设计模式-原型模式

参考

- Go设计模式(10)-原型模式_程序员麻辣烫的博客-CSDN博客
- https://www.runoob.com/design-pattern/prototype-pattern.html

---

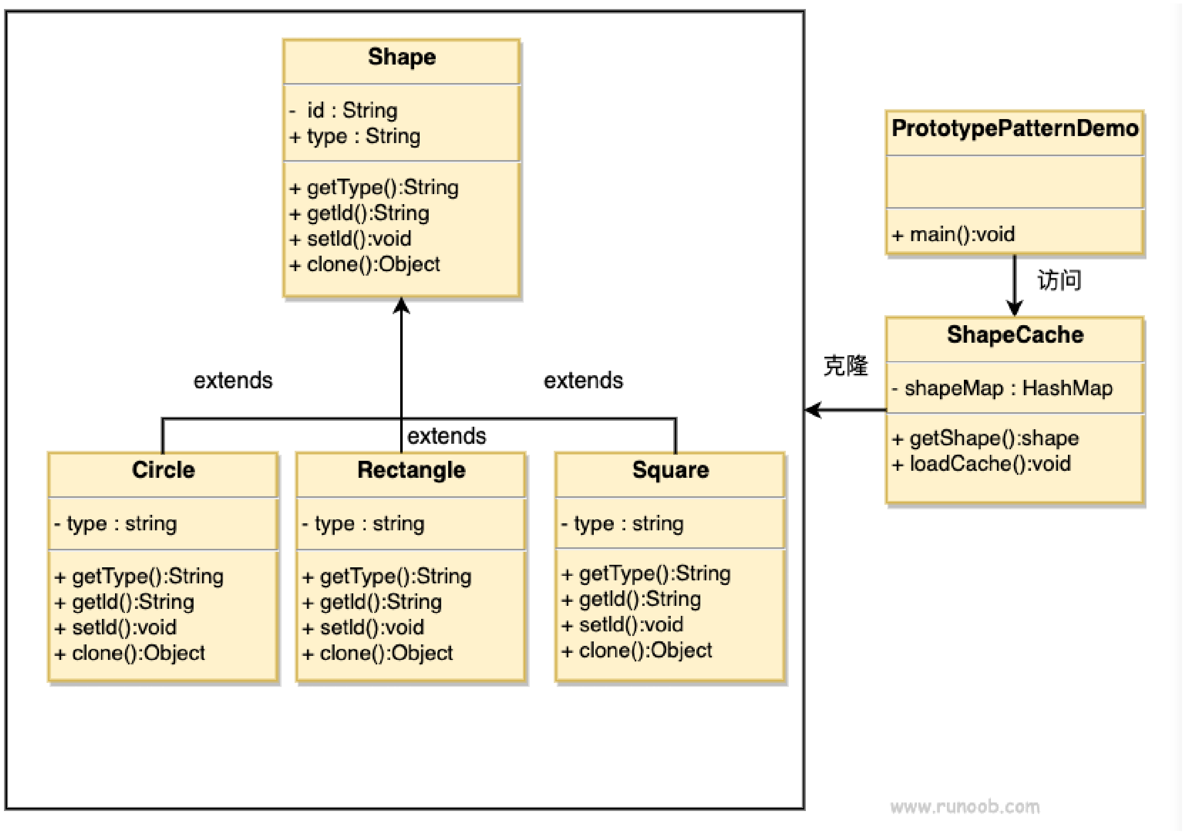
原型模式是用于创建重复的对象，同时又能保证性能。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。

这种模式是实现了一个原型接口，该接口用于创建当前对象的克隆。当直接创建对象的代价比较大时，则采用这种模式。

使用场景

- 类初始化需要消化非常多的资源、硬件资源等。
- 一个对象多个修改者的场景

Demo分析



> 分析：
>
> 我们将创建一个抽象类Shape和扩展了Shape类的实体类。下一步是定义类ShapeCache，该类把shape对象存储在一个Hashtable中，并在请求的时候返回它们的克隆。

Go实现

```
package main

import "fmt"

/**
 * 抽象类接口
 */
type Shape struct {
    Id   string
    Type string
}

func (s *Shape) getType() string {
    return s.Type
}

func (s *Shape) getId() string {
    return s.Id
}

type ShapeT interface {
    Draw()
    Clone() ShapeT
}
```

```go
type Rectangle struct {
    Shape
}

type Square struct {
    Shape
}

func (s *Square) Draw() {
    fmt.Printf("Drawing a square with id %s\n", s.getId())
}

/**
 * 深拷贝，核心
 */
func (s *Square) Clone() ShapeT {
    return &Square{Shape{
        Id:   s.Id,
        Type: s.Type,
    }}
}

func (r *Rectangle) Draw() {
    fmt.Printf("Drawing a rectangle with id %s\n", r.getId())
}

/**
 * 深拷贝，核心
 */
func (r *Rectangle) Clone() ShapeT {
    return &Rectangle{Shape{
        Id:   r.Id,
        Type: r.Type,
    }}
}

type ShapeCache struct {
    shapes map[string]ShapeT
}

func (s *ShapeCache) loadCache() {
    s.shapes = make(map[string]ShapeT)
    s.shapes["1"] = &Rectangle{
        Shape{
            Id:   "1",
            Type: "rectangle",
        },
    }
    s.shapes["2"] = &Square{
        Shape{
            Id:   "2",
            Type: "square",
        },
    }
}
```

```
func (s *ShapeCache) getShape(id string) ShapeT {
    return s.shapes[id].Clone()
}

func test() {
    shapeCache := ShapeCache{}
    shapeCache.loadCache()
    rect := shapeCache.getShape("1").(*Rectangle)
    fmt.Printf("Shape %s\n", rect.getType())
    square := shapeCache.getShape("2").(*Square)
    fmt.Printf("Shape %s\n", square.getType())
}

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

输出

```
Shape rectangle
Shape square
```

Python实现

```
class Shape(object):
  def __init__(self, Id, type):
    self.id = Id
    self.type = type
  def getId(self):
    return self.id
  def getType(self):
    return self.type
  def clone(self):
    """
      ## 原型模式的关键在于clone方法
    """
    return super(Shape, self).clone()

class Circle(Shape):
  def __init__(self, Id, type, radius):
    super(Circle, self).__init__(Id, type)
    self.radius = radius
  def draw(self):
    print("Circle with radius %d" % self.radius)

class Square(Shape):
  def __init__(self, Id, type, side):
    super(Square, self).__init__(Id, type)
    self.side = side
  def draw(self):
    print("Square with side %d" % self.side)

class ShapeCache(object):
```

```python
  def __init__(self):
    self.shapeMap = {}
  def getShape(self, Id):
    shape = self.shapeMap.get(Id, None)
    if shape == None:
      raise Exception("Shape not found")
    return shape
  def loadCache(self):
    circle = Circle("1", "Circle", 3)
    square = Square("2", "Square", 4)
    self.shapeMap["1"] = circle
    self.shapeMap["2"] = square

def PrototypePatternDemo():
  shapeCache = ShapeCache()
  shapeCache.loadCache()
  circle = shapeCache.getShape("1")
  circle.draw()
  square = shapeCache.getShape("2")
  square.draw()

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

输出

```
Circle with radius 3
Square with side 4
```

小结

原型模式就是利用对已有对象进行复制的方式，来创建对象，以达到节省创建时间的目的。