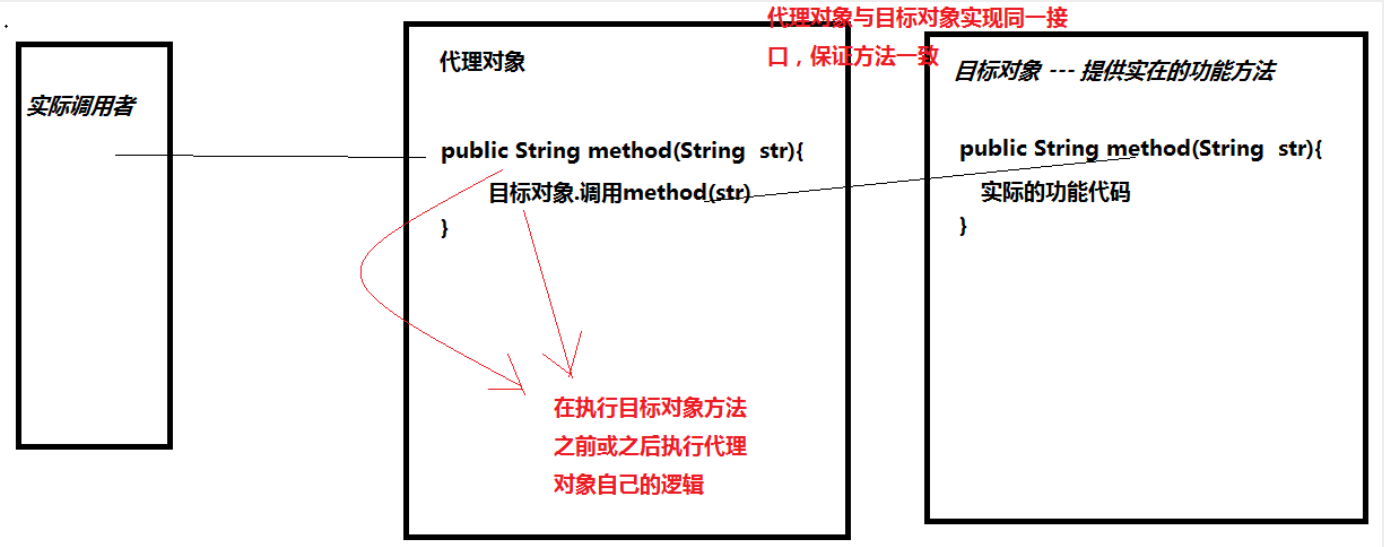
动态代理

一、什么是代理

大道理上讲代理是一种软件设计模式，目的地希望能做到代码重用。具体上讲，代理这种设计模式是通过不直接访问被代理对象的方式，而访问被代理对象的方法。这个就好比商户---->明星经纪人(代理)---->明星这种模式。我们可以不通过直接与明星对话的情况下，而通过明星经纪人(代理)与其产生间接对话。
在业务中使用动态代理，一般是为了给需要实现的方法添加预处理或者添加后续操作，但是不干预实现类的正常业务，把一些基本业务和主要的业务逻辑分离。我们一般所熟知的Spring的AOP原理就是基于动态代理实现的。



二、静态代理和动态代理

我们根据加载被代理类的时机不同，将代理分为静态代理和动态代理。如果我们在代码编译时就确定了被代理的类是哪一个，那么就可以直接使用静态代理；如果不能确定，那么可以使用类的动态加载机制，在代码运行期间加载被代理的类这就是动态代理，比如RPC框架和Spring AOP机制。

三、静态代理的实现

3.1 创建一个接口

```
1 package cn.scct.jingtai;
2
3 public interface Person {
4     public void sayHello(String content, int age);
5     public void sayGoodBye(boolean seeAgin, double time);
6 }
```

3.2 创建一个目标类实现该接口

```java
package cn.scct.jingtai;

public class Student implements Person{

    @Override
    public void sayHello(String content, int age) {

        System.out.println("student say hello" + content + " "+ age);
    }

    @Override
    public void sayGoodBye(boolean seeAgin, double time) {

        System.out.println("student sayGoodBye " + time + " "+ seeAgin);
    }

}
```

3.3 创建一个代理类实现该接口

```java
//静态代理类中实现了目标类的父接口，故而实现了目标类中的相同方法
//在代理类中我们可以在同样的方法中，调用目标类中的方法
//当然了，前提是要声明目标类或者目标类的父接口，显然声明目标类的父接口并利用多态调用会更好
//在调用之前，我们做一些操作，也就成了代理的作用，即代码增强作用当然，在调用之后也行
public class ProxyTest implements Person{

    private Person o;

    public ProxyTest(Person o){
        this.o = o;
    }


    @Override
    public void sayHello(String content, int age) {

        System.out.println("ProxyTest sayHello begin");
        //在代理类的方法中 间接访问被代理对象的方法
        o.sayHello(content, age);
        System.out.println("ProxyTest sayHello end");
    }

    @Override
    public void sayGoodBye(boolean seeAgin, double time) {

        System.out.println("ProxyTest sayGoodBye begin");
        //在代理类的方法中 间接访问被代理对象的方法
        o.sayGoodBye(seeAgin, time);
        System.out.println("ProxyTest sayGoodBye end");
    }
```

```java
    public static void main(String[] args) {

        //s为被代理的对象，某些情况下 我们不希望修改已有的代码，我们采用代理来间接访问
        Student s = new Student();
        //创建代理类对象
        ProxyTest proxy = new ProxyTest(s);//this.o=s;多态

        //调用代理类对象的方法
        proxy.sayHello("welcome to java", 20);
        System.out.println("******");
        //调用代理类对象的方法
        proxy.sayGoodBye(true, 100);

    }

}
```

结果

```
Console
<terminated> ProxyTest [Java Application] D:\runners\JavaJDK\JRE7\bin\javaw.ex
ProxyTest sayHello begin
student say hellowelcome to java 20
ProxyTest sayHello end
******
ProxyTest sayGoodBye begin
student sayGoodBye 100.0 true
ProxyTest sayGoodBye end
```

很显然，就是静态代理就是实现同一接口和利用了多态的性质，不同接口实现中的调用与被调用关系实现代码的增强。

我更愿意称双胞胎代理。

四、动态代理

4.1 目标类的接口

```java
package cn.scct.dongtai;

public interface Subject {
    void hello(String param);
}
```

4.2 目标类的实现

```java

public class SubjectImpl implements Subject {

    public void hello(String param) {
        System.out.println("hello  " + param);
    }
}
```

4.3 目标类实现InvocationHandler接口

```java
package cn.scct.dongtai;

import java.lang.reflect.InvocationHandler;
//InvocationHandler 是代理实例的调用处理程序 实现的接口。


public class SubjectProxy implements InvocationHandler {
    private Subject subject;

    public SubjectProxy(Subject subject) {
        this.subject = subject;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        //这是获取之前的加工
        System.out.println("--------------begin-------------");
        //利用反射技术而已，获得了subject的方法
        Object invoke = method.invoke(subject, args);
        //这是获取之后的加工
        System.out.println("--------------end-------------");
        return invoke;
    }
}
```

不再和目标类是兄弟，但是利用反射技术还是能调用你的方法，在调用前后我可以进行增强

4.4 利用反射技术实现目标类方法的增强

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

public class ProxyTest {
    public static void main(String[] args) {
        Subject subject = new SubjectImpl();//目标类，用多态调用
        InvocationHandler subjectProxy = new SubjectProxy(subject);//代理类，用多态调用

        //反射技术
        Subject proxyInstance = (Subject) Proxy.newProxyInstance(subjectProxy.getClass().getClassLoader(), //代理类加载
                subject.getClass().getInterfaces(), //目标类的接口，如果有多个以数组的形式出现
                subjectProxy);//代理类的实例
        proxyInstance.hello("world");
    }
        //得到实例，调用方法，调用的是InvocationHandle实现接口的增强后的代码

}
```