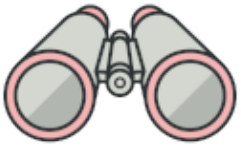

# Java 观察者模式讲解和代码示例

**观察者**是一种行为设计模式，允许一个对象将其状态的改变通知其他对象

观察者模式提供了一种作用于任何实现了订阅者接口的对象的机制，可对其事件进行订阅和取消订阅。

进一步了解观察者模式 →

## 在 Java 中使用模式

**复杂度：** ★★☆

**流行度：** ★★★

**使用示例：**观察者模式在 Java 代码中很常见，特别是在 GUI 组件中。它提供了在不与其他对象所属类耦合的情况下对其事件做出反应的方式。

这里是核心 Java 程序库中该模式的一些示例：

- `java.util.Observer` / `java.util.Observable`（极少在真实世界中使用）
- `java.util.EventListener` 的所有实现（几乎广泛存在于 Swing 组件中）
- `javax.servlet.http.HttpSessionBindingListener`
- `javax.servlet.http.HttpSessionAttributeListener`
- `javax.faces.event.PhaseListener`

**识别方法：**该模式可以通过将对象存储在列表中的订阅方法，和对于面向该列表中对象的更新方法的调用来识别。

## 导航



# 事件订阅

在本例中，观察者模式在文本编辑器的对象之间建立了间接的合作关系。每当`编辑器`（Editor）对象改变时，它都会通知其订阅者。`邮件通知监听器`（EmailNotificationListener）和`日志开启监听器`（LogOpenListener）都将通过执行其基本行为来对这些通知做出反应。

订阅者类不与编辑器类相耦合，且能在需要时在其他应用中复用。`编辑器`类仅依赖于抽象订阅者接口。这样就能允许在不改变编辑器代码的情况下添加新的订阅者类型。

# publisher

### publisher/EventManager.java: 基础发布者

```
package refactoring_guru.observer.example.publisher;

import refactoring_guru.observer.example.listeners.EventListener;

import java.io.File;
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class EventManager {
    Map<String, List<EventListener>> listeners = new HashMap<>();

    public EventManager(String... operations) {
        for (String operation : operations) {
            this.listeners.put(operation, new ArrayList<>());
        }
    }

    public void subscribe(String eventType, EventListener listener) {
        List<EventListener> users = listeners.get(eventType);
        users.add(listener);
    }

    public void unsubscribe(String eventType, EventListener listener) {
        List<EventListener> users = listeners.get(eventType);
        users.remove(listener);
    }

    public void notify(String eventType, File file) {
        List<EventListener> users = listeners.get(eventType);
        for (EventListener listener : users) {
            listener.update(eventType, file);
        }
    }
}
```

# editor

## editor/Editor.java: 具体发布者，由其他对象追踪

```
package refactoring_guru.observer.example.editor;

import refactoring_guru.observer.example.publisher.EventManager;

import java.io.File;

public class Editor {
    public EventManager events;
    private File file;

    public Editor() {
        this.events = new EventManager("open", "save");
    }
```

```
    public void openFile(String filePath) {
        this.file = new File(filePath);
        events.notify("open", file);
    }

    public void saveFile() throws Exception {
        if (this.file != null) {
            events.notify("save", file);
        } else {
            throw new Exception("Please open a file first.");
        }
    }
}
```

# listeners

## listeners/EventListener.java: 通用观察者接口

```
package refactoring_guru.observer.example.listeners;

import java.io.File;

public interface EventListener {
    void update(String eventType, File file);
}
```

## listeners/EmailNotificationListener.java: 收到通知后发送邮件

```
package refactoring_guru.observer.example.listeners;

import java.io.File;

public class EmailNotificationListener implements EventListener {
    private String email;

    public EmailNotificationListener(String email) {
        this.email = email;
    }

    @Override
    public void update(String eventType, File file) {
        System.out.println("Email to " + email + ": Someone has performed " + eventType + " op
    }
}
```

### listeners/LogOpenListener.java: 收到通知后在日志中记录一条消息

```
package refactoring_guru.observer.example.listeners;

import java.io.File;

public class LogOpenListener implements EventListener {
    private File log;

    public LogOpenListener(String fileName) {
        this.log = new File(fileName);
    }

    @Override
    public void update(String eventType, File file) {
        System.out.println("Save to log " + log + ": Someone has performed " + eventType + " o
    }
}
```

### Demo.java: 初始化代码

```
package refactoring_guru.observer.example;

import refactoring_guru.observer.example.editor.Editor;
import refactoring_guru.observer.example.listeners.EmailNotificationListener;
import refactoring_guru.observer.example.listeners.LogOpenListener;

public class Demo {
    public static void main(String[] args) {
        Editor editor = new Editor();
        editor.events.subscribe("open", new LogOpenListener("/path/to/log/file.txt"));
        editor.events.subscribe("save", new EmailNotificationListener("admin@example.com"));

        try {
            editor.openFile("test.txt");
            editor.saveFile();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

### OutputDemo.txt: 执行结果

```
Save to log \path\to\log\file.txt: Someone has performed open operation with the following fil
Email to admin@example.com: Someone has performed save operation with the following file: test
```

**继续阅读**

Java 状态模式讲解和代码示例 →

**返回**

← Java 备忘录模式讲解和代码示例

# 观察者在其他编程语言中的实现

主页
重构
设计模式
会员专属内容
论坛
联系我们


Khmelnitske shosse 19 / 27, Kamianets-Podilskyi, 乌克兰, 32305
Email: support@refactoring.guru
图片作者： Dmitry Zhart

条款与政策
隐私政策
内容使用政策