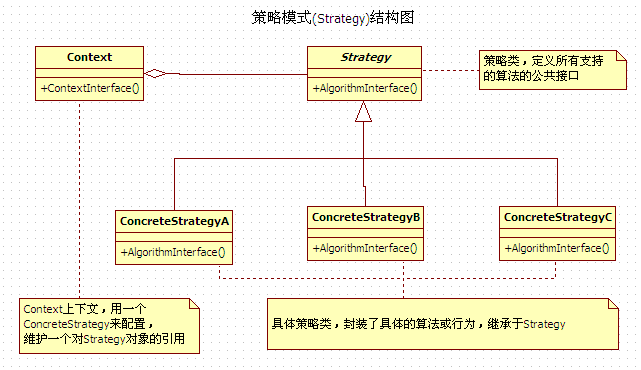
 策略模式(Strategy)：它定义了**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**家族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化，不会影响到使用算法的客户。

【1】基本概念

          策略模式是一种定义一系列算法的方法，从概念上来看，所有这些算法完成的都是相同的工作，只是实现不同，它可以以相同的方式调用所有的算法，减少了各种算法类与使用算法类之间的耦合。

【2】简单分析

          我们先来看一下该设计模式的UML结构图



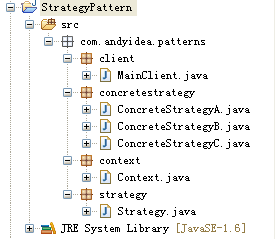
上图是Strategy 模式的结构图,让我们可以进行更方便的描述:

1. Strategy: 定义所有支持的算法的公共接口抽象类.
2. ConcreteStrategy: 封装了具体的算法或行为，继承于Strategy
3. Context: 用一个ConcreteStrategy来配置，维护一个对Strategy对象的引用。

      策略模式的Strategy类层次为Context定义了一系列的可供重用的算法或行为。继承有助于析取出这些算法中的公共功能。在实践中，我们发现可以用它来封装几乎任何类型的规则，只要在分析过程中听到需要在不同时间应用不同的业务规则，就可以考虑使用策略模式处理这种变化的可能性。

【3】如何用java语言来实现该模式

下面以一个简单的例子来展示该模式，先看下代码结构图：



3.1 首先定义一个Strategy抽象类，定义所有算法的公共接口

**[html]** [view plain](http://blog.csdn.net/cjjky/article/details/7322688) [copy](http://blog.csdn.net/cjjky/article/details/7322688) [print?](http://blog.csdn.net/cjjky/article/details/7322688)

1. **<span** style="font-family: SimSun; font-size: 13px;"**>**package com.andyidea.patterns.strategy;
3. /\*\*
4. \* 抽象算法类
5. \* @author Andy.Chen
6. \*
7. \*/
8. public abstract class Strategy {
10. //算法方法
11. public abstract void AlgorithmInterface();
13. }**</span>**

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

package com.andyidea.patterns.strategy;

/\*\*

\* 抽象算法类

\* @author Andy.Chen

\*

\*/

public abstract class Strategy {

//算法方法

public abstract void AlgorithmInterface();

}

3.2 定义具体ConcreteStrategy，分别继承Strategy

ConcreteStrategyA代码：

**[html]** [view plain](http://blog.csdn.net/cjjky/article/details/7322688) [copy](http://blog.csdn.net/cjjky/article/details/7322688) [print?](http://blog.csdn.net/cjjky/article/details/7322688)

1. package com.andyidea.patterns.concretestrategy;
3. import com.andyidea.patterns.strategy.Strategy;
5. /\*\*
6. \* 具体算法A
7. \* @author Andy.Chen
8. \*
9. \*/
10. public class ConcreteStrategyA extends Strategy{
12. @Override
13. public void AlgorithmInterface() {
14. System.out.println("算法A实现!");
15. }
17. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

package com.andyidea.patterns.concretestrategy;

import com.andyidea.patterns.strategy.Strategy;

/\*\*

\* 具体算法A

\* @author Andy.Chen

\*

\*/

public class ConcreteStrategyA extends Strategy{

@Override

public void AlgorithmInterface() {

System.out.println("算法A实现!");

}

}

ConcreteStrategyB代码：

**[html]** [view plain](http://blog.csdn.net/cjjky/article/details/7322688) [copy](http://blog.csdn.net/cjjky/article/details/7322688) [print?](http://blog.csdn.net/cjjky/article/details/7322688)

1. package com.andyidea.patterns.concretestrategy;
3. import com.andyidea.patterns.strategy.Strategy;
5. /\*\*
6. \* 具体算法B
7. \* @author Andy.Chen
8. \*
9. \*/
10. public class ConcreteStrategyB extends Strategy{
12. @Override
13. public void AlgorithmInterface() {
14. System.out.println("算法B实现!");
15. }
17. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

package com.andyidea.patterns.concretestrategy;

import com.andyidea.patterns.strategy.Strategy;

/\*\*

\* 具体算法B

\* @author Andy.Chen

\*

\*/

public class ConcreteStrategyB extends Strategy{

@Override

public void AlgorithmInterface() {

System.out.println("算法B实现!");

}

}

ConcreteStrategyC代码：

**[html]** [view plain](http://blog.csdn.net/cjjky/article/details/7322688) [copy](http://blog.csdn.net/cjjky/article/details/7322688) [print?](http://blog.csdn.net/cjjky/article/details/7322688)

1. package com.andyidea.patterns.concretestrategy;
3. import com.andyidea.patterns.strategy.Strategy;
5. /\*\*
6. \* 具体算法C
7. \* @author Andy.Chen
8. \*
9. \*/
10. public class ConcreteStrategyC extends Strategy{
12. @Override
13. public void AlgorithmInterface() {
14. System.out.println("算法C实现!");
15. }
17. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

package com.andyidea.patterns.concretestrategy;

import com.andyidea.patterns.strategy.Strategy;

/\*\*

\* 具体算法C

\* @author Andy.Chen

\*

\*/

public class ConcreteStrategyC extends Strategy{

@Override

public void AlgorithmInterface() {

System.out.println("算法C实现!");

}

}

3.3定义Context类，维护队Strategy对象的引用。

**[html]** [view plain](http://blog.csdn.net/cjjky/article/details/7322688) [copy](http://blog.csdn.net/cjjky/article/details/7322688) [print?](http://blog.csdn.net/cjjky/article/details/7322688)

1. package com.andyidea.patterns.context;
3. import com.andyidea.patterns.strategy.Strategy;
5. /\*\*
6. \* 上下文
7. \* @author Andy.Chen
8. \*
9. \*/
10. public class Context {
12. private Strategy mStrategy;
13. public Context(Strategy strategy){
14. this.mStrategy = strategy;
15. }
17. /\*\*
18. \* 上下文接口
19. \*/
20. public void ContextInterface(){
21. mStrategy.AlgorithmInterface();
22. }
24. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

package com.andyidea.patterns.context;

import com.andyidea.patterns.strategy.Strategy;

/\*\*

\* 上下文

\* @author Andy.Chen

\*

\*/

public class Context {

private Strategy mStrategy;

public Context(Strategy strategy){

this.mStrategy = strategy;

}

/\*\*

\* 上下文接口

\*/

public void ContextInterface(){

mStrategy.AlgorithmInterface();

}

}

3.4 客户端测试类 MainClient.java 源码

**[html]** [view plain](http://blog.csdn.net/cjjky/article/details/7322688) [copy](http://blog.csdn.net/cjjky/article/details/7322688) [print?](http://blog.csdn.net/cjjky/article/details/7322688)

1. package com.andyidea.patterns.client;
3. import com.andyidea.patterns.concretestrategy.ConcreteStrategyA;
4. import com.andyidea.patterns.concretestrategy.ConcreteStrategyB;
5. import com.andyidea.patterns.concretestrategy.ConcreteStrategyC;
6. import com.andyidea.patterns.context.Context;
8. /\*\*
9. \* 客户端测试类
10. \* @author Andy.Chen
11. \*
12. \*/
13. public class MainClient {
15. private static Context context;
17. public static void main(String[] args) {
19. System.out.println("Welcome to Andy.Chen Blog!" +"\n"
20. +"Strategy Patterns." +"\n"
21. +"----------------------------");
23. context = new Context(new ConcreteStrategyA());
24. context.ContextInterface();
26. context = new Context(new ConcreteStrategyB());
27. context.ContextInterface();
29. context = new Context(new ConcreteStrategyC());
30. context.ContextInterface();
31. }
33. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

package com.andyidea.patterns.client;

import com.andyidea.patterns.concretestrategy.ConcreteStrategyA;

import com.andyidea.patterns.concretestrategy.ConcreteStrategyB;

import com.andyidea.patterns.concretestrategy.ConcreteStrategyC;

import com.andyidea.patterns.context.Context;

/\*\*

\* 客户端测试类

\* @author Andy.Chen

\*

\*/

public class MainClient {

private static Context context;

public static void main(String[] args) {

System.out.println("Welcome to Andy.Chen Blog!" +"\n"

+"Strategy Patterns." +"\n"

+"----------------------------");

context = new Context(new ConcreteStrategyA());

context.ContextInterface();

context = new Context(new ConcreteStrategyB());

context.ContextInterface();

context = new Context(new ConcreteStrategyC());

context.ContextInterface();

}

}

【4】程序运行结果如下：

**[html]** [view plain](http://blog.csdn.net/cjjky/article/details/7322688) [copy](http://blog.csdn.net/cjjky/article/details/7322688) [print?](http://blog.csdn.net/cjjky/article/details/7322688)

1. Welcome to Andy.Chen Blog!
2. Strategy Patterns.
3. ----------------------------
4. 算法A实现!
5. 算法B实现!
6. 算法C实现!

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

Welcome to Andy.Chen Blog!

Strategy Patterns.

----------------------------

算法A实现!

算法B实现!

算法C实现!

从上面可以看到，策略模式的优点是简化了单元测试，因为每个算法都有自己的类，可以通过自己的接口单独测试。