【访问数据结构并处理数据】

大家知道圣诞节的故事吗？即将生产的玛利亚在丈夫约瑟夫的陪伴下来到伯利恒，这里有很多住宿的地方，他们依次敲门……

本章中我们将要学习Visitor模式。Visitor是“访问者”的意思。

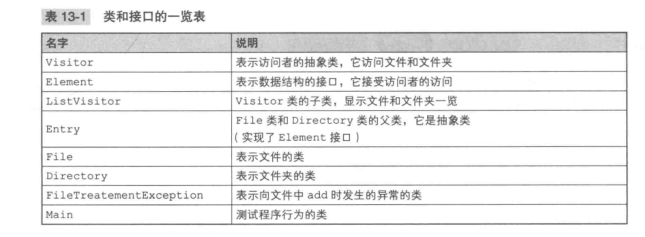
在数据结构中保存着许多元素，我们会对这些元素进行“处理”。这时，“处理”代码放在哪里比较好呢？通常的做法是将它们放在表示数据结构的类中。但是，如果“处理”有许多种呢？这种情况下，每当增加一种处理，我们就不得不去修改表示数据结构的类。

在Visitor模式中，数据结构与处理被分离开来。我们编写一个表示“访问者”的类来访问数据结构中的元素，并把对各元素的处理交给访问者类。这样，当需要增加新的处理时，我们只需要编写新的访问者，然后让数据结构可以接受访问者的访问即可。

13.2示例程序

下面我们来看看Visitor模式的示例程序。在示例程序中，我们使用Composite模式（第11章）中用到的那个文件和文件夹的例子作为访问者要访问的数据结构。访问者会访问由文件和文件夹构成的数据结构，然后显示出文件和文件夹的一览。

表13-1类和接口的一览表



|Visitor与Element之间的相互调用

读到这里，大家应该理解了Visitor模式是如何工作的吧。笔者在初次接触Visitor模式时，完全无法理解这个模式。在笔者的头脑中，accept方法和visit方法的调用关系是一片混乱的。因此，这里我们再结合时序图（图13-3）来学习一下示例程序的处理流程（关于时序图的知识请参见p.xii）。

为了方便理解，我们在图13-3中展示了当一个文件夹下有两个文件时，示例程序的处理流程。

①首先，Main类生成ListVisitor的实例。在示例程序中，Main类还生成了其他的Directory类和File类的实例，但在本图中我们省略了。

②接着，Main类调用Directory类的accept方法。这时传递的参数是ListVisitor的实例，但我们在本图中省略了。

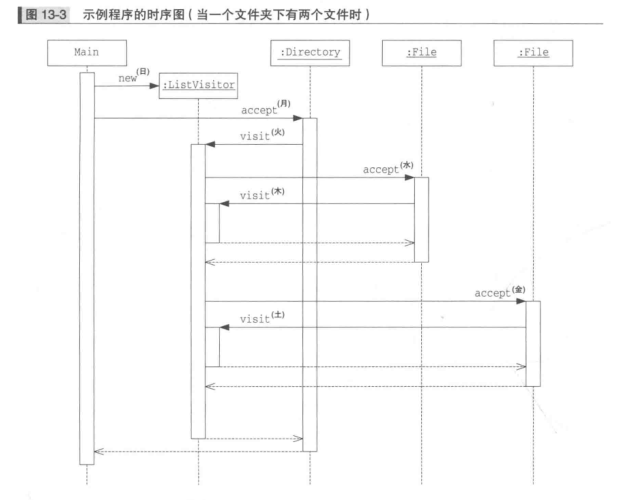
③Directory类的实例调用接收到的参数ListVisitor的visit（Directory）方法。

④接下来，ListVisitor类的实例会访问文件夹，并调用找到的第一个文件的accept方法。传递的参数是自身（this）。

⑤Fi1e的实例调用接收到的参数ListVisitor的visit（File）方法。请注意，这时Listvisitor的visit（Directory）.还在执行中（并非多线程执行，而是表示visit（Directory）还存在于调用堆栈（callstack）中的意思。在时序图中，表示生命周期的长方形的右侧发生了重叠就说明了这一点）。

⑥从visit（File）返回到accept，接着又从accept也返回出来，然后调用另外一个Fi1e的实例（同一文件夹下的第二个文件）的accept方法。传递的参数是Listvisitor的实例this。

⑦与前面一样，File的实例调用visit（File）方法。所有的处理完成后，逐步返回，最后回到Main类中的调用accept方法的地方。



在阅读时序图时，请大家注意以下几点。

·对于Directory类的实例和File类的实例，我们调用了它们的accept方法

·对于每一个Directory类的实例和File类的实例，我们只调用了一次它们的accept方法

·对于ListVisitor的实例，我们调用了它的visit（Directory）和visit（File）方法

·处理visit（Directory）和visit（File）的是同一个Listvisitor的实例通过上面的学习大家应该明白了吧。在Visitor模式中，visit方法将“处理”都集中在ListVisitor里面了。

|13.3 Visitor模式中的登场角色

在Visitor模式中有以下登场角色。

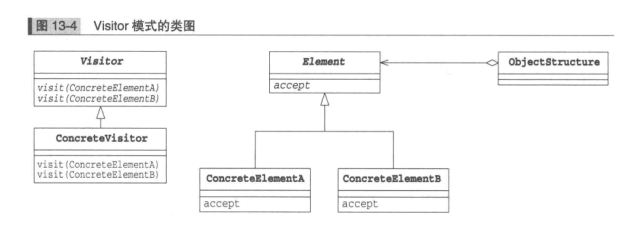
Visitor（访问者）Visitor角色负责对数据结构中每个具体的元素（ConcreteElement角色）声明一个用于访问xXxxx的visit（xxxxx）方法。visit（xxxxx）是用于处理XXXXX的方法，负责实现该方法的是ConcreteVisitor角色。在示例程序中，由Visitor类扮演此角色。

◆ConcreteVisitor（具体的访问者）ConcreteVisitor角色负责实现Visitor角色所定义的接口（APl）。它要实现所有的visit（xxxxx）方法，即实现如何处理每个ConcreteElement角色。在示例程序中，由ListVisitor类扮演此角色。如同在ListVisitor中，currentdir字段的值不断发生变化一样，随着visit（xxxxx）处理的进行，ConcreteVisitor角色的内部状态也会不断地发生变化。

◆Element（元素）Element角色表示Visitor角色的访问对象。它声明了接受访问者的accept方法。accept方法接收到的参数是Visitor角色。在示例程序中，由Element接口扮演此角色。

◆ConcreteElement ConcreteElement角色负责实现Element角色所定义的接口（APl））。在示例程序中，由File类和Directory类扮演此角色。

·ObjectStructure（对象结构）ObjectStructur角色负责处理Element角色的集合。ConcreteVisitor角色为每个Element角色都准备了处理方法。在示例程序中，由Directory类扮演此角色（一人分饰两角）。为了让ConcreteVisitor 角色可以遍历处理每个Element角色，在示例程序中，我们在Directory类中实现了iterator方法。



13.4拓展思路的要点

|双重分发

我们来整理一下Visitor模式中方法的调用关系吧。

accept（接受）方法的调用方式如下。

element.accept（visitor）；而visit（访问）方法的调用方式如下。

visitor.visit（element）；对比一下这两个方法会发现，它们是相反的关系。element接受visitor，而visitor又访问element。

在Visitor模式中，ConcreteElement和ConcreteVisitor这两个角色共同决定了实际进行的处理。

这种消息分发的方式一般被称为双重分发（double dispatch）。

为什么要弄得这么复杂

当看到上面的处理流程后，大家可能会感觉到“Visitor模式不是把简单问题复杂化了吗？”“如果需要循环处理，在数据结构的类中直接编写循环语句不就解决了吗？为什么要搞出accept方法和visit方法之间那样复杂的调用关系呢？”

Visitor模式的目的是将处理从数据结构中分离出来。数据结构很重要，它能将元素集合和关联在一起。但是，需要注意的是，保存数据结构与以数据结构为基础进行处理是两种不同的东西。

在示例程序中，我们创建了ListVisitor类作为显示文件夹内容的ConcreteVisitor角色。此外，在练习题中，我们还要编写进行其他处理的ConcreteVisitor角色。通常，ConcreteVisitor角色的开发可以独立于File类和Directory类。也就是说，Visitor模式提高了File类和Directory类作为组件的独立性。如果将进行处理的方法定义在File类和Directory类中，当每次要扩展功能，增加新的“处理”时，就不得不去修改Fi1e类（代码清单13-4）和Directory类（代码清单13-5）。

|开闭原则——对扩展开放，对修改关闭

既然谈到了功能扩展和修改，那就顺带谈一谈开闭原则（The Open-Closed Principle，OCP）。该原则是勃兰特·梅耶提出的，而后RobertC.Martin在C++Report（1996年1月）中的Engineering NoteBook专栏中对其进行了总结”。

该原则主张类应当是下面这样的。

·对扩展（extension）是开放（open）的

·对修改（modification）是关闭（close）的在设计类时，若无特殊理由，必须要考虑到将来可能会扩展类。绝不能毫无理由地禁止扩展类。这就是“对扩展是开放的”的意思。

但是，如果在每次扩展类时都需要修改现有的类就太麻烦了。所以我们需要在不用修改现有类的前提下能够扩展类，这就是“对修改是关闭的”的意思。

我们提倡扩展，但是如果需要修改现有代码，那就不行了。在不修改现有代码的前提下进行扩展，这就是开闭原则。

至此大家已经学习了多种设计模式。那么在看到这条设计原则后，大家应该都会点头表示赞同吧。

功能需求总是在不断变化，而且这些功能需求大都是“希望扩展某个功能”。因此，如果不能比较容易地扩展类，开发过程将会变得非常困难。另一方面，如果要修改已经编写和测试完成的类，又可能会导致软件产品的质量降低。

对扩展开放、对修改关闭的类具有高可复用性，可作为组件复用。设计模式和面向对象的目的正是为我们提供一种结构，可以帮助我们设计出这样的类。

易于增加ConcreteVisitor角色

使用Visitor模式可以很容易地增加ConcreteVisitor角色。因为具体的处理被交给ConcreteVisitor角色负责，因此完全不用修改ConcreteElement角色。

|难以增加ConcreteElement角色

虽然使用Visitor模式可以很容易地增加ConcreteVisitor角色，不过它却很难应对ConcreteElement角色的增加。

例如，假设现在我们要在示例程序中增加Entry类的子类Device类。也就是说，Device类是Ei1e类和Directory类的兄弟类。这时，我们需要在Visitor类中声明一个visit（Device）

方法，并在所有的Visitor类的子类中都实现这个方法。

|Visitor工作所需的条件

“在Visitor模式中，对数据结构中的元素进行处理的任务被分离出来，交给Visitor类负责。

这样，就实现了数据结构与处理的分离”这个主题，我们在本章的学习过程中已经提到过很多次了。但是要达到这个目的是有条件的，那就是Element角色必须向Visitor角色公开足够多的信息。例如，在示例程序中，visit（Directory）方法需要调用每个目录条目的accept方法。为此，Directory类必须提供用于获取每个目录条目的iterator方法。

访问者只有从数据结构中获取了足够多的信息后才能工作。如果无法获取到这些信息，它就无法工作。这样做的缺点是，如果公开了不应当被公开的信息，将来对数据结构的改良就会变得非常困难。

|13.5相关的设计模式

◆lterator模式（第1章）Iterator模式和Visitor模式都是在某种数据结构上进行处理。

Iterator模式用于逐个遍历保存在数据结构中的元素。

Visitor模式用于对保存在数据结构中的元素进行某种特定的处理。

·Composite模式（第11章）有时访问者所访问的数据结构会使用Composite模式。

◆Interpreter模式（第23章）在Interpreter模式中，有时会使用Visitor模式。例如，在生成了语法树后，可能会使用Visitor模式访问语法树的各个节点进行处理。