



深圳大学
Shenzhen University

第15-2讲

观察者模式

软件体系结构与设计模式
Software Architecture & Design Pattern

深圳大学计算机与软件学院

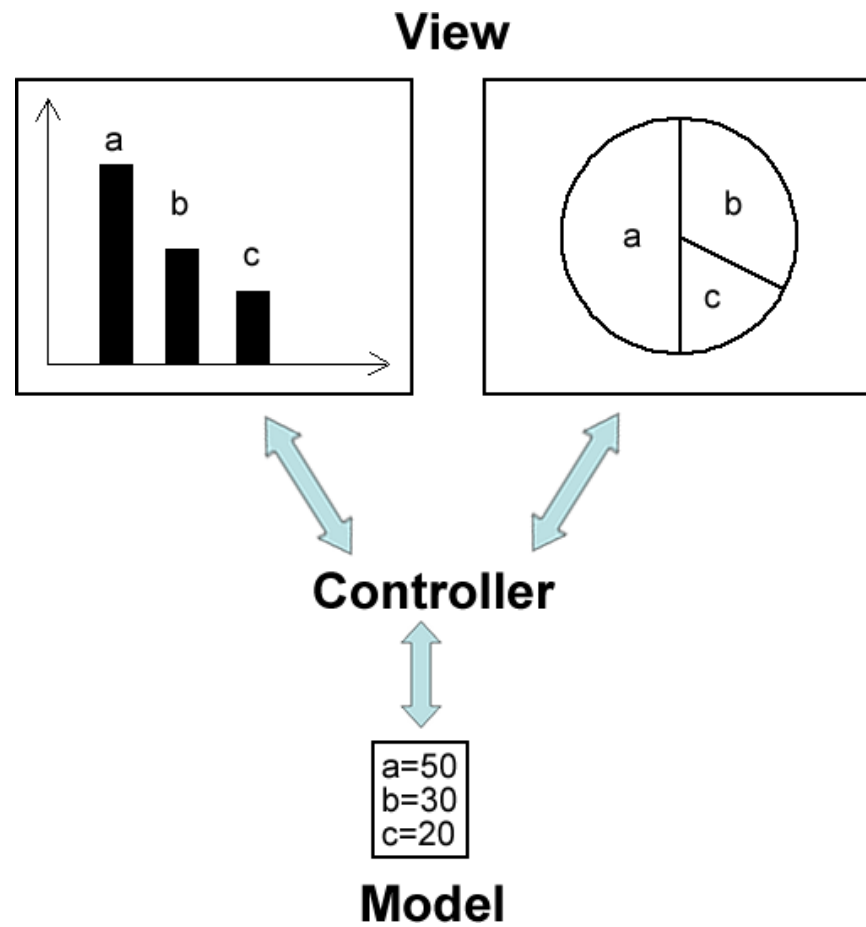


主要内容

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

观察者模式动机

■ MVC示意图





观察者模式动机

- 软件系统：一个对象的状态或行为的变化将导致其他对象的状态或行为也发生改变，它们之间将产生联动
- 观察者模式：
 - 定义了对象之间一种一对多的依赖关系，让一个对象的改变能够影响其他对象
 - 发生改变的对象称为观察目标，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

观察者模式定义

■ 对象行为型模式

观察者模式：定义对象之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，其相关依赖对象都得到通知并被自动更新。

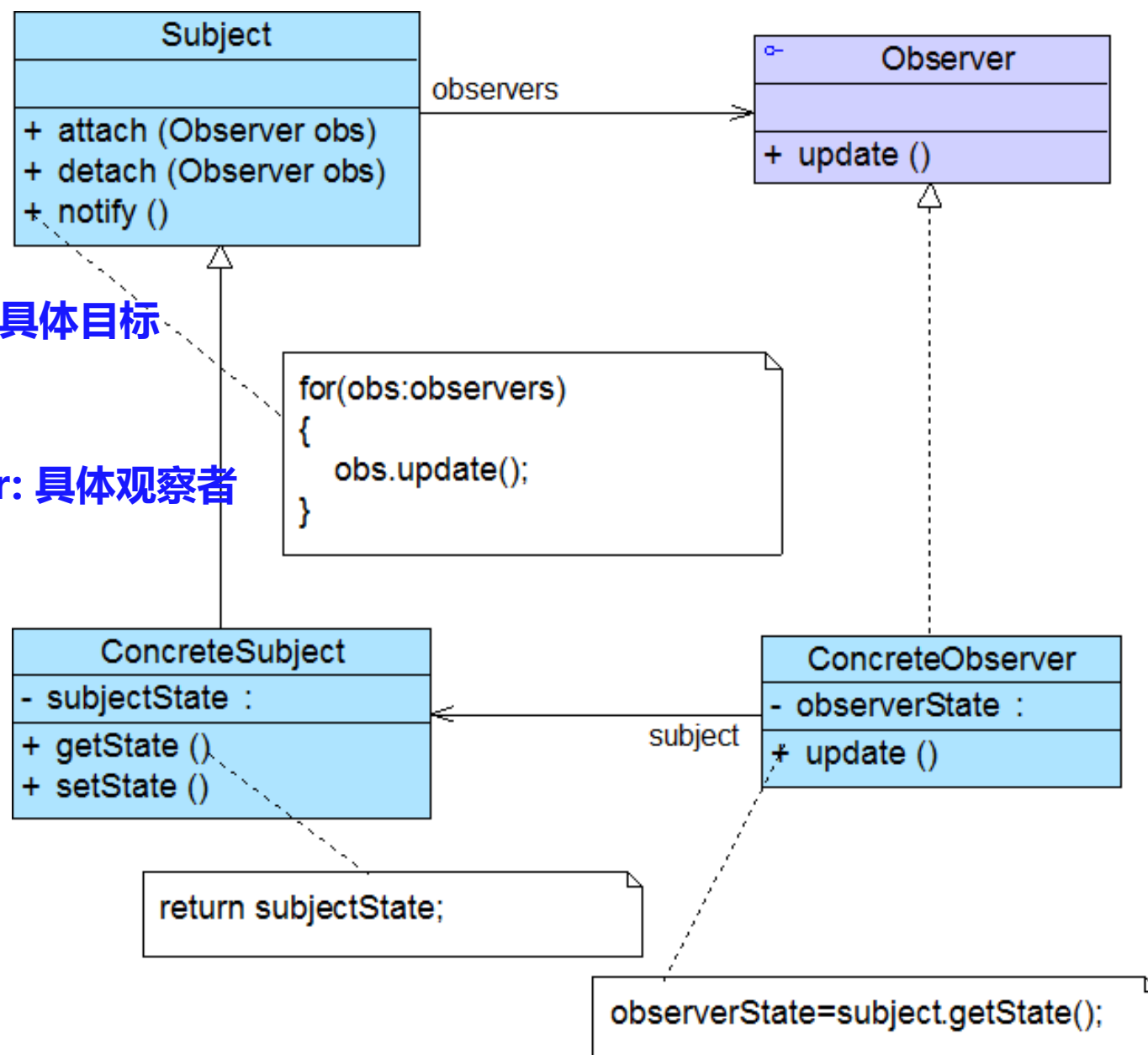
Observer Pattern: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

■ 别名

- 发布-订阅(Publish/Subscribe)模式
- 模型-视图(Model/View)模式
- 源-监听器(Source/Listener)模式
- 从属者(Dependents)模式

观察者模式结构

- 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





观察者模式分析

■ 说明：

- 有时候在具体观察者类ConcreteObserver中需要使用到具体目标类ConcreteSubject中的状态（属性），会存在关联或依赖关系
- 如果在具体层之间具有关联关系，系统的扩展性将受到一定的影响，增加新的具体目标类有时候需要修改原有观察者的代码，在一定程度上违背了开闭原则，但是如果原有观察者类无须关联新增的具体目标，则系统扩展性不受影响



观察者模式分析

- 抽象目标类示例代码：

```
import java.util.*;
public abstract class Subject {
    //定义一个观察者集合用于存储所有观察者对象
    protected ArrayList<Observer> = new ArrayList();

    //注册方法，用于向观察者集合中增加一个观察者
    public void attach(Observer observer) {
        observers.add(observer);
    }

    //注销方法，用于在观察者集合中删除一个观察者
    public void detach(Observer observer) {
        observers.remove(observer);
    }

    //声明抽象通知方法
    public abstract void notify();
}
```




观察者模式分析

- Java具体目标类示例代码：

```
public class ConcreteSubject extends Subject {  
    //实现通知方法  
    public void notify() {  
        //遍历观察者集合，调用每一个观察者的响应方法  
        for(Object obs:observers) {  
            ((Observer)obs).update();  
        }  
    }  
}
```



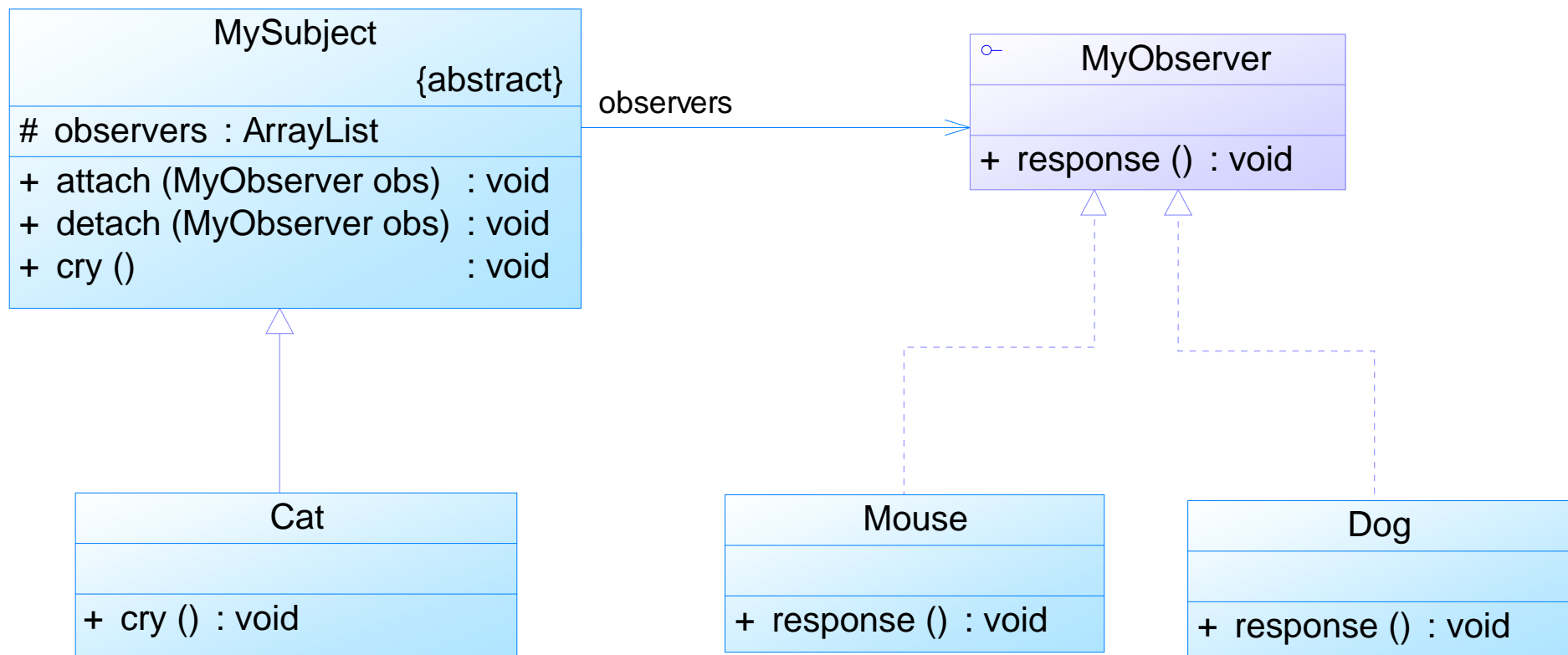
观察者模式实例

■ 猫、狗与老鼠：实例说明

- 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。

观察者模式实例与解析

■ 猫、狗与老鼠：参考类图



观察者模式实例与解析

■ 观察者模式实例

□ 猫、狗与老鼠：参考代码

■ DesignPatterns之observer包

MyObserver.java

```
1 package observer;
2
3 public interface MyObserver
4 {
5     void response(); //抽象响应方法
6 }
```

MySubject.java

```
1 package observer;
2
3 import java.util.*;
4
5 public abstract class MySubject
6 {
7     protected ArrayList observers = new ArrayList();
8
9     //注册方法
10    public void attach(MyObserver observer)
11    {
12        observers.add(observer);
13    }
14
15    //注销方法
16    public void detach(MyObserver observer)
17    {
18        observers.remove(observer);
19    }
20
21    public abstract void cry(); //抽象通知方法
22 }
```

Pig.java

```
1 package observer;
2
3 public class Pig implements MyObserver
4 {
5     public void response()
6     {
7         System.out.println("猪没有反应!");
8     }
9 }
```

Dog.java

```
1 package observer;
2
3 public class Dog implements MyObserver
4 {
5     public void response()
6     {
7         System.out.println("狗跟着叫!");
8     }
9 }
```

Mouse.java

```
1 package observer;
2
3 public class Mouse implements MyObserver
4 {
5     public void response()
6     {
7         System.out.println("老鼠努力逃跑!");
8     }
9 }
```

Cat.java

```
1 package observer;
2
3 public class Cat extends MySubject
4 {
5     public void cry()
6     {
7         System.out.println("猫叫! ");
8         System.out.println("-----");
9
10        for(Object obs:observers)
11        {
12            ((MyObserver)obs).response();
13        }
14    }
15 }
16 }
```

```
1 package observer;
2
3 public class Client
4 {
5     public static void main(String a[])
6     {
7         MySubject subject=new Cat();
8
9         MyObserver obs1,obs2,obs3;
10        obs1=new Mouse();
11        obs2=new Mouse();
12        obs3=new Dog();
13
14        subject.attach(obs1);
15        subject.attach(obs2);
16        subject.attach(obs3);
17
18        /*
19        MyObserver obs4;
20        obs4=new Pig();
21        subject.attach(obs4);
22        */
23
24        subject.cry();
25    }
26 }
```




观察者模式效果与应用

■ 观察者模式优点：

- 可以实现表示层和数据逻辑层的分离
- 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，简化了一对多系统设计的难度
- 符合开闭原则，增加新的具体观察者无须修改原有系统代码，在具体观察者与观察目标之间不存在关联关系的情况下，增加新的观察目标也很方便



观察者模式效果与应用

■ 观察者模式缺点：

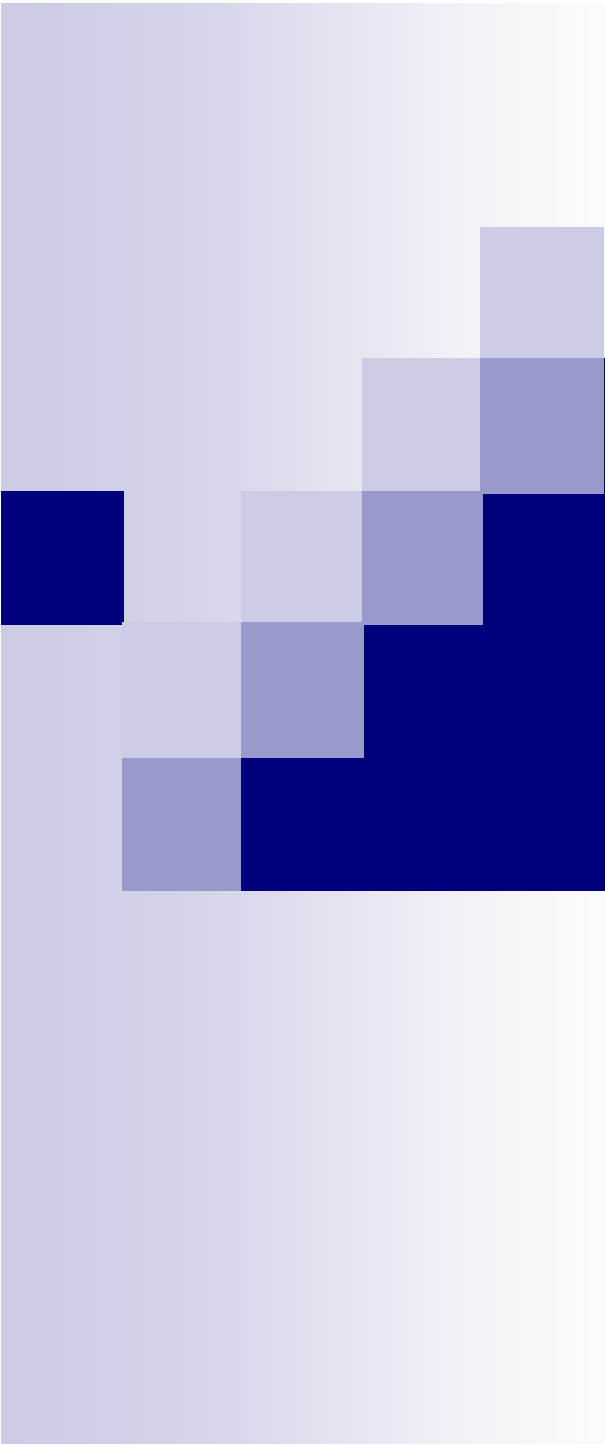
- 将所有观察者都通知到会花费很多时间
- 如果存在循环依赖时可能导致系统崩溃
- 没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的，而只是知道观察目标发生了变化



观察者模式效果与应用

■ 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，将这两个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一个或多个其他对象发生改变，且并不知道具体有多少对象将发生改变，也不知道这些对象是谁
- 需要在系统中创建一个触发链



谢谢