设计模式-外观模式

参考

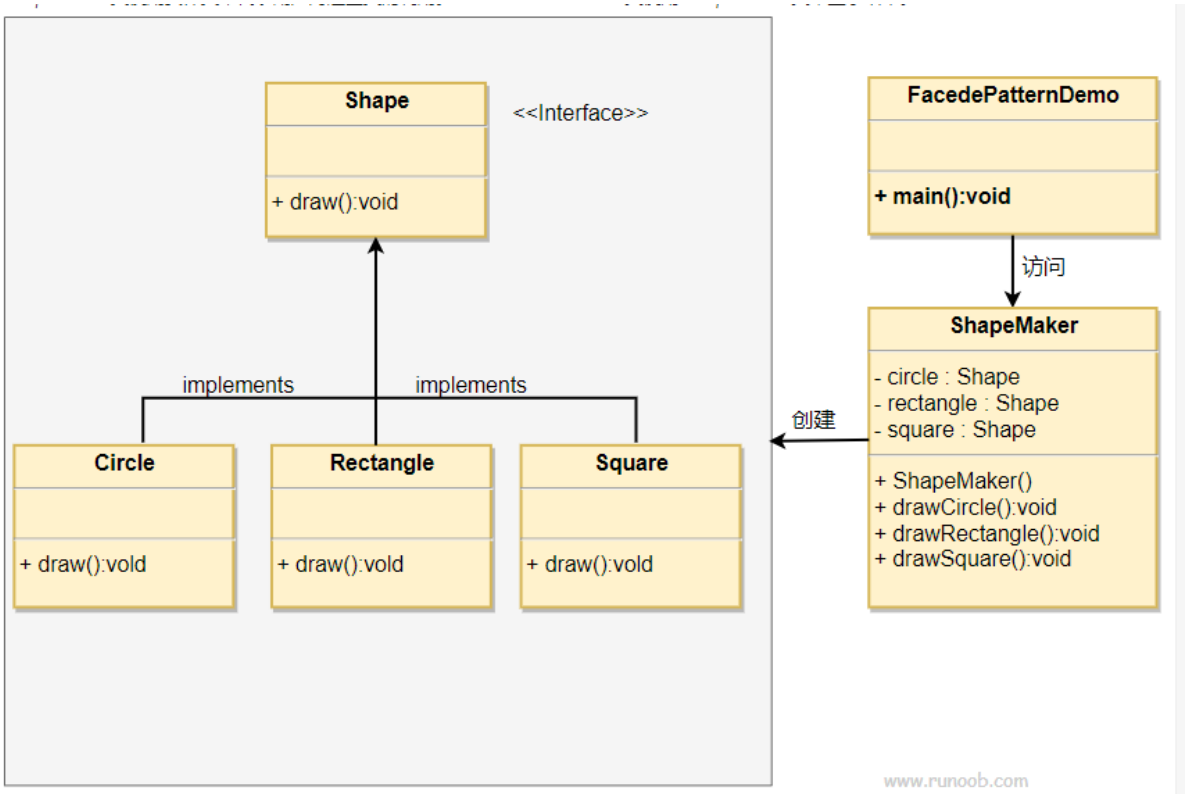
- (1条消息) Go设计模式(15)-门面模式_程序员麻辣烫的博客-CSDN博客
- https://www.runoob.com/design-pattern/facade-pattern.html

外观模式隐藏系统的复杂性，并向客户端提供了一个客户端可以访问系统的接口。这种类型的设计模式属于结构型模式，它向现有的系统添加一个接口，来隐藏系统的复杂性。

使用场景

- 子系统相对独立
- 为复杂的模块或子系统提供外界访问的模块

Demo分析



> 分析：
>
> 我们将闯进啊一个Shape接口和实现了Shape接口的实体类。下一步时是定义一个外观类ShapeMaker。ShapeMaker类使用实体类来代表用户对这些类的调用。

Go实现

```go
package main

import "fmt"

/**
* Shape的接口
 */
type Shape interface {
    draw()
}

type Circle struct{}

func (c *Circle) draw() {
    fmt.Println("draw Circle")
}

type Rectangle struct{}

func (r *Rectangle) draw() {
    fmt.Println("draw Rectangle")
}

type Square struct{}

func (s *Square) draw() {
```

```
    fmt.Println("draw Square")
}

/**
* 外观类
 */
type ShaperMaker struct {
    circle Circle
    rect   Rectangle
    square Square
}

func (m *ShaperMaker) drawCircle() {
    m.circle.draw()
}

func (m *ShaperMaker) drawSquare() {
    m.square.draw()
}

func (m *ShaperMaker) drawRectangle() {
    m.rect.draw()
}

func initShaperMaker() *ShaperMaker {
    return &ShaperMaker{
        circle: Circle{},
        rect:   Rectangle{},
        square: Square{},
    }
}

func test() {
    shapeMarker := initShaperMaker()
    shapeMarker.drawCircle()
    shapeMarker.drawSquare()
    shapeMarker.drawRectangle()
}

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

输出

```
draw Circle
draw Square
draw Rectangle
```

Python实现

```
class Shape:
  """
    ## Shape 接口
```

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

class Circle(Shape):
  def draw(self):
    print("Circle.draw")

class Square(Shape):
  def draw(self):
    print("Square.draw")

class ShapeMaker:
  """
    ## 外观类
  """
  def __init__(self):
    self.circle = Circle()
    self.square = Square()
  def drawCircle(self):
    self.circle.draw()
  def drawSquare(self):
    self.square.draw()

def test():
  shapeMaker = ShapeMaker()
  shapeMaker.drawCircle()
  shapeMaker.drawSquare()

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

输出

```
Circle.draw
Square.draw
```

小结

对比总结了前面接触的这些设计模式，其实可以感受到大部分的设计模式就是将代码按照抽象化的功能进行分块独立，可以说是各司其职。本小结的外观模式的核心思想就是具体实现的实体类做到因类而异，最后由一个外观类开创一种统一的接口提供用户调用。这样可以让用户间接调用内部具体实现的实体类，隐藏了部分代码的内部实现。