1. 代理模式的原理：

使用一個代理將對象包裝起來,然後用該代理對象取代原始對象。任何對原始對象的調用都要通過代理。代理對象決定是否以及何時將方法調用轉到原始對象上 。

1. 靜態代理

2.1舉例：

實現Runnable接口的方法創建多線程。

Class MyThread implemens Runnable{ } //相當於被代理類

Class Thread implemens Runnable{ } //相當於代理類

Main{

MyThread t = new MyThread( );

Thread thread = new Thread( );

Thread.start( ); //啟動線程; 調用線程的run( )

}

2.2靜態代理的缺點：

1. 靜態代理，特征是代理類和目標對象的類都是在編譯期間確定下來，不利於程序的擴展。
2. 每一個代理類只能為一個接口服務，這樣一來程序開發中必然產生過多的代理。
3. 動態代理的特點：

動態代理是指客戶通過代理類來調用其它對象的方法，並且是在程序運行時根據需要動態創建目標類的代理對象。

4. 動態代理的實現：

4.1 需要解決的兩個主要問題：

問題一：如何根據加載到內存中的被代理類，動態的創建一個代理類及其對象。

(通過Proxy.newProxyInstance()實現)  
問題二：當通過代理類的對象調用方法a時，如何動態的去調用被代理類中的同名方法a。

(通過InvocationHandler接口的實現類及其方法Invoke())

4.2 代碼實現

/\*  
 \* 動態代理的舉例  
 \* @author shkstart  
 \* @create 2019 上午 10:18  
 \*/  
interface Human{  
 String getBelief();  
 void eat(String food);  
}  
//被代理類  
class SuperMan implements Human{  
 @Override  
 public String getBelief() {  
 return "I believe I can fly!";  
 }  
 @Override  
 public void eat(String food) {  
 System.out.println("我喜歡吃" + food);  
 }  
}  
class HumanUtil{  
 public void method1(){  
 System.out.println("====================通用方法一====================");  
 }  
 public void method2(){  
 System.out.println("====================通用方法二====================");  
 }  
}  
class ProxyFactory{  
 //調用此方法，返回一個代理類的對象。解決問題一  
 public static Object getProxyInstance(Object obj){//obj:被代理類的對象  
 MyInvocationHandler handler = new MyInvocationHandler();  
 handler.bind(obj);  
 return Proxy.newProxyInstance(obj.getClass().getClassLoader(),obj.getClass().getInterfaces(),handler);  
 }  
}  
  
class MyInvocationHandler implements InvocationHandler{  
 private Object obj;//需要使用被代理類的對象進行賦值  
 public void bind(Object obj){  
 this.obj = obj;  
 }  
 //當我們通過代理類的對象，調用方法a時，就會自動的調用如下的方法：invoke()  
 //將被代理類要執行的方法a的功能就聲明在invoke()中  
 @Override  
 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
 HumanUtil util = new HumanUtil();  
 util.method1();  
 //method:即為代理類對象調用的方法，此方法也就作為了被代理類對象要調用的方法  
 //obj:被代理類的對象  
 Object returnValue = method.invoke(obj,args);  
 util.method2();  
 //上述方法的返回值就作為當前類中的invoke()的返回值。  
 return returnValue;  
 }  
}  
public class ProxyTest {  
 public static void main(String[] args) {  
 SuperMan superMan = new SuperMan();  
 //proxyInstance:代理類的對象  
 Human proxyInstance = (Human) ProxyFactory.getProxyInstance(superMan);  
 //當通過代理類對象調用方法時，會自動的調用被代理類中同名的方法  
 String belief = proxyInstance.getBelief();  
 System.out.println(belief);  
 proxyInstance.eat("四川麻辣燙");  
  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
  
 NikeClothFactory nikeClothFactory = new NikeClothFactory();  
  
 ClothFactory proxyClothFactory = (ClothFactory) ProxyFactory.getProxyInstance(nikeClothFactory);  
 proxyClothFactory.produceCloth();  
 }  
}