代理模式

代理模式：指为其他对象提供一种代理以控制对这个对象的访问

场景：AOP实现增强

代理对象持有被代理的对象，在执行方法前后进行增强

静态代理

显示生命被代理对象。

只能代理一类对象，固定了。没办法进行扩展。

例子：修复现场bug，需要运维人员把bug日志和问题进行反馈，工程师根据反馈进行修复并返回补丁包给运维人员，运维人员进行更新。

工程师接口

```java
package com.dml.pattern.proxy.statics;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:25
 * @describe 工程师
 */
public interface Engineer {
    void fixBug();
}
```

Java开发工程师

```java
package com.dml.pattern.proxy.statics;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:27
 * @describe java工程师
 */
public class JavaEngineer implements Engineer {
    @Override
    public void fixBug() {
        System.out.println("java开发工程师修改bug,发送补丁包");
    }
}
```

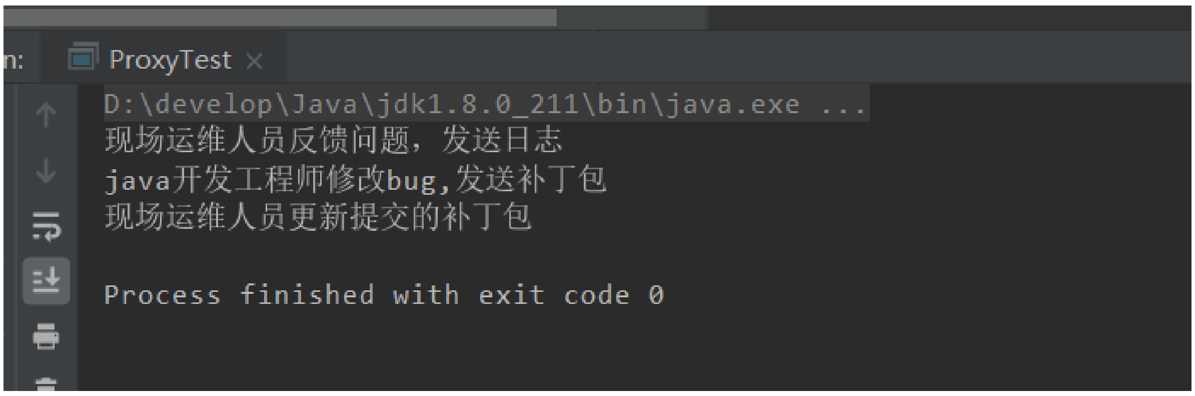
运维人员

```java
package com.dml.pattern.proxy.statics;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:29
 * @describe 现场运维人员
 */
public class OperationsStaffProxy {

    private Engineer engineer;

    public OperationsStaffProxy(Engineer engineer) {
        this.engineer = engineer;
    }

    public void fixBug() {
        System.out.println("现场运维人员反馈问题，发送日志");
        engineer.fixBug();
        System.out.println("现场运维人员更新提交的补丁包");
    }
}
```

测试

```java
package com.dml.pattern.proxy.statics;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:31
 * @describe
 */
public class ProxyTest {

    public static void main(String[] args) {
        OperationsStaffProxy operationsStaffProxy = new OperationsStaffProxy(new
JavaEngineer());
        operationsStaffProxy.fixBug();
    }
}
```

运行结果：



```
D:\develop\Java\jdk1.8.0_211\bin\java.exe ...
现场运维人员反馈问题，发送日志
java开发工程师修改bug,发送补丁包
现场运维人员更新提交的补丁包

Process finished with exit code 0
```

动态代理

方式一 JDK

需要注意，JDK的代理方式，需要被代理对象必须实现接口，否则会报错。

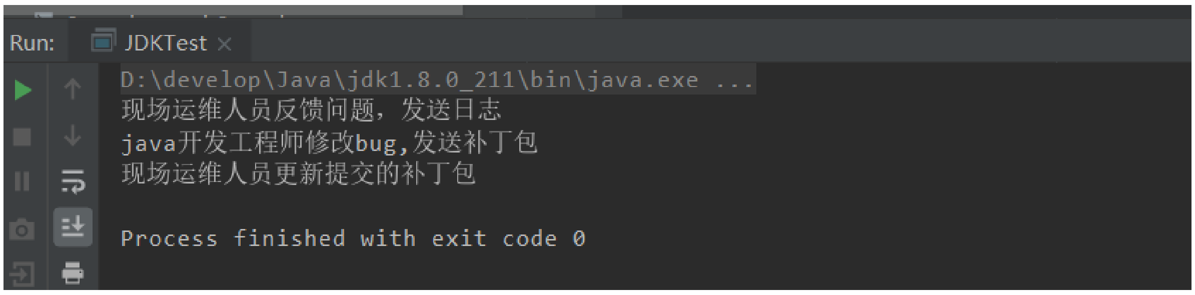
代理对象

```
package com.dml.pattern.proxy.jdk;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:42
 * @describe
 */
public class JDKProxy implements InvocationHandler {

    private Object target;

    public Object getInstance(Class clazz) {
        Object obj = null;
        try {
            obj = clazz.newInstance();
            this.target = obj;
            return Proxy.newProxyInstance(clazz.getClassLoader(),
clazz.getInterfaces(), this);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
        before();
        Object invoke = method.invoke(this.target, args);
        after();
        return invoke;
    }

    private void before(){
        System.out.println("现场运维人员反馈问题，发送日志");
    }

    private void after(){
        System.out.println("现场运维人员更新提交的补丁包");
    }
}
```

测试

```java
package com.dml.pattern.proxy.jdk;

import com.dml.pattern.proxy.Engineer;
import com.dml.pattern.proxy.JavaEngineer;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:52
 * @describe
 */
public class JDKTest {
    public static void main(String[] args) {
        JDKProxy jdkProxy = new JDKProxy();
        Engineer engineer = (Engineer) jdkProxy.getInstance(JavaEngineer.class);
        engineer.fixBug();
    }
}
```

结果：



```
D:\develop\Java\jdk1.8.0_211\bin\java.exe ...
现场运维人员反馈问题，发送日志
java开发工程师修改bug,发送补丁包
现场运维人员更新提交的补丁包

Process finished with exit code 0
```

原理分析

字节码重组

1. 那大被代理对象的引用，并且获取到它的所有的接口，反射获取
2. JDK proxy类重新生成一个新的类，同时新的类要实现被代理类所有实现的所有接口
3. 把新加的业务逻辑方法由一定的逻辑代码去调用(在代码中体现)
4. 编译新生成的java代码
5. 重新加载新生产的类到JVM中运行

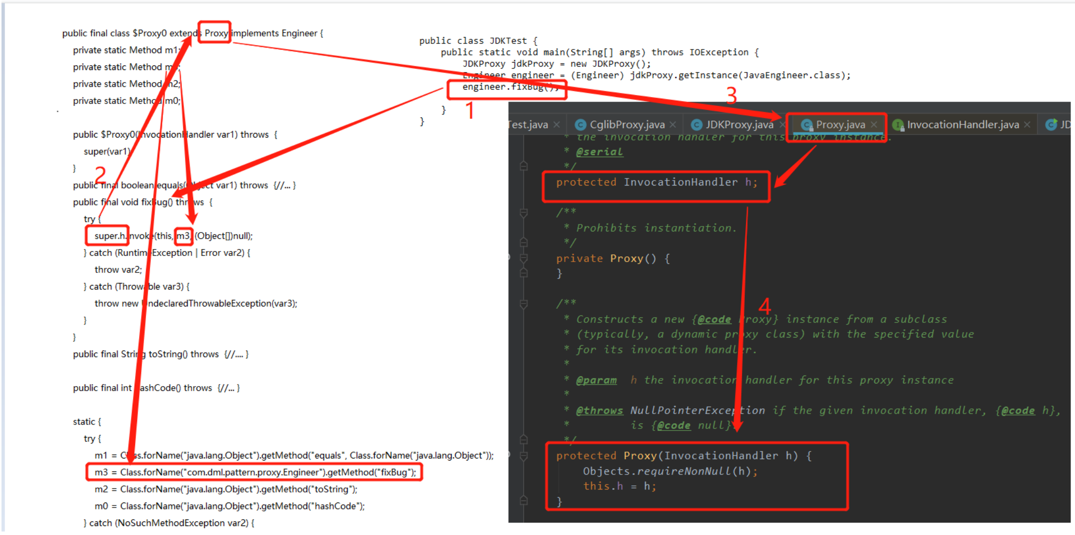
- 通过反编译拿到新生成的类

```java
package com.dml.pattern.proxy.jdk;

import com.dml.pattern.proxy.Engineer;
import com.dml.pattern.proxy.JavaEngineer;
import sun.misc.ProxyGenerator;

import java.io.FileOutputStream;
import java.io.IOException;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:52
 * @describe
 */
public class JDKTest {
    public static void main(String[] args) throws IOException {
        JDKProxy jdkProxy = new JDKProxy();
        Engineer engineer = (Engineer) jdkProxy.getInstance(JavaEngineer.class);
        engineer.fixBug();
        // 通过反编译工具可以查看源代码
        byte[] bytes = ProxyGenerator.generateProxyClass("$Proxy0", new Class[]
{Engineer.class});
        FileOutputStream os = new FileOutputStream("D:\\develop\\study\\pattern-
proxy\\src\\$Proxy0.class");
        os.write(bytes);
        os.close();
    }
}
```

- $Proxy0.class

```java
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)
//

import com.dml.pattern.proxy.Engineer;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;
import java.lang.reflect.UndeclaredThrowableException;

public final class $Proxy0 extends Proxy implements Engineer {
    private static Method m1;
    private static Method m3;
    private static Method m2;
    private static Method m0;

    public $Proxy0(InvocationHandler var1) throws  {
        super(var1);
    }

    public final boolean equals(Object var1) throws  {
        try {
            return (Boolean)super.h.invoke(this, m1, new Object[]{var1});
        } catch (RuntimeException | Error var3) {
            throw var3;
        } catch (Throwable var4) {
            throw new UndeclaredThrowableException(var4);
        }
    }

    public final void fixBug() throws  {
        try {
            super.h.invoke(this, m3, (Object[])null);
        } catch (RuntimeException | Error var2) {
            throw var2;
        } catch (Throwable var3) {
            throw new UndeclaredThrowableException(var3);
        }
    }

    public final String toString() throws  {
        try {
            return (String)super.h.invoke(this, m2, (Object[])null);
        } catch (RuntimeException | Error var2) {
            throw var2;
        } catch (Throwable var3) {
            throw new UndeclaredThrowableException(var3);
        }
    }

    public final int hashCode() throws  {
        try {
            return (Integer)super.h.invoke(this, m0, (Object[])null);
        } catch (RuntimeException | Error var2) {
            throw var2;
        } catch (Throwable var3) {
            throw new UndeclaredThrowableException(var3);
        }
    }

    static {
        try {
            m1 = Class.forName("java.lang.Object").getMethod("equals", 
Class.forName("java.lang.Object"));
            m3 = 
Class.forName("com.dml.pattern.proxy.Engineer").getMethod("fixBug");
            m2 = Class.forName("java.lang.Object").getMethod("toString");
            m0 = Class.forName("java.lang.Object").getMethod("hashCode");
        } catch (NoSuchMethodException var2) {
            throw new NoSuchMethodError(var2.getMessage());
        } catch (ClassNotFoundException var3) {
            throw new NoClassDefFoundError(var3.getMessage());
        }
    }
}
```

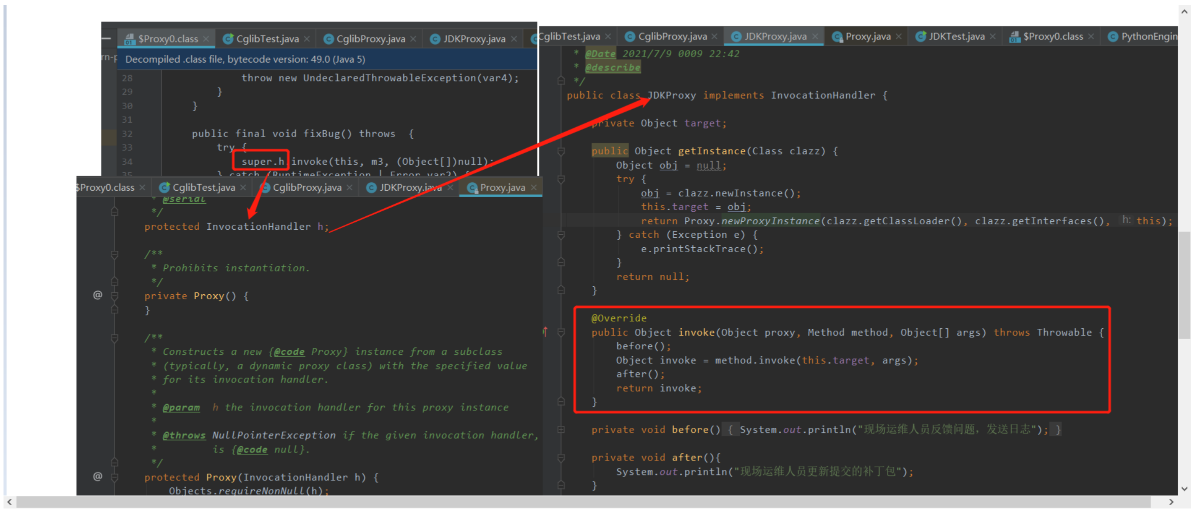
- 分析



```java
public final class $Proxy0 extends Proxy implements Engineer {
    private static Method m1;
    private static Method m3;
    private static Method m2;
    private static Method m0;

    public $Proxy0(InvocationHandler var1) throws  {
        super(var1);
    }
    public final boolean equals(Object var1) throws  {//... }
    public final void fixBug() throws  {
        try {
            super.h.invoke(this, m3, (Object[])null);
        } catch (RuntimeException | Error var2) {
            throw var2;
        } catch (Throwable var3) {
            throw new UndeclaredThrowableException(var3);
        }
    }
    public final String toString() throws  {//.... }

    public final int hashCode() throws  {//... }

    static {
        try {
            m1 = Class.forName("java.lang.Object").getMethod("equals", Class.forName("java.lang.Object"));
            m3 = Class.forName("com.dml.pattern.proxy.Engineer").getMethod("fixBug");
            m2 = Class.forName("java.lang.Object").getMethod("toString");
            m0 = Class.forName("java.lang.Object").getMethod("hashCode");
        } catch (NoSuchMethodException var2) {
```

```java
public class JDKTest {
    public static void main(String[] args) throws IOException {
        JDKProxy jdkProxy = new JDKProxy();
        Engineer engineer = (Engineer) jdkProxy.getInstance(JavaEngineer.class);
        engineer.fixBug();

    }
}
```

```java
     * the invocation handler for this proxy instance.
     * @serial
     */
    protected InvocationHandler h;

    /**
     * Prohibits instantiation.
     */
    private Proxy() {
    }

    /**
     * Constructs a new {@code Proxy} instance from a subclass
     * (typically, a dynamic proxy class) with the specified value
     * for its invocation handler.
     *
     * @param  h the invocation handler for this proxy instance
     *
     * @throws NullPointerException if the given invocation handler, {@code h},
     *         is {@code null}.
     */
    protected Proxy(InvocationHandler h) {
        Objects.requireNonNull(h);
        this.h = h;
    }
```



```java
            throw new UndeclaredThrowableException(var4);
        }
    }

    public final void fixBug() throws  {
        try {
            super.h.invoke(this, m3, (Object[])null);
        } catch (RuntimeException | Error var2) {
```

```java
     * @serial
     */
    protected InvocationHandler h;

    /**
     * Prohibits instantiation.
     */
    private Proxy() {
    }

    /**
     * Constructs a new {@code Proxy} instance from a subclass
     * (typically, a dynamic proxy class) with the specified value
     * for its invocation handler.
     *
     * @param  h the invocation handler for this proxy instance
     *
     * @throws NullPointerException if the given invocation handler,
     *         is {@code null}.
     */
    protected Proxy(InvocationHandler h) {
        Objects.requireNonNull(h);
```

```java
 * @Date 2021/7/9 0009 22:42
 * @describe
 */
public class JDKProxy implements InvocationHandler {

    private Object target;

    public Object getInstance(Class clazz) {
        Object obj = null;
        try {
            obj = clazz.newInstance();
            this.target = obj;
            return Proxy.newProxyInstance(clazz.getClassLoader(), clazz.getInterfaces(),  h: this);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        before();
        Object invoke = method.invoke(this.target, args);
        after();
        return invoke;
    }

    private void before() { System.out.println("现场运维人员反馈问题，发送日志"); }

    private void after(){
        System.out.println("现场运维人员更新提交的补丁包");
    }
```

自定义实现JDK的代理

- MLDInvocationHandler对应InvocationHandler

```java
package com.dml.pattern.proxy.custom;

import java.lang.reflect.Method;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/12 0012 22:10
 * @describe
 */
public interface MLDInvocationHandler {
    public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable;
}
```

- MLDProxy对应Proxy

```java
package com.dml.pattern.proxy.custom;

import javax.tools.JavaCompiler;
import javax.tools.StandardJavaFileManager;
import javax.tools.ToolProvider;
import java.io.File;
import java.io.FileWriter;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/12 0012 22:10
 * @describe
 */
public class MLDProxy {
    public static final String ln = "\r\n";
    public static Object newProxyInstance(ClassLoader classLoader, Class<?>
[] interfaces, MLDInvocationHandler h){
        try {
            //1、动态生成源代码.java文件
            String src = generateSrc(interfaces);

//            System.out.println(src);
            //2、Java文件输出磁盘
            String filePath = MLDProxy.class.getResource("").getPath();
//            System.out.println(filePath);
            File f = new File(filePath + "$Proxy0.java");
            FileWriter fw = new FileWriter(f);
            fw.write(src);
            fw.flush();
            fw.close();

            //3、把生成的.java文件编译成.class文件
            JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
            StandardJavaFileManager manage =
compiler.getStandardFileManager(null,null,null);
            Iterable iterable = manage.getJavaFileObjects(f);

            JavaCompiler.CompilationTask task =
compiler.getTask(null,manage,null,null,null,iterable);
            task.call();
            manage.close();

            //4、编译生成的.class文件加载到JVM中来
            Class proxyClass =
 classLoader.loadClass("com.dml.pattern.proxy.custom.$Proxy0");
            Constructor c =
proxyClass.getConstructor(MLDInvocationHandler.class);
            f.delete();

            //5、返回字节码重组以后的新的代理对象
            return c.newInstance(h);
        }catch (Exception e){
            e.printStackTrace();
        }
        return null;
    }

    private static String generateSrc(Class<?>[] interfaces){
        StringBuffer sb = new StringBuffer();
        sb.append("package com.dml.pattern.proxy.custom;" + ln);
        sb.append("import com.dml.pattern.proxy.Engineer;" + ln);
        sb.append("import java.lang.reflect.*;" + ln);
        sb.append("public class $Proxy0 implements " +
interfaces[0].getName() + "{" + ln);
        sb.append("MLDInvocationHandler h;" + ln);
        sb.append("public $Proxy0(MLDInvocationHandler h) { " + ln);
        sb.append("this.h = h;");
        sb.append("}" + ln);
        for (Method m : interfaces[0].getMethods()){
            Class<?>[] params = m.getParameterTypes();

            StringBuffer paramNames = new StringBuffer();
            StringBuffer paramValues = new StringBuffer();
```

```java
            StringBuffer paramClasses = new StringBuffer();

            for (int i = 0; i < params.length; i++) {
                Class clazz = params[i];
                String type = clazz.getName();
                String paramName = toLowerFirstCase(clazz.getSimpleName());
                paramNames.append(type + " " +  paramName);
                paramValues.append(paramName);
                paramClasses.append(clazz.getName() + ".class");
                if(i > 0 && i < params.length-1){
                    paramNames.append(",");
                    paramClasses.append(",");
                    paramValues.append(",");
                }
            }

            sb.append("public " + m.getReturnType().getName() + " " +
m.getName() + "(" + paramNames.toString() + ") {" + ln);
            sb.append("try{" + ln);
            sb.append("Method m = " + interfaces[0].getName() +
".class.getMethod(\"" + m.getName() + "\",new Class[]{" +
paramClasses.toString() + "});" + ln);
            sb.append((hasReturnValue(m.getReturnType()) ? "return " : "") +
getCaseCode("this.h.invoke(this,m,new Object[]{" + paramValues +
"})",m.getReturnType()) + ";" + ln);
            sb.append("}catch(Error _ex) { }");
            sb.append("catch(Throwable e){" + ln);
            sb.append("throw new UndeclaredThrowableException(e);" + ln);
            sb.append("}");
            sb.append(getReturnEmptyCode(m.getReturnType()));
            sb.append("}");
        }
        sb.append("}" + ln);
        return sb.toString();
    }


    private static Map<Class,Class> mappings = new HashMap<Class, Class>();
    static {
        mappings.put(int.class,Integer.class);
    }

    private static String getReturnEmptyCode(Class<?> returnClass){
        if(mappings.containsKey(returnClass)){
            return "return 0;";
        }else if(returnClass == void.class){
            return "";
        }else {
            return "return null;";
        }
    }

    private static String getCaseCode(String code,Class<?> returnClass){
        if(mappings.containsKey(returnClass)){
            return "((" + mappings.get(returnClass).getName() +  ")" + code
+ ")." + returnClass.getSimpleName() + "Value()";
        }
        return code;
    }

    private static boolean hasReturnValue(Class<?> clazz){
        return clazz != void.class;
    }

    private static String toLowerFirstCase(String src){
        char [] chars = src.toCharArray();
        chars[0] += 32;
        return String.valueOf(chars);
    }

}
```

- 测试
  - 测试代理类

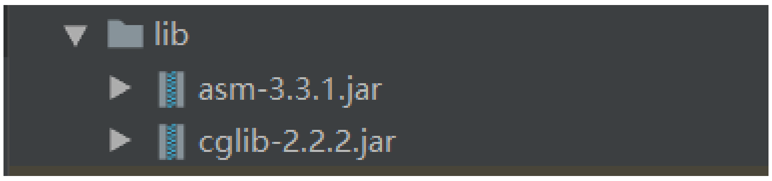
```java
package com.dml.pattern.proxy.custom;

import java.lang.reflect.Method;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:42
 * @describe
 */
public class CustomJDKProxy implements MLDInvocationHandler {

    private Object target;

    public Object getInstance(Class clazz) {
        Object obj = null;
        try {
            obj = clazz.newInstance();
            this.target = obj;
            return MLDProxy.newProxyInstance(clazz.getClassLoader(),
clazz.getInterfaces(), this);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
        before();
        Object invoke = method.invoke(this.target, args);
        after();
        return invoke;
    }

    private void before(){
        System.out.println("现场运维人员反馈问题，发送日志");
    }
    private void after(){
        System.out.println("现场运维人员更新提交的补丁包");
    }
}
```

- 测试类

  ```
  package com.dml.pattern.proxy.custom;

  import com.dml.pattern.proxy.Engineer;
  import com.dml.pattern.proxy.JavaEngineer;
  import java.io.IOException;

  /**
   * @author dml111727
   * @version 1.0
   * @Date 2021/7/9 0009 22:52
   * @describe
   */
  public class CustomJDKTest {
      public static void main(String[] args) throws IOException {
          CustomJDKProxy jdkProxy = new CustomJDKProxy();
          Engineer engineer = (Engineer)
  jdkProxy.getInstance(JavaEngineer.class);
          engineer.fixBug();
      }
  }
  ```

方式二 CGLB

导包：



代理对象

```
package com.dml.pattern.proxy.cglib;

import net.sf.cglib.proxy.Enhancer;
import net.sf.cglib.proxy.MethodInterceptor;
import net.sf.cglib.proxy.MethodProxy;

import java.lang.reflect.Method;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:57
 * @describe
 */
public class CglibProxy implements MethodInterceptor {
```

```
    public Object getInstance(Class clazz) {
        Enhancer enhancer = new Enhancer();
        // 继承那个类
        enhancer.setSuperclass(clazz);
        // 回调由代理执行
        enhancer.setCallback(this);
        return enhancer.create();
    }

    @Override
    public Object intercept(Object o, Method method, Object[] objects,
MethodProxy methodProxy) throws Throwable {
        before();
        Object obj = methodProxy.invokeSuper(o, objects);
        after();
        return obj;
    }

    private void before(){
        System.out.println("现场运维人员反馈问题，发送日志");
    }

    private void after(){
        System.out.println("现场运维人员更新提交的补丁包");
    }
}
```

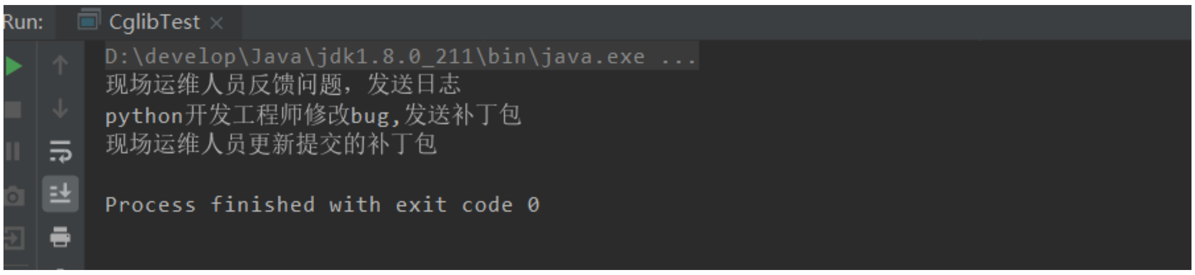
Python工程师

```
package com.dml.pattern.proxy;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 22:27
 * @describe java工程师
 */
public class PythonEngineer  {
    public void fixBug() {
        System.out.println("python开发工程师修改bug,发送补丁包");
    }
}
```

测试

```
package com.dml.pattern.proxy.cglib;

import com.dml.pattern.proxy.PythonEngineer;

/**
 * @author dml111727
 * @version 1.0
 * @Date 2021/7/9 0009 23:07
 * @describe
 */
public class CglibTest {
    public static void main(String[] args) {
        CglibProxy cglibProxy = new CglibProxy();
        PythonEngineer engineer = (PythonEngineer)
cglibProxy.getInstance(PythonEngineer.class);
        engineer.fixBug();
    }
}
```

结果:



```
D:\develop\Java\jdk1.8.0_211\bin\java.exe ...
现场运维人员反馈问题，发送日志
python开发工程师修改bug,发送补丁包
现场运维人员更新提交的补丁包

Process finished with exit code 0
```