设计模式之观察者模式

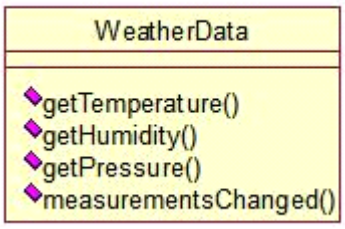
2016年5月15日　　23:34

观察者模式： 在对象之间定义一对多的依赖，这样一来，当一个对象改变状态，依赖它的对象都会收到通知，并自动更新。

举个例子：气象监测应用

此系统的三个部分是气象站（获取实际气象数据的物理装置）、WeatherData对象（追踪来自气象站的数据，并更新布告板）和布告板（有三个布告板，一个用来显示目前天气状况，一个显示气象统计，最后一个用来显示天气预报）

看一下WeatherData源文件：



上面的三个get各自返回最近的气象测量数据（分别为：温度、湿度、气压），我们不在乎这些变量如何通过气象站这个物理装置得到以及它们如何设置，WeatherData对象自己知道如何从气象站获取更新信息。

measurementsChanged() {/* 一旦气象测量更新，此方法会调用，这里面写自己的代码 */}

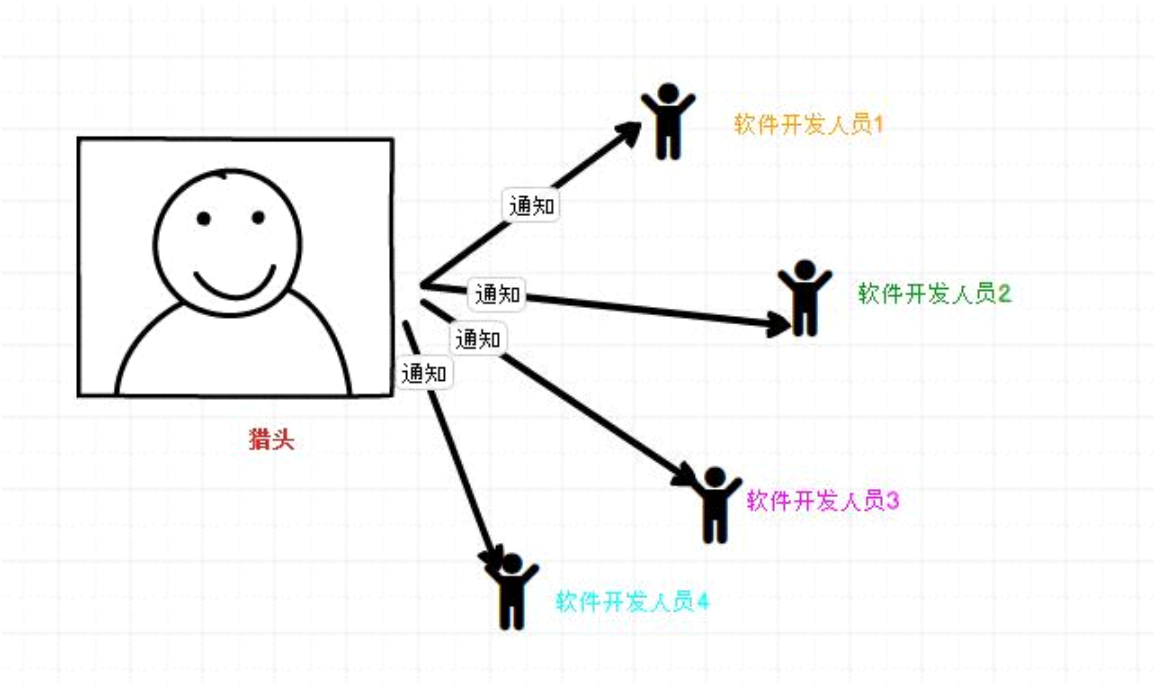
先看一个错误示范：

```
public class WeatherData{

    //实例变量声明
public void measurementsChanged()
    {
        //调用get函数，获取最近的测量值
        float temp = getTemperature();
        float humidity = getHumidity();
        float pressure = getPressure();
        //更新现在布告板、气象统计布告板、天气预报布告板
        currentConditionsDisplay.update(temp,humidity,pressure);
        statisticsDisplay.update(temp,humidity,pressure);
        forecastDisplay.update(temp,humidity,pressure);
    }
    //这里是其他的WeatherData方法
}
```

问题出在哪里？为什么这样是错误的？

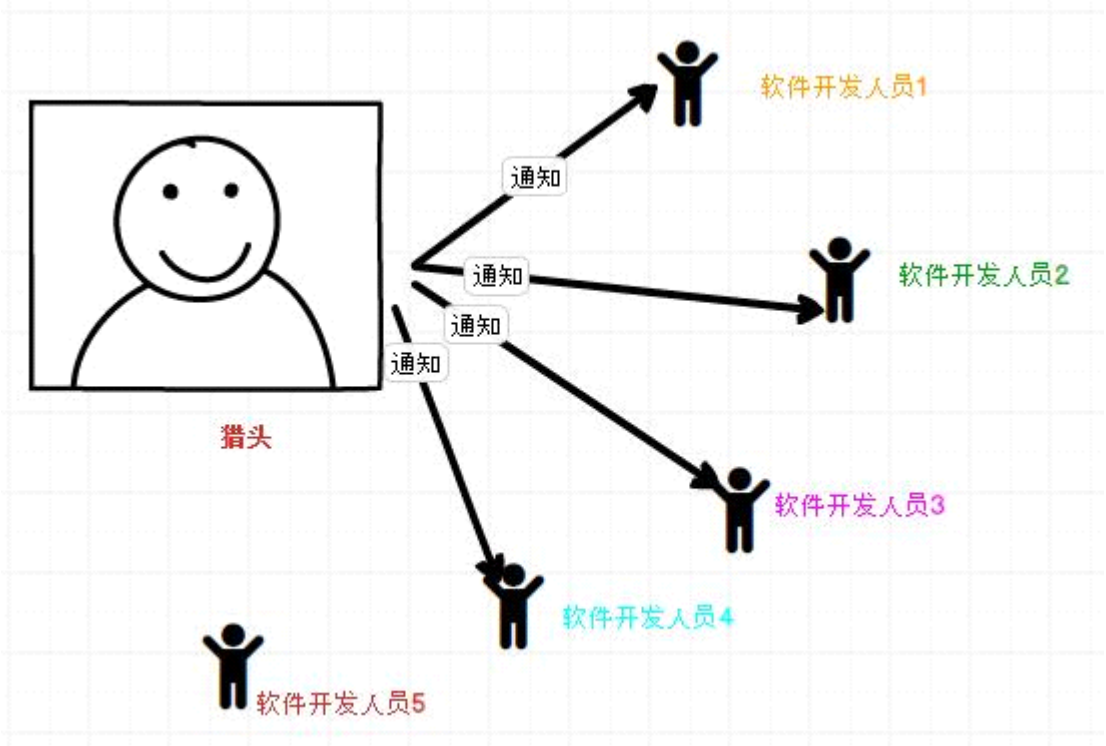
```
currentConditionsDisplay.update(temp,humidity,pressure);
statisticsDisplay.update(temp,humidity,pressure);
forecastDisplay.update(temp,humidity,pressure);
/*
以上属于具体实现编程，会导致我们以后在增加或删除布告板时必须修改程序*/
/*
另外上面的代码属于改变的部分，因此需要封装独立出来
*/
```

在继续向下走之前，先看一个观察者模式的实例

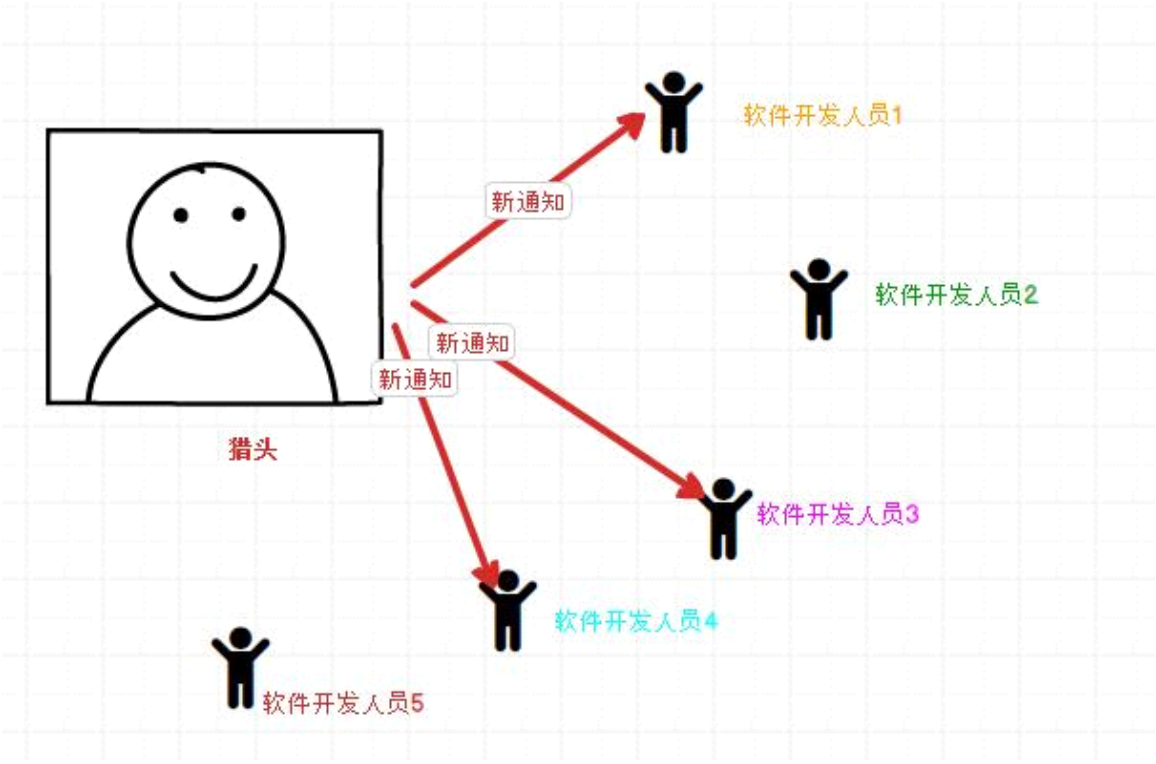


寻找工作的软件开发人员将自己的联系方式留给猎头，然后当猎头获取工作应聘消息时需要通知他手里名单上的所有人员。

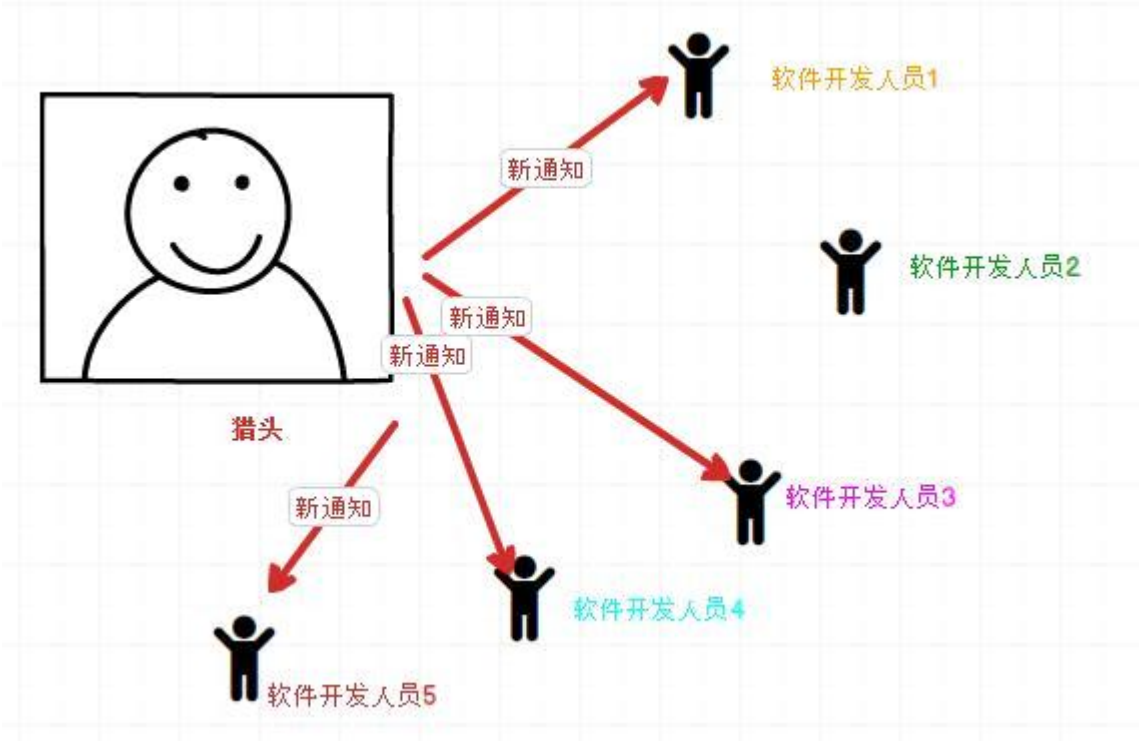
但是也有这样的情况，当软件开发人员没有向猎头提交信息时，猎头在收到消息后不同通知他。



同样接受猎头通知的求职人员同样可以取消通过猎头来寻求工作，比如**软件开发人员2取消了这种求职方式**，那么他就不会再收到猎头的通知。

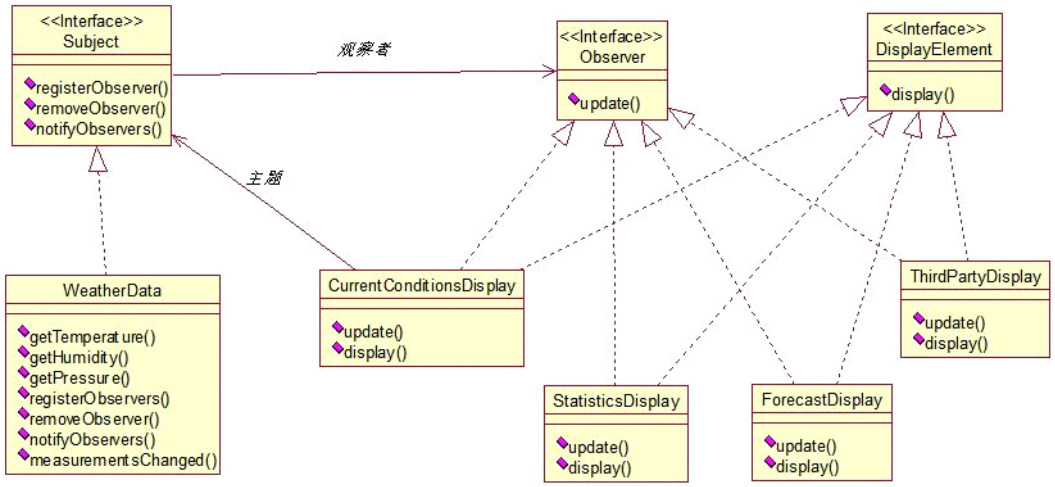


但是**软件开发人员5希望通过这种方式来找工作**，那么就需要去和猎头联系并留下自己的联系方式，那么从此软件开发人员5也可以收到通知了



在明白上面的例子之后，重新回到我们的气象监测应用上。

尝试着画出实现气象站所需要的类，其中包括WeatherData类以及布告栏组件。确定你的图像能够显示出各个部分如何结合起来，以及别的开发人员如何能够实现自己的布告栏组件。**（每个布告栏都应该有一个被命名为“subject（主题）”的指针指向WeatherData对象，为了避免混乱下面没有画出）**



实现气象站：

从建立接口开始

```
public interface Subject{
    public void registerObserver(Observer o);
    public void removeObserver(Observer o);
    public void notifyObservers();
}
public interface Observer{
    public void update(float temp,float humidity,float pressure);
}
public interface DisplayElement{
    public void display();
}
```

在WeatherData中实现主题接口

```
public class WeatherData implements Subject{//WeatherData实现了Subject接口
    private ArrayList observers;//添加一个ArrayList来记录观察者，此ArrayList是在构造
器中建立的
    private float temperature;
    private float humidity;
    private float pressure;

    public WeatherData(){
        observers = new ArrayList();
    }

    public void registerObserver(Observer o){//当注册观察者时，我们只要把它加到
ArrayList后面即可
        observers.add(o);
    }

    public void removeObserver(Observer o){//同样删除时，得到索引然后从ArrayList删除
即可
        int i = observers.indexOf(o);
        if(i >= 0){
            observers.remove(i);
        }
    }

    public void notifyObservers(){//这里我们把状态都告诉每一个观察者，因为观察者都实现了
update(),
        for(int i = 0 ;i < observers.size(); i++){
```

```
            Observer observer = (Observer)observers.get(i);
            observer.update(temperature,humidity,pressure);
        }
    }

    public void measurementsChanged(){//当气象站更新数据时，我们通知观察者
        notifyObservers();
    }

    public void setMeasurements(float temperature,float humidity,float pressure)
{
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }
}
```

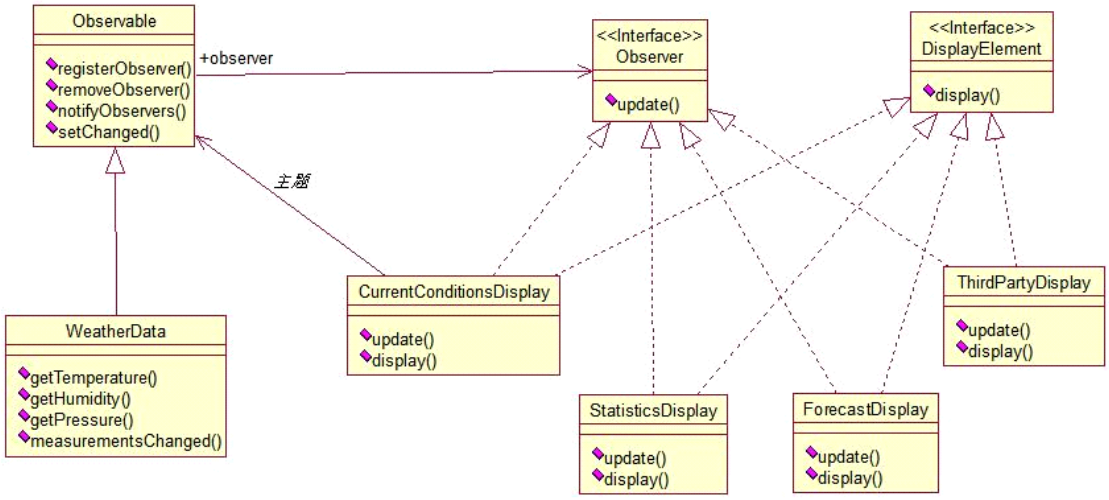
下面只给出了目前状况的布告板：

```
//此布告板实现了Observer接口，所以可以从WeatherData对象中获取改变，同样也实现了
DisplayElement的接口
public class CurrentConditionsDisplay implements Observer,DisplayElement{
    private float temperature;
    private float humidity;
    private Subject weatherData;

    public CurrentConditionsDisplay(Subject weatherData){//构造器需要weatherData对
象 作为注册使用
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

    public void update(float temperature,float humidity,float pressure){
        this.temperature = temperature;
        this.humidity = humidity;//当update()调用时，我们把温度和湿度保存起来，然后调用
display()
        display();//这里之前说的可能不清楚，目前状况只需要显示温度和湿度就可以了，
                //气象检测系统不要求在这个布告板显示气压
    }

    public void display(){
        System.out.println("Current conditions:"+temperature
            +"F degrees and "+ humidity +" % humidity");
    }
}
```

下面讨论使用Java内置的观察者模式来完成气象监测应用

在java.util包中包含最基本的Observer接口与Observable类。甚至可以实现推(push)和拉(pull)方式来传送数据。什么叫做**push**，什么叫做**pull**呢？

还用之前的例子，猎头主动将信息通知给每一位软件开发人员叫做push，那么至于拉（pull）就很直观了，就是软件开发人员主动向猎头联系索取信息。**（每个布告栏都应该有一个被命名为“subject（主题）”的指针指向WeatherData对象，为了避免混乱下面没有画出）**



在这里需要注意的地方：

在Observable里面有一个setChanged()方法，它的作用是在调用notifyObserver()之前先调用setChanged()，如果changed为true那么就进行通知，否则就不通知。（**setChanged()在每次消息有所改变时都要进行修改**）

看一下里面的代码：

```
setChanged(){
    changed= true;//将changed标志设为true
}
notifyObservers(Object arg){
    if(changed){//只有在是true是才通知
        for every observer on the list{
            call update(this.arg);
        }
        changed=false;//通知完观察者之后，把changed标志设回false
    }
}
notifyObservers(){
    notifyObservers(null);
}
```

1.**Observable是一个类**，首先既然它是个类，那么我们需要设计一个类来继承它。但是Java不支持多重继承，这就限制了Observable的服用能力。在再者因为没有Observable接口，所以自己无法创建自己的实现来和Java内置的Observer API搭配使用，也无法将java.util的实现换成另一套做法的实现。

2.Observable将关键的方法保护起来，如果查看Observable API则会发现，setChanged()方法被保护了起来（被定义为protected）。这就意味着：除非你继承自Observable，否则你无法创建Observable实例并组合到你自己的对象中来。**这违背了设计原则“多用组合，少用继承”**

到此我们现在的设计模式已经学习了三个——简单工厂模式、策略模式以及现在的观察者模式。

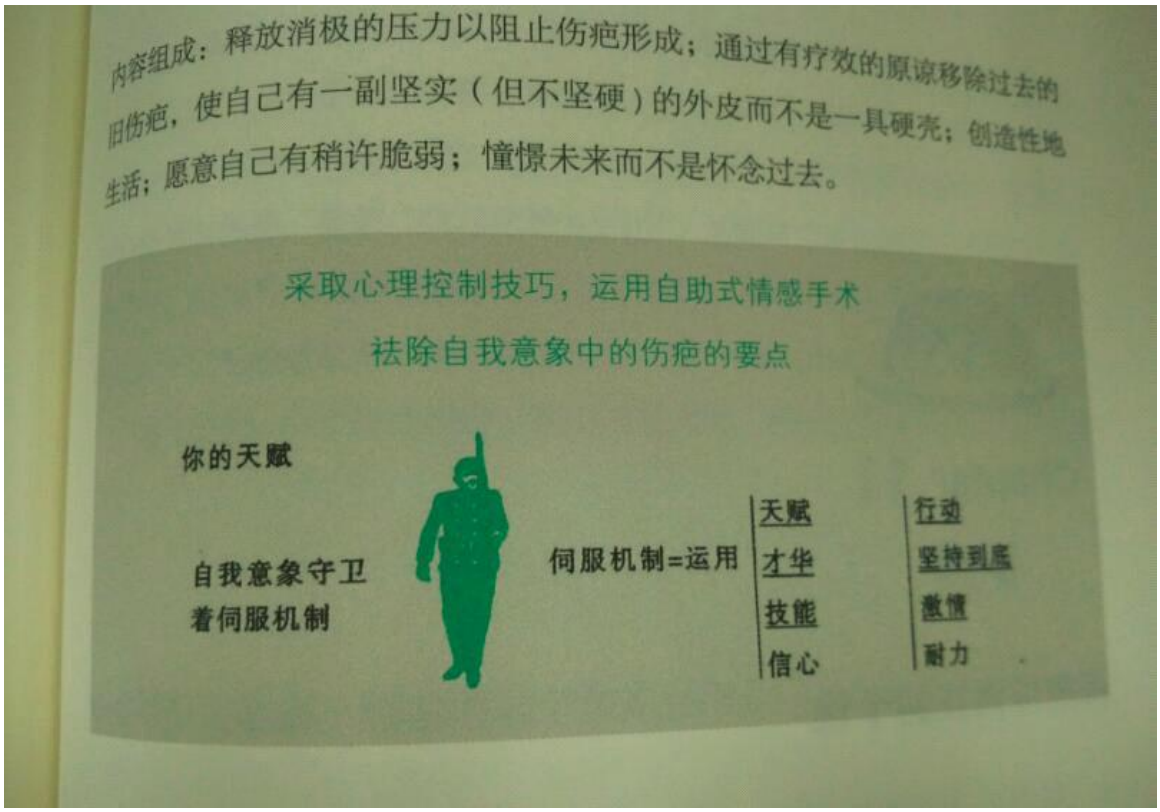
未来的路还很远，加油呐！

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5496286>

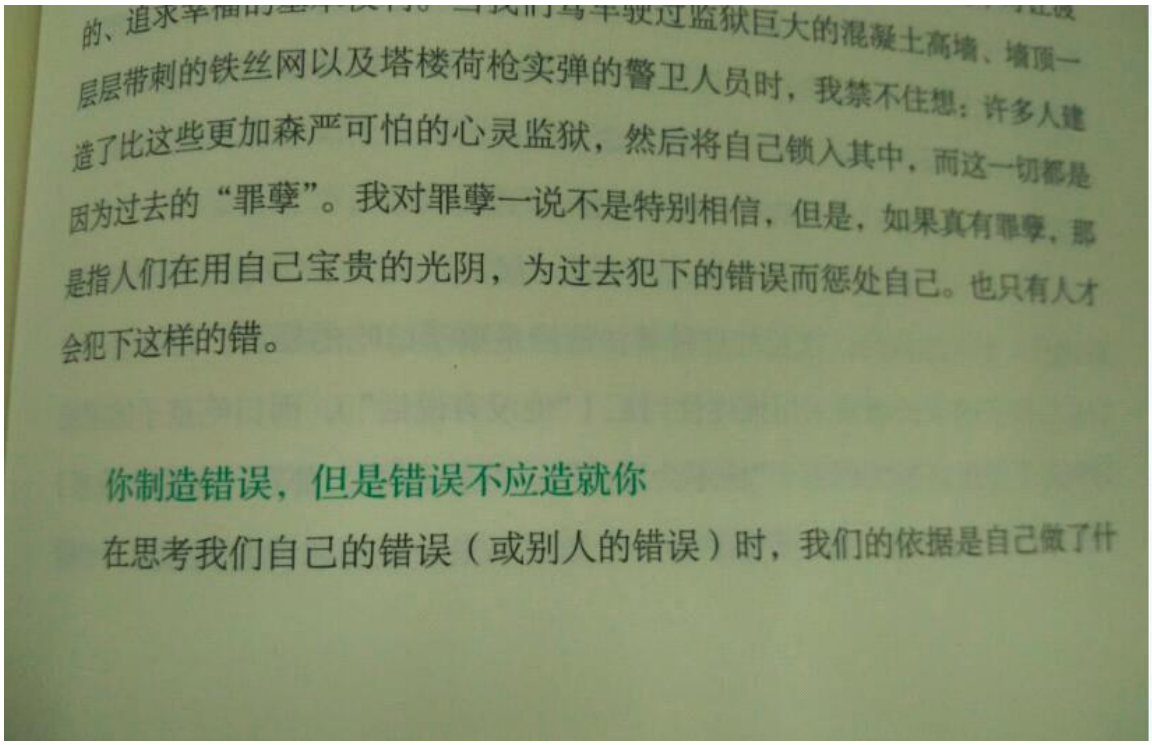
心理控制方法——阅读Notes

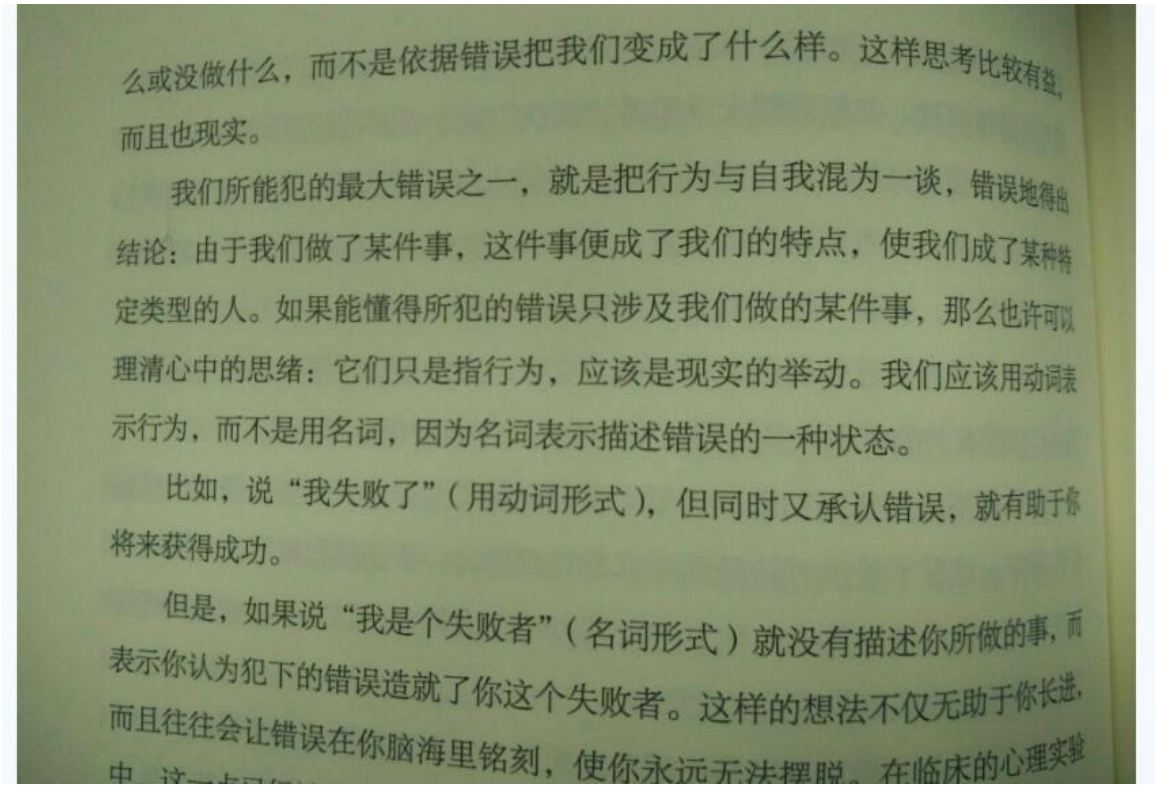
2016年5月15日　　23:34

1.自助式情感手术 祛除自我意象中的伤疤的要点



2. 你制造错误，但是错误不应造就你
你身上的缺点不是你的错





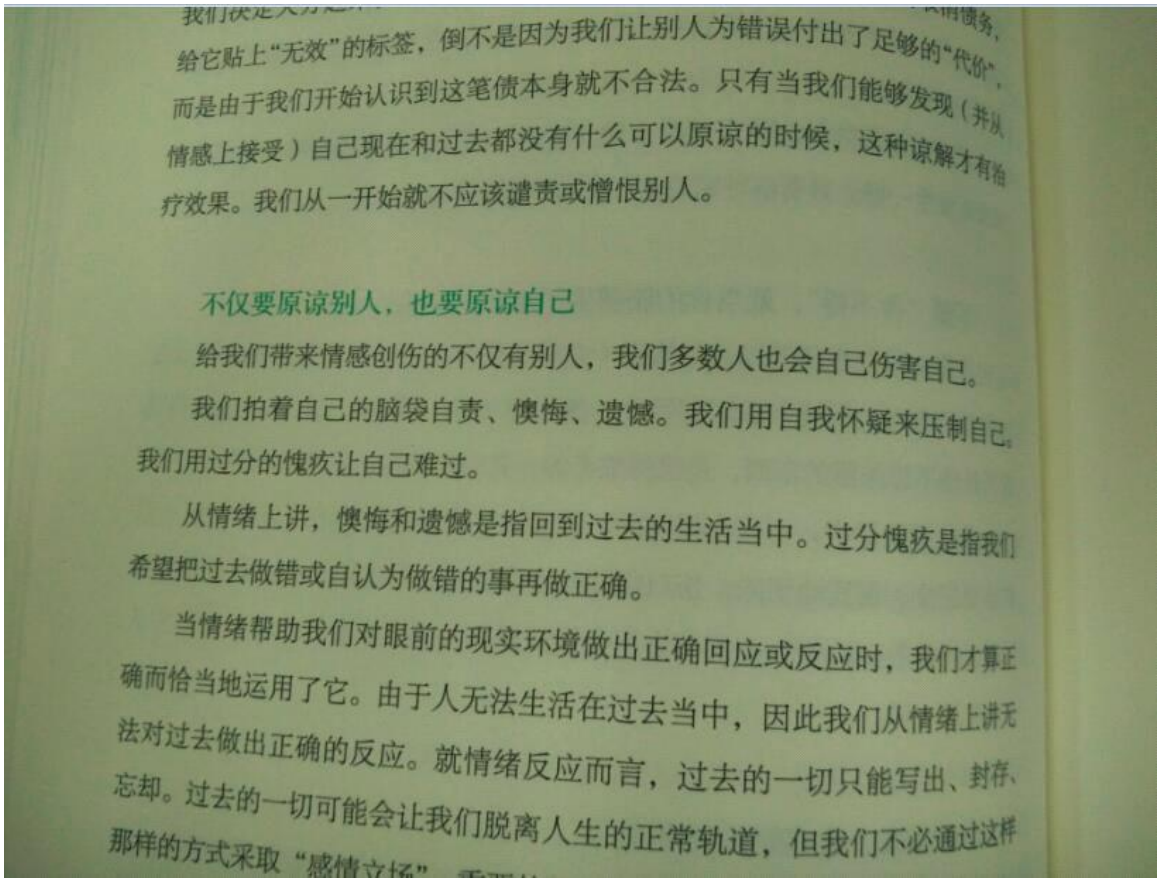
么或没做什么，而不是依据错误把我们变成了什么样。这样思考比较有益，而且也现实。

我们所能犯的最大错误之一，就是把行为与自我混为一谈，错误地得出结论：由于我们做了某件事，这件事便成了我们的特点，使我们成了某种特定类型的人。如果能懂得所犯的错误只涉及我们做的某件事，那么也许可以理清心中的思绪：它们只是指行为，应该是现实的举动。我们应该用动词表示行为，而不是用名词，因为名词表示描述错误的一种状态。

比如，说“我失败了”（用动词形式），但同时又承认错误，就有助于你将来获得成功。

但是，如果说“我是个失败者”（名词形式）就没有描述你所做的事，而表示你认为犯下的错误造就了你这个失败者。这样的想法不仅无助于你长进，而且往往会让错误在你脑海里铭刻，使你永远无法摆脱。在临床的心理实验

3. 不仅要原谅别人，也要原谅自己



给它贴上“无效”的标签，倒不是因为我们让别人为错误付出了足够的“代价”，而是由于我们开始认识到这笔债本身就不合法。只有当我们能够发现（并从情感上接受）自己现在和过去都没有什么可以原谅的时候，这种谅解才有治疗效果。我们从一开始就不应该谴责或憎恨别人。

不仅要原谅别人，也要原谅自己

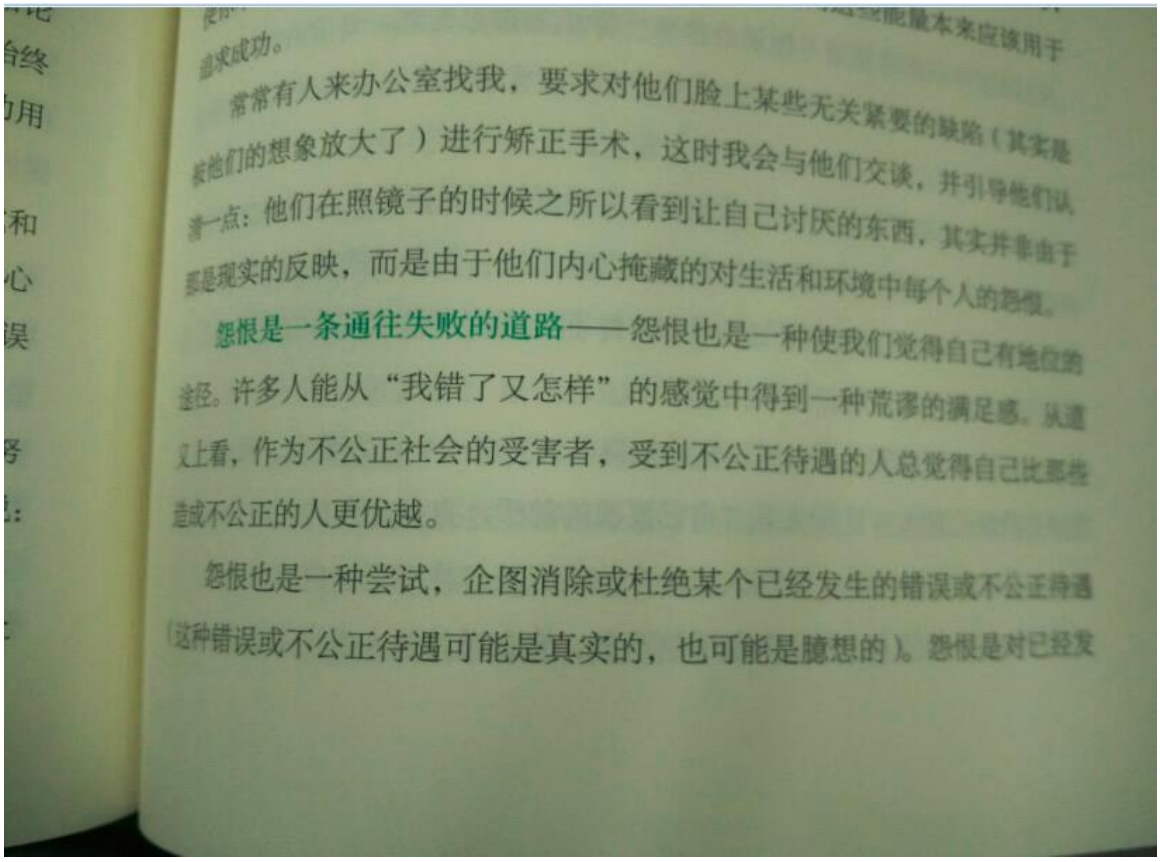
给我们带来情感创伤的不仅有别人，我们多数人也会自己伤害自己。

我们拍着自己的脑袋自责、懊悔、遗憾。我们用自我怀疑来压制自己，我们用过分的愧疚让自己难过。

从情绪上讲，懊悔和遗憾是指回到过去的生活当中。过分愧疚是指我们希望把过去做错或自认为做错的事再做正确。

当情绪帮助我们对眼前的现实环境做出正确回应或反应时，我们才算正确而恰当地运用了它。由于人无法生活在过去当中，因此我们从情绪上讲无法对过去做出正确的反应。就情绪反应而言，过去的一切只能写出、封存、忘却。过去的一切可能会让我们脱离人生的正常轨道，但我们不必通过这样那样的方式采取“感情立场”

4. 怨恨是一条通向失败的道路


常常有人来办公室找我，要求对他们脸上某些无关紧要的缺陷（其实是
他们的想象放大了）进行矫正手术，这时我会与他们交谈，并引导他们认
一点：他们在照镜子的时候之所以看到让自己讨厌的东西，其实并非由于
是现实的反映，而是由于他们内心掩藏的对生活和环境中每个人的怨恨。
怨恨是一条通往失败的道路——怨恨也是一种使我们觉得自己有地位的
径。许多人能从“我错了又怎样”的感觉中得到一种荒谬的满足感。从道
上看，作为不公正社会的受害者，受到不公正待遇的人总觉得自己比那些
或不公正的人更优越。
怨恨也是一种尝试，企图消除或杜绝某个已经发生的错误或不公正待遇
这种错误或不公正待遇可能是真实的，也可能是臆想的）。怨恨是对已经发

5. 注意来自自己的警报

才没有导致恶劣影响。而这次事件据说是因为一名工人泼了一杯咖啡引起的。

威力与危险永远密不可分。你拥有的伺服机制潜在的威力，远比你能相信的要强大得多。你越是了解它、试验它，就越能惊奇地发现它能为你所用。然而，这种可以建设性地、创造性地作为自动成功机制（ASM）的力量，是有一种潜在的破坏力，那就是它也可以转化成自动失败机制（AFM）。我们必须善于控制和调整这种内心的力量，对它那“压力计”上的红色标线时刻保持警惕，防止它“溜进”自动失败机制所在的区间。

消极情绪就是报警信号。灰心、暴怒、势不可挡的焦虑、持续不变的压抑、忌妒和怨恨、懒怠、徒劳地狂热追求、偏执和无礼，当然还有自我排斥，这些都是伺服机制的压力即将进入红色区域的信号。

人体有自己的危险信号和危险标志，医生们将其称为症状或症候。病人们往往把症状视为有害的表现，如一次发烧、疼痛或类似表现都视为“大不好”。实际上，如果病人能搞清这些症状预示什么，并采取改进措施，那这些消极信号对病人来说并不是坏事。症状或症候都是帮助人体保持健康“压力计”或“报警灯”。阑尾炎的疼痛对病人来说似乎很难忍受，但在手之后，它却能确保病人的生存。如果病人感觉不到痛苦，那么就不会采取施把阑尾割掉了。

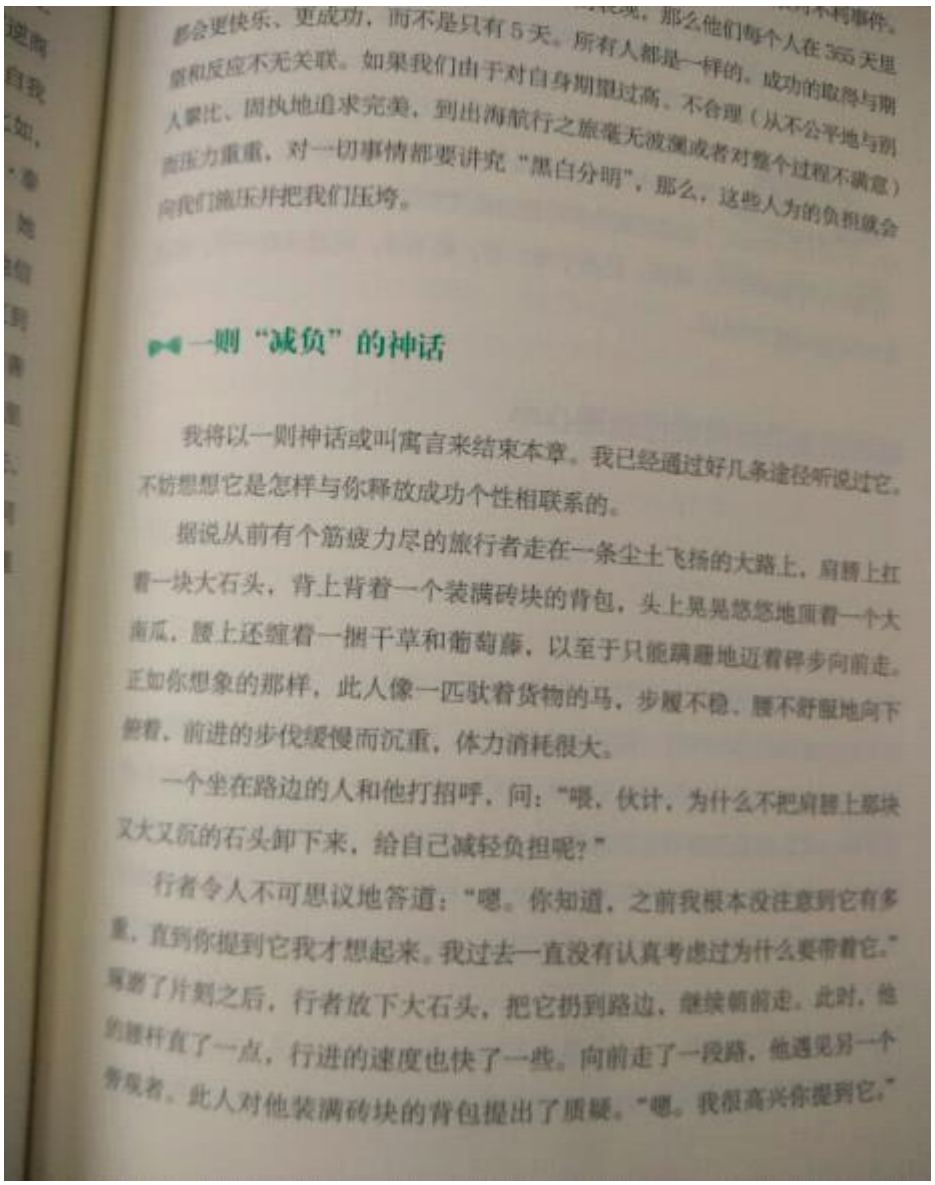
无意中可能唤醒和激活的自动失败机制也有其症状。我们应能发觉自的这些症状，以便对其采取恰当措施。当我们学会将某些性格特点界定为败的“路标”时，这些症状就会自动起到负反馈作用，引导我们取得创造性的成就。不过，

失败的画面

我再次发现，如果商人把这些消极反馈信号（或者我所称的“失败机制”）与组成“失败”（Failure）这个词的各个字母联系起来，就容易记住这些信号。对这些字母进行分解后如下：

F（沮丧、无助、徒劳，Frustration）
U（犹疑，Uncertainty）
A（好斗/错误导向，Aggressiveness）
R（怨恨，Resentment）
I（不安全感，Insecurity）
E（空虚，Emptiness）
L（孤独，Loneliness）

6.行者的减负



都会更快乐、更成功，而不是只有5天。所有人都是一样的。成功的取得与期望和反应不无关联。如果我们由于对自身期望过高、不合理（从不公平地与别人攀比、固执地追求完美，到出海航行之旅毫无波澜或者对整个过程不满意）而压力重重，对一切事情都要讲究“黑白分明”，那么，这些人为的负担就会向我们施压并把我们压垮。

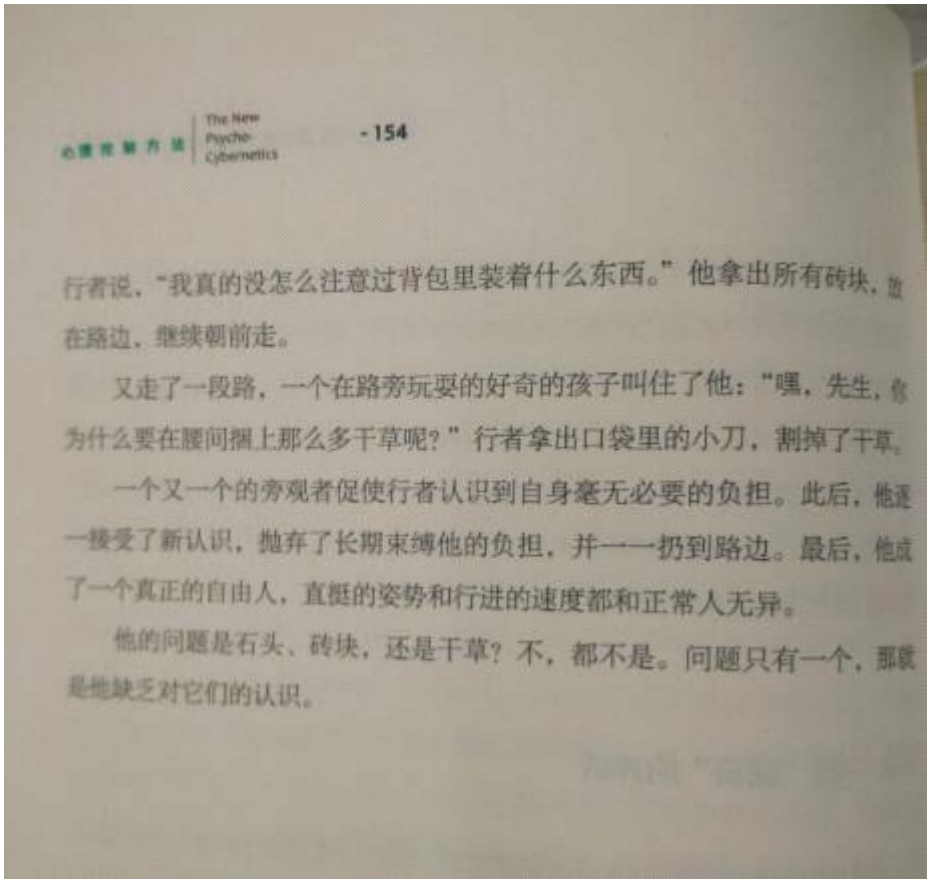
一则“减负”的神话

我将以一则神话或叫寓言来结束本章。我已经通过好几条途径听说过它。不妨想想它是怎样与你释放成功个性相联系的。

据说从前有个筋疲力尽的旅行者走在一条尘土飞扬的大路上，肩膀上扛着一块大石头，背上背着一个装满砖块的背包，头上晃晃悠悠地顶着一个大南瓜，腰上还缠着一捆干草和葡萄藤，以至于只能蹒跚地迈着碎步向前走。正如你想象的那样，此人像一匹驮着货物的马，步履不稳，腰不舒服地向下弯着，前进的步伐缓慢而沉重，体力消耗很大。

一个坐在路边的人和他打招呼，问：“喂，伙计，为什么不把肩膀上那块又大又沉的石头卸下来，给自己减轻负担呢？”

行者令人不可思议地答道：“嗯。你知道，之前我根本没注意到它有多重，直到你提到它我才想起来。我过去一直没有认真考虑过为什么要带着它。”



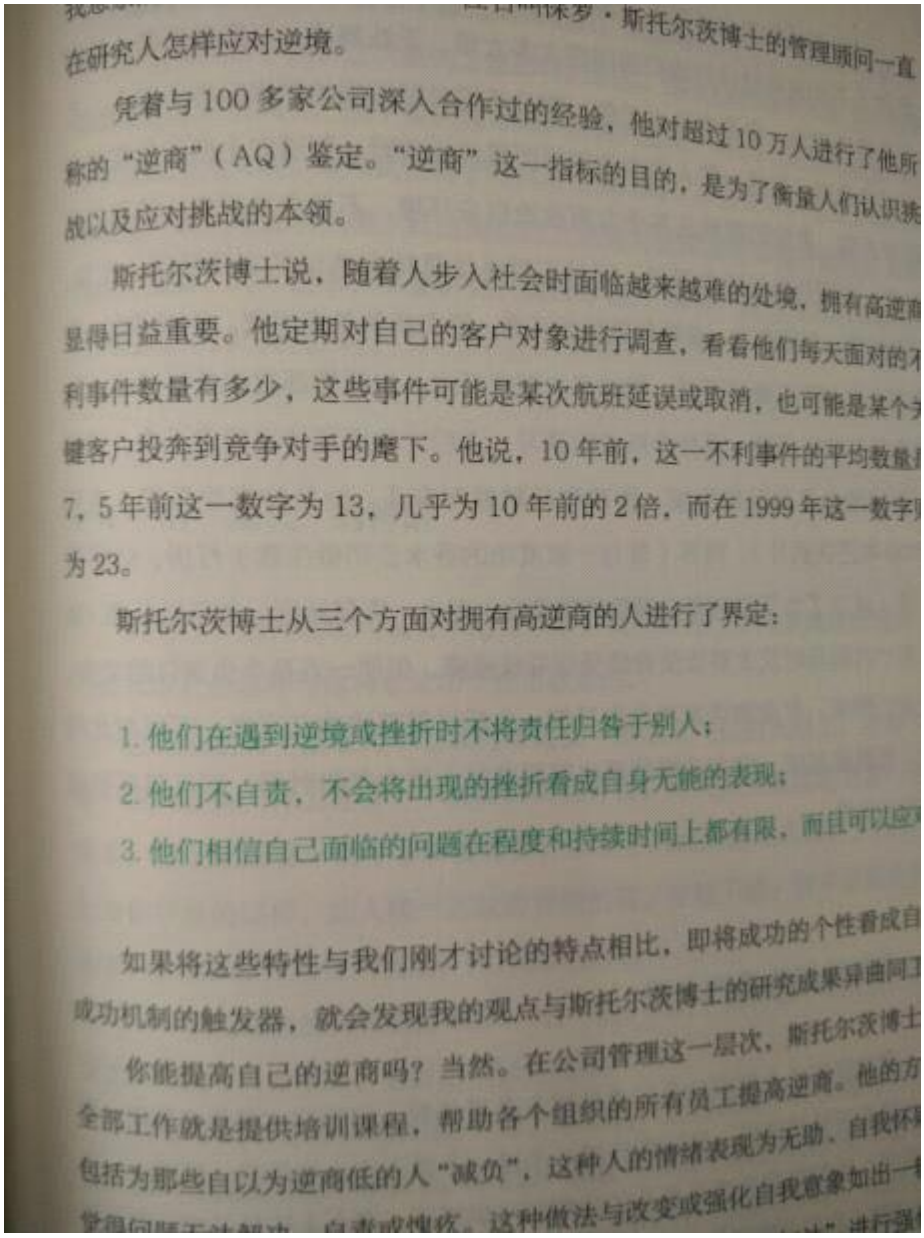
行者说，“我真的没怎么注意过背包里装着什么东西。”他拿出所有砖块，放在路边，继续朝前走。

又走了一段路，一个在路旁玩耍的好奇的孩子叫住了他：“嘿，先生，你为什么要在腰间捆上那么多干草呢？”行者拿出口袋里的小刀，割掉了干草。

一个又一个的旁观者促使行者认识到自身毫无必要的负担。此后，他逐一接受了新认识，抛弃了长期束缚他的负担，并一一扔到路边。最后，他成了一个真正的自由人，直挺的姿势和行进的速度都和正常人无异。

他的问题是石头、砖块，还是干草？不，都不是。问题只有一个，那就是他缺乏对它们的认识。

7.AQ逆境商数



在研究人怎样应对逆境。

凭着与100多家公司深入合作过的经验，他对超过10万人进行了他所称的“逆商”（AQ）鉴定。“逆商”这一指标的目的，是为了衡量人们认识挑战以及应对挑战的本领。

斯托尔茨博士说，随着人步入社会时面临越来越难的处境，拥有高逆商显得日益重要。他定期对自己的客户对象进行调查，看看他们每天面对的不利事件数量有多少，这些事件可能是某次航班延误或取消，也可能是某个关键客户投奔到竞争对手的麾下。他说，10年前，这一不利事件的平均数量是7，5年前这一数字为13，几乎为10年前的2倍，而在1999年这一数字则为23。

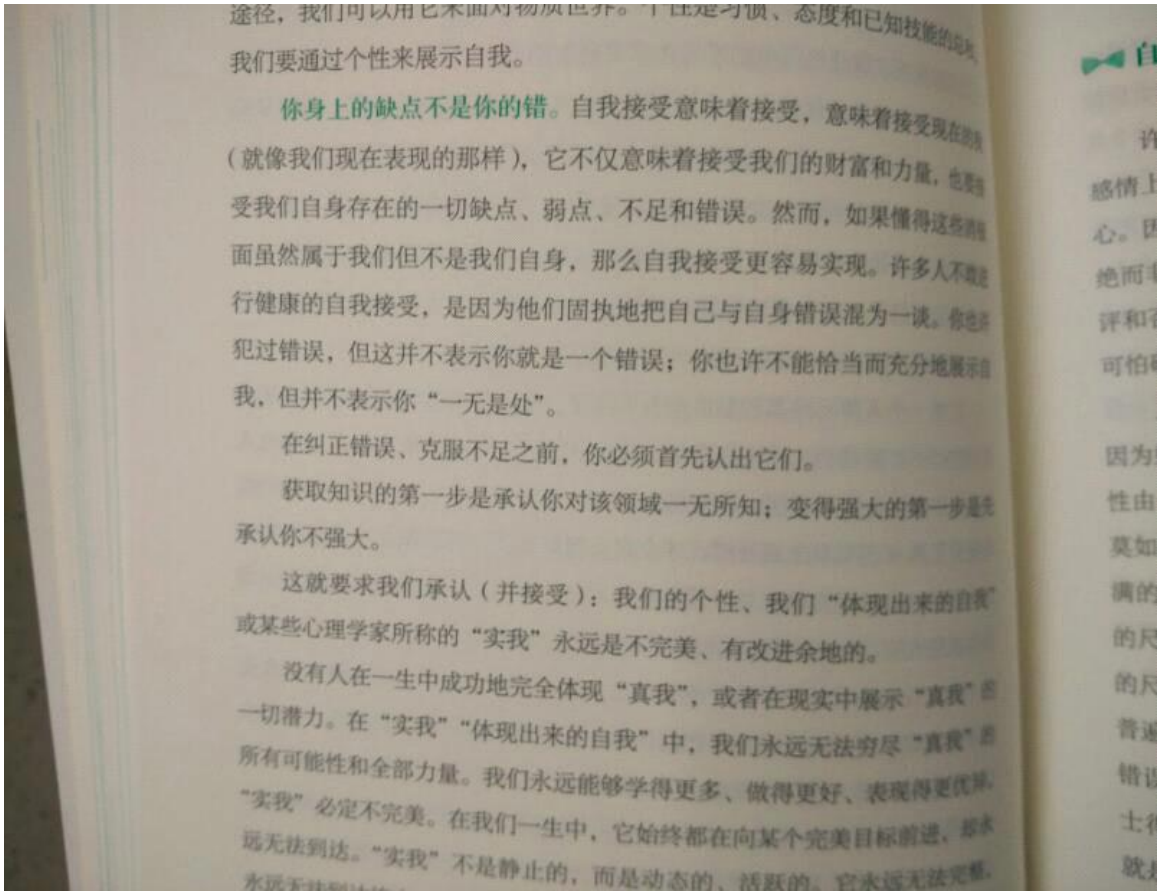
斯托尔茨博士从三个方面对拥有高逆商的人进行了界定：

1. 他们在遇到逆境或挫折时不将责任归咎于别人；
2. 他们不自责，不会将出现的挫折看成自身无能的表现；
3. 他们相信自己面临的问题在程度和持续时间上都有限，而且可以应

如果将这些特性与我们刚才讨论的特点相比，即将成功的个性看成自成功机制的触发器，就会发现我的观点与斯托尔茨博士的研究成果异曲同工

你能提高自己的逆商吗？当然。在公司管理这一层次，斯托尔茨博士全部工作就是提供培训课程，帮助各个组织的所有员工提高逆商。他的方包括为那些自以为逆商低的人“减负”，这种人的情绪表现为无助、自我怀

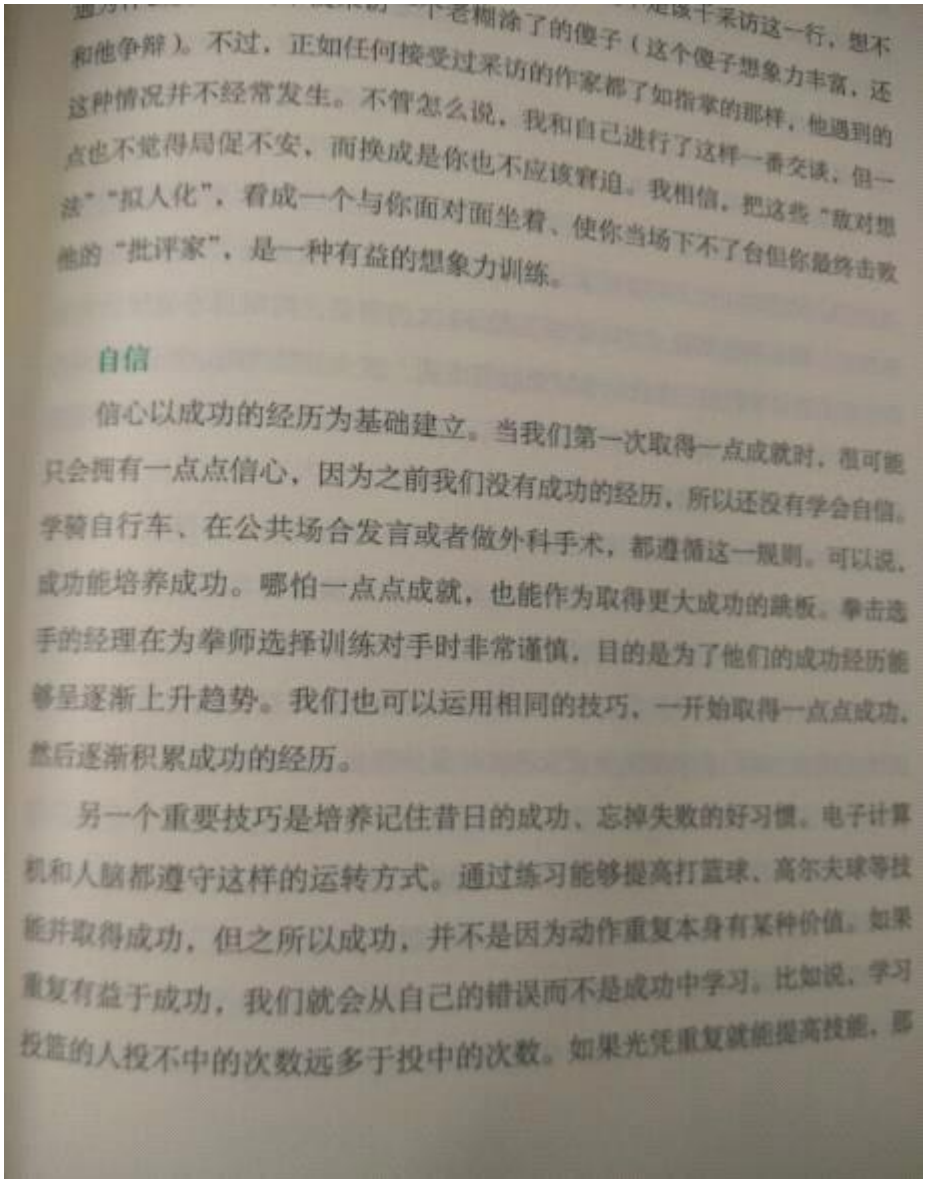
8.获取知识



9. 培养自信，记住昔日的成功，忘掉失败的好习惯

以前的失败都是成功的尝试

做好准备 在成功一次前失败**99**次，同时还不会因为失败而伤害自尊。在任何领域想取得成就，这一条都适用。



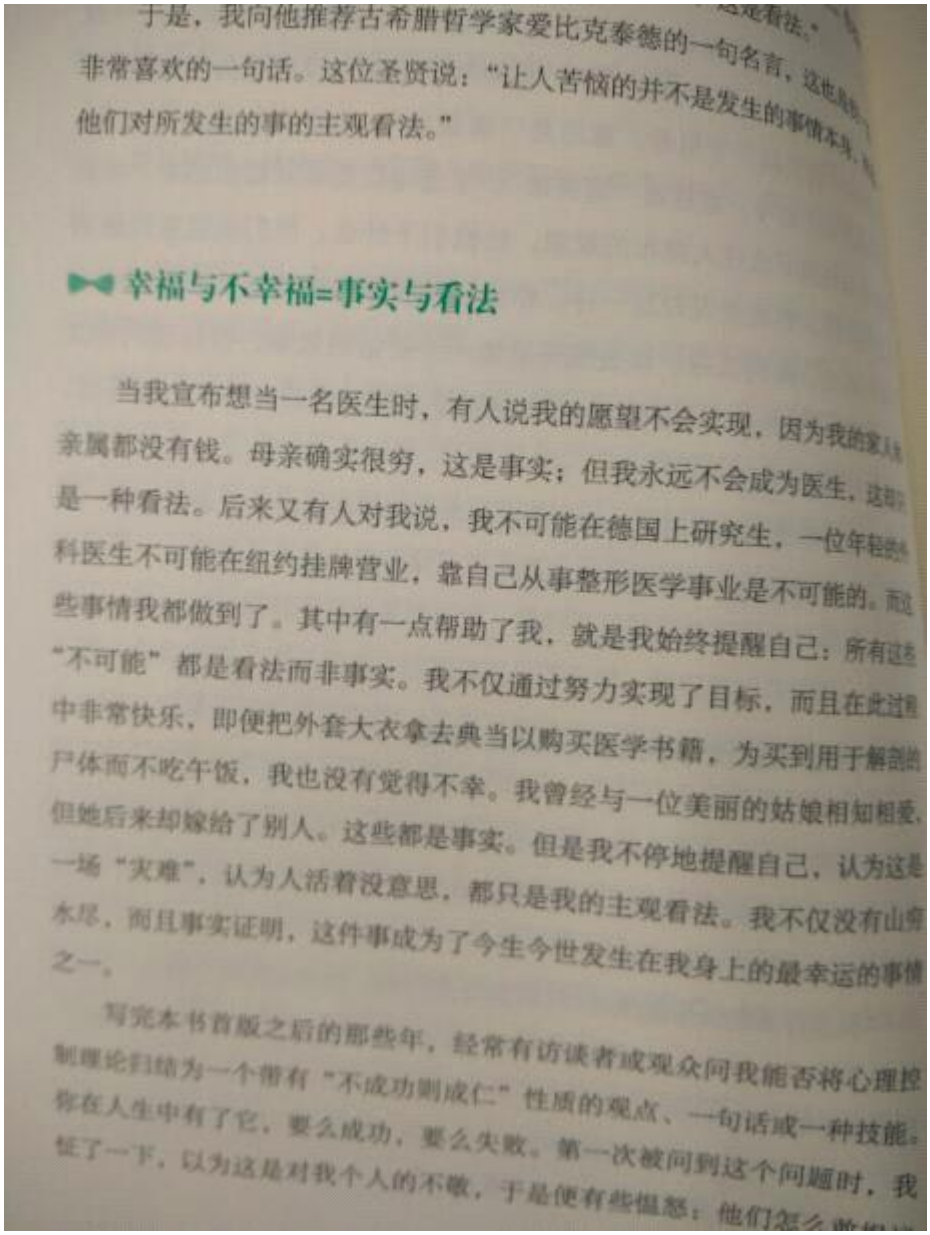
和他争辩）。不过，正如任何接受过采访的作家都了如指掌的那样，他遇到的这种情况并不经常发生。不管怎么说，我和自己进行了这样一番交谈，但一点也不觉得局促不安，而换成是你也不应该胃迫。我相信，把这些"敌对想法""拟人化"，看成一个与你面对面坐着、使你当场下不了台但你最终击败他的"批评家"，是一种有益的想象力训练。

自信

信心以成功的经历为基础建立。当我们第一次取得一点成就时，很可能只会拥有一点点信心，因为之前我们没有成功的经历，所以还没有学会自信。学骑自行车、在公共场合发言或者做外科手术，都遵循这一规则。可以说，成功能培养成功。哪怕一点点成就，也能作为取得更大成功的跳板。拳击选手的经理在为拳师选择训练对手时非常谨慎，目的是为了他们的成功经历能够呈逐渐上升趋势。我们也可以运用相同的技巧，一开始取得一点点成功，然后逐渐积累成功的经历。

另一个重要技巧是培养记住昔日的成功、忘掉失败的好习惯。电子计算机和人脑都遵守这样的运转方式。通过练习能够提高打篮球、高尔夫球等技能并取得成功，但之所以成功，并不是因为动作重复本身有某种价值。如果重复有益于成功，我们就会从自己的错误而不是成功中学习。比如说，学习投篮的人投不中的次数远多于投中的次数。如果光凭重复就能提高技能，那

10.成功的要素



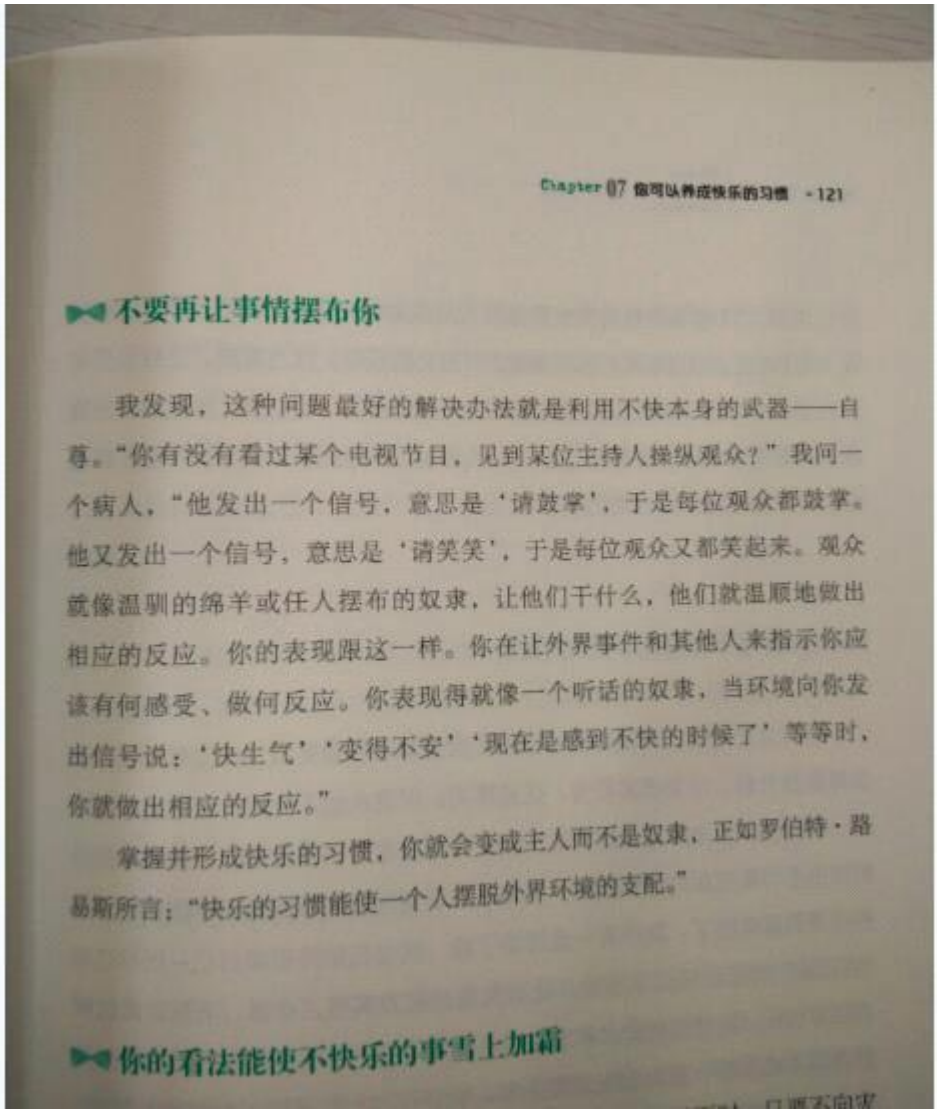
于是，我问他推荐古希腊哲学家爱比克泰德的一句名言，这也是我
非常喜欢的一句话。这位圣贤说："让人苦恼的并不是发生的事情本身，
他们对所发生的事的主观看法。"

幸福与不幸福=事实与看法

当我宣布想当一名医生时，有人说我的愿望不会实现，因为我的家人
亲属都没有钱。母亲确实很穷，这是事实；但我永远不会成为医生，这却
是一种看法。后来又有人对我说，我不可能在德国上研究生，一位年轻的外
科医生不可能在纽约挂牌营业，靠自己从事整形医学事业是不可能的。而这
些事情我都做到了。其中有一点帮助了我，就是我始终提醒自己：所有这些
"不可能"都是看法而非事实。我不仅通过努力实现了目标，而且在此过程
中非常快乐，即便把外套大衣拿去典当以购买医学书籍，为买到用于解剖的
尸体而不吃午饭，我也没有觉得不幸。我曾经与一位美丽的姑娘相知相爱，
但她后来却嫁给了别人。这些都是事实。但是我不停地提醒自己，认为这是
一场"灾难"，认为人活着没意思，都只是我的主观看法。我不仅没有山穷
水尽，而且事实证明，这件事成为了今生今世发生在我身上的最幸运的事情
之一。

写完本书首版之后的那些年，经常有访谈者或观众问我能否将心理控
制理论归结为一个带有"不成功则成仁"性质的观点、一句话或一种技能。
你在人生中有了它，要么成功，要么失败。第一次被问到这个问题时，我
怔了一下，以为这是对我个人的不敬，于是便有些恼怒：他们怎么敢

11.打断有害的思维模式



Chapter 07 你可以养成快乐的习惯 · 121

不要再让事情摆布你

我发现，这种问题最好的解决办法就是利用不快本身的武器——自尊。“你有没有看过某个电视节目，见到某位主持人操纵观众？”我问一个病人，“他发出一个信号，意思是‘请鼓掌’，于是每位观众都鼓掌。他又发出一个信号，意思是‘请笑笑’，于是每位观众又都笑起来。观众就像温驯的绵羊或任人摆布的奴隶，让他们干什么，他们就温顺地做出相应的反应。你的表现跟这一样。你在让外界事件和其他人来指示你应该有何感受、做何反应。你表现得就像一个听话的奴隶，当环境向你发出信号说：‘快生气’‘变得不安’‘现在是感到不快的时候了’等等时，你就做出相应的反应。”

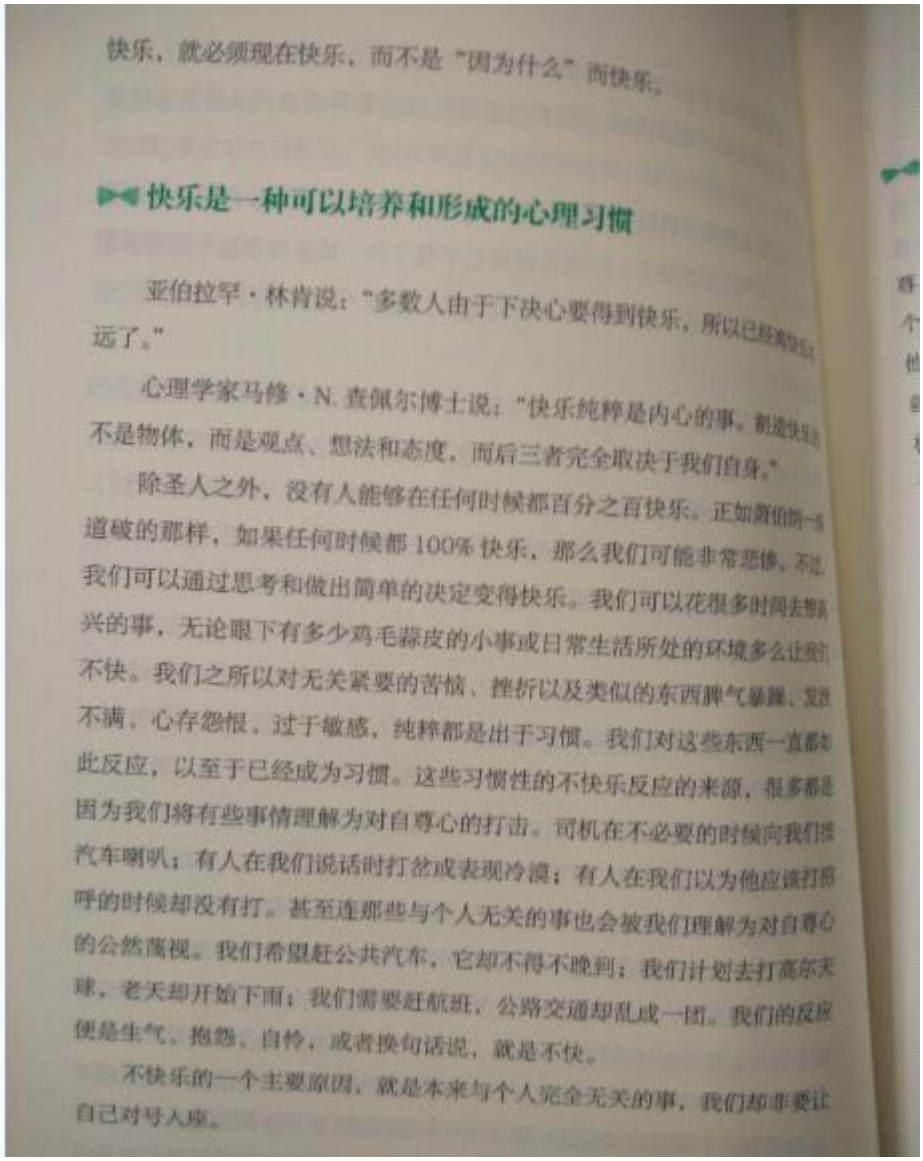
掌握并形成快乐的习惯，你就会变成主人而不是奴隶，正如罗伯特·路易斯所言：“快乐的习惯能使一个人摆脱外界环境的支配。”

你的看法能使不快乐的事雪上加霜

12.准确而冷静地，从而最终自动地将事实与假定，事实与看法，真实情况与方法的障碍分离开，使我们的行动和反应牢固地建立在真相而不是自己或者他人的看法的基础之上

13.所有不可能都是看法而不是事实

14. 掌握并形成快乐的习惯，你就会变成主人而不是奴隶。
快乐的习惯能使一个人摆脱外界环境的支配


快乐，就必须现在快乐，而不是“因为什么”而快乐。

快乐是一种可以培养和形成的心理习惯

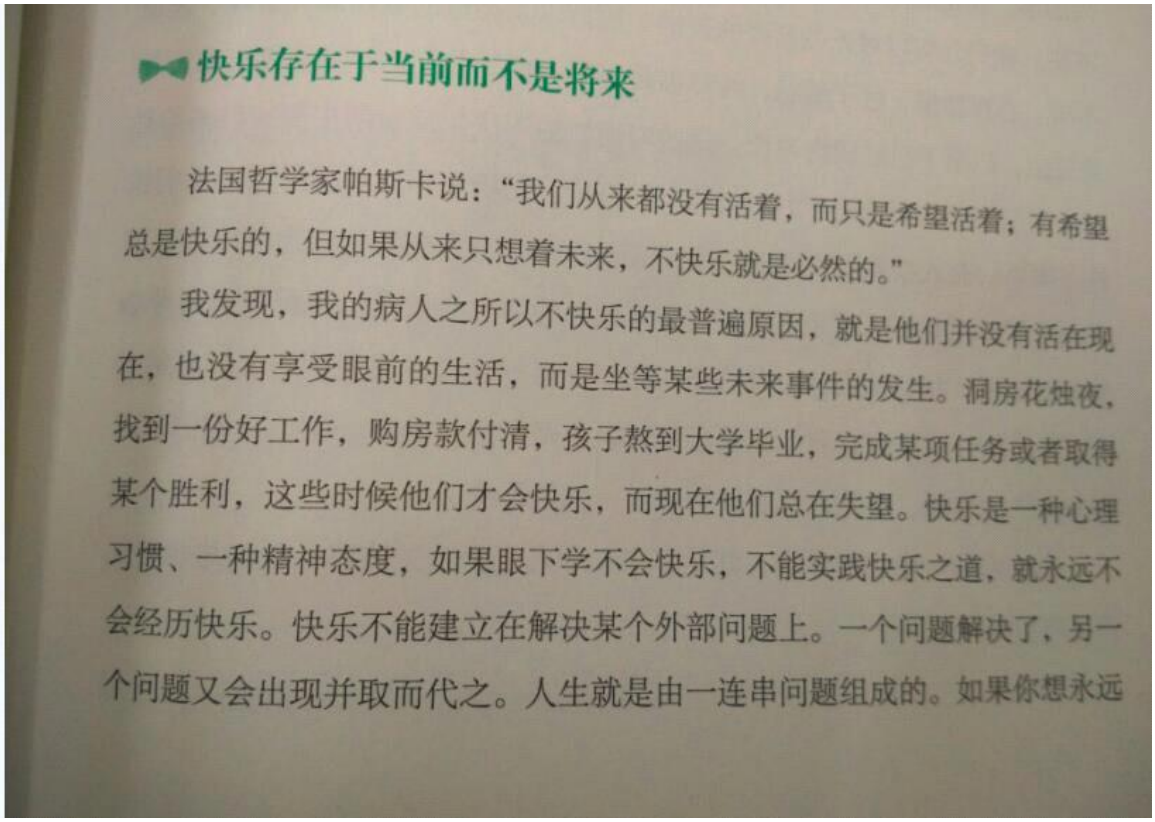
亚伯拉罕·林肯说：“多数人由于下决心要得到快乐，所以已经离快乐远了。”

心理学家马修·N. 查佩尔博士说：“快乐纯粹是内心的事。制造快乐不是物体，而是观点、想法和态度，而后三者完全取决于我们自身。”

除圣人之外，没有人能够在任何时候都百分之百快乐。正如普伯纳一道破的那样，如果任何时候都 100% 快乐，那么我们可能非常悲惨。不过，我们可以通过思考和做出简单的决定变得快乐。我们可以花很多时间去想高兴的事，无论眼下有多少鸡毛蒜皮的小事或日常生活所处的环境多么让我们不快。我们之所以对无关紧要的苦恼、挫折以及类似的东西脾气暴躁、发泄不满、心存怨恨、过于敏感，纯粹都是出于习惯。我们对这些东西一直都如此反应，以至于已经成为习惯。这些习惯性的不快乐反应的来源，很多都是因为我们将有些事情理解为对自尊心的打击。司机在不必要的时候向我们按汽车喇叭；有人在我们说话时打岔或表现冷漠；有人在我们以为他应该打招呼的时候却没有打。甚至连那些与个人无关的事也会被我们理解为对自尊心的公然蔑视。我们希望赶公共汽车，它却不得不晚到；我们计划去打高尔夫球，老天却开始下雨；我们需要赶航班，公路交通却乱成一团。我们的反应便是生气、抱怨、自怜，或者换句话说，就是不快。

不快乐的一个主要原因，就是本来与个人完全无关的事，我们却非要让自己对号入座。

15. 不快乐的一个主要原因，就是本来与个人完全无关的事，我们却非要让自己对号入座



快乐存在于当前而不是将来

法国哲学家帕斯卡说："我们从来都没有活着，而只是希望活着；有希望总是快乐的，但如果从来只想着未来，不快乐就是必然的。"

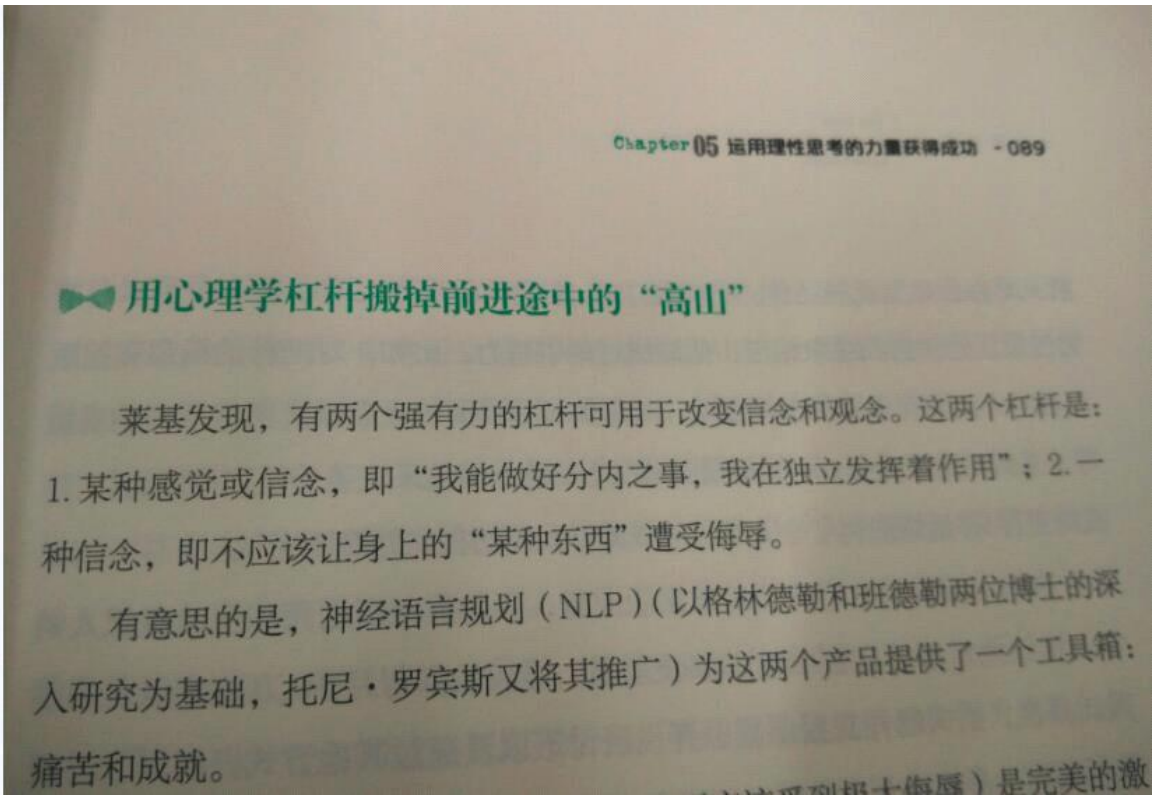
我发现，我的病人之所以不快乐的最普遍原因，就是他们并没有活在现在，也没有享受眼前的生活，而是坐等某些未来事件的发生。洞房花烛夜，找到一份好工作，购房款付清，孩子熬到大学毕业，完成某项任务或者取得某个胜利，这些时候他们才会快乐，而现在他们总在失望。快乐是一种心理习惯、一种精神态度，如果眼下学不会快乐，不能实践快乐之道，就永远不会经历快乐。快乐不能建立在解决某个外部问题上。一个问题解决了，另一个问题又会出现并取而代之。人生就是由一连串问题组成的。如果你想永远

16. 用心理学杠杆搬掉前进途中的高山

有了支点，有了杠杆，跟着感觉走，搬掉眼前的高山

某种感觉或信念

一种信念，即不应该让身上的"某种东西"遭受侮辱



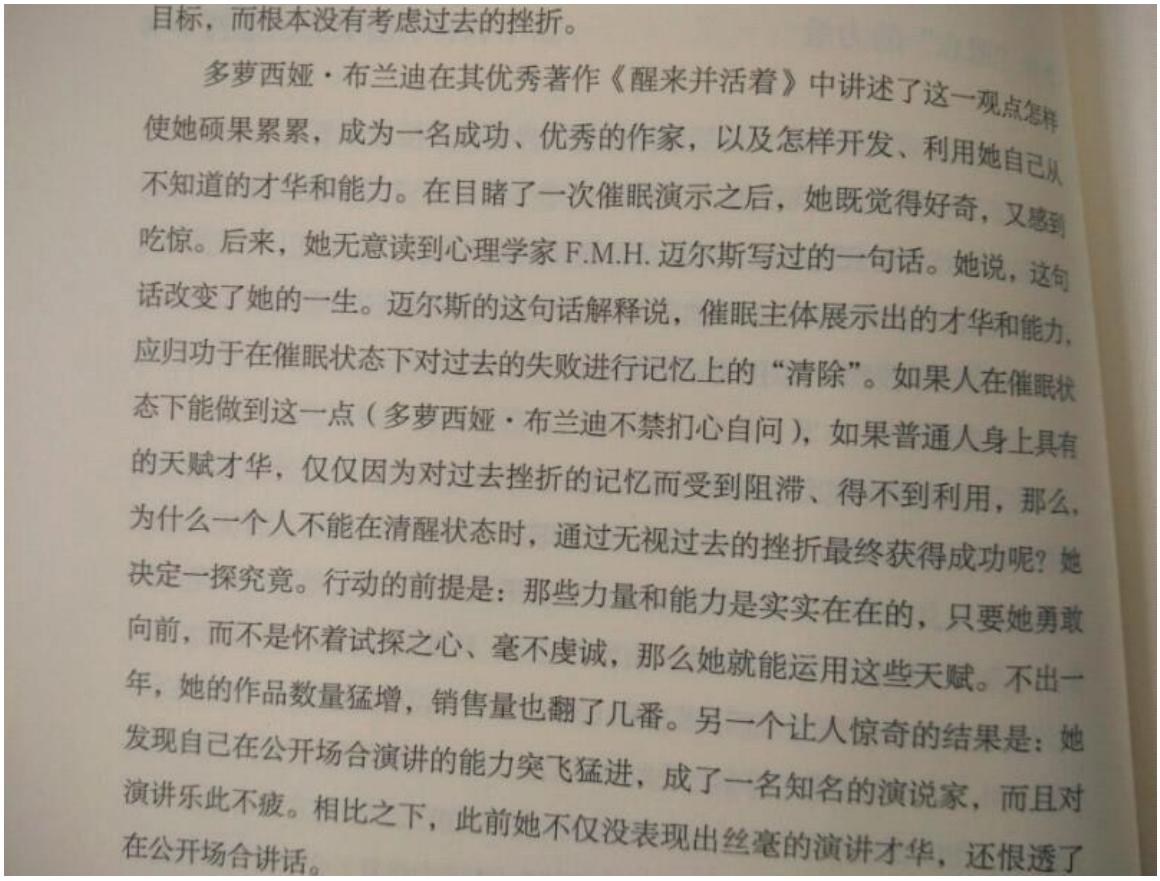
用心理学杠杆搬掉前进途中的"高山"

莱基发现，有两个强有力的杠杆可用于改变信念和观念。这两个杠杆是：1. 某种感觉或信念，即"我能做好分内之事，我在独立发挥着作用"；2. 一种信念，即不应该让身上的"某种东西"遭受侮辱。

有意思的是，神经语言规划（NLP）（以格林德勒和班德勒两位博士的深入研究为基础，托尼·罗宾斯又将其推广）为这两个产品提供了一个工具箱：

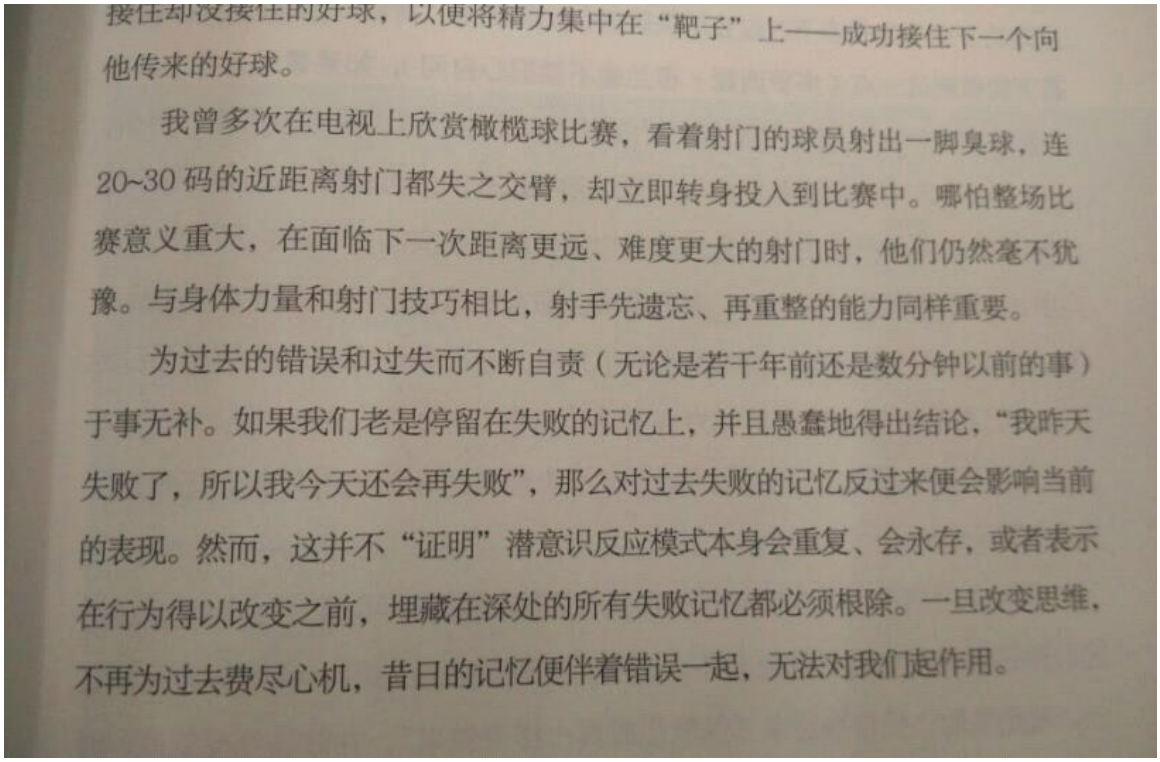
痛苦和成就。

17. 在催眠状态下对过去的失败进行记忆上的清除


目标，而根本没有考虑过去的挫折。

多萝西娅·布兰迪在其优秀著作《醒来并活着》中讲述了这一观点怎样使她硕果累累，成为一名成功、优秀的作家，以及怎样开发、利用她自己从不知道的才华和能力。在目睹了一次催眠演示之后，她既觉得好奇，又感到吃惊。后来，她无意读到心理学家 F.M.H. 迈尔斯写过的一句话。她说，这句话改变了她的一生。迈尔斯的这句话解释说，催眠主体展示出的才华和能力，应归功于在催眠状态下对过去的失败进行记忆上的“清除”。如果人在催眠状态下能做到这一点（多萝西娅·布兰迪不禁扪心自问），如果普通人身上具有的天赋才华，仅仅因为对过去挫折的记忆而受到阻滞、得不到利用，那么，为什么一个人不能在清醒状态时，通过无视过去的挫折最终获得成功呢？她决定一探究竟。行动的前提是：那些力量和能力是实实在在的，只要她勇敢向前，而不是怀着试探之心、毫不虔诚，那么她就能运用这些天赋。不出一年，她的作品数量猛增，销售量也翻了几番。另一个让人惊奇的结果是：她发现自己在公开场合演讲的能力突飞猛进，成了一名知名的演说家，而且对演讲乐此不疲。相比之下，此前她不仅没表现出丝毫的演讲才华，还恨透了在公开场合讲话。

18. 遗忘的力量，不应该为过去的错误和过失而不断的自责。


接住却没接住的好球，以便将精力集中在“靶子”上——成功接住下一个向他传来的好球。

我曾多次在电视上欣赏橄榄球比赛，看着射门的球员射出一脚臭球，连 20~30 码的近距离射门都失之交臂，却立即转身投入到比赛中。哪怕整场比赛意义重大，在面临下一次距离更远、难度更大的射门时，他们仍然毫不犹豫。与身体力量和射门技巧相比，射手先遗忘、再重整的能力同样重要。

为过去的错误和过失而不断自责（无论是若干年前还是数分钟以前的事）于事无补。如果我们老是停留在失败的记忆上，并且愚蠢地得出结论，“我昨天失败了，所以我今天还会再失败”，那么对过去失败的记忆反过来便会影响当前的表现。然而，这并不“证明”潜意识反应模式本身会重复、会永存，或者表示在行为得以改变之前，埋藏在深处的所有失败记忆都必须根除。一旦改变思维，不再为过去费尽心机，昔日的记忆便伴着错误一起，无法对我们起作用。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5493811>

使用PL/SQL编写存储过程访问数据库

2016年5月15日　　23:34

一、实验目的

熟悉使用存储过程来进行数据库应用程序的设计。

二、实验内容

对学生-课程数据库，编写存储过程，完成下面功能：

1.统计离散数学的成绩分布情况，即按照各分数段统计人数；

2.统计任意一门课的平均成绩；

3.将学生选课成绩从百分制改为等级制（即A、B、C、D、E）。

要求：提交源程序并表示必要的注释。保证程序能正确编译和运行，认真填写实验报告。

三、实验步骤

实验之前，已经建立数据库，有student，course和sc三张基本表。

（一）统计离散数学的成绩分布情况，即按照各分数段统计人数。

1.建立表 Rank，其中第一列 division 显示成绩分段划分，第二列 number 显示的是成绩在该分数 段的学生人数。

```
CREATE TABLE Rank(
 division CHAR(20),
  number      INT);
```

2.编写存储过程。

```
create procedure statistic_mark(@name char(50))
AS
DECLARE
@less60  INT,
@b60a70  INT,
@b70a80  INT,
@b80a90  INT,
@more90  INT,
```

```
@curcno VARCHAR(8);
begin
select @curcno = cno
from Course
where cname =@name;

if(@curcno is null)
raiserror('课程号为空',16,1);
else
SELECT  @less60=count(*)
FROM SC
WHERE cno =@curcno AND grade <60;

SELECT @b60a70=count(*)
FROM SC
WHERE cno =@curcno AND grade >=60 AND grade<70;

SELECT @b70a80=count(*)
FROM SC
WHERE cno =@curcno AND grade >=70 AND grade<80;

SELECT @b80a90=count(*)
FROM SC
WHERE cno =@curcno AND grade >=80 AND grade<90;

SELECT @more90=count(*)
FROM SC
WHERE cno =@curcno AND grade >=90 ;
INSERT  INTO  RANK VALUES('[0,60)',@less60);
INSERT  INTO  RANK VALUES('[60,70)',@b60a70);
INSERT  INTO  RANK VALUES('[70,80)',@b70a80);
INSERT  INTO  RANK VALUES('[80,90)',@b80a90);
INSERT  INTO  RANK VALUES('[90,100)',@more90);
END;
```

3.执行存储过程

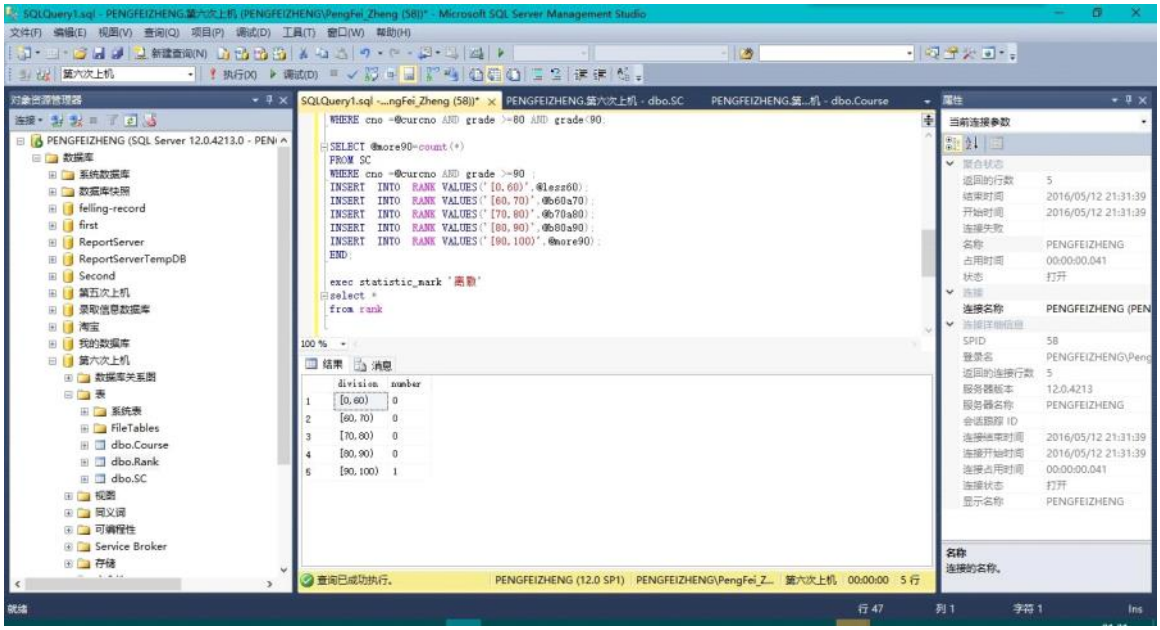
编写好存储过程statistic_mark之后，在“查询分析器”中选择菜单中的“单事务执行”命令，这样系统就创建好了存储过程

然后使用PERFORM调用该过程，在表rank中查看执行的结果。

exec statistic_mark '离散'

select *

from rank



(二)统计任意一门课程的平均成绩

1.创建存储过程

（1）创建需要的表结构。

根据实验要求，要统计任意一门课程的平均成绩，因此需要建立表avggrade，其中第一列cname 显示被统计的课程名称，第二列avg显示选修了该课程的学生的平均成绩。

create table avggrade（

cname char(50),

avg numeric(10,6)）；

（1）编写存储过程

create or replace procedure collect_avg（）

as

declare　　//声明变量

curname　char(50);

curno　　char(4);

```
curavgg   char(10,6);
cursor mycursor for              //声明游标mycursor查询课程号和课程名称
select cno,cname from course;
begin
  open mycursor;                 //打开游标
  IF  mycursor%ISOPEN THEN       //条件控制，游标打开时进行以下处理
        LOOP                     //循环控制
    FETCH  mycursor INTO curcno,curname;  //游标推进一行取结果送变量
    EXIT WHEN(mycursor%NOTFOUND);  //如果没有返回值，则退出循环
 SELECT AVG(grade)INTO curavgg FROM SC   //求该课程的平均值送变量
 WHERE cno = curcno;
      //向avggrade //表中插入记录，显示课程名称和平均成绩
 INSERE INTO avggrade VALUES(curname,curavgg);
END LOOP;              //结束循环控制
END IF;                //结束条件控制
CLOSE mycursor;
END;
```

2.执行存储过程

首先执行编写好的存储过程collect_avg,然后在表avggrade中查看执行结果。

```
PERFORM PROCEDURE collect_avg();
SELECT * FROM avggrade;
```

(三)在表SC中将学生选课成绩从百分制改为等级制

1.创建存储过程

根据实验要求，本实验中存储过程的执行不需要在客户端返回结果，因此不需要建立相应的表结构来存放存储过程的执行结果。直接编写存储过程。

```
create or replace procedure change_critical()
AS
DECLARE
  chgrade CHAR(1);
  currecord record;
BEGIN
ALTER TABLE SC ADD COLUMN(newgrade CHAR(1));
```

```
FOR currecord IN SELECT*FROM SC LOOP
IF currecord.grade<60 then
  chgrade ='E';
ELSIF  currecord.grade<70 then
  chgrade ='D';
ELSIF  currecord.grade<80 then
  chgrade ='C;
ELSIF  currecord.grade<90 then
  chgrade ='B';
ELSE
  chgrade ='A';
END IF;
UPDATE SC SET newgrade =chgrade
WHERE sno =currecord.sno AND cno=currecord.cno;
END LOOP;

ALTER TABLE SC DROP COLUMN grade;
ALTER TABLE SC RENAME newgrade TO grade;
END;
```

2执行存储过程

```
PERFORM PROCEDURE change_critical();
```

(四)删除存储过程

存储过程一旦建立，则将被保存在数据库中，便于用户随时，反复地调用和执行。如果不再需要该存储过程，可以将其删除。

删除存储过程statistic_mark。

```
 DROP PROCEDURE statistic_mark;
```

(1)删除存储过程collect_avg,

```
DROP PROCEDURE collect_avg,
```

(2)删除存储过程 change_critical;

```
DROP PROCEDURE change_critical;
```

四、实验总结

推荐阅读博客：

http://www.cnblogs.com/knowledgesea/archive/2013/01/02/2841588.html

http://www.cnblogs.com/hoojo/archive/2011/07/19/2110862.html

存储过程Procedure是一组为了完成特定功能的SQL语句集合，经编译后存储在数据库中，用户通过指定存储过程的名称并给出参数来执行。

存储过程中可以包含逻辑控制语句和数据操纵语句，它可以接受参数、输出参数、返回单个或多个结果集以及返回值。

由于存储过程在创建时即在数据库服务器上进行了编译并存储在数据库中，所以存储过程运行要比单个的SQL语句块要快。同时由于在调用时只需用提供存储过程名和必要的参数信息，所以在一定程度上也可以减少网络流量、简单网络负担。

存储过程的优点

1.存储过程允许标准组件式编程；

2.存储过程能够实现较快的执行速度；

3.存储过程减轻网络流量；

4.存储过程可被作为一种安全机制来充分利用。

常用系统存储过程有：

exec sp_databases; --查看数据库

exec sp_tables;　　--查看表

exec sp_columns student;--查看列

exec sp_helpIndex student;--查看索引

exec sp_helpConstraint student;--约束

exec sp_stored_procedures;

exec sp_helptext 'sp_stored_procedures';--查看存储过程创建、定义语句

exec sp_rename student, stuInfo;--修改表、索引、列的名称

exec sp_renamedb myTempDB, myDB;--更改数据库名称

exec sp_defaultdb 'master', 'myDB';--更改登录名的默认数据库

exec sp_helpdb;--数据库帮助，查询数据库信息

exec sp_helpdb master;

创建存储过程的参数：

1.procedure_name ：存储过程的名称，在前面加#为局部临时存储过程，加##为全局临时存储过程。

2. number：是可选的整数，用来对同名的过程分组，以便用一条 DROP PROCEDURE 语句即可将同组的过程一起除去。例如，名为 orders 的应用程序使用的过程可以命名为 orderproc;1、orderproc;2 等。DROP PROCEDURE orderproc 语句将除去整个组。如果名称中包含定界标识符，则数字不应包含在标识符中，只应在 procedure_name 前后使用适当的定界符。

3.@parameter： 存储过程的参数。可以有一个或多个。用户必须在执行过程时提供每个所声明

参数的值（除非定义了该参数的默认值）。存储过程最多可以有 2.100 个参数。

使用 @ 符号作为第一个字符来指定参数名称。参数名称必须符合标识符的规则。每个过程的参数仅用于该过程本身；相同的参数名称可以用在其它过程中。默认情况下，参数只能代替常量，而不能用于代替表名、列名或其它数据库对象的名称。有关更多信息，请参见 EXECUTE。

4.data_type：参数的数据类型。所有数据类型（包括 text、ntext 和 image）均可以用作存储过程的参数。不过，cursor 数据类型只能用于 OUTPUT 参数。如果指定的数据类型为 cursor，也必须同时指定 VARYING 和 OUTPUT 关键字。有关 SQL Server 提供的数据类型及其语法的更多信息，请参见数据类型。

说明 对于可以是 cursor 数据类型的输出参数，没有最大数目的限制。

5.VARYING： 指定作为输出参数支持的结果集（由存储过程动态构造，内容可以变化）。仅适用于游标参数。

6.default： 参数的默认值。如果定义了默认值，不必指定该参数的值即可执行过程。默认值必须是常量或 NULL。如果过程将对该参数使用 LIKE 关键字，那么默认值中可以包含通配符（%、_、[] 和 [^]）。

7.OUTPUT ：表明参数是返回参数。该选项的值可以返回给 EXEC[UTE]。使用 OUTPUT 参数可将信息返回给调用过程。Text、ntext 和 image 参数可用作 OUTPUT 参数。使用 OUTPUT 关键字的输出参数可以是游标占位符。

8.RECOMPILE: 表明 SQL Server 不会缓存该过程的计划，该过程将在运行时重新编译。在使用非典型值或临时值而不希望覆盖缓存在内存中的执行计划时，请使用 RECOMPILE 选项。

9.ENCRYPTION: 表示 SQL Server 加密 syscomments 表中包含 CREATE PROCEDURE 语句文本的条目。使用 ENCRYPTION 可防止将过程作为 SQL Server 复制的一部分发布。 说明 在升级过程中，SQL Server 利用存储在 syscomments 中的加密注释来重新创建加密过程。

10.FOR REPLICATION :指定不能在订阅服务器上执行为复制创建的存储过程。.使用 FOR REPLICATION 选项创建的存储过程可用作存储过程筛选，且只能在复制过程中执行。本选项不能和 WITH RECOMPILE 选项一起使用。

11.AS :指定过程要执行的操作。

12.sql_statement :过程中要包含的任意数目和类型的 Transact-SQL 语句。但有一些限制。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5492743>

《认知盈余》——阅读感受与体会

2016年5月15日　　23:35

《认知盈余》

作者：舍基（"互联网革命最伟大的思考者"）

书中主要讨论的问题：随着全球用户接触互联网的门槛变得越来越低，互联网用户数量变得更加庞大，它们将形成什么样的社会形态？我们应该如何顺应这种变化？而作为互联网的从业者们，又该如何从中寻找自己的机会？

作者给认知盈余的定义：就是受过教育，并拥有自由支配时间的人，他们有丰富的知识背景，同时有强烈的分享欲望，这些人的时间汇聚在一起，产生巨大的社会效应。

举例说明：facebook，twitter以及维基百科的成功，都是“认知盈余”的功劳。在中国，微博的兴起，同样有赖于它。

阅读时的notes:

1.卖出去的为什么不能再进行交流

也许商店也是交流的平台

但是畏惧顾客之间对商品价格的交流

比如图书
阅读是有意义的 但同样渴望得到分享
2.搜索引擎的差劲
我想要的和给我的不一样
满足不叫我的需求
花费在查找上的时间比阅读自己查找到的东西的时间还早长
在现在的时间计算上，每个人都有着自由的时间，但是对于这些自由的时间进行如何的支配却并没有引起关注。也许这里的自由时间值得我们去好好利用和开发。如果没有大家自愿地为维基百科修改词条，怎么会有如今的维基百科呢？如果没有大家向YouTube的视频上传，YouTube又怎么能成为视频网站的代名词？也许你会问，我才没有时间去做哪些毫无意义的事情？那么你想一下你自己的自由时间用来干什么了呢？其实莫过于刷微博，刷空间，点赞，评论等等。不过你所做的同样也为这个网络时代进行了你自己的贡献。因为这样选择处理自由时间的不止你一个人。也许你不会有很具体的感受。但是这个例子足以看到"众"的力量。
昨日，汶川“5.12”纪念日，让我们在为他们默哀3分钟.....
在经过了这些年的重建，如今的汶川已经不再是废墟一片片。提到为何重建会如此的迅速，莫过于举全国之力来助力于重建。当初的"团结一心 众志长城"在现在看来依旧是很关键的。在现在的互联网时代，不可能存在独自的个体，个体的发展和互联网的发展改革有着很重要的关系。正是因为网民数量的激增，也许在不久的将来互联网的改革也就不远了。关于这场的改革，莫过于重新看待个人的自由时间，重新将分散的时间组合在一起。使它们能够得到完美的利用，也许地球上人类的数量在不断增加，但是每个人的一天也就只有24个小时，但是正是因为人口数量的增加以及每个人自由时间的增加，这样全体的自由时间总和也就在不断地增加。
中国，作为一个人口大国。也许这场互联网革命会有这不明显的国界，那么中国肯定属于在这场竞赛中占据着有力的形势。同样中国的网民数量也在不断地增加，其中包括老年人在网络时代的贡献。但是出于恐惧，传统文化媒体的各种头条便打上了“老年人竟然也在上网？”等各种夸大的言论。为何老年人就不能使用网络？为何老年人不能享受互联网带来的便利生活以及服务？在此对目前社会上媒体的标题党感到莫名的反感以及厌恶。
《认知盈余》在个人看来算是让自己明白自己所处的互联网的状况以及将来的发展趋势。也许只能算是趋势，谁让我们学了那么久的当前世界格局的发展趋势为“一超多强”。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5491656>

数据库知识点②

2016年5月15日　　23:35

1.ECA （Event-Condition-Action）

事件触发规则 **trigger [行级或语句级; 几个事件的组合; 触发条件为真]**

2.ACID

①Atomicity requirement 原子性要求

简单地说就是：要么同时拒绝，要么同时成功。

还是举个实在一点的例子：

(小菜刚写了转账系统，想在想试试)

小菜：既然你教给我这么多设计模式，还是要给些钱吧。我有个账户**8888*********8888,**大鸟你的账户是啥？我给你转钱

大鸟：那就转这个账户吧 **6666*********6666,**

(在转账操作中的小菜)

小菜：完了，停电了。我刚转出去，你等会上去看看有没有转到。

(过了一分钟)

大鸟：没有呀。

小菜：玩完了。

根据这个例子抽象出来原子性：一个事务包含多个操作，这些操作要么全部执行，要么全都不执行。实现事务的原子性，要支持回滚操作，在某个操作失败后，回滚到事务执行之前的状态。

②Consistency requirement 一致性需求

一致性,即在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。（完整性规则、参照完整性规则、自定义完整性规则**[Check**约束或者**trigger**触发器**]**）

③Isolation requirement 隔离性要求

举个常见的例子

在**Windows**中，如果多个进程对同一个文件进行修改是不允许的，**Windows**通过这种方式来保证不同进程的隔离性**:**

而SQL Server中，通过SQL SERVER对数据库文件进行管理，从而可以让多个进程可以同时访问数据库，为此SQL Server为了解决线程之间的冲突，设置了锁协议。

④Durability requirement 持久性要求

持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响。即一旦一个事务提交，**DBMS（Database Management System）**保证它对数据库中数据的改变应该是永久性的，持久性通过数据库备份和恢复来保证。

意味着在事务完成之后，该事务对数据库所做的更改便持久的保存在数据库之中，并不会被回滚。

数据库实现持久性的原理：**SQL server** 通过 **write-ahead-transaction-log** 来保证数据的持久性。**write-ahead-transaction-log**的意思是：事务中对数据库的改变在写入数据库之前，首先写入到事务日志中。而事务日志是按照顺序号进行排序的（**LSN**），当数据库崩溃或者服务器断电时，重新启动**SQL Server**，**SQL Server** 首先会检查日志顺序号，将本应对数据库做更改而未做的部分持久化到数据库中，从而保证了持久性。

⑤SQL Server通过利用加锁和阻塞来保证事物之间不同等级的隔离性

事务之间的互相影响可分为：脏读（**Dirty Read**）、不可重复读（**no-repeated Read**）、幻读（**Phantom Read**）

1.脏读：

一个事务读取到了另一个事务未提交的数据，而这个数据有可能是在之后会被回滚的。

举个例子：

甲将**A**账户的钱转到**B**账户，同时乙在此时对**B**账户余额进行查询。

甲 乙

从A账户取出$1000

转给B账户

读取余额，发现甲已经向自己转账

甲发现转的金额有错，

便执行了回滚，取消了

之前的操作

2.不可重复读：

在数据库访问中，一个事务范围内的两个相同查询却返回了不同数据，这是由于查询时系统中其他事务修改的提交而引起的。

举个例子：

甲和乙同时对**A**账户进行操作。

甲 乙

读取账户余额

读取账户余额

存入$1000

读取账户余额

前后两次乙的读取操作得到的账户余额不相同，只就叫做不可重复读。（因为乙重复读了账户余额，发现了不同，可能是这样才叫做不可重复读吧）

3.幻读（Phantom Read）

指事务不是独立执行时发生的一种现象，例如第一个事务删除某一行全部数据，第二个事务添加了一行新数据，好像没有修改一样，出现了幻读。

⑥理解SQL Server中的隔离等级

为了避免几个事务之间的影响，**SQL Server**通过设置不同的隔离等级来进行不同的避免，因为高的隔离等级意味着更多的锁，从而牺牲性能。

SQL Server提供了**5**中隔离，隔离等级由低到高分别为：

Read Uncommited（最高性能，但可能出现脏读，不可重复读，幻读）

Read Commited (符合**99%**实际需求，可能出现不可重复读，幻读**)**

Repeatable Read(可能出现幻读**)**

Serializable (最低性能，**Range**锁会导致并发下降**)**

SNOPSHOT（**SQL Server**中不涉及到的，并不常用）

⑦好的调度：

1.ACID的可串行化

2.可恢复的调度

3.无级联的回滚

⑧并发控制：增加合理的等待

1.Locking：为事务加上锁，

锁的协议，什么时候加，什么时候释放，遇到锁什么反应

2.Time-Stamping

3.Optimistic 积极考虑

假设施加在数据库上的操作大多数并不冲突

⑨Shared locks共享锁 AND Exclusive locks排它锁

两种锁之间的存在矩阵：

	S	X
S	Yes	No
X	No	No

⑩锁协议：两段锁协议 Two-Phase-locking

1.锁定阶段(Growing Phase)：把所有要锁定的数据全部加锁，

2.释放阶段(Shrinking):当开始释放锁时，不能再加上其他的锁，只能不断将之前加上的锁解开。

严格的两段锁协议 Strict 2PL

1.所有的事务，其所拥有的全部锁只能在所有事务执行完时，才能进行释放（commit/rollback操作之后）

2.其余要求和两段锁协议要求相同

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5479733>

设计模式之策略模式

2016年5月15日　　23:35

策略模式定义了一系列的算法，并将每一个算法封装起来，而且使它们还可以相互替换。策略模式让算法独立于使用它的客户而独立变化。

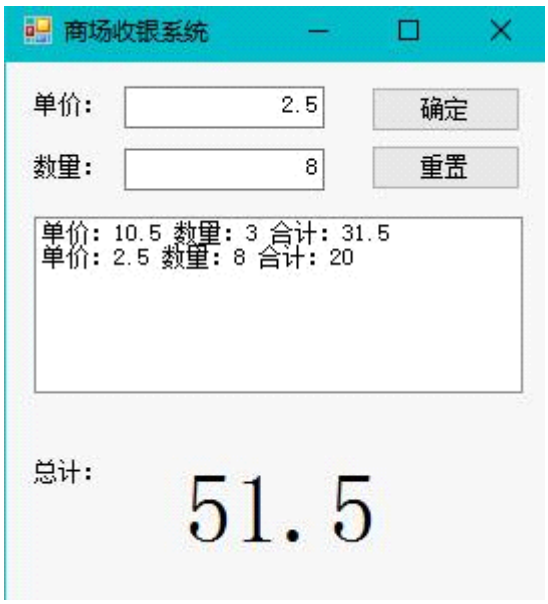
举个例子：

出场人物：小菜（菜鸟级程序员），大鸟（骨灰级程序员）

要求：做一个商场收银软件，营业员根据客户所购买的商品的单价和数量，向客户收费。

商场收银系统v1.0运行截图以及关键代码

```
 public partial class Form1 : Form    {
public Form1()
        {
            InitializeComponent();
        }
//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);
            total = total + totalPrices;//将每个商品合计计入总计
lbxList.Items.Add("单价："+txtPrice.Text+" 数量："+txtNum.Text+" 合计："+totalPrices.ToString());//在列表框中显示信息
lblResult.Text = total.ToString();//在lblResult标签上显示总计数
}
    }
}
```



现在问题来了：

大鸟：昨天妈妈节商场的所有商品打八折，那么上面的代码需要怎样修改呢？

小菜：上面的totalPrices乘以0.8不就可以了，没什么大不了的。

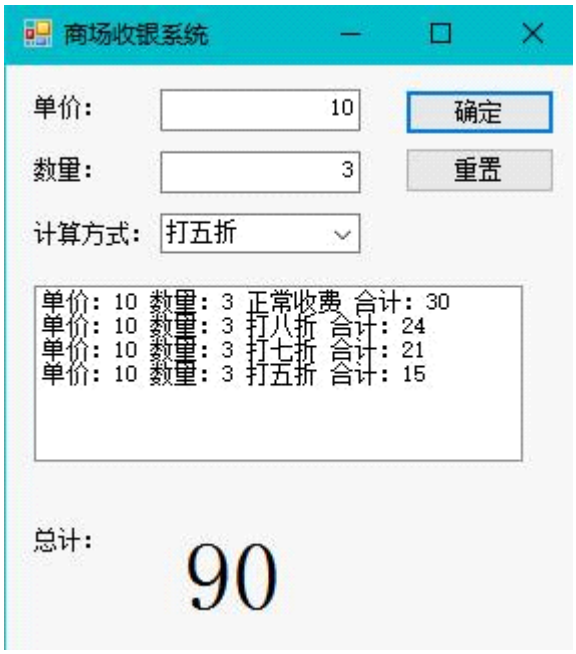
大鸟：但是过了妈妈节商场老板决定恢复原来的价格，那你是不是又要再改回来？不过再过不久也就六一儿童节了，商场要是打算打七折呢，你怎么办？

小菜：这样呐，我就加一个combox，里面给出打折选项就可以啦。

大鸟笑而不语。

商场收银系统v1.1运行截图以及关键代码

```
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"});//在combox中添加下拉选项
            cbxType.SelectedIndex = 0;
        }
private void btnOk_Click(object sender, EventArgs e)
        {
            double totalPrices=0d;
            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();
}
```



小菜：这样就可以了吧。

大鸟：比之前灵活了些，但是问题也不少：首先Convert.ToDouble()在这里就写了8遍，另外商场决定加大活动力度，满300返100这样的促销算法，你觉得该怎么办？

小菜：那意思就是满300返100，700的话就返200了？写函数就可以了吧？

大鸟：看来之前的简单工厂模式是白学了。

小菜：哦哦哦，这样子呐。那就先写一个父类，在继承它实现多个打折和返利的子类，使用多态，是这样子吧？

大鸟：这样你准备写多少个子类？

小菜：根据商店需求呀，比如9折，8折，6折，满300返100，满400返180.....要多少写多少O(∩_∩)O~~

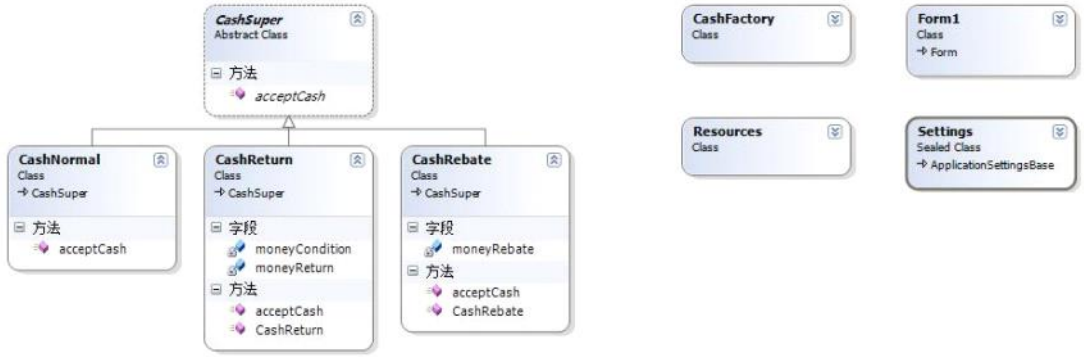
大鸟：小菜没有认真考虑这些促销活动之间的相同点和不同点呐

小菜：(⊙o⊙)哦，好像是有相同的地方，可以分为打折促销，返利促销，正常收费三个模式。这样可以写一个**父类，包含抽象方法：收钱()，然后写三个子类（正常收费，返利收费，打折收费）**分别实现这个收钱的函数，然后使用简单工厂模式，创建**一个收费工厂类**，在里面进行收费的选择，创建不同的收费实例。

大鸟：**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。**

商场收银系统v1.1运行截图以及关键代码

代码结构图：



现金收费子类：

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

namespace 商场管理软件
{
//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);
}
}
```

正常收费子类：

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

namespace 商场管理软件
{
    //正常收费，继承CashSuper
    class CashNormal : CashSuper
    {
```

```
        public override double acceptCash(double money)
        {
            return money;
        }
    }
}
```

打折收费子类：

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

namespace 商场管理软件
{
    //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;
        }
    }
}
```

返利收费子类：

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

namespace 商场管理软件
{
    //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;
        }
    }
}
```

现金收费工厂类：

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

namespace 商场管理软件
{
//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;
    }
}
}
```

客户端程序主要部分：

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace 商场管理软件
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
//客户端窗体程序（主要部分）
        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
            CashSuper 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(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();
        }
private void btnClear_Click(object sender, EventArgs e)
        {
            total = 0d;
```

```
            txtPrice.Text = "0.00";
            txtNum.Text = "0";
            lbxList.Items.Clear();
            lblResult.Text = "0.00";
        }
    }
}
```

小菜：大鸟，这样不管怎样修改，我都可以简单处理了。

大鸟：那我要实现打五折和满500返200的返利活动，你怎么修改？

小菜：在现金工厂中添加打五折和满500返200的case语句，然后在下拉选择框中添加两个选项就可以了。

大鸟：现金工厂？应该是收费对象生成工厂才准确。但是如果促销修改为满100积分加10，当积分达到一定时候就可以领取奖品怎么做？

小菜：有了工厂，何难？添加一个积分算法，构造方法里面有两个参数：条件和返点，继承CashSuper，然后在现金工厂，哦，不对，是收费对象生成工厂里面增加满100积分加10的分支条件，然后在下拉选择框里面添加这个选项就可以了。

大鸟：但是在以后的学习中我们会发现这样的设计模式并不是最好的选择，因为每次维护或者扩展都要修改工厂这个类，以至于代码需要重新编译部署，这样的处理很糟糕，自己去研究一下看看那个设计模式可以用在其中。

小菜（陷入沉思.....）

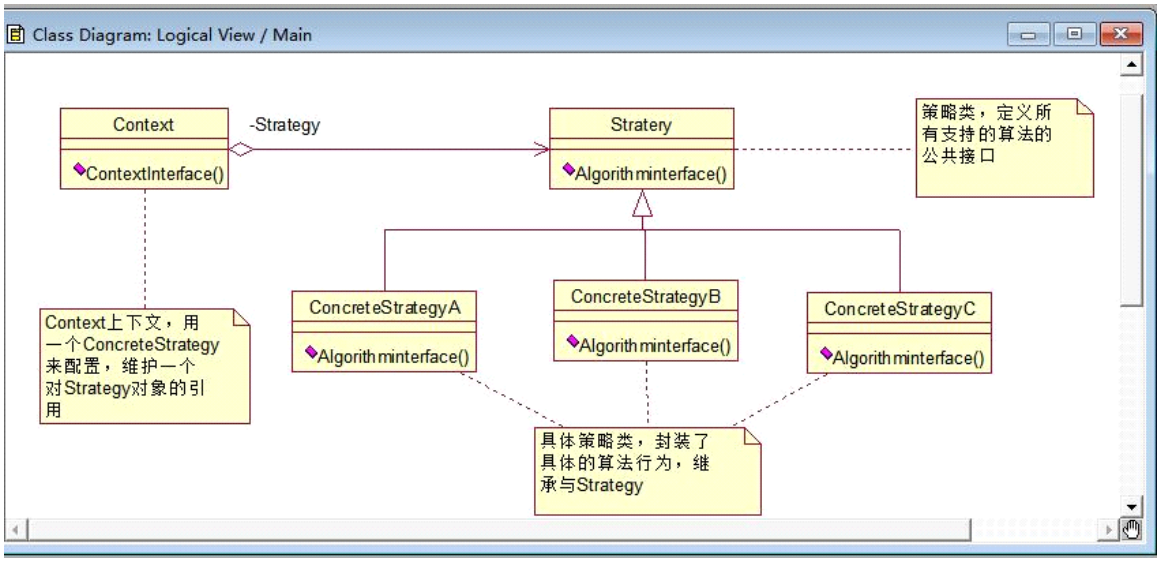
次日

小菜：大鸟我知道该使用哪种设计模式了，选择策略模式就很好地解决了这个问题。

策略模式(Strategy):它定义了算法家族，分别封装起来，让它们之间可以相互替换，此模式让算法的变化，不会影响到使用算法的客户。

大鸟：你说说看。

小菜：这里是我画的策略模式结构图




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

namespace 策略模式
{
    class Program
    {
        static void Main(string[] args)
        {
            Context context;
context = new Context(new ConcreteStrategyA());
            context.ContextInterface();
context = new Context(new ConcreteStrategyB());
            context.ContextInterface();
context = new Context(new ConcreteStrategyC());
            context.ContextInterface();
Console.Read();
        }
    }
//抽象算法类
    abstract class Strategy
    {
        //算法方法
        public abstract void AlgorithmInterface();
    }
    //具体算法A
```

```
    class ConcreteStrategyA : Strategy
    {
        //算法A实现方法
        public override void AlgorithmInterface()
        {
            Console.WriteLine("算法A实现");
        }
    }
    //具体算法B
    class ConcreteStrategyB : Strategy
    {
        //算法B实现方法
        public override void AlgorithmInterface()
        {
            Console.WriteLine("算法B实现");
        }
    }
    //具体算法C
    class ConcreteStrategyC : Strategy
    {
        //算法C实现方法
        public override void AlgorithmInterface()
        {
            Console.WriteLine("算法C实现");
        }
    }
    //上下文
    class Context
    {
        Strategy strategy;
public Context(Strategy strategy)
        {
            this.strategy = strategy;
        }
        //上下文接口
        public void ContextInterface()
        {
            strategy.AlgorithmInterface();
        }
    }
}
```

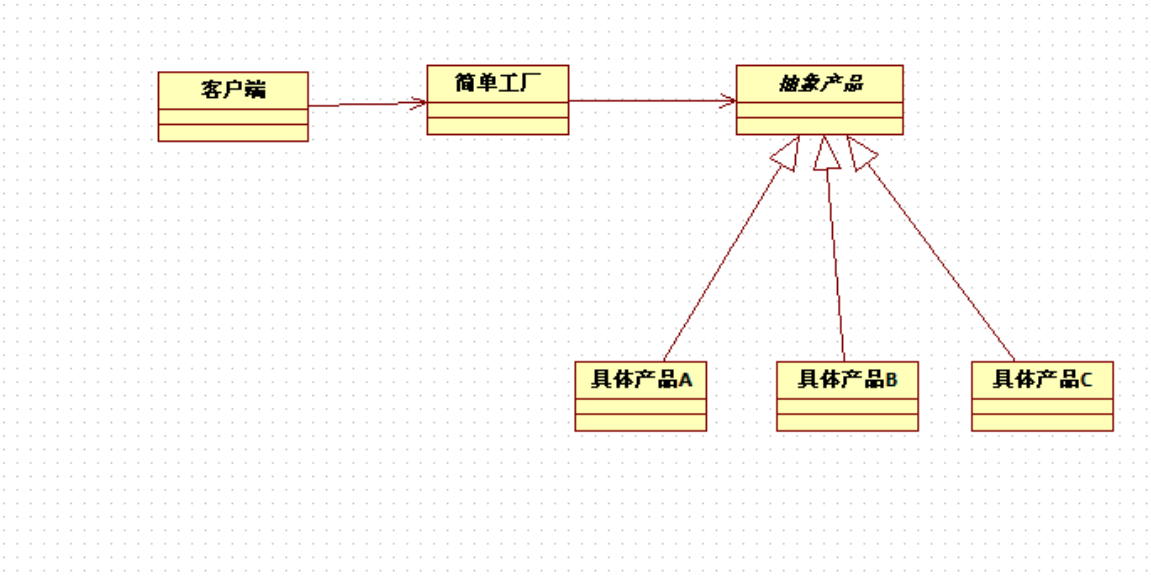
来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5475908>

设计模式之简单工厂模式

2016年5月15日　　23:35

简单工厂模式是属于创建型模式，又叫做静态工厂方法（Static Factory Method）模式，但不属于23种GOF设计模式之一。简单工厂模式是由一个工厂对象决定创建出哪一种产品类的实例。简单工厂模式是工厂模式家族中最简单实用的模式，可以理解为是不同工厂模式的一个特殊实现。

简单工厂模式的UML图：



简单工厂模式深入分析：

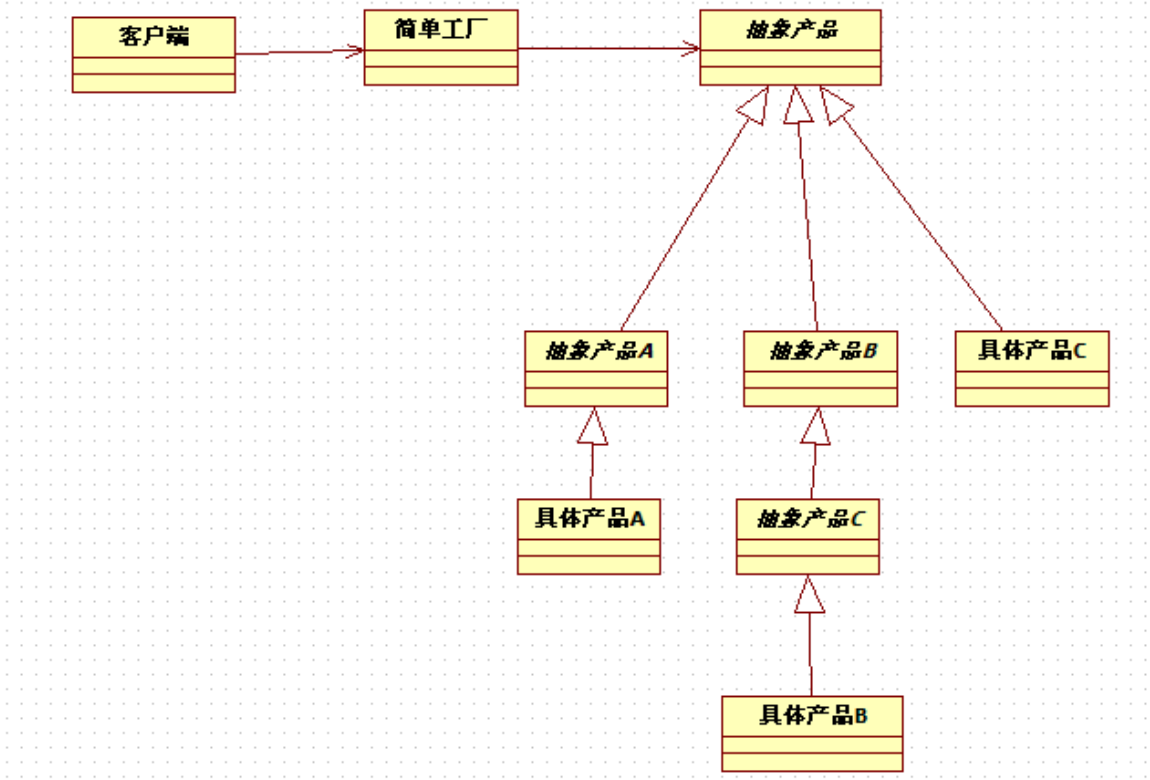
简单工厂模式解决的问题是如何去实例化一个合适的对象。

简单工厂模式的核心思想就是：有一个专门的类来负责创建实例的过程。

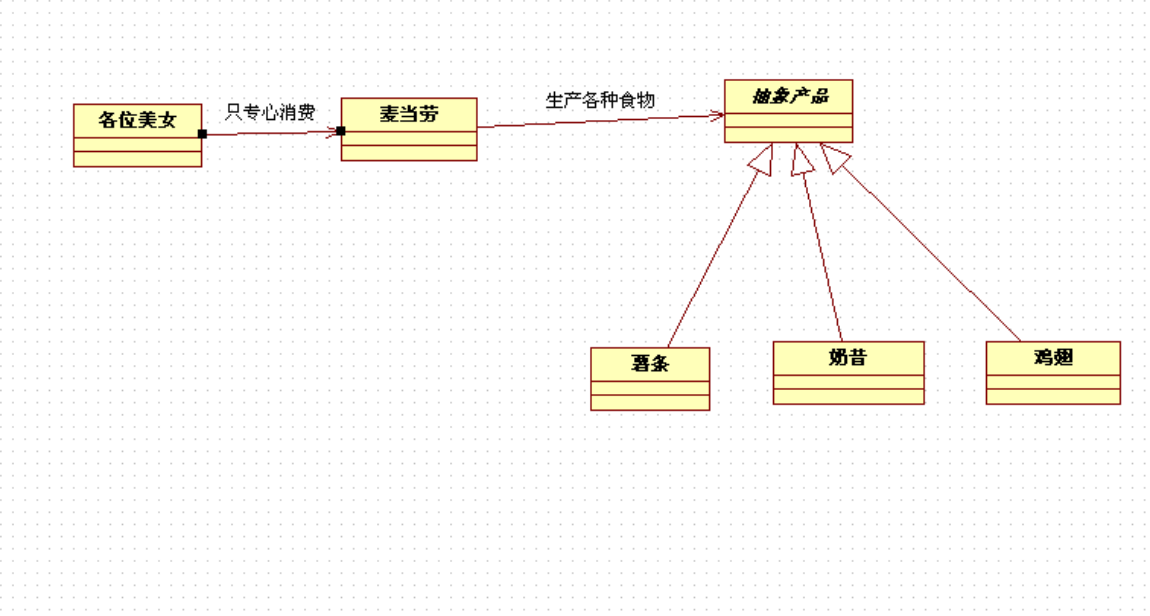
具体来说，把产品看着是一系列的类的集合，这些类是由某个抽象类或者接口派生出来的一个对象树。而工厂类用来产生一个合适的对象来满足客户的要求。

如果简单工厂模式所涉及到的具体产品之间没有共同的逻辑，那么我们就可以使用接口来扮演抽象产品的角色；如果具体产品之间有功能的逻辑或，我们就必须把这些共同的东西提取出来，放在一个抽象类中，然后让具体产品继承抽象类。为实现更好复用的目的，共同的东西总是应该抽象出来的。

在实际的的使用中，抽闲产品和具体产品之间往往是多层次的产品结构，如下图所示：



来一个具体的例子：



优点：

工厂类是整个模式的关键.包含了必要的逻辑判断,根据外界给定的信息,决定究竟应该创建哪个具体类的对象.

通过使用工厂类,外界可以从直接创建具体产品对象的尴尬局面摆脱出来,仅仅需要负责“消费”对象就可以了。而不必管这些对象究竟如何创建及如何组织的.

明确了各自的职责和权利，有利于整个软件体系结构的优化。

缺点：

由于工厂类集中了所有实例的创建逻辑，违反了高内聚责任分配原则，将全部创建逻辑集中到了一个工厂类中；它所能创建的类只能是事先考虑到的，如果需要添加新的类，则就需要改变工厂类了；

当系统中的具体产品类不断增多时候，可能会出现要求工厂类根据不同条件创建不同实例的需求．这种对条件的判断和对具体产品类型的判断交错在一起，很难避免模块功能的蔓延，对系统的维护和扩展非常不利。

使用场景

工厂类负责创建的对象比较少；
客户只知道传入工厂类的参数，对于如何创建对象（逻辑）不关心；
由于简单工厂很容易违反高内聚责任分配原则，因此一般只在很简单的情况下应用。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5465369>