**总结：**

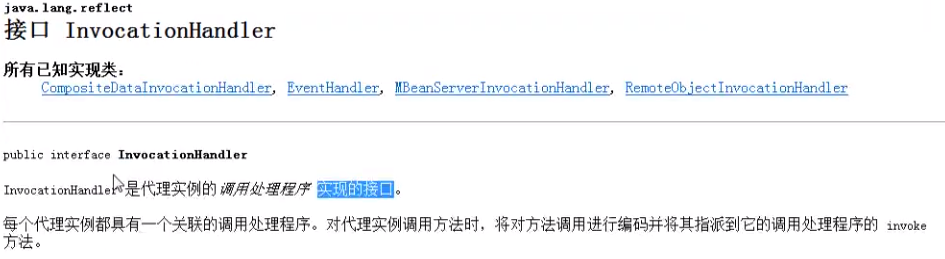
静态代理就是，房东只负责租房，中介做租房、收费的工作，客户。房东跟中介都实现了租房这个接口。在这个接口与客户进行附加的工作。客户与房东不接触。无关联。（但需要每一个都要写一个中介，麻烦）

适用：有公共的代码如日志打印，性能统计等不重要的公共业务（非功能性）。

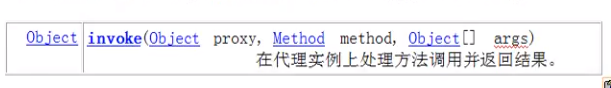
动态代理：

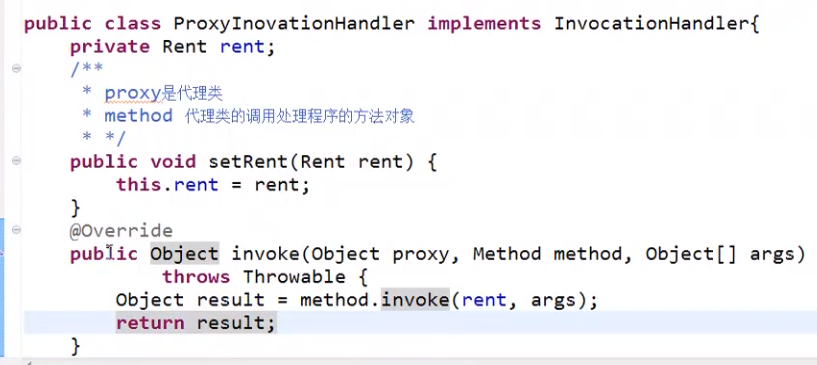
1. 可以动态设置‘目标’这个接口。
2. 然后根据该目标接口，以get的方式拿到动态生成的代理类对象（反射包下的代理类的静态方法，在调用该静态方法的时候，第三个参数将代理对象与handler进行了关联。）。
3. 该对象调用目标接口的方法的时候，将在调用处理程序上调用invoke方法。在invoke方法中实现目标接口方法以及附加操作。
4. （注：InvocationHandler是代理实例的调用处理程序实现的接口，每一个代理实例都有一个关联的调用处理程序，也就是说当该对象调用方法时，将对调用方法进行编码并将其指派到它的调用处理程序的invoke方法，可以简单理解为：将代理实例跟调用的方法作为参数传进去invoke）
5. （代理对象是在jvm运行时动态生成的一个对象，它并不是我们的InvocationHandler类型，也不是我们定义的那组接口的类型，而是在运行是动态生成的一个对象）

**动态代理**：--------------------------------------------------------------------------------------###############

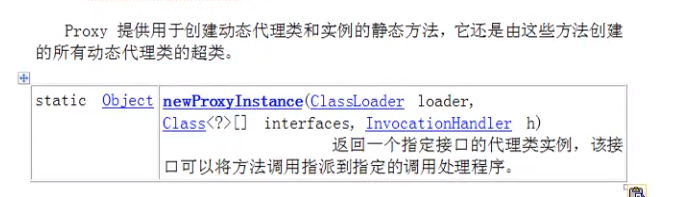


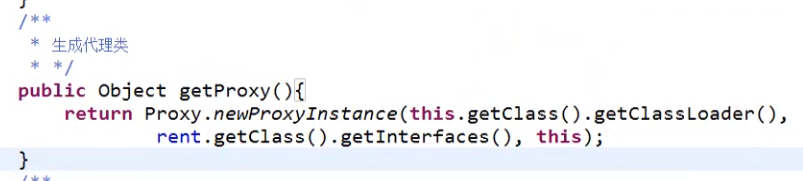
代理实例就是proxy对象。调用处理程序Handler，就是实际在处理业务的函数。也就是实际的业务处理函数就是重写实现了IncocationHandler接口。重写的就是invoke方法。也就是在该方法里来写这些实际处理业务代码。（proxy这个是生成的代理类对象，method利用反射来执行真实对象的函数。）



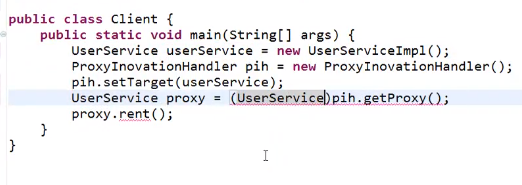


接着利用proxy这个类来创建动态代理类，由于动态生成了，因此需要类加载器来将其加载到jvm虚拟机里面。第三个参数handler， 我们这里将这个代理对象关联到了上方的 InvocationHandler 这个对象上



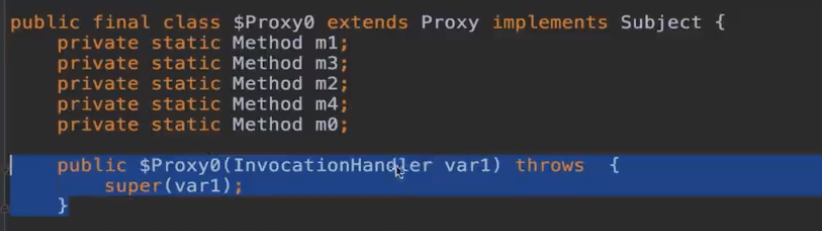


使用



可以查看生成的代理类（https://www.imooc.com/learn/869）：

在构造函数中传入handler



之后实现对应的接口的方法，在里面调用我们重写的invoke方法：

