

## 68 | 访问者模式（上）：手把手带你还原访问者模式诞生的思维过程

2020-04-08 王争

设计模式之美

[进入课程 >](#)



讲述：冯永吉

时长 09:10 大小 8.41M



前面我们讲到，大部分设计模式的原理和实现都很简单，不过也有例外，比如今天要讲的访问者模式。它可以算是 23 种经典设计模式中最难理解的几个之一。因为它难理解、难实现，应用它会导致代码的可读性、可维护性变差，所以，访问者模式在实际的软件开发中很少被用到，在没有特别必要的情况下，建议你不要使用访问者模式。

尽管如此，为了让你以后读到应用了访问者模式的代码的时候，能一眼就能看出代码的设计意图，同时为了整个专栏内容的完整性，我觉得还是有必要给你讲一讲这个模式。此外，为了最大化学习效果，我今天不只是单纯地讲解原理和实现，更重要的是，我会手把手带你还原访问者模式诞生的思维过程，让你切身感受到创造一种新的设计模式出来并不是件难事。



话不多说，让我们正式开始今天的学习吧！

## 带你“发明”访问者模式

假设我们从网站上爬取了很多资源文件，它们的格式有三种：PDF、PPT、Word。我们现在要开发一个工具来处理这批资源文件。这个工具的其中一个功能是，把这些资源文件中的文本内容抽取出来放到 txt 文件中。如果让你来实现，你会怎么来做呢？

实现这个功能并不难，不同的人有不同的写法，我将其中一种代码实现方式贴在这里。其中，ResourceFile 是一个抽象类，包含一个抽象函数 extract2txt()。PdfFile、PPTFile、WordFile 都继承 ResourceFile 类，并且重写了 extract2txt() 函数。在 ToolApplication 中，我们可以利用多态特性，根据对象的实际类型，来决定执行哪个方法。

 复制代码

```
1 public abstract class ResourceFile {
2     protected String filePath;
3
4     public ResourceFile(String filePath) {
5         this.filePath = filePath;
6     }
7
8     public abstract void extract2txt();
9 }
10
11 public class PPTFile extends ResourceFile {
12     public PPTFile(String filePath) {
13         super(filePath);
14     }
15
16     @Override
17     public void extract2txt() {
18         //...省略一大坨从PPT中抽取文本的代码...
19         //...将抽取出来的文本保存在跟filePath同名的.txt文件中...
20         System.out.println("Extract PPT.");
21     }
22 }
23
24 public class PdfFile extends ResourceFile {
25     public PdfFile(String filePath) {
26         super(filePath);
27     }
28
29     @Override
30     public void extract2txt() {
31         //...
```

```

32     System.out.println("Extract PDF.");
33 }
34 }
35
36 public class WordFile extends ResourceFile {
37     public WordFile(String filePath) {
38         super(filePath);
39     }
40
41     @Override
42     public void extract2txt() {
43         //...
44         System.out.println("Extract WORD.");
45     }
46 }
47
48 // 运行结果是:
49 // Extract PDF.
50 // Extract WORD.
51 // Extract PPT.
52 public class ToolApplication {
53     public static void main(String[] args) {
54         List<ResourceFile> resourceFiles = listAllResourceFiles(args[0]);
55         for (ResourceFile resourceFile : resourceFiles) {
56             resourceFile.extract2txt();
57         }
58     }
59
60     private static List<ResourceFile> listAllResourceFiles(String resourceDirectory) {
61         List<ResourceFile> resourceFiles = new ArrayList<>();
62         //...根据后缀(pdf/ppt/word)由工厂方法创建不同的类对象(PdfFile/PPTFile/WordFile)
63         resourceFiles.add(new PdfFile("a.pdf"));
64         resourceFiles.add(new WordFile("b.word"));
65         resourceFiles.add(new PPTFile("c.ppt"));
66         return resourceFiles;
67     }
68 }

```


如果工具的功能不停地扩展，不仅要能抽取文本内容，还要支持压缩、提取文件元信息（文件名、大小、更新时间等等）构建索引等一系列的功能，那如果我们继续按照上面的实现思路，就会存在这样几个问题：

违背开闭原则，添加一个新的功能，所有类的代码都要修改；

虽然功能增多，每个类的代码都不断膨胀，可读性和可维护性都变差了；

把所有比较上层的业务逻辑都耦合到 PdfFile、PPTFile、WordFile 类中，导致这些类的职责不够单一，变成了大杂烩。

针对上面的问题，我们常用的解决方法就是拆分解耦，把业务操作跟具体的数据结构解耦，设计成独立的类。这里我们按照访问者模式的演进思路来对上面的代码进行重构。重构之后的代码如下所示。

 复制代码

```
1 public abstract class ResourceFile {
2     protected String filePath;
3     public ResourceFile(String filePath) {
4         this.filePath = filePath;
5     }
6 }
7
8 public class PdfFile extends ResourceFile {
9     public PdfFile(String filePath) {
10         super(filePath);
11     }
12     //...
13 }
14 //...PPTFile、WordFile代码省略...
15 public class Extractor {
16     public void extract2txt(PPTFile pptFile) {
17         //...
18         System.out.println("Extract PPT.");
19     }
20
21     public void extract2txt(PdfFile pdfFile) {
22         //...
23         System.out.println("Extract PDF.");
24     }
25
26     public void extract2txt(WordFile wordFile) {
27         //...
28         System.out.println("Extract WORD.");
29     }
30 }
31
32 public class ToolApplication {
33     public static void main(String[] args) {
34         Extractor extractor = new Extractor();
35         List<ResourceFile> resourceFiles = listAllResourceFiles(args[0]);
36         for (ResourceFile resourceFile : resourceFiles) {
37             extractor.extract2txt(resourceFile);
38         }
39     }
40 }
```

```

41     private static List<ResourceFile> listAllResourceFiles(String resourceDirect
42         List<ResourceFile> resourceFiles = new ArrayList<>();
43         //...根据后缀(pdf/ppt/word)由工厂方法创建不同的类对象(PdfFile/PPTFile/WordFile)
44         resourceFiles.add(new PdfFile("a.pdf"));
45         resourceFiles.add(new WordFile("b.word"));
46         resourceFiles.add(new PPTFile("c.ppt"));
47         return resourceFiles;
48     }
49 }

```

这其中最关键的一点设计是，我们把抽取文本内容的操作，设计成了三个重载函数。函数重载是 Java、C++ 这类面向对象编程语言中常见的语法机制。所谓重载函数是指，在同一类中函数名相同、参数不同的一组函数。

不过，如果你足够细心，就会发现，上面的代码是编译通过不了的，第 37 行会报错。这是为什么呢？

我们知道，多态是一种动态绑定，可以在运行时获取对象的实际类型，来运行实际类型对应的方法。而函数重载是一种静态绑定，在编译时并不能获取对象的实际类型，而是根据声明类型执行声明类型对应的方法。

在上面代码的第 35 ~ 38 行中，resourceFiles 包含的对象的声明类型都是 ResourceFile，而我们并没有在 Extractor 类中定义参数类型是 ResourceFile 的 extract2txt() 重载函数，所以在编译阶段就通过不了，更别说在运行时根据对象的实际类型执行不同的重载函数了。那如何解决这个问题呢？

解决的办法稍微有点难理解，我们先来看代码，然后我再来给你慢慢解释。

 复制代码

```

1  public abstract class ResourceFile {
2      protected String filePath;
3      public ResourceFile(String filePath) {
4          this.filePath = filePath;
5      }
6      abstract public void accept(Extractor extractor);
7  }
8
9  public class PdfFile extends ResourceFile {
10     public PdfFile(String filePath) {
11         super(filePath);

```

```

12     }
13
14     @Override
15     public void accept(Extractor extractor) {
16         extractor.extract2txt(this);
17     }
18
19     //...
20 }
21
22 //...PPTFile、WordFile跟PdfFile类似, 这里就省略了...
23 //...Extractor代码不变...
24
25 public class ToolApplication {
26     public static void main(String[] args) {
27         Extractor extractor = new Extractor();
28         List<ResourceFile> resourceFiles = listAllResourceFiles(args[0]);
29         for (ResourceFile resourceFile : resourceFiles) {
30             resourceFile.accept(extractor);
31         }
32     }
33
34     private static List<ResourceFile> listAllResourceFiles(String resourceDirect
35         List<ResourceFile> resourceFiles = new ArrayList<>();
36         //...根据后缀(pdf/ppt/word)由工厂方法创建不同的类对象(PdfFile/PPTFile/WordFile)
37         resourceFiles.add(new PdfFile("a.pdf"));
38         resourceFiles.add(new WordFile("b.word"));
39         resourceFiles.add(new PPTFile("c.ppt"));
40         return resourceFiles;
41     }
42 }

```

在执行第 30 行的时候，根据多态特性，程序会调用实际类型的 accept 函数，比如 PdfFile 的 accept 函数，也就是第 16 行代码。而 16 行代码中的 this 类型是 PdfFile 的，在编译的时候就确定了，所以会调用 extractor 的 extract2txt(PdfFile pdfFile) 这个重载函数。这个实现思路是不是很有技巧？这是理解访问者模式的关键所在，也是我之前所说的访问者模式不好理解的原因。

现在，如果要继续添加新的功能，比如前面提到的压缩功能，根据不同的文件类型，使用不同的压缩算法来压缩资源文件，那我们该如何实现呢？我们需要实现一个类似 Extractor 类的新类 Compressor 类，在其中定义三个重载函数，实现对不同类型资源文件的压缩。除此之外，我们还要在每个资源文件类中定义新的 accept 重载函数。具体的代码如下所示：



```


1  public abstract class ResourceFile {
2      protected String filePath;
3      public ResourceFile(String filePath) {
4          this.filePath = filePath;
5      }
6      abstract public void accept(Extractor extractor);
7      abstract public void accept(Compressor compressor);
8  }
9
10 public class PdfFile extends ResourceFile {
11     public PdfFile(String filePath) {
12         super(filePath);
13     }
14
15     @Override
16     public void accept(Extractor extractor) {
17         extractor.extract2txt(this);
18     }
19
20     @Override
21     public void accept(Compressor compressor) {
22         compressor.compress(this);
23     }
24
25     //...
26 }
27
28 //...PPTFile、WordFile跟PdfFile类似, 这里就省略了...
29 //...Extractor代码不变
30
31 public class ToolApplication {
32     public static void main(String[] args) {
33         Extractor extractor = new Extractor();
34         List<ResourceFile> resourceFiles = listAllResourceFiles(args[0]);
35         for (ResourceFile resourceFile : resourceFiles) {
36             resourceFile.accept(extractor);
37         }
38
39         Compressor compressor = new Compressor();
40         for (ResourceFile resourceFile : resourceFiles) {
41             resourceFile.accept(compressor);
42         }
43     }
44
45     private static List<ResourceFile> listAllResourceFiles(String resourceDirect
46         List<ResourceFile> resourceFiles = new ArrayList<>();
47         //...根据后缀(pdf/ppt/word)由工厂方法创建不同的类对象(PdfFile/PPTFile/WordFile)
48         resourceFiles.add(new PdfFile("a.pdf"));
49         resourceFiles.add(new WordFile("b.word"));
50         resourceFiles.add(new PPTFile("c.ppt"));

```

```
51     return resourceFiles;
52 }
53 }
```

上面代码还存在一些问题，添加一个新的业务，还是需要修改每个资源文件类，违反了开闭原则。针对这个问题，我们抽象出来一个 Visitor 接口，包含三个命名非常通用的 visit() 重载函数，分别处理三种不同类型的资源文件。具体做什么业务处理，由实现这个 Visitor 接口的具体的类来决定，比如 Extractor 负责抽取文本内容，Compressor 负责压缩。当我们新添加一个业务功能的时候，资源文件类不需要做任何修改，只需要修改 ToolApplication 的代码就可以了。

按照这个思路我们可以对代码进行重构，重构之后的代码如下所示：

 复制代码

```
1  public abstract class ResourceFile {
2      protected String filePath;
3      public ResourceFile(String filePath) {
4          this.filePath = filePath;
5      }
6      abstract public void accept(Visitor visitor);
7  }
8
9  public class PdfFile extends ResourceFile {
10     public PdfFile(String filePath) {
11         super(filePath);
12     }
13
14     @Override
15     public void accept(Visitor visitor) {
16         visitor.visit(this);
17     }
18
19     //...
20 }
21 //...PPTFile、WordFile跟PdfFile类似，这里就省略了...
22
23 public interface Visitor {
24     void visit(PdfFile pdfFile);
25     void visit(PPTFile pdfFile);
26     void visit(WordFile pdfFile);
27 }
28
29 public class Extractor implements Visitor {
30     @Override
31     public void visit(PPTFile pptFile) {
```



```
32     //...
33     System.out.println("Extract PPT.");
34 }
35
36 @Override
37 public void visit(PdfFile pdfFile) {
38     //...
39     System.out.println("Extract PDF.");
40 }
41
42 @Override
43 public void visit(WordFile wordFile) {
44     //...
45     System.out.println("Extract WORD.");
46 }
47 }
48
49 public class Compressor implements Visitor {
50     @Override
51     public void visit(PPTFile pptFile) {
52         //...
53         System.out.println("Compress PPT.");
54     }
55
56     @Override
57     public void visit(PdfFile pdfFile) {
58         //...
59         System.out.println("Compress PDF.");
60     }
61
62     @Override
63     public void visit(WordFile wordFile) {
64         //...
65         System.out.println("Compress WORD.");
66     }
67 }
68
69
70 public class ToolApplication {
71     public static void main(String[] args) {
72         Extractor extractor = new Extractor();
73         List<ResourceFile> resourceFiles = listAllResourceFiles(args[0]);
74         for (ResourceFile resourceFile : resourceFiles) {
75             resourceFile.accept(extractor);
76         }
77
78         Compressor compressor = new Compressor();
79         for (ResourceFile resourceFile : resourceFiles) {
80             resourceFile.accept(compressor);
81         }
82     }
83 }
```

```
84     private static List<ResourceFile> listAllResourceFiles(String resourceDirecti
85         List<ResourceFile> resourceFiles = new ArrayList<>();
86         //...根据后缀(pdf/ppt/word)由工厂方法创建不同的类对象(PdfFile/PPTFile/WordFile)
87         resourceFiles.add(new PdfFile("a.pdf"));
88         resourceFiles.add(new WordFile("b.word"));
89         resourceFiles.add(new PPTFile("c.ppt"));
90         return resourceFiles;
91     }
92 }
```

## 重新来看访问者模式

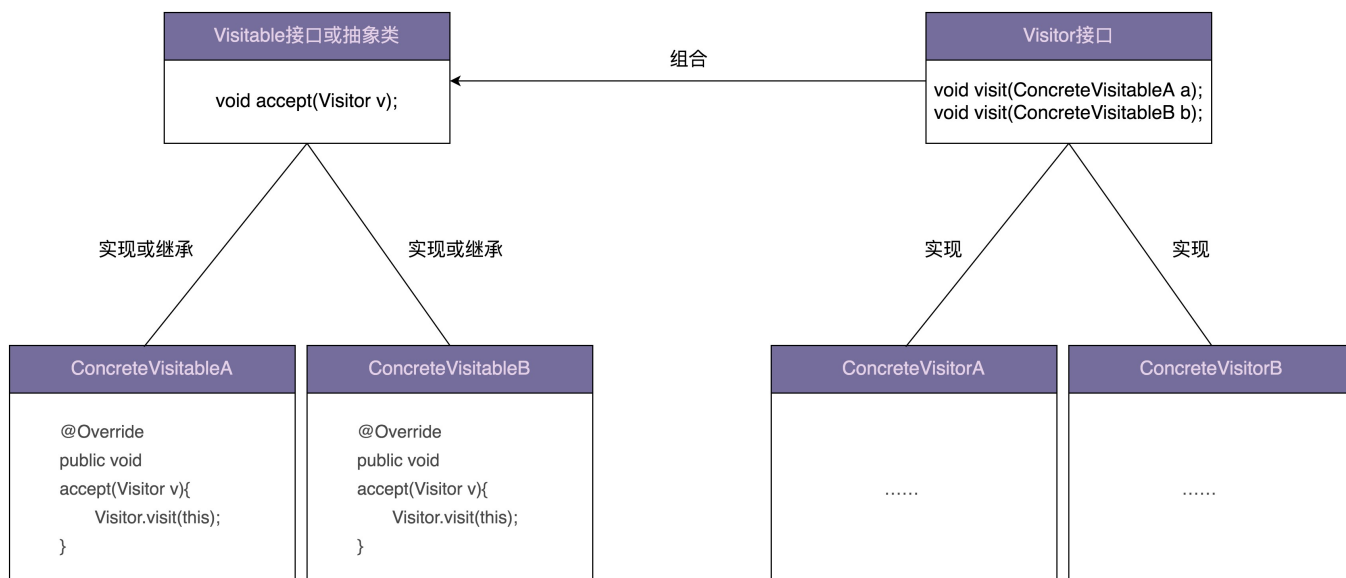
刚刚我带你一步一步还原了访问者模式诞生的思维过程，现在，我们回过头来总结一下，这个模式的原理和代码实现。

访问者者模式的英文翻译是 Visitor Design Pattern。在 GoF 的《设计模式》一书中，它是这么定义的：

Allows for one or more operation to be applied to a set of objects at runtime, decoupling the operations from the object structure.

翻译成中文就是：允许一个或者多个操作应用到一组对象上，解耦操作和对象本身。

定义比较简单，结合前面的例子不难理解，我就不过多解释了。对于访问者模式的代码实现，实际上，在上面例子中，经过层层重构之后的最终代码，就是标准的访问者模式的实现代码。这里，我又总结了一张类图，贴在了下面，你可以对照着前面的例子代码一块儿来看一下。



极客时间

最后，我们再来看下，访问者模式的应用场景。

一般来说，访问者模式针对的是一组类型不同的对象（PdfFile、PPTFile、WordFile）。不过，尽管这组对象的类型是不同的，但是，它们继承相同的父类（ResourceFile）或者实现相同的接口。在不同的应用场景下，□我们需要对这组对象进行一系列不相关的业务操作（抽取文本、压缩等），但为了避免不断添加功能导致类（PdfFile、PPTFile、WordFile）不断膨胀，职责越来越不单一，以及避免频繁地添加功能导致的频繁代码修改，我们使用访问者模式，将对象与操作解耦，将这些业务操作抽离出来，定义在独立细分的访问者类（Extractor、Compressor）中。

## 重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

访问者模式允许一个或者多个操作应用到一组对象上，设计意图是解耦操作和对象本身，保持类职责单一、满足开闭原则以及应对代码的复杂性。

对于访问者模式，学习的主要难点在代码实现。而代码实现比较复杂的主要原因是，函数重载在大部分面向对象编程语言中是静态绑定的。也就是说，调用类的哪个重载函数，是在编译期间，由参数的声明类型决定的，而非运行时，根据参数的实际类型决定的。

正是因为代码实现难理解，所以，在项目中应用这种模式，会导致代码的可读性比较差。如果你的同事不了解这种设计模式，可能会读不懂、维护不了你写的代码。所以，除非不得

已，不要使用这种模式。

## 课堂讨论

实际上，今天举的例子不用访问者模式也可以搞定，你能够想到其他实现思路吗？

欢迎留言和我分享你的想法。如果有收获，也欢迎你把这篇文章分享给你的朋友。

更多活动推荐



39元惊喜福袋

iPad Pro、机械键盘、免费课程等你来拿

仅限 3 天

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 67 | 迭代器模式（下）：如何设计实现一个支持“快照”功能的iterator？

下一篇 加餐二 | 设计模式、重构、编程规范等相关书籍推荐

## 精选留言 (5)

写留言



Monday

2020-04-08

独立细分的访问者类（Extractor、Compressor），这些类分别对应所有类型对象某一操作的实现，如果类型多了，这些访问者类也会爆炸。

展开



3



**小晏子**

2020-04-08

课后思考：可以使用策略模式，对于不同的处理方式定义不同的接口，然后接口中提供对于不同类型文件的实现，再使用静态工厂类保存不同文件类型和不同处理方法的映射关系。对于后续扩展的新增文件处理方法，比如composer，按同样的方式实现一组策略，然后修改application代码使用对应的策略。

展开 ∨



👍 2



**Liam**

2020-04-08

antlr(编译器框架) 对语法树进行解析的时候就是通过visitor模式实现了扩展



👍 1



**Frank**

2020-04-08

打卡 今日学习访问者设计模式，收获如下：

访问者模式表示允许一个或者多个操作应用到一组对象上，解耦操作和对象本身。体现了SRP原则。这个原则的代码实现比较复杂，关键要理解“函数重载是一种静态绑定，在编译时并不能获取对象的实际类型，而是根据声明类型执行声明类型对应的方法”。

文章的例子通过4个版本的迭代，从一个最小原型实现，逐渐重构成一个符合访问者...  
展开 ∨



👍 1



**L**

2020-04-08

课后题，策略模式+抽象工厂模式

展开 ∨

