命令模式

2020年2月21日 11:06

1. 概念

将一个请求封装为一个对象，使发出请求的责任和执行请求的责任分割开。这样两者之间通过命令对象进行沟通，这样方便将命令对象进行储存、传递、调用、增加与管理。
使得发送者和接收者完全解耦，发送者只知道发送命令，不需知道如何完成请求，接收者也是类似。

2. UML接口图

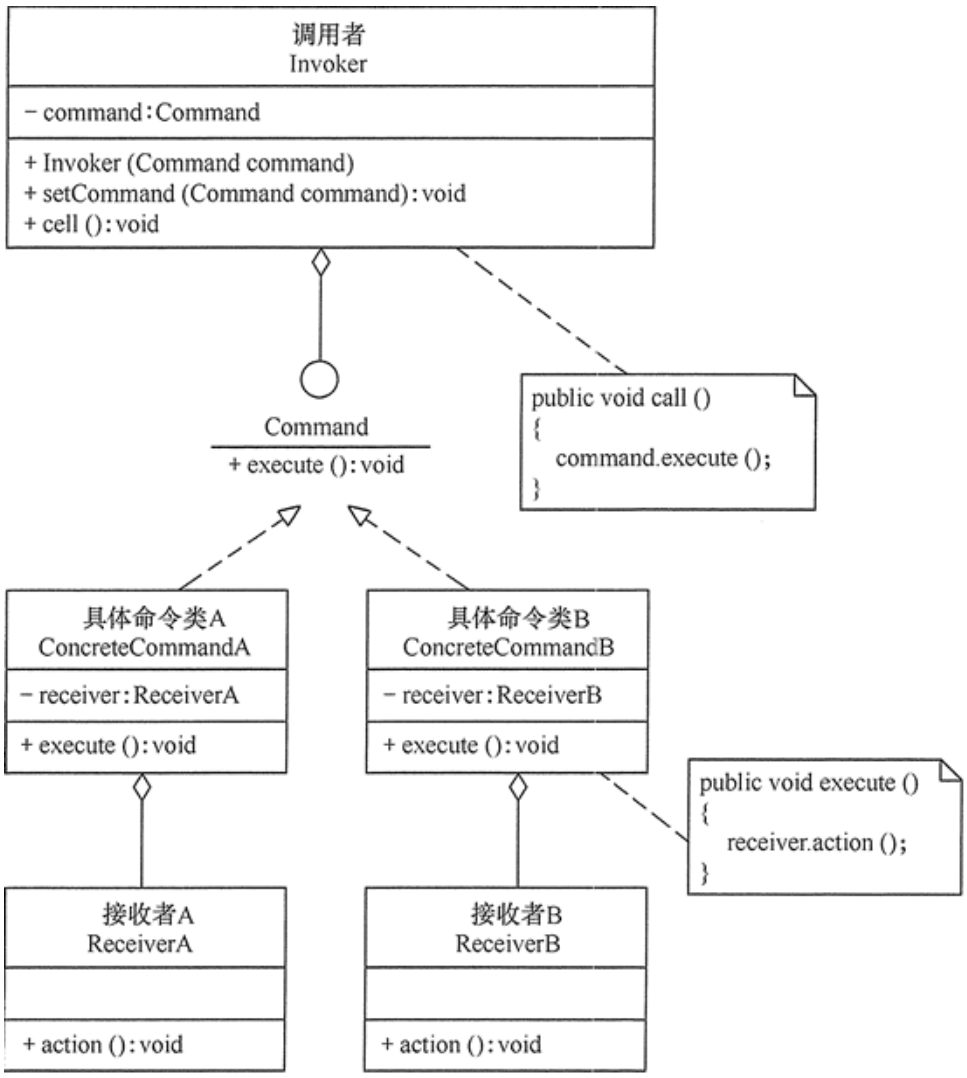

图2.1 命令模式UML结构图

1. 抽象命令类（Command）角色：声明执行命令的接口，拥有执行命令的抽象方execute()。
2. 具体命令角色（Concrete Command）角色：是抽象命令类的具体实现类，它拥有接收者对象，并通过调用接收者的功能来完成命令要执行的操作。
3. 实现者/接收者（Receiver）角色：执行命令功能的相关操作，是具体命令对象业务的真正实现者。
4. 调用者/请求者（Invoker）角色：是请求的发送者，它通常拥有很多的命令对象，并通过访问命令对象来执行相关请求，它不直接访问接收者。

3. 命令模式示例

对于在线课程学习网站中的教学视频，管理者可以选择开放视频给用户进行观看或者关闭视频观看渠道。利用命令模式把开发视频观看命令和关闭视频观看命令进行处理，实现命令请求者和具体实现者的解耦。

1. 抽象命令类

```
public interface Command {
    void execute();
}
```

2. 具体命令类

```
public class CloseCourseVideoCommand implements Command{
    private CourseVideo courseVideo;

    public CloseCourseVideoCommand(CourseVideo courseVideo) {
        this.courseVideo = courseVideo;
    }

    @Override
    public void execute() {
        courseVideo.close();
    }
}
```

```
public class OpenCourseVideoCommand implements Command{
    private CourseVideo courseVideo;

    public OpenCourseVideoCommand(CourseVideo courseVideo) {
        this.courseVideo = courseVideo;
    }

    @Override
    public void execute() {
        courseVideo.open();
    }
}
```

3. 实现者/接受者

```
public class CourseVideo {
    private String name;

    public CourseVideo(String name) {
        this.name = name;
    }

    public void open(){
        System.out.println(this.name + "课程视频开放");
    }

    public void close(){
        System.out.println(this.name + "课程视频关闭");
    }
}
```

4. 调用者/请求者

```
public class Staff {
    List<Command> commandList = new ArrayList<Command>();

    public void addCommand(Command command){
        commandList.add(command);
    }

    public void executeCommand(){
        for (Command command : commandList){
            command.execute();
        }
        commandList.clear();
    }
```

5. 客户端

```
 @Test
    public void commandTest(){
        CourseVideo courseVideo = new CourseVideo("java设计模式视频");
        Staff staff = new Staff();
        System.out.println("发出开放视频命令");
        Command openCommand = new OpenCourseVideoCommand(courseVideo);
        staff.addCommand(openCommand);
        staff.executeCommand();
```

```
        System.out.println("发出关闭视频命令");
        Command closeCommand = new CloseCourseVideoCommand(courseVideo);
        staff.addCommand(closeCommand);
        staff.executeCommand();
    }
```

6. 结果

```
发出开放视频命令
java设计模式视频课程视频开发
发出关闭视频命令
java设计模式视频课程视频关闭
```

4. 优/缺点

1. 优点

- 降低耦合，将调用操作的对象和实现该操作的对象解耦。
- 增加或删除命令非常方便。采用命令模式增加与删除命令不会影响其他类，它满足“开闭原则”，对扩展比较灵活。
- 可以实现宏命令。命令模式可以与**组合模式**结合，将多个命令装配成一个组合命令，即宏命令。
- 和备忘录模式一起可以实现undo和redo操作。

2. 缺点

- 命令的无限扩展会增加类的数量，提供系统的复杂度。

5. 适用场景

1. 请求调用者和请求接收者需要解耦，使得调用者和接收者不直接交互。
2. 当系统需要随机请求命令或经常增加或删除命令时，命令模式比较方便实现这些功能。
3. 当系统需要执行一组操作时，命令模式可以定义宏命令来实现该功能。
4. 当系统需要支持命令的撤销（Undo）操作和恢复（Redo）操作时，可以将命令对象存储起来，采用备忘录模式来实现。

6. 扩展

在软件开发中，有时将命令模式与前面学的组合模式联合使用，这就构成了宏命令模式，也叫组合命令模式。宏命令包含了一组命令，它充当了具体命令与调用者的双重角色，执行它时将递归调用它所包含的所有命令，其具体结构图如图所示。

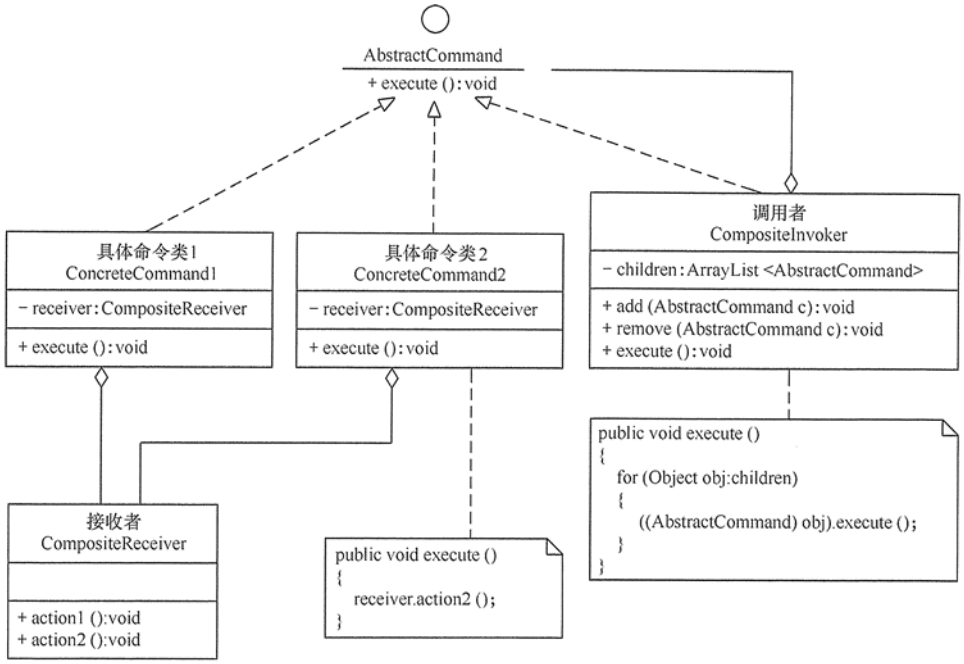

图6.1 组合命令模式UML结构图

7. 参考资源

[1]http://c.biancheng.net/view/1380.html