设计模式-职责链模式

参考

- Go设计模式(21)-职责链模式_程序员麻辣烫的博客-CSDN博客
- https://www.runoob.com/design-pattern/chain-of-responsibility-pattern.html

---

责任链模式为请求创建了一个接受者对象的链。这种模式给予请求的类型，对请求的发送者和接受者进行解耦，这种类型的设计模式属于行为型模式。

使用场景

- 有多个对象可以处理同一个请求，具体哪个对象处理该请求由运行时刻自动确定
- 可动态指定一组对象处理请求

Demo分析

> 分析：
>
> 如果请求被多个对象进行处理，就可以用职责链模式。一些框架中的过滤器，拦截器等都是职责链模式的适用场景。下面我们仿照Gin，实现Gin的中间件。

Go实现

```
package main

import "fmt"

var status = 0

type HandlerFunc func()

type HandlersChain []HandlerFunc

/**
* 路由组件
 */
type RouteGroup struct {
    Handlers HandlersChain
    index    int8
}
```

```go
/**
* 添加中间件，将其组成链式
 */
func (group *RouteGroup) Use(handlers ...HandlerFunc) {
    group.Handlers = append(group.Handlers, handlers...)
}

/**
* 链式执行
 */
func (group *RouteGroup) Next() {
    for group.index < int8(len(group.Handlers)) {
        group.Handlers[group.index]()
        if status == 1 {
            fmt.Printf("执行失败，请重试\n")
            break
        }
        group.index++
    }
    group.index = 0
}

/**
* 中间件1
 */
func middleWare1() {
    fmt.Printf("middleWare1\n")
}

/**
* 中间件1
 */
func middleWare2() {
    fmt.Printf("middleWare2\n")
    status = 1
}

func test() {
    r := &RouteGroup{}
    r.Use(middleWare1, middleWare2)
    r.Next()

}

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

输出

```
middleWare1
middleWare2
执行失败，请重试
```

Python实现

```
class RouteGroup(object):
  def __init__(self):
    self.handlers = []
    self.index = 0
  def Use(self, *args):
    self.handlers += [*args]
  def Next(self):
    index = self.index
    for task in self.handlers[index:]:
      task()
      self.index += 1

def middleware1():
  print('middleware1')

def middleware2():
  print('middleware2')

def test():
  routeGroup = RouteGroup()
  routeGroup.Use(middleware1, middleware2)
  routeGroup.Next()

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

输出

```
middleware1
middleware2
```

小结

通过中间件的作用可以很好的说明职责链的扩展性，简单的使用Use增加自定的中间件，让开发者可以很方便的增加鉴权、限流、拦截器等操作。