Visitor（访问者）

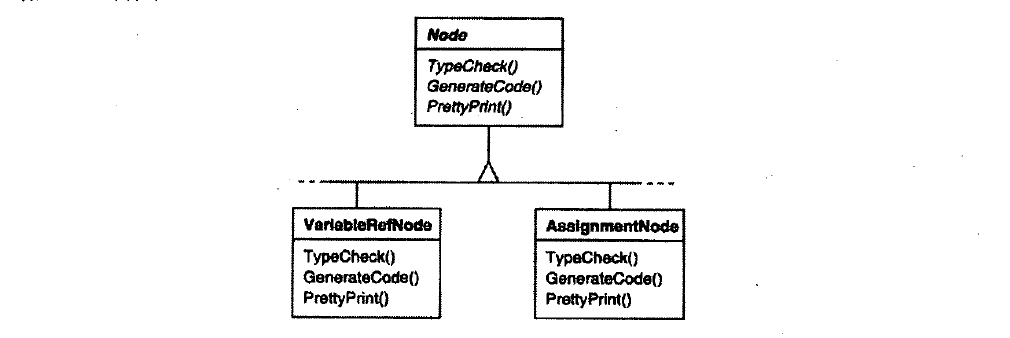
意图

表示一个作用于某个对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。

动机

考虑一个编译器，它将源程序表示为一个抽象语法树。该编译器需在抽象语法树上实施某些操作以进行“静态语法”分析，例如检查是否所有的变量都已经被定义了。它也不需要生成代码。因此它可能要定义许多操作以进行类型检查、代码优化、流程分析，检查变量是否在使用前被赋初值，等等。此外，还可以使用抽象语法树进行优美格式打印、程序重构以及对程序进行多种度量。

这些操作要求对不同的节点进行不同的处理。例如代表赋值语句的节点的处理就不同于代表变量或算术表达式的结点的处理。因此有用于赋值语句的类，有用于变量访问的类，还有用于算术表达式的类，等等。结点类的集合当然依赖于被编译的语言，但对于一个给定的语言其变化不大。

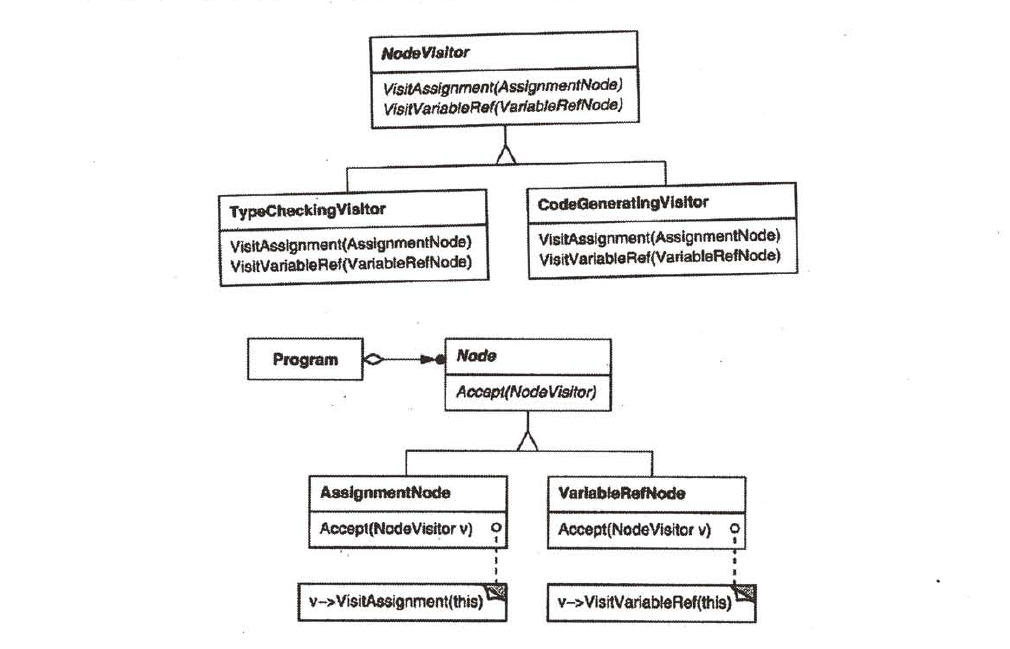


上面的框图显示了Node类层次的一部分。这里的问题是，将所有这些操作分散到各种结点类中会导致整个系统难以理解、难以维护和修改。将类型检查代码与优美格式化打印代码或流程分析代码放在一起，将产生混乱。此外，增加新的操作通常需要重新编译所有这些类。如果可以独立地增加新的操作，并且使这些结点类独立于作用于其他的操作，将会更好一些。

要实现上述两个目标，我们可以将每一个类中相关的操作包装在一个独立的对象（称为一个Visitor）中，并遍历抽象语法树时将此对象传递给当前访问的元素，当一个元素“接受”该访问者时，该元素向访问者发送一个包含自身类信息的请求。该请求同时也将该元素本身作为一个参数。然后访问者将为该元素执行该操作—这一操作以前是该元素的类中的。

例如，一个不使用访问者的编译器可能会通过在它的抽象语法树上调用TypeCheck操作对一个过程进行类型检查。将一个节点将对调用它的成员的TypeCheck以实现自身的TypeCheck（参看前面的类框图）。如果该编译器使用访问者对一个过程进行类型检查，那么它将会创建一个TypeCheckingVisitor对象，并以这个对象为一个参数在抽象语法树上调用Accept操作。每一个节点在实现Accept时将会回调访问者：一个赋值结点调用访问者的VisitAssignment操作，而一个变量应用将调用VisitVariableReference。以前类AssignmentNode的TypeCheck操作现在成为TypeCheckingVisitor的VisitAssignment操作。

为访问者不仅仅只做类型检查，我们需要所有抽象语法树的访问者有一个抽象的父类NodeVisitor。NodeVisitor必须为每一个节点类定义一个操作。一个需要计算程序度量的应用将定义NodeVisitor的新的子类，并且将不再需要在结点类中增加与特定应用相关的代码。Visitor模式将每一个编译步骤的操作封装在一个与该步骤相关的Visitor中。如下图



使用Visitor模式，必须定义两个类层次：一个对应于接受操作的元素（Node层次）另一个对应于定义对元素的操作的访问者（NodeVisitor层次）。给访问者类层次增加一个新的子类即可创建一个新的操作。只要该编译器接受的语法不改变（即不需要增加新的Node子类），我们就可以简单的定义新的NodeVisitor子类以增加新的功能。

结构图

