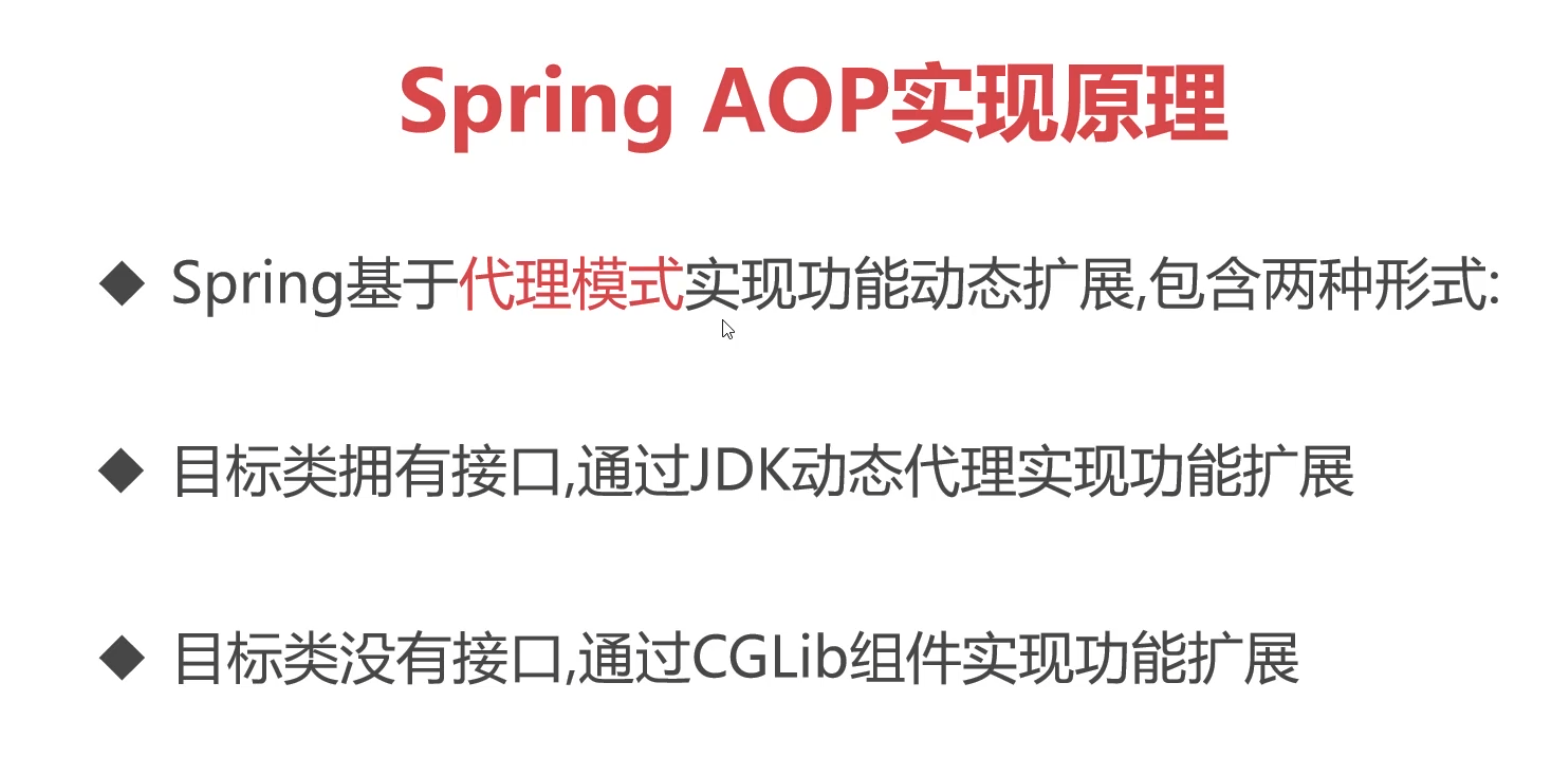
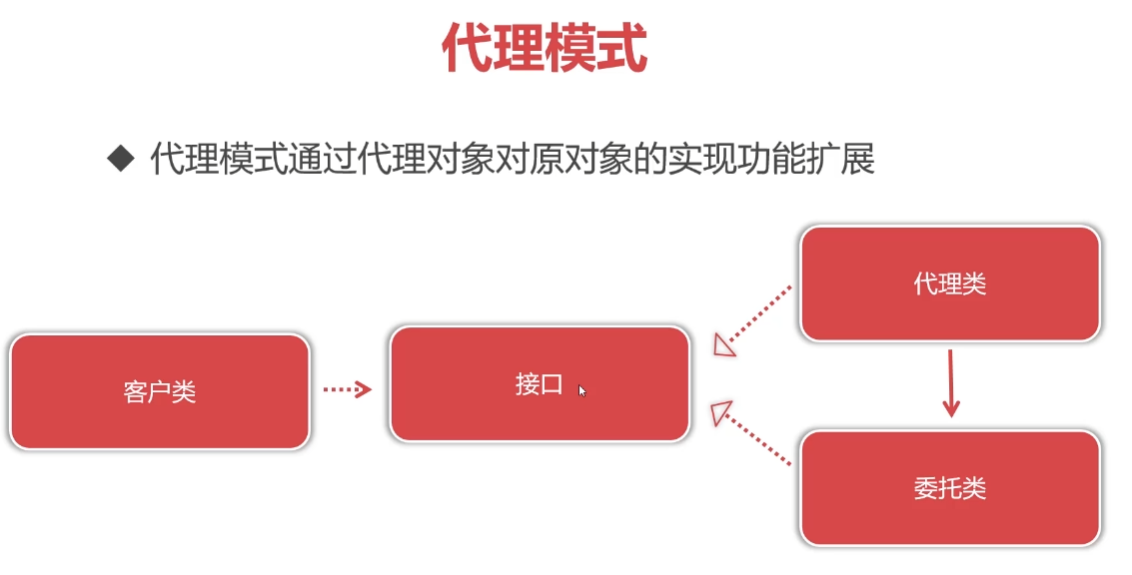
AOP底層實現原理



一、代理模式

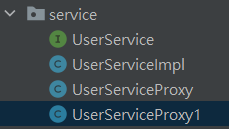


正常狀態下，客戶端是與委託類直接接觸的，例如:租客(客戶類)自己去找房東(委託類)，但這樣效率很低­。

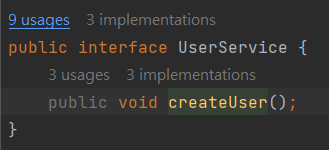
代理模式就是讓租客(客戶類)透過一個房屋仲介系統(接口)，讓租客和接口互動，由房仲(代理類)和房東(委託類)來實作該接口，而房仲必須持有委託類，就像房仲持有房間的鑰匙。透過房仲(代理類)代理房東原本要做的事，並且從中增加其他的業務，例如:帶看租金，等等的其他功能。

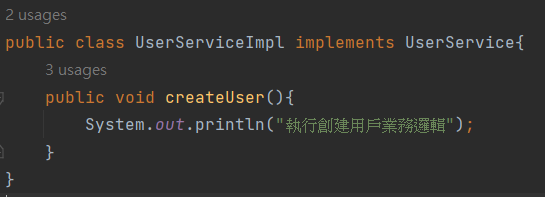
二、靜態代理

透過創建代理類並持有委託類，在提供委託類功能時，加入額外的代理功能。

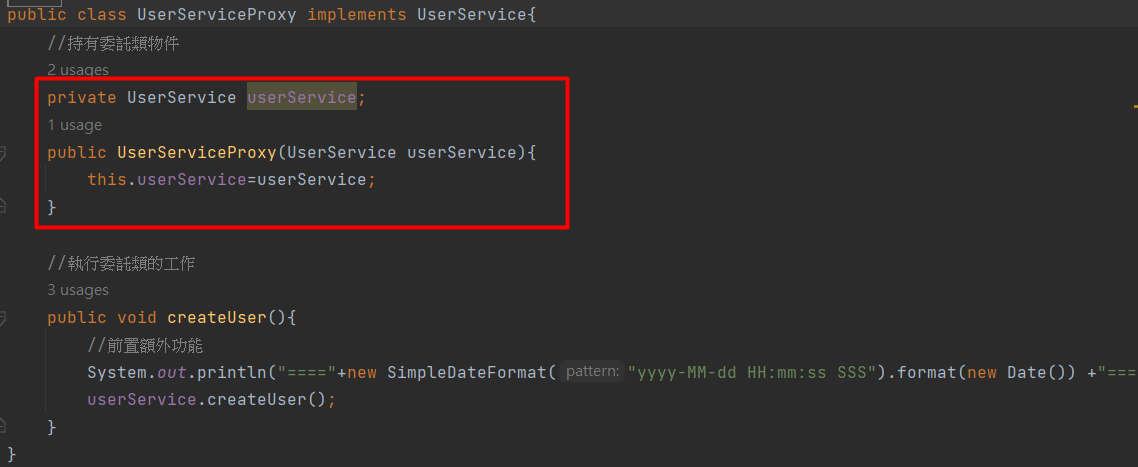


首先UserService接口提供創建使用者的功能，然後UserServiceImpl類實作該功能。

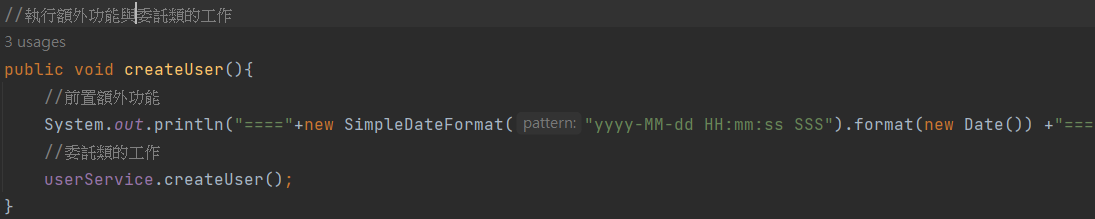




再來創建代理類UserServiceProxy實作UserService街口，並且持有委託類也就是UserService類。

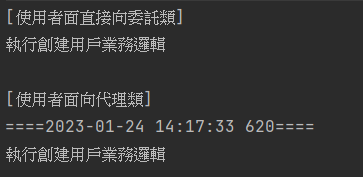


然後在原有的實作上，增加代理類要加入的功能，例如在執行委託類功能前，前置一個代理類的功能。

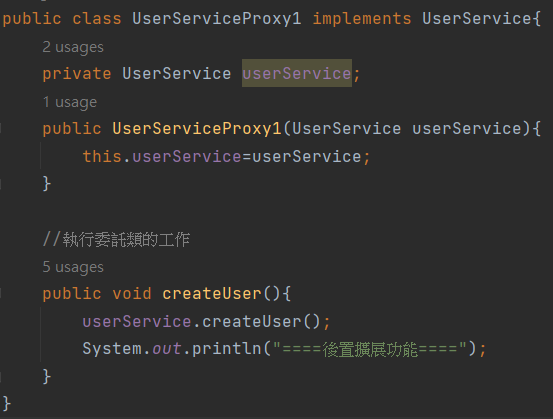


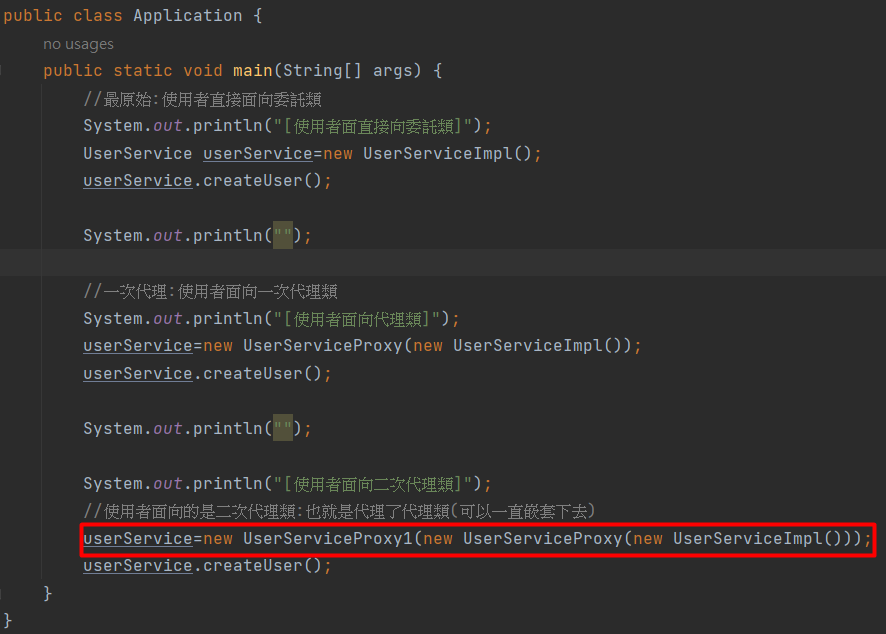
客戶在使用UserService時，面對的物件就從UserServiceImpl類變成了UserServiceProxy類了。

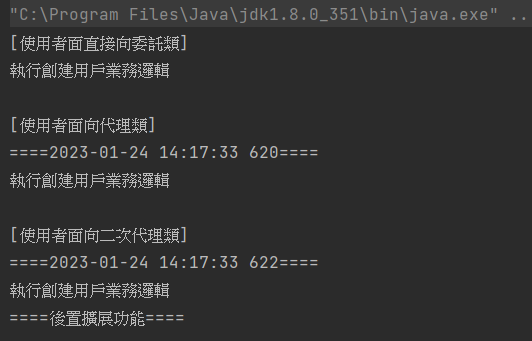




由於Java接口的特性，我們可以嵌套無限多層代理物件，只要代理物件都持有委託類並實作接口。如下:二次代理先代理了一次代理的工作，再加入二次代理的額外功能。





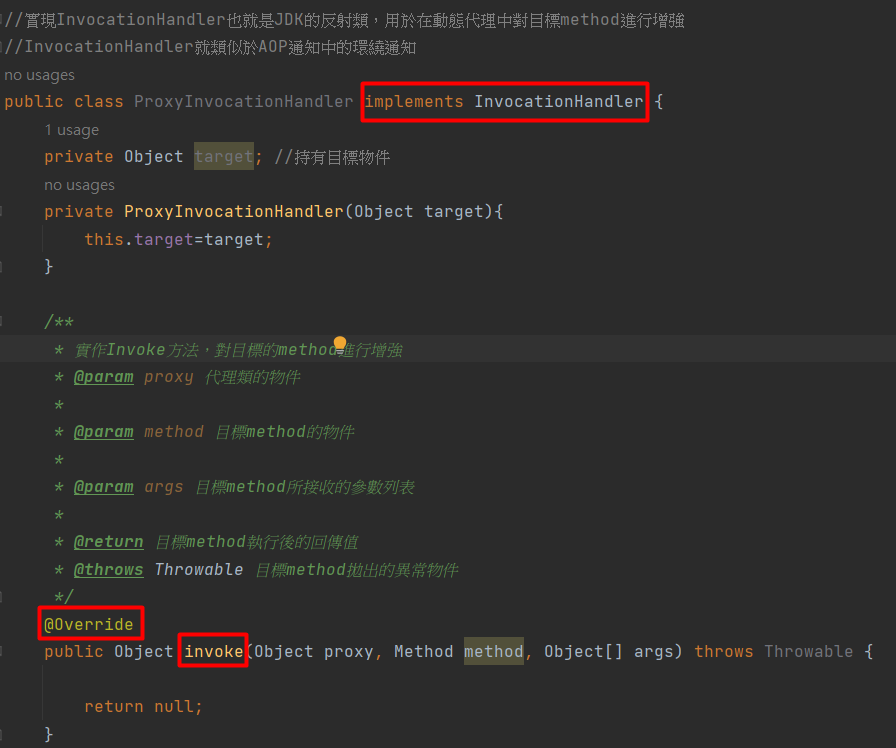


靜態代理的缺點就是每一層的代理都要重新創建新的代理類並手動的創建代理對象。而使用JDK的反射機制，可以做到動態代理的功能。

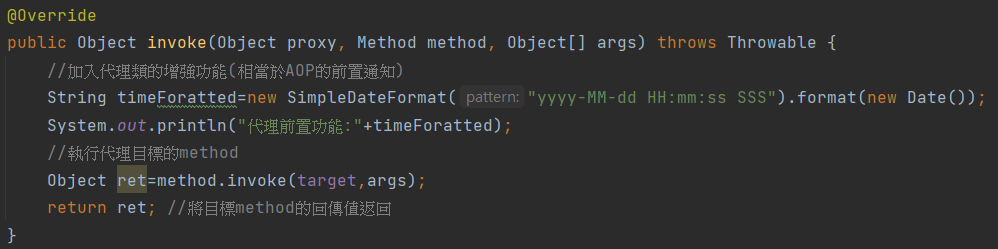
三、動態代理

透過JDK反射機制，自動生成實現接口的代理類。

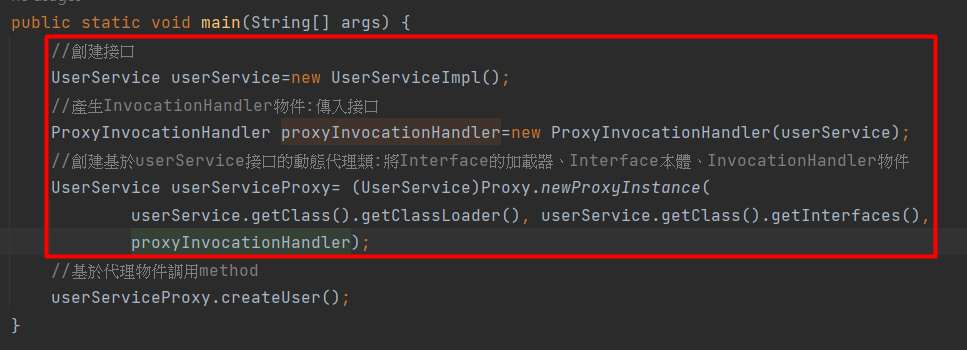
透過創建自動代理類並實作JDK中的InvocationHandler接口中的invoke方法。當然，也必須持有要代理的委託物件target。



接著將增強的內容寫入至invoke中，對目標method進行增強。

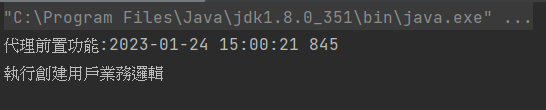


最後創建動態代理的物件

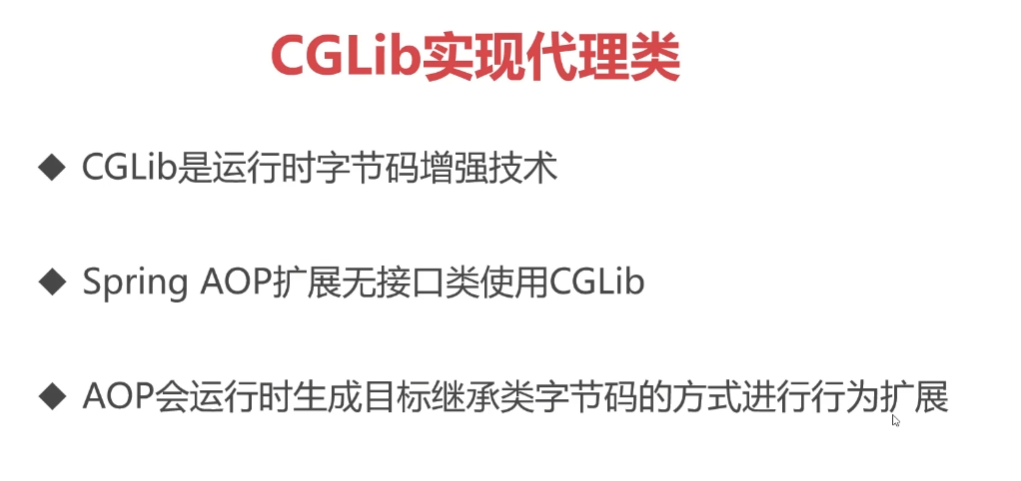


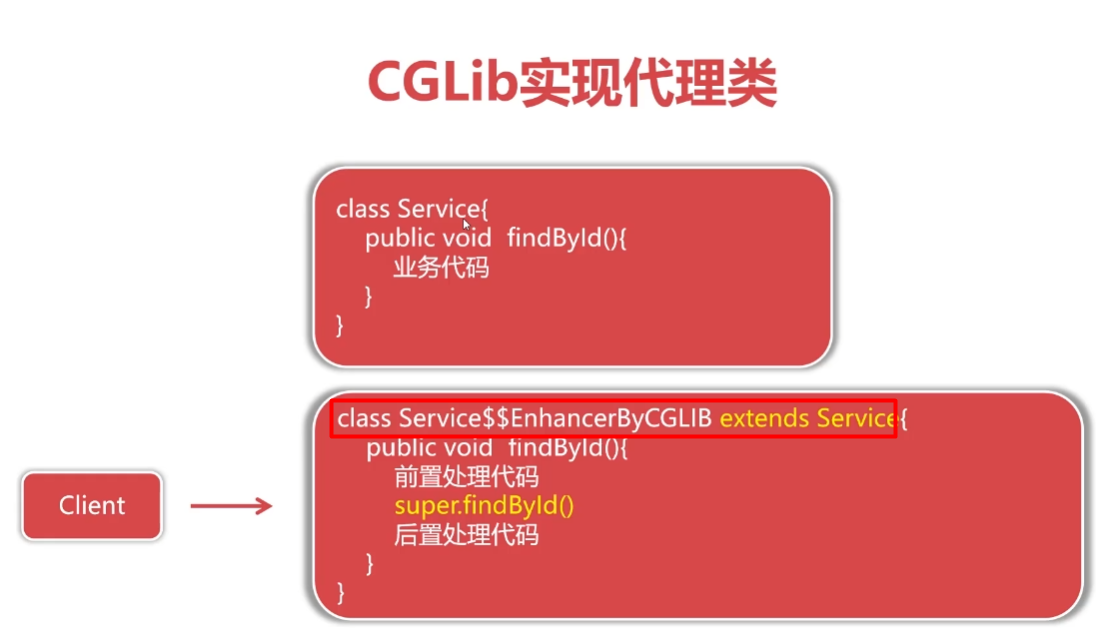
最後基於代理物件調用method





五、Spring AOP底層實現原理



CGLib實現代理類的邏輯示意圖

簡單來說，CGLib會自動產生一個子類，繼承目標類，並實作目標method的增強功能。

五、Spring AOP底層實現原理

(1)如果目標類有實現接口:使用JDK反射技術實現動態代理，進行AOP實現。

(2)如果目標類沒有實現接口:則使用CGLib對目標類自動生成繼承類，並對目標method進行擴展，實現AOP。