代理模式

|  |  |
| --- | --- |
|  | 🢂 内容概览 |
|  | Why：此文档用来做什么？它存在的意义是什么？为解决什么问题？   |  | | --- | |  |   What：当前包含了那些内容？   |  | | --- | |  |   How：此文档应如何参考？   |  | | --- | |  |   Who：此文档适用于那些人员阅读参考？   |  | | --- | |  |   Summary：摘要   |  | | --- | |  |   Reference：参考文献   |  | | --- | |  | |

目录

[1 模式描述 3](#_Toc484448239)

[2 模式类图 3](#_Toc484448240)

[3 相关原则 4](#_Toc484448241)

[4 模式实现 4](#_Toc484448242)

[5 适用场景 5](#_Toc484448243)

[6 缺点&权衡点 5](#_Toc484448244)

[7 应用案例 5](#_Toc484448245)

[8 相关模式 5](#_Toc484448246)

[9 问题思考 5](#_Toc484448247)

# 模式描述

1. 表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素类的前提下定义作用于这些元素的新操作；
2. 模式将特定操作抽象成”访问者”，每一个访问者的具体实现，代表一种操作类型；
3. 新增访问者实现，对被访问元素的代码无影响，从而实现对已有元素的”赋能”，而又不违背开闭原则；

# 模式类图

|  |
| --- |
|  |

**模式角色如下：**

1. **Visitor**: 访问者的抽象，定义了访问者可以访问哪些Element，可访问的Element由Visitor的入参决定，由于入参引用指向ConcreteElement，因此当ConcreteElement数据结构有变化时，需要更改相应ConcreteVisitor的实现。因此访问者不适用于ConcreteElement数据结构频繁变化的场景；

注：Visitor中可以定义针对一个或者一组ConcreteElement的访问操作，这依据具体的需求场景而定，此处的”元素”和”操作”的划分是灵活的，但被访问的一个或者一组ConcreteElement其数据结构一定是不频繁变化的；

1. **ConcreteVisitor**: 定义对ConcreteElement操作的具体实现，依照需求的不同，实现不同的ConcreteVisitor；
2. **Element**: 被访问对象的抽象，用于注入访问者到具体的元素中，从而为访问者访问元素数据结构提供途径；

注：accept方法用于执行注入操作，但不一定负责执行Visitor的访问方法；Visitor的访问方法可以根据需求，在特定的时机被Element调用；

1. **ConcreteElement**: 被访问元素的具体实现，需要通过实现Element的方法，来获取注入的Visitor；
2. **ObjectStructure**：用于存放一组Element的数据结构，可以是一个集合或者复合对象，该角色非关键角色，依据需要进行取舍；

# 模式实现

# 模式扩展

# 适用场景

# 权衡点

# 应用案例

# 相关模式

# 相关原则

|  |  |  |  |
| --- | --- | --- | --- |
| 原则名称 | 符合 | 违反 | 描述 |
| 单一职责原则 | √ |  | 每一个访问者只实现一项特定的操作，符合单一职责原则； |
| 里是替换原则 |  |  |  |
| 依赖倒转原则 |  | √ | Visitor的实现依赖于具体的Element实现，故不符合依赖倒转原则； |
| 接口隔离原则 |  |  |  |
| 迪米特法则 |  | √ | Visitor了解Element元素的具体数据结构，违背最小知识原则； |
| 开闭原则 | √ |  | 在Element结构不变的情况下，新增Visitor实现，并不会对Element代码进行修改，符合开闭原则； |
| 合成复用原则 |  |  |  |

# 问题思考