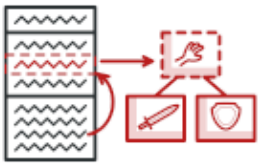

Java 策略模式讲解和代码示例

策略是一种行为设计模式，它将一组行为转换为对象，并使其在原始上下文对象内部能够相互替换。

原始对象被称为上下文，它包含指向策略对象的引用并将执行行为的任务分派给策略对象。为了改变上下文完成其工作的方式，其他对象可以使用另一个对象来替换当前链接的策略对象。

进一步了解策略模式 →

在 Java 中使用模式

复杂度： ★☆☆

流行度： ★★★

使用示例：策略模式在 Java 代码中很常见。它经常在各种框架中使用，能在不扩展类的情况下向用户提供改变其行为的方式。

Java 8 开始支持 lambda 方法，它可作为一种替代策略模式的简单方式。

这里有一些核心 Java 程序库中策略模式的示例：

- 对 `java.util.Comparator#compare()` 的调用来自 `Collections#sort()`.
- `javax.servlet.http.HttpServlet`：`service()` 方法，还有所有接受 `HttpServletRequest` 和 `HttpServletResponse` 对象作为参数的 `doXXX()` 方法。

- `javax.servlet.Filter#doFilter()`

识别方法：策略模式可以通过允许嵌套对象完成实际工作的方法以及允许将该对象替换为不同对象的设置器来识别。

导航

简介

电子商务应用中的支付方法

strategies

PayStrategy

PayByPayPal

PayByCreditCard

CreditCard

Order

Demo

OutputDemo

电子商务应用中的支付方法

在本例中，策略模式被用于在电子商务应用中实现各种支付方法。客户选中希望购买的商品后需要选择一种支付方式：Paypal 或者信用卡。

具体策略不仅会完成实际的支付工作，还会改变支付表单的行为，并在表单中提供相应的字段来记录支付信息。

strategies

strategies/PayStrategy.java: 通用的支付方法接口

```
package refactoring_guru.strategy.example.strategies;

/**
 * Common interface for all strategies.
 */
public interface PayStrategy {
    boolean pay(int paymentAmount);
```

```
    void collectPaymentDetails();
}
```

strategies/PayByPayPal.java: 使用 PayPal 支付

```
package refactoring_guru.strategy.example.strategies;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Map;

/**
 * Concrete strategy. Implements PayPal payment method.
 */
public class PayByPayPal implements PayStrategy {
    private static final Map<String, String> DATA_BASE = new HashMap<>();
    private final BufferedReader READER = new BufferedReader(new InputStreamReader(System.in))
    private String email;
    private String password;
    private boolean signedIn;

    static {
        DATA_BASE.put("amanda1985", "amanda@ya.com");
        DATA_BASE.put("qwerty", "john@amazon.eu");
    }

    /**
     * Collect customer's data.
     */
    @Override
    public void collectPaymentDetails() {
        try {
            while (!signedIn) {
                System.out.print("Enter the user's email: ");
                email = READER.readLine();
                System.out.print("Enter the password: ");
                password = READER.readLine();
                if (verify()) {
                    System.out.println("Data verification has been successful.");
                } else {
                    System.out.println("Wrong email or password!");
                }
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private boolean verify() {
```

```
        setSignedIn(email.equals(DATA_BASE.get(password)));
        return signedIn;
    }

    /**
     * Save customer data for future shopping attempts.
     */
    @Override
    public boolean pay(int paymentAmount) {
        if (signedIn) {
            System.out.println("Paying " + paymentAmount + " using PayPal.");
            return true;
        } else {
            return false;
        }
    }

    private void setSignedIn(boolean signedIn) {
        this.signedIn = signedIn;
    }
}
```

strategies/PayByCreditCard.java: 使用信用卡支付

```
package refactoring_guru.strategy.example.strategies;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 * Concrete strategy. Implements credit card payment method.
 */
public class PayByCreditCard implements PayStrategy {
    private final BufferedReader READER = new BufferedReader(new InputStreamReader(System.in))
    private CreditCard card;

    /**
     * Collect credit card data.
     */
    @Override
    public void collectPaymentDetails() {
        try {
            System.out.print("Enter the card number: ");
            String number = READER.readLine();
            System.out.print("Enter the card expiration date 'mm/yy': ");
            String date = READER.readLine();
            System.out.print("Enter the CVV code: ");
            String cvv = READER.readLine();
            card = new CreditCard(number, date, cvv);
```

```
            // Validate credit card number...

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    /**
     * After card validation we can charge customer's credit card.
     */
    @Override
    public boolean pay(int paymentAmount) {
        if (cardIsPresent()) {
            System.out.println("Paying " + paymentAmount + " using Credit Card.");
            card.setAmount(card.getAmount() - paymentAmount);
            return true;
        } else {
            return false;
        }
    }

    private boolean cardIsPresent() {
        return card != null;
    }
}
```

strategies/CreditCard.java: 信用卡类

```
package refactoring_guru.strategy.example.strategies;

/**
 * Dummy credit card class.
 */
public class CreditCard {
    private int amount;
    private String number;
    private String date;
    private String cvv;

    CreditCard(String number, String date, String cvv) {
        this.amount = 100_000;
        this.number = number;
        this.date = date;
        this.cvv = cvv;
    }

    public void setAmount(int amount) {
        this.amount = amount;
    }
```

```
    public int getAmount() {
        return amount;
    }
}
```

order/Order.java: 订单类

```
package refactoring_guru.strategy.example.order;

import refactoring_guru.strategy.example.strategies.PayStrategy;

/**
 * Order class. Doesn't know the concrete payment method (strategy) user has
 * picked. It uses common strategy interface to delegate collecting payment data
 * to strategy object. It can be used to save order to database.
 */
public class Order {
    private int totalCost = 0;
    private boolean isClosed = false;

    public void processOrder(PayStrategy strategy) {
        strategy.collectPaymentDetails();
        // Here we could collect and store payment data from the strategy.
    }

    public void setTotalCost(int cost) {
        this.totalCost += cost;
    }

    public int getTotalCost() {
        return totalCost;
    }

    public boolean isClosed() {
        return isClosed;
    }

    public void setClosed() {
        isClosed = true;
    }
}
```

Demo.java: 客户端代码

```
package refactoring_guru.strategy.example;

import refactoring_guru.strategy.example.order.Order;
```

```
import refactoring_guru.strategy.example.strategies.PayByCreditCard;
import refactoring_guru.strategy.example.strategies.PayByPayPal;
import refactoring_guru.strategy.example.strategies.PayStrategy;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Map;

/**
 * World first console e-commerce application.
 */
public class Demo {
    private static Map<Integer, Integer> priceOnProducts = new HashMap<>();
    private static BufferedReader reader = new BufferedReader(new InputStreamReader(System.in)
    private static Order order = new Order();
    private static PayStrategy strategy;

    static {
        priceOnProducts.put(1, 2200);
        priceOnProducts.put(2, 1850);
        priceOnProducts.put(3, 1100);
        priceOnProducts.put(4, 890);
    }

    public static void main(String[] args) throws IOException {
        while (!order.isClosed()) {
            int cost;

            String continueChoice;
            do {
                System.out.print("Please, select a product:" + "\n" +
                        "1 - Mother board" + "\n" +
                        "2 - CPU" + "\n" +
                        "3 - HDD" + "\n" +
                        "4 - Memory" + "\n");
                int choice = Integer.parseInt(reader.readLine());
                cost = priceOnProducts.get(choice);
                System.out.print("Count: ");
                int count = Integer.parseInt(reader.readLine());
                order.setTotalCost(cost * count);
                System.out.print("Do you wish to continue selecting products? Y/N: ");
                continueChoice = reader.readLine();
            } while (continueChoice.equalsIgnoreCase("Y"));

            if (strategy == null) {
                System.out.println("Please, select a payment method:" + "\n" +
                        "1 - PalPay" + "\n" +
                        "2 - Credit Card");
                String paymentMethod = reader.readLine();

                // Client creates different strategies based on input from user,
                // application configuration, etc.
                if (paymentMethod.equals("1")) {
```

```
                    strategy = new PayByPayPal();
                } else {
                    strategy = new PayByCreditCard();
                }
            }

            // Order object delegates gathering payment data to strategy object,
            // since only strategies know what data they need to process a
            // payment.
            order.processOrder(strategy);

            System.out.print("Pay " + order.getTotalCost() + " units or Continue shopping? P/C
            String proceed = reader.readLine();
            if (proceed.equalsIgnoreCase("P")) {
                // Finally, strategy handles the payment.
                if (strategy.pay(order.getTotalCost())) {
                    System.out.println("Payment has been successful.");
                } else {
                    System.out.println("FAIL! Please, check your data.");
                }
                order.setClosed();
            }
        }
    }
}
```

OutputDemo.txt: 执行结果

```
Please, select a product:
1 - Mother board
2 - CPU
3 - HDD
4 - Memory
1
Count: 2
Do you wish to continue selecting products? Y/N: y
Please, select a product:
1 - Mother board
2 - CPU
3 - HDD
4 - Memory
2
Count: 1
Do you wish to continue selecting products? Y/N: n
Please, select a payment method:
1 - PalPay
2 - Credit Card
1
Enter the user's email: user@example.com
Enter the password: qwerty
```

```
Wrong email or password!
Enter user email: amanda@ya.com
Enter password: amanda1985
Data verification has been successful.
Pay 6250 units or Continue shopping?  P/C: p
Paying 6250 using PayPal.
Payment has been successful.
```

继续阅读

Java 模版方法模式讲解和代码示例 →

返回

← Java 状态模式讲解和代码示例

策略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