链滴

关于 jdkDynamicProxy 和 cglib 的使用

作者：Ouyuone
原文链接：https://ld246.com/article/1571132958193
来源网站：链滴

一. 使用jdk动态代理是需要建立在接口的实现类上面(就是说需要提供接口和实现类)

如下定义一个接口jdkDynamic

```
/**
 * <pre>
 * @Auther: ousakai
 * @Date: 2019-10-15 15:23
 * @Description:
 * 修改版本: 1.0
 * 修改日期:
 * 修改人 :
 * 修改说明: 初步完成
 * 复审人 :
 * </pre>
 */
public interface JdkDynamic {
    default public void lookUp(){
        System.out.println("我要去看看, 能找到什么");
    };
}
```

在定义一个实现类来实现它

```
/**
 * <pre>
 * @Auther: ousakai
 * @Date: 2019-10-15 15:26
 * @Description:
 * 修改版本: 1.0
 * 修改日期:
 * 修改人 :
 * 修改说明: 初步完成
 * 复审人 :
 * </pre>
 */
public class JdkDynamicImpl implements JdkDynamic {

    public void seeYou(){
        System.out.println("看看你自己我已经对你来不起了");
    }
}
```

jdk动态代理是java jdk自带的一个api类想要使用它也是很简单,新建一个代理类实现接口InvocationHndler实现他的方法invoke

```
/**
 * <pre>
 * @Auther: ousakai
```

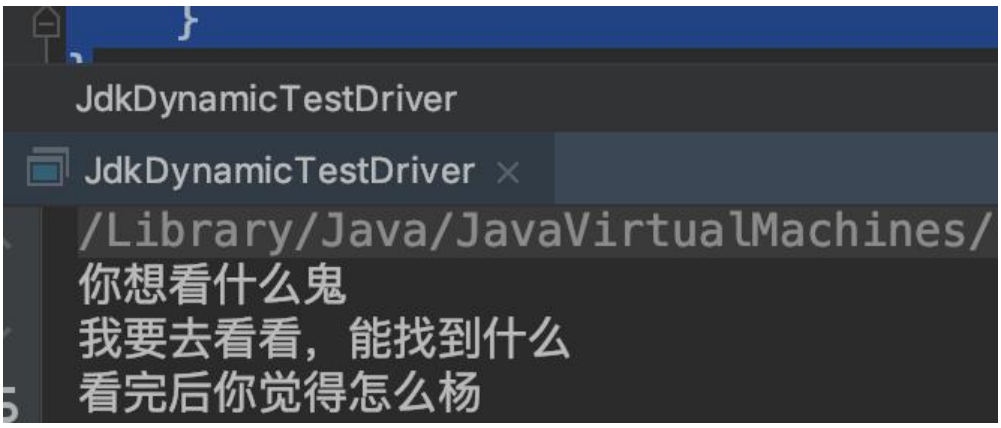
```
 * @Date: 2019-10-15 15:27
 * @Description:
 * 修改版本: 1.0
 * 修改日期:
 * 修改人 :
 * 修改说明: 初步完成
 * 复审人 :
 * </pre>
 */
public class JdkDynamicInvok<T> implements InvocationHandler {
    private T object;

    public JdkDynamicInvok(T object){
        this.object=object;
    }
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println("你想看什么鬼");
        Object objects=method.invoke(object,args);
        System.out.println("看完后你觉得怎么杨");
        return objects;
    }
    public T getInstance(){
        T proxy = (T)Proxy.newProxyInstance(object.getClass().getClassLoader(), object.getClass()
getInterfaces(), this);
        return proxy;
    }
}
```

proxy代理对象要求传入三个参数第一个是目标类的加载类,第二个是目标类实现的接口类,第三个是回的类也就是代理类会执行invoke方法:下面我们来写写测试用例:

```
public class JdkDynamicTestDriver {
    public static void main(String[] args) {
        //jdk动态代理
        JdkDynamic object = new JdkDynamicInvok<JdkDynamic>(new JdkDynamicImpl()).getIn
tance();
        object.lookUp();
    }
}
```

执行结果如下:



是吧是很简单就实现了代理

二.cglib代理就不需要有接口类只有是个类就可以使用代理

我们还是使用jdkDynamic的实现类来做测试

```
 * <pre>
 * @Auther: ousakai
 * @Date: 2019-10-15 15:26
 * @Description:
 * 修改版本: 1.0
 * 修改日期:
 * 修改人 :
 * 修改说明: 初步完成
 * 复审人 :
 * </pre>
 */
public class JdkDynamicImpl implements JdkDynamic {

    public void seeYou(){
        System.out.println("看看你自己我已经对你来不起了");
    }
}
```

这其实是可以不用实现jdkDynamic的我这里为了方便就没有删除

小伙伴们可以只使用类来做目标类

开始我们的表演 新建一个Cglib的类实现MethodInterceptor的interceptor

对了methodInterceptor类是需要cglib-nodep-2.2.jar

```
/**
 * <pre>
 * @Auther: ousakai
 * @Date: 2019-10-15 15:45
 * @Description:
 * 修改版本: 1.0
 * 修改日期:
 * 修改人 :
 * 修改说明: 初步完成
 * 复审人 :
 * </pre>
 */
public class Cglib implements MethodInterceptor {

    public <T>T proxy(T target) {
        Enhancer enhancer = new Enhancer();
        enhancer.setSuperclass(target.getClass());
        enhancer.setCallback(this);
        return (T) enhancer.create();
    }
    @Override
```

```
    public Object intercept(Object o, Method method, Object[] objects, MethodProxy methodP
oxy) throws Throwable {
        System.out.println("看什么看米色吗好看的");
        Object t= methodProxy.invokeSuper(o,objects);
        System.out.println("看来你还是看了");
        return t;
    }
}
```

测试使用

```
    public static void main(String[] args) {
        //jdk动态代理
        JdkDynamic object = new JdkDynamicInvok<JdkDynamic>(new JdkDynamicImpl()).getIn
tance();
        object.lookUp();
        //cglib代理
        JdkDynamicImpl jdkDynamic=new Cglib().proxy(new JdkDynamicImpl());
        jdkDynamic.seeYou();
    }
}
```

最后输出结果为:

```
看什么看米色吗好看的
看看你自己我已经对你来不起了
看来你还是看了

Process finished with exit code 0
```

很简单吧! 这里面有个重点就是Enhancer(增强) 它的方法setSuperclass接收目标类作为夫类,setCallbck接受一个回调类用于执行intercept