外观模式 代理模式

享元模式

享元模式（Flyweight Pattern）主要用于减少创建对象的数量，以减少内存占用和提高性能。这种类型的设计模式属于结构型模式，它提供了减少对象数量从而改善应用所需的对象结构的方式。

享元模式尝试重用现有的同类对象，如果未找到匹配的对象，则创建新对象。我们将通过创建 5 个对象来画出 20 个分布于不同位置的圆来演示这种模式。由于只有 5 种可用的颜色，所以 color 属性被用来检查现有的 *Circle* 对象。

介绍

意图：运用共享技术有效地支持大量细粒度的对象。

主要解决：在有大量对象时，有可能会造成内存溢出，我们把其中共同的部分抽象出来，如果有相同的业务请求，直接返回在内存中已有的对象，避免重新创建。

何时使用： 1、系统中有大量对象。 2、这些对象消耗大量内存。 3、这些对象的状态大部分可以外部化。 4、这些对象可以按照内蕴状态分为很多组，当把外蕴对象从对象中剔除出来时，每一组对象都可以用一个对象来代替。 5、系统不依赖于这些对象身份，这些对象是不可分辨的。

如何解决：用唯一标识码判断，如果在内存中有，则返回这个唯一标识码所标识的对象。

关键代码：用 HashMap 存储这些对象。

应用实例： 1、JAVA 中的 String，如果有则返回，如果没有则创建一个字符串保存在字符串缓存池里面。 2、数据库的数据池。

优点：大大减少对象的创建，降低系统的内存，使效率提高。

缺点：提高了系统的复杂度，需要分离出外部状态和内部状态，而且外部状态具有固有化的性质，不应该随着内部状态的变化而变化，否则会造成系统的混乱。

使用场景： 1、系统有大量相似对象。 2、需要缓冲池的场景。

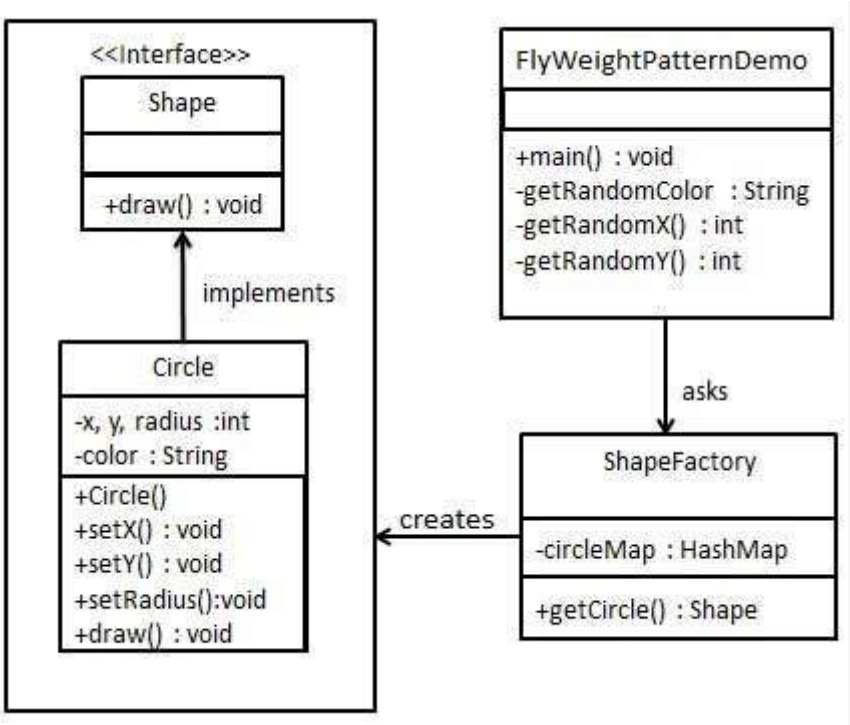
注意事项： 1、注意划分外部状态和内部状态，否则可能会引起线程安全问题。 2、这些类必须有一个工厂对象加以控制。

实现

我们将创建一个 *Shape* 接口和实现了 *Shape* 接口的实体类 *Circle*。下一步是定义工厂类 *ShapeFactory*。

ShapeFactory 有一个 *Circle* 的 *HashMap*，其中键名为 *Circle* 对象的颜色。无论何时接收到请求，都会创建一个特定颜色的圆。*ShapeFactory* 检查它的 *HashMap* 中的 circle 对象，如果找到 *Circle* 对象，则返回该对象，否则将创建一个存储在 hash map 中以备后续使用的新对象，并把该对象返回到客户端。

FlyWeightPatternDemo，我们的演示类使用 *ShapeFactory* 来获取 *Shape* 对象。它将向 *ShapeFactory* 传递信息（*red / green / blue/ black / white*），以便获取它所需对象的颜色。



步骤 1

创建一个接口。

Shape.java

```
public interface Shape {
void draw();
}
```

步骤 2

创建实现接口的实体类。

Circle.java

```
public class Circle implements Shape {
private String color;
private int x;
private int y;
private int radius;
public Circle(String color){
this.color = color;
}
public void setX(int x) {
this.x = x;
}
public void setY(int y) {
this.y = y;
}
public void setRadius(int radius) {
this.radius = radius;
}
@Override
public void draw() {
```

```
System.out.println("Circle: Draw() [Color : " + color
+", x : " + x +", y :" + y +", radius :" + radius);
}
}
```

步骤 3

创建一个工厂，生成基于给定信息的实体类的对象。

ShapeFactory.java

```
import java.util.HashMap;
public class ShapeFactory {
private static final HashMap<String, Shape> circleMap = new HashMap<>();
public static Shape getCircle(String color) {
Circle circle = (Circle)circleMap.get(color);
if(circle == null) {
circle = new Circle(color);
circleMap.put(color, circle);
System.out.println("Creating circle of color : " + color);
}
return circle;
}
}
```

步骤 4

使用该工厂，通过传递颜色信息来获取实体类的对象。

FlyweightPatternDemo.java

```
public class FlyweightPatternDemo {
private static final String colors[] =
{ "Red", "Green", "Blue", "White", "Black" };
public static void main(String[] args) {
for(int i=0; i < 20; ++i) {
Circle circle =
(Circle)ShapeFactory.getCircle(getRandomColor());
circle.setX(getRandomX());
circle.setY(getRandomY());
circle.setRadius(100);
circle.draw();
}
}
private static String getRandomColor() {
return colors[(int)(Math.random()*colors.length)];
}
private static int getRandomX() {
return (int)(Math.random()*100 );
}
private static int getRandomY() {
return (int)(Math.random()*100);
}
}
```

步骤 5

执行程序，输出结果：

```
Creating circle of color : Black
Circle: Draw() [Color : Black, x : 36, y :71, radius :100
Creating circle of color : Green
Circle: Draw() [Color : Green, x : 27, y :27, radius :100
Creating circle of color : White
Circle: Draw() [Color : White, x : 64, y :10, radius :100
Creating circle of color : Red
Circle: Draw() [Color : Red, x : 15, y :44, radius :100
Circle: Draw() [Color : Green, x : 19, y :10, radius :100
Circle: Draw() [Color : Green, x : 94, y :32, radius :100
Circle: Draw() [Color : White, x : 69, y :98, radius :100
Creating circle of color : Blue
Circle: Draw() [Color : Blue, x : 13, y :4, radius :100
Circle: Draw() [Color : Green, x : 21, y :21, radius :100
Circle: Draw() [Color : Blue, x : 55, y :86, radius :100
Circle: Draw() [Color : White, x : 90, y :70, radius :100
Circle: Draw() [Color : Green, x : 78, y :3, radius :100
Circle: Draw() [Color : Green, x : 64, y :89, radius :100
Circle: Draw() [Color : Blue, x : 3, y :91, radius :100
Circle: Draw() [Color : Blue, x : 62, y :82, radius :100
Circle: Draw() [Color : Green, x : 97, y :61, radius :100
Circle: Draw() [Color : Green, x : 86, y :12, radius :100
Circle: Draw() [Color : Green, x : 38, y :93, radius :100
Circle: Draw() [Color : Red, x : 76, y :82, radius :100
Circle: Draw() [Color : Blue, x : 95, y :82, radius :100
```

← 外观模式 代理模式 →

1 篇笔记

写笔记

享元模式，换句话说就是共享对象，在某些对象需要重复创建，且最终只需要得到单一结果的情况下使用。因为此种模式是利用先前创建的已有对象，通过某种规则去判断当前所需对象是否可以利用原有对象做相应修改后得到想要的效果，如以上教程的实例，创建了20个不同效果的圆，但相同颜色的圆只需要创建一次便可，相同颜色的只需要引用原有对象，改变其坐标值便可。此种模式下，同一颜色的圆虽然位置不同，但其地址都是同一个，所以说此模式适用于结果注重单一结果的情况。

举一个简单例子，一个游戏中有不同的英雄角色，同一类型的角色也有不同属性的英雄，如刺客类型的英雄有很多个，按此种模式设计，利用英雄所属类型去引用原有同一类型的英雄实例，然后对其相应属性进行修改，便可得到最终想得到的最新英雄；比如说你创建了第一个刺客型英雄，然后需要设计第二个刺客型英雄，你利用第一个英雄改变属性得到第二个刺客英雄，最新的刺客英雄是诞生了，但第一个刺客英雄的属性也随之变得与第二个相同，这种情况显然是不可以的。

yjb 4个月前 (11-27)