链滴

JDK 动态代理一定要有代理对象吗？请结合 Mybatis 回答

作者：xiaodaojava

原文链接：https://ld246.com/article/1592313948009

来源网站：链滴

动态代理

有一段时间没有写文章了, 主要是回想起这两年多的时间,多多少少,每个知识点差不多都有写到了, 一也想不起什么新鲜的知识分享给大家.

今天写动态代理,主要是在看Mybatis源码时,发现真得是把动态代理用的是太6 了, 感叹之余,有一些心得和大家分享一下.

我所理解的动态代理

其实网上对动态代理的解释有很多了,我就不赘述那些概念了, 于小刀看来, 目的只有一个,那就是可以定义逻辑,可以添加逻辑. 在本文中,我想写的是可以自定义逻辑, 在此之前,我们先看一下通常的动态代的代码

动态代理代码

接口

```
/**
 * @author lixiang
 * @date 2020/6/16
 **/
public interface Greet {
    /**
     * 加油的接口定义
     */
    public void cheer();
}
```

实现类

```
/**
 * @author lixiang
 * @date 2020/6/16
 **/
public class GreetImpl implements Greet{

    @Override
    public void cheer() {
        System.out.println("加油, 为了美好的明天!");
    }
}
```

代理类

```
/**
 * @author lixiang
 * @date 2020/6/16
 **/
public class GreetProxy implements InvocationHandler {

    /** 被代理的对象,真正的逻辑,还是要请求这个 */
    private final Greet greet;

    public GreetProxy(Greet greet) {
        this.greet = greet;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println("在真正调用之前");
        greet.cheer();
        System.out.println("在真正调用之后");
        return null;
    }
}
```

Main函数

```
/**
 * @author lixiang
 * @date 2020/6/16
 **/
public class ProxyMain {
    public static void main(String[] args) {
        // 真正的对象
        Greet greet = new GreetImpl();
        // 代理对象
        InvocationHandler handler = new GreetProxy(greet);
        // 使用的jdk代理生成代理类
        Greet proxy = (Greet)Proxy.newProxyInstance(ProxyMain.class.getClassLoader(), new Cla
s[]{Greet.class}, handler);
```

```
        proxy.cheer();
    }
}
```

我们在运行的时候打个断点,可以看到:

如上图所示,我们虽然把jdk生成的代理对象强转成了Greet,但实际上是Proxy类型,运行结果如下图所示:

进入正文

上面这些代码, 是平常的增加逻辑的用法,但,今天小刀想和大家聊的是: 自定义逻辑.先看代码

接口不变,

代理类

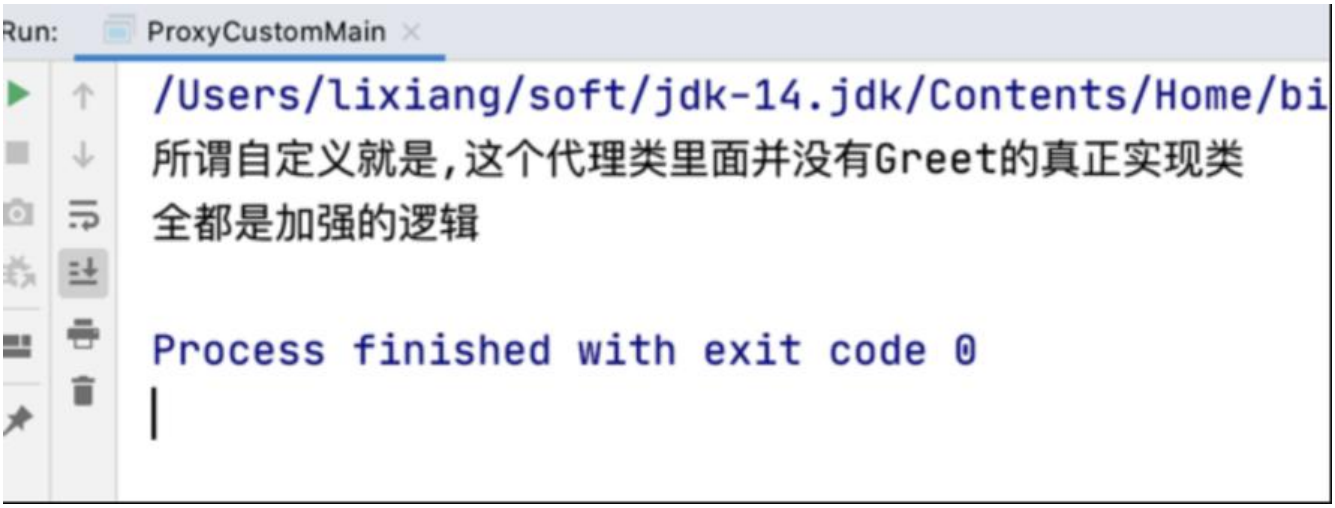
```
/**
 * @author lixiang
 * @date 2020/6/16
 **/
public class GreetCustomProxy implements InvocationHandler {

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println("所谓自定义就是,这个代理类里面并没有Greet的真正实现类");
        System.out.println("全都是加强的逻辑");
        return null;
    }
}
```

main函数

```
/**
 * @author lixiang
 * @date 2020/6/16
 **/
public class ProxyCustomMain {
    public static void main(String[] args) {
        InvocationHandler handler = new GreetCustomProxy();
        Greet proxy = (Greet) Proxy.newProxyInstance(ProxyMain.class.getClassLoader(), new Cla
s[]{Greet.class}, handler);
        proxy.cheer();
    }
}
```

运行结果如下:



全文的重点

是可以正常运行的, 这里会打破大家一个思维定式,就是代理类里面并不一定需要真正的处理对象. 可能部都是自定义的逻辑.

源码中的应用

主要是mybatis , 我们想一下, 在写sql时, 我们经常DAO里面都是接口和定义的方法, 然后mapper的xml里面写SQL, 那么这两者是怎么对应起来的呢? 今天先不细讲, 只是看看动态代理的使用,要出场的是MaperProxy

MapperProxyFactory:

```
protected T newInstance(MapperProxy<T> mapperProxy) {
    return (T) Proxy.newProxyInstance(mapperInterface.getClassLoader(), new Class[] { mapperI
terface }, mapperProxy);
  }
```

我们可以看到,传入的InvocationHandler实际上是mapperProxy

```
 @Override
  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    try {
    // 处理Object相关的方法
      if (Object.class.equals(method.getDeclaringClass())) {
        return method.invoke(this, args);
      } else {
    // 我们的重点关注对象
        return cachedInvoker(method).invoke(proxy, method, args, sqlSession);
      }
    } catch (Throwable t) {
      throw ExceptionUtil.unwrapThrowable(t);
    }
  }
```

cachedInvoker 通过源码,我们可以跟踪到的代码:

```
// 如果是接口中的default方法,则执行
if (m.isDefault()) {
          try {
            if (privateLookupInMethod == null) {
              return new DefaultMethodInvoker(getMethodHandleJava8(method));
            } else {
              return new DefaultMethodInvoker(getMethodHandleJava9(method));
            }
          } catch (IllegalAccessException | InstantiationException | InvocationTargetException
              | NoSuchMethodException e) {
            throw new RuntimeException(e);
          }
        } else {
    // 其他的,就是sql语句之类的
          return new PlainMethodInvoker(new MapperMethod(mapperInterface, method, sqlSess
on.getConfiguration()));
        }
```

最终我们可以看到:

```
@Override
    public Object invoke(Object proxy, Method method, Object[] args, SqlSession sqlSession) t
rows Throwable {
```

```
    return mapperMethod.execute(sqlSession, args);
  }
```

然后使用sqlSession去执行Sql

总结

如上mybatis中对动态代理的使用,并没有实现类,真是在invoke方法中,直接调用了sqlSession去执行SL , 刚开始看到这块时, 不是很好理解 , 要破开思维, 为什么动态代理一定要有代理对象呢? 我们也完全以自己模拟逻辑.

来一起学习吧

欢迎各位大佬关注我的众号: java技术大本营, 也可以加我微信, 进群一起讨论,小弟微信: best39697582