代理

公告

今天学习内容：

- 代理
  - 静态代理
  - 动态代理
  - Cglib代理
  - 常见面试问题
- 课后练习

代理

代理是 Java SE 1.3 新增加的一种特性，利用代理可以在运行时创建一个实现了一组给定接口的新类，通过代理类实现对目标类的调用。

代理是一种设计模式，它可以在目标对象的基础上，增加一些额外的功能，扩展目标对象的功能而不用修改已经存在的代码。

代理主要有3种方式来实现，下边分别来介绍：

静态代理

静态代理需要定义接口或父类，被代理对象与代理对象一起实现相同的接口或者继承相同的父类。

例如：我们现在有一个功能用户信息保存(直接打印出来模拟保存)，代码如下：

```
public class User {
    private String name;
    private Integer age;

    public String getName() {
        return name;
    }

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

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
```

```
    @Override
    public String toString() {
        StringBuilder tmp = new StringBuilder();

        tmp.append("姓名: ");
        if (name != null && name.length() > 0) {
            tmp.append(name);
        }
        tmp.append(',');
        tmp.append("年龄: ");
        if (age != null) {
            tmp.append(age);
        }

        return tmp.toString();
    }
}

public interface UserDao {
    void save(User user);
}

public class UserDaoImpl implements UserDao {

    @Override
    public void save(User user) {
        System.out.println("保存用户信息: " + user.toString());
    }

}

public class UserDaoTest {
    public static void main(String[] args) {
        User user = new User();
        user.setName("张三");
        user.setAge(20);

        UserDao dao = new UserDaoImpl();
        dao.save(user);
    }
}
```

如上述代码所示，因为调用方无法保证 User 实例中的姓名 name 有值、年龄 age 在有效范围之内 0~200。我们需要增加一个验证，name 不能为空且 age 在 0~200范围内时才进行保存操作；并且不允许修改 UserDaoImpl 类的代码。

那么我们怎么来实现？可以通过创建一个 UserDaoImpl 的代理类来实现，如下所示：

```
public class UserDaoProxy implements UserDao {
    private UserDao target;
```

```
    public UserDaoProxy(UserDao dao) {
        target = dao;
    }

    @Override
    public void save(User user) {
        if (user.getName() == null || user.getName().length() == 0) {
            System.out.println("name 为空，不保存");
            return;
        }
        if (user.getAge() == null || user.getAge() < 0 || user.getAge() >
200) {
            System.out.println("age 不在有效范围 0~200 内，不保存");
            return;
        }
        target.save(user);
    }
}

public class UserDaoTest {
    public static void main(String[] args) {
        User user = new User();
        user.setName("张三");
        user.setAge(-1);

        UserDao dao = new UserDaoImpl();

        UserDaoProxy proxy = new UserDaoProxy(dao);
        proxy.save(user);
    }
}
```

上述代码演示的是静态代理，它可以实现对目标对象的扩展而不用修改目标对象，但是也存在一些缺点：

- 代理对象和目标对象需要实现一样的接口或继承自相同父类，这样会存在很多代理类。
- 一旦接口或父类增减了方法，那么代理对象和目标对象都要做相应的修改。

那么如何解决上述问题，这就需要用到我们下边要讲到的动态代理。

动态代理

动态代理是 JDK 提供的一个特性，被代理对象必须是实现了一组接口。创建代理类需要用到 JDK 提供的 `Proxy.newProxyInstance` 方法，它的语法为：

```
public static Object newProxyInstance(ClassLoader loader,
                                       Class<?>[] interfaces,
                                       InvocationHandler h)
```

该方法是一个静态方法，接收三个参数：

- `ClassLoader loader` 当前目标对象使用的类加载器
- `Class<?>[] interfaces` 目标对象实现的接口类型，1 个或多个
- `InvocationHandler h` 事件处理，执行目标对象的方法时，自动触发事件处理器的 invoke 方法，并将方法和参数信息作为参数传入

下边我们使用动态代理方式来实现一下上述用户信息保存示例，代码如下（新增加的代码）：

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

public class CheckHandler implements InvocationHandler {
    private Object target;

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

    private boolean checkSaveUserParam(String methodName, Object[] args) {
        if (!(target instanceof UserDao)) {
            return true;
        }
        if (!"save".equals(methodName)) {
            return true;
        }
        if (args == null || args.length < 1 || !(args[0] instanceof User))
{
            return true;
        }
        User user = (User) args[0];

        if (user.getName() == null || user.getName().length() == 0) {
            System.out.println("name 为空，不保存");
            return false;
        }
        if (user.getAge() == null || user.getAge() < 0 || user.getAge() >
200) {
            System.out.println("age 不在有效范围 0~200 内，不保存");
            return false;
        }

        return true;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args)
throws Throwable {
        if (checkSaveUserParam(method.getName(), args)) {
            return method.invoke(target, args);
        }
        return null;
    }
}
```

```
UserDao dao = new UserDaoImpl();

ClassLoader classLoader = dao.getClass().getClassLoader();
Class[] interfaces = dao.getClass().getInterfaces();
CheckHandler handler = new CheckHandler(dao);

UserDao proxy = (UserDao) Proxy.newProxyInstance(classLoader, interfaces,
handler);

User user = new User();
user.setName("张三");
user.setAge(1);

proxy.save(user);
```

根据上述示例代码，我们可以看出，代理对象不需要实现接口，但是目标对象必须是实现了接口，否则就不能使用动态代理。

动态代理是 JDK 提供的一个特性，因此我们也可称为 JDK 代理。

Cglib代理

对于没有实现任何接口的类，我们需要对其进行代理时，可以使用 `cglib` 等开源方案来实现。

cglib 通过在内存中建立一个子类对象从而实现对目标对象的代理扩展，我们也可以称为是子类代理。cglib 代理有以下一些特性：

- cglib 是一个强大的高性能代码生成包，它在运行时扩展 Java 类或实现 Java 接口，它被广泛用于支持 AOP 的一些框架（比如：Spring AOP），提供方法的拦截（interception）。
- cglib 底层通过使用字节处理框架 ASM 来转换字节码并生成新的类。
- 被代理的类不能声明为 final。
- 目标对象的方法如果声明为 final 或 static，则不会被拦截。
- Spring 包中已经包含 cglib 的功能。在 Sping AOP 中，如果被代理对象是基于接口实现的，则使用 JDK 代理；否则用 Cglib 代理。

下边我们使用 cglib 方式来改写一下前边的示例：

首先在 pom.xml 中加入 cglib 依赖。

```
<dependency>
    <groupId>cglib</groupId>
    <artifactId>cglib-nodep</artifactId>
    <version>3.3.0</version>
</dependency>
```

或者引入 spring-core 依赖。

```

```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.2.2.RELEASE</version>
</dependency>
```

改写为 cglib 代理相关代码：

```
public class CgligProxy implements MethodInterceptor {
    private Object target;

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

    private boolean checkSaveUserParam(String methodName, Object[] args) {
        if (!(target instanceof UserDao)) {
            return true;
        }
        if (!"save".equals(methodName)) {
            return true;
        }
        if (args == null || args.length < 1 || !(args[0] instanceof User))
{
            return true;
        }
        User user = (User) args[0];

        if (user.getName() == null || user.getName().length() == 0) {
            System.out.println("name 为空，不保存");
            return false;
        }
        if (user.getAge() == null || user.getAge() < 0 || user.getAge() >
200) {
            System.out.println("age 不在有效范围 0~200 内，不保存");
            return false;
        }

        return true;
    }

    public Object getProxyInstance() {
        Enhancer en = new Enhancer();
        en.setSuperclass(target.getClass());
        en.setCallback(this);
        return en.create();
    }

    @Override
    public Object intercept(Object o, Method method, Object[] args,
MethodProxy proxy) throws Throwable {
```

```
            if (checkSaveUserParam(method.getName(), args)) {
                return method.invoke(target, args);
            }
            return null;
        }
    }

    UserDao dao = new UserDaoImpl();

    User user = new User();
    user.setName("张三");
    user.setAge(-1);

    UserDao proxy = (UserDao) new CgligProxy(dao).getProxyInstance();
    proxy.save(user);
```

常见面试问题

什么是代理？什么是动态代理？

动态代理和cglib代理的区别？

简述一下 Spring AOP 的实现？

课后练习

1. 自己编写练习一下课程中的示例和练习。
2. 根据课程内容回答常见面试问题。