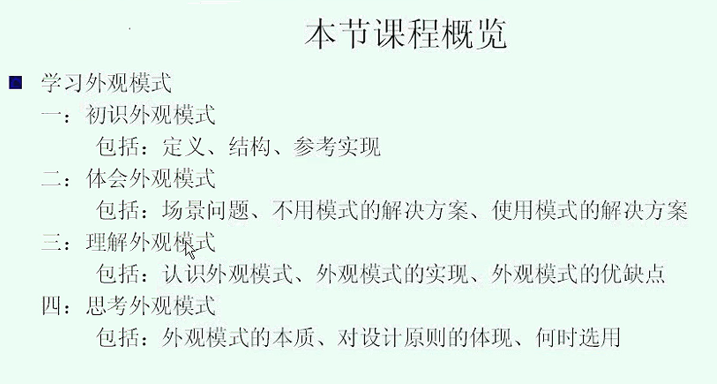
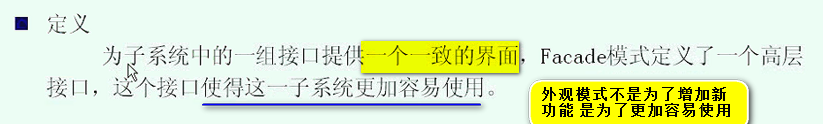
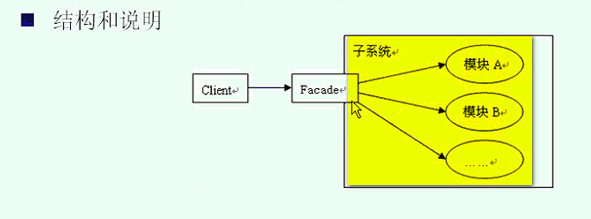
外观模式

Facade 注意发音 fəˈsɑːd ----- 第二个音节重度 并且 不是ei 是 a:



外观模式用来为客户端提供一个界面 ---- 为一组接口提供了一致的界面





界面是什么意思？------ 事实上 这个界面 不是图形化的界面 而是**从客户端看待子系统 看到什么东西 这个什么东西就是一个界面**

**接口是实现类对外的一个界面【这句话很好】**

子系统中 有多个模块 A B

每个模块对外都要提供一个接口

**一致性**：对外来讲 A B C…这些子系统提供的访问方式都是一样的 那么这个界面就是一致的

所以 client只需要访问Façade 就可以 实际上 client仅仅能看到Facade 子模块A B C

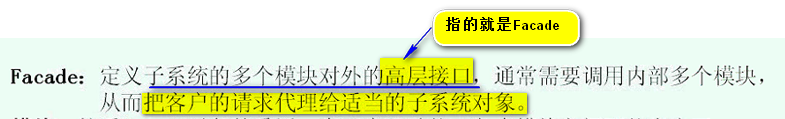
高层：子系统中的A B C模块是更底层的具体更能打交道

 ----- 这个Facade提供的高层接口---- 不仅仅是interface ---- 表示外部client和子系统交互的方法

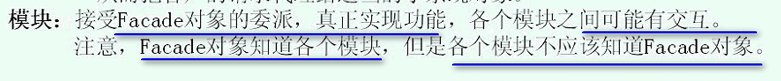
也就是除了子系统对外提供的一致的界面Facade

这个接口使得这个子系统更加容易被Client使用 ----- 这样 客户端Client仅仅和Façade打交道 不要和子系统中模块A B C打交道

以前做一件事 做一件事 可能要和各种政府机构大交到 ----- 现在开启一站式的窗口 ---- 交到这个一站式的窗口 然后几个工作日之后 还是到这个一站式窗口 获取结果 这样 办事人员不用到具体的机构 ----- 直接Façade打交道



所以 不是Facade本身完成客户端的功能request 而是 交给内部的子模块来完成



我们的图上并没有表现出来模块之间会通信

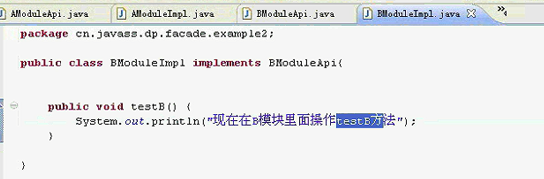
Façade模块知道子系统A B C 但是反过来 子模块A B C不知道Façade

这就是外观模式定义 结构 和 基本说明

========== 参考示例代码 ==========

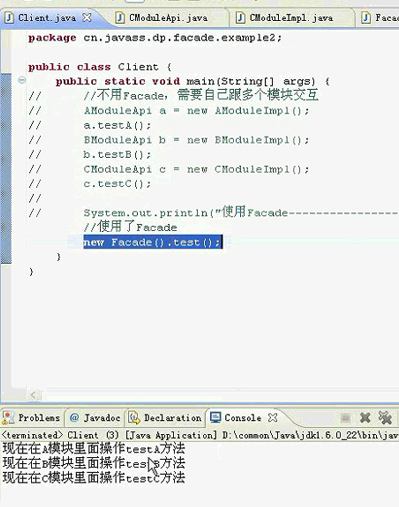






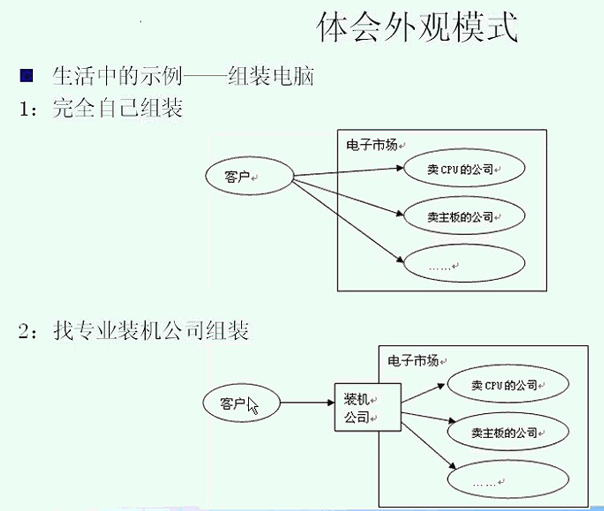
现在Façade实现成了一个class

【基本示例 是没有】



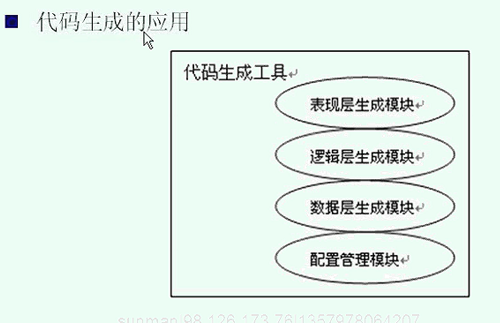
这个是Façade基本的实现示例

现在看一下生活中的示例

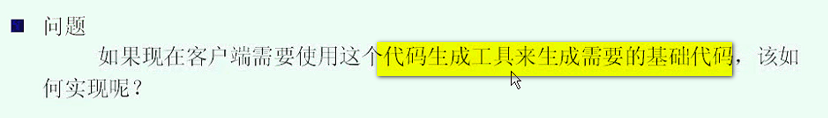


装机公司 就相当于对客户的外观

SSH做一些应用的时候 需要代码生成



代码生成的工具 要做什么事情 全部从头到尾罗列起来



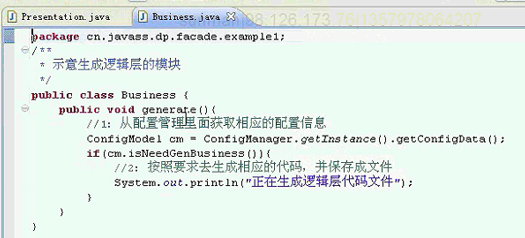
这个说明了 已经有了代码生成工具 该怎么使用？

先看不用模式的解决方案？

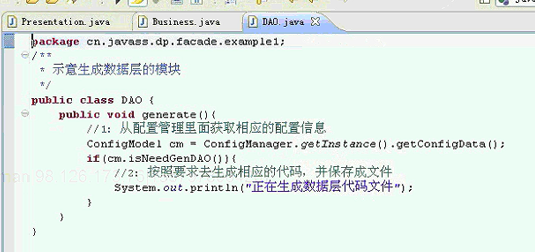


这个是表现层的生成模块

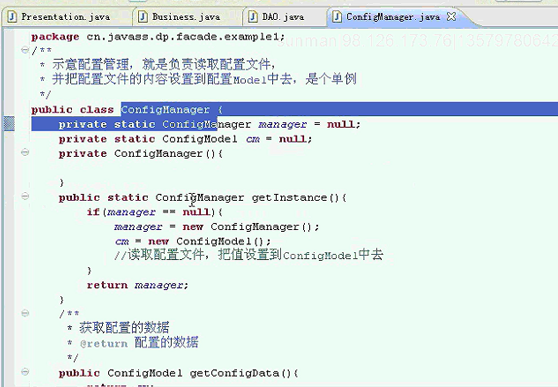
从配置管理里面获取相应的配置信息 ---- 如果需要生成 就生成



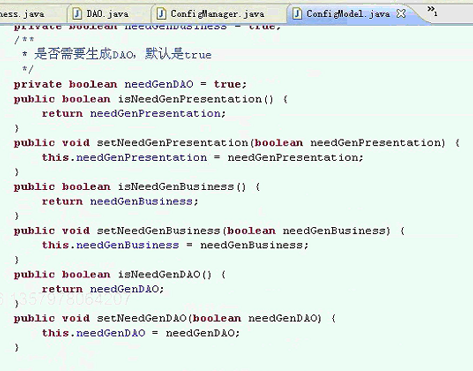
这个是逻辑层的生成模块



这个是DAO层的生成模块



这个是配置管理的生成模块





看一下ConfigManager

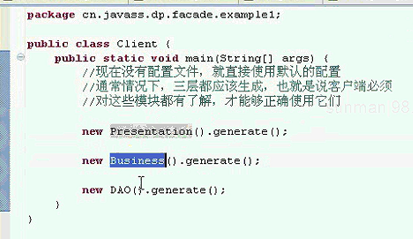


这里面简化了 就是 返回了一个空的 全新的ConfigModel()

这就是现在已有的代码生成工具

我们看一下客户端 要是用这个代码工具 该怎么办？

不用模式

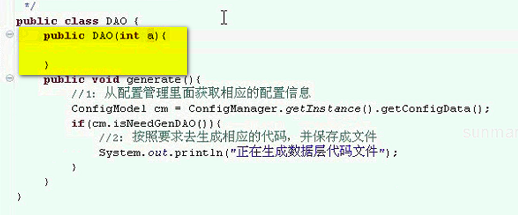


如果说 这么做 有什么问题？

如果客户端想要把这个模块生成出来 必须对生辰工具的内部都要有了解 才能正确的生成 ----- 违反了迪米特原则

如果不知道如何使用某一个模块 客户端可能就无法生成

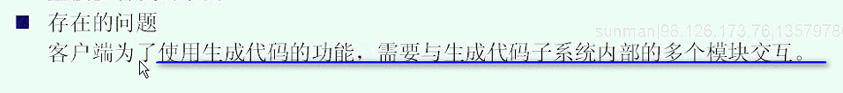
如果其中的某一个子模块修改了构造方法

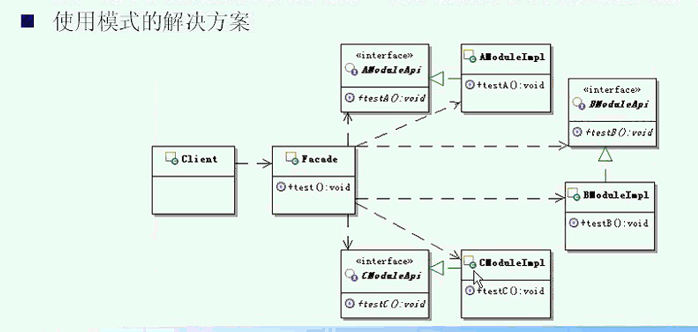


这样客户端调用就会出现错误

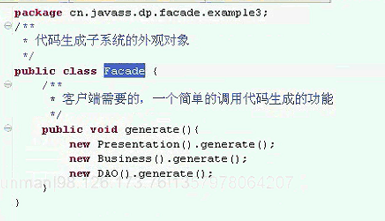


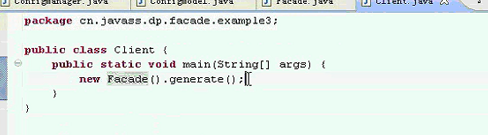
所以 没有办法 系统的扩展性和维护性 是非常糟糕的



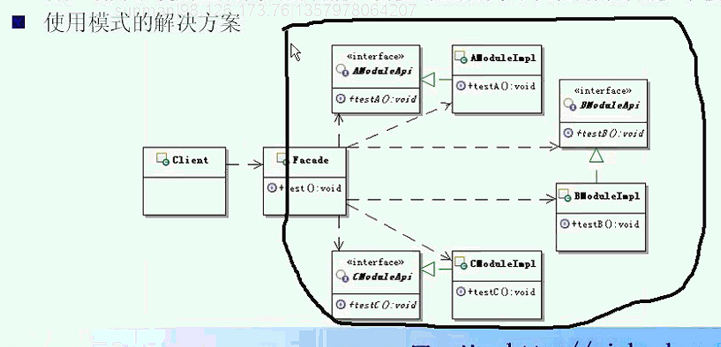


使用Façade 模式 来解决这个问题





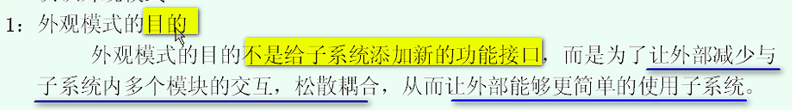
这么实现过后 客户端就不需要和里面各种进行打交道了 不需要关心子模块的生成机制



Client看代码生成工具 就只是看到了Façade这个一致性的界面

实现了客户端和内部子系统之间的解耦合

如果还是刚才的实现 也出现了Dao里面有了改变 传一个参数 该怎么办 修改的是Façade



门面模式 代理模式 和 适配器模式的区别

<http://blog.csdn.net/jljf_hh/article/details/51655008>

【从示例代码来看 子模块什么都 都没有聚合发生 最大的问题 就体现在无法满足OCP 其他两个都没哟问题】

【本质上 外观模式 就是为了使得子系统易用而已 没有添加新功能 但是 代理模式不是 他是为了对代理对象的访问权限进行控制 适配器模式正常来讲 是让不能在一起工作的可以一起工作 ---- 也没有新功能生成 所以 区分适配器和门面要从场景区分】

<http://blog.csdn.net/puma_dong/article/details/23081081>

【这个博客很好】