



这些要从模式的本质上进行思考

最后 看这个模式的使用程度



变形使用很重要





有些设计模式 策略 状态 会有很多对象 耗费了更多的内存 比如备忘录 等等 都要求评估

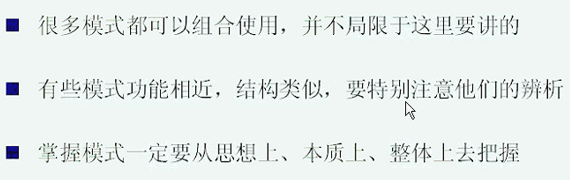
要面临实际问题的时候 进行实际开发处理 进行思考

这个地方 最适合的 就是最好的

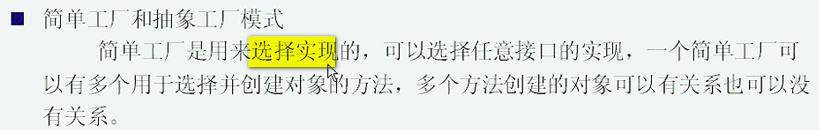


深刻理解 每个模式的本质

万变不离其中



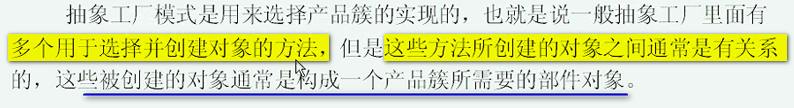
-------- 按照课程的先后顺序来-----------------





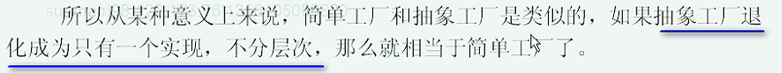
抽象工厂 是 选择产品族的实现

简单工厂 就是万能工厂 并不关心接口之间有什么样子的联系



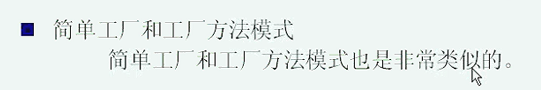
这个通常是产品族创建的 ----- 部件对象 为了构成某一个产品族需要的

某种意义上面来说 简单工厂 和 抽象工厂都是一样的

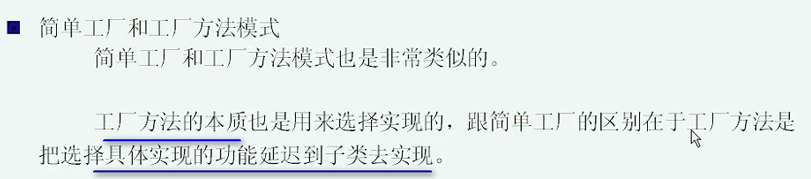


这个是简单工厂 和 抽象工厂之间的辨析

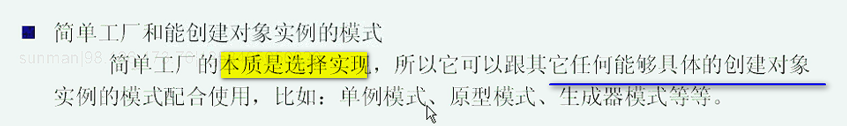
再看简单工厂和工厂方法的模式



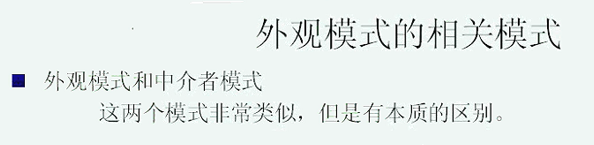
简单工厂是特殊的工厂方法







简单工厂仅仅是简单的选 不负责实现 ----- 所以 可以和任何具体的创建对象的实例模式进行联合使用 比如单例 原型 生成器模式 等等 ------ 比如你的对象通过单例 或者原型 或者生成器模式来的

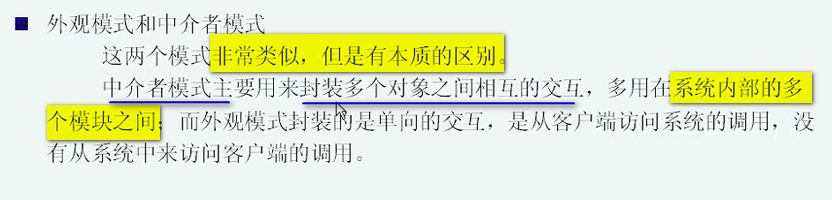


类似的原因





都有封装交互



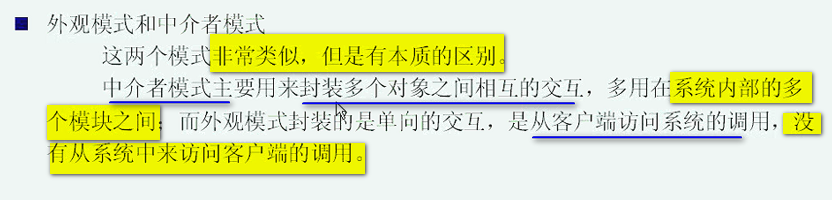
或者 一个系统内部的多个模块之间

外观是从客户端 到系统电泳 整个系统对外 ---- 客户端对系统使用的

外观模式是单向的

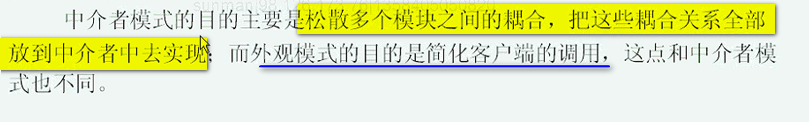
中介者模式在内部各个模块是平等

外观模式是单向的 但是 中介者模式应该是都可以从中介者访问对方



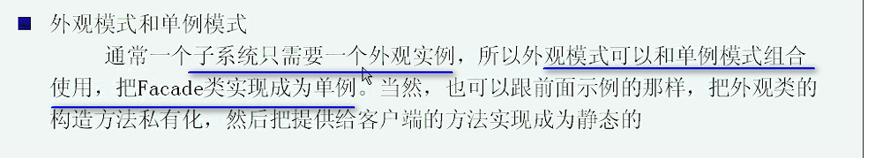
A做了一件事 连带让B或者C做啥 都在中介者模式

但是 外观模式 就是组合调用 转掉内部实现的功能 只是封装了一个高层接口 本身不负责实现这些业务功能 ---- 这个是有差别的



这个描述出来的内容不同 实现方式不同 本身的目的不同

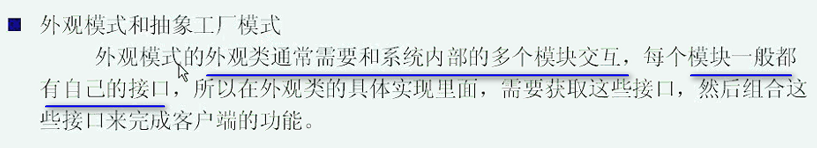
这两个模式虽然类似 但是 本质还是有区别的



这个时候 外观模式和单例模式组合使用

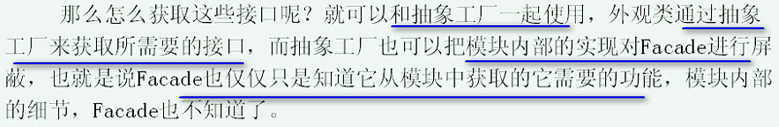
这都是可以的

外观模式和抽象工厂模式



通过工厂模式 获取抽象的接口 来实现客户所要求的功能

客户端找Façade Façade找具体模块的接口



直接在这里面想创建没有关联的接口 ----- 那就是简单工厂

如果想要创建有关联的接口 ---- 那就是抽象工厂

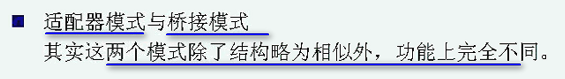
如果想要把创建放到子类中实现 ---- 那就使用工厂方法

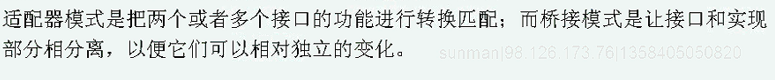
这就是模式在组合的时候 有很多不同的选择 需要对每个模式有足够的认识

这个就是外观模式的相关模式

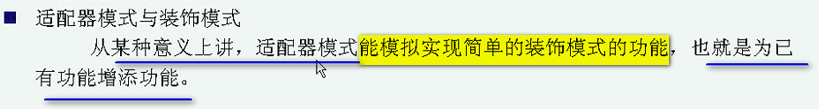


这两个模式 除了接口相似 功能 结构都不太相似



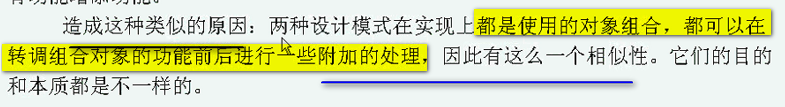


两者的目标完全不一样



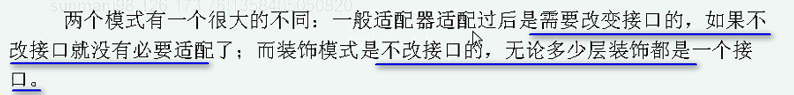
看起来是一样的

就是适配器能够为已有的功能添加功能



类似的这种模式 还有一个代理模式 转调真的被代理的对象 也可以增加功能

既然有这样的相似性 本质不一样

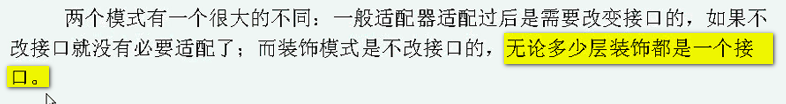


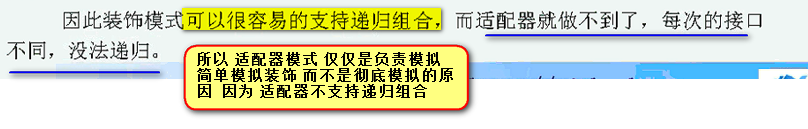
如果不改接口 就不用修改适配器了

装饰模式 是用来增加新功能

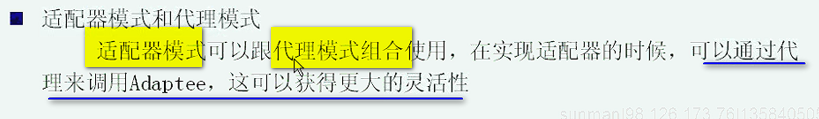
又要改接口 又要增加功能 ---- 那一定是适配器

如果在增加功能 增加的功能复杂 --- 那么就是适配器+装饰器

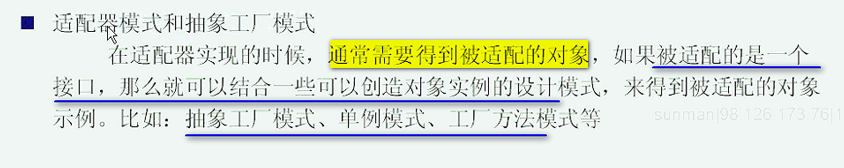


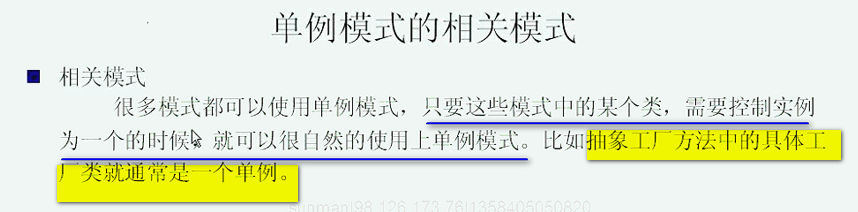


适配器和代理模式

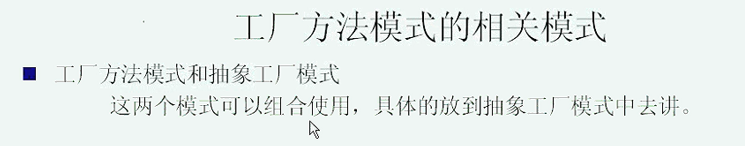


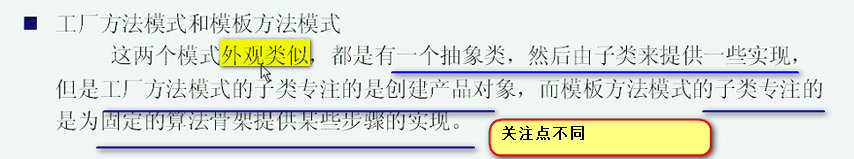
适配的时候 怎么能够获取被适配的对象？ ----- 代理也可以 这样 被适配的对象可以灵活的变化了



接下来的是 单例模式的相关模式  


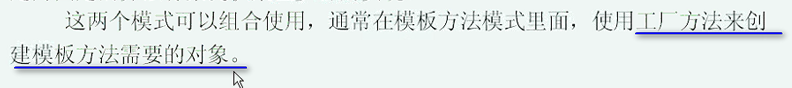
只要是控制实例的数目 都可以考虑使用单例模式

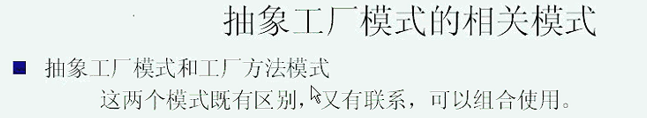


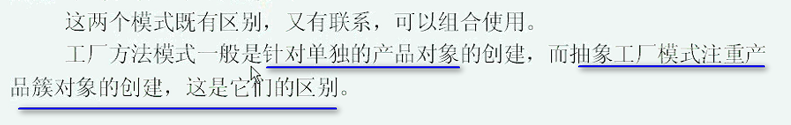


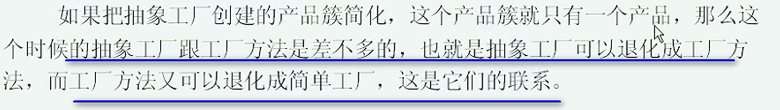
工厂方法模式的子类是专注创建对象

但是 模板方法模式的子类 专注 固定的孙发骨架提供步骤的实现

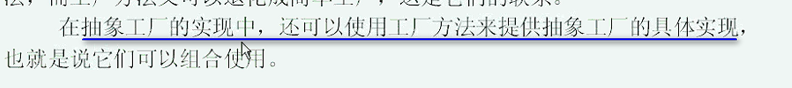


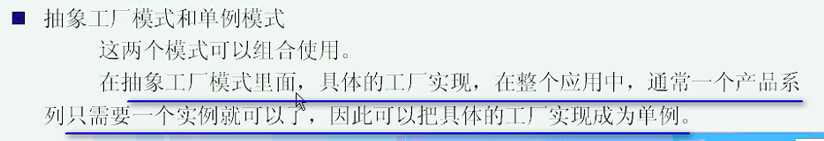




 各种退化

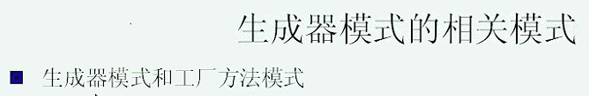
归根揭底 都由选择实现 ----- 简单是最原始的选择实现 工厂方法是延迟到子类选择实现 抽象工厂是产品族的选择实现



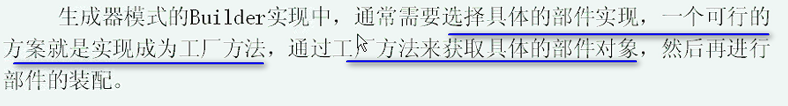


这就组合起来了

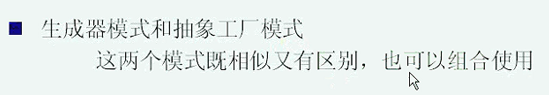
============



这两个模式是可以组合使用的

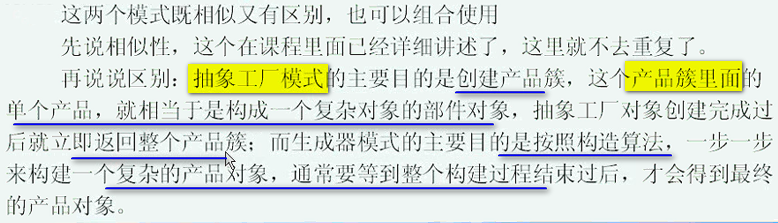


也可以简单工厂 这个都行 不一定非得是和工厂方法

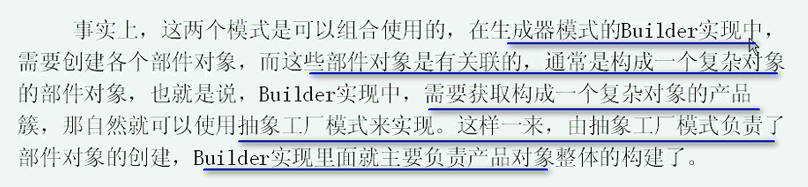


相似在课程中说了

抽象工厂目的是创建产品族



这两个可以组合使用



3.11