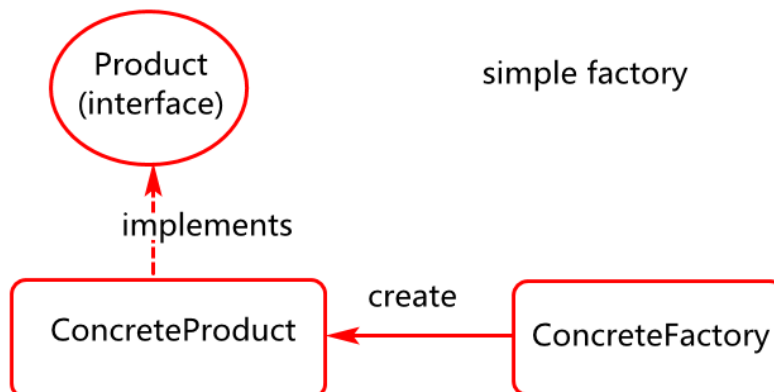


设计模式加强

1.1. 创建型模式

1.1. 简单工厂模式

- 1) 如何理解简单工厂模式?
 - a) 通过静态方法(相对比较多)或实例方法创建对象
 - b) 封装对象创建过程 (基于条件的不同创建不同的具体产品)
- 2) 简单工厂应用场景分析?
 - a) JDBC(DriverManager.getConnection(...))
 - b) Druid (getConnection())
 - c) Mybatis(Configuration,...)
- 3) 简单工厂中的对象角色?
 - a) 抽象产品对象 (例如 Executor,...)
 - b) 具体产品对象 (例如 SimpleExecutor,...)
 - c) 具体工厂对象 (例如 Configuration,...)



- 4) 简单工厂应用分析?
 - a) 优势: 解耦 (对象应用者与对象创建之间的耦合), 简单
 - b) 劣势: 可扩展性相对较差(创建产品的对象的工厂方法不够灵活)

1.2. 工厂方法模式 (Factory Method) ?

1) 如何理解工厂方法模式?

- a) 创建型模式 (负责创建对象)
- b) 工厂模式(平时所说的工厂模式就是工厂方法模式)
- c) 此模式的特点是基于抽象工厂扩展具体工厂然后创建产品对象。

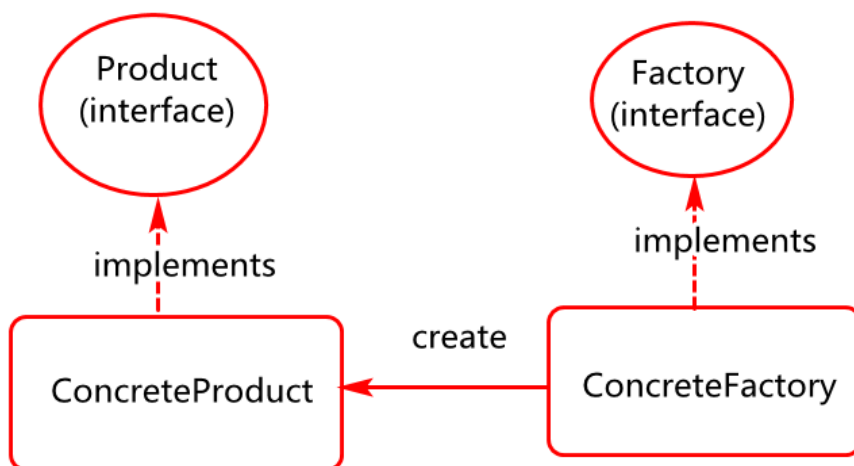
2) 工厂方法模式的应用场景分析?

- a) Mybatis (SqlSessionFactoryBean)
- b) Shiro (ShiroFilterFactoryBean)
- c) Spring(DataSourceFactory, TransactionFactory)
- d) ...

3) 工厂方法模式角色分析?

- a) 抽象产品(Product)
- b) 具体产品(ConcreteProduct)
- c) 抽象工厂(Factory)
- d) 具体工厂(ConcreteFactory)

Factory Pattern



例如:

- mybatis 中创建 SqlSession 对象
- ✓ SqlSession(抽象产品)
- ✓ DefaultSqlSession(具体产品)

- ✓ SqlSessionFactory(抽象工厂)
- ✓ DefaultSqlSessionFactory(具体工厂)
- Spring 整合 mybatis 时,SqlSessionFactory 对象过程分析。
- ✓ SqlSessionFactory (抽象产品)
- ✓ DefaultSqlSessionFactory(具体产品)
- ✓ FactoryBean(抽象工厂)
- ✓ SqlSessionFactoryBean(具体工厂)

4) 工厂方法模式应用分析?

- a) 优势: 相对于简单工厂更加灵活,更加适合创建具备等级结构(继承关系)的产品。
- b) 劣势: 假如每个抽象产品都对应一个具体工厂, 那么工厂类可能会比较多。

1.3. 抽象工厂(Abstract Factory)

1) 如何理解抽象工厂?

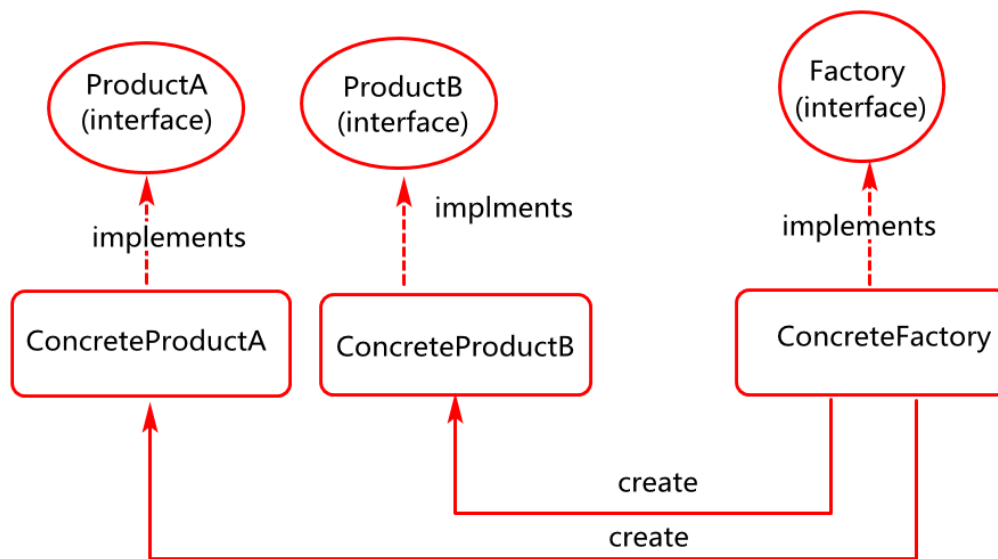
- a) 工厂方法模式用于创建具备一定等级结构的产品。
- b) 抽象工厂是多个工厂方法的综合应用, 因为它要同时创建多个具备一定等级结构的产品, 我们可以将这些产品理解产品族。

2) 抽象工厂的应用场景? (产品族→多个抽象产品)

- a) Spring(ClientHttpRequestFactory): 了解消息头, 消息体对象创建
- b) ...

3) 抽象工厂对象角色分析?

- a) 抽象产品(Product)
- b) 具体产品(ConcreteProduct)
- c) 抽象工厂(Factory)
- d) 具体工厂(ConcreteFactory)



例如：ClientHttpRequestFactory 抽象工厂的应用

- ✓ 抽象工厂 (ClientHttpRequestFactory)
- ✓ 具体工厂 (SimpleClientHttpRequestFactory)
- ✓ 抽象产品 (ClientHttpRequest, HttpHeaders,...)
- ✓ 具体产品 (SimpleStreamingClientHttpRequest,...)

4) 抽象工厂应用分析？

- a) 优势：工厂方法模式可能会产生很多个工厂类，基于此劣势借助抽象工厂创建产品族对象，可以减少工厂类对象的个数，从而更好节省资源。
- b) 劣势：一旦有新的产品族的诞生，这个抽象工厂扩展起来就会比较复杂。

1.4. 建造模式(Builder)

1) 如何理解建造模式？

建造模式又称之为构建模式，通常用于构建相对比较复杂对象，例如。

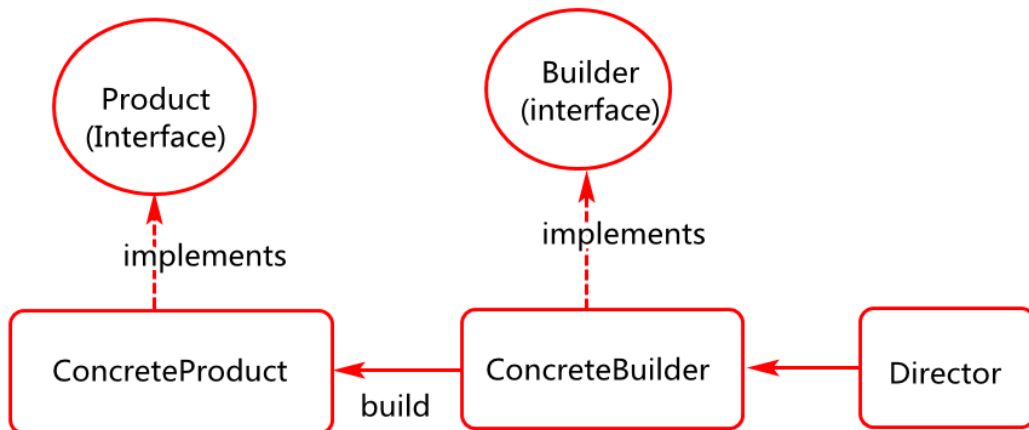
- a) 构建过程复杂。(除了复杂很多过程又重复比较多)
- b) 对象依赖关系复杂。

2) 建造模式应用场景分析？

- a) Mybatis(XmlConfigBuilder,XmlStatementBuilder)
- b) Spring(XmlBeanDefinitionReader)
- c) ...

3) 建造模式对象角色分析?

- a) 抽象建造对象角色(Builder):可能是接口,也可能是抽象类,此角色也可以省略。
- b) 具体建造对象角色(ConcreteBuilder)
- c) 导演角色(Director): 持有建造者对象,可以省略
- d) 抽象产品角色(Product) 抽象产品角色,可以省略
- e) 具体产品角色(ConcreteProduct) 具体产品角色



4) 建造模式应用分析?

- a) 优势:解耦对象的应用以及对象创建,通过建造者创建复杂产品对象,尤其是基于配置文件创建对象的场景。
- b) 劣势:要构建的对象结构假如频繁变化可能导致构建者对象的设计比较复杂。

1.5. 单例模式(Singleton)

1) 如何理解单例模式?

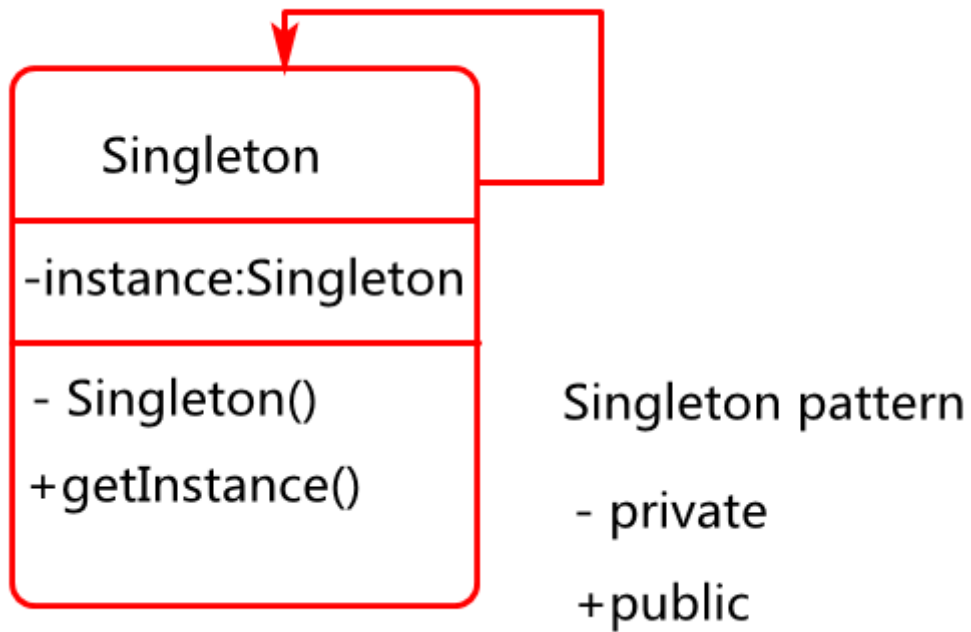
保证一个类的实例在“特定范围”只有一份(例如一个 JVM 内部,一个线程内部),并且提供一个全局访问点可以访问到这份实例。

2) 单例模式的应用场景?

- a) Spring(Singleton 作用域的 Bean 对象)
- b) MyBatis(ErrorContext 对象是每个线程一份此类实例)
- c) ...

3) 单例模式对象角色构成

a) 具体产品对象 (例如 Singleton)



4) 单例模式应用分析

- a) 优势：科学使用资源，避免频繁创建，销毁对象时造成的资源浪费。
- b) 劣势：设计不够严谨会存在线程安全问题，可扩展性相对较差。

2. 结构型模式

2.1. 适配器模式 (Adapter)

1. 如何理解适配器模式？

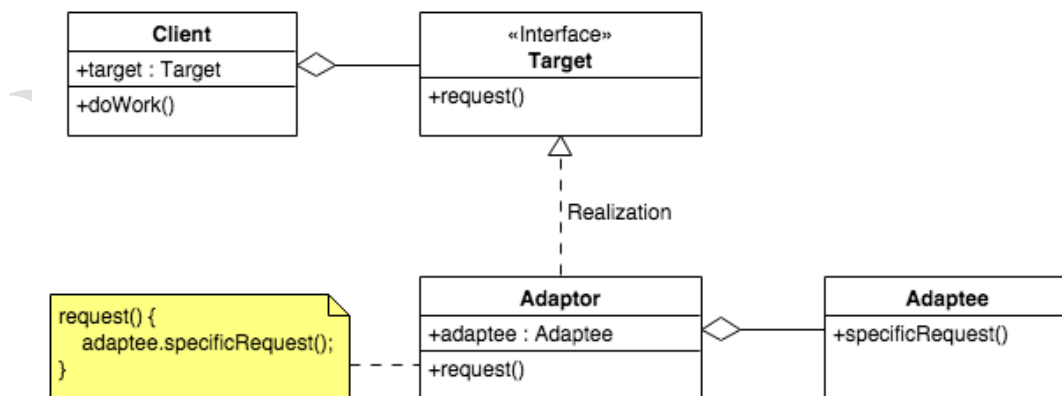
- 1) 适配器模式(Adapter Pattern) 将一个接口转换成客户希望的另一个接口，以解决接口不兼容问题。
- 2) 又名包装器(Wrapper)模式
- 3) 分为类适配器，对象适配器。

2. 适配器模式场景分析？

- 1) 生活中(一拖三充电头、HDMI 转 VGA)
- 2) mybatis (Log 接口)
- 3) spring (HandlerAdapter, AdvisorAdapter)

3. 适配器模式对象角色构成？

- 1) Target(目标抽象类)：抽象类或接口，也可以是具体类
- 2) Adapter(适配器类)：负责对 Adaptee 和 Target 进行适配。
- 3) Adaptee(适配者类)：适配者即被适配的角色。

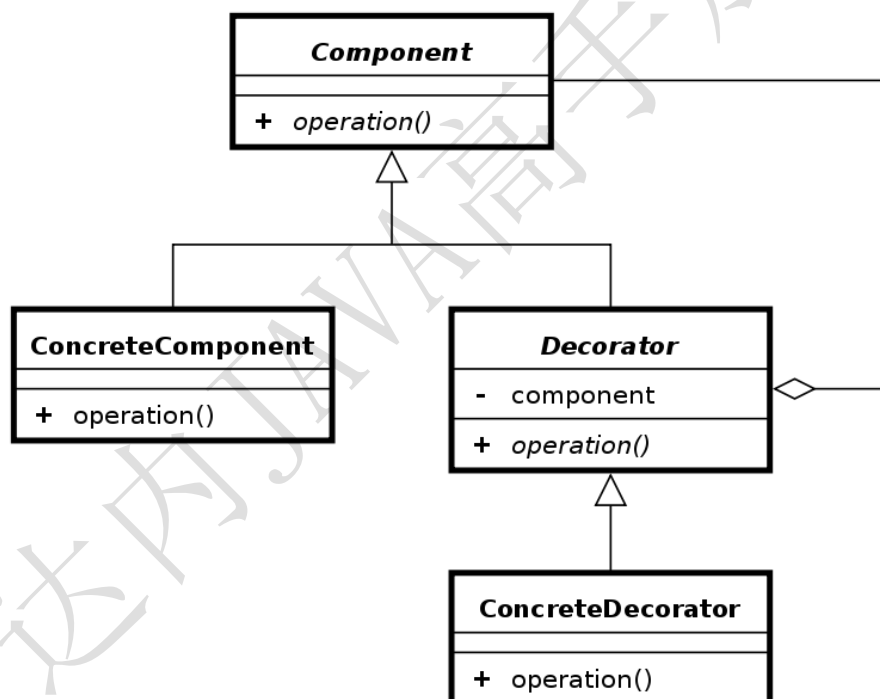


4. 适配器模式应用分析？

- 1) 优势：对客户端透明，扩展性好，复用性好
- 2) 劣势：是一种补偿机制的实现，主要用于后期扩展。

2.2. 装饰模式 (Decorator)

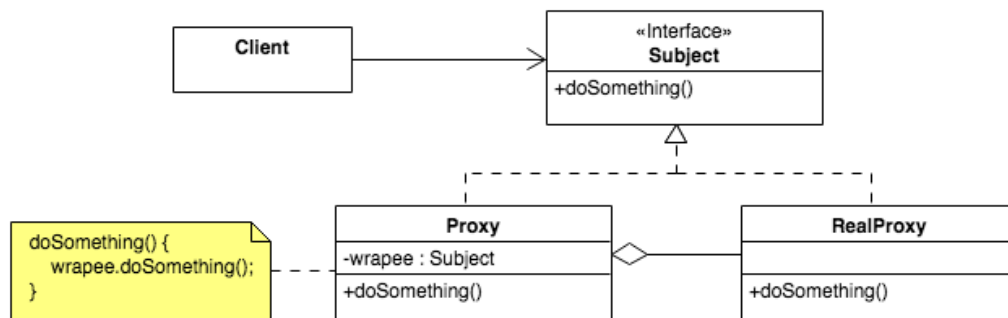
1. 如何理解装饰模式?
 - 1) 油漆工模式
 - 2) 基于目标对象添加额外职责 (功能扩展)
2. 装饰模式场景分析?
 - 1) mybatis(Executor)
 - 2) spring()
3. 装饰模式角色构成?
 - 1) Component: 抽象构件
 - 2) ConcreteComponent: 具体构件
 - 3) Decorator: 抽象装饰类
 - 4) ConcreteDecorator: 具体装饰类



4. 装饰模式应用分析?
 - 1) 提供了相对继承更加灵活的功能扩展方式
 - 2) 实现相对复杂。

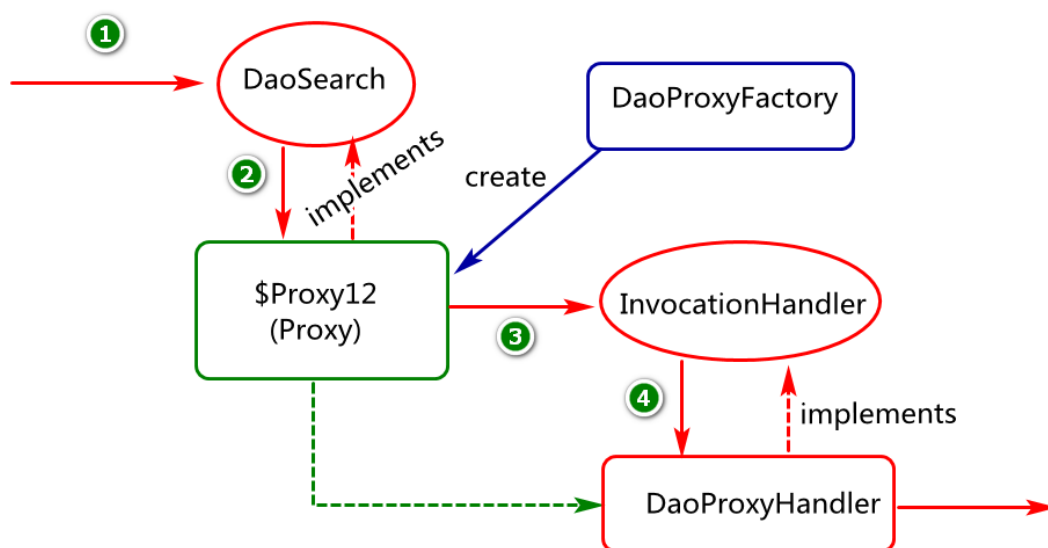
4.1.1. 代理模式 (Adapter)

1. 如何理解代理模式？
 - 5) 基于目标对象创建代理对象，并由代理对象控制目标对象的执行。
 - 6) 基于 OCP 原则扩展目标对象的功能。
2. 代理模式场景分析？
 - 1) mybatis (为接口创建代理对象，拦截器应用)
 - 2) spring (AOP)
3. 代理模式角色构成？
 - 1) Subject: 抽象主题角色
 - 2) Proxy: 代理主题角色
 - 3) RealSubject: 真实主题角色

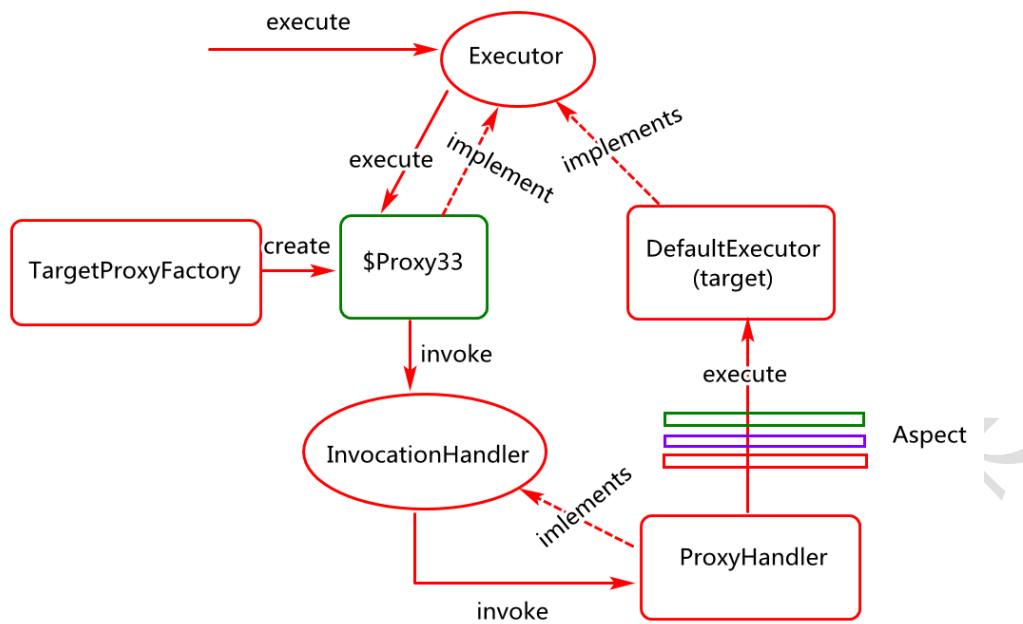


强化分析

- 1) MyBatis 中 mapper 接口的代理



- 2) JDK 动态代理 (Spring AOP 思想)



4. 代理模式应用分析？

- 1) 优势：基于 OCP 进行控制，扩展目标对象
- 2) 劣势：由于代理对象的创建可能会导致性能上的缺陷。

4.2. 行为模式

4.2.1. 策略模式(Strategy)

5. 如何理解策略模式？
6. 策略模式场景分析？
7. 策略模式角色构成？
8. 策略模式应用分析？

8.1.1. 模板方法模式(Template Method)

1. 如何理解模板方法模式？
2. 模板方法模式场景分析？
3. 模板方法模式角色构成？

4. 模板方法模式应用分析？

4.1.1. 观察者模式(Template Method)

1. 如何理解代理模式？
2. 代理模式场景分析？
3. 代理模式角色构成？
4. 代理模式应用分析？

达内JAVA高手加薪课