Difference between visitor design pattern and depth first search?

Ask Question

A depth first search seem able to perform similar functions as the visitor design pattern. A visitor allows you to define some data structures and add operations on those structures (in the form of multiple visitors) as needed without modifying the structures themselves. A description of the visitor pattern is provided on wikipedia. If we do a depth first search (or any other graph search algorithm like breadth first search) on the data structure, and every time an element of the structure is found, we run our desired operation, then this seems to perform the same function as the visitor. For example, consider a tree. Even if some nodes of the tree have different type, we can still check for the node types when doing DFS and then perform different operations based on the node type.

oop search design-patterns tree visitor-pattern

edited Oct 20 '13 at 5:54

asked Jun 9 '11 at 20:52

john

196 3 4 13

8 Answers

You can have Visitor implementation without having DFS. Similarly, you can do a DFS without use of the Visitor pattern. They are completely separate.

I happen to agree with the implication that they could be used together in an elegant way.

As a note, for the canonical Visitor pattern the objects being visited need to implement some sort of `AcceptVisitor` interface -- the clause "without modifying the structures themselves" in your question leads me to question whether you are doing that.

answered Jun 9 '11 at 20:55

hvgotcodes

88.6k 21 160 218

A depth-first search is just that--a search. A Visitor pattern is orthogonal to a depth-first search, in the sense that a Visitor doesn't necessarily care *how* the tree is traversed; it just knows what it needs to do on/to/with each node.

answered Jun 9 '11 at 20:57

NRitH

6,936 1 23 30

Let me answer the question in code:

```
/**
 * This method makes it easier for a class to process all
 * the nodes in an item. The class should implement
 * the NodeVisitor interface. This method will cause the
 * NodeVisitor.processNode() method to be called once for each
 * node in this item.
 * @param visitor the class that will visit each node
 * @throws PipelineException some recoverable exception
 */
public void visitNodes(NodeVisitor visitor) throws PipelineException {
    visitNodes(visitor, root);
}

private void visitNodes(NodeVisitor visitor, Node node) throws PipelineException
{
    visitor.processNode(node);
    if (node.hasChildren()) {
        int childCount = node.getChildCount();
        NodeList children = node.getChildren();
        for (int i = 0; i < childCount; i++) {
            visitNodes(visitor, children.get(i));
        }
    }
}
```

```
/**
 * Classes that implement this interface can be used with
 * Item.visitNodes(). This method provides a convenient
 * way to iterate over all the nodes in an item.
 */
public interface NodeVisitor {
        public void processNode(Node node) throws PipelineException;
}
```

In this case the visitNodes() method implements a depth-first search, but it doesn't have to. It could implement any search that hits all nodes. It's the combination of the visitNodes() method and the NodeVisitor interface that comprise the "visitor pattern" (or one particular manifestation of it.)

There's no "tradeoff" between the design pattern and the search algorithm. The design pattern just makes the algorithm easier to use.

answered Oct 21 '13 at 1:13

ccleve

6,467 11 53 104

In graph theory you can construct a graph such that the minimum spanning tree is not a path from a depth-first search, precisley it's a non-path:https://cs.stackexchange.com/questions/6749/depth-first-search-to-find-minimum-spanning-tree. I think you cannot apply this to a design pattern.

edited Apr 13 '17 at 12:48

Community ♦

1 1

answered Oct 21 '13 at 0:24

Bytemain

11.3k 6 21 67

Depth first seach is an "algorithm" and the *visitor pattern* is a way to forget algorithm concerns and focus on actions to perform.

Actually, the *visitor Pattern* may be a good way to **index** content, as it provides a "structure-agnostic" behaviour (you can change structure without rewriting visitors)

But if you want to perform a **search**, I would not advise you to use it. Any search algorithm is related to a special type of structure (tree, digraph, flow graphs, etc)

In some cases, you can achieve a *depth-first search* with a *visitor pattern*, but this is not the purpose of this pattern.

The use of visitor pattern doesn't depend of what kind of parsing you are using, but what the parsing must be performed for.

edited Oct 21 '13 at 8:53

answered Oct 21 '13 at 8:35

Charles-Edouard Coste

1,111 7 20

The `visitor` instance may choose to visit its child nodes however it wishes and is not bound by the traversal order of Depth First Search. (For example it could use Breath First Search)

Another thing to mention is the structure getting traversed does not need to be an explicit tree. I've implemented Visitors that iterate over structured data that wasn't tree like at all. I used a visitor in that instance because I could hide the complicated binary format of the file I was parsing and allow clients to control which parts of the structure they wanted to parse without needing them to know the file format specification.

answered Oct 21 '13 at 18:22

dkatzel

24.2k 2 38 53

I too go with NRITH's answer as visitor pattern know **'only what to do with the node'** on top of that Visitor lets you **'define a new operation without changing the classes of the elements on which it operates'** and the DFS talks about how the node search is performed. But for performing depth traversal and short-circuit branch traversal we use **Hierarchical Visitor pattern**(http://c2.com/cgi/wiki?HierarchicalVisitorPattern).

You can also look into When should I use the Visitor Design Pattern? which talks about the relation between visitor pattern/DFS/Hierarchical visitor pattern.

edited May 23 '17 at 12:17

Community ♦

1 1

answered Oct 22 '13 at 18:57

srsyogesh

499 3 16

The visitor pattern as (first?) described in "Design Patterns" by Erich Gamma et. al. does not necessarily include the traversal through the data structure in the accept method. Although this is a very convenient combination, there is an explicit example for external iteration at the end of the Sample Code chapter in the book.

So as others already said, a depth first traversal implemented outside of the accept method could still implement the Visitor Pattern. The question is then, what's the difference between calling element.accept(visitor) which in turn directly calls visitor.visitElement(me) compared to directly calling visitor.visitElement(element)? I can see only two reasons one may want to do that:

1. You can not or do not want to find out the concrete class of element, and by just stupidly calling element.accept(visitor), the element itself has to decide, whether visitor.visitElement or e.g. visitor.visitAnotherElement is the right operation to be called.
2. Some of the elements are composites without external access to the contained inner elements, and the visitor operations are defined for the inner elements. So the accept method would loop over the inner elements and would call visitor.visitInnerElement(innerElement). And as you do not get hold of the inner elements from outside, you also can not call visitor.visitInnerElememt(innerElement) from outside.

In summary: If you have a nicely encapsulated traversal algorithm, where you can pass in a "Visitor"-like class, and which is able to dispatch to the matching visit methods depending on the object type encountered during traversal, you do not need to bother about accept-methods. You will still be able to add new operations by just adding new Visitor implementations, and without touching your data structure nor your traversal algorithm. Whether this should still be called Visitor Pattern is a pretty academic discussion. I think in most cases bothering with an accept method makes only sense if implementation of the accept method also includes the traversal.

answered Oct 25 '13 at 14:22

user2808624

1,904 8 20