**静态代理与动态代理的区别：**

1.代理即一个人或者一个机构代表另一个人或者另一个机构采取行动。

在一些情况下，一个客户不想或者不能够直接引用一个对象，而代理对象

则可以在客户端和目标对象之间起到中介的作用。

代理对象就是把被代理对象包装一层，在其内部做一些额外的工作，比如用户需要

登录facebook,而普通网络无法直接访问，网络代理帮助用户先翻墙，然后再访问facebook。这即是代理的作用。纵观静态代理与动态代理，它们都能实现相同的功能，而我们看从静态代理到动态代理的这个过程，我们会发现其实动态代理只是对类做了进一步抽象和封装，使其复用性和易用性得到进一步提升，而这不仅仅符合了面向对象的设计理念，其中还有AOP的身影，这也提供给我们对类抽象的一种参考。关于动态代理与AOP（面向切面编程）的关系，个人觉得AOP是一种思想，而动态代理是一种AOP思想的实现。(将日志记录，性能统计，安全控制，事务处理，异常处理等代码从业务逻辑代码中划分出来，通过对这些行为的分离，我们希望可以将它们独立到非指导业务逻辑的方法中，进而改变这些行为的时候不影响业务逻辑的代码也就是解耦)。

2.代理是一种常用的设计模式，其目的就是为其他对象提供一个代理以控制对某个对象的访问。代理类负责为委托类预处理消息，过滤消息并转发消息，以及进行消息被委托类执行后的后续处理。为了保持行为的一致性，代理类和委托类通常会实现相同的接口，所以在访问者看来两者没有丝毫的区别。通过代理类这中间一层，能有效控制对委托类对象的直接访问，也可以很好地隐藏和保护委托类对象，同时也为实施不同控制策略预留了空间，从而在设计上获得了更大的灵活性。更通俗的说，代理解决的问题当两个类需要通信时，引入第三方代理类，将两个类的关系解耦，让我们只了解代理类即可，而且代理的出现还可以让我们完成与另一个类之间的关系的统一管理，但是切记，代理类和委托类要实现相同的接口，因为代理真正调用的还是委托类的方法。代理类可以委托类预处理消息，把消息转发给委托类和事后处理消息等。

1. 按照代理的创建时期，代理可分为静态代理与动态代理：

静态：（1）由程序员创建代理类或特定工具自动生成源代码再对其编译。在程序运行前代理类的.class文件就已经存在了。a.优点：代理使客户端不需要知道实现类是什么，怎么做的，而客户端只需知道代理即可（解耦合）.

b.缺点:代理类和委托类实现了相同的接口，代理类通过委托类实现了相同的方法。这样就出现了大量的代码重复。如果接口增加一个方法，除了所有实现类需要实现这个方法外，所有代理类也需要实现此方法。增加了代码维护的复杂度。代理对象只服务于一种类型的对象，如果要服务多类型的对象。势必要为每一种对象都进行代理，静态代理在程序规模稍大时就无法胜任了。c:由程序员自己编写创建或由特定工具自动生成源代码，编写创建代理类，然后再进行编译，在程序运行前，代理类的.class文件就已存在。

（2）代理可以对实现类进行统一的管理，如在调用具体实现类之前，需要打印日志等信息，这样我们只需要添加一个代理类，在代理类中添加打印日志的功能，然后调用实现类，这样就避免了修改具体实现类。满足我们所说的开闭原则。但是如果想让每个实现类都添加打印日志的功能的话，就需要添加多个代理类，以及代理类中各个方法都需要添加打印日志功能即静态代理类只能为特定的接口(Service)服务。如想要为多个接口服务则需要建立很多个代理类，因为一个代理类能为一个接口，代理一种类型（在编译器就已确定被代理对象）。

4:newProxyClass()静态方法创建动态代理类

publicstatic Class<?> getProxyClass(ClassLoader loader, Class<?>[] interfaces) throws IllegalArgumentException  
参数loader：指定动态代理类的类加载器，参数interfaces指定：动态代理类需要实现的所有接口。

newProxyInstance()静态方法负责创建动态代理类的实例，它的完整定义如下：

public static Object newProxyInstance(ClassLoader loader, Class<?>[] interfaces, InvocationHandler handler) throws

IllegalArgumentException

参数loader 指定动态代理类的类加载器，参数interfaces 指定动态代理类需要实现的所有接口，参数handler 指定与动态代理类关联的 InvocationHandler 对象。

动态：在程序运行时运用反射机制动态创建而成。

动态代理与静态代理相比较，最大的好处是接口中声明的所有方法都被转移到调用处理器一个集中的方法中处理（InvocationHandler.invoke）。这样，在接口方法数量比较多的时候，我们可以进行灵活处理，而不需要像静态代理那样每一个方法进行中转。而且动态代理的应用使我们的类职责更加单一，复用性更强。

动态代理类是public、final和非抽象类型的；  
动态代理类继承了java.lang.reflect.Proxy类；  
动态代理类的名字以“$Proxy”开头；  
动态代理类实现getProxyClass()和newProxyInstance()方法中参数interfaces指定的所有接口；

动态代理在运行时，通过反射机制实现动态代理，并且能够代理各种类型的对象。因为Java反射机制可生成任意类型的动态代理类。在实现阶段不用关心代理谁，而在运行阶段通过反射机制才指定代理哪一个对象。

动态代理类不仅简化了编程工作，而且提高了软件系统的可扩展性。