案例

1. 先写一个实现加减乘除的接口

**interface** ArithmeticCalculator {

**int** add(**int** i,**int** j);

**int** sub(**int** i,**int** j);

**int** mul(**int** i,**int** j);

**int** div(**int** i,**int** j);

}

1. 写一个这个接口的实现类

**public** **class** ArithmeticCalculatorImpl **implements** ArithmeticCalculator {

**public** **int** add(**int** i, **int** j) {

**return** i+j;

}

**public** **int** sub(**int** i, **int** j) {

**return** i-j;

}

**public** **int** mul(**int** i, **int** j) {

**return** i\*j;

}

**public** **int** div(**int** i, **int** j) {

**return** i/j;

}

}

1. 有新需求，需要在运算的时候做上日志，使用动态代理

动态代理

步骤：

1. **创建一个私有对象为需要代理接口的对象，并用构造器传入初始化**
2. **写一个获取代理对象的方法**
3. **在获取代理类方法中，使用Proxy类的静态方法newProxyInstance(loader, interfaces, h)，获取代理类对象**

newProxyInstance(loader, interfaces, h)有三个参数，必须先获取出来传入

① **代理类的类加载器**，这种创建对象的方式较为特殊，所以需要知道代理类的类加载器

② **代理对象需要代理的方法属于的接口**，必须是接口的Class对象数组

③ **InvocationHandler h = new InvocationHandler() {}** 这个接口的实现对象，可以直接实现

该接口有一个方法，是代理对象在调用原对象方法时实际调用的方法

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

**第一个参数为原对象**，一般不用。

**第二个参数为代理执行的方法**，可以通过对这个方法的执行前后的操作，改变行为。

**第三个参数为执行方法的参数**。

**返回值为执行方法实际的返回值**

**public** **class** ArithmeticCalculatorProxy {

// 需要代理类的对象

ArithmeticCalculator arithmeticCalculator;

// 获取这个对象

**public** ArithmeticCalculatorProxy(ArithmeticCalculator arithmeticCalculator) {

**this**.arithmeticCalculator = arithmeticCalculator;

}

// 获取代理对象方法

**public** ArithmeticCalculator getLogginProxy() {

**// 获取类加载器，在实行动态代理时，要知道所代理的类是由哪个类加载器加载的**

ClassLoader loader = arithmeticCalculator.getClass().getClassLoader();

**// 获取实现类的类型，（需要代理的方法所属的接口，必须是一个接口的class）**

Class[] interfaces = **new** Class[] { ArithmeticCalculator.**class** };

InvocationHandler h = **new** InvocationHandler() {

**// 代理对象在执行代理方法时，实际执行的方法**

@Override

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

String methodName = method.getName();

System.***out***.println("执行" + methodName + "方法，参数为" + args[0] + "," + args[1]);

//执行代理对象的执行方法

Object resault = method.invoke(arithmeticCalculator, args);

System.***out***.println("执行结果为：" + resault);

//返回实际返回结果

**return** resault;

}

};

// 获取这个代理类最终生成的对象并返回

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

**return** proxy;

}

}

1. 使用动态代理

**public** **static** **void** main(String[] args) {

//创建一个对象

ArithmeticCalculator arithmeticCalculator=**new** ArithmeticCalculatorImpl();

//获取代理工厂

ArithmeticCalculatorProxy proxy=**new** ArithmeticCalculatorProxy(arithmeticCalculator);

//获取代理对象

arithmeticCalculator=proxy.getLogginProxy();

//代理对象执行代理方法

arithmeticCalculator.add(1, 2);

arithmeticCalculator.sub(5, 3);

}