

访问者模式

在访问者模式（Visitor Pattern）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型模式。根据模式，元素对象已接受访问者对象，这样访问者对象就可以处理元素对象上的操作。

介绍

意图：主要将数据结构与数据操作分离。

主要解决：稳定的数据结构和易变的操作耦合问题。

何时使用：需要对一个对象结构中的对象进行很多不同的并且不相关的操作，而需要避免让这些操作"污染"这些对象的类，使用访问者模式将这些封装到类中。

如何解决：在被访问的类里面加一个对外提供接待访问者的接口。

关键代码：在数据基础类里面有一个方法接受访问者，将自身引用传入访问者。

应用实例：您在朋友家做客，您是访问者，朋友接受您的访问，您通过朋友的描述，然后对朋友的描述做出一个判断，这就是访问者模式。

优点： 1、符合单一职责原则。 2、优秀的扩展性。 3、灵活性。

缺点： 1、具体元素对访问者公布细节，违反了迪米特原则。 2、具体元素变更比较困难。 3、违反了依赖倒置原则，依赖了具体类，没有依赖抽象。

使用场景： 1、对象结构中对象对应的类很少改变，但经常需要在此对象结构上定义新的操作。 2、需要对一个对象结构中的对象进行很多不同的并且不相关的操作，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。

注意事项：访问者可以对功能进行统一，可以做报表、UI、拦截器与过滤器。

实现

我们将创建一个定义接受操作的 *ComputerPart* 接口。*Keyboard*、*Mouse*、*Monitor* 和 *Computer* 是实现了 *ComputerPart* 接口的实体类。我们将定义另一个接口 *ComputerPart Visitor*，它定义了访问者类的操作。*Computer* 使用实体访问者来执行相应的动作。*VisitorPatternDemo*，我们的演示类使用 *Computer*、*ComputerPartVisitor* 类来演示访问者模式的用法。




```
}  
}
```

Computer.java

```
public class Computer implements ComputerPart {  
  
    ComputerPart[] parts;  
  
    public Computer(){  
        parts = new ComputerPart[] {new Mouse(), new Keyboard(), new  
Monitor()};  
    }  
  
    @Override  
    public void accept(ComputerPartVisitor computerPartVisitor) {  
        for (int i = 0; i < parts.length; i++) {  
            parts[i].accept(computerPartVisitor);  
        }  
        computerPartVisitor.visit(this);  
    }  
}
```

步骤 3

定义一个表示访问者的接口。

ComputerPartVisitor.java

```
public interface ComputerPartVisitor {  
    public void visit(Computer computer);  
    public void visit(Mouse mouse);  
    public void visit(Keyboard keyboard);  
    public void visit(Monitor monitor);  
}
```

步骤 4

创建实现了上述类的实体访问者。

ComputerPartDisplayVisitor.java

```
public class ComputerPartDisplayVisitor implements ComputerPartVis  
itor {  
  
    @Override  
    public void visit(Computer computer) {  
        System.out.println("Displaying Computer.");  
    }  
  
    @Override  
    public void visit(Mouse mouse) {  
        System.out.println("Displaying Mouse.");  
    }  
  
    @Override  
    public void visit(Keyboard keyboard) {  
        System.out.println("Displaying Keyboard.");  
    }  
  
    @Override
```



```
public void visit(Monitor monitor) {
    System.out.println("Displaying Monitor.");
}
}
```

步骤 5

使用 `ComputerPartDisplayVisitor` 来显示 `Computer` 的组成部分。

VisitorPatternDemo.java

```
public class VisitorPatternDemo {
    public static void main(String[] args) {

        ComputerPart computer = new Computer();
        computer.accept(new ComputerPartDisplayVisitor());
    }
}
```

步骤 6

执行程序，输出结果：

```
Displaying Mouse.
Displaying Keyboard.
Displaying Monitor.
Displaying Computer.
```

← 模板模式

MVC 模式 →



2 篇笔记



写笔记



Python 代码：

55

```
# Visitor Pattern with Python Code
from abc import abstractmethod, ABCMeta
# 定义一个表示元素 (Element) 的接口
class ComputerPart(metaclass=ABCMeta):
    @abstractmethod
    def accept(self, inComputerPartVisitor):
        pass
#创建阔爱站了ComputerPart的实体类
class Keyboard(ComputerPart):
    def accept(self, inComputerPartVisitor):
        inComputerPartVisitor.visitKeyboard(self)
class Monitor(ComputerPart):
    def accept(self, inComputerPartVisitor):
        inComputerPartVisitor.visitMonitor(self)
class Mouse(ComputerPart):
    def accept(self, inComputerPartVisitor):
        inComputerPartVisitor.visitMouse(self)
class Computer(ComputerPart):
    _parts = []
    def __init__(self):
        self._parts.append(Mouse())
```



```

        self._parts.append(Keyboard())
        self._parts.append(Monitor())
    def accept(self, inComputerPartVisitor):
        for aPart in self._parts :
            aPart.accept(inComputerPartVisitor)
        inComputerPartVisitor.visitComputer(self)
# 定义一个表示访问者的接口
class ComputerPartVisitor(metaclass=ABCMeta):
    @abstractmethod
    def visitComputer(self,inComputer):
        pass
    @abstractmethod
    def visitMouse(self,inMouse):
        pass
    @abstractmethod
    def visitKeyboard(self,inKeyboard):
        pass
    @abstractmethod
    def visitMonitor(self,inMonitor):
        pass
# 实现访问者接口的实体类
class ComputerPartDisplayVisitor(ComputerPartVisitor):
    def visitComputer(self,inComputer):
        print("Displaying {0}. Called in {1}".format(inCom
    def visitMouse(self,inMouse):
        print("Displaying {0}. Called in {1}".format(inMou
    def visitKeyboard(self,inKeyboard):
        print("Displaying {0}. Called in {1}".format(inKey
    def visitMonitor(self,inMonitor):
        print("Displaying {0}. Called in {1}".format(inMon
# 调用输出
if __name__ == '__main__':
    aComputer = Computer()
    aComputer.accept(ComputerPartDisplayVisitor())

```

Siskin.xu 3年前 (2020-03-10)



8

```

from abc import abstractmethod, ABCMeta

class device(metaclass=ABCMeta):
    pass

# 键盘类
class Keyboard(device):
    def __init__(self, function):
        self.function = function # 功能

    def accept(self, visitor):
        # 把自身交给访问器
        visitor.visitKeyboard(self)

```



显示器类

```
class Monitor(device):  
    def __init__(self, function):  
        self.function = function  
  
    def accept(self, visitor):  
        visitor.visitMonitor(self)
```

鼠标类

```
class Mouse(device):  
    def __init__(self, function):  
        self.function = function  
  
    def accept(self, visitor):  
        visitor.visitMouse(self)
```

电脑类

```
class Computer:  
    parts = []  
  
    def __init__(self, parts):  
        self.parts.extend(parts)  
  
    def play(self):  
        for part in self.parts:  
            print(f"{type(part).__name__}的功能: {part.function}")  
  
    def accept(self, visitor):  
        for part in self.parts:  
            part.accept(visitor)
```

访问者

```
class Visitor(metaclass=ABCMeta):  
    @abstractmethod  
    def visitMouse(self, mouse):  
        pass  
  
    @abstractmethod  
    def visitKeyboard(self, keyboard):  
        pass  
  
    @abstractmethod  
    def visitMonitor(self, monitor):  
        pass
```

```
class updateVisitor(Visitor):  
    def visitMouse(self, mouse):  
        mouse.function += "点击"  
  
    def visitKeyboard(self, keyboard):  
        keyboard.function += "字母"
```



```
def visitMonitor(self, monitor):  
    monitor.function = "彩色屏幕"
```

```
computer = Computer([Mouse("移动"), Monitor("黑白屏幕"), Ke  
computer.play()
```

```
print()
```

```
# 接收升级包
```

```
computer.accept(updateVisitor())
```

```
computer.play()
```

ARCTURUS 1年前 [2022-01-25]

在线实例

- HTML 实例
- CSS 实例
- JavaScript 实例
- Ajax 实例
- jQuery 实例
- XML 实例
- Java 实例

字符集&工具

- HTML 字符集设置
- HTML ASCII 字符集
- JS 混淆/加密
- PNG/JPEG 图片压缩
- HTML 拾色器
- JSON 格式化工具
- 随机数生成器

最新更新

- Vue3 创建单文件...
- Vue3 指令
- Matplotlib imre...
- Matplotlib imsa...
- Matplotlib imsh...
- Matplotlib 直方图
- Python object()...

站点信息

- 意见反馈
- 免责声明
- 关于我们
- 文章归档

关注微信

