Visitor

Propósito

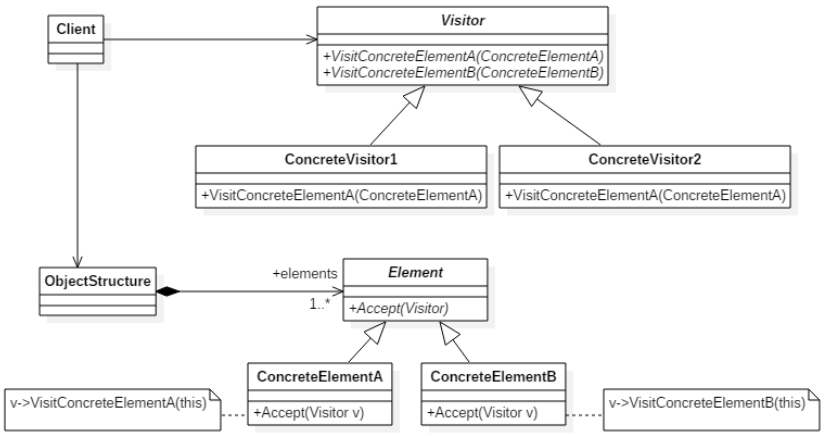
Representa una operación a realizar en los elementos de un objeto compuesto. El patrón Visitor nos permite definir una nueva operación sin modificar las clases de los objetos sobre los que se aplica.

Aplicabilidad

El patrón Visitor se aplica cuando:

- Un objeto compuesto contiene muchos tipos de objetos con diferentes interfaces y se desea realizar operaciones sobre esos objetos y esas operaciones dependen de las clases concretas
- Se deben realizar muchas operaciones diferentes y no relacionadas sobre objetos de una estructura y no se desea "polucionar" las clases con esas operaciones. El Visitor nos permite mantener en un sitio las operaciones relacionadas.
- Las clases que definen el objeto compuesto rara vez cambian, aunque puede ser frecuente definir nuevas operaciones sobre la estructura. Si las clases del objeto compuesto cambian con frecuencia, entonces es mejor definir las nuevas operaciones en esas clases.

Estructura



Participantes

- Visitor: Declara una operación Visit por cada clase ConcreteElement del objeto compuesto. El nombre de la operación identifica la clase que realiza la solicitud al Visitor. Esto permite al visitor determinar la clase concreta que está visitando. Después el visitor puede acceder al elemento directamente utilizando su interface.
- ConcreteVisitor: implementa cada operación declarada por el Visitor.
- Element: define una operación Accept que toma un visitor como argumento
- ConcreteElement: implementa la operación Accept que tiene un visitor como argumento
- ObjectStructure:

- Puede enumerar sus elementos
- Puede proporcionar una interface de alto nivel para permitir al visitor visitar sus elementos
- Puede ser un Composite o una colección

Colaboraciones

- Un cliente que utiliza el Visitor debe crear objetos ConcreteVisitor y recorrer la estructura compuesta, visitando cada elemento
- Cuando un elemento es visitado, llama a la operación correspondiente del Visitor. El elemento se pasa a sí mismo como argumento para que el visitor pueda acceder a él

Consecuencias

Algunos beneficios y desventajas del patrón Visitor:

1. El Visitor facilita agregar nuevas operaciones. Los Visitor facilitan agregar nuevas operaciones que dependen de objetos compuestos complejos. Puedes definir una nueva operación en una estructura con sólo definir un nuevo visitor. En contraste, si diseminas la funcionalidad en muchas clases, entonces debes cambiar cada clase para definir una nueva operación.
2. Un visitor reúne operaciones relacionadas y separa las no relacionadas. El comportamiento relacionado no se disemina por las clases del objeto compuesto, sino que está localizado en el visitor. Conjuntos no relacionados de comportamiento se dividen en sus propias subclases visitor. Esto simplifica tanto la estructura compuesta como los algoritmos definidos en los visitor. Los algoritmos específicos de la estructura de datos permanecen ocultos para el visitor.
3. Agregar nuevos ConcreteElement es costoso. El Visitor hace más difícil agregar nuevas subclases a Element. Cada nuevo ConcreteElement obliga a definir una operación abstracta en Visitor y una implementación en cada clase ConcreteVisitor. Se puede definir un comportamiento predeterminado en Visitor, pero esto debería ser una excepción.
   Con lo cual, hay que ponderar si es más probable modificar el algoritmo que se aplica sobre el objeto compuesto, o las clases que componen la estructura. Si éstas últimas varían mucho, entonces es mejor poner las operaciones en la estructura. Si la estructura es estable y varían las operaciones sobre ella, entonces el Visitor sirve para gestionar estas variaciones.
4. Visitar jerarquías de clases. El Iterator puede visitar objetos en una estructura, pero no puede trabajar con objetos diferentes. El Visitor no tiene esta restricción.
5. Acumular estados. Los Visitor pueden acumular los estados de los elementos que van visitando. Sin un visitor, este estado habría que pasarlo como argumento a las operaciones que realizan el recorrido, o podrían aparecer como variables globales.
6. Romper la encapsulación. La solución del Visitor asume que la interface de los ConcreteElement es suficientemente potente para permitir a los visitor realizar su trabajo. Como consecuencia, el patrón nos fuerza a hacer públicas las operaciones que acceden a su estado interno, lo cual compromete la encpasulación.