【设计模式】趣说访问者模式，颇有些无奈之举

原创 编程新说李新杰 编程新说 2019-07-22

老实说，在实际编程中，访问者设计模式应用的并不多，至少我是这样认为的，因为它的主要使用场景并不多。

那么肯定会有人问，访问者模式的主要使用场景是什么呢？继续往下看便知。

新闻联播看多了之后

首先要说的是，设计模式中的“访问者”和现实生活中的“访问者”其本质是一回事。虽然设计模式中的不太熟悉，但现实生活中的再熟悉不过了。

我在以前的文章中多次提到过，有时站在现实生活的角度看待某些技术点反而会更容易看清楚，那照例还是从生活中的事情说起吧。

说起访问者，我能够想到最高大上的，莫过于国家领导人的国事访问。以中方访问美方来说吧，这里面的大致内容（我猜）应该是这样的：

中方专机什么时间在哪个机场降落，美方派谁在机场迎接，然后就是欢迎仪式和欢迎晚宴，接着就是会见哪些人，开哪些会议，签署哪些文件，参观哪些地方等等，最后就是结束访问启程回国。

当然这些内容肯定是双方外交部门都提前沟通好的。但是仍然是在美方的安排下一步一步进行，因为我们是作为访客的身份，人家是东道主，要尽地主之谊的。（虽然老美不是什么好东西）

比如人家安排的吃牛排，那我们就吃吧。我们总不能要求他们改成“主席套餐”吧，再说他们的厨师也搞定啊。

等到老美的总统来我们国家访问的时候，他们就成了访客了，我们就是东道主，整体就得由我们来安排。让他喝稀饭他就不能吃大米，不喜欢吃就晚上自己回酒店泡方便面，哈哈哈哈。

如果把这个事情抽象一下就是，一方在另一方的安排下，逐步有序的做一些事情。

自己想体验的话就来个这吧

国事访问这事啊离我们太遥远了，那就再看个和我们息息相关的吧。没错，就是旅游。无论是国内游还是出境游，其实差别不大，大致内容应该是这样的：

先报好旅行社，在指定的时间乘坐交通工具到达目的地后，会有一辆大巴车拉着我们，按照行程开始去景点，去吃饭，去酒店等等。

我们什么都不用操心，跟着走就行了，因为旅行社和导游都安排好了。再说了，即我们使有意见，导游也不会听我们的。

颇令人讨厌的可能就是逛购物店了，但是没办法，因为协议已经签了。我们有义务进购物店，听相关人员讲解，想买的就买，不想买的随便看看，但是不能提前出去。

其实还有更坑的，那就是导游在大巴车上强行收费，说些很难听的话，甚至骂人/威胁。尤其是在境外，他们觉得此生很难再见面，有时话说的特别难听。

所以整个行程下来，既有高兴的时候，也有心烦的时候。导游给我们讲解当地历史

的时候，觉得他是“好人”，领我们进购物店时，又觉得他是“坏人”。

其实都不是，他是“工人”，一个从事旅游行业工作的人。一个需要养家糊口的人，跟我们没啥区别。我是不是很善解人意啊。

如果把这个事情也抽象一下就是，必须按照既定的规则走完所有事项，如果对某个事项关心，那就积极的去获取自己想要的信息，如果对某个事项不关心，那就默默的跟随即可，什么都不用做，但是不准离开。

做一个善于思考总结的人

我想说的是既然报团游有如此多的问题，为什么还有那么多人报团，而不选自由行呢。答案是显而易见的。

以出境游来说，当你达到国外，人生地不熟，语言又不通，很多事情的推进会特别艰难。

当然也可以提前研究攻略，制定好路线，订好酒店机票等。但是旅游主要是想放松一下，为了出国一周，在家看三个月攻略，岂不是更累吗？

总结一下，当你对所要做的事情完全不了解，而且想要了解的话需花费很大的精力时，只能选择第三方给出的方案，虽然明知里面可能会有坑。

对于像国事访问的，因为有许多礼仪礼节或规则约束需要遵守，所以一般也都听从东道主的安排。

虽然出访和旅游这两件事的本质完全不一样，但是他们的宏观进行模式却基本一致。

到此我们已经讲了两件事，两个特点，两个原因。请仔细体会下。看起来有点让人不爽，但又颇有些无奈。

这两件事都是站在“访问者”的立场来说的，下面从多角度来看下。

从一个具体的示例说起

假如小明在北京工作多年，对北京非常熟悉。他的朋友小白来找他玩，而且是第一次来北京，打算去一些有名的景点。

在这件事中，小明就是东道主，小白就是访客。其实就是一方带另一方参观嘛。

站在东道主的角度，他要安排访客参观景点，所以是这样的：

```
/**
 * <p>东道主
 */
public interface Host {

    //带朋友去故宫
    void show(PalaceMuseum PalaceMuseum, Guest guest);

    //带朋友去长城
    void show(GreatWall GreatWall, Guest guest);

    //带朋友去颐和园
    void show(SummerPalace SummerPalace, Guest guest);
}
```

站在访客的角度，他是要参观景点的，所以是这样的：

```
/**
 * <p>客人
```

```
 */
public interface Guest {

    //看故宫
    void look(PalaceMuseum PalaceMuseum);

    //看长城
    void look(GreatWall GreatWall);

    //看颐和园
    void look(SummerPalace SummerPalace);
}
```

站在景点的角度，它是要接受访客的参观的，所以是这样的：

```
/**
 * <p>故宫
 */
public interface PalaceMuseum {

    //让访客看
    void accept(Guest guest);
}

/**
 * <p>长城
 */
public interface GreatWall {

    //让访客看
    void accept(Guest guest);
}

/**
 * <p>颐和园
 */
public interface SummerPalace {

    //让访客看
    void accept(Guest guest);
}
```

在这个事件中共有三种角色，它们的职责、目的和作用都非常清晰：

小明：东道主，职责是负责协助

小白：访客，目的是欣赏景点的景色

景点：被欣赏者，作用是提供景色

这就是一个访问者的模型，我们把它抽象并一般化，发现这是一个固定的套路或模式，称之为访问者模式。

在访问者模式中，共有三方参与者，它们的分工非常明确：

一方：访问者，获取信息的人

二方：被访问者，提供信息的人

三方：协调者，安排一二双方进行交互的人

可以这样来理解三方的定位，一方是购买者（出钱），二方是提供者（出力），三方是协调者（和稀泥）。哈哈。

注意，这里的一方二方三方都是访问者模式内部的概念，它们是一家人或一个单位的。

换个角度来看就是，访问者在协调者制定的规则下完成对被访问者的访问，期间获取关心的信息，忽略不关心的信息。

把访问者模式放到一个宏观应用中，应该是这样的：

用户程序->|访问者->协调者->被访问者|->底层复杂数据

访问者模式的推导

对于设计模式，一定要活学活用，不能拘泥于GOF。一定要按自己的场景需求来用，不能死搬硬套。

在访问者模式中，通常把被访问者称为元素，访问者自然还是访问者，抽象一下：

```
//元素
public interface Element<V extends Visitor> {

    //接受访问者
    void accept(V visitor);
}

//访问者
public interface Visitor<E extends Element> {

    //访问元素
    void visit(E element);
}
```

请注意这里的泛型。

然后再抽象一下协调者：

```
public interface ObjectStructure {

    //所有元素
    List<Element> getElements();

    //所有访问者
    List<Visitor> getVisitors();
}
```

协调者拥有所有的元素和所有的访问者，它可以自己来实现访问规则，使访问者完成对元素的访问。

注意，我的抽象和GOF的不完全一样，因为前面已经说了要活学活用嘛。

那究竟是一个访问者访问一个元素，还是一个访问者访问多个元素，仍然是没有标准答案，应该根据实际情况来定。

比如大公司，一岗（位）一（个）人，事情做得精细。小公司，多岗（位）一（个）人，办事效率高。各有千秋，适合的才是最好的。

下面给出一个访问者访问一个元素的情况。

访问者A访问元素A：

```
//元素A
public interface ElementA extends Element<VisitorA> {

    @Override
    void accept(VisitorA visitor);
}

//访问者A
public interface VisitorA extends Visitor<ElementA> {

    @Override
    void visit(ElementA element);
}
```

由于上面使用了泛型，这里的方法参数可以换为精确的类型。

访问者B访问元素B：

```
//元素B
public interface ElementB extends Element<VisitorB> {

    @Override
    void accept(VisitorB visitor);
}

//访问者B
public interface VisitorB extends Visitor<ElementB> {

    @Override
    void visit(ElementB element);
}
```

后续就按照这个模式去扩展即可。有新的数据需要访问时，就添加新的元素和新的访问者，同时还可能需要修改协调者。

作者想要说的话

我们经常听到说，要学习西方模式，XX模式，YY模式等，其实主要是学习人家的理念和思想，而不是照抄，因为照抄有水土不服的问题。

所以，每一个开发人员或设计人员都不应该直接照抄GOF的设计模式。特别是为了使用而使用，就更没意思了。

访问者模式的主要应用场景之一就是，底层数据过于复杂，是的，过于复杂，上层应用无法直接访问。

如Java的字节码文件，我们的应用程序根本就无法直接访问。

还有一种就是不想让别人随意访问，可以通过访问者模式去约束访问者访问的方式。

比如我在大四时就去参观过汽车制造车间，由专人领着我们按照路线行走，因为随意乱跑不安全嘛。

访问者模式的中心思想就是，协调者制定好合理的规则，访问者按照规则进行访问，从自己关心的被访者上获取需要的数据信息。

当然也可以采用被动式的，协调者按照一定规则，把被访问者的信息逐步推送给访问者，访问者根据自己的需要来选择保存或忽略。

最后，设计模式是一种理念，一种思想，需要去思考，去领悟。

程序代码：

https://github.com/coding-new-talking/java-code-demo.git

（END）

作者是工作超过**10年**的码农，现在任架构师。喜欢研究技术，崇尚简单快乐。**追求以通俗易懂的语言解说技术，希望所有的读者都能看懂并记住。**下面是公众号和知识星球的二维码，欢迎关注！