

# 一、代理定义

- 1 定义：给目标对象提供一个代理对象，并由代理对象控制对目标对象的引用。主要功能就是在不修改类源代码的情况下，增强类的方法。代理分为静态代理和动态代理。

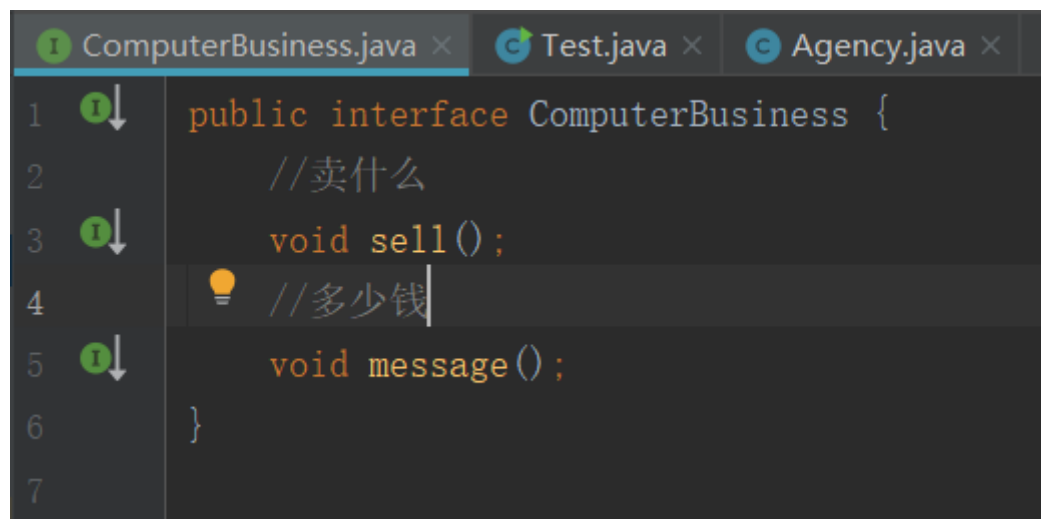
## 二、静态代理

### 1、代码思路

- 1 ①代理类与被代理类要实现相同的接口。
- 2
- 3 ②代理类中写构造方法用于接收被代理对象，然后对其进行增强。
- 4
- 5 ③客户端调用。

### 2、代码实现

#### (1) 接口



```
ComputerBusiness.java x Test.java x Agency.java x
1 public interface ComputerBusiness {
2     //卖什么
3     void sell();
4     //多少钱
5     void message();
6 }
7
```

#### (2) 代理类与实现类

##### ①代理类

```

ComputerBusiness.java × Test.java × Agency.java × SunBusiness.java × AppleBusiness.java ×
public class Agency implements ComputerBusiness {
    private ComputerBusiness business;
    //构造方法参数写接口来接收
    public Agency(ComputerBusiness business){
        this.business=business;
    }

    @Override
    public void sell() {
        System.out.println("执行sell方法前可以进行一系列逻辑操作，进行方法增强");
        business.sell();
        System.out.println("执行sell方法后可以进行一系列逻辑操作，进行方法增强");
    }

    @Override
    public void message() {
        System.out.println("我是代理，我要收取一部分代理费用");
        business.message();
        System.out.println("增强message方法后执行逻辑操作，进行方法增强。");
    }
}

```

## ②实现类

```

ComputerBusiness.java × Test.java × Agency.java × SunBusiness.java × AppleBusiness.java ×
1      public class SunBusiness implements ComputerBusiness {
2          @Override
3          public void sell() {
4              System.out.println("卖的是三星电脑");
5          }
6
7          @Override
8          public void message() {
9              System.out.println("我的三星笔记本价格是5000元");
10         }
11     }

```

## 实现类

```

ComputerBusiness.java × Test.java × Agency.java × SunBusiness.java × AppleBusiness.java ×
1      public class AppleBusiness implements ComputerBusiness {
2          @Override
3          public void sell() {
4              System.out.println("卖的是苹果电脑");
5          }
6
7          @Override
8          public void message() {
9              System.out.println("我的苹果笔记本价格为10000元");
10         }
11     }

```

## ③客户端

```

1 public class Client {
2     public static void main(String[] args) {
3         SunBusiness sun=new SunBusiness();
4         Agency agency=new Agency(sun);
5         agency.sell();
6         agency.message();
7         System.out.println("-----");
8         AppleBusiness apple=new AppleBusiness();
9         Agency agency2=new Agency(apple);
10        agency2.sell();
11        agency2.message();
12    }
13 }

```

执行sell方法前可以进行一系列逻辑操作，进行方法增强  
卖的是三星电脑  
执行sell方法后可以进行一系列逻辑操作，进行方法增强  
我是代理，我要收取一部分代理费用  
我的三星笔记本价格是5000元  
增强message方法后执行逻辑操作，进行方法增强。

-----

执行sell方法前可以进行一系列逻辑操作，进行方法增强  
卖的是苹果电脑  
执行sell方法后可以进行一系列逻辑操作，进行方法增强  
我是代理，我要收取一部分代理费用  
我的苹果笔记本价格为10000元  
增强message方法后执行逻辑操作，进行方法增强。

### 3、总结

- 1 可以看出，在代理类和真实类都实现一个接口的情况下，一个代理类对应一个接口，且一个代理类可以代理多个被代理对象。但缺点也很明显。如果真实类的要增强的方法非常的多，且其中增强的逻辑都差不多，那么代理类就要写非常多个方法去增强真实类方法，也会造成代码重复。如果真实对象有很多个的话，就要写很多个代理对象，这样就显的冗余。所以就有了动态代理的出现。

## 三、动态代理

### 1、定义

- 1 动态代理：无需自己去定义代理类，在内存中形成代理类。

### 2、操作步骤

- 1 1、真实对象和代理对象实现同一个接口
- 2 2. 代理对象 = Proxy.newProxyInstance();
- 3 3、在InvocationHandler中增强方法。
- 4 3. 使用代理对象调用方法。

### 3、具体代码

## (1) Computer接口

```
1 package proxy;
2
3 public interface ComputerBusiness {
4     String buy(double money);
5     void talk();
6 }
7
```

## (2) ComputerImp实现类

```
1 package proxy;
2
3 public class ComputerBusinessImp implements ComputerBusiness {
4     @Override
5     public String buy(double money) {
6         return "恭喜你买到电脑";
7     }
8
9     @Override
10    public void talk() {
11        System.out.println("买电脑，找我....");
12    }
13 }
14
```

## (3) Client类

```
1 package Proxy;
2
3 import java.lang.reflect.InvocationHandler;
4 import java.lang.reflect.Method;
5 import java.lang.reflect.Proxy;
6
7 public class Client {
8     public static void main(String[] args) {
9         ComputerBusinessImp business=new ComputerBusinessImp();
10        /*
11            被代理的对象必须要有接口。
12            使用Proxy调用newProxyInstance方法
13            三个参数写法比较固定
14            第一个参数类加载器:被代理的对象.getClass().getClassLoader()或者接口获
15            取的类加载器也可以。
16            第二个参数接口数组:被代理的对象.getClass().getInterfaces();
17            第三个参数处理器:匿名内部类,代理类调用方法时,总会触发这个匿名类里的
18            invoke方法。
19            */
20        /*
21            因为Proxy调用的方法所返回的值是一个实现了ComputerBusiness接口的代理对象。我们
22            不知道这个代理对象的类名叫什么,且proxy调用的方法返回值是Object,
23            所以我们需要对Object向下转型。如果我们知道代理对象的类名叫什么,我们只需要 "实现
24            类类名 x=(实现类类名)Proxy.newProxyInstance(...)", 这样既可。
25            但关键我们不知道实现类叫什么,但还是需要向下转型。因为代理对象和真实对象实现同一
26            个接口,所以我们可以向下转型为接口。
27        */
28    }
29 }
```

```

22     即ComputerBusiness business=
(ComputerBusiness)Proxy.newProxyInstance(...);这样的逻辑相当于
23     GrandFather g=new Son();
24     Father f=(Father)g;
25     f.talk();
26     */
27
28     ComputerBusiness agency = (ComputerBusiness)
Proxy.newProxyInstance(business.getClass().getClassLoader(),
business.getClass().getInterfaces(), new InvocationHandler() {
29
30         /*
31         当代理类调用方法时，总会触发invoke方法
32         第一个参数: proxy,代理对象
33         第二个参数:代理类调用的方法.被封装为一个对象。使用method.invoke(被
代理对象,接收到的参数)，用于执行真实对象的方法，返回值是Object。
34         第三个参数:代理类调用的方法，所传入的参数。
35         */
36
37         @Override
38         public Object invoke(Object proxy, Method method, Object[] args)
throws Throwable {
39
40             return method.invoke(business, args);
41
42         }
43     });
44
45     String buy = agency.buy(2323);
46     System.out.println(buy);
47     agency.talk();
48
49 }
50 }
51

```

```

7  public class Client {
8      public static void main(String[] args) {
9          ComputerBusinessImp business=new ComputerBusinessImp();
10         /*
11         被代理的对象必须要有接口。
12         使用Proxy调用newProxyInstance方法
13         三个参数写法比较固定
14         第一个参数类加载器:被代理的对象.getClass().getClassLoader()
15         第二个参数接口数组:被代理的对象.getClass().getInterfaces();
16         第三个参数处理器:匿名内部类，代理类调用方法时，总会触发这个匿名类里的invoke方法。
17         */
18         /*
19         因为Proxy调用的方法所返回的值是一个实现了ComputerBusiness接口的代理对象，我们不知道这个代理对象的类名叫什么，且Proxy调用的方法返回值是Object，
20         所以我们需要对Object向下转型。如果我们知道代理对象的类名叫什么，我们只需要“实现类类名 s=(实现类类名)Proxy.newProxyInstance(...)”，这样既可。
21         但关键我们不知道实现类叫什么，但还是需要向下转型。因为代理对象和真实对象实现同一个接口，所以我们可以向下转型为接口。
22         即ComputerBusiness business=(ComputerBusiness)Proxy.newProxyInstance(...);这样的逻辑相当于
23         GrandFather g=new Son();
24         Father f=(Father)g;
25         f.talk();
26         */
27
28         ComputerBusiness agency = (ComputerBusiness) Proxy.newProxyInstance(business.getClass().getClassLoader(), business.getClass().getInterfaces(), new InvocationHandler() {
29
30             /*
31             当代理类调用方法时，总会触发invoke方法
32             第一个参数: proxy,代理对象

```

```

33      第二个参数:代理类调用的方法,被封装为一个对象,使用method.invoke(被代理对象,接收到的参数),用于执行真实对象的方法,返回值是Object。
34      第三个参数:代理类调用的方法,所传入的参数。
35
36
37      @Override
38      public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
39
40          return method.invoke(business, args);
41      }
42
43  });
44
45  String buy = agency.buy( money: 2223);
46  System.out.println(buy);
47  agency.talk();
48
49  }
50
51

```

- 1 执行逻辑:当代理类调用方法时,会触发匿名内部类的invoke方法。通过invoke方法里的method.invoke(对象, 参数)执行被代理对象对应的方法并获取返回值Object对象.最后通过return method.invoke(对象, 参数)后,代理类调用的方法获取到了返回值。
- 2 agency调用了buy方法,触发了invoke,如果invoke里return的是null,则agency调用buy方法获取到的也是null.要想获取到值,只有method.invoke方法才能获取到.agency调用了talk方法,此方法的返回值是一个void,但触发invoke方法并通过method.invoke方法执行,最终还是可以执行里面的代码。但如果在invoke方法里打印一个void类型的方法会得到一个null。

## 执行结果

```

恭喜你买到电脑
买电脑, 找我....

```

## 方法返回值是void的情况

- 1 方法返回值为空的情况.调用method.invoke方法执行真实对象方法.因为调用了两次method.invoke方法,所以执行了两次。其中因为talk方法是void类型,所以输出为null。

```

*/
@Override
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    Object invoke = method.invoke(business, args);
    System.out.println(invoke);
    return method.invoke(business, args);
}

});

agency.talk();

```

```

买电脑, 找我....
null
买电脑, 找我....

```

## 4、增强方式

- (1) 增强参数列表
- (2) 增强返回值类型
- (3) 增强执行逻辑

```
ComputerBusiness agency = (ComputerBusiness) Proxy.newProxyInstance(business.getClass().getClassLoader(),
    business.getClass().getInterfaces(), new InvocationHandler() {
        @Override
        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
            //如果是buy方法, 则执行if语句里的代码
            if(method.getName().equals("buy")){
                //增强参数列表
                double arg = (double) args[0];
                arg=1.8*2;

                Object invoke = method.invoke(business, args);

                //增强返回值
                return invoke+"增强";
            }
            return method.invoke(business, args);
        }
    });

String buy = agency.buy( money: 123);
System.out.println(buy);
}
```

结果

```
D:\Java\jdk1.8.0\bin\
恭喜你买到电脑增强
```

## 四、数组为方法参数问题

调用s(...)这样传参是错误的,

```
s({1,2,3});

}
public static void s(int []i){
    System.out.println(i);
}
```

要这样传。

```
s(new int []{1,2,3});

}
public static void s(int []i){
    System.out.println(i);
}
```