动态代理

创建InvocationHandler对象

//创建一个与代理对象相关联的InvocationHandler

InvocationHandler stuHandler = new MyInvocationHandler<Person>(stu);

//创建一个代理对象stuProxy，代理对象的每个执行方法都会替换执行Invocation中的invoke方法

Person stuProxy= (Person) Proxy.newProxyInstance(Person.class.getClassLoader(), new Class<?>[]{Person.class}, stuHandler);

动态代理例子

班长帮学生交学费给老师

1.创建公共类接口Person

public interface Person {

//上交班费

void giveMoney();

}

2.创建需要被代理的类

public class Student implements Person {

private String name;

public Student(String name) {

this.name = name;

}

@Override

public void giveMoney() {

try {

//假设数钱花了一秒时间

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println(name + "上交班费50元");

}

}

3. 再定义一个检测方法执行时间的工具类，在任何方法执行前先调用start方法，执行后调用finsh方法，就可以计算出该方法的运行时间，这也是一个最简单的方法执行时间检测工具。

[复制代码](javascript:void(0);)

public class MonitorUtil {

private static ThreadLocal<Long> tl = new ThreadLocal<>();

public static void start() {

tl.set(System.currentTimeMillis());

}

//结束时打印耗时

public static void finish(String methodName) {

long finishTime = System.currentTimeMillis();

System.out.println(methodName + "方法耗时" + (finishTime - tl.get()) + "ms");

}

}

4, 创建StuInvocationHandler类，实现InvocationHandler接口，这个类中持有一个被代理对象的实例target。InvocationHandler中有一个invoke方法，所有执行代理对象的方法都会被替换成执行invoke方法。



再在invoke方法中执行被代理对象target的相应方法。当然，在代理过程中，我们在真正执行被代理对象的方法前加入自己其他处理。这也是Spring中的AOP实现的主要原理，这里还涉及到一个很重要的关于java反射方面的基础知识。



public class StuInvocationHandler<T> implements InvocationHandler {



//invocationHandler持有的被代理对象

T target;



public StuInvocationHandler(T target) {

this.target = target;

}

/\*\*

\* proxy:代表动态代理对象

\* method：代表正在执行的方法

\* args：代表调用目标方法时传入的实参

\*/

@Override

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



System.out.println("代理执行" +method.getName() + "方法");

\*/

//代理过程中插入监测方法,计算该方法耗时

MonitorUtil.start();

Object result = method.invoke(target, args);

MonitorUtil.finish(method.getName());

return result;

}

}

做完上面的工作后，我们就可以具体来创建动态代理对象了，上面简单介绍了如何创建动态代理对象，我们使用简化的方式创建动态代理对象：

public class ProxyTest {

public static void main(String[] args) {

//创建一个实例对象，这个对象是被代理的对象

Person zhangsan = new Student("张三");

//创建一个与代理对象相关联的InvocationHandler

InvocationHandler stuHandler = new StuInvocationHandler<Person>(zhangsan);

//创建一个代理对象stuProxy来代理zhangsan，代理对象的每个执行方法都会替换执行Invocation中的invoke方法

Person stuProxy = (Person) Proxy.newProxyInstance(Person.class.getClassLoader(), new Class<?>[]{Person.class}, stuHandler)；

//代理执行上交班费的方法

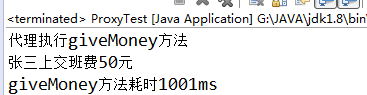
stuProxy.giveMoney();

}

}

我们执行这个ProxyTest类，先想一下，我们创建了一个需要被代理的学生张三，将zhangsan对象传给了stuHandler中，我们在创建代理对象stuProxy时，将stuHandler作为参数了的，上面也有说到所有执行代理对象的方法都会被替换成执行invoke方法，也就是说，最后执行的是StuInvocationHandler中的invoke方法。所以在看到下面的运行结果也就理所当然了。

运行结果：



上面说到，动态代理的优势在于可以很方便的对代理类的函数进行统一的处理，而不用修改每个代理类中的方法。是因为所有被代理执行的方法，都是通过在InvocationHandler中的invoke方法调用的，所以我们只要在invoke方法中统一处理，就可以对所有被代理的方法进行相同的操作了。例如，这里的方法计时，所有的被代理对象执行的方法都会被计时，然而我只做了很少的代码量。

动态代理的过程，代理对象和被代理对象的关系不像静态代理那样一目了然，清晰明了。因为动态代理的过程中，我们并没有实际看到代理类，也没有很清晰地的看到代理类的具体样子，而且动态代理中被代理对象和代理对象是通过InvocationHandler来完成的代理过程的，其中具体是怎样操作的，为什么代理对象执行的方法都会通过InvocationHandler中的invoke方法来执行。带着这些问题，我们就需要对java动态代理的源码进行简要的分析，弄清楚其中缘由。

**四、动态代理原理分析**