**Java 策略模式**

@author ixenos

摘要： 定义、策略模式的实现(面向接口编程)、示例、泛型化策略模式

**定义**

　　1.封装算法：定义一组算法，将每个算法都封装起来，并且使他们之间可以互换

　　2.分割行为和环境：对用户屏蔽内部实现，使客户端在调用算法的时候能够互不影响地互换

**策略模式的实现(面向接口编程)**

**方法：**

　　1.**接口多态**：策略模式的用意是针对一组算法，将每个算法封装到具有**共同接口**的**独立的类**中，从而使他们之间可以相互替换

　　2.**具体策略提供不同算法，环境负责维持和查询策略**，把具体策略和环境分割开来，使得算法可以在不影响客户端和环境的情况下修改

**角色分工：**

　　1.**抽象策略角色**：通常由一个接口或抽象类实现

　　2.**具体策略角色**：实现或继承抽象角色，封装对应策略的算法和行为

　　3.**环境角色**：持有一个抽象策略角色的引用

　　4.客户端：通过环境角色执行策略

**示例**

**抽象策略角色：**

1 public interface Strategy {

2

3 public void doSomething();

4 }

**具体策略角色：**

[复制代码](javascript:void(0);)

1 public class StrategyA implements Strategy {

2

3 @Override

4 public void doSomething() {

5 System.out.println("Strategy A do something.");

6 }

7

8 }

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

1 public class StrategyB implements Strategy {

2

3 @Override

4 public void doSomething() {

5 System.out.println("Strategy B do something.");

6 }

7

8 }

[复制代码](javascript:void(0);)

**环境角色：**

[复制代码](javascript:void(0);)

1 public class Context {

2

3 private Strategy strategy;

4

5 public Context(Strategy strategy) { //利用接口多态

6 this.strategy = strategy;

7 }

8

9 public void execute() {

10 this.strategy.doSomething();

11 }

12 }

[复制代码](javascript:void(0);)

**客户端：**

[复制代码](javascript:void(0);)

1 public class StrategyCaller {

2

3 public static void main(String[] args) {

4 Context c = new Context(new StrategyA());

5 c.execute();

6

7 Context c2 = new Context(new StrategyB());

8 c2.execute();

9 }

10 }

[复制代码](javascript:void(0);)

**泛型化策略模式**

**泛型环境角色：**

[复制代码](javascript:void(0);)

1 @SuppressWarnings("all")

2 public class Context<T extends Strategy> {

3 private Strategy s;

4 　 /\*

5 \* 这里允许传递一个Strategy类或其子类的Class对象

6 \* 传递进来的是确切类型的Class对象，持有的对象类型是确定的！

7 \* 类型擦除擦的是类和方法的类型参数，不是你传进来的对象类型！

8 \*/

9 public Context(Class<? extends Strategy> c) {

10 try {

11

12 　 /\*

13 \* c.getName()以 String 的形式返回此 Class 对象所表示的实体（类、接口、数组类、基本类型或 void）名称。

14 \*/

15 s = (T) Class.forName(c.getName()).newInstance(); //普通类型Class对象的newInstance将返回Object对象

16

17 } catch (Exception e) {

18 e.printStackTrace();

19 }

20 }

21 public void execute() {

22 s.doSomething();

23 }

24 }

[复制代码](javascript:void(0);)

 　　　　s = (T) Class.forName(c.getName()).newInstance();

　　　　这里需要强制转换是因为，普通类型Class对象的newInstance将返回Object对象

*详情请看Java Class 对象之newInstance的坑 ：*[*http://www.cnblogs.com/ixenos/p/5682088.html*](http://www.cnblogs.com/ixenos/p/5682088.html)

**客户端：**

[复制代码](javascript:void(0);)

1 public class StrategyCaller {

2

3 public static void main(String[] args) {

4 Context<StrategyA> c = new Context<StrategyA>(StrategyA.class);

5 c.execute();

6

7 Context<StrategyB> c2 = new Context<StrategyB>(StrategyB.class);

8 c2.execute();

9 }

10 }