面向对象设计模式　观察者

曹东刚
caodg@pku.edu.cn

北京大学信息学院研究生课程 - 面向对象的分析与设计
http://sei.pku.edu.cn/~caodg/course/oo

一个应用：建造下一代 Internet 气象观察站

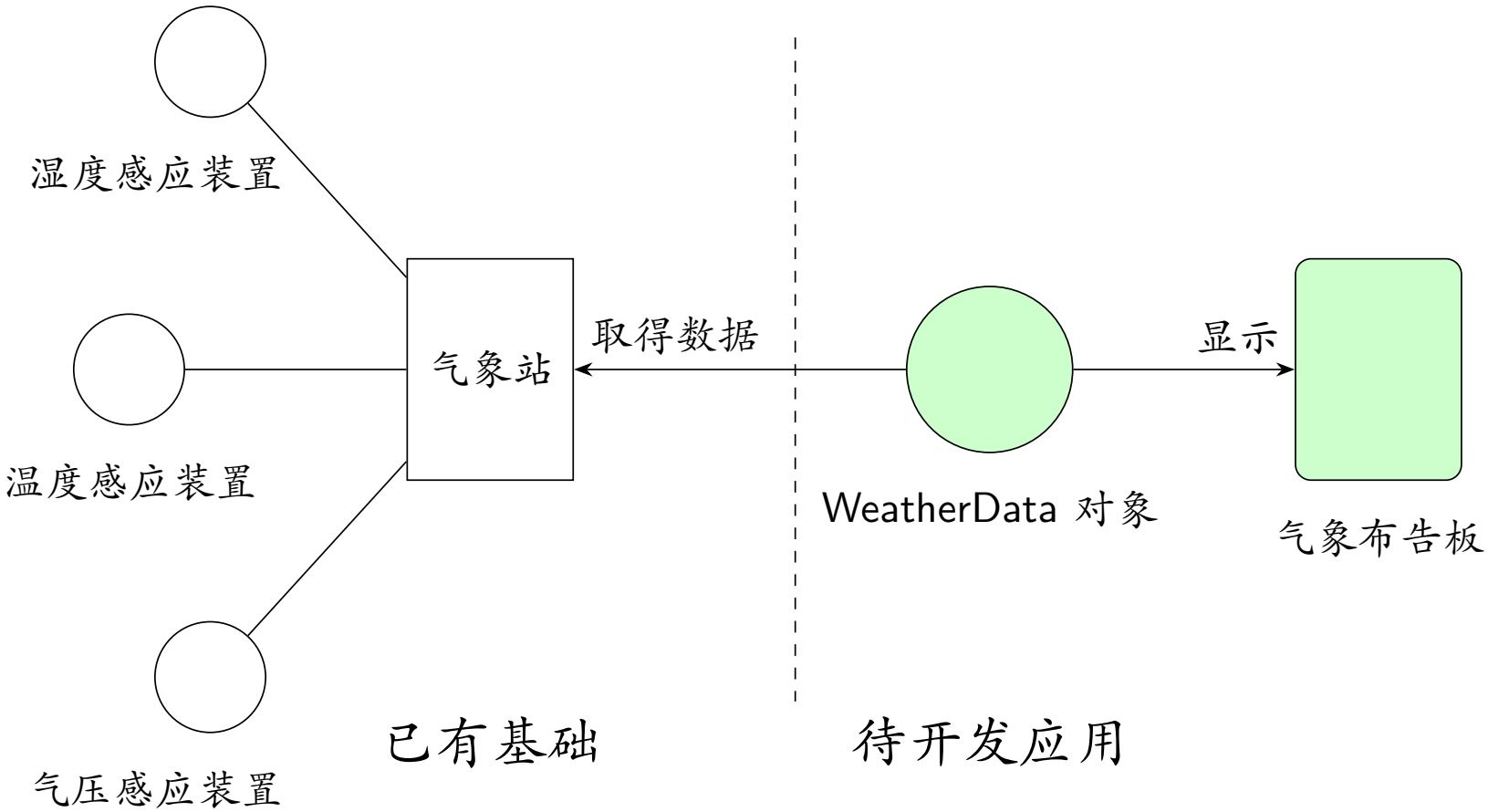
已有基础：

- 一个已开发的 WeatherData 对象，负责追踪目前的天气状况（温度、湿度、气压）

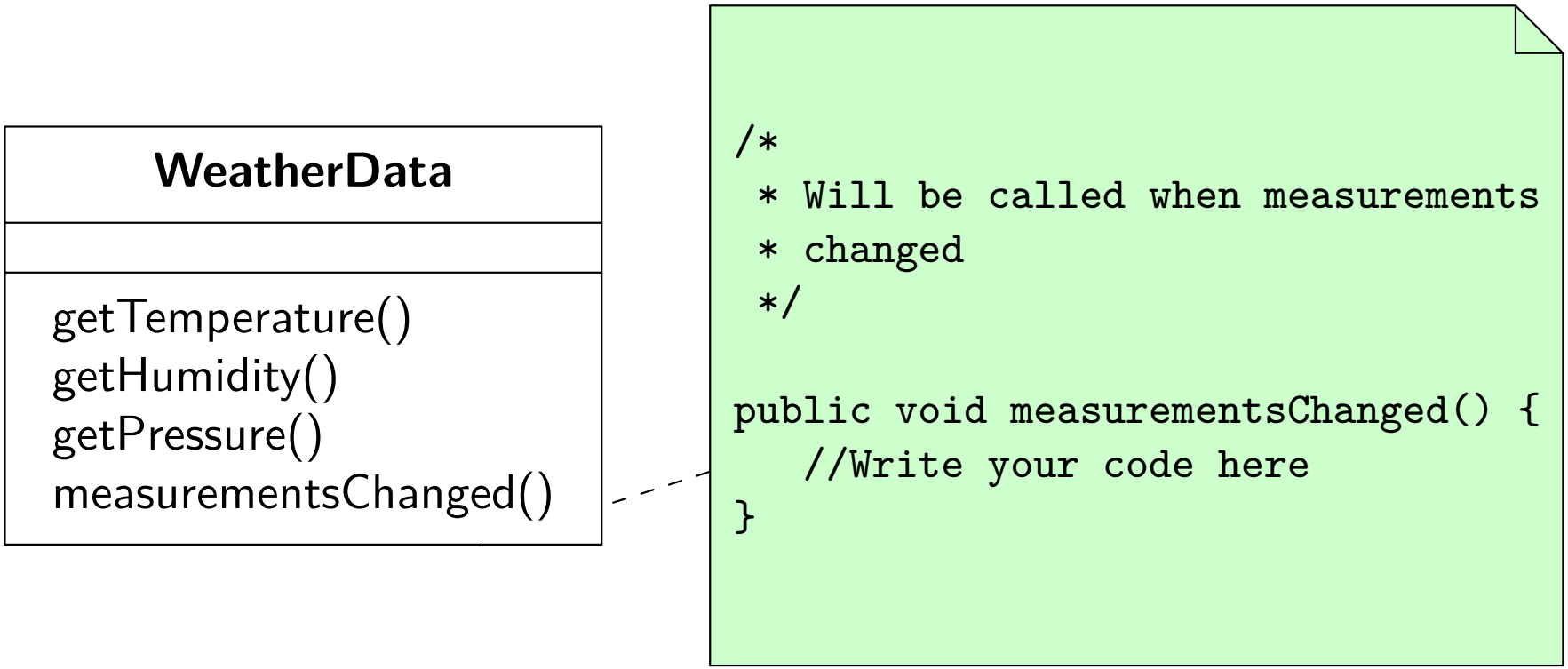
要求开发一个气象观察应用：

- 有三种布告板，分别显示目前的状况、气象统计及简单的预报，当 WeatherData 对象获得最新的气象数据时，三种布告板必须实时更新
- 气象站必须可扩展，能公开一组 API，允许其他开发人员开发出自己的气象布告板，并插入此应用中

气象观察应用的概况


湿度感应装置
温度感应装置
气压感应装置
气象站
取得数据
WeatherData 对象
显示
气象布告板
已有基础
待开发应用

WeatherData 类



我们知道的信息

- WeatherData 类的 getter 方法可以取得温度、湿度、气压的值
- 当有新数据的时候，measurementsChanged 方法就会被调用
  - 我们不关心它如何被调用，只知道它一定会被调用
- 我们需要实现三个布告板：目前状况布告、气象统计布告、天气预报布告，这些布告必须能够实时更新
- 系统必须允许在运行时添加新的布告板

常见做法及问题分析

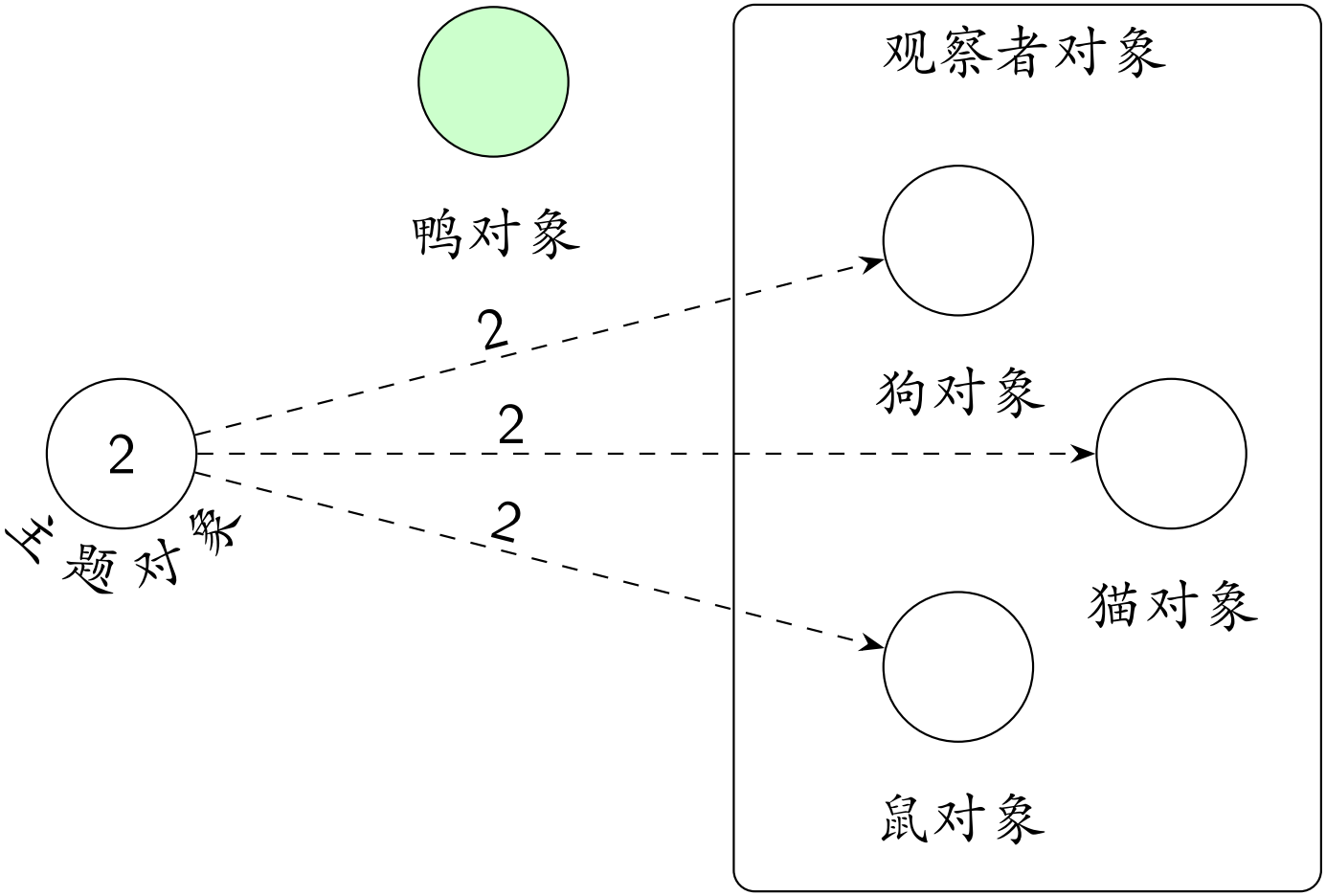
一种常见的做法

```
1  public void WeatherData {
2    public void measurementsChanged() {
3      float temp = getTemerature() ;
4      float humidity = getHumidity() ;
5      float pressure = getPressure() ;
6
7      currentConditionsDisplay.update(
8          temp, humidity, pressure) ;
9      statisticsDisplay.update(temp, humidity, pressure) ;
10     forecastDisplay.update(temp, humidity, pressure) ;
11   }
12 }
```

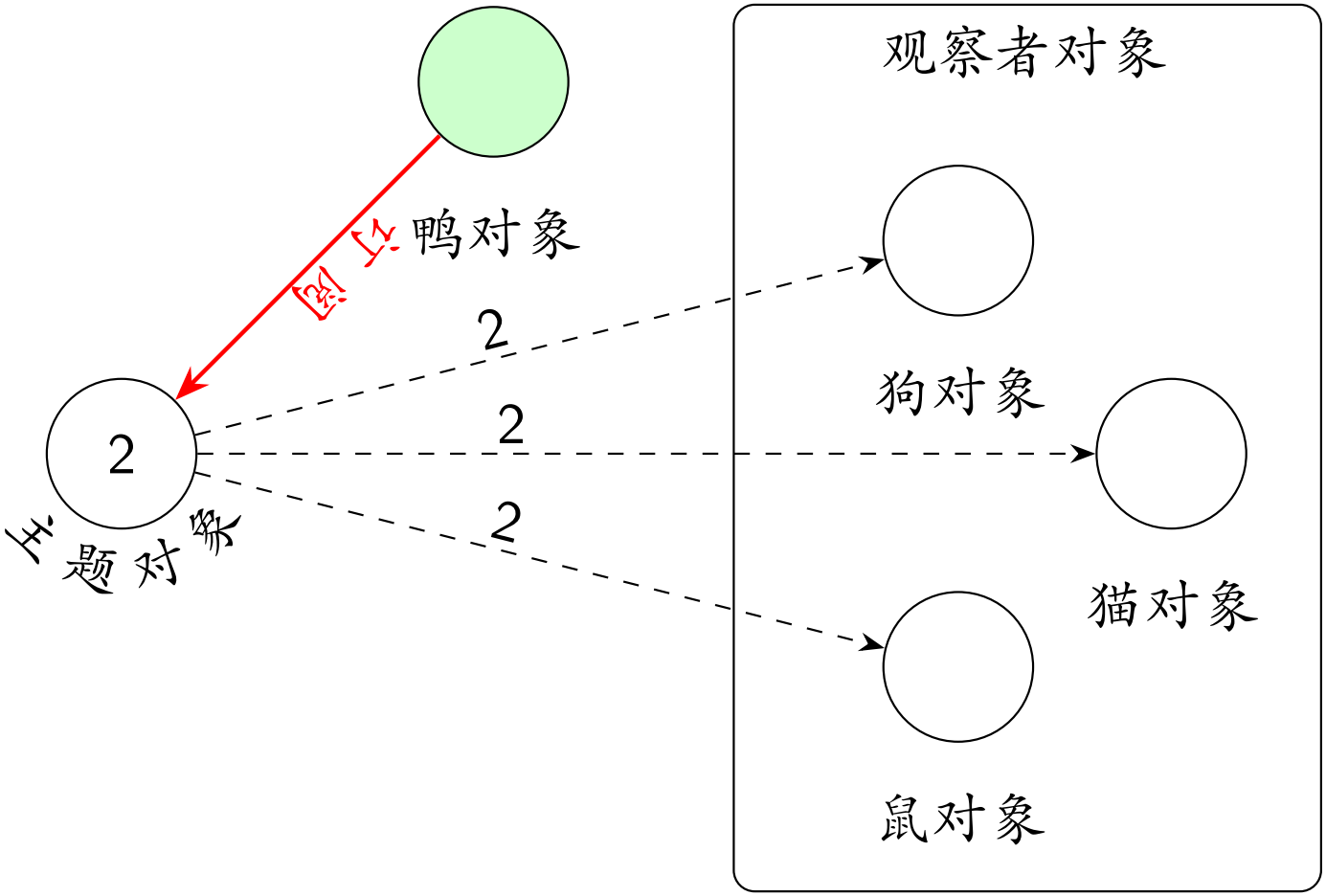
观察者模式：从报纸订阅开始

1. 报社的业务是出版报纸
2. 向某家报社订阅报纸，以后只要他们有新报纸出版，就会给你送来
3. 当你不想再看报的时候，可以取消订阅，他们就不会送报给你
4. 只要报社在运营，就会一直有人向他们订阅或取消订阅报纸

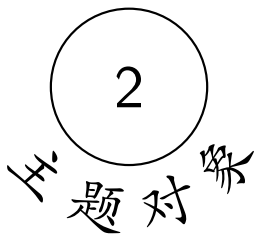
出版者 + 订阅者 = 观察者模式

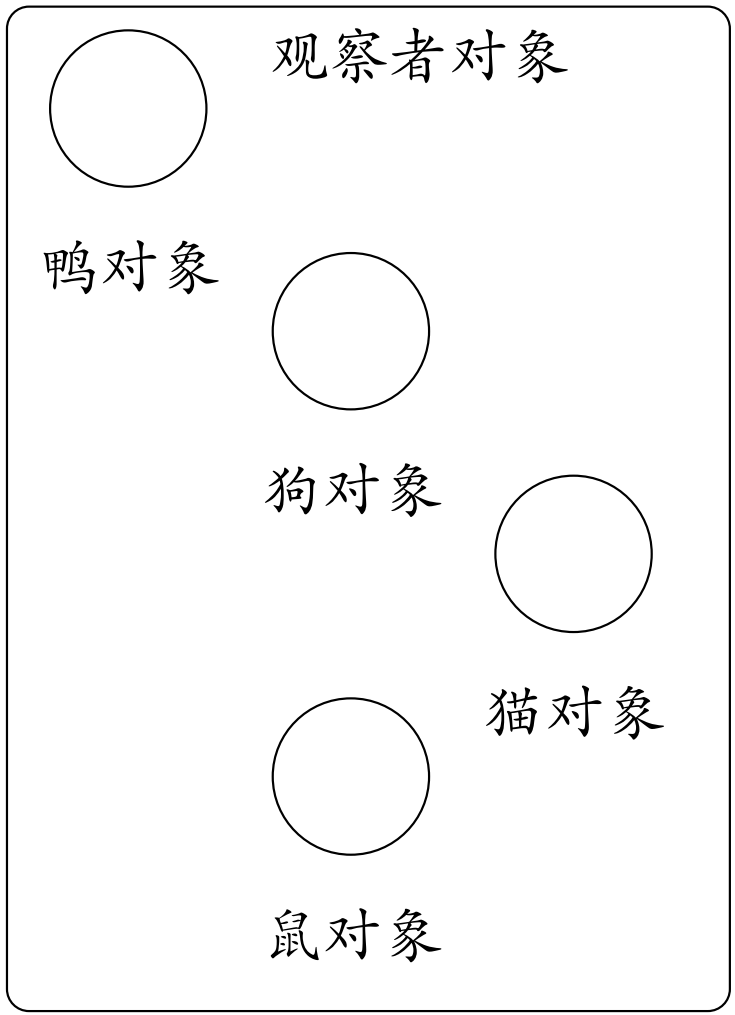


出版者 + 订阅者 = 观察者模式

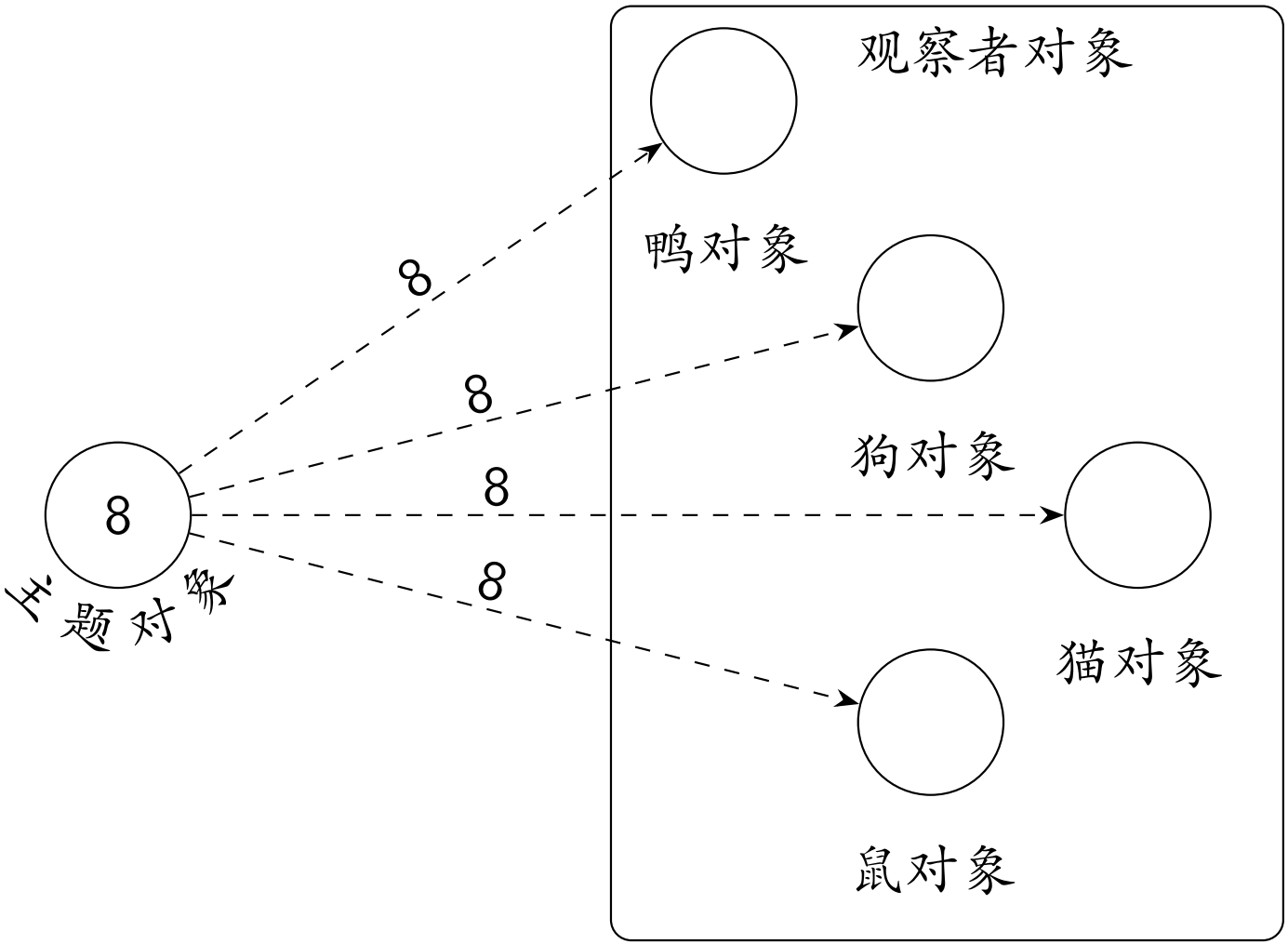


出版者 + 订阅者 = 观察者模式

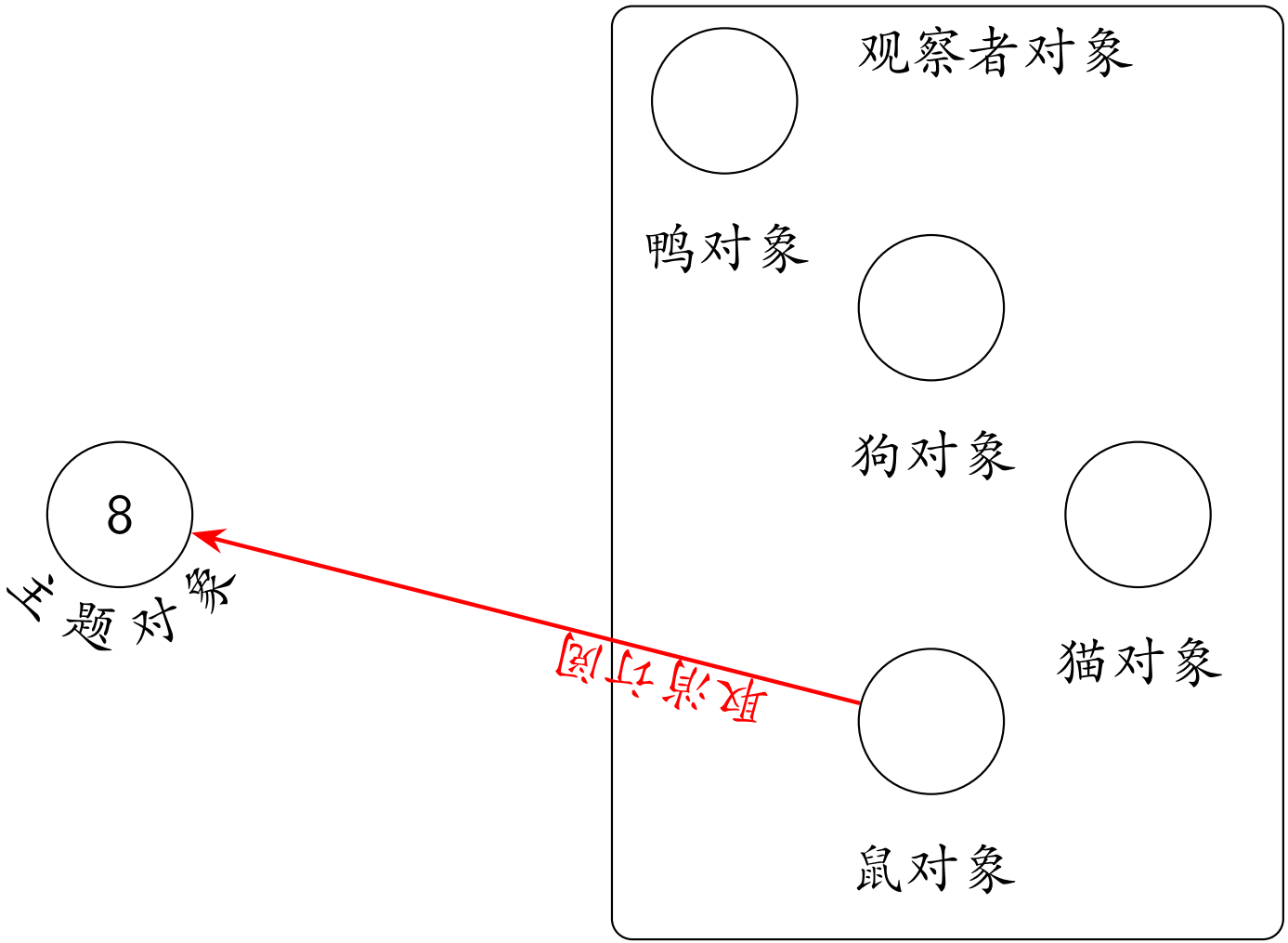




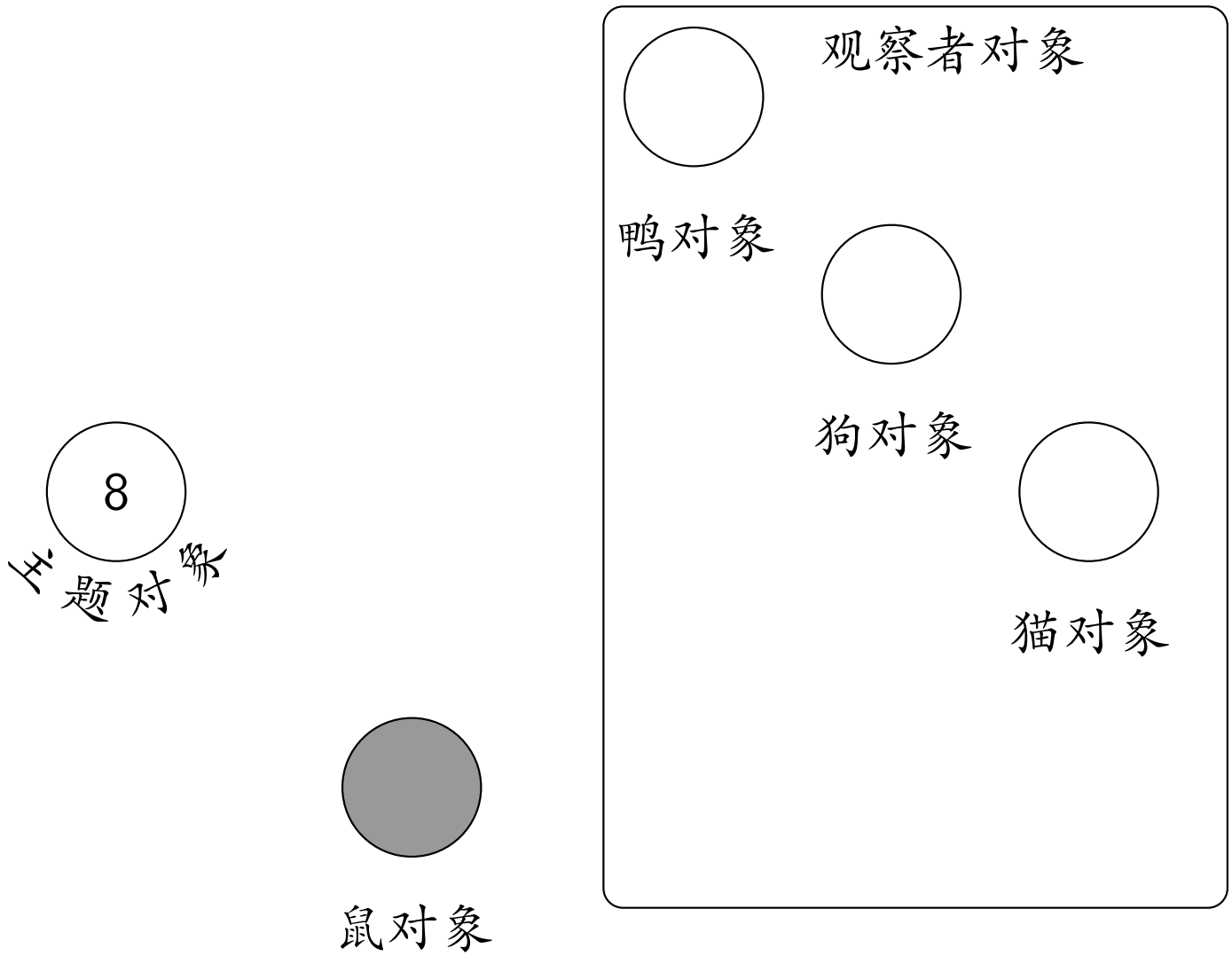
出版者 + 订阅者 = 观察者模式



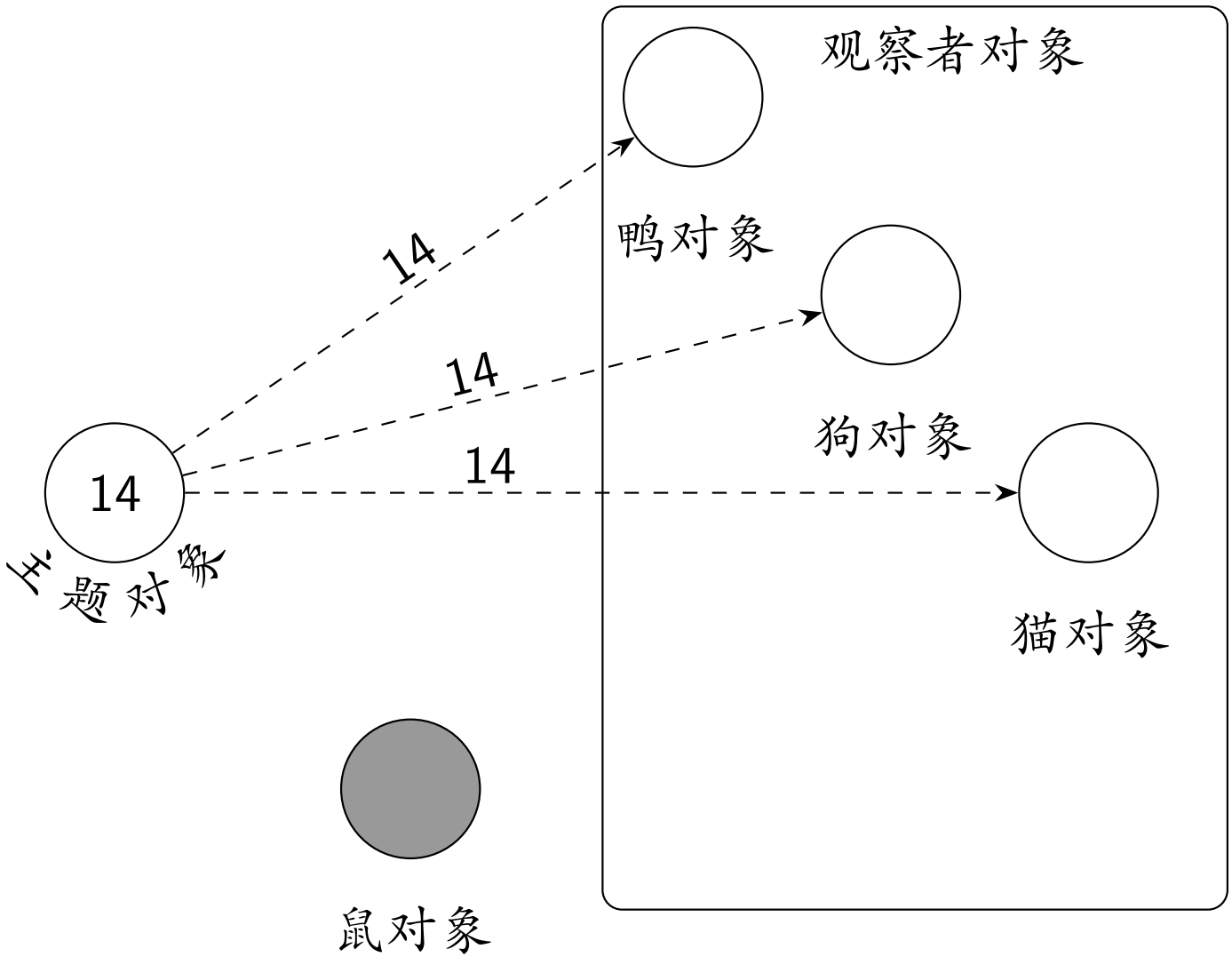
出版者 + 订阅者 = 观察者模式



出版者 + 订阅者 = 观察者模式



出版者 + 订阅者 = 观察者模式

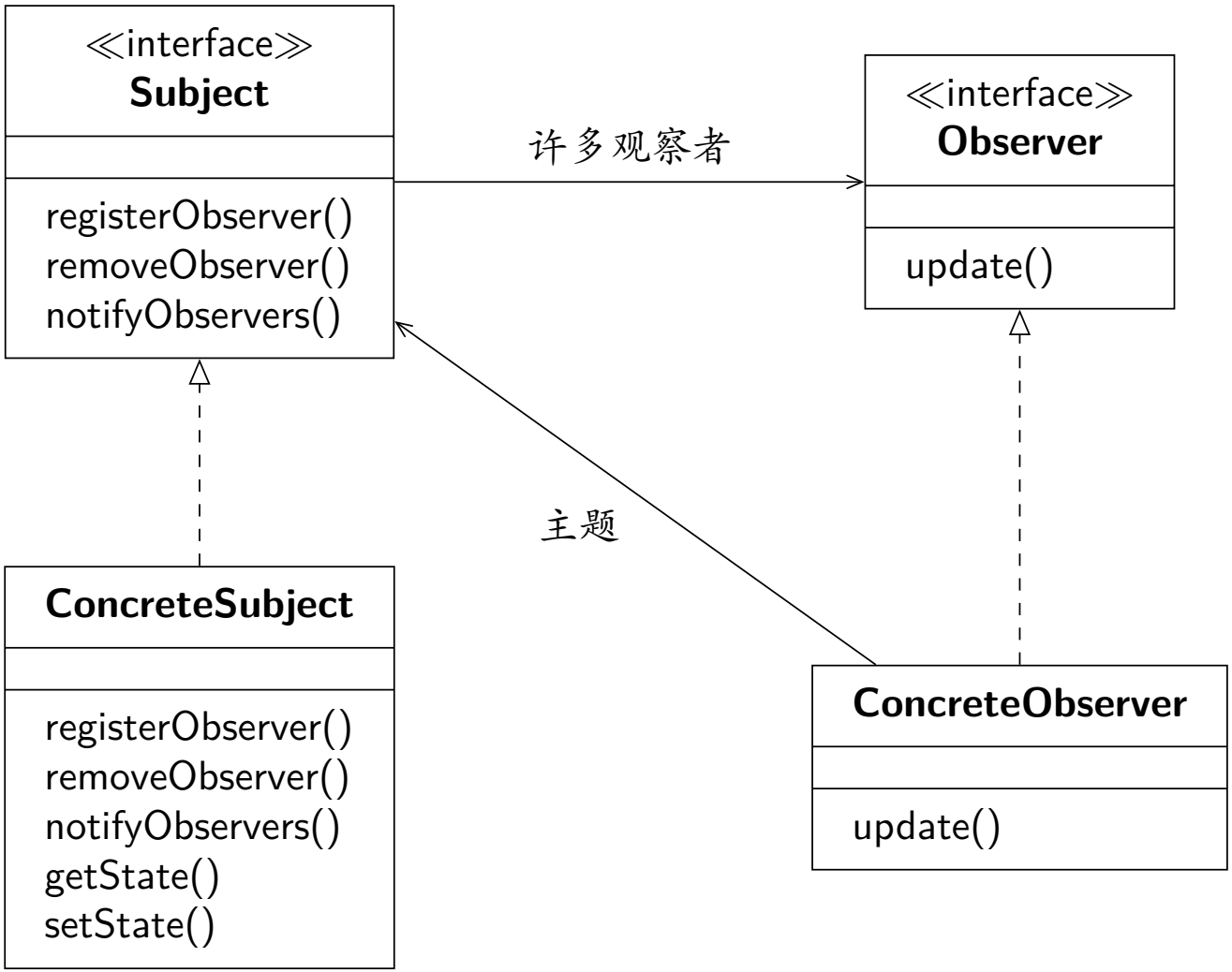


观察者模式: 定义

观察者模式

观察者模式定义了对象间的一对多的依赖关系，当一个对象改变状态时，它所有的依赖者都会收到通知并自动更新

观察者模式: 类图


≪interface≫
Subject
registerObserver()
removeObserver()
notifyObservers()
许多观察者
≪interface≫
Observer
update()
主题
ConcreteSubject
registerObserver()
removeObserver()
notifyObservers()
getState()
setState()
ConcreteObserver
update()

观察者模式使得对象间松耦合

松耦合的威力 — 彼此能够交互，但互相知之甚少

- 主题对象只知道观察者对象实现了一个接口
- 可以在任何时刻添加各种观察者对象
- 增加新的观察者时，不需要更改主题对象
- 可以独立复用主题对象或者观察者对象
- 主题对象或观察者对象发生的改动不会影响对方

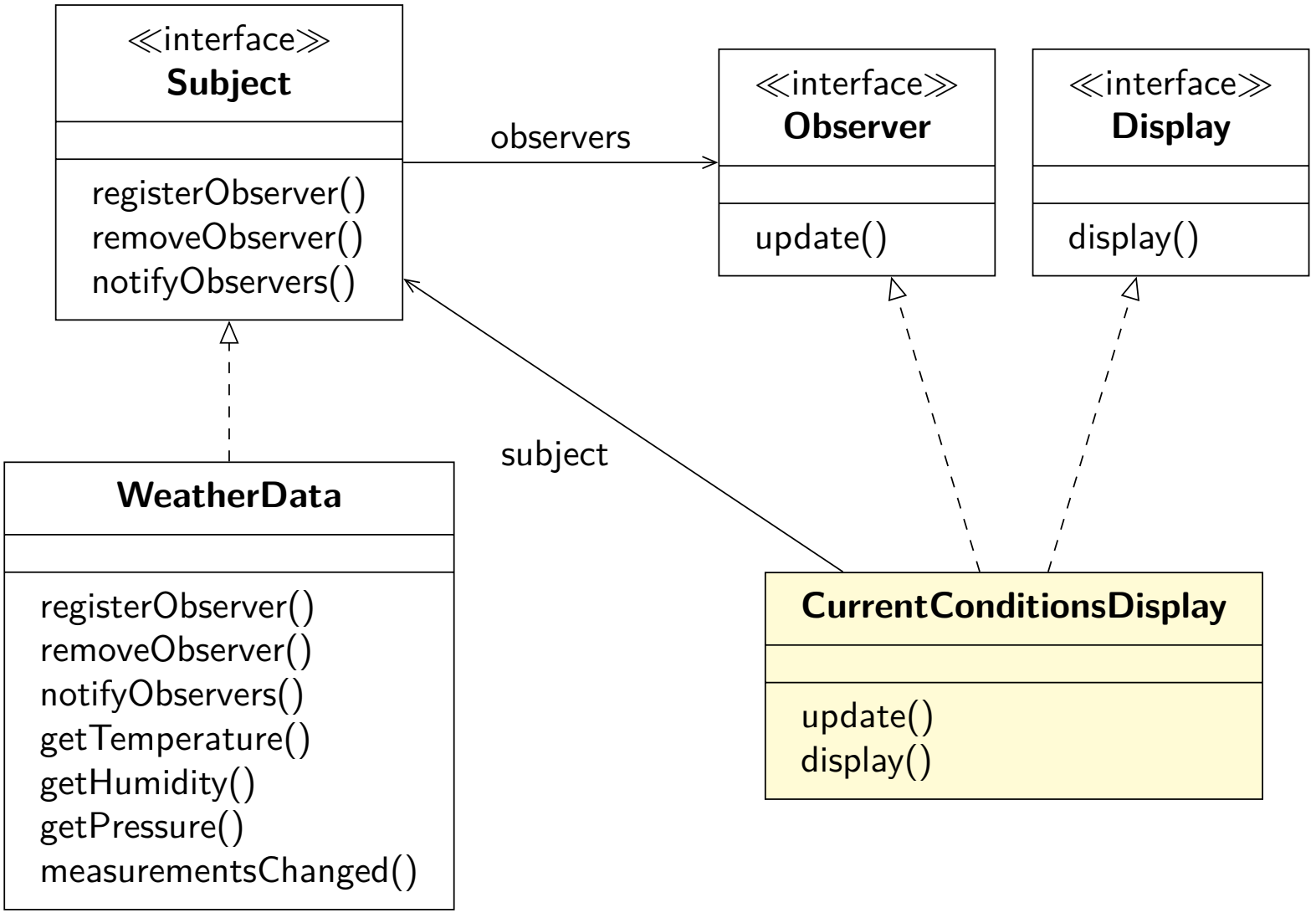
观察者模式使得对象间松耦合

设计原则：松耦合

尽最大可能将要交互的对象设计为松耦合的

松耦合的设计允许我们构建灵活的 OO 系统，以应对未来的变动，因为松耦合可以使对象间的彼此依赖降低

设计气象站应用


≪interface≫
Subject
registerObserver()
removeObserver()
notifyObservers()
observers
≪interface≫
Observer
update()
≪interface≫
Display
display()
subject
WeatherData
registerObserver()
removeObserver()
notifyObservers()
getTemperature()
getHumidity()
getPressure()
measurementsChanged()
CurrentConditionsDisplay
update()
display()

实现气象站应用

建立接口

```
1  public interface Subject {
2      public void registerObserver(Observer o);
3      public void removeObserver(Observer o);
4      public void notifyObservers();
5  }
6  public interface Observer {
7      public void update(float temp, float humidity,
8        float pressure);
9  }
10 public interface Display{
11     public void display();
12 }
```

实现气象站应用

WeatherData 对象

```
1  public class WeatherData implements Subject {
2    private ArrayList observers;
3    private float temperature;
4    private float humidity;
5    private float pressure;
6
7    public WeatherData() {
8      observers = new ArrayList();
9    }
10
11   public void registerObserver(Observer o) {
12     observers.add(o);
13   }
```

实现气象站应用

WeatherData 对象

```
14    public void removeObserver(Observer o) {
15      int i = observers.indexOf(o);
16      if (i >= 0) {
17        observers.remove(i);
18      }
19    }
20
21    public void notifyObservers() {
22      for (int i = 0; i < observers.size(); i++) {
23        Observer observer = (Observer)observers.get(i);
24        observer.update(temperature, humidity, pressure);
25      }
26    }
```

实现气象站应用

WeatherData 对象

```
27    public void measurementsChanged() {
28        notifyObservers();
29    }
30
31    public void setMeasurements(float temperature,
32      float humidity, float pressure) {
33        this.temperature = temperature;
34        this.humidity = humidity;
35        this.pressure = pressure;
36        measurementsChanged();
37    }
38    // other methods ...
39  }
```

实现气象站应用

显示当前天气布告板类

```
1  public class CurrentConditionsDisplay
2    implements Observer, Display{
3    private float temperature;
4    private float humidity;
5    private Subject weatherData;
6
7    public CurrentConditionsDisplay(Subject weatherData) {
8      this.weatherData = weatherData;
9      weatherData.registerObserver(this);
10   }
```

实现气象站应用

显示当前天气布告板类

```
11    public void update(float temperature, float humidity,
12      float pressure) {
13      this.temperature = temperature;
14      this.humidity = humidity;
15      display();
16    }
17
18    public void display() {
19      System.out.println("Current conditions:"+ temperature
20        + "F degrees and " + humidity + "% humidity");
21    }
22  }
```

实现气象站应用

测试

```
1  public class WeatherStation {
2    public static void main(String[] args) {
3      WeatherData wd = new WeatherData();
4
5      CurrentConditionsDisplay cd =
6        new CurrentConditionsDisplay(wd);
7      StatisticsDisplay sd = new StatisticsDisplay(wd);
8      ForecastDisplay fd = new ForecastDisplay(wd);
9
10     wd.setMeasurements(80, 65, 30.4f);
11     wd.setMeasurements(82, 70, 29.2f);
12     wd.setMeasurements(78, 90, 29.2f);
13   }
14 }
```

实现气象站应用

输出结果

```
%java WeatherStation
Current conditions: 80.0F degrees and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0
Forecast: Improving weather on the way!
Current conditions: 82.0F degrees and 70.0% humidity
Avg/Max/Min temperature = 81.0/82.0/80.0
Forecast: Watch out for cooler, rainy weather
Current conditions: 78.0F degrees and 90.0% humidity
Avg/Max/Min temperature = 80.0/82.0/78.0
Forecast: More of the same
%
```

新的需求: 酷热指数布告板

系统开发完成后，用户来电告知其需要酷热指数布告板。
酷热指数是一个结合温度 t 和湿度 h 的指数，用于显示人的温度感受。酷热指数的计算公式:

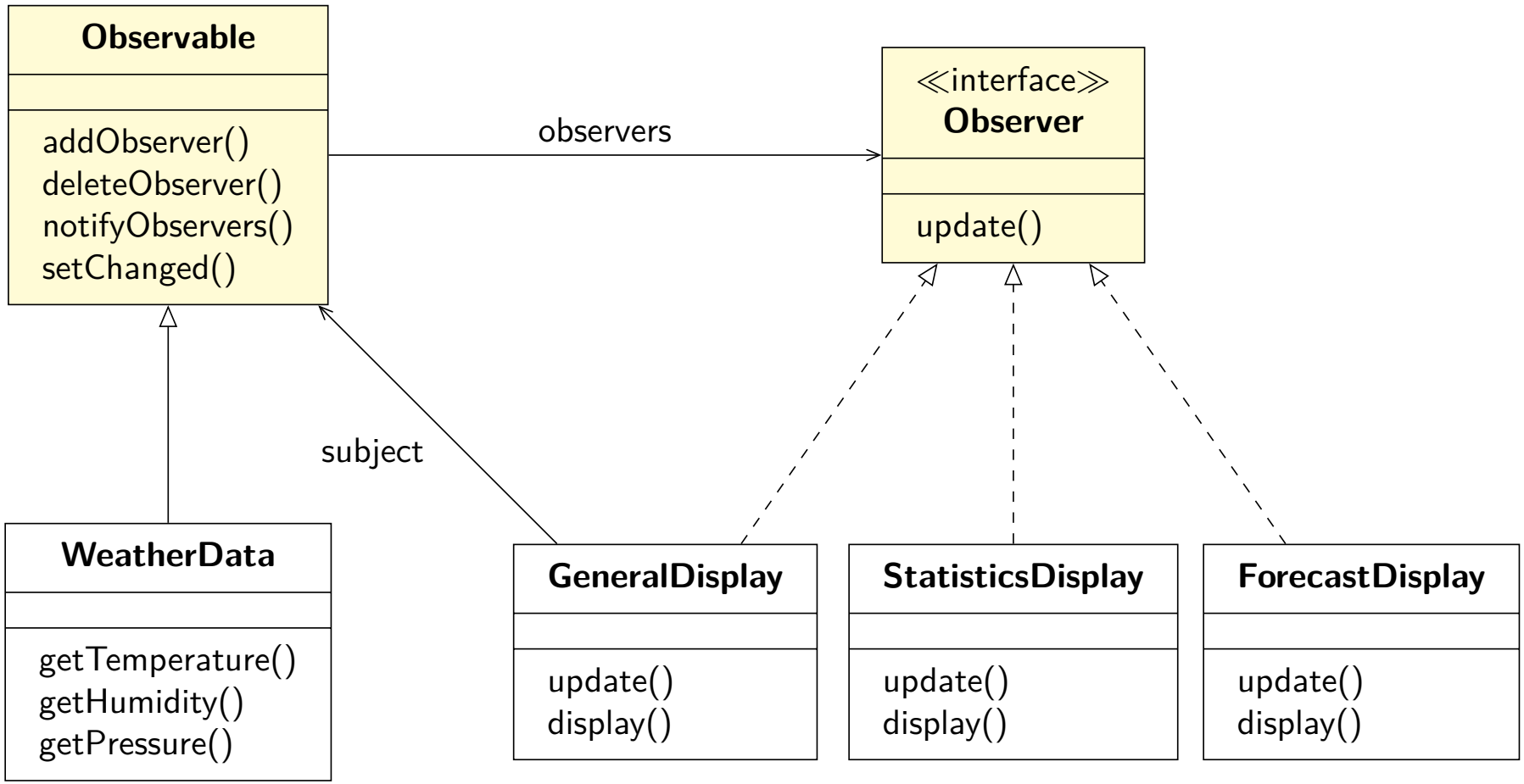
$$
\begin{aligned}
heatindex = \\
& 16.923 + 1.85212 \times 10^{-1} \times t + 5.37941 \times h - 1.00254 \times 10^{-1} \times t \times h + \\
& 9.41695 \times 10^{-3} \times t^2 + 7.28898 \times 10^{-3} \times h^2 + 3.45372 \times 10^{-4} \times t^2 \times h - \\
& 8.14971 \times 10^{-4} \times t \times h^2 + 1.02102 \times 10^{-5} \times t^2 \times h^2 - 3.8646 \times 10^{-5} \times t^3 + \\
& 2.91583 \times 10^{-5} \times h^3 + 1.42721 \times 10^{-6} \times t^3 \times h + 1.97483 \times 10^{-7} \times t \times h^3 - \\
& 2.18429 \times 10^{-8} \times t^3 \times h^2 + 8.43296 \times 10^{-10} \times t^2 \times h^3 - 4.81975 \times 10^{-11} \times \\
& t^3 \times h^3
\end{aligned}
$$

新的需求: 酷热指数布告板

新的输出结果

```
%java WeatherStation
Current conditions: 80.0F degrees and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0
Forecast: Improving weather on the way!
Heat index is 82.95535
Current conditions: 82.0F degrees and 70.0% humidity
Avg/Max/Min temperature = 81.0/82.0/80.0
Forecast: Watch out for cooler, rainy weather
Heat index is 86.90124
Current conditions: 78.0F degrees and 90.0% humidity
Avg/Max/Min temperature = 80.0/82.0/78.0
Forecast: More of the same
Heat index is 83.64967
```

Java 内置的观察者模式



Java 内置观察者模式如何工作

- 如何成为观察者？

  实现 java.util.Observer 接口，调用 Observable 类的 addObserver 方法

- 被观察者如何发送数据改变的通知？
  1. 首先，调用 setChanged 方法，通告该对象状态已改变
  2. 其次，调用 notifyObservers() 或者 notifyObservers(Object arg)

- 观察者如何接收通知？

  实现 update(Observable o, Object arg) 方法，支持“推”和“拉”两种获得数据的方式

Java 内置观察者模式如何工作

Observable 的部分伪码

```
1  setChanged() {
2    changed = true
3  }
4  notifyObservers(Object arg) {
5    if (changed) {
6      for every observer on the list {
7        call update (this, arg)
8      }
9      changed = false
10   }
11 }
12 notifyObservers() {
13   notifyObservers(null)
14 }
```

用 Java 内置观察者模式实现气象站应用

WeatherData 部分代码

```
1  import java.util.Observable;
2  import java.util.Observer;
3
4  public class WeatherData extends Observable {
5    private float temperature;
6    private float humidity;
7    private float pressure;
8
9    public WeatherData() { }
10   public void measurementsChanged() {
11     setChanged();
12     notifyObservers();
13   }
```

用 Java 内置观察者模式实现气象站应用

WeatherData 部分代码

```
14    public void setMeasurements(float temperature,
15      float humidity, float pressure) {
16      this.temperature = temperature;
17      this.humidity = humidity;
18      this.pressure = pressure;
19      measurementsChanged();
20    }
21
22    public float getTemperature() {
23      return temperature; // pulled by observers
24    }
25    // other methods including getHumidity() getPressure()
26  }
```

用 Java 内置观察者模式实现气象站应用

新的 CurrentConditionsDisplay

```
1  import java.util.Observable;
2  import java.util.Observer;
3
4  public class CurrentConditionsDisplay
5    implements Observer, Display{
6    Observable observable;
7    private float temp;
8    private float humidity;
9
10   public CurrentConditionsDisplay(Observable observable){
11     this.observable = observable ;
12     observable.addObserver(this) ;
13   }
```

用 Java 内置观察者模式实现气象站应用

新的 CurrentConditionsDisplay

```
14    public void update(Observable obs, Object arg) {
15      if (obs instanceof WeatherData) {
16        WeatherData weatherData = (WeatherData)obs;
17        this.temp = weatherData.getTemperature();
18        this.humidity = weatherData.getHumidity();
19        display();
20      }
21    }
22    public void display() {
23      System.out.println("Current conditions: " + temp
24        + "F degrees and " + humidity + "% humidity");
25    }
26  }
```

用 Java 内置观察者模式实现气象站应用

运行新代码，观察输出和之前版本的差别

```
%java WeatherStation
Forecast: Improving weather on the way!
Avg/Max/Min temperature = 80.0/80.0/80.0
Current conditions: 80.0F degrees and 65.0% humidity
Forecast: Watch out for cooler, rainy weather
Avg/Max/Min temperature = 81.0/82.0/80.0
Current conditions: 82.0F degrees and 70.0% humidity
Forecast: More of the same
Avg/Max/Min temperature = 80.0/82.0/78.0
Current conditions: 78.0F degrees and 90.0% humidity
```

相同的计算结果，不同的输出次序，为什么？有无危害？

java.util.Observable 的问题

- Observable 是一个类 $\Longrightarrow$ 用户必须设计一个类继承它
  - 某类难以同时具有 Observable 和另一个超类的行为
  - 没有 Observable 接口，无法建立用户自己的实现
- Observable 将关键的方法 setChanged 保护起来
  - 只能继承 Observable，不能创建 Observable 实例聚合到自己的对象中
  - 违反了"多用聚合、少用继承"的设计原则

小结

关于观察者模式:

- 观察者模式定义了对象之间一对多的关系
- 主题（被观察者）用一个共同的接口更新观察者
- 主题和观察者之间是松耦合的，主题不知道具体观察者的细节，只知道其实现了观察者接口
- 可以用推（push）或拉（pull）的方式将数据传给观察者
- 有多个观察者时，不可以依赖特定的通知次序
- Java 的 Observable 实现有一些问题

小结

关于设计原则:

1. 封装变化
2. 多用聚合、少用继承
3. 针对接口编程，不针对实现编程
4. 尽最大可能将要交互的对象设计为松耦合的