JDK动态代理

C-2 创建: 张林伟, 最后修改: 张林伟 2018-09-25 18:21

> 在程序运行时，运用反射机制动态创建而成。动态代理类的字节码在程序运行时由 Java 反射机制动态生成，无需程序员手工编写它的源代码。动态代理类不仅简化了编程工作，而且提高了软件系统的可扩展性，因为 Java 反射机制可以生成任意类型的动态代理类。java.lang.reflect 包中的 Proxy 类和 InvocationHandler 接口提供了生成动态代理类的能力。

Demo

IHelloWorld 接口

代码块 Java

```java
public interface IHelloWorld {
    void sayHello();
}
```

IHelloWorld 实现类 HelloWorldImpl

代码块 Java

```java
public class HelloWorldImpl implements IHelloWorld {
    @Override public void sayHello() {
        System.out.println("hello world");
    }
}
```

InvocationHandler 接口实现类 HelloWorldProxy

代码块 Java

```java
public class HelloWorldProxy implements InvocationHandler {

    private Object target;

    public HelloWorldProxy(Object target) {
        this.target = target;
    }

    @Override public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println("method: " + method.getName() + " is invoked.");
        return method.invoke(target, args);
    }
}
```

测试用例:

代码块 Java

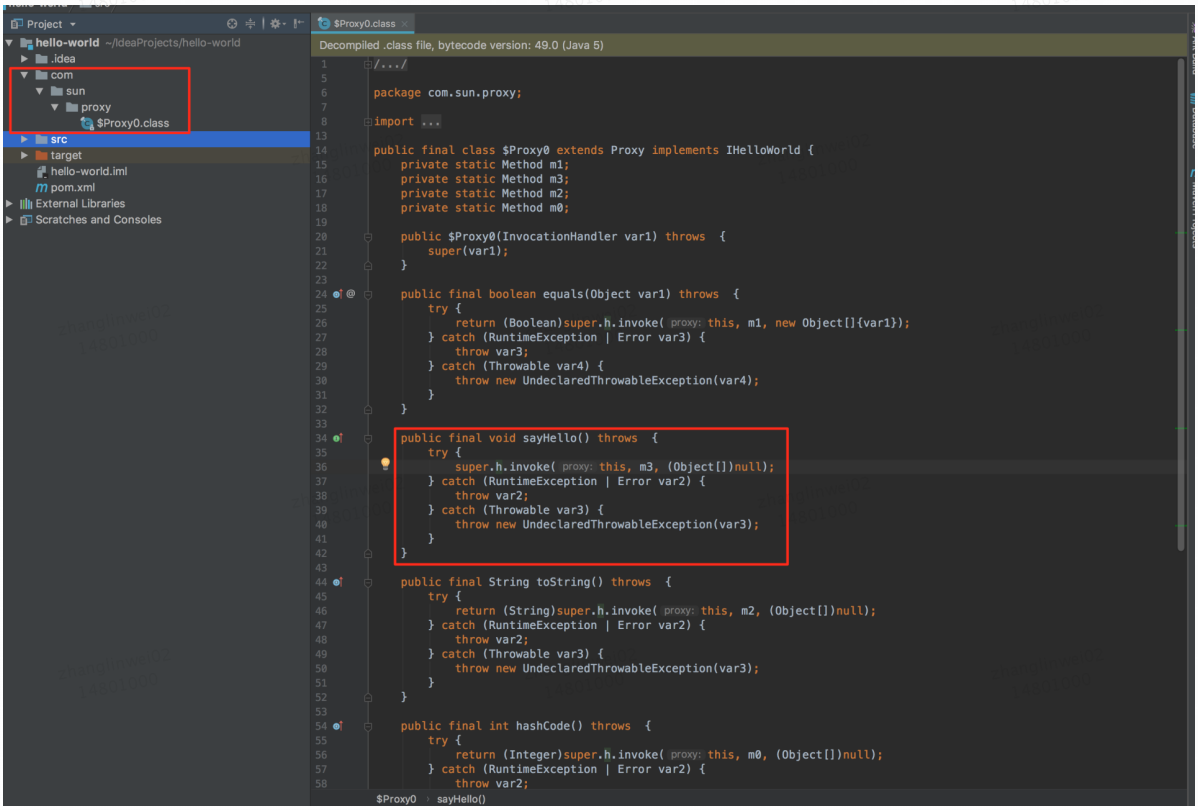
```java
public class Test {

    public static void main(String[] args) throws NoSuchMethodException, IllegalAccessException,
            InvocationTargetException, InstantiationException {
        // 第一种写法
        Class<?> proxyClass = Proxy.getProxyClass(Test.class.getClassLoader(), IHelloWorld.class);
        final Constructor<?> cons = proxyClass.getConstructor(InvocationHandler.class);
        final InvocationHandler ih = new HelloWorldProxy(new HelloWorldImpl());
        IHelloWorld helloWorld = (IHelloWorld) cons.newInstance(ih);
        helloWorld.sayHello();

        // 第二种写法，内部实现跟第一种写法一样
        IHelloWorld helloWorld1 = (IHelloWorld) Proxy.newProxyInstance(Test.class.getClassLoader(),
                new Class<?>[]{IHelloWorld.class},
                new HelloWorldProxy(new HelloWorldImpl()));
        helloWorld1.sayHello();

    }
}
```

打印结果：

原理剖析

把-Dsun.misc.ProxyGenerator.saveGeneratedFiles=true 参数加入到JVM 启动参数中，作用是帮我们把JDK动态生成的proxy class 的字节码保存到硬盘中，帮助我们查看具体生成proxy的内容

运行后生成代理类字节码



点击展开内容

由代码可以看出：

equals、toString、hashCode 三个方法也是走 InvocationHandler 实现类，因此也可以代理。

其他Object上的方法将不会走代理处理逻辑，直接走Proxy继承的Object上方法逻辑。

代理类生成过程主要包括两部分：

- **代理类字节码生成**
- **把字节码通过传入的类加载器加载到虚拟机中**

优缺点

只能对实现了接口的类进行，没有实现接口的类不能使用JDK动态代理。

个人理解，针对接口代理，要比针对实现类代理节省更多代码，当接口新增实现类时，不用添加额外代码。

参考资料：

http://www.importnew.com/23168.html

https://www.jianshu.com/p/1712ef4f2717