

05 观察者模式、抽象工厂模式

C-3 创建: 张林伟, 最后修改: 张林伟 2018-10-19 18:07

观察者模式 (发布-订阅模式)

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主题对象。这个主题对象在状态发生变化时, 会通知所有的观察者对象, 使它们能够自动更新自己。

Demo

主题

^ 代码块 Java

```
1 public class Subject {
2
3     private List<Observer> observers = new ArrayList<>();
4
5     @Setter
6     @Getter
7     private String name;
8
9     @Getter
10    private boolean state;
11
12    public Subject(String name, boolean state) {
13        this.name = name;
14        this.state = state;
15    }
16
17    public Subject(Observer... observers) {
18        for (int i = 0; i < observers.length; i++) {
19            this.observers.add(observers[i]);
20        }
21    }
22
23    public void setState(boolean state) {
24        if (state) {
25            System.out.println("今天国庆放假");
26        } else {
27            System.out.println("今天未到国庆放假");
28        }
29        this.state = state;
30        notifyAllObservers();
31    }
32
33    public void attach(Observer observer) {
34        this.observers.add(observer);
35    }
36
37    public void notifyAllObservers() {
38        observers.forEach(observer -> observer.listen());
39    }
40 }
```

观察者

^ 代码块 Java

```
1 // Observer.java
2 public abstract class Observer {
3
4     protected Subject subject;
5
6     public abstract void listen();
7 }
8
9 // StudentObserver.java
```

```
10 public class StudentObserver extends Observer {
11
12     public StudentObserver(Subject subject) {
13         this.subject = subject;
14         this.subject.attach(this);
15     }
16
17     @Override public void listen() {
18         System.out.println("学生：放假咯。");
19     }
20 }
21
22 // WorkerObserver.java
23 public class WorkerObserver extends Observer {
24
25     public WorkerObserver(Subject subject) {
26         this.subject = subject;
27         this.subject.attach(this);
28     }
29
30     @Override public void listen() {
31         System.out.println("工人：终于可以休息了。");
32     }
33 }
```

客户端

代码块

Java

```
1 public class Client {
2
3     public static void main(String[] args) {
4         Subject subject = new Subject("国庆节", false);
5
6         new StudentObserver(subject);
7         new WorkerObserver(subject);
8
9         subject.setState(true);
10    }
11 }
```

抽象工厂模式

提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。

系统的产品有多于一个的产品族，而系统只消费其中某一族的产品。

Demo

产品父类

代码块

Java

```
1 // AbstractProductA.java
2 public abstract class AbstractProductA {
3     public abstract void use();
4 }
5
6 // AbstractProductB.java
7 public abstract class AbstractProductB {
8     public abstract void eat();
9 }
```

产品具体类

代码块

```
1 // ProductA1.java
2 public class ProductA1 extends AbstractProductA {
3     @Override public void use() {
4         System.out.println("use productA1");
5     }
6 }
7
8 // ProductA2.java
9 public class ProductA2 extends AbstractProductA {
10     @Override public void use() {
11         System.out.println("use productA2");
12     }
13 }
14
15 // ProductB1.java
16 public class ProductB1 extends AbstractProductB {
17     @Override public void eat() {
18         System.out.println("eat productB1");
19     }
20 }
21
22 // ProductB2.java
23 public class ProductB2 extends AbstractProductB {
24     @Override public void eat() {
25         System.out.println("eat productB2");
26     }
27 }
```

工厂父类

^ 代码块

Java

```
1 public abstract class AbstractFactory {
2     public abstract AbstractProductA createProductA();
3     public abstract AbstractProductB createProductB();
4 }
```

工厂具体类

^ 代码块

Java

```
1 // ConcreteFactory1.java
2 public class ConcreteFactory1 extends AbstractFactory {
3
4     @Override public AbstractProductA createProductA() {
5         return new ProductA1();
6     }
7
8     @Override public AbstractProductB createProductB() {
9         return new ProductB1();
10    }
11 }
12
13 // ConcreteFactory2.java
14 public class ConcreteFactory2 extends AbstractFactory {
15
16     @Override public AbstractProductA createProductA() {
17         return new ProductA2();
18     }
19
20     @Override public AbstractProductB createProductB() {
21         return new ProductB2();
22     }
23 }
```

客户端

^

```
1 public class Client {
2
3     public static void main(String[] args) {
4         AbstractFactory factory1 = new ConcreteFactory1();
5
6         AbstractProductA productA = factory1.createProductA();
7         AbstractProductB productB = factory1.createProductB();
8
9         productA.use();
10        productB.eat();
11    }
12 }
```

🔒 仅供内部使用，未经授权，切勿外传