作者：伍迷 http://cj723.cnblogs.com
http://www.cnblogs.com/cj723/archive/2007/04/02/697431.html
整理：秦韶华

作者：伍迷戏说面向对象程序设计

C＃版

目 录

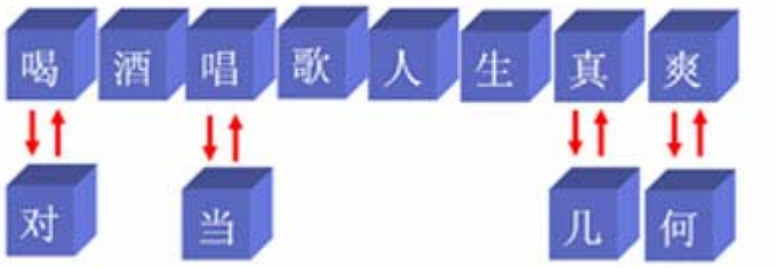
四大发明之活字印刷——面向对象思想的胜利

话说三国时期，曹操带领百万大军攻打东吴，大军在长江赤壁驻扎，军船连成一片，眼看就要灭掉东吴，统一天下，曹操大悦，于是大宴众文武，在酒席间，曹操诗性大发，不觉吟道：“喝酒唱歌，人生真爽。……”。众文武齐呼：“丞相好诗！”于是一臣子速命印刷工匠刻版印刷，以便流传天下。

样张出来给曹操一看，曹操感觉不妥，说道：“喝与唱，此话过俗，应改为‘对酒当歌’较好！”，于是此臣就命工匠重新来过。工匠眼看连夜刻版之工，彻底白费，心中叫苦不喋。只得照办。

样张再次出来请曹操过目，曹操细细一品，觉得还是不好，说：“人生真爽太过直接，应改问语才够意境，因此应改为‘对酒当歌，人生几何？……’！”当臣转告工匠之时，工匠晕倒……！

可惜三国时期活字印刷还未发明，所以类似事情应该时有发生，如果是有了活字印刷。则只需更改四个字就可，其余工作都未白做。实在妙哉。



第一，要改，只需更改要改之字，此为**可维护**；第二，这些字并非用完这次就无用，完全可以在后来的印刷中重复使用，此乃**可复用**；第三，此诗若要加字，只需另刻字加入即可，这是**可扩展**；第四，字的排列其实有可能是竖有可能是横排，此时只需将活字移动就可做到满足排列需求，此是**灵活性好**。

而在活字印刷术之前，上面的四种特性都无法满足，要修改，必须重刻，要加字，必须重刻，要重新排列，必须重刻，印完这本书后，此版已无任何可再利用价值。

小时候，我一直奇怪，为何火药、指南针、造纸术都是从无到有，从未知到发现的伟大发明，而活字印刷仅仅是从刻版印刷到活字印刷的一次技术上的进步，为何不是评印刷术为四大发明之一呢？

做了软件开发几年后，经历了太多的客户（曹操）改变需求，更改最初想法的事件，才逐渐明白当中的道理。其实客观的说，客户的要求也并不过分（改几个字而已），但面对已完成的程序代码，却是需要几乎重头来过的尴尬，这实在是痛苦不堪。说白了，原因就是因为我们*原先所写的程序，不容易维护，灵活性差，不容易扩展，更谈不上复用，因此面对需求变化，加班加点，对程序动大手术的那种无耐也就非常正常的事了*。

之后当我学习了面向对象分析设计编程思想，开始考虑**通过封装、继承、多态把程序的耦合度降低**（传统印刷术的问题就在于所有的字都刻在同一版面上造成耦合度太高所制），开始**用设计模式使得程序更加的灵活，容易修改，并且易于复用**。体会到面向对象带来的好处，那种感觉应该就如同是一中国酒鬼第一次喝到了茅台，西洋酒鬼第一次喝到了XO一样，怎个爽字可形容呀。

再次回顾中国古代的四大发明，另三种应该都是科技的进步，伟大的创造或发现。而唯有活字印刷，实在是思想的成功，面向对象的胜利。不知您是否也有所感呢？

第一章 面试受挫——代码无错就是好？

小菜今年计算机专业大四了，学了不少软件开发方面的东西，也学着编了些小程序，踌躇满志，一心要找一个好单位。当投递了无数份简历后，终于收到了一个单位的面试通知，小菜欣喜若狂。

到了人家单位，前台小姐给了他一份题目，上面写着，“请用C++、Java、C#或VB.NET任意一种面向对象语言实现一个计算器控制台程序，要求输入两个数和运算符号，得到结果。”

小菜一看，这个还不简单，三下五除二，10分钟不到，小菜写完了，感觉也没错误。交卷后，单位说一周内等通知吧。于是小菜只得耐心等待。可是半个月过去了，什么消息也没有，小菜很纳闷，我的代码实现了呀，为什么不给我机会呢。

小菜找到工作三年的师哥大鸟，请教原因，大鸟问了题目和了解了小菜代码的细节以后，哈哈大笑，说道：“小菜呀小菜，你上当了，人家单位出题的意思，你完全都没明白，当然不会再联系你了”。

小菜说：“我的代码有错吗？单位题目不就是要我实现一个计算器的代码吗，我这样写有什么问题。”

```
class Program
{
    static void Main(string[] args)
    {
        Console.Write("请输入数字A：");
        string A = Console.ReadLine();
        Console.Write("请选择运算符号(+、-、*、/)：");
        string B = Console.ReadLine();
        Console.Write("请输入数字B：");
        string C = Console.ReadLine();
        string D = "";

        if (B == "+")
            D = Convert.ToString(Convert.ToDouble(A) + Convert.ToDouble(C));
        if (B == "-")
            D = Convert.ToString(Convert.ToDouble(A) - Convert.ToDouble(C));
        if (B == "*")
            D = Convert.ToString(Convert.ToDouble(A) * Convert.ToDouble(C));
        if (O == "/")
            D = Convert.ToString(Convert.ToDouble(A) / Convert.ToDouble(C));

        Console.WriteLine("结果是：" + D);
    }
}
```

小菜的代码有什么问题呢？

第二章 代码规范、重构

大鸟说：“且先不说出题人的意思，单就你现在的代码，就有很多不足的地方需要改进。比如变量命名，你的命名就是ABCD，变量不带有任何具体含义，这是非常不规范的；判断分支，你这样的写法，意味着每个条件都要做判断，等于计算机做了三次无用功；数据输入有效性判断等，如果用户输入的是字符符号而不是数字怎么办？如果除数时，客户输入了0怎么办？这些都是可以改进的地方。”

“哦，说得没错，这个我以前听老师说过，可是从来没有在意过，我马上改，改完再给你看看。”

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.Write("请输入数字A：");
            string strNumberA = Console.ReadLine();
            Console.Write("请选择运算符号(+、-、*、/)：");
            string strOperate = Console.ReadLine();
            Console.Write("请输入数字B：");
            string strNumberB = Console.ReadLine();
            string strResult = "";

switch (strOperate)
{
    case "+":
        strResult = Convert.ToString
            (Convert.ToDouble(strNumberA) + Convert.ToDouble(strNumberB));
        break;
    case "-":
        strResult = Convert.ToString
            (Convert.ToDouble(strNumberA) - Convert.ToDouble(strNumberB));
        break;
    case "*":
        strResult = Convert.ToString
            (Convert.ToDouble(strNumberA) * Convert.ToDouble(strNumberB));
        break;
    case "/":
        if (strNumberB != "0")
            strResult = Convert.ToString
                (Convert.ToDouble(strNumberA) / Convert.ToDouble(strNumberB));
        else
            strResult = "除数不能为0";
        break;
}
```

```
            Console.WriteLine("结果是：" + strResult);

            Console.ReadLine();


        }
        catch (Exception ex)
        {
            Console.WriteLine("您的输入有错：" + ex.Message);
        }
    }
}
```

大鸟：“吼吼，不错，不错，改得很快吗？至在目前代码来说，实现计算器是没有问题了，但这样写出的代码是否合出题人的意思呢？”

小菜：“你的意思是面向对象？”

大鸟：“哈，小菜非小菜也！”

第三章 复制VS 复用

小菜：“我明白了，他说用任意一种面向对象语言实现，那意思就是要用面向对象的编程方法去实现，对吗？OK，这个我学过，只不过当时我没想到而已。”

大鸟：“所有编程初学者都会有这样的问题，就是碰到问题就直觉的用计算机能够理解的逻辑来描述和表达待解决的问题及具体的求解过程。这其实是用计算机的方式去思考，比如计算器这个程序，先要求输入两个数和运算符号，然后根据运算符号判断选择如何运算，得到结果，这本身没有错，但这样的思维却使得我们的程序只为满足实现当前的需求，程序不容易维护，不容易扩展，更不容易复用。从而达不到高质量代码的要求。”

小菜：“鸟哥呀，我有点糊涂了，如何才能容易维护，容易扩展，又容易复用呢，能不能具体点？”

大鸟：“比如说，我现在要求你再写一个 windows 的计算器，你现在的代码能不能复用呢？”

小菜：“那还不简单，把代码复制过去不就行了吗？改动又不大，不算麻烦。”

大鸟：“小菜看来还是小菜呀，有人说初级程序员的工作就是 Ctrl+C 和 Ctrl+V，这其实是非常不好的编码习惯，因为当你的代码中重复的代码多到一定程度，维护的时候，可能就是一场灾难。越大的系统，这种方式带来的问题越严重，编程有一原则，就是用尽可能的办法去避免重复。想想看，你写的这段代码，有哪些是和控制台无关的，而只是和计算器有关的？”

第四章 业务的封装

小菜：“你的意思是分一个类出来？ 哦，对的，让计算和显示分开。”

大鸟：“准确的说，就是让业务逻辑与界面逻辑分开，让它们之间的耦合度下降。只有分离开，才容易达到容易维护或扩展。”

小菜：“让我来试试看。”

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.Write("请输入数字A：");
            string strNumberA = Console.ReadLine();
            Console.Write("请选择运算符号(+、-、*、/)：");
            string strOperate = Console.ReadLine();
            Console.Write("请输入数字B：");
            string strNumberB = Console.ReadLine();
            string strResult = "";

            strResult = Convert.ToString(
                Operation.GetResult(Convert.ToDouble(strNumberA),
                Convert.ToDouble(strNumberB), strOperate));

            Console.WriteLine("结果是：" + strResult);

            Console.ReadLine();


        }
        catch (Exception ex)
        {
            Console.WriteLine("您的输入有错：" + ex.Message);
        }
    }
}

public class Operation
{
    public static double GetResult(double numberA, double numberB, string operate)
    {
        double result = 0d;
```

```
        switch (operate)
        {
            case "+":
                result = numberA + numberB;
                break;
            case "-":
                result = numberA - numberB;
                break;
            case "*":
                result = numberA * numberB;
                break;
            case "/":
                result = numberA / numberB;
                break;
        }
        return result;
    }
}
```

小菜：“鸟哥，我写好了，你看看！”

大鸟：“哈，孺鸟可教也，:)，写得不错，这样就完全把业务和界面分离了。”

小菜心中暗骂：“你才是鸟呢。” 口中说道：“如果你现在要我写一个Windows应用程序的计算器，我就可以复用这个运算类（Operation）了。”

大鸟：“不单是Windows程序，Web版程序需要运算可以用它，PDA，手机等需要移动系统的软件需要运算也可以用它呀。”

小菜：“哈，面向对象不过如此。下会写类似代码不怕了。”

大鸟：“别急，仅此而已，实在谈不上完全面向对象，你只用了面向对象三大特性的一个，还两个没用呢？”

小菜：“面向对象三大特性不就是封装、继承和多态吗，这里我用到的应该是封装。这还不够吗？……我实在看不出，这么小的程序如何用到继承。至于多态，其它我一直也不太了解它到底有什么好处，如何使用它。”

大鸟：“慢慢来，有的东西好学了，你好好想想吧，我要去“魔兽”了，改时聊。”

第五章 体会简单工厂模式的美妙

次日，小菜再来找大鸟，问道：“你昨天说计算器这样的小程序还可以用到面向对象三大特性？继承和多态怎么可能用得上，我实在不可理解。”

大鸟：“小菜很有钻研精神吗？好，今天我让你功力加深一级。你先要考虑一下，你昨天写的这个代码，能否做到很灵活的可修改和扩展呢？”

小菜：“我已经把业务和界面分离了呀，这不是很灵活了吗？”

大鸟：“那我问你，现在如果我希望增加一个开根（sqrt）运算，你如何改？”

小菜：“那只需要改Operation类就行了，在switch中加一个分支就行了。”

大鸟：“问题是你要加一个平方根运算，却需要把加减乘除的运算都得来参与编译，如果你一不小心，把加法运算改成了减法，这不是大大的糟糕。打个比方，如果现在公司要求你为公司的薪资管理系统做维护，原来只有技术人员（月薪），市场销售人员（底薪+提成），经理（年薪+股份）三种运算算法，现在要增加兼职工作人员的（时薪）算法，但按照你昨天的程序写法，公司就必须要把包含有的原三种算法的运算类给你，让你修改，你如果心中小算盘一打，‘TMD，公司给我的工资这么低，我真是郁闷，这会有机会了’，于是你除了增加了兼职算法以外，在技术人员（月薪）算法中写了一句

```
if（员工是小菜）
{
    salary = salary * 1.1;
}
```

那就意味着，你的月薪每月都会增加10%（小心被抓去坐牢），本来是让你加一个功能，却使得原有的运行良好的功能代码产生了变化，这个风险太大了。你明白了吗？”

小菜：“哦，你的意思是，我应该把加减乘除等运算分离，修改其中一个不影响另外的几个，增加运算算法也不影响其它代码，是这样吗？”

大鸟：“自己想去吧，如何用继承和多态，你应该有感觉了。”

小菜：“OK，我马上去写。”

```
/// <summary>
/// 运算类
/// </summary>
class Operation
{
    private double _numberA = 0;
    private double _numberB = 0;

    /// <summary>
    /// 数字A
    /// </summary>
    public double NumberA
    {
        get { return _numberA; }
        set { _numberA = value; }
    }

    /// <summary>
```

```
        /// 数字B
        /// </summary>
        public double NumberB
        {
            get { return _numberB; }
            set { _numberB = value; }
        }

        /// <summary>
        /// 得到运算结果
        /// </summary>
        /// <returns></returns>
        public virtual double GetResult()
        {
            double result = 0;
            return result;
        }
    }

    /// <summary>
    /// 加法类
    /// </summary>
    class OperationAdd : Operation
    {
        public override double GetResult()
        {
            double result = 0;
            result = NumberA + NumberB;
            return result;
        }
    }

    /// <summary>
    /// 减法类
    /// </summary>
    class OperationSub : Operation
    {
        public override double GetResult()
        {
            double result = 0;
            result = NumberA - NumberB;
            return result;
        }
    }
```

```
/// <summary>
/// 乘法类
/// </summary>
class OperationMul : Operation
{
    public override double GetResult()
    {
        double result = 0;
        result = NumberA * NumberB;
        return result;
    }
}

/// <summary>
/// 除法类
/// </summary>
class OperationDiv : Operation
{
    public override double GetResult()
    {
        double result = 0;
        if (NumberB == 0)
            throw new Exception("除数不能为0。");
        result = NumberA / NumberB;
        return result;
    }
}
```

小菜：“大鸟哥，我按照你说的方法写出来了一部分，首先是一个运算类，它有两个Number属性，主要用于计算器的前后数，然后有一个虚方法GetResult()，用于得到结果，然后我把加减乘除都写成了运算类的子类，继承它后，重写了GetResult()方法，这样如果要修改任何一个算法，都不需要提供其它算法的代码了。但问题来了，我如何让计算器知道我是希望用哪一个算法呢？”

大鸟：“写得很不错吗，大大超出我的想象了，你现在的问题其实就是如何去实例化对象的问题，哈，今天心情不错，再教你一招‘简单工厂模式’，也就是说，到底要实例化谁，将来会不会增加实例化的对象（比如增加开根运算），这是很容易变化的地方，应该考虑用一个单独的类来做这个创造实例的过程，这就是工厂，来，我们看看这个类如何写。”

```
/// <summary>
/// 运算类工厂
/// </summary>
class OperationFactory
{
    public static Operation createOperate(string operate)
    {
        Operation oper = null;
```

```
            switch (operate)
            {
                case "+":
                    {
                        oper = new OperationAdd();
                        break;
                    }
                case "-":
                    {
                        oper = new OperationSub();
                        break;
                    }
                case "*":
                    {
                        oper = new OperationMul();
                        break;
                    }
                case "/":
                    {
                        oper = new OperationDiv();
                        break;
                    }
            }

            return oper;
        }
    }
```

大鸟：“哈，看到吧，这样子，你只需要输入运算符号，工厂就实例化出合适的对象，通过多态，返回父类的方式实现了计算器的结果。”

```
    Operation oper;
    oper = OperationFactory.createOperate("+");
    oper.NumberA = 1;
    oper.NumberB = 2;
    double result = oper.GetResult();
```

大鸟：“哈，界面的实现就是这样的代码，不管你是控制台程序，Windows程序，Web程序，PDA或手机程序，都可以用这段代码来实现计算器的功能，当有一天我们需要更改加法运算，我们只需要改哪里？”

小菜：“改OperationAdd 就可以了。”

大鸟：“那么我们需要增加各种复杂运算，比如平方根，立方根，自然对数，正弦余弦等，如何做？”

小菜：“只要增加相应的运算子类就可以了呀。”

大鸟：“嗯？够了吗？”

小菜：“对了，还需要去修改运算类工厂，在switch中增加分支。”

大鸟：“哈，那才对，那如果要修改界面呢？”

小菜：“那就去改界面呀，关运算什么事呀。”

小菜：“ 回想那天我面试题写的代码，我终于明白我为什么写得不成功了，原来一个小小的计算器也可以写出这么

精彩的代码，谢谢大鸟。”

(下为当时面试题时小菜所写代码，见《小菜编程成长记（一）》)

```
Program
{
    static void Main(string[] args)
    {
        Console.Write("请输入数字A：");
        string A = Console.ReadLine();
        Console.Write("请选择运算符号(+、-、*、/)：");
        string B = Console.ReadLine();
        Console.Write("请输入数字B：");
        string C = Console.ReadLine();
        string D = "";

        if (B == "+")
            D = Convert.ToString(Convert.ToDouble(A) + Convert.ToDouble(C));
        if (B == "-")
            D = Convert.ToString(Convert.ToDouble(A) - Convert.ToDouble(C));
        if (B == "*")
            D = Convert.ToString(Convert.ToDouble(A) * Convert.ToDouble(C));
        if (O == "/")
            D = Convert.ToString(Convert.ToDouble(A) / Convert.ToDouble(C));

        Console.WriteLine("结果是：" + D);
    }
}
```

大鸟：“吼吼，记住哦，**编程是一门技术，更加是一门艺术**，不能只满足于写完代码运行结果正确就完事，时常考虑如何让代码更加简炼，更加容易维护，容易扩展和复用，只有这样才可以是真的提高。写出优雅的代码真的是一种很爽的事情。不过学无止境，其实这才是理解面向对象的开始呢。给你出个作业，做一个商场收银软件，营业员根据客户购买商品单价和数量，向客户收费。”

小菜：“就这个？没问题呀。”

第六章 关于Flex的争论？

声明：本文略有广告之嫌，不过写此文的初衷却是因为对上篇博客引发争议的感想，所以看此文前不妨先看看上文的评论，感谢您的阅读，欢迎拍砖。

几天后，小菜跑到大鸟处，说道：“大鸟哥，今天看到一个网站，叫妙句网，是用来收藏好句子用的，网站本身没什么太多希奇，但它用的技术却是最近非常火的Flex技术，作者说他也就学了不到两个月就做出了这个网站，你说会不会是真的？如果真的是这样，我也想学学，毕竟做出来的效果比一般的网站真的要酷很多哦。”

大鸟：“哦？让我来看看。”大鸟打开了浏览器，键入了小菜说的网站。

“嗯？怎么是 flash 的？”大鸟说。

小菜点头道：“Flex 其实就是开发出可以在 Flash Player 插件上运行的编程工具！你看，它整个网站就是一个 Flash，开始下载是慢了点，但下载好了，大部分操作都不用再访问网络了，局部的更新数据，也只是局部刷新而已。你点这个登录，哈，登录就从上框跳出来，还会抖动两下，再看，点这个‘Show’出来框后，点‘播放’，那个句子翻页的 3D 效果是不是很酷？还有……”

“Stop！ 停！”大鸟突然说道，“你是来给我做广告来了？这种纯粹为炫点效果而做的网站我见得多了，有什么希奇的。我虽然没做过 Flash，但开发这些年，我还不知道？就象我没得过禽流感，但感冒啥滋味，我不会比鸡知道的少了。”

“可是听说这技术很容易学，用不了多久就可以上手，做出漂亮的东西。你之前不是还整天跟我说 AJAX 太他妈麻烦，JavaScript 调试极度困难，还要考虑跨浏览器等诸多问题，写 Ajax 网站吃力不讨好吗？”

“哈哈，小菜呀小菜，外行了吧。写这网站的人说是只学了 Flex 一个多月做出这东西，我也相信他说的是真的，但这里面有多少细节技术是你看不到的？我说几个细节给你听听，比如注册，Email 如何验证？这用正则表达式判断最好，你得懂才行，哪怕上网查，至少你要知道如何用吧。再比如这网站上的分页，每页显示 10 条信息，如果把一千多条句子全下载不就傻了吗？可若每次换页只下载 10 条，这就得懂得如何写这样的 Sql 语句，当然最好是存储过程。至于整个网站包括了从注册、登录、增删改句子到句子列表，正文显示，搜索查询，这就是一整套的网站基本架构，这些都是要技术积累的。”

“啊，这倒是真的，正则表达式学过，但还是不会用，分页存储过程干脆就写不来，要独立写个网站，看来也的确是有难度的。”

“不过，这网站其实做了也没太大的用处。”大鸟说道，“第一，Flash 占的系统资源太大，你看，我才点击 tag 查询，我的 CPU 就 100%，我在放的 MP3，声音都开始打抖了，这真的很不爽。第二，Flash 是封装成 swf 文件下载的，也就是说搜索引擎是根本找不到里面的数据，里面的句子也就谈不上被搜索到。这样这些数据又如何能被大众用户查看到？”

“咦，我在家里好像感觉不到速度问题吗？哦，我明白了，因为我的笔记本是双核的，最多也就占用 50%，看来硬件的提升也是促进软件的发展呀。不过搜索引擎好像是很难办，有没有什么解决方案？”

大鸟说：“搜索引擎对整站 SWF 的索引的问题，目前没有什么完美的解决方法，或许，我们可以生成一个 sitemap，让搜索引擎的爬虫自己爬上来。”

大鸟继续说道：“所以我说这种网站，没什么太大用处，Flex 这种富客户端技术，不过是个噱头罢了。”

“唉，大鸟，我就不同意的你看法。”小菜有点激动，“现在有了 DVD，你还会去小摊买 SVCD 看吗？现在有了 Windows，你还会去整天对着黑屏用 Dos 敲打命令行吗？富客户端可以做到更加好的客户感受度，又不用安装和升级，这结合 C/S 和 B/S 的优点于一身的好技术。”

“小菜，html 是 Internet 的主流，要客户感受度，还是好好研究 JavaScript 的 AJAX 技术吧。”大鸟语重声长的说。

“降龙十八掌固然强悍勇猛，那是因为没有发明手枪；赤兔固然是良驹，那是因为关老儿没有宝马大奔。谁说 html 今天是主流，就意味着未来是主流？我觉得 AJAX 有很大的局限性，是一种过渡技术。未来一定是富客户端的天下。”

“小菜别激动，你说得也有道理。我举个例子你就会明白，Flex 或 Flash 以前是 Macromedia 现在上 Adobe 的主推的产品吧，那我问你，他们的官方网站是什么写的？”大鸟慢条斯理道。

“这个……好像还是 html”。小菜犹豫后答道。

“你有想过为什么？还有那些 Flex 的技术网站，比如博客呀论坛呀什么的，我敢肯定的说，一定不是用富客户端技术做的。”

“是不是因为不好搜索？”小菜疑问中。

“这是一方面吧，其实文字信息很大的网站，都不太适合这种技术的，因为它要的不是过渡效果，而是内容本身。如果内容不够好，网站再漂亮也是没有用的，而内容够好，却没有人能找得到这些内容，这内容也是价值不大的，这也正是为什么连 Adobe 公司自己在发布网站时也用的是传统的 html 模式。”

“是，我理解你的意思，就是说这种文字信息网站不适合，但有些网站，比如公司产品展示，动漫游戏等才更适合是吗？”

大鸟说：“当然，还不只是这些，Google 推出一种理念，就是要弱化客户端，强化网络，也就是说，过去我们上网的习惯是下载，把好的东西都下载到本地来，现在要反过来。比如以前我们都是把 Email 收到本地，不然网络信箱就会满而收不了信，但这就带来了一个坏处，你要是没有在这台机器旁，你想看你以前的信就不可能了，甚至操作系统崩溃后可能这些信就没了，当然照片，文档也是如此。现在呢，最好的办法是把所有的信息，都放在网络上，比如 Gmail 里，比如 163 里，或者什么网站硬盘里（最好是大公司）。只要有网络你想在什么地方用它都行。”

“明白，同样道理，我现在看到的 在线Office、在线Web OS、在线图片处理等应用，都是Flex的应用，也就是说以后这些都不需要操作系统里安装什么软件了。”

“哈，小菜开窍了，这样操作系统的功能就下降了，以前要听音乐看电影都需下载，现在都是在线听，以前游戏都是在家独自玩，现在都网游，现在聊天都 QQ，MSN，将来都 WebIM，总有一天，客户端会只是一个类似浏览器的东东就行的，不只是电脑，手机，家电都能上 Internet，那才是未来发展的方向。不过，你知道吗？真这样的话，谁最不开心？”

“嗯？谁会不开心？这不是大家都好吗？”

“哈，当然是微软最不开心，因为他们是卖操作系统的呀，操作系统功能越来越弱化了，他们的市场不就小了吗？所以同样是推出富客户端应用（WPF 和 Flex），微软和 Adobe 的想法是不一样的，微软是希望和操作系统连接更紧密，而 Adobe 或 Google 却是希望客户端越简单越好。”

“大鸟呀，你说话前后矛盾的，刚才还在和我说‘Flex 这种富客户端技术，不过是个噱头罢了’，现在又在大谈富客户端的好，你到底是什么意思？”

“咱们领袖毛老人家说过，看问题要用辩证法，事物都有两面性。富客户端有好的方面，当然也存在着不足，现在这技术还不够成熟，至少在以文字内容为主的网站，这种技术还是没太大的用处，所以妙句网只是一个噱头，不能成大气候。”

“我却觉得，新事物总会有人说不好的，妙句这个创意本身很好，用这个技术也感觉和一般网站不一样，只不过现在刚开始，还没有发挥 Flex 的特性吧，听说 AJAX 和 Flex 可以直接通讯的，或许和 html 结合一下，句子能被搜索引擎找到，加上性能优化后让客户端 CPU 压力不太大，我相信它会越来越好起来。我决定了，我也要学 Flex，做一个贼酷的富客户端网站。”

“得了吧你，”大鸟说道，“你还是好好打好基础，上会你写完计算器的程序，后面我留的作业你做了没有，就是做一个商场收银软件，营业员根据客户购买商品单价和数量，向客户收费。”

“啊，我忘记了，那个没什么大问题，我尽快去写。”

第七章 工厂不好用了？

小菜心里想：“大鸟要我做的是一个商场收银软件，营业员根据客户购买商品单价和数量，向客户收费。这个很简单，两个文本框，输入单价和数量，再用个列表框来记录商品的合计，最终用一个按钮来算出总额就可，对，还需要一个重置按钮来重新开始，不就行了？！”

代码样例（可使用）：

商场收银系统v1.0关键代码如下：

```
//声明一个double变量total来计算总计
double total = 0.0d;
private void btnOk_Click(object sender, EventArgs e)
{
    //声明一个double变量totalPrices来计算每个商品的单价（txtPrice）*数量(txtNum)后的合计
    double totalPrices = Convert.ToDouble(txtPrice.Text) *
Convert.ToDouble(txtNum.Text);
    //将每个商品合计计入总计
    total = total + totalPrices;
    //在列表框中显示信息
    lbxList.Items.Add(
        "单价：" + txtPrice.Text + " 数量：" +
        txtNum.Text + " 合计：" + totalPrices.ToString());
    //在lblResult标签上显示总计数
    lblResult.Text = total.ToString();
}
```

“大鸟，”小菜叫道，“来看看，这不就是你要的收银软件吗？我不到半小时就搞定了。”

“哈哈，很快吗，”大鸟说着，看了看小菜的代码。接着说：“现在我要求商场对商品搞活动，所有的商品打 8 折。”

“那不就是在 totalPrices 后面乘以一个 0.8 吗？”

“小子，难道商场活动结束，不打折了，你还要再把程序改写代码再去把所有机器全部安装一次吗？再说，我现在还有可能因为周年庆，打五折的情况，你怎么办？”

小菜不好意思道：“啊，我想得是简单了点。其实只要加一个下拉选择框就可以解决你说的问题。”

大鸟微笑不语。

商场收银系统 v1.1 关键代码如下：

```
double total = 0.0d;
private void btnOk_Click(object sender, EventArgs e)
{
    double totalPrices = 0d;
    //cbxType是一个下拉选择框，分别有“正常收费”、“打8折”、“打7折”和“打5折”
    switch (cbxType.SelectedIndex)
    {
    case 0:
        totalPrices =
            Convert.ToDouble(txtPrice.Text) * Convert.ToDouble(txtNum.Text);
        break;
    case 1:
        totalPrices =
            Convert.ToDouble(txtPrice.Text) * Convert.ToDouble(txtNum.Text) * 0.8;
        break;
    case 2:
        totalPrices =
            Convert.ToDouble(txtPrice.Text) * Convert.ToDouble(txtNum.Text) * 0.7;
        break;
    case 3:
        totalPrices =
            Convert.ToDouble(txtPrice.Text) * Convert.ToDouble(txtNum.Text) * 0.5;
        break;

    }
    total = total + totalPrices;
    lbxList.Items.Add(
        "单价：" + txtPrice.Text +
        " 数量：" + txtNum.Text + " " + cbxType.SelectedItem +
        " 合计：" + totalPrices.ToString());
    lblResult.Text = total.ToString();
```

}

“这下可以了吧，只要我事先把商场可能的打折都做成下拉选择框的项，要变化的可能性就小多了。”小菜说道。

“这比刚才灵活性上是好多了，不过重复代码很多，像Convert.ToDouble（），你这里就写了8遍，而且4个分支要执行的语句除了打折多少以外几乎没什么不同，应该考虑重构一下。不过还不是最主要的，现在我的需求又来了，商场的活动加大，需要有满300返100的促销算法，你说怎么办？”

“满300返100，那要是700就要返200了？这个必须要写函数了吧？”

“小菜呀，看来之前教你的白教了，这里面看不出什么名堂吗？”

“哦！我想起来了，你的意思是简单工厂模式是吧，对的对的，我可以先写一个父类，再继承它实现多个打折和返利的子类，利用多态，完成这个代码。”

“你打算写几个子类？”

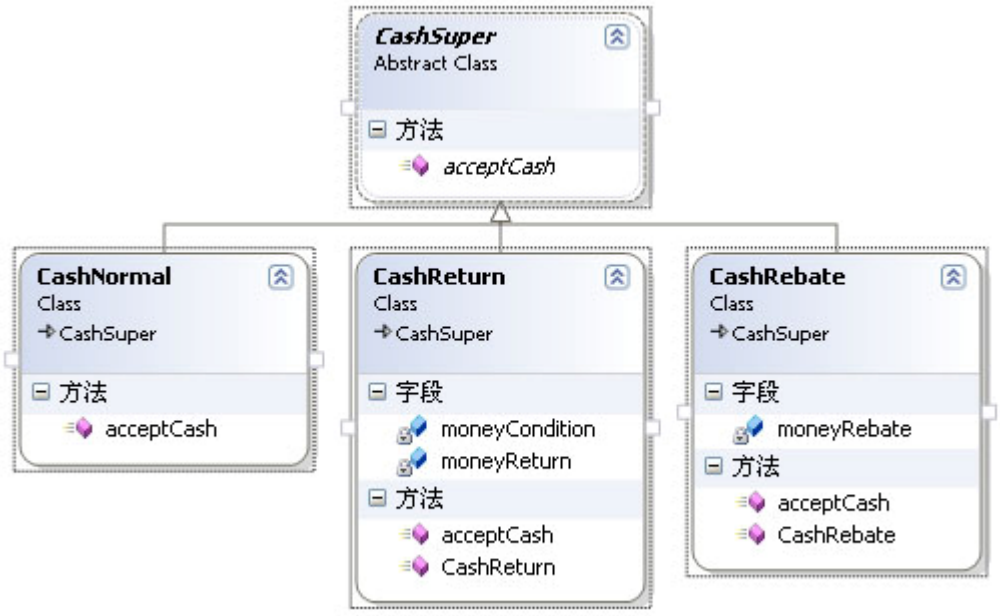
“根据需求呀，比如8折、7折、5折、满300送100、满200送50……要几个写几个。”

“小菜又不动脑子了，有必要这样吗？如果我现在要3折，我要满300送80，你难道再去加子类？你不想想看，这当中哪些是相同的，哪些是不同的？”

“  对的，这里打折基本都是一样的，只要有个初始化参数就可以了。满几送几的，需要两个参数才行，明白，现在看来不麻烦了。”

“面向对象的编程，并不是类越多越好，类的划分是为了封装，但分类的基础是抽象，具有相同属性和功能的对象的抽象集合才是类 。打一折和打九折只是形式的不同，抽象分析出来，所有的打折算法都是一样的，所以打折算法应该是一个类。好了，空话已说了太多，写出来再是真的懂。”

大约1个小时后，小菜交出了第三份的作业



商场收银系统v1.3关键代码如下

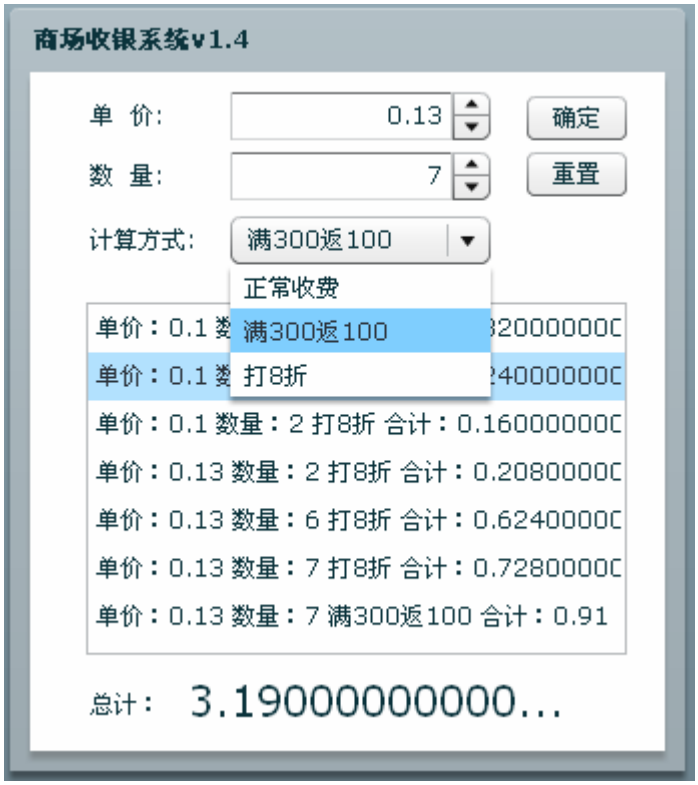
```
//现金收取父类
abstract class CashSuper
{
```

```
    //抽象方法：收取现金，参数为原价，返回为当前价
    public abstract double acceptCash(double money);
}
//正常收费，继承CashSuper
class CashNormal : CashSuper
{
    public override double acceptCash(double money)
    {
        return money;
    }
}
//打折收费，继承CashSuper
class CashRebate : CashSuper
{
    private double moneyRebate = 1d;
    //初始化时，必需要输入折扣率,如八折，就是0.8
    public CashRebate(string moneyRebate)
    {
        this.moneyRebate = double.Parse(moneyRebate);
    }

    public override double acceptCash(double money)
    {
        return money * moneyRebate;
    }
}
//返利收费，继承CashSuper
class CashReturn : CashSuper
{
    private double moneyCondition = 0.0d;
    private double moneyReturn = 0.0d;
    //初始化时必须要输入返利条件和返利值，比如满300返100，
    //则moneyCondition为300，moneyReturn为100
    public CashReturn(string moneyCondition, string moneyReturn)
    {
        this.moneyCondition = double.Parse(moneyCondition);
        this.moneyReturn = double.Parse(moneyReturn);
    }

    public override double acceptCash(double money)
    {
        double result = money;
        //若大于返利条件，则需要减去返利值
        if (money >= moneyCondition)
```

```
            result = money - Math.Floor(money / moneyCondition) * moneyReturn;

        return result;
    }
}
//现金收取工厂
class CashFactory
{
    //根据条件返回相应的对象
    public static CashSuper createCashAccept(string type)
    {
        CashSuper cs = null;
        switch (type)
        {
            case "正常收费":
                cs = new CashNormal();
                break;
            case "满300返100":
                CashReturn cr1 = new CashReturn("300", "100");
                cs = cr1;
                break;
            case "打8折":
                CashRebate cr2 = new CashRebate("0.8");
                cs = cr2;
                break;
        }
        return cs;
    }
}

//客户端窗体程序（主要部分）
CashSuper csuper;//声明一个父类对象
double total = 0.0d;
private void btnOk_Click(object sender, EventArgs e)
{
    //利用简单工厂模式根据下拉选择框，生成相应的对象
    csuper = CashFactory.createCashAccept(cbxType.SelectedItem.ToString());
    double totalPrices = 0d;
    //通过多态，可以得到收取费用的结果
    totalPrices = csuper.acceptCash(
        onvert.ToDouble(txtPrice.Text) * Convert.ToDouble(txtNum.Text)
        );
    total = total + totalPrices;
    lbxList.Items.Add(
```

```
            "单价：" + txtPrice.Text +
            " 数量：" + txtNum.Text + " " + cbxType.SelectedItem +
            " 合计：" + totalPrices.ToString());
        lblResult.Text = total.ToString();
}
```

代码样例（可使用）



“大鸟，搞定，这次无论你要怎么改，我都可以简单处理就行了。”小菜自信满满的说。

“是吗，我要是需要打 5 折和满 500 送 200 的促销活动，如何办？”

“只要在现金工厂当中加两个条件，在界面的下拉选项框里加两项，就OK了。”

“现金工厂？！你当是生产钞票呀。是收费对象生成工厂才准确。说得不错，如果我现在需要增加一种商场促销手段，满 100 积分 10 点，以后积分到一定时候可以领取奖品如何做？”

“有了工厂，何难？加一个积分算法，构造方法有两个参数：条件和返点，让它继承CashSuper，再到现金工厂，哦，不对，，是收—费—对—象—生—成—工—厂里加满 100 积分 10 点的分支条件，再到界面稍加改动，就行了。”

“嗯，不错，那我问你，如果商场现在需要拆迁，没办法，只能跳楼价销售，商场的所有商品都需要打 8 折，打折后的价钱再每种商品满 300 送 50，最后计总价的时候，商场还满 1000 送 200，你说如何办？”

“搞没搞错哦，这商场不如白送得了，哪有这样促销的？老板跳楼时估计都得赤条条的了。”

“商场大促销你还不高兴呀！当然，你是软件开发者，客户老是变动需求的确不爽，但你不能不让客户提需求呀，我不是说过吗，**需求的变更是必然**！所以**开发者应该的是考虑如何让自己的程序更能适应变化，而不是抱怨客户的无理，客户不会管程序员加班时的汗水，也不相信程序员失业时的眼泪**，因为客户自己正在为自己的放血甩卖而流泪呀。”

大鸟接着说：“简单工厂模式虽然也能解决这个问题，但的确不是最好的办法，另外由于商场是可能经常性的更改打折额度和返利额度，每次更改都需要改写代码重新编译部署真的是很糟糕的处理方式，面对算法的时常变动，应该有更好的办法。好好去研究一下设计模式吧，推荐你看一本书，《深入浅出设计模式》，或许你看完第一章，就会有解决办法了。”

小菜进入了沉思中……

本例C#源代码CashAcceptSystem.rar

另：建议大家去阅读《深入浅出设计模式》，第一章下载，本人非常喜欢这本书的风格，这是真正的做到了深入浅出呀。我也希望自己可以用类似的方式讲述问题。

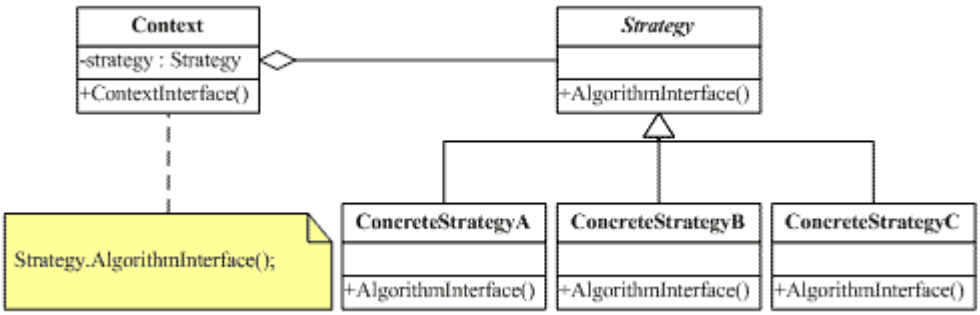
本文还有一个用意是对一些初学者，可以考虑一下大鸟提出的问题，在我的下一篇《小菜编程成长记 八》出来之前，改写我的源代码，实现更灵活更方便的商场收银程序共享给大家讨论，或许您写的东东比我写的还要好，那样就大家都有提高了。程序不是看出来的，是写出来的。好好加油！

第八章 用“策略模式”是一种好策略

小菜次日来找大鸟，说：“《深入浅出设计模式》的第一章我看完了，它讲的是策略模式(Strategy)。**『策略模式』定义了算法家族，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**。看来商场收银系统应该考虑用策略模式？”

“你问我？你说呢？”大鸟笑道，“商场收银时如何促销，用打折还是返利，其实都是一些算法，用工厂来生成算法对象，感觉是不是很怪？而最重要的是这些算法是随时都可能互相替换的，这就是变化点，而**封装变化点**是我们面向对象的一种很重要的思维方式。”

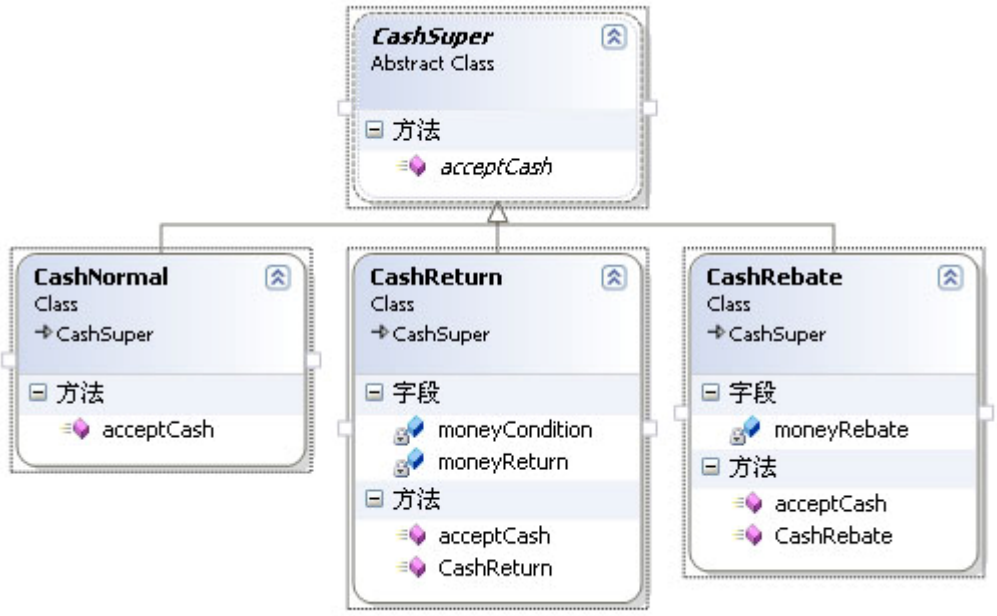
策略模式的结构 （源自吕震宇 博客）



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。

“我明白了，”小菜说，“我昨天写的 CashSuper 就是抽象策略，而正常收费 CashNormal、打折收费 CashRebate 和返利收费 CashReturn 就是三个具体策略，也就是策略模式中说的具体算法，对吧？”



“是的，那么关键就在于 Context 以及客户端程序如何写了？去查查资料，研究后把代码写出来给我看。”大鸟鼓励道。

“好的，我一定很快写出来给你看！”小菜很兴奋。

过一小时后，小菜给出商场收银程序的第四份作业。

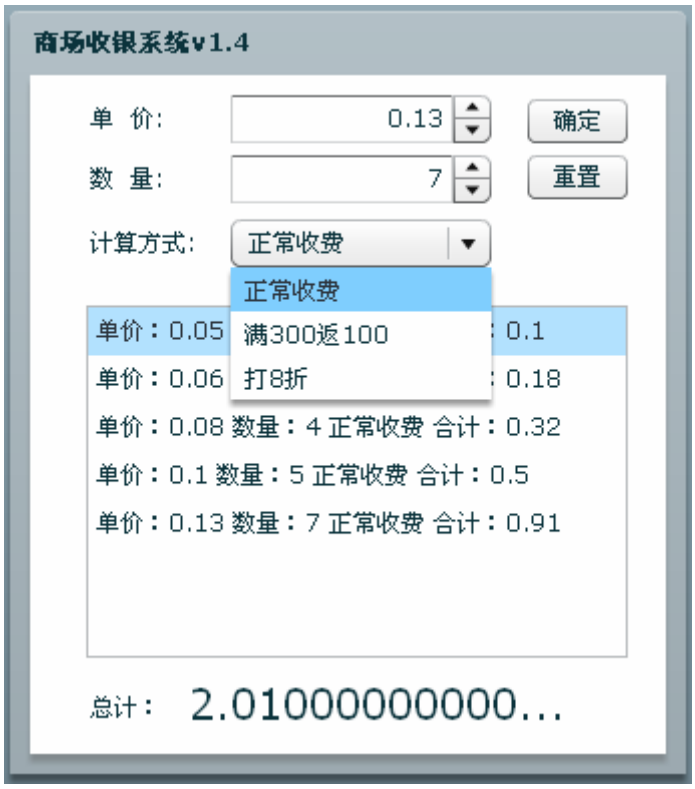
CashContext 类代码如下：

```
//收费策略Context
class CashContext
{
    //声明一个现金收费父类对象
    private CashSuper cs;

    //设置策略行为，参数为具体的现金收费子类（正常，打折或返利）
    public void setBehavior(CashSuper csuper)
    {
        this.cs = csuper;
    }

    //得到现金促销计算结果（利用了多态机制，不同的策略行为导致不同的结果）
    public double GetResult(double money)
    {
        return cs.acceptCash(money);
    }
```

```
}
```

客户端主要代码如下：

```
double total = 0.0d;//用于总计
private void btnOk_Click(object sender, EventArgs e)
{
    CashContext cc = new CashContext();
    switch (cbxType.SelectedItem.ToString())
    {
        case "正常收费":
            cc.setBehavior(new CashNormal());
            break;
        case "满300返100":
            cc.setBehavior(new CashReturn("300", "100"));
            break;
        case "打8折":
            cc.setBehavior(new CashRebate("0.8"));
            break;
    }

    double totalPrices = 0d;
    totalPrices = cc.GetResult(Convert.ToDouble(txtPrice.Text) *
Convert.ToDouble(txtNum.Text));
    total = total + totalPrices;
    lbxList.Items.Add(
        "单价：" + txtPrice.Text +
        " 数量：" + txtNum.Text + " " + cbxType.SelectedItem +
        " 合计：" + totalPrices.ToString());
    lblResult.Text = total.ToString();
}
```



“大鸟，我用策略模式是实现了，但有些疑问，用了策略模式，则把分支判断又放回到客户端来了，这等于要改变需求算法时，还是要去更改客户端的程序呀？”

“问得好，如果不是因为前面有工厂的例子，再来通过你的思考写出的这个策略模式的程序，你就问不出这样的问题的。”大鸟很开心，继续讲道，“**最初的策略模式是有缺点的，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**。”

“那还不如工厂模式好用，至少要增加促销或改进打折额度时，不用去大改界面，而现在，界面程序要承担的责任还是太大。没有体现你说的封装变化点的作用呀。”小菜疑问多多。

“就目前而言，的确是这样，这样的程序确实还是不够完善，要改的地方还很多。”大鸟说道，“不过正所谓病毒时间长了会有变种，杀毒软件本身也会随着病毒的变化而升级改良，如果我们对策略模式做一些改进，引入一些新的技术处理方式，就可以避免现在的这种耦合了。小菜，又有新的东西要学了，好好加油呀！”

“大鸟，谢谢你，，你总是让我带着问题去思考，而不是直接说答案，我觉得这样学习进步很快，也不觉得设计模式很难了。”

“，用不着这么客气，我只是觉得，**没有人是天生就牛X的**，有一些所谓的技术牛人总会在人面前说什么，‘你连这都不懂’，‘这还不简单了，你够笨的’等等说词。给人感觉他非常聪明，天生就会的样子，其实他在之前也不知走过多少弯路，犯过多少错，或许他之前也被更早的牛人羞辱过，所以再继续把羞辱传给后人。”大鸟有些激动。

小菜小心的说道：“大鸟，你是不是也曾经被人羞……”

“哈哈，马云曾说过，**男人的胸怀是被冤枉撑大的**！天天在这行当里混，阅人无数，被羞辱也是正常的事了。问题在于是不是**头脑清醒，自己不能放弃**呀。所以我希望能真正的帮助初学者成长，而不是去显示牛气充当狂人。小菜，记住，**学习一定是一个自己感悟的过程，而程序员的感悟就是自己写程序做项目，通过实践再学习，最终升华为牛人。**”

“嗯，我记住了，不过到底如何改良策略模式呢？”

大鸟微笑不语。

本文相关源代码CashAcceptSystem2.rar

第九章 反射——程序员的快乐！

“到底如何去改良策略模式呢？”小菜恳切地问道。

“你仔细观察过没有，你的代码，不管是用工厂模式写的，还是用策略模式写的，那个分支的switch依然去不掉。原因在哪里？”大鸟反问道。

“因为程序里有下拉选择，用户是有选择的，那么程序就必须要根据用户的选择来决定实例化哪一个子类对象。无论是在客户端窗体类编程还是到工厂类里编程，这个switch总是少不掉的。问题主要出在这里。”小菜十分肯定的说。

“是呀，”大鸟道，“所以我们要考虑的就是可不可以不在程序里写明‘如果是打折就去实例化CashRebate类，如果是返利就去实例化CashReturn类’这样的语句，而是在当用户做了下拉选择后，再根据用户的选择去某个地方找应该要实例化的类是哪一个。这样，我们的switch就可以对它说再见了。”

“听不太懂哦，什么叫‘去某个地方找应该要实例化的类是哪一个’？’小菜糊涂地说。

“，我要说的就是一种编程方式：**依赖注入**（Dependency Injection），从字面上不太好理解，我们也不去管它。关键在于如何去用这种方法来解决我们的switch问题。本来依赖注入是需要专门的IoC容器提供，比如spring.net，显然当前这个程序不需要这么麻烦，你只需要再了解一个简单的.net技术‘**反射**’就可以了。”

“大鸟，你一下子说出又是‘依赖注入’又是‘反射’这些莫名其妙的名词，我有点晕哦！”小菜有些犯困，“我就想知道，如何向switch说bye-bye！至于那些什么概念我不想了解。”

“**心急讨不了好媳妇**！你急什么？”大鸟嘲笑道，“反射技术看起来很玄乎，其实实际用起来不算难。”

“请看下面的两个样例：

```
1  //实例化方法一
2  //原来我们把一个类实例化是这样的
3  Animal animal=new Cat();  //声明一个动物对象，名称叫animal，然后将animal实例化成猫类的对象
4
5  //实例化方法二
6  //我们还可以用反射的办法得到这个实例
7  using System.Reflection;//先引用System.Reflection
8  //假设当前程序集是AnimalSystem，名称空间也是AnimalSystem
9  Animal animal = (Animal)Assembly.Load("AnimalSystem").CreateInstance("AnimalSystem.Cat");
```

其中关键是

Assembly.Load("程序集名称").CreateInstance("名称空间.类名称")

那也就是说，我们可以在实例化的时候，再给计算机一个类的名称字符串，来让计算机知道应该实例化哪一个类。”大鸟讲解道。

“你的意思是，我之前写的‘cc.setBehavior(new CashNormal());’可以改写为‘cc.setBehavior((CashSuper)Assembly.Load("商场管理软件").CreateInstance("商场管理软件.CashNormal")’，不过，这只不过是换了种写法而已，又有什么神奇之处呢？”小菜依然迷茫。

“分析一下，原来new CashNormal()是什么？是否是写死在程序里的代码，你可以灵活更换吗？”大鸟问。

“不可以，那还换什么，写什么就是什么了呗。”

“那你说，在反射中的CreateInstance("商场管理软件.CashNormal")，可以灵活更换‘CashNormal’吗？”大鸟接着问。

“还不是一样，写死在代码……等等，哦！！！我明白了。”小菜一下子顿悟过来，  ，兴奋起来。“因为这里是字符串，可以用变量来处理，也就可以根据需要更换。哦，My God！太妙了！”

“  哈哈，博客园中的有篇博文《四大发明之活字印刷——面向对象思想的胜利》中曾经写过，‘体会到面向对象带来的好处，那种感觉应该就如同是一中国酒鬼第一次喝到了茅台，西洋酒鬼第一次喝到了XO一样，怎个爽字可形容呀。’，你有没有这种感觉了？”

“嗯，我一下子知道这里的差别主要在原来的实例化是写死在程序里的，而现在用了反射就可以利用字符串来实例化对象，而变量是可以
更换的。”小菜说道。

“由于字符串是可以写成变量，而变量的值到底是CashReturn（返利），还是CashRebate（打折），完全可以由谁决定？”大鸟再问。

“当然是由用户在下拉中选择的选项决定，也就是说，我只要把下拉选项的值改成这些算法子类的名称就好了，是吧？”

“你说得对，不过还不是最好。因为把comboBox的每个选项value都改为算法子类的名称。以后我们要加子类，你不是还要去改comboBox吗？继续往下想，现在我们的代码对有谁依赖？”

“对下拉控件comboBox的选项有依赖。”

“那么怎么办，这个控件的选项可不可以通过别的方式生成。比如利用它的绑定？”

“你的意思是读数据库？”

“读数据库当然最好了，其实用不着这么麻烦，我们不是有XML这个东东吗，写个配置文件不就解决了？”

“哦，我知道你的意思了，让它去读XML的配置文件，来生成这个下拉列表框，然后再根据用户的选择，通过反射实时的实例化出相应的算法对象，最终利用策略模式计算最终的结果。好的好的，我马上去写出来。我现在真有一种不把程序写出来就难受的感觉了。”小菜急切的说。

“OK，还有一个小细节，你的CashRebate和CashReturn在构造函数中都是有参数的，这需要用到CreateInstance()方法的重载函数，不会用去查帮助吧！”

“好嘞！你别走哦，等我，不见不散！”小菜向外跑着还叫道。

大鸟摇头苦笑，嘴里嘟囔着：“这小子，忒急了吧！还不见不散呢，难道真没完没了啦！”

一个小时后，小菜交出了商场收银程序的第五份作业。

客户端主要代码：

```
using System.Reflection;

DataSet ds;//用于存放配置文件信息
double total = 0.0d;//用于总计

private void Form1_Load(object sender, EventArgs e)
{
    //读配置文件
    ds = new DataSet();
    ds.ReadXml(Application.StartupPath + "\\CashAcceptType.xml");
    //将读取到的记录绑定到下拉列表框中
    foreach (DataRowView dr in ds.Tables[0].DefaultView)
    {
        cbxType.Items.Add(dr["name"].ToString());
    }
    cbxType.SelectedIndex = 0;
}

private void btnOk_Click(object sender, EventArgs e)
{
    CashContext cc = new CashContext();
    //根据用户的选项，查询用户选择项的相关行
    DataRow dr = ((DataRow[])ds.Tables[0].Select
        ("name='" + cbxType.SelectedItem.ToString() + "'")
        )[0];
    //声明一个参数的对象数组
    object[] args = null;
    //若有参数，则将其分割成字符串数组，用于实例化时所用的参数
    if (dr["para"].ToString() != "")
        args = dr["para"].ToString().Split(',');
    //通过反射实例化出相应的算法对象
```

```
        cc.setBehavior(
            (CashSuper)Assembly.Load("商场管理软件").CreateInstance(
                "商场管理软件." + dr["class"].ToString(), false,
                BindingFlags.Default, null, args, null, null
            ));

        double totalPrices = 0d;
        totalPrices = cc.GetResult(Convert.ToDouble(txtPrice.Text) *
Convert.ToDouble(txtNum.Text));
        total = total + totalPrices;
        lbxList.Items.Add(
            "单价：" + txtPrice.Text +
            " 数量：" + txtNum.Text + " " + cbxType.SelectedItem +
            " 合计：" + totalPrices.ToString());
        lblResult.Text = total.ToString();
    }
```

配置文件 CashAcceptType.xml 的代码

```
<?xml version="1.0" encoding="utf-8" ?>
<CashAcceptType>
  <type>
    <name>正常收费</name>
    <class>CashNormal</class>
    <para></para>
  </type>
  <type>
    <name>满300返100</name>
    <class>CashReturn</class>
    <para>300,100</para>
  </type>
  <type>
    <name>满200返50</name>
    <class>CashReturn</class>
    <para>200,50</para>
  </type>
  <type>
    <name>打8折</name>
    <class>CashRebate</class>
    <para>0.8</para>
  </type>
  <type>
    <name>打7折</name>
    <class>CashRebate</class>
    <para>0.7</para>
  </type>
```

</CashAcceptType>

“大鸟，我再次搞定了，这会是真的明白了。”小菜说。

“说说看，你现在的理解！”大鸟问。

“无论你的需求是什么，我现在连程序都不动，只需要去改改 XML 文件就全部摆平。比如你如果觉得现在满 300 送 100 太多，要改成送 80，我只需要去 XML 文件里改就行，再比如你希望增加新的算法，比如积分返点，那我先写一个返点的算法类继承 CashSuper，再去改一下 XML 文件，对过去的代码依然不动。总之，现在是真的做到了程序易维护，可扩展。”  小菜得意地坏笑道，“吼吼！此时商场老板以为要改一天的程序，我几分钟就搞定，一天都可以休息。反射——真是程序员的快乐呀！”

“在做梦了吧，你当老板是傻瓜，会用反射才是正常水平，不会用的早应该走人了。”大鸟打击了小菜的情绪，“不过呢小菜的确是有长进，不再是小菜鸟了。那你说说看，现在代码还有没有问题。”

“还有不足？不会吧，我都改 5 次了，重构到了这个地步，还会有什么问题？”小菜不以为然。

“**知足是可以常乐，但知足如何能进步**！你的代码真的没有问题了，比如说，你现在把列表是打印在了 listBox 列表框中，我现在还需要输出到打印机打印成交易单据，我还希望这些清单能存入数据库中，你需要改客户端的代码吗？”

“这个，你这是加需求了，更改当然是必须的。”

“更改是必须的没有错，但为什么我只是要对交易清单加打印和存数据，就需要去改客户端的代码呢？这两者没什么关系吧？”大鸟说。

“啊，你的意思是……”

“别急着下结论，先去好好思考一下再说。”大鸟打断了小菜。

本文源代码CashAcceptSystem3.rar

第十章 会修电脑不会修收音机？——聊设计模式原则

小菜学会了反射后，正在兴奋，想着大鸟的问题。此时，突然声音响起。

“死了都要爱，不淋漓尽致不痛快，感情多深只有这样，才足够表白。死了都要爱……”

原来是小菜的手机铃声，大鸟吓了一跳，说道：”你小子，用这歌做铃声，吓唬人啊！这要是在公司开大会时响起，你要被领导淋漓尽致爱死！MD，还在唱，快接！”

小菜很是郁闷，拿起手机一看，一个美女来的电话，由  转  ，马上接通了手机，“喂！”

“小菜呀，我是娇娇我电脑坏了你快点帮帮我呀！”手机里传来急促的女孩声音。

“哈，是你呀，你现在好吗？最近怎么不和我聊天了？”小菜慢条斯理的说道。

“快点帮帮我呀，  ，电脑不能用了啊！”娇娇略带哭腔的说。

“别急别急，怎么个坏法？”

“每次打开QQ，一玩游戏，机器就死了。出来蓝底白字的一堆乱七八糟的英文，过一会就重启了，再用QQ还是一样。怎么办呀？”

“哦，明白了，蓝屏死机吧，估计内存有问题，你的内存是多少兆的？”

“什么内存多少兆，我听不懂呀，你能过来帮我修一下吗？”

“啊，你在金山，我在宝山，虽说在上海两地名都钱味儿十足，可两山相隔万重路呀！现在都晚上了，又是星期一，周六我去你那里帮你修吧！”小菜无耐的说。

“要等五天那不行，你说什么蓝屏？怎么修法？”娇娇依然急不可待。

“蓝屏多半内存坏了，你要不打开机箱看看，或许有两个内存，可以拔一根试试，如果只有一根内存，那就没戏了。”

“机箱怎么打开呢？”娇娇开始认真起来。

“这个，你找机箱后面，四个角应该都有螺丝，靠左侧边上两个应该就可以打开左边盖了。”小菜感觉有些费力，远程手机遥控修电脑，这是头一次。

“我好象看到了，要不先挂电话，我试试看，打开后再打给你。”

“哦，好的。”小菜正说着，只听娇娇边嘟囔着“老娘就不信收拾不了你这破电脑”边挂掉了电话。

“呵！”小菜长出一口气，“不懂内存为何物的美眉修电脑，强！”

“你小子，人家在困难时刻想得到你，说明心中有你，懂吗？这是机会！”大鸟说道。

“这倒也是，这小美眉长得蛮漂亮的，我看过照片。就是脾气大些，不知道有没有男朋友了。”

“切，你干吗不对她说，‘你可以找男友修呀’，真是没脑子，要是有男友，就算男友不会修也要男友找人搞定，用得着

找你求助呀，笨笨！”大鸟嘲笑道，“你快把你那该死的手机铃声换掉——死了都要爱，死了还爱个屁！”

“噢！知道了。”

十分钟后。

“我在这儿等着你回来，等着你回来，看那桃花开。我在这儿等着你回来，等着你回来，把那花儿采……”小菜的手机铃声再次响起。

“菜花痴，你就不能找个好听的歌呀。”大鸟气着说道。

“好好好，我一会改，一会改。”小菜拿起手机，一副很听话的样子，嘴里却跟着哼“我在这儿等着你回来哎”，把手机放到耳边。

“小菜，我打开机箱了，快说下一步怎么走！”娇娇仍然着急着说。

“你试着找找内存条，内存大约是 10 公分长，2 公分宽，上有多个小长方形集成电路块的长条，应该是竖插着的。”小菜努力把内存样子描述得容易理解。

“我看到一个风扇，没有呀，在哪里？”娇娇说道，“哦，我找到了，是不是很薄，很短的小长条？咦，怎么有两根？”

“啊，太好了，有两根估计就能解决问题了，你先试着拔一根，然后开机试试看，如果还是死机，再插上，拨另一根试，应该总有一根可以保证不蓝屏。”

“我怎么拨不下来呢？”

“旁边有卡子，你扳开再试。”

“嗯，这下好了，你别挂，我这就重启看看。”

十分钟后。

“哈，没有死机了啊，小菜，你太厉害了，我竟然可以修电脑了，要我怎么感谢你呢！”娇娇兴奋地说。

“最好以身相许吧，”小菜心里这么遐想着，口中却谦虚地说：“不客气，都是你聪明，敢自己独自打开机箱修电脑的女孩很少的。你把换下的内存去电脑城换掉，就可以了。”

“我不懂的，要不周六你帮我换？周六我请你吃饭吧！”

“这怎么好意思——你说在什么时间在哪碰面？”小菜假客气着，却不愿意放弃机会。

“周六下午 5 点在徐家汇太平洋数码门口吧。”

“好的，没问题。”

“今天真的谢谢你，那就先Bye-Bye了！”

“嗯，拜拜！”

“小菜走桃花运了哦，”大鸟有些羡慕道，“那铃声看来有些效果，不过还是换掉吧，俗！”

“嘿嘿，你说也怪，修电脑，这在以前根本不可能的事，怎么就可以通过电话就教会了，而且是真的修到可以用了呢。”

“你有没有想过这里的最大原因？”大鸟开始上课了。

“蓝屏通常是内存本身有问题或内存与主板不兼容，主板不容易换，但内存却只需要更换就可以了，而且换起来很容易。”

“如果是别的部件坏了，比如硬盘，显卡，光驱等，是否也只需要更换就可以了？”

“是呀，确实很方便，只需要懂一点点计算机知识，就可以试着修电脑了。”

“想想和我们编程有什么联系？”

“你的意思是——面向对象？”

“说说看，**面向对象的四个好处**？”

“这个我记得最牢了，就是活字印刷那个例子啊，**是可维护、可扩展、可复用和灵活性好**。我知道了，可以把PC电脑理解成是大的软件系统，任何部件如CPU、内存、硬盘，显卡等都可以理解为程序中封装的类或程序集，由于PC易插拔的方式，那么不管哪一个出问题，都可以在不影响别的部件的前题下进行修改或替换。”

“PC电脑里叫易插拔，面向对象里把这种关系叫什么？”

“应该是叫**强内聚、松耦合**吧。”

“对的，非常好，我们电脑里的CPU全世界也就是那么几家生产的，大家都在用，可是就是不知道Intel/AMD等是如何做出这个小东西。去年国内不是还出现了汉芯造假的新闻吗！这就说明CPU的强内聚的确是强。但它又独自成为了产品可以在千千万万的电脑主板上插上就可以使用，这是什么原因？”大鸟又问。

“因为CPU的对外都是针脚式或触点式等标准的接口。啊，我明白了，这就是接口的最大好处。CPU只需要把接口定义好，内部再复杂我也不让外界知道，而主板只需要预留与CPU针脚的插槽就可以了。”

“很好，你已经在无意的谈话间提到了设计模式其中的几大设计原则，单一职责原则，开放—封闭原则，依赖倒转原则（参考《敏捷软件开发——原则、模式与实践》）”大鸟接着讲道，“所谓**单一职责原则，就是指就一个类而言，应该仅有一个引起它变化的原因**，就刚才修电脑的事，显然内存坏了，不应该成为更换CPU的理由。**开放—封闭原则是说对扩展开发，对修改关闭**，通俗的讲，就是我们在设计一个模块的时候，应当使这个模块可以在不被修改的前提下被扩展，换句话说就是，应当可以在不必修改源代码的情况下改变这个模块的行为。比如内存不够只要插槽多就可以加，比如硬盘不够了，可以用移动硬盘等，PC的接口是有限的，所以扩展有限，软件系统设计得好，却可以无限的扩展。**依赖倒转原则**，原话解释**是抽象不应该依赖细节，细节应该依赖于抽象**，这话绕口，说白了，就是要针对接口编程，不要对实现

编程，无论主板、CPU、内存、硬盘都是在针对接口编程，如果针对实现编程，那就会出现换内存需要把主板也换了的尴尬。你想在小MM面前表现也就不那么容易了。所以说，PC电脑硬件的发展，和面向对象思想发展是完全类似的。这也说明**世间万物都是遵循某种类似的规律，谁先把握了这些规律，谁就最早成为了强者**。”

“还好，她没有问我如何修收音机，收音机里都是些电阻、三极管，电路板等等东东，我可不会修的。”小菜庆幸道。

“哈，小菜你这个比方打得好，”大鸟开心的说，“收音机就是典型的耦合过度，只要收音机出故障，不管是声音没有、不能调频、有杂音，反正都很难修理，不懂的人根本没法修，因为任何问题都可能涉及其它部件。非常复杂的PC电脑可以修，反而相对简单的收音机不能修，这其实就说明了很大的问题。当然，电脑的所谓修也就是更换配件，CPU或内存要是坏了，老百姓是没法修的。其实现在在软件世界里，收音机式强耦合开发还是太多了，比如前段时间某银行出问题，需要服务器停机大半天的排查修整，这要损失多少钱。如果完全面向对象的设计，或许问题的查找和修改就容易得多。”

“是的是的，我听说很多银行目前还是纯C语言的面向过程开发，非常不灵活，维护成本是很高昂的。”

“那也是没办法的，银行系统哪是说换就换的，所以现在是大力鼓励年轻人学设计模式，直接面向对象的设计和编程，从大的方向上讲，这是国家大力发展生产力的很大保障呀。”

“大鸟真是高瞻远瞩呀，我对你的敬仰犹如滔滔江水，连绵不绝！”小菜怪笑道  ，“我去趟WC”。

“浪奔，浪流，万里江海点点星光耀，人间事，多纷扰，化作滚滚东逝波涛，有泪，有笑……”

“小菜，电话。小子，怎么又换成上海滩的歌了，这歌好听。”大鸟笑道，“刚才是死了都要爱，现在是为爱复仇而死。你怎么找的歌都跟爱过不去呀。快点，电话，又是刚才那个叫娇娇的小MM的。”

“来了来了，尿都只尿了一半！”小菜心急地接起电话，“喂！”

“小菜呀，我家收音机坏了，你能不能教我修修呢！”

第十一章 三层架构，分层开发

“大鸟，我们继续讨论吧！”小菜很沮丧的说。

“小伙子，不会修收音机也是很正常的，没什么大不了的，用不着丧着一个脸。好象失恋一样，男人再强也要学会说‘不’。”大鸟安慰着说，“如果你的目标是要成为修理电器专家，那么你连收音机都不会修，那是很郁闷的事。但你现在的目标是什么？”

“我想成为软件架构师，编程专家。”小菜毫不含糊的说。

“就是，你的**人生目标很明确，别的方面弱一些有什么关系呢**。”大鸟继续说道，“现在电视节目《波士堂》里请来的嘉宾，全是中国的大企业家，许多人身家上亿，节目中都要求他们要有一个 Boss 秀，难道真的要把他们的才艺去和人家艺术家比吗，我看老板们唱歌虽很业余，但却也感觉得到他们那份认真和情趣——原来亿万富翁也是会唱歌，会跳舞，会食人间烟火的。至于他们歌唱得是不是跑调没有人在意的，明白吗？”

“我明白！  ，我一定要好好努力，成为编程专家。”，小菜说，“我们言归正传，你说我那程序用了反射后，还有什么需要修改的呢？”

“嗯，好！”大鸟清了清嗓子，开始上课，“如果你的程序再也不修改了，或者就是改改打折的额度和返利额度，那么你的代码是足够可以了。不过需求却是会不断产生的。比如说，现在这个程序是单机版的程序，如果需要商场多层楼的所有收银机都要使用，那该怎么办？”

“那用 XML 的配置文件就不合适了，应该用数据库会比较好！”

“那么老板听说了 C/S 架构的坏处，更新麻烦，不够安全等等，他也不是傻瓜，每次更新都需要针对每台机器部署，一次就半天，那些工作时间他是需要给程序员付薪水的。所以他提出要改为 B/S 架构，客户端用浏览器支持，你怎么办？”

“那需要改界面了，把应用程序改成 Web 程序。”

“就你现在的代码，改起来容易吗？”

“好象不容易，需要重新写，尽管可以复制一些代码过去，不过要重新写的东西还是很多的。”

“好，那你有没有发现，我说了这么多的需求变动，但系统中有一些东西一直没有变，是哪些？”

“我知道，是策略模式用到的那几个类，也就是正常收费、打折消费、返利消费等算法是没有变化的。”

“是呀，其实不是算法不会变，而是之前我们已经考虑它很多了，用了策略模式，用了反射技术使得它的变化相对稳定。你刚才也说，要把应用程序改为 Web 是需要复制粘贴的，可实际上，改改界面和这些算法有什么关系？”

“没有关系。”

“还有，把配置文件改为数据库访问，这其实是读取写入数据的操作，和算法又有什么关系呢？”

“也没有关系，我知道了，你是说，他们之间完全可以分离开，互不影响，改动其一，不要影响其它两者？哦，这是不是就是所谓的三层架构？”

“对，说得好，就是三层架构。**三层架构或者分层开发说起来容易，在程序开发时的初学者还是有很多的误解**。比如有些初学者以为，DBServer-WebServer-Client 是三层架构，其实这是物理意思上的三层架构，和程序的三层架构没有什么关系。还有人以为，WinForm 界面的窗体或者 WebForm 的 aspx 是最上一层，它们对应的代码后置（codebehind）文件 Form.cs 或 aspx.cs 是第二层，然后再有一个访问数据库的代码，比如 ado.cs 或 SqlHelper.cs 是最下一层，这其实也是非常错误的理解。再有，很多人认为 MVC 模式（Model-View-Controler）就是三层架构，这是比较经典的错误理解了。总之，尽管三层架构不算难，不过由于现在很多数书籍材料的讲解不透，所以让我们初学者都概念模糊，理解有误，非常的可惜的。”

“啊，我一直以为 MVC 就是三层架构呀，看来真的弄错了。那么三层具体是什么呢？”

“我不是已经告诉你了吗？你说说看，不管是应用程序 WinForm，还是网页程序 Aspx，它们主要用来干吗的？”

“用来界面显示和处理的，对的，它们可以看作是一层。叫界面层？”

“界面层这种叫法可以，或者叫 UI 层、表现层都可以。”

“访问配置文件或处理数据库是不是就是数据层了？”

“哈，三层架构是不是不难理解呀！说得很对，不过名称应该叫做数据访问层（Data Access Layer）或简称 DAL 层。”

“那么第三个层就是那些算法类了，这叫什么层呢？”

“这些算法是谁制定的？由谁来决定其变化？”

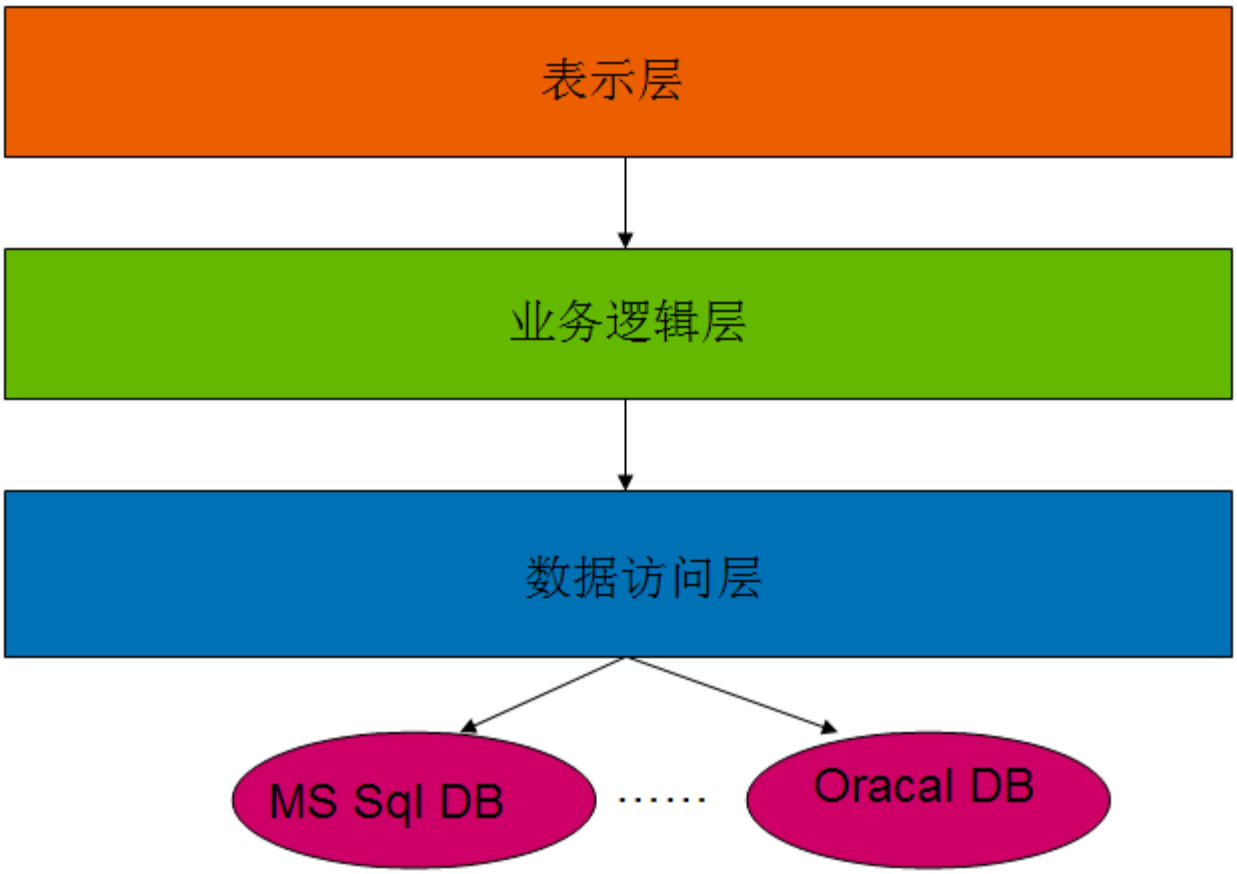
“当然是需求提出者，即软件系统所有者制定的，他们要改算法，我们开发就得改。这都是他们的业务算法呀！”

“哈，好，你说到了一个词，业务（Business）或叫商务，这其实是软件的核心，我们就是根据业务规则来开发软件提供服务的，所以这个层叫做业务逻辑层（Business Logic Layer）。不过它应该是中间的一层，介于另两者之间。”

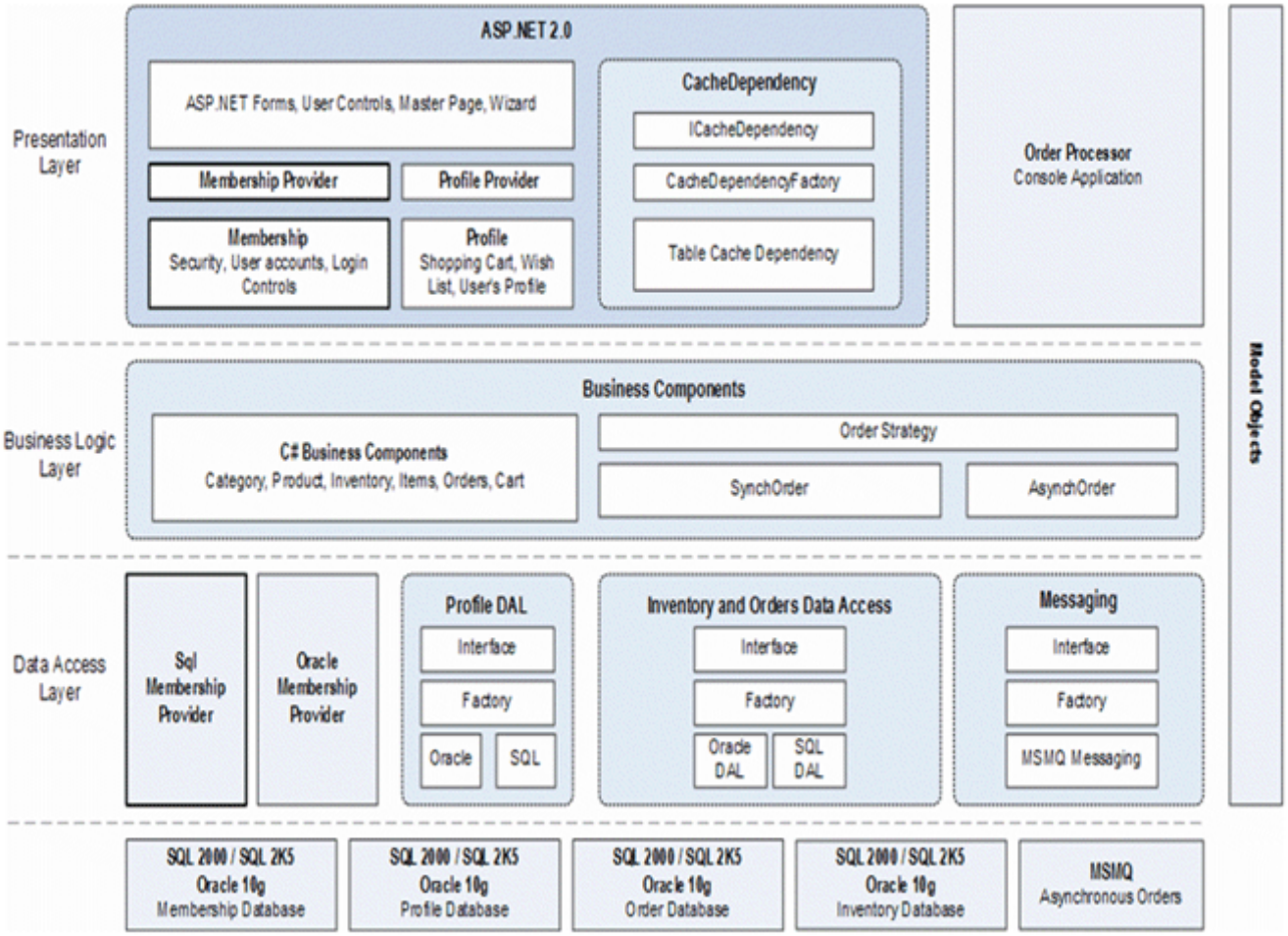
“哦，**所谓的三层开发，就是关于表现层、业务逻辑层和数据访问层的开发**。那么他们之间的关系呢？”

“你需要知道，**这其实只是大方向的分层，每个层中都有可能再细分为多个层次和结构**。比如 PetShop4，这是微软用它来展示.Net 企业系统开发的能力的范例，PetShop 尽管作为对大型软件系统开发的样例还是不够，但可以理解为儿童的智力玩具。不过对于初学编程的小菜你来说，玩具却是最好的学习道具。”

下面图源自Bruce Zhang博客



“如果是要细化，可能结构就会变得很复杂。比如给你看看 PetShop4 的结构图。”大鸟继续说道。



“啊，上面那图我是明白了，下面这图看得晕晕乎乎的，哪有这样复杂的玩具，大鸟又在故弄玄虚，快点解释一下？”小菜疑惑的说。

“第一次看到就完全看明白，那不就成天才了。学习它还需要慢慢来，以后再说。你现在应该对改写商场收银系统有点数了吧，应该怎么做呢？”

“应该原来的解决方案分为三个项目，一个 UI 项目，目前是 WinForm 的程序，一个 BLL 项目，用来把算法类都封装，还有一个 DAL 项目，用来访问配置文件。对吗？”

“嗯，差不多了，快去改吧，口说容易，实践中会有很多细节问题等着你去解决的。”

“好的，不过今天不行了，我前几天面试的一家公司给我 Offer 了，我明天就要去第一天上班，明晚我再去改写这个程序。”小菜说道。

“恭喜恭喜，就是你之前提到了那家做物流软件的公司吗？找到工作你得请客啦。”

“No problem，不过等我发工资吧。就是那一家。感觉公司还是很大的。”

“那你快去休息吧，第一天要好好表现哦！”

（待续）

应一些回复朋友的要求，专门写了关于 Web 架构方面的文章，本篇还只是简单介绍。其实这些都不是新鲜的东西，如果

你认为自己的确是小菜，我建议你去下载上一篇的代码来根据本篇的介绍去改写，编程是实践性很强的技术，理解不等于会应用的。

第十二章 无熟人难办事？——聊设计模式迪米特法则

次日傍晚，小菜敲开了大鸟家的门。

“回来啦！怎么样？第一天上班感受多吧。”大鸟关心的问道。

“感受真是多哦！！！”小菜一脸的不屑一顾  。

“怎么了？受委屈了吗。说说看怎么会事？”

“委屈谈不上，就感觉公司氛围不是很好。我一大早就到他们公司，正好我的主管出去了不在公司。人事处的小杨让我填了表后，就带我到IT部领取电脑，她向我介绍了一个叫‘小张’的同事认识，说我跟他办领取电脑的手续就可以了。小张还蛮客气，正打算要装电脑的时候，来了个电话，叫他马上去一个客户那里处理PC故障，他说要我等等，回来帮我弄。我坐了一上午，都没有见他回来，但我发现IT部其实人还有两个人，他们都在电脑前，一个忙于QQ，一个好象在看新闻。我去问人事的小杨，可不可以请其他人帮我办理领取手续，她说她现在也在忙，让我自己去找一下IT部的小李，他或许有空。我又返回IT部办公室，问小李帮忙，小李先是忙着回了两个QQ后才接过我领取电脑的单子，看到上面写着‘张XX’负责电脑领取安装工作，于是说这个事是小张负责的，他不管，叫我还是等小张回来再做吧。我就这样又像皮球一样被踢到桌边继续等待，还好我带着一本《重构》在看，不然真要郁闷死。小张快到下班的时候才回来，开始帮我装系统，加域，设置密码等，其实也就Gohost恢复再设置一下，差不多半小时就弄好了。”小菜感叹的说道，“就这样，我这人生一个最重要的第一次就这么渡过了。”

“哈哈，就业、结婚、生子，人生三大事，你这第一大事的第一次是够郁闷的。”大鸟同情道，“不过现实社会就是这样的，他们又不认识你，不给你面子，也是很正常的。就象现在曹启泰主持的电视《上班这点事》节目，当中可聊可学之事还真不少，上班可不是上学，复杂着呢。罢了罢了，谁叫你运气不好，你的主管在公司，事情就会好办多了。”

“不过，这家公司让你感觉不好原因在于管理上存在一些问题。”大鸟接着说，“这倒是让我想起来我们设计模式的一个原则，你的这个经历完全可以体现这个原则观点。”

“哦，是什么原则？”小菜情绪被调动了起来，“你怎么什么事都可以和软件设计模式搭界呢？”

“  ，大鸟我显然不是吹出来的……”大鸟洋洋得意道。

“啧啧，行了行了，大鸟你强！！！不是吹的，是天生的！  ，快点说说，什么原则？”小菜对大鸟的吹鸟腔调颇为不满，希望快些进入正题。

“你到了公司，通过人事部小杨，认识了IT部小张，这时，你已认识了两个人。但因没人介绍你并不认识IT部小李。而既然小张小李都属于IT部，本应该都可以给你装系统配帐号的，但却因小张有事，而你又不认识小李，而造成你的人生第一次大大损失，你说我分析得对吧？”

“你这都是废话，都是我告诉你的事情，哪有什么分析。”小菜失望道。

“如果你同时认识小张和小李，那么任何一人有空都可以帮你搞定了，你说对吧？”

“还是废话。”

“这就说明，你如果把人际关系搞好，所谓‘无熟人难办事’，你在IT部‘有人’，不就万事不愁了吗？”大鸟一脸坏笑  。

“大鸟，你到底想说什么？我要是有关系，对公司所有人都熟悉，还用得着你说呀。”

“小菜，瞧你急的，其实我想说的是，如果IT部有一个主管，负责分配任务，不管任何需要IT部配合的工作都让主管安排，不就没有问题了吗？”大鸟开始正经起来。

“你的意思是说，如果小杨找到的是IT的主管，那么就算小张没空，还可以通过主管安排小李去做，是吗？”

“对头（四川方言发音）。”大鸟笑着鼓励道。

“我明白了，关键在于公司里可能没有IT主管，他们都是找到谁，就请谁去工作，如果都熟悉，有事可以协调着办，如果不熟悉，那么就会出现我碰到的情况了，有人忙死，有人空着，而我在等待。”

“没有管理，单人情协调也很难办成事的。如果公司IT部就一个小张，那什么问题也没有，只不过效率低些。后来再来个小李，那工作是叫谁去做呢？外人又不知道他们两人谁忙谁闲的，于是抱怨、推诿、批评就随风而至。要是三个人在IT部还没有管理人员，则更加麻烦了。正所谓一个和尚挑水吃，两个和尚抬水吃，三个和尚没水吃。”

“看来哪怕两个人，也应该有管理才好。我知道你的意思了，不过这是管理问题，和设计模式有关系吗？”

“急什么，还没讲完呢？就算有IT主管，如果主管正好不在办公室怎么办呢？公司几十号人用电脑，时时刻刻都有可能出故障，电话过来找主管，人不在，难道就不解决问题了？”

“这个，看来需要规章制度，不管主管在不在，谁有空先去处理，过后汇报给主管，再来进行工作协调。”小菜也学着分析起来。

“是呀，就像有人在路上被车撞了，送到医院，难道还要问清楚有没有钱才给治疗吗，‘人命大于天’，同样的，在软件公司，‘电脑命大于天’，开发人员工资平均算下来每天按数百记的，耽误一天半天，实在是公司的大损失呀——所以你想过应该怎么办？”

“我觉得，不管认不认识IT部的人，我只要电话或亲自找到IT部，他们都应该想办法帮我解决问题。”

“好，说得没错，那你打电话时，怎么说呢？是说‘经理在吗？……小张在吗？……’，还是‘IT部是吧，我是小菜，电脑已坏，再不修理，软件歇菜。’”

“  ，当然是软件歇菜来得更好！你这家伙，就拿我开心！”

“这样子的话，公司不管任何人，找IT部就可以了，不管认识不认识人，反正他们会想办法找人来解决。”

“哦，我明白了，我真的明白了。你的意思是说，IT部代表是抽象类或接口，小张小李代表是具体类，之前你在分析会修电脑不会修收音机里讲的依赖倒置原则，即面向接口编程，不要面向实现编程就是这个意思？”小菜  ，兴奋异常。

“当然，这个原则也是满足的，不过我今天想讲的是另一个原则：‘**迪米特法则（LoD）’也叫最少知识原则，简单的**

说，就是如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用。如果其中一个类需要调用另一个类的某一个方法的话，可以通过第三者转发这个调用。其实道理就是你今天碰到的这个例子，你第一天去公司，怎么会认识IT部的人呢，如果公司有很好的管理，那么应该是人事的小杨打个电话到IT部，告诉主管安排人给小菜你装电脑，就算开始是小张负责，他临时有急事，主管也可以再安排小李来处理，如果小李当时不忙的话。其实**迪米特法则还是在讲如何减少耦合的问题，类之间的耦合越弱，越有利于复用，一个处在弱耦合的类被修改，不会对有关系的类造成波及。也就是说，信息的隐藏促进了软件的复用**。”

“明白，由于IT部是抽象的，哪怕里面的人都离职换了新人，我的电脑出问题也还是可以找IT部解决，而不需要认识其中的同事，纯靠关系帮忙了。就算需要认识，我也只要认识IT部的主管就可以了，由他来安排工作。”

“小菜动机不纯吗！你不会是希望小李快些被炒鱿鱼吧？哈！”大鸟瞧着小菜笑道  。

“去！！！我是那样的人吗？好了，你昨天说过，要我改商场收银代码为三层架构，有些麻烦的。我得想想。”

注：有回复说到《小菜编程成长记》系列讲问题不透，其实这是正常的，毕竟这不是上课，而是在写对话，聊天而已，建议看文章后若有学习的想法再去搜索相关主题研究，千万不能认为看了小菜系列就可以学懂设计模式。伍迷更希望是在你工作学习辛苦这余，看看《小菜》系列，调剂一下笑笑而已。另：本文迪米特法则知识来自《Java 与模式》，一本国人写的难得的好书，给出“购买”评级。

第十三章 有了门面，程序员的程序会更加体面

大鸟说道：“实际上没有学过设计模式去理解三层架构会有失偏颇的，毕竟分层是更高一级别的模式，所谓的架构模式。不过在程序中，有意识的遵循设计原则，却也可以有效的做出好的设计。”

“不要告诉我，刚才讲的‘迪米特法则’就会在分层中用得上？”小菜说。

“当然用得上，否则讲它干吗，你当我是在安慰你而临时编个法则来骗骗你呀？来，再来看看你上次写的代码。”

```
private void Form1_Load(object sender, EventArgs e)
{
    //读配置文件
    ds = new DataSet();
    ds.ReadXml(Application.StartupPath + "\\CashAcceptType.xml");
    //将读取到的记录绑定到下拉列表框中
    foreach (DataRowView dr in ds.Tables[0].DefaultView)
    {
        cbxType.Items.Add(dr["name"].ToString());
    }
    cbxType.SelectedIndex = 0;
}
```

“这是Form_Load的代码，里面有没有什么与界面无关的东西？”大鸟问道。

“第4、5行是读配置文件的代码，它应该属于DAL层。对吧？”

“很好，再看下面的这段，里面又有哪些呢？”

```
private void btnOk_Click(object sender, EventArgs e)
{
    CashContext cc = new CashContext();
    //根据用户的选项，查询用户选择项的相关行
    DataRow dr = (
        (DataRow[])ds.Tables[0].Select("name='" + cbxType.SelectedItem.ToString() +
"'")
        )[0];
    //声明一个参数的对象数组
    object[] args = null;
    //若有参数，则将其分割成字符串数组，用于实例化时所用的参数
    if (dr["para"].ToString() != "")
        args = dr["para"].ToString().Split(',');
    //通过反射实例化出相应的算法对象
    cc.setBehavior(
        (CashSuper)Assembly.Load("商场管理软件").CreateInstance("商场管理软件." +
            dr["class"].ToString(),
        false, BindingFlags.Default, null, args, null, null));

    double totalPrices = 0d;
    totalPrices = cc.GetResult(Convert.ToDouble(txtPrice.Text) *
Convert.ToDouble(txtNum.Text));
```

```
    total = total + totalPrices;
    lbxList.Items.Add(
        "单价：" + txtPrice.Text +
        " 数量：" + txtNum.Text + " " + cbxType.SelectedItem +
        " 合计：" + totalPrices.ToString());
    lblResult.Text = total.ToString();
}
```

“这里 3-13 行，是为确定哪种算法而创建 CashContext 对象，其中用到了反射技术，为计算做准备。第 16 行是真正的计算打折价或返利，17-19 是界面显示的部分。所以应该把 3-16 行都搬到 BLL 层去。不过，我还有些疑问，这样做会让配置文件的数据要先从 DAL 转到 BLL，再转到表示层，多麻烦呀，什么不直接表示层读 DAL，它想要数据就去读 DAL，它想算结果就去请求 BLL 处理？”

“那是说明你没有真的了解什么叫迪米特法则，象你那样说，不就等于，你小菜又要认识小张，又要认识小李了，这不就耦合过度吗？本来你只需要认识一个人就可以了，这样依赖才会小呀！”

“可是我就得在 BLL 里写一个专门返回从 DAL 里得到数据的方法，这个方法不属于现在的任何类，我就还得再写一个类来做这种传声筒的角色。而且由于界面还要涉及到其它的类，如 CashContext，感觉 UI 和 BLL 耦合还是很高。”

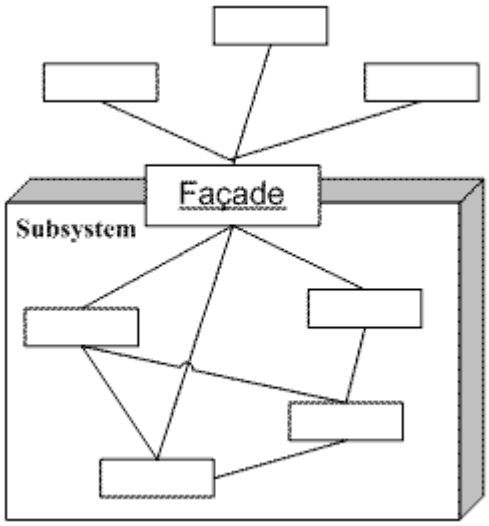
“说得没错，你的确是讲到点子上了，由于表示层 UI 需要与 BLL 有两个类进行交互，这是很麻烦，不过前辈们就想了了一个较好的办法，另一个设计模式，‘门面模式’（Facade）或叫外观模式”

（以下源自吕震宇 博客）

门面模式要求一个子系统的外部与其内部的通信必须通过一个统一的门面(Facade)对象进行。门面模式提供一个高层次的接口，使得子系统更易于使用。

门面模式的结构

门面模式是对象的结构模式。门面模式没有一个一般化的类图描述，下图演示了一个门面模式的示意性对象图：



在这个对象图中，出现了两个角色：

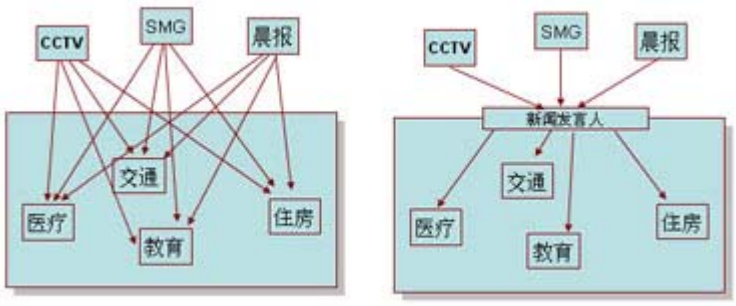
门面(Facade)角色：客户端可以调用这个角色的方法。此角色知晓相关的(一个或者多个)子系统的功能和责任。在正常情况下，本角色会将所有从客户端发来的请求委派到相应的子系统去。

子系统(subsystem)角色：可以同时有一个或者多个子系统。每一个子系统都不是一个单独的类，而是一个类的集合。每一个子系统都可以被客户端直接调用，或者被门面角色调用。子系统并不知道门面的存在，对于子系统而言，门面仅仅是另外一个客户端而已。

“哦，你这样一讲，我就明白了。”小菜说，“上篇所讲的 IT 部，其实可以由部门主管就是门面，我们只需要找到部门主管，就可以通过他安排相关的人来提供服务，我们不需要了解 IT 部的具体情况了。”

“其实现实中这样的例子很多。比如以前上海市没有新闻发言人，当要到春运时，所有的记者都跑到交通部去了解信息，当有非典或禽流感时，所有的记者又跑到卫生部去打听情况，突然这时候楼市大跌，记者们又得马不停蹄前往建设部收集新闻。辛苦呀，有什么办法呢，吃这口饭的。但其实辛苦地又何止只是记者。各个政府部门都需要专人来应付这些记者，不能多说话，不能说错话，但也不能不说话。也辛苦呀，谁叫他们是政府呢。”大鸟仿佛自己感同身受似的描述着，“于是，新闻发言人横空出世，一位知识女性焦扬，代表上海市政府发言，从此，老记们不需要头顶骄阳奔跑于各大政府部门之间，只需要天天等在新闻发言厅门口守着就可以写出准确及时的新闻。而政府部门也不用专人来应付老记们的围追堵截，有更多的时间为人民做实事办好事。这里就只辛苦一个人。”

“那一定是新闻发言人自己了，因为她需要先与政府部门沟通好，要说些什么、如何说、如何回答刁钻问题。然后要站在镁光灯下承受压力接受记者的访问。不过，干这一行就是需要辛苦的，这是政府的**门面**呀。”小菜感慨到。



“好了，去改写吧，你一定会感受到分层后代码的漂亮。”大鸟鼓励道。

过一小时后，小菜给出商场收银程序的第六份作业。。

DAL 层代码（目前是读配置文件，以后可以很容易的修改为访问数据库）

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace 商场管理软件.DAL
{
    public class CashAcceptType
    {
        public DataSet GetCashAcceptType()
        {
            //读配置文件到DataSet
            DataSet ds = new DataSet();
            ds.ReadXml("CashAcceptType.xml");
            return ds;
        }
    }
}
```

BLL层主要代码（Facade类代码）

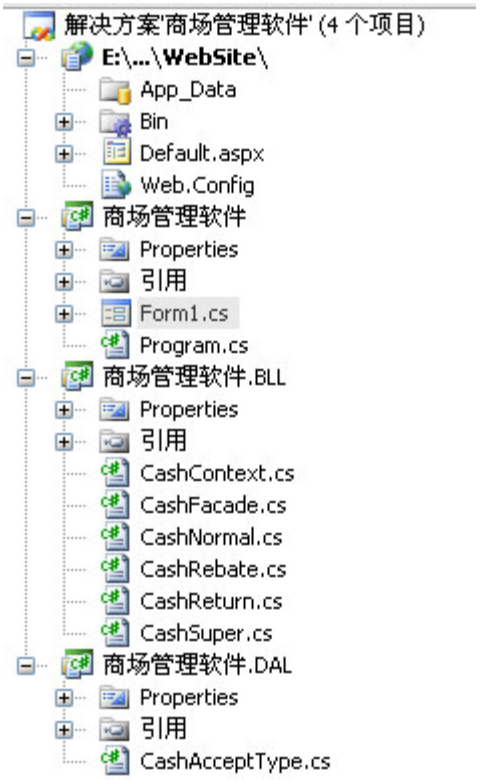
```
namespace 商场管理软件.BLL
{
    public class CashFacade
    {
        const string ASSEMBLY_NAME = "商场管理软件.BLL";

        //得到现金收取类型列表，返回字符串数组
        public string[] GetCashAcceptTypeList()
        {
            CashAcceptType cat = new CashAcceptType();
```

```
            DataSet ds = cat.GetCashAcceptType();
            int rowCount = ds.Tables[0].DefaultView.Count;
            string[] arrarResult = new string[rowCount];

            for (int i = 0; i < rowCount; i++)
            {
                arrarResult[i] = (string)ds.Tables[0].DefaultView[i]["name"];
            }
            return arrarResult;
        }

        /**/
        /// <summary>
        /// 用于根据商品活动的不同和原价格，计算此商品的实际收费
        /// </summary>
        /// <param name="selectValue">下拉列表选择的折价类型</param>
        /// <param name="startTotal">原价</param>
        /// <returns>实际价格</returns>
        public double GetFactTotal(string selectValue, double startTotal)
        {
            CashAcceptType cat = new CashAcceptType();
            DataSet ds = cat.GetCashAcceptType();

            CashContext cc = new CashContext();
            DataRow dr = (
                (DataRow[])ds.Tables[0].Select("name='" + selectValue + "'")
                )[0];
            object[] args = null;
            if (dr["para"].ToString() != "")
                args = dr["para"].ToString().Split(',');

            cc.setBehavior(
                (CashSuper)Assembly.Load(ASSEMBLY_NAME).CreateInstance
                (
                    ASSEMBLY_NAME + "." + dr["class"].ToString(),
                    false, BindingFlags.Default, null, args, null, null
                )
                    );
            return cc.GetResult(startTotal);

        }
    }
}
```

UI层代码（可以很容易的转换为Web页面）

```
double total = 0.0d;//用于总计
CashFacade cf = new CashFacade();

private void Form1_Load(object sender, EventArgs e)
{
    //读数据绑定下拉列表
    cbxType.DataSource = cf.GetCashAcceptTypeList();

    cbxType.SelectedIndex = 0;
}

private void btnOk_Click(object sender, EventArgs e)
{
    double totalPrices = 0d;
    //传进下拉选择值和原价，计算实际收费结果
    totalPrices = cf.GetFactTotal(
        cbxType.SelectedItem.ToString(),
        Convert.ToDouble(txtPrice.Text) * Convert.ToDouble(txtNum.Text)
        );
    total = total + totalPrices;
    lbxList.Items.Add(
        "单价：" + txtPrice.Text +
        " 数量：" + txtNum.Text + " " + cbxType.SelectedItem +
        " 合计：" + totalPrices.ToString());
    lblResult.Text = total.ToString();
}
```

项目文件结构图



“大鸟，来看看这下怎么样，还有没有可修改的地方？”小菜问道。

“小菜开始谦虚了吗！以前不是一直信誓旦旦，现在怎么，没信心了？”

“越学越觉得自己知道的少，感觉**代码重构没有最好，只有更好**呀。”小菜诚心的答道。

“写得很不错。BLL 层的 CashFacade 类其实就是新闻发言人，程序的门面；而应用程序或 Web 其实就类似 CCTV 和 SMG，都是新闻单位，他们不应该也不需要关心门面后面的实现是如何的。，**现在用了门面模式以后，耦合比以前要少很多了，更改会更加方便，扩展也很容易了**。你要是再回过头来看看最初的代码和现在的代码，你会体会更深刻，更加明白重构的魅力。”

重构前代码

```
double total = 0.0d;

private void Form1_Load(object sender, EventArgs e)
{
    cbxType.Items.AddRange(new object[] {"正常收费","打八折","打七折","打五折"});
    cbxType.SelectedIndex = 0;
}

private void btnOk_Click(object sender
{
    double totalPrices=0d;
    switch(cbxType.SelectedIndex)
    {
        case 0:
            totalPrices = Convert.ToDo
            break;
        case 1:
            totalPrices = Convert.ToDo
            break;
        case 2:
            totalPrices = Convert.ToDo
            break;
        case 3:
            totalPrices = Convert.ToDo
            break;

    }
    total = total + totalPrices;
    lbxList.Items.Add("单价：" + txtPrice.Text + " 数量：" + txtNum.Text
        + " "+cbxType.SelectedItem+ " 合计：" + totalPrices.ToString());
    lblResult.Text = total.ToString();
}
```

重构后代码

```
double total = 0.0d;//用于总计
CashFacade cf = new CashFacade();

private void Form1_Load(object sender, EventArgs e)
{
    //读数据绑定下拉列表
    cbxType.DataSource=cf.GetCashAcceptTypeList();
    cbxType.SelectedIndex = 0;
}

private void btnOk_Click(object sender, EventArgs e)
{
    double totalPrices = 0d;
    //传进下拉选择值和原价，计算实际收费结果
    totalPrices = cf.GetFactTotal(cbxType.SelectedItem.ToString(),
        Convert.ToDouble(txtPrice.Text) * Convert.ToDouble(txtNum.Text));
    total = total + totalPrices;
    lbxList.Items.Add("单价：" + txtPrice.Text + " 数量：" + txtNum.Text
        + " "+cbxType.SelectedItem+ " 合计：" + totalPrices.ToString());
    lblResult.Text = total.ToString();
}
```

大鸟接着说：“之前的代码，下拉控件的绑定是硬编码，所以只要改动需求就得改代码，现在是读配置文件，大大增加灵活性；之前的代码是根据用户选择，分支判断执行相应的算法，现在整个算法类全部搬走，做到了业务与界面的分离；之前的代码由于全写在 form 里，所以要更换成 Web 方式，即 C/S 改为 B/S 非常困难，要全部重新写（注意真实的软件系统不会这么简单，所以简单复制不能解决问题），现在的代码由于把业务运算分离，所以界面的更改不会影响业务的编写。还有就是现在的代码由于 DAL 与 BLL 分离，配置文件可以很容易的更换为数据库读取，且不需要影响表示层与业务逻辑层的代码。总的来讲，**若是程序不会变化，原有的设计就没什么问题，运行结果正确足够了，但若是程序可能会时常随业务而变化，新的设计就大大提高了应变性，这其实就是应用设计模式的目的所在**。”

“我现在越来越有信心学好它，设计模式真的很有意思，学它不学它，写出来的代码大不一样。老大，跟你混，看来没有错。”

“嗨，小菜，我不做老大已经很久了！”大鸟仰身长叹，扬长而去。

本文源代码。CashAcceptSystem4.rar

其中分四个项目，DAL、BLL、WebUI 和 WinUI，可设置 WebUI 和 WinUI 为启动项目，注意由于只是学习源代码，配置路径没有做处理（实际应用需要 config 文件），WebUI 配置文件 CashAcceptType.xml 在“\商场管理软件 06 分层\”根目录下，而 WinUI 的配置文件 CashAcceptType.xml 在“\商场管理软件 06 分层\商场管理软件\bin\Debug\”目录下。

第十四章 设计模式不能戏说！设计模式怎就不能戏说？

次日，小菜来到大鸟处。

“大鸟，你在写什么东西？”小菜看到大鸟的电脑上开着记事本。

“哦，我打算写篇博客，名字就叫《设计模式不能戏说？》”大鸟解释道。

“嘻嘻，废话，这又不是电视剧《戏说 XX》，可以乱讲不负责任，设计模式戏说了如何讲得清楚。怎么突然会想起来写这样的文章？”

“你知道为什么《Head First Design Patterns》（深入浅出设计模式）一直没有翻译成中文吗？”大鸟不答反问道。

“不知道，这本书国外出版好久了吧，得了 Jolt 大奖后，英文影印版在 2005 年国内也有出版了，近三年来一直不见中文翻译版本的出现，我也很奇怪，里面的英文其实也不算太难呀。”

“这就是因为在国内有一种观念，设计模式是不允许被戏说的！”

“戏说？那本书里写了很多生活中趣味的例子，又是鸭子又是匹萨店的，很有意思，我觉得这样写很生动，你指的戏说是这个？出版社不至于就因为这样而不翻译吧？”

“谁知道呢，反正没出版我们只能自己去猜想原因了。”

“不如你把它翻译了，去找出版社发吧，这书这么好，一定大大地有销量。”

“小菜说得轻松。你可以想象一下这样的场景：我通宵达旦、卧薪尝胆、励精图治把这本书翻译完，欢天喜地、兴高采烈、手舞足蹈拿着稿子到了 XX 出版社，XX 出版社的大编辑看了后说：‘大爷，您走错地儿了，到隔壁楼的出版社更合适一些。’于是我再怀着心神不宁、忐忑不安、焦急上火地跑到隔壁楼，却被门卫拦住，我抬头一看门牌——顿时怒火中烧、单脚蹬地，破口骂娘!  ”

“你看到了什么？”

“儿童画报社！”

为何要用设计模式？ 《Head First Design Patterns》书中截图

“，大鸟，说相声呐，搞笑呀！怎么可能呢？这书只不过图文并茂，通俗易懂而已，讲得可是实实在在的设计模式。”小菜大笑说道。

“怎么不可能，昨天一技术杂志社编辑找到我，说想和我约稿。我一想，最近教你的那些东西还很有意思的，也是我多年开发学习的经验积累，写写也不错。我问他能不能写成对话方式，他说不行，他们的杂志面向中高端用户，文章需要中规中矩。我觉得这其实也不算难，于是就写了一篇设计模式的文章。花了大半天时间，尽管去掉了以往对话中的调侃，算是很认真的正统了一把。不过正所谓**当戏说已成习惯，想改都难**。所以文章一给他看后，他很客气的说，‘您写得很认真，但是非常抱歉，不能采用’。”大鸟情绪开始有些激动，“我强压怒火，追问其原因，才得知是语言不专业、文章没深度，问他如何才叫语言专业，文章有深度。他给发了一篇样稿。小菜你可以看看？”

“《设计模式**********应用》（省去标题多字）”小菜望向屏幕，读了文章标题，扫了内容一遍。

……

“大鸟，这文章里面都是理论的东西，有点象论文，太深了，我看不太懂哦。”小菜实话实说，“不过，人家不是说了吗，面向中高端用户，我们这种菜鸟是没能力看这文章的。”

“哈，小菜也够谦虚，这下你明白了吧，《Head First Design Patterns》是不能出中文版的，因为它竟敢把如此神圣的高深的设计模式，写得如此通俗浅显，连菜鸟都读得懂，实在是太不给专家们面子了。”大鸟说，“当然细细想想，我那文章的确语言啰嗦、内容粗浅，不能发表也是可以理解的。换位思考，人家办个杂志也不容易，怎能随便冒被骂浅薄的风险。”

大鸟接着问道：“最近有听说过于丹这个人吗？”

“哦，你是说那个被称为‘学术超女’，在央视的《百家讲坛》讲课的于丹吧，最近好象新闻里有什么‘十博士联名批于丹’的消息，不过具体是怎么会事不清楚。”小菜说道。

“你有空可以去网上搜搜看，已经吵翻天了。那个所谓的十博士说要让在古典文学方面只有初中文化程度的于丹下课，并且为她的错误向全国人民道歉。你觉得是否荒唐？于丹讲的《论语》、《庄子》可能是有些地方解释不准确，举例有曲解，但和她将这些语录结合人生哲理和时尚元素，使得更多的人，特别是吸引了当下的年轻人去了解中国传统文化相比，贡献就远远大于瑕疵。实话说，《三国演义》我是读过的，所以听易中天讲不过是换种思路，《论语》《庄子》可真是没看过，我也相信现今 99%的年轻人如果不是于丹是不会去接触这些老古董，现在于丹火了，孔子庄子也跟着就火了，这其实是大大的好事。”

“是呀，年轻人也不是傻瓜。照本宣科的教育谁要听呀？我在大学里学了半年的面向对象，都不明白为什么要Object Oriented，这几天天天用设计模式，理解面向对象就深刻多了，我感觉面向对象的发明真的是奇迹。”

“我不知道现在大学里是否讲设计模式，但我能想象，如果老教授上课开口就说：‘今天我们讲桥接模式。’下边先倒下三个睡觉的，‘这个桥接模式是一个非常有用的模式，也是比较复杂的一个模式。……’，此时又倒下五个做梦去了。‘桥接模式的用意是"将抽象化(Abstraction)与实现化(Implementation)脱耦，使得二者可以独立地变化’，估计此时梦周公的人已经不下一半。……‘艾水娇同学，请问今天讲什么设计模式呀？’老教授有些气愤，突然提问一个趴在桌上的学生。‘今天讲的是……是……’小姑娘迷迷糊糊站起来。‘是桥接！’一个很小的声音从旁边传过来。小姑娘听到后说：‘是桥接……桥接……，哦，是巧 结 良 缘 模式！’  ，众人大笑，老教授欲哭  。”

“  我们以前上课老师就是这样照着书念呀，哈，那你说怎么上才好呢？”小菜问道。

“吭，”大鸟摆出一副老师的样，清了清嗓子，开始讲道，“我想大家小时候都有用蜡笔画画的经历吧。红红绿绿的蜡笔一大盒，根据想象描绘出格式图样。而毛笔下的国画更是工笔写意，各展风采。而今天我们的故事从蜡笔与毛笔说起。……呵呵，您是不是已经看出来了，不错，我今天要说的就是桥接模式（Bridge）。为了一幅画，我们需要准

备 36 支型号不同的蜡笔，而改用毛笔三支就够了，当然还要搭配上 12 种颜料。通过Bridge模式，我们把乘法运算 3×12＝36 改为了加法运算 3＋12＝15，这一改进可不小。那么我们这里蜡笔和毛笔到底有什么区别呢？……”（见吕震宇 的设计模式随笔－蜡笔与毛笔的故事 ）

“啊，这下子应该没人会去睡觉了，听故事比听模式好玩呀。”

“如果只听故事，那又何必要花那么多时间和金钱上大学。通过故事，再讲设计模式的原理，从现实生活入手去理解复杂的设计模式就要容易得多，这才是关键。再举个例子，小学时候老师让你学英语背单词，老是记不住，可是自从有了红白机（游戏机），什么‘Start’、‘Game Over’记得比谁都牢，也是同样的道理。”大鸟解释道。

“嗯，看来设计模式是需要戏说一下才会有劲，听得才会入味。就像于丹讲《论语》一样，如果都是正儿八经的解释字面含义，就没人看她的《百家讲坛》了。”

“我也不是崇洋媚外，同样是好酒，茅台的广告实在是不怎么样，除了最早去国外摔瓶子外，一直没有太好的创意。人家XO那句‘让灵感不只是空想’听再多也不觉得腻味，而‘人头马一开，好事自然来’把那种在酒桌上的吉利话都说绝了，这一开‘人头马’，你说生意还做不成吗？同样道理，设计模式也是需要一些噱头的，天天都是正统的论文式文章，没意思不说，主要是不容易看懂呀。这么好的东西就因为没意思而推广不出去，多让人心碎哦。”

“是呀，以前我刚听说设计模式的时候，去买了GoF的《设计模式：可复用面向对象软件的基础》，以为《葵花宝典》收为己有，**一书在手，万事不愁**，可我发现很多都不是太懂，理解不了。”

“那可是设计模式四大名著之首哦！”

“四大名著？”

“哈，这是大鸟我的认为，《**设计模式：可复用面向对象软件的基础**》、《**重构：改善既有代码的设计**》、《**Java与模式**》、《**重构与模式**》我认为是设计模式的四大名著，本来想把《**敏捷软件开发：原则、模式与实践**》也列入的，但考虑到《Java与模式》是国人之经典，加之《敏捷》中还有敏捷开发等软件工程的内容，所以没有列入。”

大鸟接着说：“GoF的书之所以位之首位，是因为他们第一次把设计模式系统的划分成了 23 个模式，总结了面向对象设计中最有价值的经验，并且用简洁可复用的形式表达出来。尽管是 95 年的作品，但到现在为止，他们总结的模式仍然是最经典的模式，没太大变化。可惜这书却也是这几本书中最难读懂的一本。要我说，**它的噱头最少，通篇都是精华，但由于晦涩难懂，这书也不知阻挡了多少有志青年学习设计模式的脚步**。”

“所以说，经典的也不是什么人都可以去读的，需要有初级，中级和高级读物。”小菜也总结道，“那你为什么不把《**Head First Design Patterns**》列入四大名著呢，它可是最受欢迎的设计模式书籍呀？要说戏说，我看《Java与模式》里讲故事的地方不比它少，又是西游记孙悟空，又是三国的锦囊妙计的。”

“是呀，可它老是不被翻译成中文版，总感觉国人不认可它。其实在 10 年前，美国人也只认可GoF的论文式书籍的，可后来 5 年内他们发现，任何模式不是一来就想到并用上的，那往往会造成设计模式的过度使用，而通过重构逐步演化并合理应用一些设计模式，却可让程序达到非常好的效果，所以就有了《重构》，再过 5 年，他们感觉就算只是对程序论程序也不足以说明问题，所以就开始戏说了。于是《敏捷》中用了不少有趣的例子，而后就有颠覆性的书籍《Head First》系列，把技术书籍写得如同儿童画报一样的通俗。这可是伟大的进步。”

“你的意思是，在国内还没有认识到这一点，所以依然还在如同美国 10 年前的论文方式？嗯，不管是书籍还是教学，国内的确都不足已达到与美国相提并论的程度。”

“还好国内也有了类似《Java与模式》的好书，以及博客园内如吕震宇、Bruce Zhang、李会军、WebCast的李建忠等等这样为设计模式推广做出大贡献的朋友。所以小菜呀，好好努力，要想超过老美，估计靠写技术书是不行了，哪天

弄出个什么设计模式小说、设计模式电影、设计模式话剧、设计模式相声等等才会有机会超英赶美呀！”

“设计模式相声？不会吧你，你戏说过头了哦！”

“哈哈，刚才那些什么电影、相声都是在扯淡，**设计模式游戏**，我却觉得是有可能的，**利用多媒体技术，让程序员与电脑交互，达到学习的目的，这比老师直接上课效果还要好得多，由于每个人基础不同，理解力不同，用游戏可以教育效果最大化**。好比美国军方都是利用电脑游戏来训练军人，而我们国家利用电视剧《亮剑》来教育军人展现精神，道理都差不多。”

“哇！设计模式游戏，多么吸引人的东东。如果现在就有该多好！”

“游戏哪是那么容易做出来的，首先要的是钱，不赚钱的事谁愿意投资呀？好了，和你说了那么多话，我气也消了，我决定修改那篇文章的题目，不叫《设计模式不能戏说？》了，应该叫《设计模式怎就不能戏说！》”

“是，设计模式怎么就不能戏说呢？——把戏说进行到底，让设计模式的戏说来得更猛烈些吧。大鸟，加油！！！”

注：本文纯属虚构，请不要对号入座，若有雷同，实属意外。另四大名著实在是伍迷一家之言，有不认可之处还望谅解。

更详细的图书已出版
http://www.cnblogs.com/cj723/archive/2007/11/23/962823.html