设计模式-访问者模式

参考

- (1条消息) Go设计模式(24)-访问者模式_程序员麻辣烫的博客-CSDN博客
- https://www.runoob.com/design-pattern/visitor-pattern.html

在访问者模式中，我们使用了一个访问者类，它改变了元素类的执行算法。通过这种方式，元素的执行算法可以随着访问者改变而改变。这种类型的设计模式属于行为模式。根据模式，元素对象已接收访问者，这样访问者对象就可以处理元素对象上的操作。

使用场景

- 对象结构中对象对应的类很少改变，但进场需要在此结构上定义新的操作
- 需要对一个对象结构中的很多不同的且不相关的操作，而需要避免让这些操作"污染"这些对象的类，也不希望在增加新操作时修改这些类。

Demo分析

例如我们需要完成一个文件解析器，需要实现对不同类型的文件有内容提取压缩等操作。我们可以使用if-else判断语句完成，但是当添加的功能比较多时，代码堆积的就会比较厉害，看起来会很不优雅。

Go实现

```
package main

import "fmt"

type ReadFile interface {
    Read(filePath string)
    Accept(visitor Visitor)
}

type ReadPdfFile struct{}

func (r *ReadPdfFile) Read(filePath string) {
    fmt.Printf("读取pdf文件：%s\n", filePath)
}

/**
```

```
* 接收访问者
 */
func (r *ReadPdfFile) Accept(visitor Visitor) {
    visitor.VistorPdfFile(r)
}

type ReadTxtFile struct{}

func (r *ReadTxtFile) Read(filePath string) {
    fmt.Printf("读取txt文件：%s\n", filePath)
}

func (r *ReadTxtFile) Accept(visitor Visitor) {
    visitor.VistorTxtFile(r)
}

type Visitor interface {
    VistorPdfFile(r *ReadPdfFile)
    VistorTxtFile(r *ReadTxtFile)
}

/**
 * 读取文件类
 */
type ExactFile struct{}

func (e *ExactFile) VistorPdfFile(r *ReadPdfFile) {
    fmt.Println("读取pdf文件")
}

func (e *ExactFile) VistorTxtFile(r *ReadTxtFile) {
    fmt.Println("读取txt文件")
}

/**
* 压缩文件类
 */
type CompressionFile struct{}

func (c *CompressionFile) VistorPdfFile(r *ReadPdfFile) {
    fmt.Println("压缩pdf文件")
}

func (c *CompressionFile) VistorTxtFile(r *ReadTxtFile) {
    fmt.Println("压缩txt文件")
}

func test() {
    fileList := []ReadFile{
        &ReadPdfFile{},
        &ReadTxtFile{},
        &ReadTxtFile{},
    }
    extract := ExactFile{}
    for _, f := range fileList {
```

```
        f.Accept(&extract)
    }
    fmt.Println("------------------------------------")
    compress := CompressionFile{}
    for _, f := range fileList {
        f.Accept(&compress)
    }
}

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

输出

```
读取pdf文件
读取txt文件
读取txt文件
------------------------------------
压缩pdf文件
压缩txt文件
压缩txt文件
```

Python实现

```
class ReadFile:
  """
    ## 各种文件的抽象类型
  """
  def Read(self):
    pass
  def Accept(self):
    pass

class ReadPdfFile(ReadFile):
  def Read(self):
    print("Read PDF File")
  def Accept(self, visitor):
    visitor.VisitPdfFile(self)

class ReadTxtFile(ReadFile):
  def Read(self):
    print("Read TXT File")
  def Accept(self, visitor):
    visitor.VisitTxtFile(self)

class Visitor:
  """
    ## 访问者抽象类
  """
  def VistorPdfFile(self, r):
    pass
  def VistorTxtFile(self, r):
    pass
```

```
class ExactFile(Visitor):
  """
    ## 读取文件的类
  """
  def VisitPdfFile(self, file):
    print("读取pdf文件")
  def VisitTxtFile(self, file):
    print("读取txt文件")

class CompressionFile(Visitor):
  """
    ## 压缩文件的类
  """
  def VisitPdfFile(self, file):
    print("压缩pdf文件")
  def VisitTxtFile(self, file):
    print("压缩txt文件")

def test():
  fileList = [
    ReadPdfFile(),
    ReadTxtFile(),
    ReadPdfFile(),
  ]
  extract = ExactFile()
  compress = CompressionFile()
  for file in fileList:
    file.Accept(extract)
  for file in fileList:
    file.Accept(compress)

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

输出

```
读取pdf文件
读取txt文件
读取pdf文件
压缩pdf文件
压缩txt文件
压缩pdf文件
```

小结

面对上述的多种文件类型的多种操作方式的场景。我们采用访问者的模式，根据不同的功能设置不同的访问者（不是经常会出现来者何意嘛，这里不同的功能代表来意带入场景应该就可以方便记住访问者的身份了）。这样的好处就是我们将这样一个文件使用的场景分成了两个大的模块：功能的具体实现模块，面向使用的模块。