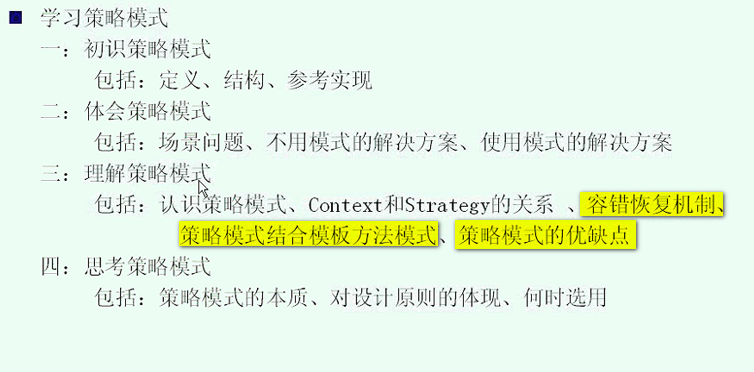
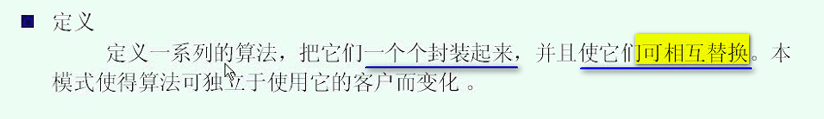
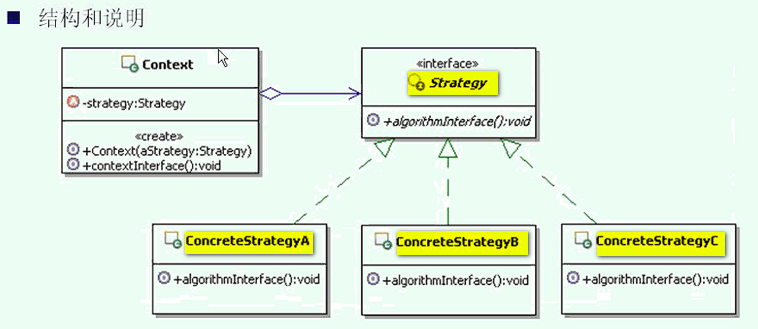
策略模式





目的就是让让算法 可以相互替换 ----- 相互替换是策略模式要做的

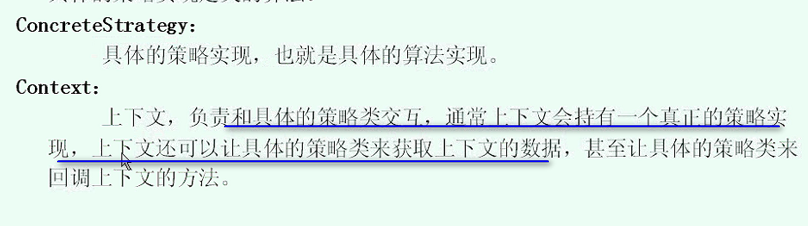
算法可以独立于使用它的客户而变化



A B C三个是一系列的算法 都实现了同一个Strategy接口

ConcreteStrategy是具体的算法实现

Context叫做上下文 负责具体的策略类交互



如果说 这个时候 客户找上下文 找策略的实现

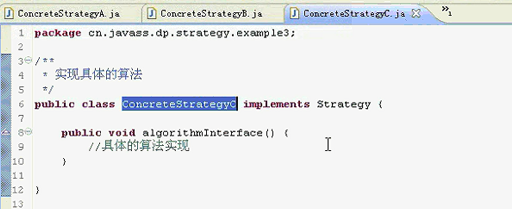
上下文还可以让具体的策略类 获取上下文的

真正的客户端通过上下文使用算法

看一下策略模式的参考实现



A B C具体的实现



很明显 A B C里面 ---- 这里面的实现 不会完全一样

看看Context对象



通过构造方法 传递进来

客户端调用这个上下文的接口

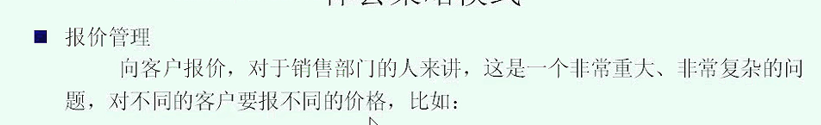
这个就是参考示例代码

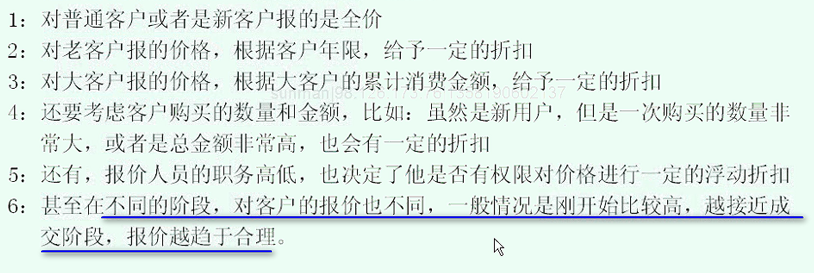
这样 对策略模式有一个基本的认识

现在体会策略模式

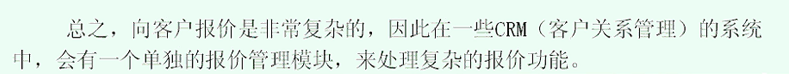
实际开发中怎么使用

报价管理 在很多系统里面都有





总而言之 客户报价 是非常复杂的

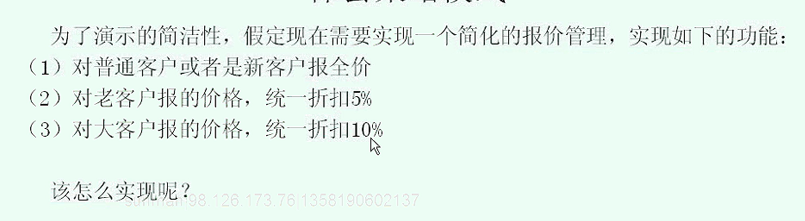


那个不仅仅是这些考虑

比如每次报价信息要记录下来

这是说我们这个实际应用

进行了简化



不用模式的解决方案

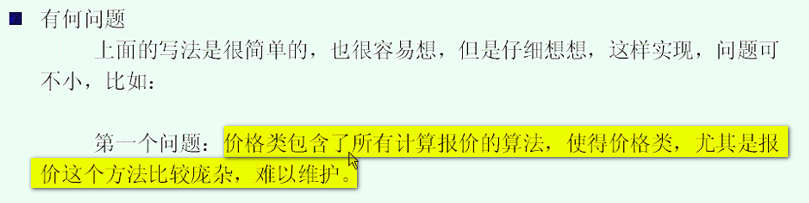


这个确实很简单 ---- 这个是简化之后的实现 看起来实现很简单

但是 有没有什么问题么

大体上会有什么问题？

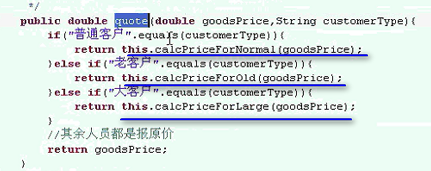
价格类里面包含了所有报价计算的算法



里面的算法会越来越复杂

简单的就是 把这些算法从报价中拿出来

写出如下的代码



每一种 都是封装成一个方法了

这样这个方法没有那么麻烦了

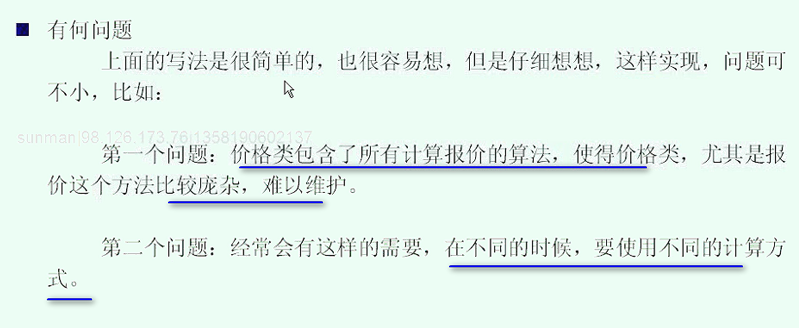
这个实现比刚才的哪个好 毕竟接近了单一职责原则

虽然把这些算法 挪到了了外面 但是 还是在Price里面 这个解决的不彻底

现在还有100中群体 这样算法和算法的方法很多 同样是的这个价格的类非常庞大

还有一个毛病 添加功能 或者修改功能 都要到这个类中修改代码

这是咱们的第一个问题 价格类比较庞杂



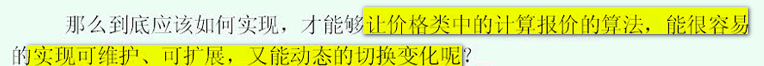
公司庆典的时候 都额外增加3% ----- 就是报价的方式都需要修改 或者替换

促销或者周年庆 这个报价要改变 ----- 否则就要切换回来

不同的时候 使用不同的计算方式 报价的策略 和 算法

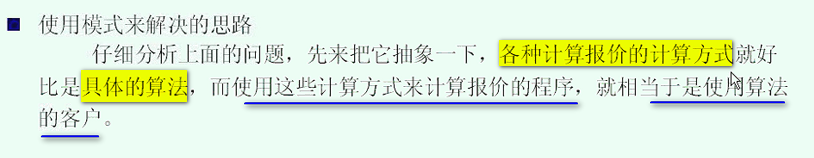
现在的代码是固定调用各种计算方式

这样就要导致修改各种代码 才能满足需要

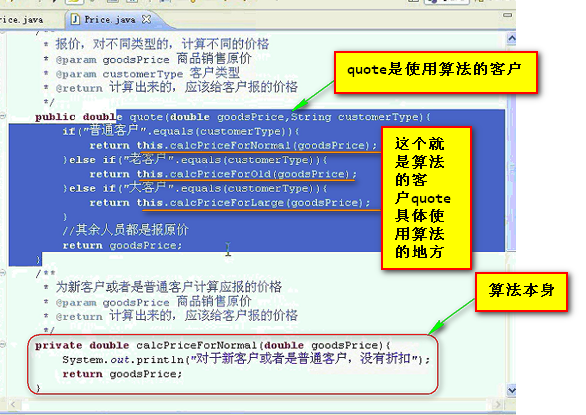


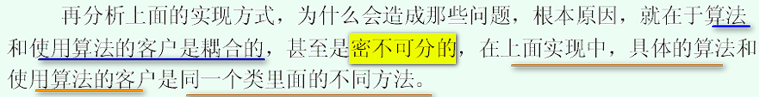
解决这些问题 就是策略模式

使用模式解决问题的思路



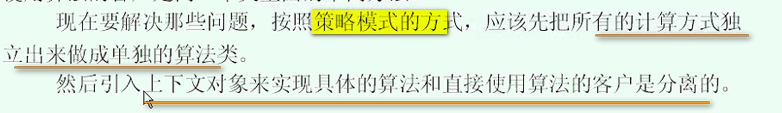
**根本原因 就在于 算法和使用算法的客户是紧耦合的**



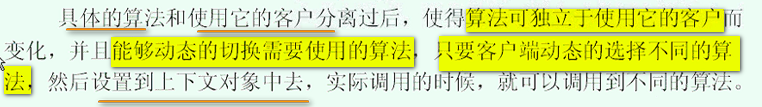


【**这么多模式 学过之后 发现没有谁用案例的造成的难以维护的原因 就是某几个之间紧耦合 没有分工 ----- 模式不就是分工么**】

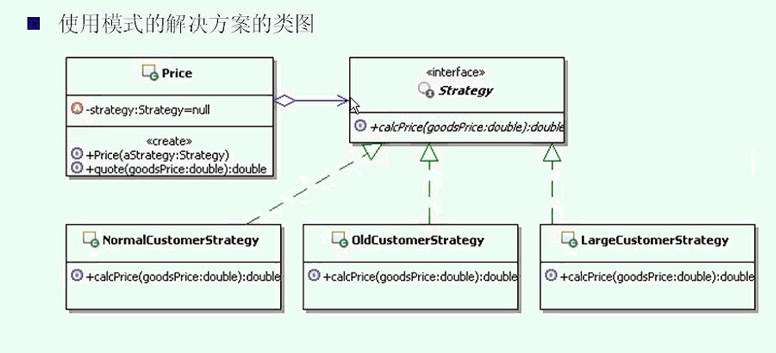
现在把计算方式独立出来 做成单独的算法类 这样就形成了一系列的算法



上下文是持有这个算法 但是 选择具体的算法 还是客户端来定



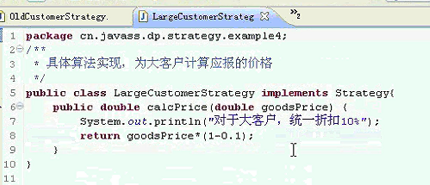
上下文职负责持有算法 ---- 具体运行的算法还是客户说了算 这就是使用模式来接问题的思路



这个时候 Price是上下文 是使用算法的使用者 这个Price并不知道用哪个策略

如果是促销的 就使用促销的算法 如果是平常的 就使用平常的算法 ------ 这个就是替代的关系【和状态模式不同】





这就是算法及其实现的算法类

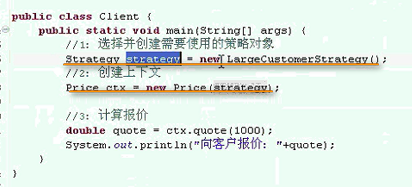
看一下Price

【这个Context并不负责维护状态数据 和 状态数据的切换】

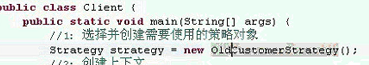
这就直接计算出来 来的非常简单

有了这个之后 看一下客户端

选择什么样的计算 还在客户端手里



现在是大客户 我就可以那啥

【从这个里面可以看到 我这个地方 策略都是客户端来选 ----- 现在 如果我想使用老客户来计算 ------ 那我就在client段进行修改  随时都可以切换算法 ---- 这不就可以替换了么 ----- 策略之间就是平行的（也就是 我这个客户之间不具有连续性 但是投票是连续的） ----- 并且客户端是知道 我有哪些客户 什么客户 我该使用什么策略来进行计算 --------但是 状态模式那个投票的 好像 投票是连续的动作 if-else之间的内容 不是可以替换的 所以在每一个State中 都由有状态切换的过程 ----- 并且客户端是不知道这个状态机有几个状态】

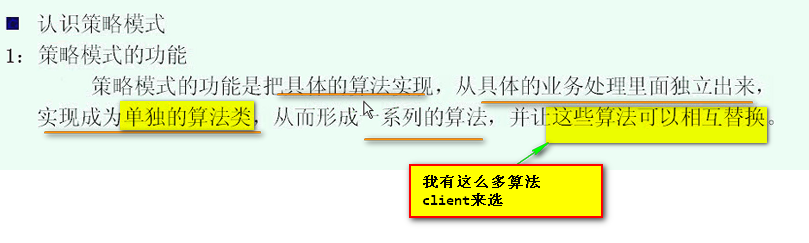


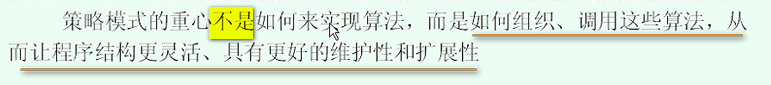
【实际上 我这里面 把策略的维护放到了Client端 这种if-else的替换方式 实际上 就是 我没有完整的替换 也就是if else中的每一个内容 真的可疑相互替换 那么 我客户端每次修改代码的时候 都相当于 要么执行的是以前的if 要么是以前的else if 要么是以前的else】

这样算法之间是可以替换的 这就变得非常容易了

【实际上 更加完整的程序 可以是 有一个Customer类型 有一个String type字段 type是xxx 我也可以调用xxxStrategy 之类的 还是有一个else if的类型 这样也未必好 ---- 其实 这也是 不能整体运转 】

================================

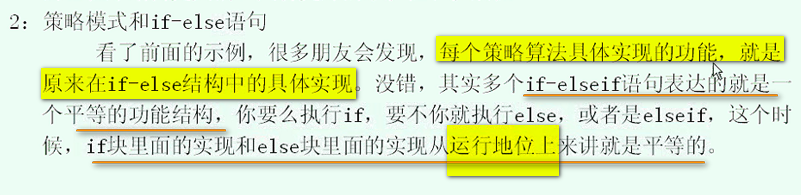




这个是从结构上来 不是从功能上来

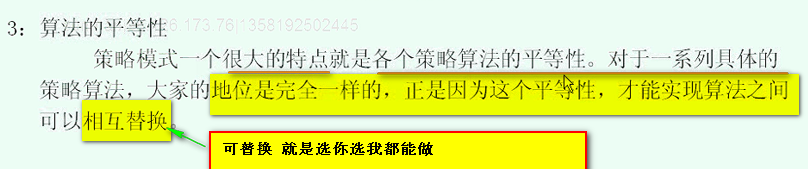
这个是策略模式的功能

策略模式和if-else语句



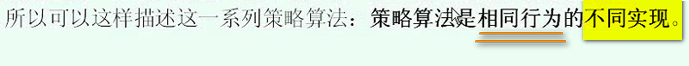


**从这个角度来讲 如果出现多个if-else 可以考虑使用策略模式 ---- -但是 不一定一定可以使用策略模式**



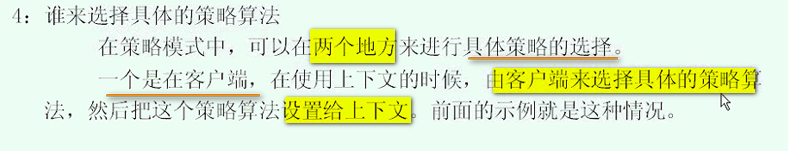
相互之间没有依赖



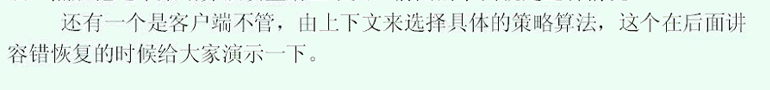


有的资料把这种情况称为多态的行为

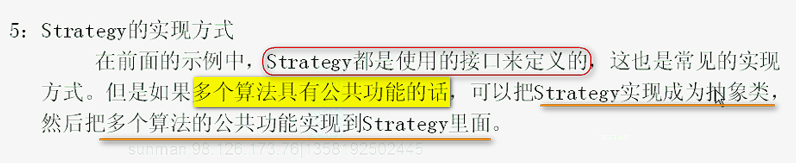
有了一系列的算法 谁来选择？ 策略模式可以在两个地方进行具体策略的选择



另一个是客户端不管 就是上下文来选择 客户端说 我就要计算



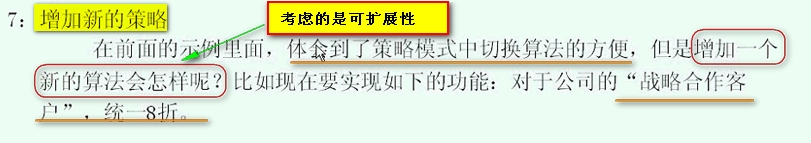
具体的策略的实现方式



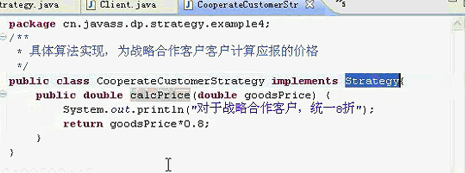
这个就是抽象类和接口的选择 ---- 就是既要约束子类的行为 又要为子类提供公共功能 选抽象类 而不是接口



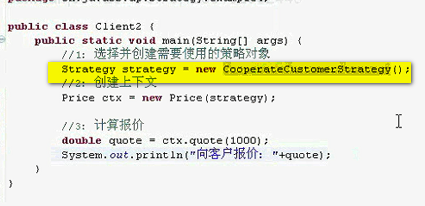
虽然可以动态切换策略 但是 运行的时候 是唯一的



首先新家一个合作伙伴的策略类

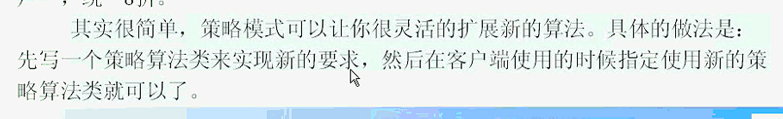


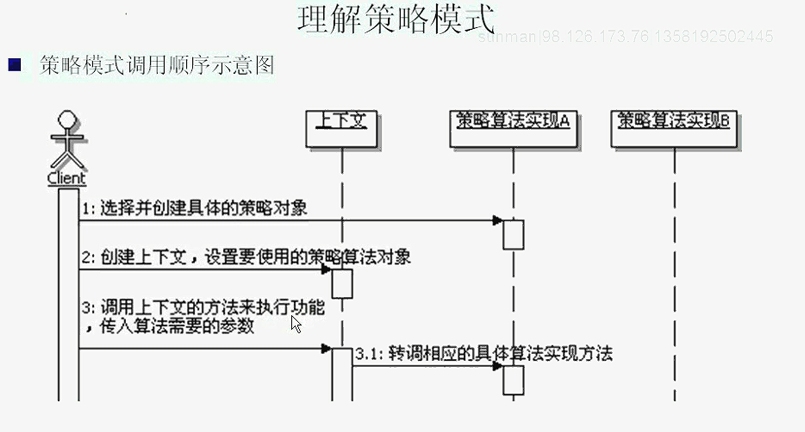
此时在客户端使用的时候 就可以选择这个策略类



我们都没有修改 就有的功能没有变 只是添加了新的功能 客户端使用这个新的算法就可以了

使用了这个策略模式 动态切换 扩展strategy是非常方便的 明显有好坏之分



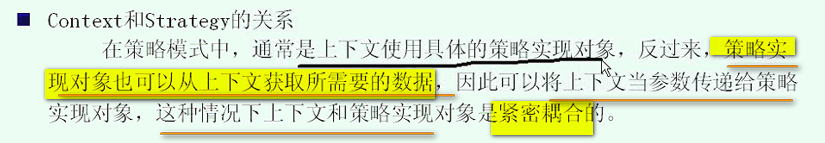


这个是从客户端控制使用哪一个策略

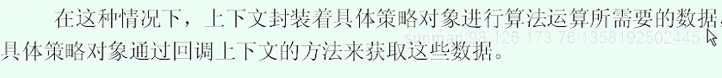
这个就是第一种 客户端选策略的情况

到这个地方 对策略模式的认识 就很深入了

上下文和策略的关系



现在的这个Price这个类是松耦合的 是单向的



上下文有一个 所有的策略都能得到上下文对象

策略A 需要 id name 把这个id name放到Context里面

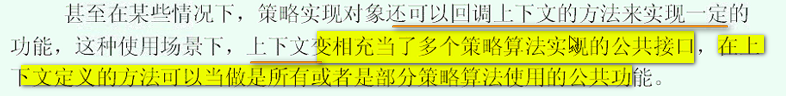
B需要id age name

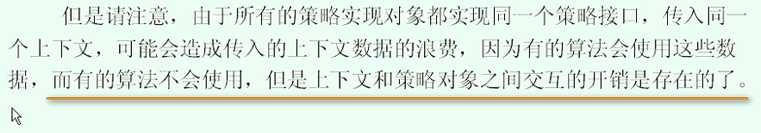
======= 抽取

A 什么都没放

B 放了age

Context中除了放公共数据 还放公共方法

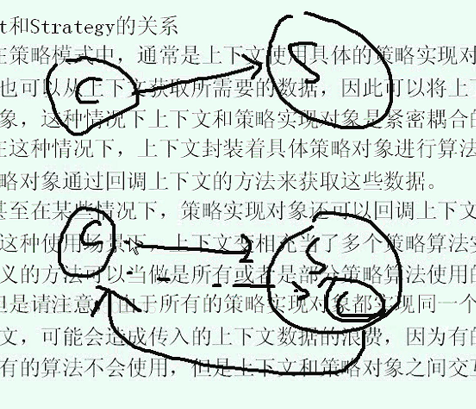




可能会造成传入的上下文数据的浪费 并不是大问题

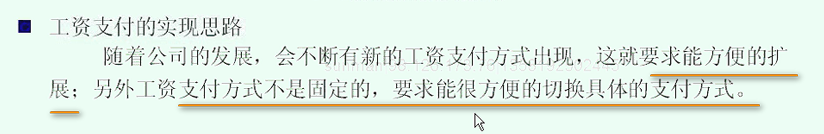
现在Context 就是持有Strategy ---- 单向的

现在 Context调用Strategy的时候 ----- 并把自己作为参数 传入Strategy ---- 双向的



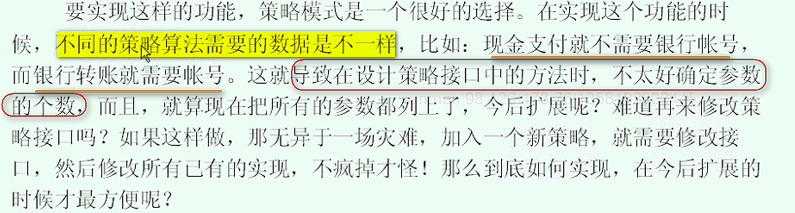
接下来看看例子

使用一个实际的案例



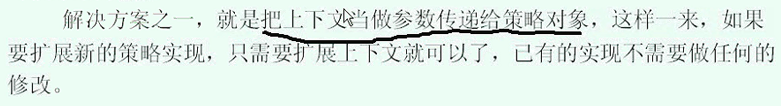
支付方式很多 另一个问题 随着公司的发展 不同的支付方式出现了 比如后面采用了股权 要求能扩展新的支付方式

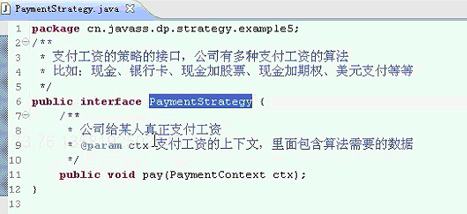
要求能扩展支付方式 + 要求能方便切换支付方式 ----- 典型使用策略模式



接口这个坐上了 肯定不能改

要解决这个问题 就是把上下文当参数 传入策略对象



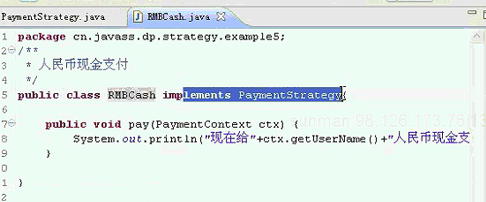


这个方法 有一个参数 就是PaymentContext参数

策略需要的数据都封装在上下文里面

看下各种各样的具体实现

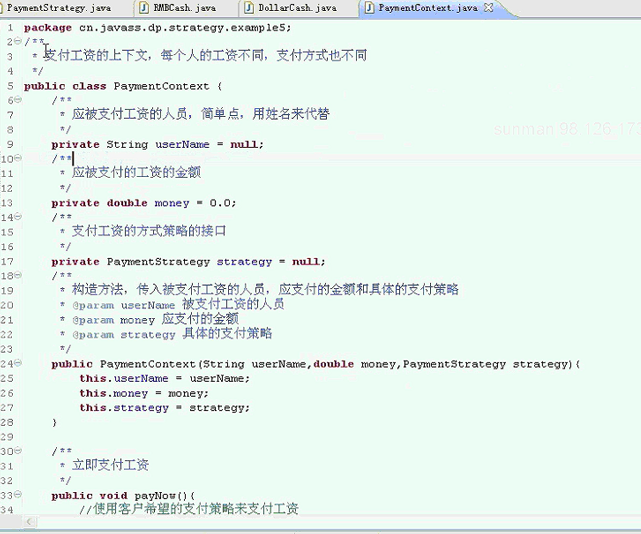
先看人民币支付 和 美元支付

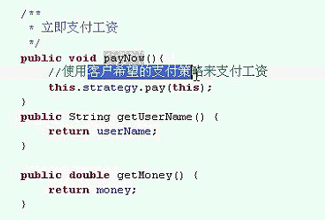




这些数据都来自于上下文

上下文的实现





看些客户端



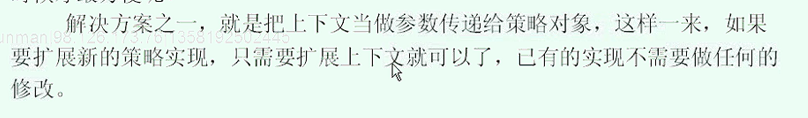


这里面是自己选择算法 通过上下文 把这个传递进去

这个体会一下 把context传递到策略中 也就是把上下文当做参数传递给策略

在策略中回调上下文的方法

还差一件事：



客户端切换没问题 那么这里面 实现扩展怎么办？

比方说 实现一个新的功能

实现转账到银行卡这个功能

要想转账到卡 需要一个账号 但是 context里面有人名 钱 策略



32：21