**一、 策略（Strategy）模式**

策略模式的用意是针对一组算法，将每一个算法封装到具有共同接口的独立的类中，从而使得它们可以相互替换。策略模式使得算法可以在不影响到客户端的情况下发生变化。

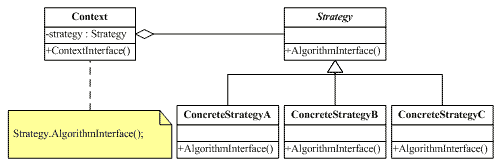
假设现在要设计一个贩卖各类书籍的电子商务网站的购物车（Shopping Cat）系统。一个最简单的情况就是把所有货品的单价乘上数量，但是实际情况肯定比这要复杂。比如，本网站可能对所有的教材类图书实行每本一元的折扣；对连环画类图书提供每本7％的促销折扣，而对非教材类的计算机图书有3％的折扣；对其余的图书没有折扣。由于有这样复杂的折扣算法，使得价格计算问题需要系统地解决。

使用策略模式可以把行为和环境分割开来。环境类负责维持和查询行为类，各种算法则在具体策略类（ConcreteStrategy）中提供。由于算法和环境独立开来，算法的增减、修改都不会影响环境和客户端。当出现新的促销折扣或现有的折扣政策出现变化时，只需要实现新的策略类，并在客户端登记即可。策略模式相当于"可插入式（Pluggable）的算法"。

**二、 策略模式的结构**

策略模式是对算法的包装，是把使用算法的责任和算法本身分割开，委派给不同的对象管理。策略模式通常把一个系列的算法包装到一系列的策略类里面，作为一个抽象策略类的子类。用一句话来说，就是："准备一组算法，并将每一个算法封装起来，使得它们可以互换。"

策略又称做政策（Policy）模式【GOF95】。下面是一个示意性的策略模式结构图：



这个模式涉及到三个角色：

* 环境（Context）角色：持有一个Strategy类的引用。
* 抽象策略（Strategy）角色：这是一个抽象角色，通常由一个接口或抽象类实现。此角色给出所有的具体策略类所需的接口。
* 具体策略（ConcreteStrategy）角色：包装了相关的算法或行为。

**三、 示意性源代码**

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// Strategy pattern -- Structural example    
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "Strategy"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifabstract class Strategy  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void AlgorithmInterface();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteStrategyA"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass ConcreteStrategyA : Strategy  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  override public void AlgorithmInterface()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("Called ConcreteStrategyA.AlgorithmInterface()");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteStrategyB"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass ConcreteStrategyB : Strategy  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  override public void AlgorithmInterface()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("Called ConcreteStrategyB.AlgorithmInterface()");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteStrategyC"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass ConcreteStrategyC : Strategy  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  override public void AlgorithmInterface()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("Called ConcreteStrategyC.AlgorithmInterface()");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "Context"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass Context  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  Strategy strategy;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Constructors  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Context( Strategy strategy )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.strategy = strategy;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void ContextInterface()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    strategy.AlgorithmInterface();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// Client test  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Client  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public static void Main( string[] args )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Three contexts following different strategies  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Context c = new Context( new ConcreteStrategyA() );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    c.ContextInterface();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Context d = new Context( new ConcreteStrategyB() );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    d.ContextInterface();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Context e = new Context( new ConcreteStrategyC() );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    e.ContextInterface();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

**四、 何时使用何种具体策略角色**

在学习策略模式时，学员常问的一个问题是：为什么不能从策略模式中看出哪一个具体策略适用于哪一种情况呢？

答案非常简单，策略模式并不负责做这个决定。换言之，应当由客户端自己决定在什么情况下使用什么具体策略角色。策略模式仅仅封装算法，提供新算法插入到已有系统中，以及老算法从系统中"退休"的方便，策略模式并不决定在何时使用何种算法。

**五、 一个实际应用策略模式的例子**

下面的例子利用策略模式在排序对象中封装了不同的排序算法，这样以便允许客户端动态的替换排序策略（包括Quicksort、Shellsort和Mergesort）。

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// Strategy pattern -- Real World example    
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System.Collections;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "Strategy"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifabstract class SortStrategy  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void Sort( ArrayList list );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteStrategy"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass QuickSort : SortStrategy  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Sort(ArrayList list )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    list.Sort(); // Default is Quicksort  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("QuickSorted list ");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteStrategy"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass ShellSort : SortStrategy  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Sort(ArrayList list )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    //list.ShellSort();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("ShellSorted list ");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteStrategy"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass MergeSort : SortStrategy  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Sort( ArrayList list )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    //list.MergeSort();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("MergeSorted list ");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "Context"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass SortedList  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private ArrayList list = new ArrayList();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private SortStrategy sortstrategy;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Constructors  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void SetSortStrategy( SortStrategy sortstrategy )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.sortstrategy = sortstrategy;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void Sort()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    sortstrategy.Sort( list );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void Add( string name )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    list.Add( name );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void Display()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    foreach( string name in list )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      Console.WriteLine( " " + name );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// StrategyApp test  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class StrategyApp  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public static void Main( string[] args )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Two contexts following different strategies  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    SortedList studentRecords = new SortedList( );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    studentRecords.Add( "Samual" );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    studentRecords.Add( "Jimmy" );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    studentRecords.Add( "Sandra" );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    studentRecords.Add( "Anna" );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    studentRecords.Add( "Vivek" );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    studentRecords.SetSortStrategy( new QuickSort() );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    studentRecords.Sort();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    studentRecords.Display();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

**六、 在什么情况下应当使用策略模式**

在下面的情况下应当考虑使用策略模式：

1. 如果在一个系统里面有许多类，它们之间的区别仅在于它们的行为，那么使用策略模式可以动态地让一个对象在许多行为中选择一种行为。

2. 一个系统需要动态地在几种算法中选择一种。那么这些算法可以包装到一个个的具体算法类里面，而这些具体算法类都是一个抽象算法类的子类。换言之，这些具体算法类均有统一的接口，由于多态性原则，客户端可以选择使用任何一个具体算法类，并只持有一个数据类型是抽象算法类的对象。

3. 一个系统的算法使用的数据不可以让客户端知道。策略模式可以避免让客户端涉及到不必要接触到的复杂的和只与算法有关的数据。

4. 如果一个对象有很多的行为，如果不用恰当的模式，这些行为就只好使用多重的条件选择语句来实现。此时，使用策略模式，把这些行为转移到相应的具体策略类里面，就可以避免使用难以维护的多重条件选择语句，并体现面向对象设计的概念。

**七、 策略模式的优点和缺点**

策略模式有很多优点和缺点。它的优点有：

1. 策略模式提供了管理相关的算法族的办法。策略类的等级结构定义了一个算法或行为族。恰当使用继承可以把公共的代码移到父类里面，从而避免重复的代码。

2. 策略模式提供了可以替换继承关系的办法。继承可以处理多种算法或行为。如果不是用策略模式，那么使用算法或行为的环境类就可能会有一些子类，每一个子类提供一个不同的算法或行为。但是，这样一来算法或行为的使用者就和算法或行为本身混在一起。决定使用哪一种算法或采取哪一种行为的逻辑就和算法或行为的逻辑混合在一起，从而不可能再独立演化。继承使得动态改变算法或行为变得不可能。

3. 使用策略模式可以避免使用多重条件转移语句。多重转移语句不易维护，它把采取哪一种算法或采取哪一种行为的逻辑与算法或行为的逻辑混合在一起，统统列在一个多重转移语句里面，比使用继承的办法还要原始和落后。

策略模式的缺点有：

1. 客户端必须知道所有的策略类，并自行决定使用哪一个策略类。这就意味着客户端必须理解这些算法的区别，以便适时选择恰当的算法类。换言之，策略模式只适用于客户端知道所有的算法或行为的情况。

2. 策略模式造成很多的策略类。有时候可以通过把依赖于环境的状态保存到客户端里面，而将策略类设计成可共享的，这样策略类实例可以被不同客户端使用。换言之，可以使用享元模式来减少对象的数量。

**八、 其它**

策略模式与很多其它的模式都有着广泛的联系。Strategy很容易和Bridge模式相混淆。虽然它们结构很相似，但它们却是为解决不同的问题而设计的。Strategy模式注重于算法的封装，而Bridge模式注重于分离抽象和实现，为一个抽象体系提供不同的实现。Bridge模式与Strategy模式都很好的体现了"Favor composite over inheritance"的观点。

推荐大家读一读《IoC 容器和Dependency Injection 模式》，作者Martin Fowler。网上可以找到中文版的PDF文件。为策略模式的实施提供了一个非常好的方案。