设计模式-解释器模式

参考

- Go设计模式(27)-解释器模式*程序员麻辣烫的博客-CSDN博客*go语言解释器
- https://www.runoob.com/design-pattern/interpreter-pattern.html

---

解释器模式提供了评估语言的语法或表达式的方式，它属于行为模式。这种模式实现了一个表达式接口，该接口解释一个特定的上下文。

使用场景

- 可以将一个需要解释执行的语言中的句子表示为一个抽象语法树
- 一个简答语法需要解释的场景

Demo分析

当我们需要一个特别的翻译器，需要对音乐、舞蹈等进行翻译，将这些艺术形式翻译为我们熟知的语言。此时我们可以利用解释器模式将不同的解析方式拆分到各个小类中从而达到解耦。

Go实现

```
package main

import "fmt"

/**
* 内容信息
 */
type Context struct {
    action  string
    context string
}

/**
* 翻译接口
 */
type InterPreter interface {
    Interpret(context Context)
}
```

```go
type MusicInterpreter struct {
}

/**
* 音乐翻译器
 */
func (m *MusicInterpreter) Interpret(context Context) {
    fmt.Printf("%s 中的 %s 的意思是温婉尔雅\n", context.action, context.context)
}

/**
* 舞蹈翻译器
 */
type DanceInterpreter struct {
}

func (d *DanceInterpreter) Interpret(context Context) {
    fmt.Printf("%s 中的 %s 意境很ok \n", context.action, context.context)
}

func test() {
    actionList := []Context{
        {action: "music", context: "高音"},
        {action: "music", context: "低音"},
        {action: "dance", context: "跳跃"},
        {action: "dance", context: "旋转"},
    }
    for _, c := range actionList {
        if c.action == "music" {
            (&MusicInterpreter{}).Interpret(c)
        } else if c.action == "dance" {
            (&DanceInterpreter{}).Interpret(c)
        }
    }
}

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

输出

```
music 中的 高音 的意思是温婉尔雅
music 中的 低音 的意思是温婉尔雅
dance 中的 跳跃 意境很悲凉
dance 中的 旋转 意境很悲凉
```

Python实现

```python
class Context(object):
  def __init__(self, action ,context):
    self.action = action
    self.context = context
```

```
class InterPreter(object):
  """
    ## 翻译类的接口
  """
  def InterPreter(self, context):
    pass

class MusicInterpreter(InterPreter):
  """
    ## 音乐翻译类
  """
  def InterPreter(self, context):
    print(f"{context.action} 中的 {context.context} 很好听")

class MovieInterpreter(InterPreter):
  """
    ## 电影翻译类
  """
  def InterPreter(self, context):
    print(f"{context.action} 中的 {context.context} 很好看")

def test():
  actionList = [
    Context("music", "高音"),
    Context("music", "低音"),
    Context("movie", "电影"),
  ]
  for c in actionList:
    if c.action == "music":
      MusicInterpreter().InterPreter(c)
    elif c.action == "movie":
      MovieInterpreter().InterPreter(c)

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

输出

```
music 中的 高音 很好听
music 中的 低音 很好听
movie 中的 电影 很好看
```

小结

解释器模式的主要作用解释分而治之，将同一功能的不同细分功能分开进行管理，方便了解释器的扩展和维护。