Day 01 To read list:

Dynamic proxy

<https://www.jianshu.com/p/471c80a7e831>

/\*\*

\* 动态代理,动态代理类不要显示的实现被代理类所实现的接口

\* @author yujie.wang

\*

\*/

public class MyInvocationHandler implements InvocationHandler{

private Object object;

public MyInvocationHandler(Object object){

this.object = object;

}

@Override

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable {

// TODO Auto-generated method stub

System.out.println("MyInvocationHandler invoke begin");

System.out.println("proxy: "+ proxy.getClass().getName());

System.out.println("method: "+ method.getName());

for(Object o : args){

System.out.println("arg: "+ o);

}

//通过反射调用 被代理类的方法

method.invoke(object, args);

System.out.println("MyInvocationHandler invoke end");

return null;

}

public static void main(String [] args){

//创建需要被代理的类

Student s = new Student();

//这一句是生成代理类的class文件，前提是你需要在工程根目录下创建com/sun/proxy目录，不然会报找不到路径的io异常

System.getProperties().put("sun.misc.ProxyGenerator.saveGeneratedFiles","true");

//获得加载被代理类的 类加载器

ClassLoader loader = Thread.currentThread().getContextClassLoader();

//指明被代理类实现的接口

Class<?>[] interfaces = s.getClass().getInterfaces();

// 创建被代理类的委托类,之后想要调用被代理类的方法时，都会委托给这个类的invoke(Object proxy, Method method, Object[] args)方法

MyInvocationHandler h = new MyInvocationHandler(s);

//生成代理类

Person proxy = (Person)Proxy.newProxyInstance(loader, interfaces, h);

//通过代理类调用 被代理类的方法

proxy.sayHello("yujie.wang", 20);

proxy.sayGoodBye(true, 100);

System.out.println("end");

}

}

仔细分析上面的动态代理实现代码，我们看到这里涉及到java反射包下的一个接口InvocationHandler和一个类Proxy。

package java.lang.reflect;

public interface InvocationHandler {

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable;

这个接口只有一个invoke方法，我们在通过代理类调用被代理类的方法时，最终都会委托给这个invoke方法执行，

//通过代理类调用 被代理类的方法

proxy.sayHello("yujie.wang", 20);

proxy.sayGoodBye(true, 100);

Java Decompiler