链滴

代理（Proxy）设计模式

作者：vcjmhg
原文链接：https://ld246.com/article/1570452090726
来源网站：链滴

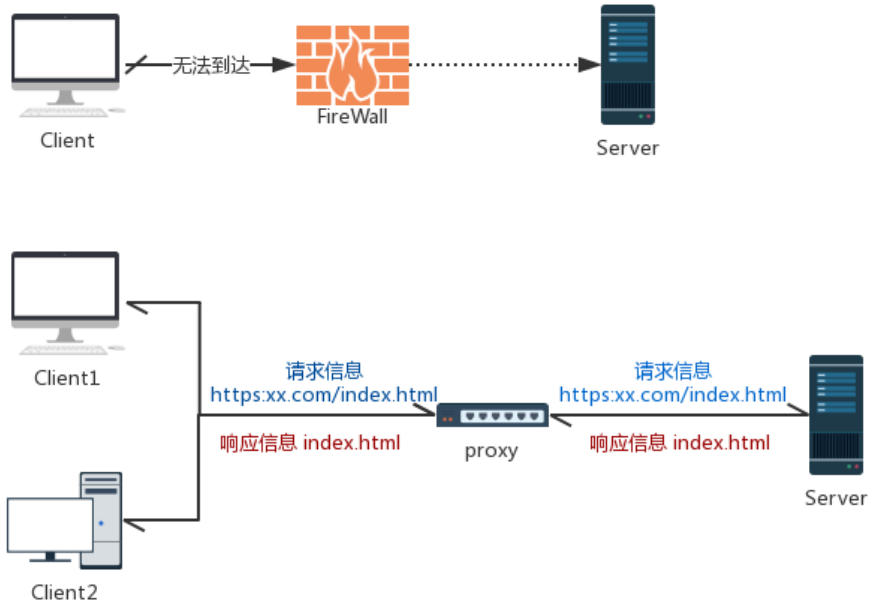
概述

正文开始之前我们先考虑一个问题：什么叫做代理(Proxy)?

按照维基百科定义:

代理（英语：Proxy）也称网络代理，是一种特殊的网络服务，允许一个网络终端（一般为客户端）过这个服务与另一个网络终端（一般为服务器）进行非直接的连接。一些网关、路由器等网络设备具网络代理功能。一般认为代理服务有利于保障网络终端的隐私或安全，防止攻击。

通俗讲代理就类似一个连接在客户端和服务器之间的桥梁，连通客户端和服务器之间的请求和响应，理的存在一方面可以保护服务器的安全，在代理部分可以对请求信息进行过滤，隔绝一部分非法的请信息吗，另一方面可以提高用户的访问速度，其具体功能可以借助下边的图来帮助理解。



原文链接：代理（Proxy）设计模式

如果你理解上述代理的概念，那么代理设计模式也就不难理解了。代理设计模式就是对上边上客户端代理-服务器三者链式关系的一种抽象，进而应用到软件开发中的一种通用设计模式。

代理设计模式有如下三个优点：

1. 保护真实对象
2. 让对象职责更加明确
3. 易于扩展

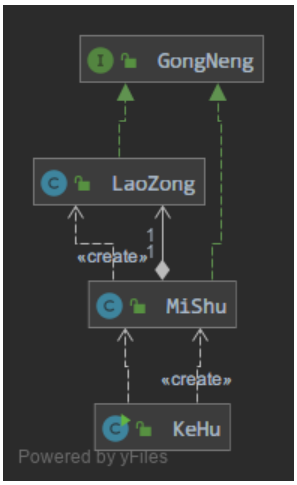
在java开发中代理设计模式有三种实现方法：

- 静态代理
- 动态代理 jdk实现
- 动态代理 cglib实现

下边我们分三种情况对这三种代理设计模式的实现进行讨论和分析

静态代理

UML类图



- KeHu :客户端
- MiShhu:中介
- LaoZong:服务器
- GongNeng:服务器和中介要同时实现的功能接口

代码实现

GongNeng的java代码：

```
/**
 * @program: TestBlog
 * @description:
 秘书和老总都要实现的功能接口
 * @author: vcjmhg
```

```
 * @create: 2019-10-07 16:31
 **/
public interface GongNeng {
    public void ZuoShengYi();
    public void eat();
}
```

Kehu的java代码

```
/**
 * @program: TestBlog
 * @description:
 客户相当于客服端
 * @author: vcjmhg
 * @create: 2019-10-07 16:31
 **/
public class KeHu {
    public static void main(String[] args) {
        MiShu miShu=new MiShu();
        miShu.ZuoShengYi();
    }
}
```

MiShu的java代码

```
/**
 * @program: TestBlog
 * @description:
 * @author: vcjmhg
 * @create: 2019-10-07 16:31
 **/
public class MiShu implements GongNeng{
    private LaoZong laoZong=new LaoZong();
    public void ZuoShengYi() {
        System.out.println("秘书：请问您预约来吗？");
        laoZong.ZuoShengYi();
        System.out.println("秘书备注访客信息");
    }

    public void eat() {
        System.out.println("秘书：请问您预约来吗？");
        laoZong.eat();
        System.out.println("秘书备注访客信息");
    }
}
```

LaoZong的java代码：

```
package proxy.staticproxy;

/**
 * @program: TestBlog
 * @description:
 * @author: vcjmhg
 * @create: 2019-10-07 16:31
```

```
 **/
public class LaoZong implements GongNeng{

    public void ZuoShengYi() {
        System.out.println("老总：谈个小项目！！");
    }

    public void eat() {
        System.out.println("老总：吃饭！！");
    }
}
```

运行结果为：

```
秘书：请问您预约来吗?
老总：谈个小项目！！
秘书备注访客信息

Process finished with exit code 0
```

代码地址

详细的代码可以参看github的上的代码

静态代理的不足

毫无疑问静态代理作为最容易实现或者说最直观的的代理设计模式的实现方式，代理模式具有的优点必然也具有，但另一方面它也有许多缺点：

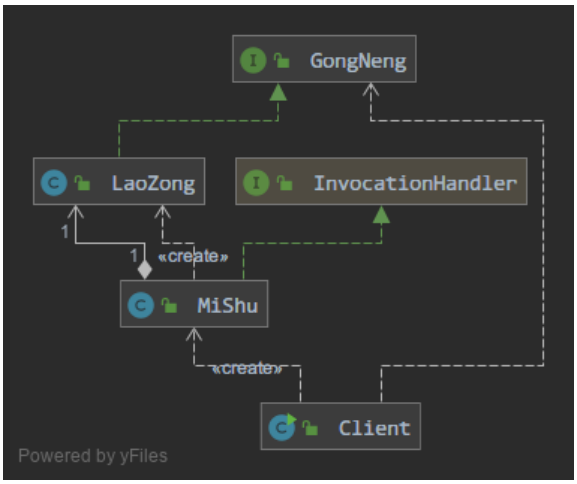
1. 代理类和委托类实现了相同的接口，代理类通过委托类实现了相同的方法。这样就出现了大量的代重复。如果接口增加一个方法，除了所有实现类需要实现这个方法外，所有代理类也需要实现此方法增加了代码维护的复杂度。

2. 代理对象只服务于一种类型的对象，如果要服务多类型的对象。势必要为每一种对象都进行代理，态代理在程序规模稍大时就无法胜任了。

为了解决该问题我们引入了动态代理

动态代理之jdk实现

UML类图



- Client :客户端
- MiShhu:中介
- LaoZong:服务器
- GongNeng:服务器和中介要同时实现的功能接口

代码实现

Client的java代码

```
/**
* @program: TestBlog
* @description:
* @author: vcjmhg
* @create: 2019-10-07 17:04
**/
public class Client {
   public static void main(String[] args) {
       //第一个参数:反射时使用的类加载器
       //第二个参数:Proxy需要实现什么接口
       //第三个参数:通过接口对象调用方法时,需要调用哪个类的invoke方法
       GongNeng gongneng = (GongNeng) Proxy.newProxyInstance(Client.class.getClassLoader
), new Class[]{GongNeng.class}, new MiShu());
       gongneng.eat();
   }
}
```

GongNeng的java代码：

```
public interface GongNeng {
   public void ZuoShengYi();
   public  void eat();
}
```

MiShu的java代码

```
/**
* @program: TestBlog
* @description:
```

```
* @author: vcjmhg
* @create: 2019-10-07 16:56
**/
public class MiShu implements InvocationHandler {
   private LaoZong laozong=new LaoZong() ;
   //代理类针对被代理对象类似的功能不需要重复实现多次
   public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
       System.out.println("预约时间");
       Object result = method.invoke(laozong, args);
       System.out.println("记录访客信息");
       return result;
   }
}
```

LaoZong的java代码：

```
/**
 * @program: TestBlog
 * @description:
 * @author: vcjmhg
 * @create: 2019-10-07 16:49
 **/
public class LaoZong implements GongNeng{
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }


    public void ZuoShengYi() {
        System.out.println("老总：谈生意");
    }

    public void eat() {
        System.out.println("老总：吃饭！！");
    }
}
```

运行结果为：

```
预约时间
老总：吃饭！！
记录访客信息

Process finished with exit code 0
```

利用JDK实现动态代理的优点

相比与静态代理，利用JDK实现动态代理的方式实现了代理类和功能接口之间的解耦。对于委托类如增加某个方法，对于代理类代码几乎可以不变，减少了代码的复杂性，使其更加易于维护。另一方面代理不同类型对象时可以实现代码一定程度的复用。

利用JDK实现动态代理的不足

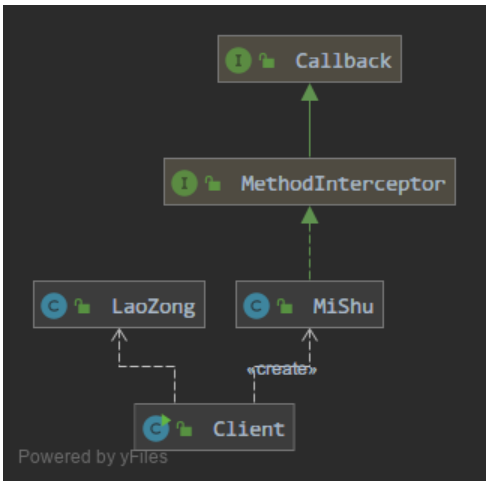
但是该方法实现动态代理也有一定不足，由于其内部借助反射实现代理设计模式，系统开销大效率低而且其委托类仍需实现功能接口，代码耦合性还是不够低。

代码地址

详细的代码可以参看github的上的代码

动态代理之cglib实现

UML类图



- Client :客户端
- MiShhu:中介
- LaoZong:服务器

代码实现

Client的java代码

```
public class Client {
   public static void main(String[] args) {
       Enhancer enhancer = new Enhancer();
       enhancer.setSuperclass(LaoZong.class);
       enhancer.setCallback(new MiShu());

       LaoZong laozong = (LaoZong) enhancer.create();
       laozong.chifan();

   }
}
```

原文链接：代理（Proxy）设计模式

MiShu的java代码：

```
public class MiShu implements MethodInterceptor{
    public Object intercept(Object arg0, Method arg1, Object[] arg2, MethodProxy arg3) throws
Throwable {
        System.out.println("预约时间");
        Object result = arg3.invokeSuper(arg0, arg2);
        System.out.println("备注");
        return result;
    }

}
```

LaoZong的java代码：

```
public class LaoZong {

    public void chifan() {
        System.out.println("吃饭");
    }

    public void mubiao() {
        System.out.println("目标");
    }
```

运行结果为：

```
预约时间
吃饭
备注

Process finished with exit code 0
```

利用cglib实现动态代理的优点

通过cglib方式几乎完美的解决来jdk方式所具有的缺点一方面cglib方式内部是通过字节码方式实现动代理，效率高，执行速度快；另一方面，该方式解耦了委托类和功能接口之间的耦合，提高了代码的活性。

代码地址

详细的代码可以参看github的上的代码