0);
        byte[] buf = new byte[32];
        int c = file.read(buf);
        // 检测是否读取到文件的末尾
        if (c == -1) {
```

```
            return null;
        }
        String name = new String(buf, "iso8859-1").trim();
        // 读取登录进程号码 68
        file.seek(start + 68);
        int pid = file.readInt();
        // 读取登录类型 72
        file.seek(start + 72);
        short type = file.readShort();
        // 读取时间
        file.seek(start + 80);
        int time = file.readInt();
        // 读取 IP
        file.seek(start + 114);
        buf = new byte[258];
        file.read(buf);
        String ip = new String(buf, "iso8859-1").trim();
        return name + "," + pid + "," + type + "," + time + "," + ip;
    }

    public void matchLogs() {
        frame.showMatchLogsMsg("开始匹配计算.");
        if (!textLogFile.exists()) {
            frame.showMatchLogsMsg("没有文件:" + textLogFile);
            return;
        }
        if (matchedFile.exists()) {
            frame.showMatchLogsMsg("还存在匹配的日志, 没有发送");
            return;
        }
        // 读取日志
        Map<String, Log> logins = new HashMap<String, Log>();
        Map<String, Log> logouts = new HashMap<String, Log>();
        try {
            readLogs(loginFile, logins, logouts);
            readLogs(textLogFile, logins, logouts);
        } catch (IOException e) {
            frame.showErrMsg(e);
            return;
        }
        List<LogPair> matched = new ArrayList<LogPair>();
        Set<String> keySet = logouts.keySet();
        for (String key : keySet) {
            frame.showMatchLogsMsg("key:" + key);
            Log login = logins.remove(key);
            if (login != null) {
                Log logout = logouts.get(key);
                frame.showMatchLogsMsg("fond:" + login + "," + logout);
                matched.add(new LogPair(login, logout));
            }
        }
        // 匹配结束保存文件
        File tempL = new File(System.nanoTime() + "L.txt");// logins
        File tempM = new File(System.nanoTime() + "m.txt");// matched
        try {
            save(matched, tempM);
            save(logins.values(), tempL);
            frame.showMatchLogsMsg("保存文件成功了");
            if (tempM.renameTo(matchedFile)) {
                loginFile.delete();
                if (tempL.renameTo(loginFile)) {
                    textLogFile.delete();
                    frame.showMatchLogsMsg("删除:" + textLogFile);
                    frame.showMatchLogsMsg("结束匹配.");
                }
            }
```

```
        } catch (Exception e) {
            frame.showErrMsg(e);
            return;
        }
    }

    private void save(Collection col, File file) throws IOException {
        PrintWriter out = new PrintWriter(file);
        for (Object obj : col) {
            out.println(obj);// obj.toString()
        }
        out.close();
    }

    private void readLogs(File file, Map<String, Log> logins,
            Map<String, Log> logouts) throws IOException {
        if (!file.exists()) {
            return;
        }
        BufferedReader in = new BufferedReader(new InputStreamReader(
                new BufferedInputStream(new FileInputStream(file))));
        String str;
        while ((str = in.readLine()) != null) {
            Log log = new Log(str);
            String key = log.getUser() + "," + log.getPid() + "," + log.getIp();
            if (log.getType() == LOGIN) {
                logins.put(key, log);
            } else if (log.getType() == LOGOUT) {
                logouts.put(key, log);
            }
        }
        in.close();
    }

    public void sendLogs() {
        if (!matchedFile.exists()) {
            frame.showSendLogsMsg("没输入数据文件!" + matchedFile);
            return;
        }
        frame.showSendLogsMsg("读取" + matchedFile + "到XML!");
        Document doc = DocumentHelper.createDocument();
        Element root = doc.addElement("request");
        Element logs = root.addElement("logs");
        BufferedReader in = null;
        try {
            in = new BufferedReader(new InputStreamReader(new FileInputStream(
                    matchedFile), "utf-8"));
            String str;
            while ((str = in.readLine()) != null) {
                logs.addElement("log").addText(str);
            }
        } catch (IOException e) {
            frame.showErrMsg(e);
            frame.showSendLogsMsg(e.getMessage());
            return;
        } finally {
            try {
                if (in != null)
                    in.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        frame.showSendLogsMsg("连接到服务器,发送数据");
        Socket s = null;
        try {
            s = new Socket(host, port);
```

```
            OutputStream out = s.getOutputStream();
            NetService.send(doc, out);
            InputStream netIn = s.getInputStream();
            Document response = NetService.receive(netIn);
            root = response.getRootElement();
            String code = root.elementTextTrim("code");
            String msg = root.elementTextTrim("message");
            frame.showSendLogsMsg(msg);
            if (code.equals("200")) {
                matchedFile.delete();
            }
            frame.showSendLogsMsg("结束发送!");
        } catch (Exception e) {
            frame.showErrMsg(e);
            frame.showSendLogsMsg(e.getMessage());
        } finally {
            try {
                if (s != null)
                    s.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args) {
        DMSClient client = new DMSClient();
        client.action();// 启动方法
    }
}
```

DMSClientFrame 类的完整代码如下所示：

```
package com.tarena.dms;

import java.awt.GridLayout;
import java.io.PrintWriter;
import java.io.StringWriter;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.TitledBorder;

/**
 * DMS 客户端界面
 *
 */
public class DMSClientFrame extends JFrame {
    private JPanel root;
    private JTextArea readLogArea;
    private JTextArea matchLogArea;
    private JTextArea sendLogArea;
    private JTextArea errLogArea;

    public DMSClientFrame() {
        // 初始化界面
        setTitle("DMS Client");
        setSize(800, 600);
        setLocationRelativeTo(null);
        // 控件的大小和位置成为控件的布局位置
        // GridLayout 表格布局，将面板设置为
        // 等宽等高的格子，如：(2,2) 两行两列 4 个
        // 区域，添加控件，控件的大小和位置就
```

```
        // 自动的出现在这个区域
        root = new JPanel(new GridLayout(2, 2));
        add(root);
        readLogArea = add("读取日志");
        matchLogArea = add("匹配日志");
        sendLogArea = add("发送日志");
        errLogArea = add("异常消息");
    }

    /**
     * 生产一个显示区域 在当前 root 面板上添加一个显示消息的区域
     *
     * @param title
     *            是边框的标题
     */
    public JTextArea add(String title) {
        JTextArea area = new JTextArea();
        // JTextArea 必须放置在 JScrollPane 中
        JScrollPane pane = new JScrollPane(area);
        // 将滚动面板添加到 窗口的主面板上
        root.add(pane);
        // 给滚动面板增加 带有标题的边框
        pane.setBorder(new TitledBorder(title));
        // 禁止 area 编辑功能
        area.setEditable(false);
        return area;
    }
    /** 在 DMSClientFrame 上添加方法, 显示
     * 读取日志的消息 */
    public void showReadLogsMsg(String msg) {
        //在这个界面的 readLogArea 区域增加消息
        this.readLogArea.append(msg+"\n");
        //Area 区域
    }

    /** 显示匹配日志消息 */
    public void showMatchLogsMsg(String msg){
        this.matchLogArea.append(msg+"\n");
    }
    /** 显示发送日志消息 */
    public void showSendLogsMsg(String msg){
        this.sendLogArea.append(msg+"\n");
    }
    /** 在界面显示异常消息 */
    public void showErrMsg(Exception e) {
        //StringWriter == StringBuilder
        //  writer(char)   append(char)
        //建立一个字符串缓冲区
        //StringWriter 是字符节点流, 是字符缓冲区
        StringWriter buf = new StringWriter();
        PrintWriter out = new PrintWriter(buf);
        //将异常跟踪消息写到缓冲区
        e.printStackTrace(out);//PrintWriter
        out.close();
        //读取缓冲区中的数据, 在界面显示出来
        String msg = buf.toString();
        this.errLogArea.append(msg+"\n");
    }

    public static void main(String[] args) {
        // 界面的测试方法, 不是软件启动方法
        DMSClientFrame frame = new DMSClientFrame();
        frame.setDefaultCloseOperation(EXIT ON CLOSE);
        frame.setVisible(true);
```

```
    }
}
```

DMSServer 类的完整代码如下所示：

```
package com.tarena.dms;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;

import javax.swing.JFrame;

import org.dom4j.Document;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;

public class DMSServer {
 /** 读取服务端配置文件的 */
 private Properties config;
 /** Tcp 服务监听端口 */
 private int port;
 /** 客户服务线程池线程数量 */
 private int poolSize;
 /** 缓冲队列大小 */
 private int queueSize;
 /** 接收数据的保存文件 */
 private File serverLogFile;
 /** 客户端服务线程池 */
 private ExecutorService threadPool;
 /** 数据接收缓冲队列 */
 private BlockingQueue<String> queue;
 /** 服务端服务端口 */
 private ServerSocket ss;
 /** 图形界面 */
 private DMSServerFrame frame;

 /** 网络监听线程引用 */
 private Listener listener;
 /** 写文件线程引用 */
 private LogFileWriter fileWriter;

 public DMSServer() {
     frame = new DMSServerFrame();
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setLocationRelativeTo(null);
     config = new Properties();
     try {
         config.load(new FileInputStream("server.properties"));
     } catch (Exception e) {
         e.printStackTrace();
```

```
            frame.showErrMsg(e);
        }
        port = Integer.parseInt(config.getProperty("server.port"));
        poolSize = Integer.parseInt(config.getProperty("pool.size"));
        queueSize = Integer.parseInt(config.getProperty("queue.size"));
        serverLogFile = new File(config.getProperty("server.log.file"));
        threadPool = Executors.newFixedThreadPool(poolSize);
        queue = new LinkedBlockingQueue<String>(queueSize);
    }

    /** 网络服务监听线程 */
    class Listener extends Thread {
        public void run() {
            while (true) {
                try {
                    frame.showNetMsg("等待客户端的连接");
                    Socket s = ss.accept();
                    frame.showNetMsg("客户端成功连接");
                    String ip = s.getInetAddress().toString();
                    frame.showNetMsg("客户端 IP:" + ip);
                    frame.showNetMsg("创建客户端服务线程");
                    LogReceiver receiver = new LogReceiver(s);
                    frame.showNetMsg("执行服务线程");
                    threadPool.execute(receiver);
                } catch (Exception e) {
                    frame.showErrMsg(e);
                    e.printStackTrace();
                }
            }
        }
    }

    /** 客户服务线程，接收日志写入到缓冲队列 */
    class LogReceiver implements Runnable {
        Socket socket;

        public LogReceiver(Socket socket) {
            this.socket = socket;
        }

        public void run() {
            try {
                frame.showClientMsg("从 in 接收客户端提交的 xml");
                InputStream in = socket.getInputStream();
                Document request = NetService.receive(in);
                frame.showClientMsg(request.asXML());

                frame.showClientMsg("xml 存储到 list");
                List<String> list = new ArrayList<String>();
                String xpath = "/request/logs/log";
                List<Element> elements = request.selectNodes(xpath);
                for (Element e : elements) {
                    String log = e.getTextTrim();
                    list.add(log);
                }
                frame.showClientMsg("将每个 log 插入到缓冲队列中");
                String code = "200";
                String msg = "OK";
                for (String log : list) {
                    // 将 log 向 queue 插入，如果 5 秒之内
                    // 还没有插入成功，返回 false
                    boolean added = queue.offer(log, 5, TimeUnit.SECONDS);
                    if (!added) {
                        frame.showClientMsg("队列满,插入失败！撤回插入数据");
                        // 从 queue 中删除包含在 list 中的数据
                        queue.removeAll(list);
```

```
                    code = "500";
                    msg = "队列满插入失败";
                    break;
                }
            }
            frame.showClientMsg("反馈消息");
            Document response = DocumentHelper.createDocument();
            Element root = response.addElement("response");
            root.addElement("code").addText(code);
            root.addElement("message").addText(msg);
            frame.showClientMsg(response.asXML());
            OutputStream out = socket.getOutputStream();
            frame.showClientMsg("发送XML");
            NetService.send(response, out);
            socket.close();
        } catch (Exception e) {
            frame.showErrMsg(e);
        }
    }
}

/** 缓冲队列写出到文件的线程 */
class LogFileWriter extends Thread {
    public void run() {
        while (true) {
            try {
                frame.showSaveMsg("等待数据...");
                String log = queue.take();
                frame.showSaveMsg("等到数据" + log);
                PrintWriter out = new PrintWriter(new FileWriter(
                        serverLogFile, true));
                frame.showSaveMsg("打开文件" + serverLogFile);
                do {
                    frame.showSaveMsg("保存日志:" + log);
                    out.println(log);
                    log = queue.poll();
                } while (log != null);
                frame.showSaveMsg("关闭文件");
                out.close();
            } catch (Exception e) {
                frame.showErrMsg(e);
            }
        }
    }
}

/** 软件启动方法 */
public void action() {
    frame.setVisible(true);
    try {
        frame.showNetMsg("绑定端口:" + port);
        // 绑定网络端口
        ss = new ServerSocket(port);
        // 创建并且开始网络端口监听线程
        listener = new Listener();
        listener.start();
        frame.showNetMsg("创建并且开始网络端口监听线程");
        // 启动文件写出线程
        fileWriter = new LogFileWriter();
        fileWriter.start();
        frame.showNetMsg("启动文件写出线程");
    } catch (Exception e) {
        frame.showErrMsg(e);
        e.printStackTrace();
    }
```

```
    }

    public static void main(String[] args) {
        DMSServer server = new DMSServer();
        server.action();
    }
}
```

DMSServerFrame 类的完整代码如下所示：

```
package com.tarena.dms;

import java.awt.GridLayout;
import java.io.PrintWriter;
import java.io.StringWriter;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.TitledBorder;

public class DMSServerFrame extends JFrame {
    private JPanel root;
    private JTextArea netLogArea;
    private JTextArea clientLogArea;
    private JTextArea saveLogArea;
    private JTextArea errLogArea;
    public DMSServerFrame() {
        //初始化界面
        setTitle("DMS Server");
        setSize(800, 600);
        setLocationRelativeTo(null);
        root = new JPanel(new GridLayout(2,2));
        add(root);
        netLogArea = add("网络连接日志");
        clientLogArea = add("客户服务日志");
        saveLogArea = add("保存日志");
        errLogArea = add("异常消息");
    }

    /**
     * 生产一个显示区域
     * 在当前 root 面板上添加一个显示消息的区域
     * @param title 是边框的标题
     */
    public JTextArea add(String title){
        JTextArea area = new JTextArea();
        //JTextArea 必须放置在 JScrollPane 中
        JScrollPane pane = new JScrollPane(area);
        //将滚动面板添加到 窗口的主面板上
        root.add(pane);
        //给滚动面板增加 带有标题的边框
        pane.setBorder(new TitledBorder(title));
        //禁止 area 编辑功能
        area.setEditable(false);
        return area;
    }

    public static void main(String[] args) {
        //界面的测试方法，不是软件启动方法
        DMSServerFrame frame =
            new DMSServerFrame();
        frame.setDefaultCloseOperation(
```

```
            EXIT ON CLOSE);
        frame.setVisible(true);
    }
    // 在 DMSServerFrame 中添加
    /** 显示网络连接消息 */
    public void showNetMsg(String msg){
        this.netLogArea.append(msg+"\n");
    }
    /** 接收数据情况的消息 */
    public void showClientMsg(String msg){
        this.clientLogArea.append(msg+"\n");
    }
    /** 保存文件的消息 */
    public void showSaveMsg(String msg){
        this.saveLogArea.append(msg+"\n");
    }
    /** 接收错误消息 */
    public void showErrMsg(Exception e) {
        StringWriter buf = new StringWriter();
        PrintWriter out=new PrintWriter(buf);
        e.printStackTrace(out);
        out.close();
        String msg = buf.toString();
        this.errLogArea.append(msg+"\n");
    }
}
```

Log 类的完整代码如下所示：

```
package com.tarena.dms;

import java.io.Serializable;

public class Log implements Serializable {

    private String user;
    private int pid;
    private short type;
    private int time;
    private String ip;

    public Log() {
    }

    public Log(String user, int pid, short type, int time, String ip) {
        super();
        this.user = user;
        this.pid = pid;
        this.type = type;
        this.time = time;
        this.ip = ip;
    }
    /**
     * str="lidz,441232,7,1375334515,192.168.1.61"
     */
    public Log(String str){
        String[] data=str.split(",");
        user = data[0];
        pid = Integer.parseInt(data[1]);
        type = Short.parseShort(data[2]);
        time = Integer.parseInt(data[3]);
        ip = data[4];
    }
    public String toString() {
        return user+","+pid+","+type+","+time+","+ip;
```

```
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public int getPid() {
        return pid;
    }

    public void setPid(int pid) {
        this.pid = pid;
    }

    public short getType() {
        return type;
    }

    public void setType(short type) {
        this.type = type;
    }

    public int getTime() {
        return time;
    }

    public void setTime(int time) {
        this.time = time;
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }

}
```

LogPair类的完整代码如下所示：

```
package com.tarena.dms;

import java.io.Serializable;
/**
 * 登录日志对
 */
public class LogPair implements Serializable {
 private Log login;
 private Log logout;

 public LogPair() {
 }

 public LogPair(Log login, Log logout) {
     this.login = login;
     this.logout = logout;
 }
 public LogPair(String str){
```

```
    //str="user,pid,7,time,ip|user,pid,8,time,ip"
        String[] data = str.split("\\|");
        login = new Log(data[0]);
        logout = new Log(data[1]);
    }
    @Override
    public String toString() {
        //user,pid,7,time,ip|user,pid,8,time,ip
        return login+"|"+logout;
        //login.toString()+"|"+logout.toString();
    }

    public Log getLogin() {
        return login;
    }

    public void setLogin(Log login) {
        this.login = login;
    }

    public Log getLogout() {
        return logout;
    }

    public void setLogout(Log logout) {
        this.logout = logout;
    }

    }
```

NetService类的完整代码如下所示：

```
    package com.tarena.dms;

    import java.io.ByteArrayInputStream;
    import java.io.ByteArrayOutputStream;
    import java.io.IOException;
    import java.io.InputStream;
    import java.io.OutputStream;
    import java.util.Arrays;

    import org.dom4j.Document;
    import org.dom4j.DocumentException;
    import org.dom4j.io.OutputFormat;
    import org.dom4j.io.SAXReader;
    import org.dom4j.io.XMLWriter;

    public class NetService {
     /**
      * 将 doc 对象通过网络流 out 发送到服务器
      *
      * @param doc
      *          xml 对象
      * @param out
      *          网络输出流
      */
    public static void send(Document doc, OutputStream out) throws IOException
{
        // 先将 doc 转换为 bytes
        ByteArrayOutputStream buf = new ByteArrayOutputStream();
        OutputFormat fmt = OutputFormat.createPrettyPrint();
        XMLWriter writer = new XMLWriter(buf, fmt);
        writer.write(doc);
        byte[] bytes = buf.toByteArray();
```

```
        // 发送 50 个 bytes: type,length
        String str = "XML," + bytes.length;
        byte[] tl = str.getBytes("utf-8");
        tl = Arrays.copyOf(tl, 50);
        out.write(tl);
        // 发送 bytes
        out.write(bytes);
        out.flush();
    }

    /**
     * 从流 in 中接收 XML Document
     *
     * @param in
     *          输入流
     * @return XML Document
     */
    public static Document receive(InputStream in) throws IOException,
            DocumentException {
        // 先接收 header (type + length)
        byte[] bytes = new byte[50];
        in.read(bytes);
        // 解析 Header 得到 length
        String header = new String(bytes, "utf-8").trim();
        String[] data = header.split(",");
        String type = data[0];
        if (!"XML".equals(type)) {
            throw new DocumentException("不是 XML 文件格式");
        }
        int length = Integer.parseInt(data[1]);
        // 再根据长度接收 bytes
        bytes = new byte[length];
        in.read(bytes);
        // 最后将 bytes 解析为 Document
        SAXReader reader = new SAXReader();
        Document doc = reader.read(new ByteArrayInputStream(bytes));
        return doc;
    }
}
```

TestCase类的完整代码如下所示：

```
package com.tarena.dms;

import java.io.FileInputStream;
import java.io.FileOutputStream;

import org.dom4j.Document;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;
import org.junit.Test;

public class TestCase {
    @Test
    public void testReadLogs() {
        // 对日志读取方法进行单独测试.
        DMSClient client = new DMSClient();
        client.readLogs();
    }

    @Test
    public void testMatchLogs() {
        // 对日志读取方法进行单独测试.
```

```
        DMSClient client = new DMSClient();
        client.matchLogs();
    }

    @Test
    public void testSend() throws Exception {
        // 利用文件流对网络协议进行测试
        Document doc = DocumentHelper.createDocument();
        Element root = doc.addElement("test");
        root.addText("Hello World!");
        // 使用文件流 替换网络流
        FileOutputStream out = new FileOutputStream("demo.txt");
        NetService.send(doc, out);
        out.close();

        FileInputStream in = new FileInputStream("demo.txt");
        Document doc2 = NetService.receive(in);
        in.close();
        System.out.println(doc2.asXML());
    }
}
```

client.properties 文件的完整内容如下所示：

```
# client.properties
server.ip=localhost
# server.ip=192.168.100.45
server.port=8899
log.file=wtmpx
#log.file=/var/adm/wtmpx
position.file=position.txt
text.log.file=log.txt
batch=10
interval=5000
matched.log.file=matched.txt
login.log.file=login.txt
```

server.properties文件的完整内容如下所示：

```
# server.properties
server.port=8899
server.log.file=server.txt
pool.size=200
queue.size=1000
```