设计模式-桥接模式

参考

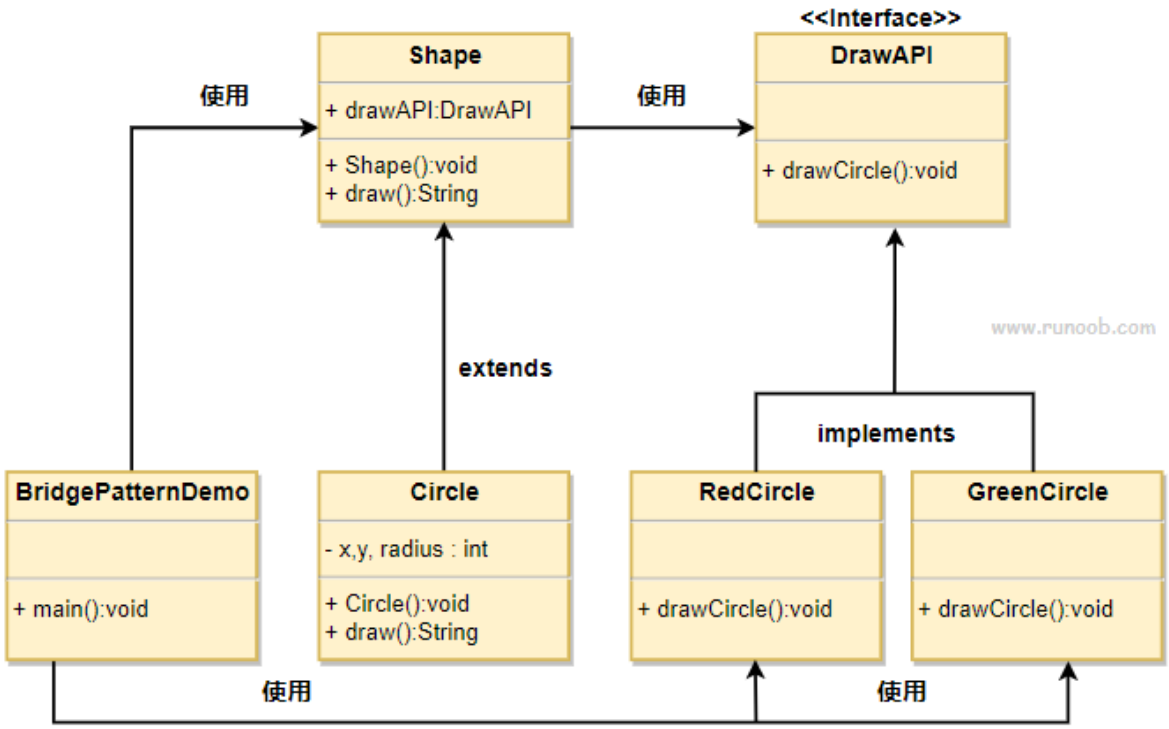
- Go设计模式(12)-桥接模式*程序员麻辣烫的博客-CSDN博客*go 桥接模式
- https://www.runoob.com/design-pattern/bridge-pattern.html

桥接适用于把抽象话与实现化解耦，使得二者可以独立变化，这种类型的设计模式属于结构型模式，它通过提供抽象化和实现化之间的桥接结构，来实现二者的解耦。这种模式设计到一个作为桥接的接口，使得实体类的功能独立于接口实现类。这两种类型的类可被结构化改变而互不影响。

使用场景

- 如果一个系统需要在后见的抽象化角色和具体化角色之间增加更多的灵活性，比米娜在两个层次之间建立静态的继承联系，通过桥接模式可以使它们在抽象层建立一个关联关系。
- 一个类存在两个独立变化的维度，且这两个维度都需要进行扩展。

Demo分析



> 分析：
>
> 我们有一个作为桥接实现的 `DrawApi` 接口和实现了 `DrawAPI` 接口的实体类 `RedCircle`、`GreenCircle`。`Shpae` 是一个抽象类，将使用 `DrawAPI` 的对象。`BridgePatternDemo` 类使用 `Shape` 类来画出不同颜色的圆。

Go实现

```
package main

import "fmt"

/**
*桥接的抽象类
 */
type DrawAPI interface {
    drawCircle(x, y, radius int)
}

type RedCircle struct {
}

func (rc *RedCircle) drawCircle(x, y, radius int) {
    fmt.Printf("draw red circle [x:%d, y:%d, radius:%d]\n", x, y, radius)
}

type GreenCircle struct{}

func (rc *GreenCircle) drawCircle(x, y, radius int) {
    fmt.Printf("draw green circle [x:%d, y:%d, radius:%d]\n", x, y, radius)
}

// 形状的抽象类
type Shape struct {
    DrawAPI
}

func (s *Shape) shape(drawAPI DrawAPI) {
    s.DrawAPI = drawAPI
}

type Circle struct {
    x, y, radius int
    Shape
}

func (c *Circle) draw() {
    c.drawCircle(c.x, c.y, c.radius)
}

func (c *Circle) initShape(x, y, radius int, drawAPI DrawAPI) {
    c.x = x
    c.y = y
    c.radius = radius
    c.shape(drawAPI)
```

```
}

func test() {
    c := &Circle{}
    c.initShape(1, 2, 3, &GreenCircle{})
    c.draw()
}

func main() {
    test()
}
```

输出

```
draw green circle [x:1, y:2, radius:3]
```

Python实现

```
class DrawApI:
  def drawCircle(self, x, y, radius):
    print("Drawing circle at ({},{}) with radius {}".format(x, y, radius))

class RedCircle(DrawApI):
  def drawCircle(self, x, y, radius):
    print("Drawing red circle at ({},{}) with radius {}".format(x, y, radius))

class BlueCircle(DrawApI):
  def drawCircle(self, x, y, radius):
    print("Drawing blue circle at ({},{}) with radius {}".format(x, y, radius))

class Shape():
  def __init__(self, drawAPI):
    self.drawAPI = drawAPI
  def shape(self, drawAPI):
    self.drawAPI = drawAPI
  def draw():
    pass

class Circle(Shape):
  def __init__(self, x, y, radius, drawAPI):
    super().__init__(drawAPI)
    self.x = x
    self.y = y
    self.radius = radius
  def draw(self):
    self.drawAPI.drawCircle(self.x, self.y, self.radius)

def test():
  c = Circle(1, 2, 3, RedCircle())
  c.draw()

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

输出

```
Drawing red circle at (1,2) with radius 3
```

小结

桥接模式，可以理解成将一组功能桥接到一个实体上，从而实现对原实体功能的扩展并且保证功能的分离。