代理类

yuanfengqiao 于 2019-08-16 16:26:21 发布

前几天有人问我了解Spring AOP的实现原理吗？我说Spring AOP的底层实现是基于动态代理的，是采用cglib来实现的，我视乎只是大致的知道是基于代理类的，但是具体是怎么实现的我貌似也是一知半解，今天有空上网看了下，将自己的思路写出来，谈谈自己的想法。

一、代理类主要分为

1、静态代理

2、动态代理

二、代理的主要作用大致可以总结如下：

1、保证了目标类的安全性：使用代理模式，避免了你直接操作目标对象（即：被代理的对象），你直接操作的是代理对象。

2、功能增强：我们可以将核心的业务写在目标类中，而在代理类中对目标类执行前后进行增强功能的编写，这样可以在不改变目标对象的前提下达到了功能增强的目的。

3、解耦：核心业务逻辑写在目标类中，辅助业务逻辑写在代理类中，从而达到了解耦的目的

4、简化了开发：和代理类打交道，不和目标类打交道，简化了开发

三、代理类的适用场景：

3.1、静态代理的典型应用就是Thread和Runnable

3.2、动态代理的典型应该就是Spring AOP

四、静态代理

总结下编写静态代理的步骤：第一步：将业务抽象为接口；第二步：创建代理类和目标类，让代理类和目标类都实现这个接口；第三：代理角色持有真正角色的引用，真正执行具体业务时交给目标类执行。

接口：

```java
@FunctionalInterface
public interface Runnable {
    /**
     * When an object implementing interface <code>Runnable</code> is used
     * to create a thread, starting the thread causes the object's
     * <code>run</code> method to be called in that separately executing
     * thread.
     * <p>
     * The general contract of the method <code>run</code> is that it may
     * take any action whatsoever.
     *
     * @see     java.lang.Thread#run()
     */
    public abstract void run();
}
```


目标类：目标类实现了Runnable接口，并且重写了 run方法

```java
public class MyThread implements Runnable {
    @Override
    public void run() {
        System.out.println("目标类：this is MyThread");
    }


}
```

代理类：

```java
public
class Thread implements Runnable {
    /* Make sure registerNatives is the first thing <clinit> does. */
```

Thread 实现了Runnable接口，并且重写run方法

```java
     */
    @Override
    public void run() {
        if (target != null) {
            target.run();
        }
    }

```

调用Thread的构造函数，传入目标类

```
public Thread(Runnable target) {
    init( g: null, target,  name: "Thread-" + nextThreadNum(),  stackSize: 0);
}
```

传入目标类：MyThread

测试类

```
public class Client {

    public static void main(String[] args) {
        MyThread myThread=new MyThread();
        Thread thread=new Thread(myThread);
        thread.start();
    }

}
```

```
Client › main()
Run:  Client ×
D:\software\JDK\bin\java.exe ...
目标类：this is MyThread

Process finished with exit code 0
```


五、动态代理

动态代理的实现有两种，一种是基于JDK的动态代理，另一种是基于 cglib 的动态代理。

JDK动态代理主要是利用反射机制生成一个实现代理接口的代理对象，代理对象的生成主要是利用java的api，在内存中动态的生成代理对象。代理对象不需要实现接口，目标类必须实现接口，否则无法使用基于JDK的动态代理。

具体代码如下：

```java
/**
 * @Author: yuanfengqiao
 * @Date: 2019/8/16 23:26
 * @Version 1.0
 * @Description: 定义一个接口
 */
public interface UserService {
    String userService(String name);
}
```


定义一个实现接口的实现类

```java
/**
 * @Author: yuanfengqiao
 * @Date: 2019/8/16 23:26
 * @Version 1.0
 * @Description: 定义一个实现类，该类为目标类
 */
public class UserServiceImpl implements UserService {
    @Override
    public String userService(String name) {
        System.out.println("姓名为: "+name);
        return "姓名为"+name;
    }
}
```


定义工厂类用于实现代理类

```java
/**
 * @Author: yuanfengqiao
 * @Date: 2019/8/16 23:27
 * @Version 1.0
 * @Description: No Description
 */
public class ProxyFactory implements InvocationHandler {
    //定义目标类
    private Object target;
    //初始化目标对象
    public ProxyFactory(Object target) {
        this.target = target;
    }
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println("代理开始");
        Object invoke = method.invoke(target, args);
        System.out.println("代理结束");
        return invoke;
    }
    //利用反射技术生成代理对象
    public Object getTarget(){
        return Proxy.newProxyInstance(target.getClass().getClassLoader(),
                target.getClass().getInterfaces(), h: this);
    }
}
```


测试类：

```java
/**
 * @Author: yuanfengqiao
 * @Date: 2019/8/16 23:33
 * @Version 1.0
 * @Description: No Description
 */
public class ClientTest {
    public static void main(String[] args) {
        ProxyFactory proxyFactory=new ProxyFactory(new UserServiceImpl());
        UserService target =(UserService) proxyFactory.getTarget();
        System.out.println(target.getClass());
        target.userService( name: "tom");
    }
}
```

```
ClientTest
Run:  ClientTest ×
D:\software\java\jdk\bin\java.exe ...
class com.sun.proxy.$Proxy0
代理开始
姓名为：tom
代理结束

Process finished with exit code 0
```


六、cglib动态代理

静态代理和jdk动态代理都需要目标类实现接口，而cglib代理不需要目标类实现任何接口，目标对象只是一个单独的对象时也能使用cglib代理，cglib底层是使用ASM框架（使用cglib时需要引入asm和cglib这两个jar包），将目标对象的字节码文件加载进来，然后通过修改字节码文件创建一个子类，然后通过这个子类来实现动态代理。注意目标类不能被final修饰，因为若目标类被final修饰，cglib无法再内存中动态生成的它的子类。

代码如下：

目标类

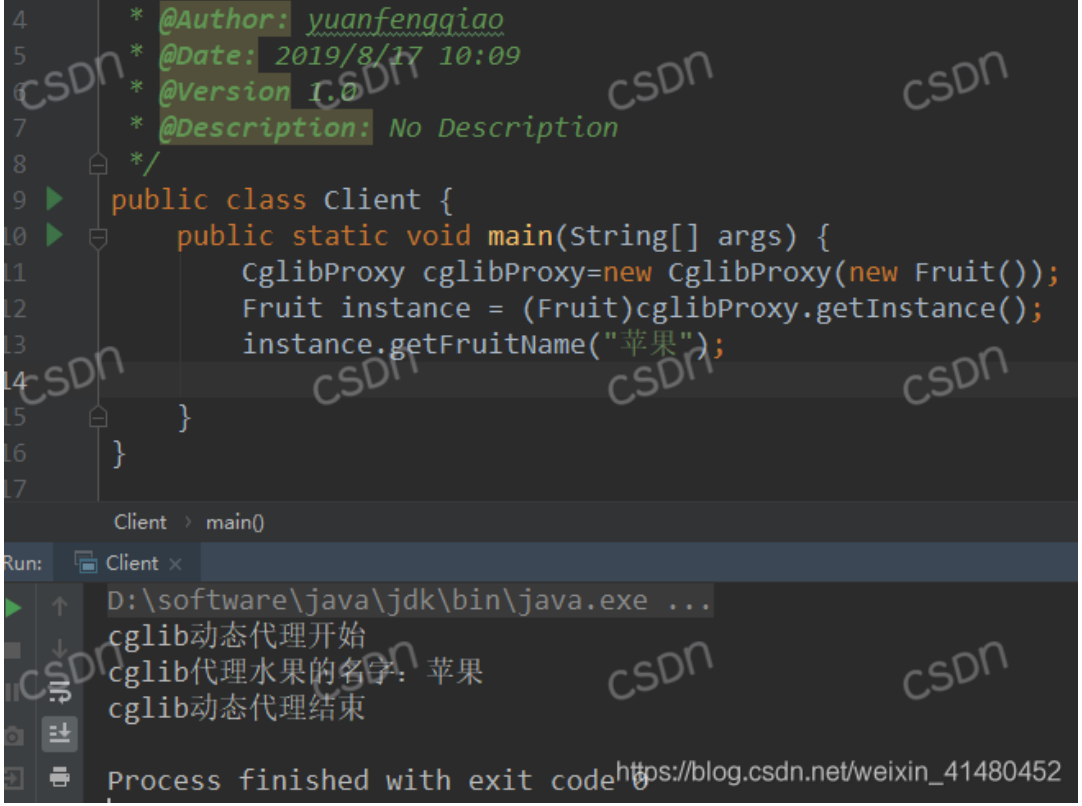
```java
/**
 * @Author: yuanfengqiao
 * @Date: 2019/8/17 9:58
 * @Version 1.0
 * @Description: No Description
 */
public class Fruit {
    public String getFruitName(String name){
        System.out.println("cglib代理水果的名字：+name);
        return name;
    }
}
```


代理工厂类

```java
/**
 * @Author: yuanfengqiao
 * @Date: 2019/8/17 10:00
 * @Version 1.0
 * @Description: No Description
 */
public class CglibProxy implements MethodInterceptor {
    //被代理对象
    private  Object target;
    //对代理对象进行初始化
    public CglibProxy(Object target){
        this.target=target;
    }
    //获得代理对象
    public Object getInstance(){
        Enhancer enhancer=new Enhancer();
        //设置父类，因为cglib的原理就是利用加载字节码文件来动态生成一个子类对象，来实现动态代理,父类就是target对象
        enhancer.setSuperclass(target.getClass());
        //设置回调函数
        enhancer.setCallback(this);
        //创建并返回代理对象
        Object o = enhancer.create();
        return o;
    }
    //重写拦截方法
    @Override
    public Object intercept(Object o, Method method, Object[] objects, MethodProxy methodProxy) throws Throwable

        System.out.println("cglib动态代理开始");
        Object invoke = method.invoke(target, objects);
        System.out.println("cglib动态代理结束");
        return invoke;
    }
}
```


测试类：



总结：JDK动态代理和cglib动态代理的区别?

（1）、JDK动态代理只能针对实现接口的类实现动态代理

（2）、cglib可以针对无需实现接口的类实现代理，主要是针对代理类生成一个子类，然后覆盖其中的方法，因为要生成子类，所以目标类不能用fianl修饰。