中介者模式

2020年2月22日 10:38

1. 概念

定义一个中介对象来封装一系列对象之间的交互，使原有对象之间的耦合松散，且可以独立地改变它们之间的交互。中介者模式又叫调停模式，它是迪米特法则的典型应用。
很多对象之间都存在者复杂的"网状结构"的交互关系，要求每个对象都需要知道它需要进行交互的对象，而若改为"星型结构"，借助中介者就可以很方便进行交互，大大降低了对象之间的耦合。例如MVC框架之中的控制器(C)就是模型(M)和视图(V)的中介者。

2. UML结构图

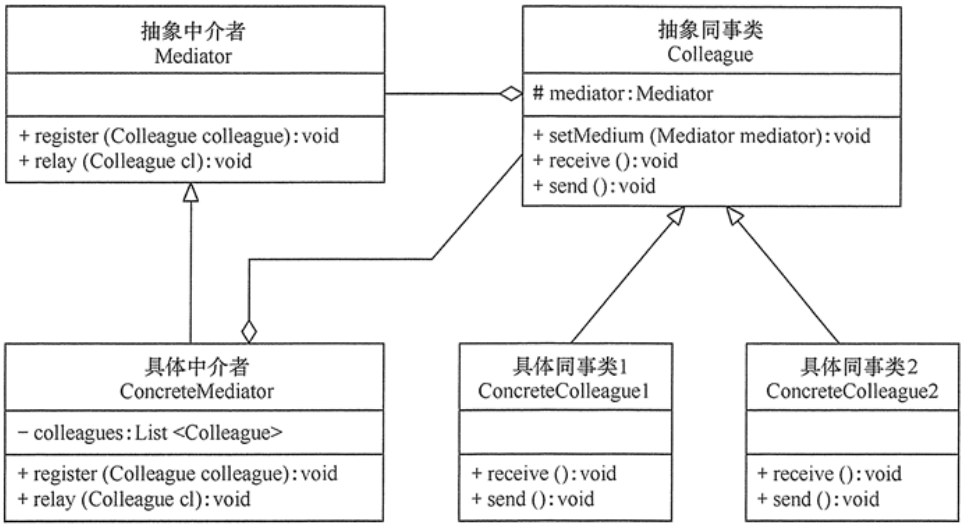

图2.1 中介者模式UML结构图

1. 抽象中介者（Mediator）角色：它是中介者的接口，提供了同事对象注册与转发同事对象信息的抽象方法。
2. 具体中介者（ConcreteMediator）角色：实现中介者接口，定义一个 List 来管理同事对象，协调各个同事角色之间的交互关系，因此它依赖于同事角色。
3. 抽象同事类（Colleague）角色：定义同事类的接口，保存中介者对象，提供同事对象交互的抽象方法，实现所有相互影响的同事类的公共功能。
4. 具体同事类（Concrete Colleague）角色：是抽象同事类的实现者，当需要与其他同事对象交互时，由中介者对象负责后续的交互。

3. 中介者模式实现

1. 抽象中介类

```
public abstract class Mediator {
    public abstract void register(Colleague colleague);
    public abstract void relay(Colleague cl);
}
```

2. 具体中介类

```
public class ConcreteMediator extends Mediator {
    List<Colleague> colleagueList = new ArrayList<Colleague>();

    @Override
    public void register(Colleague colleague) {
        if (!colleagueList.contains(colleague)){
            colleagueList.add(colleague);
            // 为相应对象设置中介者
            colleague.setMediator(this);
```

```
        }
    }

    @Override
    public void relay(Colleague cl) {
        for (Colleague ob : colleagueList){
            if (!ob.equals(cl)){
                ((Colleague)ob).receive();
            }
        }
    }
}
```

3. 抽象同事类

```
public abstract class Colleague {
    protected Mediator mediator;

    // 设置对象的中介者
    public void setMediator(Mediator mediator) {
        this.mediator = mediator;
    }

    public abstract void receive();
    public abstract void send();
}
```

4. 具体同事类

```
public class ConcreteColleague1 extends Colleague {
    @Override
    public void receive() {
        System.out.println("同事1收到请求");
    }

    @Override
    public void send() {
        System.out.println("同事1发出请求");
        mediator.relay(this);
    }
}
```

```
public class ConcreteColleague2 extends Colleague{
    @Override
    public void receive() {
        System.out.println("同事2收到请求");
    }

    @Override
    public void send() {
        System.out.println("同事2发出请求");
        mediator.relay(this);
    }
}
```

```
public class ConcreteColleague3 extends Colleague {
    @Override
    public void receive() {
        System.out.println("同事3收到请求");
    }

    @Override
    public void send() {
        System.out.println("同事3发出请求");
        mediator.relay(this);
    }
}
```

5. 客户端

```
@Test
    public void mediatorTest2(){
        Mediator concreteMediator = new ConcreteMediator();
        Colleague concreteColleague1 = new ConcreteColleague1();
        Colleague concreteColleague2 = new ConcreteColleague2();
        Colleague concreteColleague3 = new ConcreteColleague3();


        concreteMediator.register(concreteColleague1);
        concreteMediator.register(concreteColleague2);
        concreteMediator.register(concreteColleague3);
        concreteColleague1.send();
    }
}
```

6. 结果

```
同事1发出请求
```

```
同事2收到请求
同事3收到请求
```

4. 优/缺点

1. 优点

- 将一对多转化成一对一，降低程序复杂度。
- 降低了对象之间的耦合性，使得对象易于独立地被复用。

2. 缺点

当同事类太多时，中介者的职责将很大，它会变得复杂而庞大，以至于系统难以维护。

5. 适用场景

1. 系统中对象之间存在复杂的引用关系，产生的相互依赖关系结构混乱且难以理解。
2. 交互的公共行为，如果需要改变行为则可以增加新的中介者类。

6. 扩展

在实际开发中，通常采用以下两种方法来简化中介者模式，使开发变得更简单。

1. 不定义中介者接口，把具体中介者对象实现成为单例。
2. 同事对象不持有中介者，而是在需要的时直接获取中介者对象并调用。

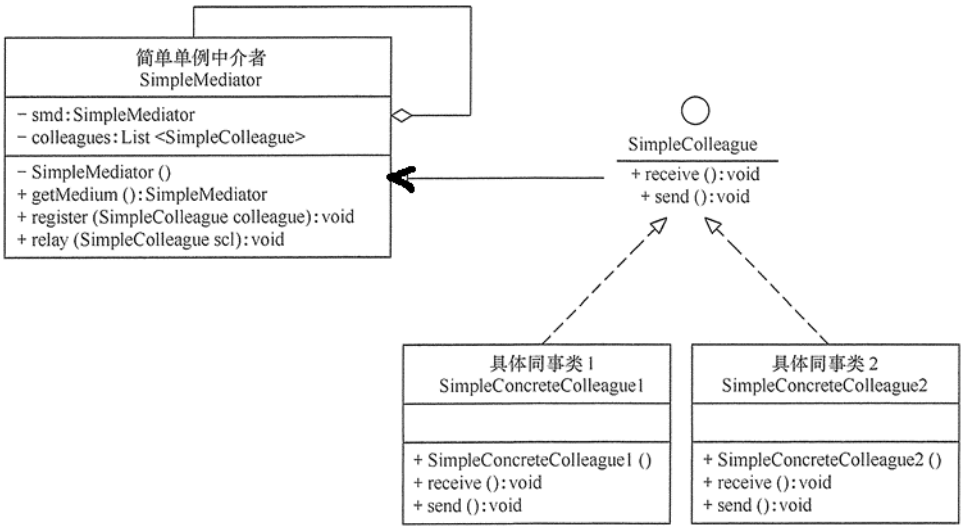

图6.1 中介者模式扩展

7. 扩展

[1] http://c.biancheng.net/view/1393.html