# 设计模式-享元模式


## 参考

- Go设计模式(17)-享元模式*程序员麻辣烫的博客-CSDN博客*go 享元模式
- https://www.runoob.com/design-pattern/flyweight-pattern.html

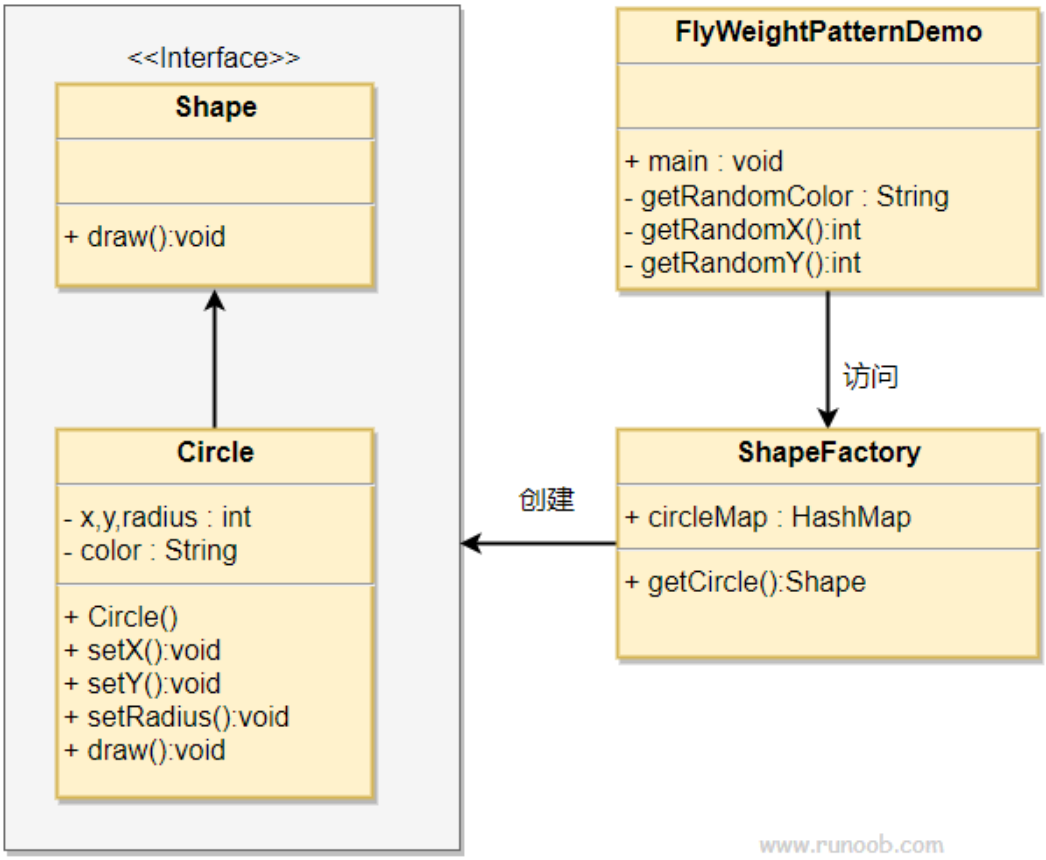
---

享元模式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，它提供来减少对象数量从而改善应用所需的对象结构的方式。

## 使用场景

- 系统有大量相似对象
- 需要缓冲池的场景

## Demo分析




> 分析：
>
> 我们将创建一个 `Shape` 接口实现了 `Shape` 接口的实体类 `Circle`。下一步是定义工厂类 `ShapeFactory`，`ShapeFactor` 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hashmap中以备后续使用的新对象，并把该对象返回到客户端。

## Go实现

```
package main

import (
    "fmt"
    "math/rand"
)

type Shape interface {
    Draw()
}

/**
 * 创建实现接口的实体类
 */
type Circle struct {
    x, y, radius int
    color        string
}

func (c *Circle) Draw() {
```

```
    fmt.Printf("Circle: x=%d, y=%d, radius=%d, color=%s\n", c.x, c.y, c.radius,
c.color)
}

/**
 * 创建一个工厂，生成基于给定信息的实体类的对象
 */
type ShapeFactory struct {
    circleMap map[string]Shape
}

func (f *ShapeFactory) getCircle(name string) *Circle {
    if c, ok := f.circleMap[name]; ok {
        return c.(*Circle)
    } else {
        fmt.Printf("Creating circle of color : %s\n", name)
        circle := &Circle{color: name}
        f.circleMap[name] = circle
        return circle
    }
}

func FlyWeightPatternDemo() {
    colors := []string{"Red", "Green", "Blue", "White", "Black"}
    f := &ShapeFactory{circleMap: make(map[string]Shape)}
    for i := 0; i < 5; i++ {
        cIndex := rand.Intn(len(colors))
        circle := f.getCircle(colors[cIndex])
        circle.Draw()
    }
}

func main() {
    FlyWeightPatternDemo()
}
```

**输出**

```
Creating circle of color : Green
Circle: x=0, y=0, radius=0, color=Green
Creating circle of color : Blue
Circle: x=0, y=0, radius=0, color=Blue
Circle: x=0, y=0, radius=0, color=Blue
Creating circle of color : Black
Circle: x=0, y=0, radius=0, color=Black
Circle: x=0, y=0, radius=0, color=Green
```

# Python实现

```
import random

class Shape():
  """
    ## 抽象类
```

```
  """
  def draw(self):
    print("draw")

class Circle(Shape):
  """
    ## 实现接口的实体类
  """
  def __init__(self, x, y, radius, color):
    self.x = x
    self.y = y
    self.radius = radius
    self.color = color
  def draw(self):
    print(f"draw circle at {self.x},{self.y} with radius {self.radius} and color
{self.color}")

class ShapeFactory():
  def __init__(self):
    self.circleMap = dict()
  def getCircles(self, name):
    if name not in self.circleMap:
      print(f"create {name} circle ")
      self.circleMap[name] = Circle(random.randint(0, 100), random.randint(0,
100), random.randint(0, 100), name)
    return self.circleMap[name]

def FlyWeightPatternDemo():
  factory = ShapeFactory()
  colors = ["red", "green", "blue", "yellow", "pink", "black", "white"]
  for _ in range(5):
    circle = factory.getCircles(colors[random.randint(0, len(colors)-1)])
    circle.draw()

if __name__ == '__main__':
  FlyWeightPatternDemo()
```

**输出**

```
create blue circle
draw circle at 49,3 with radius 67 and color blue
create yellow circle
draw circle at 57,43 with radius 69 and color yellow
draw circle at 57,43 with radius 69 and color yellow
create white circle
draw circle at 64,0 with radius 93 and color white
create black circle
draw circle at 23,49 with radius 18 and color black
```

# 小结

享元模式主要是为了复用对象，节省内存。它与代理模式类似，将系统中可能出现的重复对象缓存起来，从而节省空间。