|  |  |
| --- | --- |
| 一、实验内容  设计一个子系统，该子系统有三个类：ReadFile，AnalyzeInformation和SaveFile，各个子类的职责如下：  （1）ReadFile类的实例可以读取文本文件  （2）AnalyzeInformation类的实例可以从一个文本中删除用户不需要的内容，可以有单独的方法来实现不同的删除效果。  （3）SaveFile的实例能将一个文本保存到文本文件  应用外观模式设计该子系统，以便简化用户和上述子系统间进行的交互。要求：用户把需要读取的HTML文件名，及要保存的文件名告诉系统即可。例如，用户想要读取一个HTML文件时，将文件中的全部HTML标记去掉后保存到另一个.txt文本文件中。 | |
| 二、实验过程记录 | |
| 类图 |  |
| 实验过程记 录 | 在本例的设计中，最初是没有抽象层的。而后来经过思考，我想到题目中的AnalyzeInformation子系统可能可以通过不同的方式处理文件，删除HTML标记可能只是其中一种操作。如果系统要扩展功能的话，增加、删除、更换子系统类的同时必须修改客户端和外观类的源代码，不符合开闭原则。为了满足可能出现的新的业务需求，我给最初的版本增加了AbstractFacade用来充当抽象外观类，Facade充当继承自抽象外观类的具体外观类，ReadFile、AnalyzeInformation和SaveFile充当子系统类。同时，我增加了XML配置文件用来存储具体外观类的类名，用XMLUtil工具类读取配置文件并反射生成相应的对象。这样，客户端测试类Client就可以进一步针对抽象编程。而具体外观类实现的是把读取HTML文件、删除HTML标记、保存为.txt文件的方法，把用户和各个子系统的交互简化为统一和具体外观类进行交互，发挥了外观模式的优势。 |
| 总结 | 外观模式是一种对象结构型模式，它为子系统中的一组接口提供了一个统一的入口，使子系统更加容易使用。外观模式的优点是对客户端屏蔽子系统组件，减少了客户端所需处理的对象数目，简化了客户端的使用。但是有些时候如果对客户端访问子系统类做太多限制会降低系统的灵活性，如果设计不当还有可能在增加子系统时修改外观类的源代码，违背开闭原则。外观模式适用的情况有：要为访问一系列复杂的子系统提供一个简单入口；客户端程序与多个子系统间依赖性过强；层次化结构中用外观模式定义层与层之间的接口，降低层间耦合度。 |