责任链模式

2020年2月14日　　10:48

1. 概念

为了避免请求发送者与多个请求处理者耦合在一起，将所有请求的处理者通过前一对象记住其下一个对象的引用而连成一条链；当有请求发生时，可将请求沿着这条链传递，直到有对象处理它为止。即为请求创建一个接受此次请求对象的链。
也叫做职责链模式。

2. UML结构图

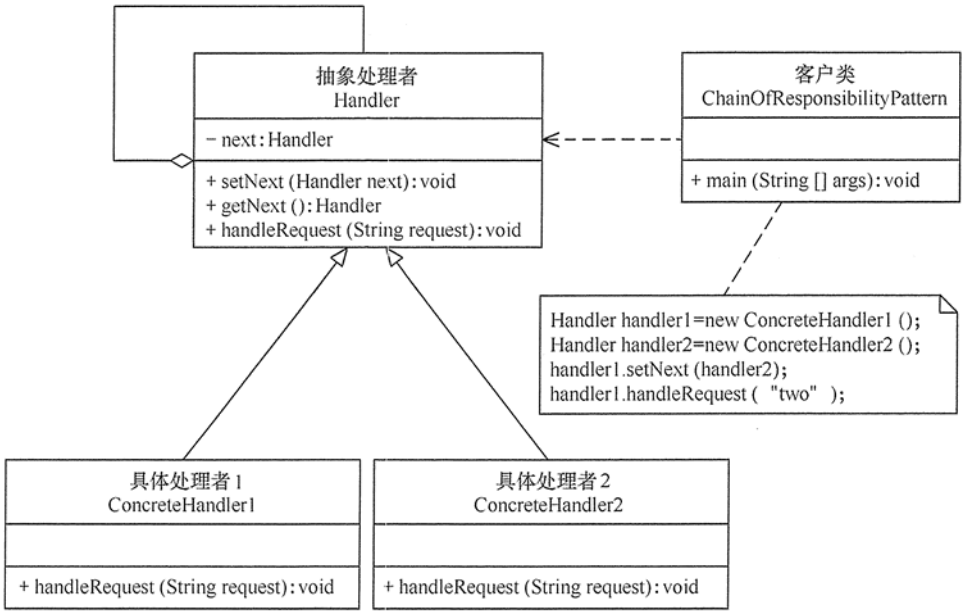

图2.1 责任链模式UML结构图

1. 抽象处理者（Handler）角色：定义一个处理请求的接口，包含抽象处理方法和一个后继连接。
2. 具体处理者（Concrete Handler）角色：实现抽象处理者的处理方法，判断能否处理本次请求，如果可以处理请求则处理，否则将该请求转给它的后继者。
3. 客户类（Client）角色：创建处理链，并向链头的具体处理者对象提交请求，它不关心处理细节和请求的传递过程。
4. 处理对象：处理链需要进行处理的对象。

3. 责任链模式实例

在一个在线学习网站中，若要上传一个在线课程，需要经过多层审批，包括课程信息审批，课程视频审批和课程手记手记，采用责任链模式对上述审批进行处理。

1. 抽象处理者

```
public abstract class Approver {
    public Approver approver;
    // 设置下一个批准者
    public void setNextApprover(Approver approver){
        this.approver = approver;
    }

    // 课程审批处理
    public abstract void deploy(Course course);


}
```

```
}
```

2. 具体处理者

```
public class CourseInfoApprover extends Approver{
    @Override
    public void deploy(Course course) {
        if (StringUtils.isNoneEmpty(course.getCourseInfo())){
            System.out.println(course.getName() + "信息完善，审批通过");

            // 拿到父类approver,下面还有人需要审批
            if (approver != null){
                approver.deploy(course);
            }
        }else{
            System.out.println(course.getName() + "信息不完善，审批不通过");
            return;
        }
    }
}
```

```
public class VideoApprover extends Approver {
    @Override
    public void deploy(Course course) {
        if (StringUtils.isNoneEmpty(course.getVideo())){
            System.out.println(course.getName() + "视频不为空，审批通过");

            // 拿到父类approver,下面还有人需要审批
            if (approver != null){
                approver.deploy(course);
            }
        }else{
            System.out.println(course.getName() + "视频为空，审批不通过");
            return;
        }
    }
}
```

```
public class ArticleApprover extends Approver {
    @Override
    public void deploy(Course course) {
        if (StringUtils.isNoneEmpty(course.getArticle())){
            System.out.println(course.getName() + "手记不为空，审批通过");

            // 拿到父类approver,下面还有人需要审批
            if (approver != null){
                approver.deploy(course);
            }
        }else{
            System.out.println(course.getName() + "手记为空，审批不通过");
            return;
        }
```

3. 处理对象

```
public class Course {
    private String name;
    private String article;
    private String video;
    private String courseInfo;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
      ……

    public void setVideo(String video) {
        this.video = video;
    }

    public String getCourseInfo() {
        return courseInfo;
    }

    public void setCourseInfo(String courseInfo) {
        this.courseInfo = courseInfo;
    }
}
```

4. 客户端

```
 @Test
    public void chainOfResponsibilityTest(){
        Approver articleApprover = new ArticleApprover();
```

```
        Approver videoApprover = new VideoApprover();
        Approver courseInfoApprover = new CourseInfoApprover();

        Course course = new Course();
        course.setName("java课程");
        course.setArticle("java课程手记");
        course.setCourseInfo("关于java的学习课程");
        course.setVideo("java课程视频");

        // 设置责任链顺序 articleApprover -> videoApprover
        /*articleApprover.setNextApprover(videoApprover);
        articleApprover.deploy(course);*/

        // 设置责任链顺序 videoApprover -> articleApprover
       /* videoApprover.setNextApprover(articleApprover);
        videoApprover.deploy(course);*/

        // 设置责任链 courseInfoApprover -> videoApprover -> articleApprover
        courseInfoApprover.setNextApprover(videoApprover);
        videoApprover.setNextApprover(articleApprover);
        courseInfoApprover.deploy(course);
    }
```

5. 结果

```
java课程信息完善，审批通过
java课程视频不为空，审批通过
java课程手记不为空，审批通过
```

4. 优/缺点

1. 优点

- 请求的发送者和接收者（请求的处理）解耦。用户只需要将请求发送到责任链上即可，无须关心请求的处理细节和请求的传递过程，所以责任链将请求的发送者和请求的处理者解耦了。
- 责任链可以动态组合和扩展。
- 责任链简化了对象之间的连接。每个对象只需保持一个指向其后继者的引用，不需保持其他所有处理者的引用，这避免了使用众多的 if 或者 if…else 语句。
- 责任分担。每个类只需要处理自己该处理的工作，不该处理的传递给下一个对象完成，明确各类的责任范围，符合类的单一职责原则。

2. 缺点

- 责任链太长或者处理时间过长，影响性能。
- 职责链建立的合理性要靠客户端来保证，增加了客户端的复杂性，可能会由于职责链的错误设置而导致系统出错，如可能会造成循环调用。

5. 适用场景

一个请求的处理需要多个对象中的一个或几个协作处理。例如OA中的审批流程，就涉及了HR，主管，董事长多个对象的审批。

6. 参考资料

[1]http://c.biancheng.net/view/1383.html