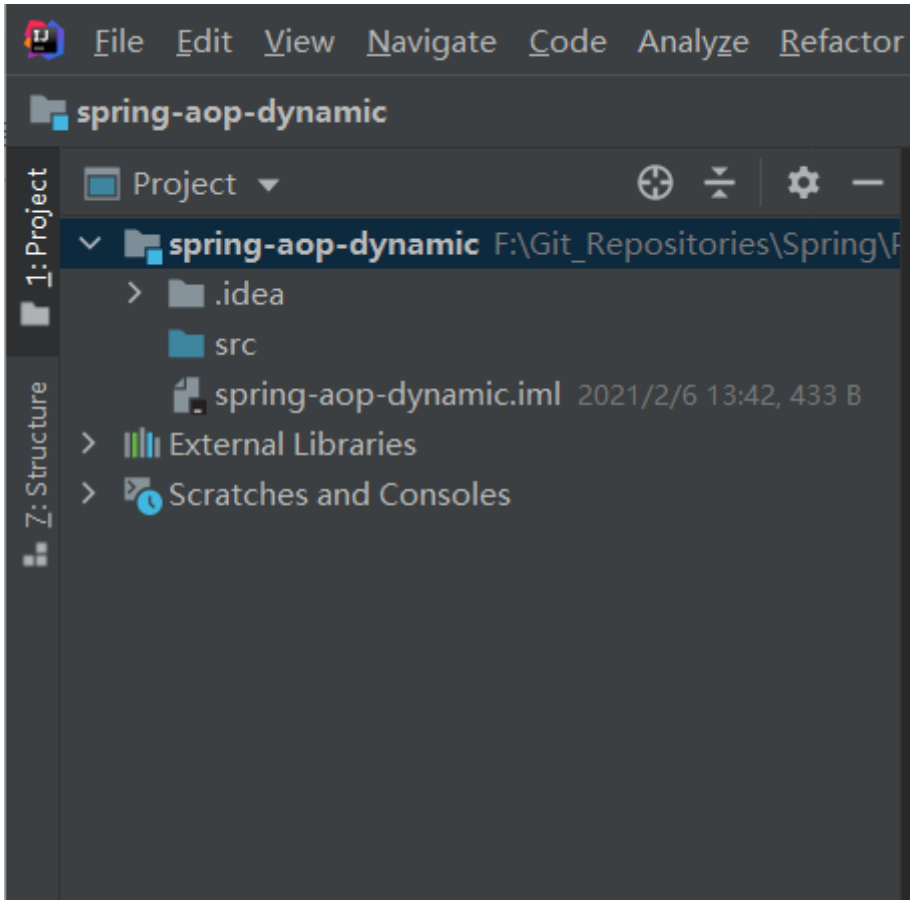
编写一个普通的java工程实现动态代理

1）、创建java项目spring-aop-dynamic



2）、创建一个接口com.studymyself.service.Service和该接口的实现类com.studymyself.service.impl.ServiceImpl

```
package com.studymyself.service;

public interface Service {

    public void doSome();

    public void doOther();

}
```

```
package com.studymyself.service.impl;

import com.studymyself.service.Service;

public class ServiceImpl implements Service {
    @Override
    public void doSome() {
        System.out.println("开始执行doSome方法中的事务代码\n事务执行完毕！");
    }
```

```
    @Override
    public void doOther() {
        System.out.println("开始执行doOther方法中的事务代码\n事务执行完毕！");
    }
}
```

3）、当项目中需要添加一些和业务无关的功能：需要在doSome方法前添加执行时间日志，执行完该方法后提交事务；而doOther方法执行却不需要添加这些功能，就使用动态代理来实现功能的增强。再创建一个工具类com.studymyself.util.ServiceTool，其中有两个方法，一个是时间日志的方法一个是提交事务的方法

```
package com.studymyself.util;

import java.util.Date;

public class ServiceTool {

    /**
     * 时间日志功能方法
     */
    public static void doLog(){
        System.out.println("doSome开始执行时间"+new Date());
    }

    /**
     * 提交事务功能方法
     */
    public static void doTrans(){
        System.out.println("正在提交事务...\n事务提交成功");
    }
}
```

4）、首先分析目标类ServiceImpl类有接口，可以使用JDK动态代理，先创建一个InvocationHandler接口的实现类，com.studymyself.handler.Handler来给目标方法增加功能

```
package com.studymyself.handler;

import com.studymyself.util.ServiceTool;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

public class Handler implements InvocationHandler {

    //定义一个动态的目标类存储变量
    Object target = null;
```

```
    //在创建这个类的对象时将目标类对象给target赋值
    public Handler(Object target) {
        this.target = target;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {

        Object res = null;
        //根据新功能需求，判断调用方法是doSome还是doOther
        if ("doSome".equals(method.getName())){

            //执行doSome时添加新功能
            //doSome执行的时间日志（非事务相关的功能，就是有没有下面方法中事务都能执行）
            ServiceTool.doLog();

            //执行doSome
            res = method.invoke(target,args);

            //doSome执行完后提交事务（非事务相关的功能，就是有没有上面方法中事务都能执行）
           ServiceTool.doTrans();

        }else if ("doOther".equals(method.getName())){

            //不添加新功能，直接执行doOther方法
           res = method.invoke(target, args);

        }
        return res;
    }
}
```

5）、使用JDK中的Proxy类，创建代理对象。新建一个启动类com.studymyself.MyApp

```
package com.studymyself;

import com.studymyself.handler.Handler;
import com.studymyself.service.Service;
import com.studymyself.service.impl.ServiceImpl;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Proxy;

public class MyApp {

    public static void main(String[] args) {

        //使用JDK的Proxy类中的方法动态创建代理对象

        //创建目标类对象,使用多态
        Service target = new ServiceImpl();

        //创建InvocationHandler接口实现类对象
```

```
        InvocationHandler handler = new Handler(target);

        //使用JDK中Proxy类中的静态方法创建代理对象，为了能够使用目标类接口的方法
        //方法的返回对象要强制转换成目标类的接口类型
        Service proxy = (Service) Proxy.newProxyInstance(
                ServiceImpl.class.getClassLoader(),//目标类的类加载器
                ServiceImpl.class.getInterfaces(),//目标类的接口
                handler//InvocationHandler接口实现类对象
        );

        //用这个代理对象执行接口中的方法
        proxy.doOther();
        System.out.println("====================================");
        proxy.doSome();
    }

}
```

简单理解上面的执行原理：

```
    Service proxy = (Service) Proxy.newProxyInstance(
                ServiceImpl.class.getClassLoader(),//目标类的类加载器
                ServiceImpl.class.getInterfaces(),//目标类的接口
                handler//InvocationHandler接口实现类对象
        );
    //用这个代理对象执行接口中的方法
        proxy.doOther();
        System.out.println("====================================");
        proxy.doSome();
```

1、使用Proxy.newProxyInstance方法时需要目标类的接口是因为创建的代理对象也是一个该接口的实现类。

2、需要的InvocationHandler接口实现类对象是因为执行代理对象的doSome或doOther方法内部就是执行handler.invoke(...)

3、该代理对象中会有一个字符串变量存储对象中doSome或doOther方法的方法名，在doSome方法中该变量值是"doSome"

在doOther中该变量值是"doOther",而创建代理对象时需要的目标类的类加载器就是和这个字符串变量来获取目标类的

doSome或doOther方法的Method对象，然后作为参数传给handler对象的invoke方法的Method参数值。这样我们使用proxy代理对象执行doSome时，内部执行代理对象的doSome，然后doSome内部执行invoke方法，方法中传入的Method是目标类中doSome的Method对象，invoke内部通过判断执行为true的语句，然后执行method.invoke()，该方法实际执行的就是目标类中的doSome方法，而不是doOther方法。

这样就能理解为什么JDK动态代理为什么必须要目标类有接口了：

动态代理的代理对象类实际是目标类的接口的另一个实现类。