1. 适配器模式

类比插座，有欧洲标准，中国标准，为了在欧洲使用中国的插头，所以需要一个适配器，做到兼容

在正常开发中，一般是针对一个已有的类，想要用它的功能，由于各种原因，不能直接用，需要写一个中间类，自己调用这个中间类，由中间类去调用已有的类

优点：

1. 通过适配器，客户端可以直接调用同一接口，因而对客户端来说是透明的。这样做更简单，更直接，更紧凑。
2. 复用了现存的类，解决了现存类和复用环境要求不一致的问题。
3. 将目标类和适配者类解耦，通过引入一个适配器现有的适配者类，而无须修改原有代码。

一个对象适配器可以把多个不同的适配器类适配到同一个目标，也就是说，同一个适配器可以把适配者类和他的子类都适配到目标接口。

缺点：

1. 对于适配器来说，更换适配器的实现过程比较复杂　。

2. 代理模式

优点:

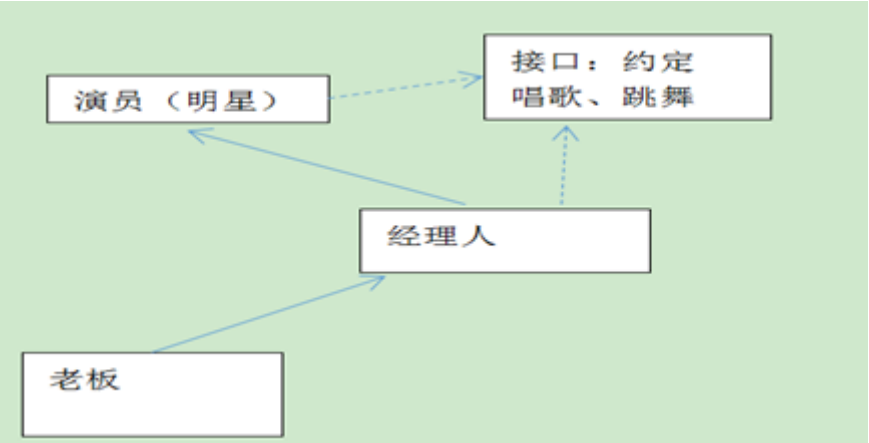
1. 动态代理采用在运行时动态生成代码的方式，取消了对被代理类的扩展限制，遵循了开闭原则
2. 若动态代理要对目标类的增强逻辑进行扩展，结合策略模式，只需要新增策略类便可完成，无需修改代理类的代码
3. 在编码时，代理逻辑与业务逻辑是互相独立的，没有耦合

> https://blog.csdn.net/wb_snail/article/details/80632038

2.1. 静态代理

定义：为其他对象提供一种代理以控制对这个对象的访问。在某些情况下，一个对象不适合或者不能直接引用另一个对象，而代理对象可以在客户端和目标对象之间起到中介的作用。

作用：增强一个类中的某个方法.对程序进行扩展，Spring框架中AOP。



2.2 动态代理

1、动态代理：（它与装饰者模式有点相似，它比装饰者模式还要灵活）

- 动态代理它可以直接给某一个目标(被代理 对象)对象(实现了某个或者某些接口)生成一个代理对象，而不需要代理类存在，如上图中经理人需要存在。
- 动态代理与代理模式原理是一样的，只是它没有具体的代理类，直接通过反射生成了一个代理对象

2、动态代理的分类

- jdk提供一个Proxy类可以直接给实现接口类的对象直接生成代理对象 API：调用，缺少什么参数,传什么
- spring中动态代理:cglib 继承

3. 单例模式

一个类能返回对象一个引用(永远是同一个)和一个获得该实例的方法（必须是静态方法，通常使用getInstance这个名 称）；当我们调用这个方法时，如果类持有的引用不为空就返回这个引用，如果类保持的引用为空就创建该类的实例并将实例的引用赋予该类保持的引用；同时我们 还将该类的构造函数定义为私有方法，这样其他处的代码就无法通过调用该类的构造函数来实例化该类的对象，只有通过该类提供的静态方法来得到该类的唯一实例。

优点：

1.在单例模式中，活动的单例只有一个实例，对单例类的所有实例化得到的都是相同的一个实例。这样就 防止其它对象对自己的实例化，确保所有的对象都访问一个实例

2.单例模式具有一定的伸缩性，类自己来控制实例化进程，类就在改变实例化进程上有相应的伸缩性。

3.提供了对唯一实例的受控访问。

4.由于在系统内存中只存在一个对象，因此可以 节约系统资源，当 需要频繁创建和销毁的对象时单例模式无疑可以提高系统的性能。

5.允许可变数目的实例。

6.避免对共享资源的多重占用。

缺点：

1.不适用于变化的对象，如果同一类型的对象总是要在不同的用例场景发生变化，单例就会引起数据的错误，不能保存彼此的状态。

2.由于单利模式中没有抽象层，因此单例类的扩展有很大的困难。

3.单例类的职责过重，在一定程度上违背了“单一职责原则”。

4.滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢出；如果实例化的对象长时间不被利用，系统会认为是垃圾而被回收，这将导致对象状态的丢失

适用场景：

单例模式只允许创建一个对象，因此节省内存，加快对象访问速度，因此对象需要被公用的场合适合使用，如多个模块使用同一个数据源连接对象等等。如：

1.需要频繁实例化然后销毁的对象。

2.创建对象时耗时过多或者耗资源过多，但又经常用到的对象。

3.有状态的工具类对象。

4.频繁访问数据库或文件的对象。

以下都是单例模式的经典使用场景：

1.资源共享的情况下，避免由于资源操作时导致的性能或损耗等。如上述中的日志文件，应用配置。

2.控制资源的情况下，方便资源之间的互相通信。如线程池等。