



第9章 设计模式

9.4 观察者模式

刘其成

计算机与控制工程学院

ytliuqc@163.com

2018-09

本章教学内容

- 观察者模式
 - 模式动机与定义
 - 模式结构与分析
 - 模式实例与解析
 - 模式效果与应用
 - 模式扩展

观察者模式

■ 模式动机

- 建立一种对象与对象之间的依赖关系，一个对象发生改变时将自动通知其他对象，其他对象将相应做出反应。
- 在此，发生改变的对象称为观察目标，而被通知的对象称为观察者，一个观察目标可以对应多个观察者，而且这些观察者之间没有相互联系，可以根据需要增加和删除观察者，使得系统更易于扩展，这就是观察者模式的模式动机。

观察者模式

■ 模式定义

- 观察者模式(Observer Pattern): 定义对象间的一种一对多依赖关系, 使得每当一个对象状态发生改变时, 其相关依赖对象皆得到通知并被自动更新。
- 观察者模式又叫做发布-订阅(Publish/Subscribe)模式、模型-视图(Model/View)模式、源-监听器(Source/Listener)模式或从属者(Dependents)模式。
- 观察者模式是一种对象行为型模式。

观察者模式

■ 模式定义

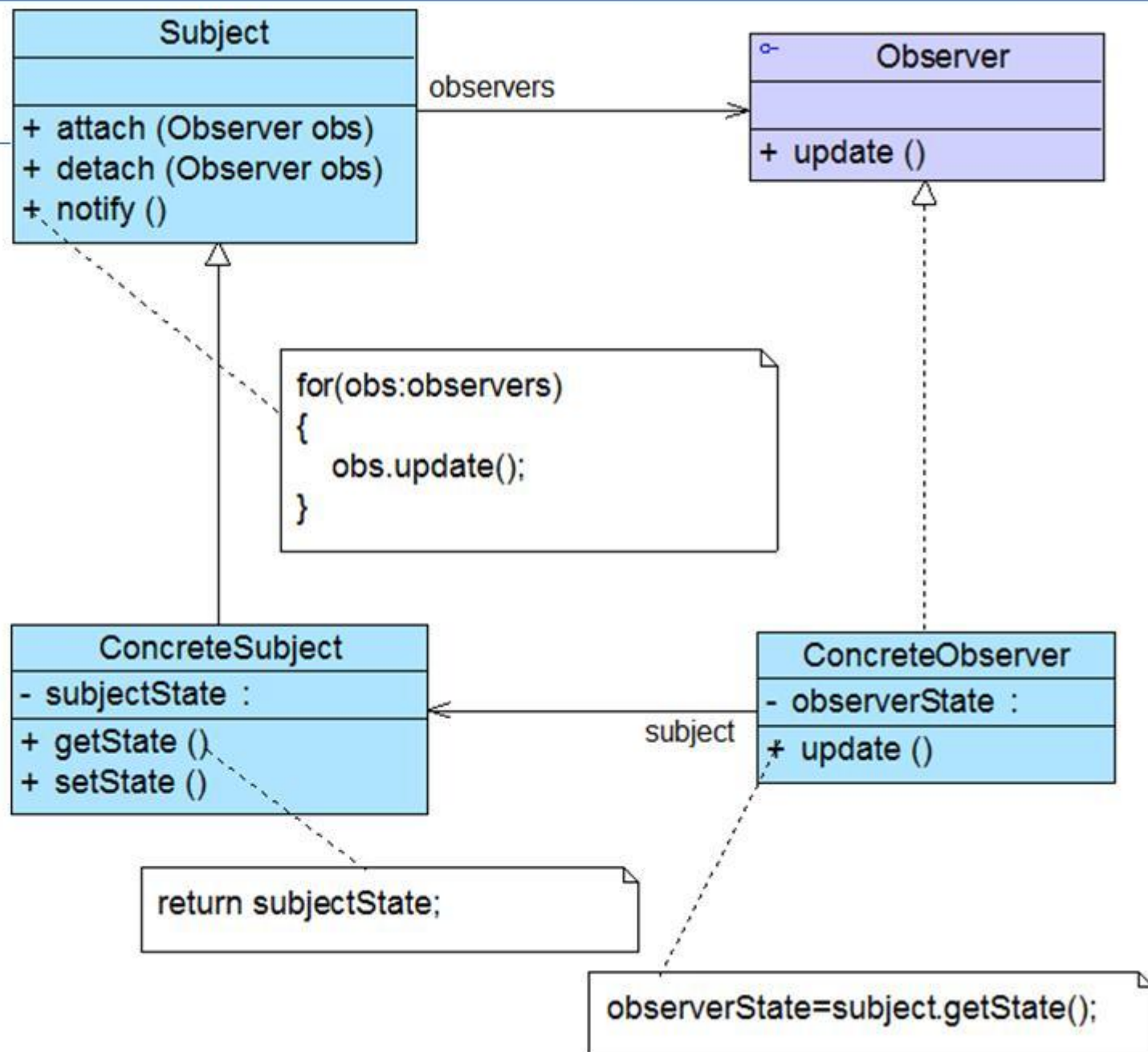
- **Observer Pattern:** Define a **one-to-many dependency** between objects so that when **one object changes state**, **all its dependents are notified and updated automatically**.

- **Frequency of use:** **high**



观察者模式

■ 模式结构



观察者模式

■ 模式结构

— 观察者模式包含如下角色：

- **Subject**: 目标
- **ConcreteSubject**: 具体目标
- **Observer**: 观察者
- **ConcreteObserver**: 具体观察者

观察者模式

■ 模式分析

- 观察者模式描述了如何建立对象与对象之间的依赖关系，如何构造满足这种需求的系统。
- 这一模式中的关键对象是观察目标和观察者，一个目标可以有任意数目的与之相依赖的观察者，一旦目标的状态发生改变，所有的观察者都将得到通知。
- 作为对这个通知的响应，每个观察者都将即时更新自己的状态，以与目标状态同步，这种交互也称为发布-订阅(publish-subscribe)。目标是通知的发布者，它发出通知时并不需要知道谁是它的观察者，可以有任意数目的观察者订阅它并接收通知。

观察者模式

- 模式分析

- 典型的抽象目标类代码如下所示：

```
import java.util.*;  
public abstract class Subject{  
    protected ArrayList observers = new ArrayList();  
    public abstract void attach(Observer observer);  
    public abstract void detach(Observer observer);  
    public abstract void notify();  
}
```

观察者模式

■ 模式分析

— 典型的具体目标类代码如下所示：

```
public class ConcreteSubject extends Subject{  
    public void attach(Observer observer){  
        observers.add(observer);  
    }  
    public void detach(Observer observer){  
        observers.remove(observer);  
    }  
    public void notify(){  
        for(Object obs:observers){  
            ((Observer)obs).update();  
        }  
    }  
}
```

观察者模式

- 模式分析

- 典型的抽象观察者代码如下所示：

```
public interface Observer{  
    public void update();  
}
```

观察者模式

- 模式分析

- 典型的具体观察者代码如下所示：

```
public class ConcreteObserver implements Observer{  
    public void update(){  
        //具体更新代码  
    }  
}
```


观察者模式

- 模式分析

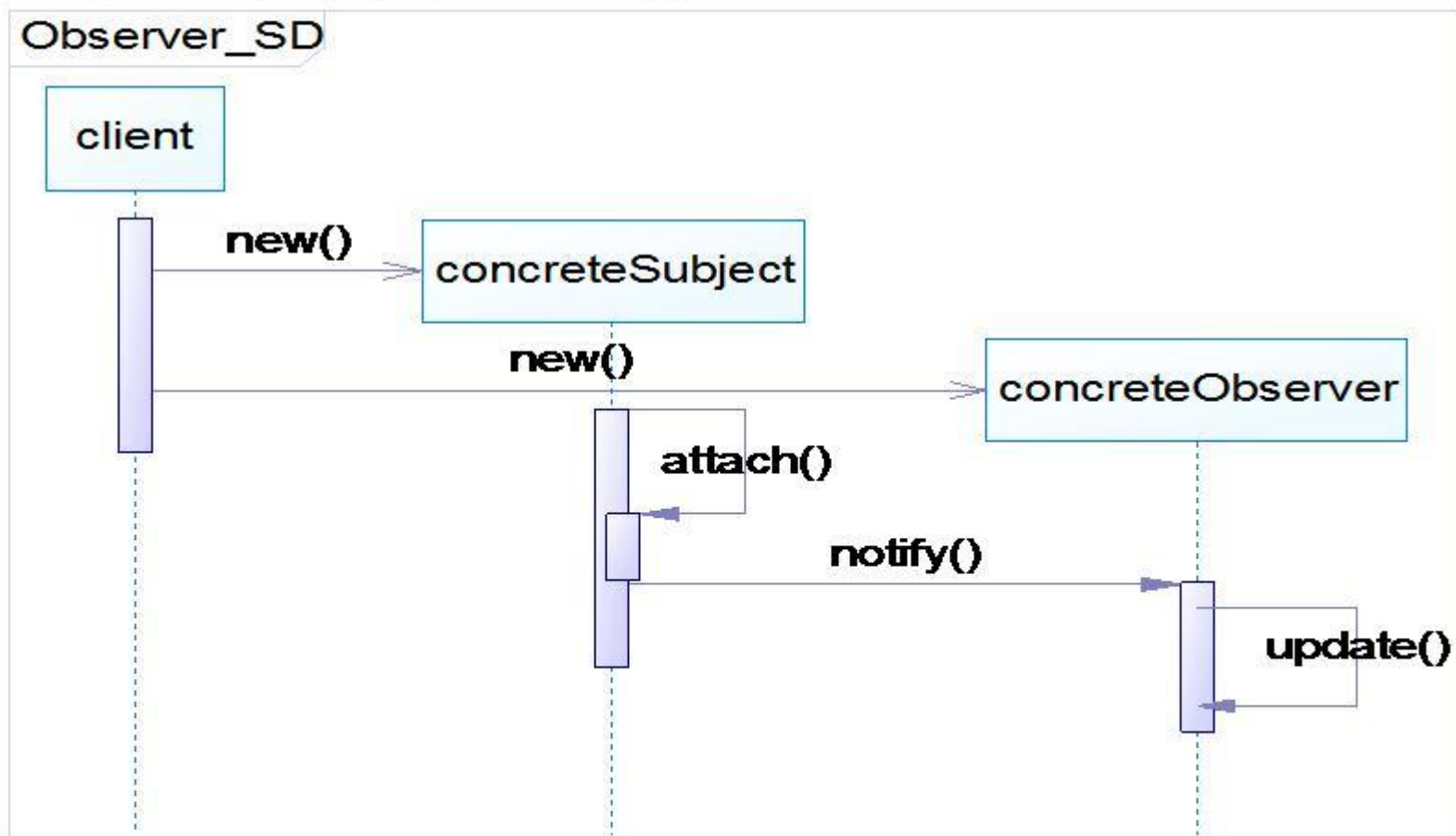
- 客户端代码片段如下所示:

```
Subject subject = new ConcreteSubject();  
Observer observer = new ConcreteObserver();  
subject.attach(observer);  
subject.notify();
```

观察者模式

■ 模式分析

— 观察者模式顺序图如下所示：



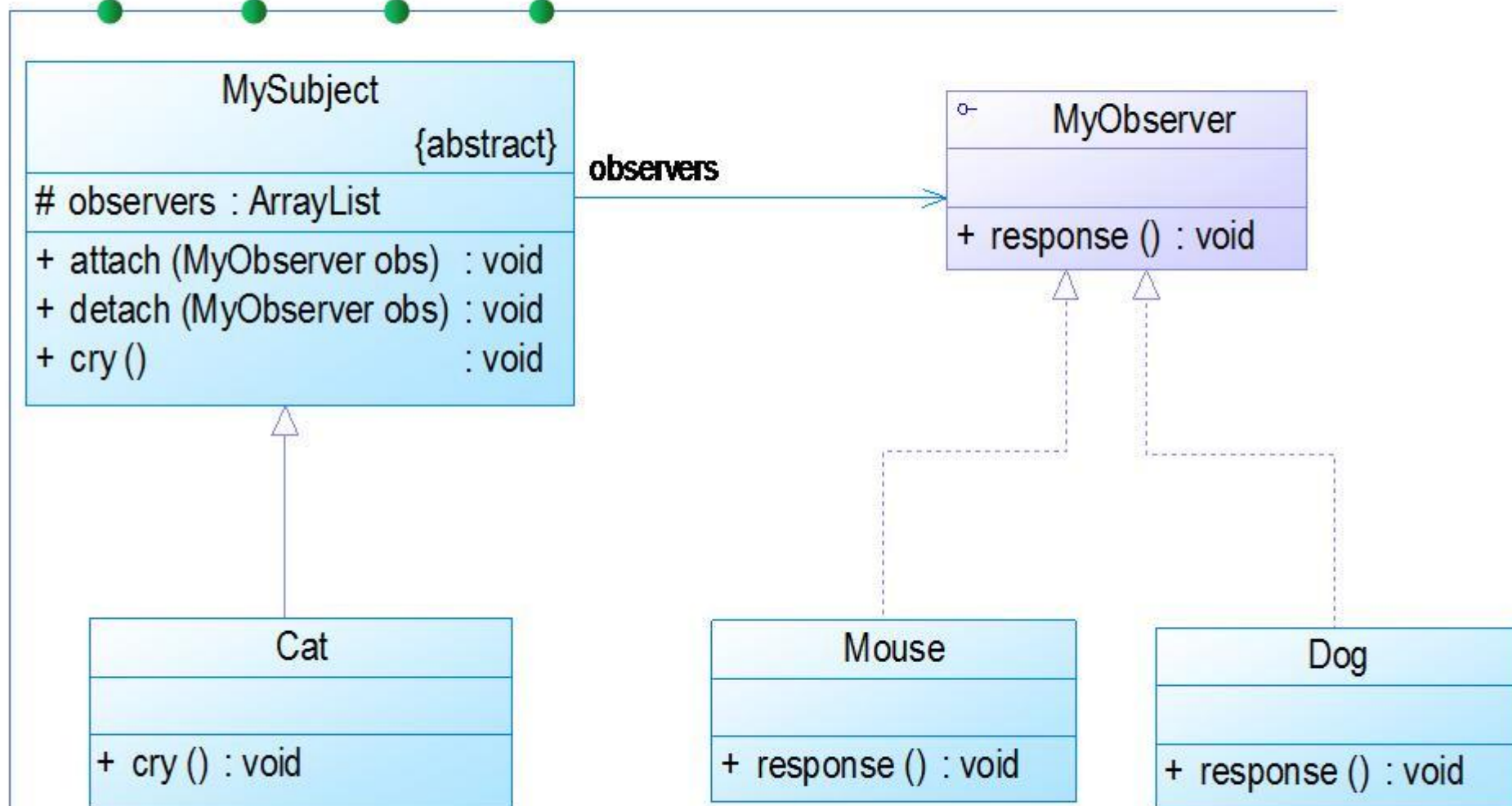
观察者模式

■ 观察者模式实例与解析

— 实例：猫、狗与老鼠

- 假设猫是老鼠和狗的观察目标，老鼠和狗是观察者，猫叫老鼠跑，狗也跟着叫，使用观察者模式描述该过程。


观察者模式




观察者模式

```
import java.util.*;
public abstract class MySubject{
    protected ArrayList observers = new ArrayList();
    public void attach(MyObserver observer){//注册方法


        observers.add(observer);
    }
    public void detach(MyObserver observer){//注销方法
        observers.remove(observer);
    }
    public abstract void cry(); //抽象通知方法
}
```



```
public class Cat extends MySubject{  
    public void cry()  {  
        System.out.println("猫叫! ");  
        System.out.println("-----");  
        for(Object obs:observers)      {  
            ((MyObserver)obs).response();  
        }  
    }  
}
```




```
public interface MyObserver{  
    void response(); //抽象响应方法  
}  
  
public class Pig implements MyObserver{  
    public void response() {  
        System.out.println("猪没有反应！");  
    }  
}
```



```
public class Mouse implements MyObserver{  
    public void response() {  
        System.out.println("老鼠努力逃跑！");  
    }  
}
```

```
public class Dog implements MyObserver{  
    public void response() {  
        System.out.println("狗跟着叫！");  
    }  
}
```

```
public class Client{  
    public static void main(String a[]) {  
        MySubject subject=new Cat();  
  
        MyObserver obs1,obs2,obs3;  
        obs1=new Mouse();  
        obs2=new Mouse();  
        obs3=new Dog();  
  
        subject.attach(obs1);  
        subject.attach(obs2);  
        subject.attach(obs3);  
  
        subject.cry();  
    }  
}
```

观察者模式

■ 模式优缺点

— 观察者模式的优点

- 观察者模式可以**实现表示层和数据逻辑层的分离**，并定义了稳定的消息更新传递机制，抽象了更新接口，使得可以有各种各样不同的表示层作为具体观察者角色。
- 观察者模式在观察目标和观察者之间**建立一个抽象的耦合**。
- 观察者模式**支持广播通信**。
- 观察者模式**符合“开闭原则”**的要求。

观察者模式

■ 模式优缺点

— 观察者模式的缺点

- 如果一个观察目标对象有很多直接和间接的观察者的话，**将所有的观察者都通知到会花费很多时间**。
- 如果在观察者和观察目标之间有**循环依赖的话**，观察目标会触发它们之间进行循环调用，**可能导致系统崩溃**。
- 观察者模式**没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的**，而仅仅只是知道观察目标发生了变化。

观察者模式

■ 模式适用环境

— 在以下情况下可以使用观察者模式：

- 一个抽象模型有两个方面，其中一个方面依赖于另一个方面。将这些方面封装在独立的对象中使它们可以各自独立地改变和复用。
- 一个对象的改变将导致其他一个或多个对象也发生改变，而不知道具体有多少对象将发生改变，可以降低对象之间的耦合度。
- 一个对象必须通知其他对象，而并不知道这些对象是谁。
- 需要在系统中创建一个触发链，A对象的行为将影响B对象，B对象的行为将影响C对象……，可以使用观察者模式创建一种链式触发机制。

观察者模式

■ 模式扩展

— MVC模式

- MVC模式是一种架构模式，它包含三个角色：**模型(Model)**，**视图(View)**和**控制器(Controller)**。
- 观察者模式可以用来实现MVC模式，观察者模式中的**观察目标**就是MVC模式中的**模型(Model)**，而**观察者**就是MVC中的**视图(View)**，**控制器(Controller)**充当两者之间的中介者(Mediator)。
- **当模型层的数据发生改变时，视图层将自动改变其显示内容。**

本章小结

- 观察者模式定义对象间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象皆得到通知并被自动更新。观察者模式又叫做发布-订阅模式、模型-视图模式、源-监听器模式或从属者模式。观察者模式是一种对象行为型模式。
- 观察者模式包含四个角色：目标又称为主题，它是指被观察的对象；具体目标是目标类的子类，通常它包含有经常发生改变的数据，当它的状态发生改变时，向它的各个观察者发出通知；观察者将对观察目标的改变做出反应；在具体观察者中维护一个指向具体目标对象的引用，它存储具体观察者的有关状态，这些状态需要和具体目标的状态保持一致。
- 观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个目标对象，当这个目标对象的状态发生变化时，会通知所有观察者对象，使它们能够自动更新。

本章小结

- 观察者模式的主要优点在于可以实现表示层和数据逻辑层的分离，并在观察目标和观察者之间建立一个抽象的耦合，支持广播通信；其主要缺点在于如果一个观察目标对象有很多直接和间接的观察者的话，将所有的观察者都通知到会花费很多时间，而且如果在观察者和观察目标之间有循环依赖的话，观察目标会触发它们之间进行循环调用，可能导致系统崩溃。
- 观察者模式适用情况包括：一个抽象模型有两个方面，其中一个方面依赖于另一个方面；一个对象的改变将导致其他一个或多个对象也发生改变，而不知道具体有多少对象将发生改变；一个对象必须通知其他对象，而并不知道这些对象是谁；需要在系统中创建一个触发链。



谢谢

2019年5月29日