设计模式-代理模式

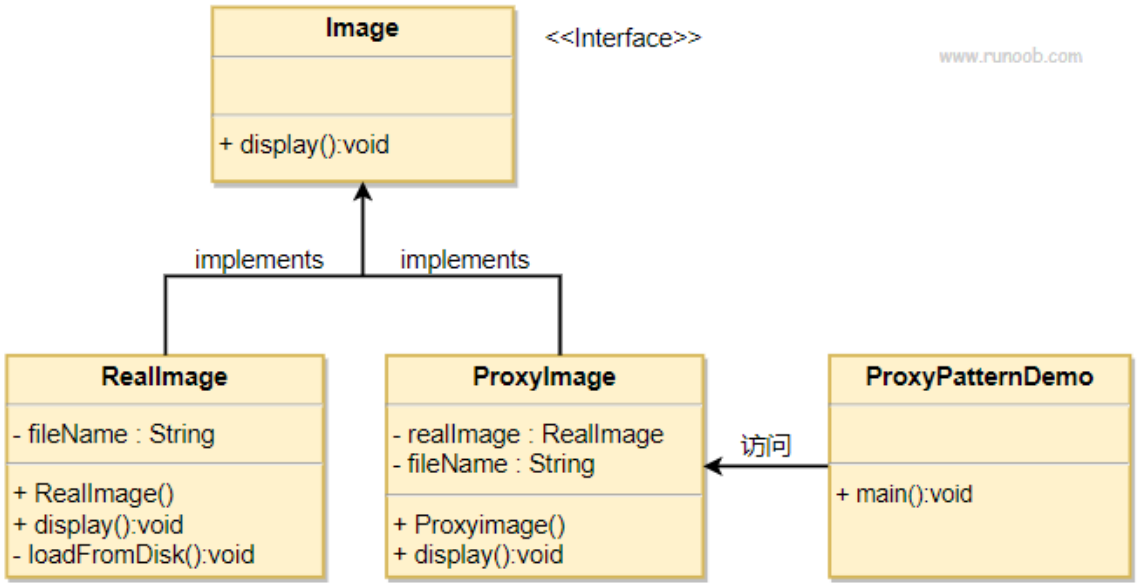
参考

- Go设计模式(11)-代理模式*程序员麻辣烫的博客-CSDN博客*go 代理模式
- https://www.runoob.com/design-pattern/proxy-pattern.html

使用场景

- 远程代理
- 虚拟代理
- 防火墙代理

Demo分析



> 分析:
>
> 根据上述的UML图，我们将创建一个Image接口和实现Image接口的实体类。 `ProxyImage` 可以减少 `RealImage` 对象的重复加载的内存占用。

Go实现

RealImage

```
type RealImage struct {
```

```
    fileName string
}

func (r *RealImage) LoadImage() {
    fmt.Printf("Loading image: %s\n", r.fileName)
}

func (rImage *RealImage) Display() {
    fmt.Printf("Displaying %s\n", rImage.fileName)
}

func initRealImage(fileName string) *RealImage {
    realImage := &RealImage{fileName}
    realImage.LoadImage()
    return realImage
}
```

ProxyImage

```

type ProxyImage struct {
    realImage *RealImage
    fileName  string
}

func (pImage *ProxyImage) Display() {
    if pImage.realImage == nil {
        pImage.realImage = &RealImage{fileName: pImage.fileName}
    }
    pImage.realImage.Display()
}
```

测试

```
func test() {
    image := initRealImage("test.jpg")
    // 要从磁盘加载
    image.Display()
    // 不需要从磁盘加载
    image.Display()
}

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

输出

```
Loading image: test.jpg
Displaying test.jpg
Displaying test.jpg
```

python实现

```python
class Image(object):
  """
    ## 图像的抽象类
  """
  def __init__(self, fileName):
    self.fileName = fileName
    pass
  def Display(self):
    pass

class RealImage(Image):
  def __init__(self, fileName):
    super().__init__(fileName)
    self.LoadImage()
  def LoadImage(self):
    print(f"Loading {self.fileName}")
  def Display(self):
    print(f"Displaying {self.fileName}")

class ProxyImage(Image):
  """
    ## 图像的代理类
  """
  def __init__(self, fileName):
    super().__init__(fileName)
    self.realImage = None
  def Display(self):
    if self.realImage is None:
      self.realImage = RealImage(self.fileName)
    self.realImage.Display()

def test():
  image = ProxyImage("test.jpg")
  image.Display()
  image.Display()

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

输出

```
Loading test.jpg
Displaying test.jpg
Displaying test.jpg
```

小结

> 总的来说代理模式，使用起来简单快捷。同时将具体业务和通用逻辑的分离，这使得系统的可扩展性增强。