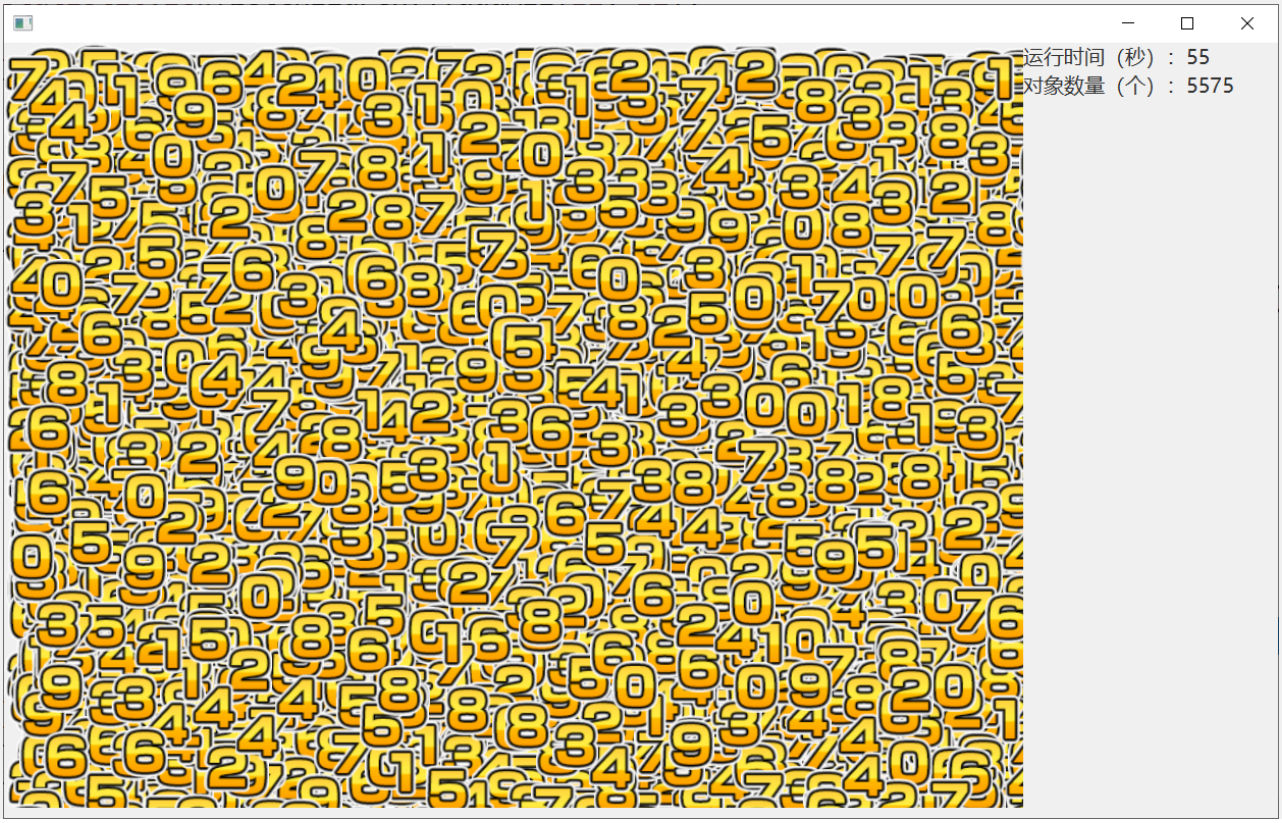
第2章 享元模式

提出问题

> 问题描述：现在要采用图片替代文本的方式显示数字，并将其应用到游戏或视频制作，例如下图将数字动态的铺满屏幕的场景。



定义一个图片样式的数字对象，包括数字、图片和坐标信息。

```
public class ImageNumber {
    private Image image;
    private int number;
    private double x;
    private double y;
}
```

设计GUI客户端创建并显示数字对象：

```
public class MainApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Font f = new Font(16);
        long start = System.currentTimeMillis();

        // GUI布局
        HBox layout = new HBox();
        Canvas canvas = new Canvas(800, 600);
        GraphicsContext gc =  canvas.getGraphicsContext2D();
        VBox statisticView = new VBox();
        Label l1 = new Label("运行时间: ");
        l1.setFont(f);
        Label l2 = new Label("总对象数: ");
        l2.setFont(f);
        statisticView.getChildren().addAll(l1, l2);
        layout.getChildren().addAll(canvas, statisticView);

        // 产生数字并显示
        List<ImageNumber> list = new ArrayList<ImageNumber>();
        Timeline timer = new Timeline(new KeyFrame(Duration.millis(10), e -> {
            l1.setText("运行时间（秒）: " + (System.currentTimeMillis() - start) / 1000);
            l2.setText("对象数量（个）: " + list.size());
            ImageNumber in = createNumber((int)(Math.random() * 10));

            in.setX(800 * Math.random());
            in.setY(600 * Math.random());
            list.add(in);
            gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
            for(ImageNumber n:list) {
                gc.drawImage(n.getImage(), n.getX(), n.getY());
            }
        }));
        timer.setCycleCount(Timeline.INDEFINITE);
        timer.play();
        primaryStage.setScene(new Scene(layout, 1000, 650));
        primaryStage.show();
    }
    // 生产一个图片数字对象
    public static ImageNumber createNumber(int i) {
        Image img = new Image(Paths.get("bin/images/res/res04.png").toUri().toString());
        img = ImageTool.clipImage(img, 36 * i, 0, 36, 40);
        ImageNumber in = new ImageNumber(i, img);
        return in;
    }
```

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

上述程序产生的图片对象会占用大量内存，思考是否存在一种节省内存的设计方式。

模式名称

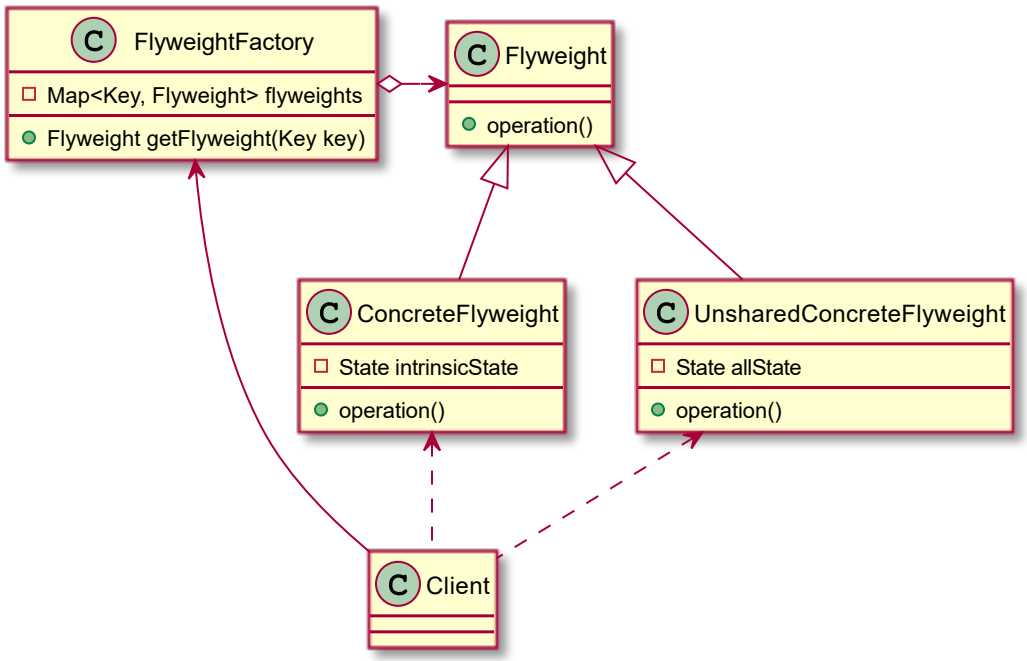
享元模式（Flyweight）即共享轻量级单元。

设计意图

享元模式把一个对象的状态分成内部状态和外部状态，内部状态时不变的，外部状态时变化的。通过共享不变的部分，达到减少对象数量并节约内存的目的。

> Use sharing to support large numbers of fine-grained objects efficiently.

设计结构



参与者

- 抽象享元（Flyweight）：享元对象基类或接口，定义出对象的外部状态和内部状态的接口或实现。
- 具体享元（ConcreteFlyweight）：抽象享元的具体实现。内部状态处理与环境无关。
- 享元工厂（FlyweightFactory）：负责管理享元对象池和创建享元对象。

解决问题

根据享元模式的设计理念，由于`Image`对象创建后状态不变，因此可以充当`Flyweihgt`对象，对于数字图片的生产工厂设计如下：

```
public class ImageNumberFactory {
    private static final ImageNumberFactory instance = new ImageNumberFactory();
    // 图片数字池
    Map<String, Image> numbers = new HashMap<String, Image>();

    private ImageNumberFactory() {}

    public static ImageNumberFactory getInstance() {
        return instance;
    }

    // 获取目标数字对应的图片
    public Image get(String key) {
        if(numbers.containsKey(key)) {
            return numbers.get(key);
        }else {
            int i = Integer.parseInt(key);
            Image img = new Image(Paths.get("bin/images/res/res04.png").toUri().toString());
            img = ImageTool.clipImage(img, 36 * i, 0, 36, 40);
            numbers.put(key, img);
            return img;
        }
    }
}
```

数字对象设计如下，含有一个`Image`对象，获取方式是通过享元工厂。

```
public class Number {
    private Image image;
    private int number;
    private double x;
    private double y;

    public Number(int number) {
        this.setNumber(number);
    }
    public Image getImage() {
        return image;
    }
    public int getNumber() {
        return number;
    }
    // 图片信息通过ImageNumberFactory获取
    public void setNumber(int number) {
        this.number = number;
        this.image = ImageNumberFactory.getInstance().get(String.valueOf(number));
    }
}
```

效果与适用性

优点

- 减少对象创建，降低系统内存消耗，提高效率；
- 减少内存之外的其他资源占用

缺点：

- 对内外状态的处理可能导致额外的时间开销；

适用性

- 系统中存在大量相似对象，需要缓冲池的场景；
- 系统底层开发，解决系统性能问题。

扩展案例

JDK中的享元模式

`Integer`内部采用享元模式，对于`[-128,127]`区间内的整数对象直接从预先准备的缓冲池中获取。例如如下代码：

```
public class Test {
    public static void main(String[] args) {
        System.out.println(Integer.valueOf(-129) == Integer.valueOf(-129));
        System.out.println(Integer.valueOf(-128) == Integer.valueOf(-128));
        System.out.println(Integer.valueOf(127) == Integer.valueOf(127));
        System.out.println(Integer.valueOf(128) == Integer.valueOf(128));
    }
}
```

输出为：

```
false
true
true
false
```

整型对象缓冲池的实现方式如下。

```
public final class Integer extends Number
    implements Comparable<Integer>, Constable, ConstantDesc{
    public static Integer valueOf(int i) {
        if (i >= IntegerCache.low && i <= IntegerCache.high)
            return IntegerCache.cache[i + (-IntegerCache.low)];
        return new Integer(i);
    }
    private static class IntegerCache {
        static final int low = -128;
        static final int high;
        static final Integer[] cache;
        ...
        Integer[] c = new Integer[size];
        int j = low;
        for(int i = 0; i < c.length; i++) {
            c[i] = new Integer(j++);
        }
        ...
    }
}
```