代理模式实验

**【实验目的及要求】**

1. 掌握代理模式的结构与实现；
2. 学会使用代理模式开发应用程序；
3. 明白代理模式的扩展应用。

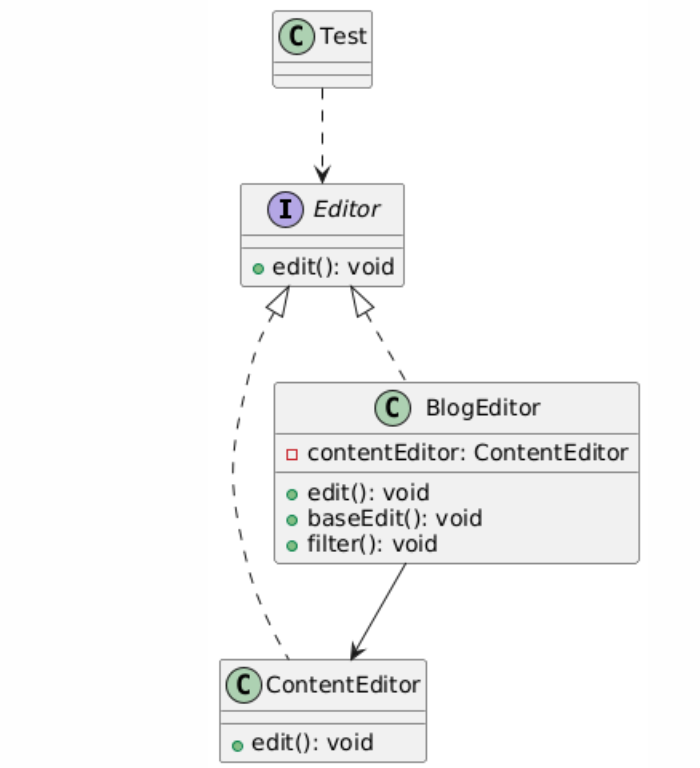
**【实验内容】**

**在软件开发中，已定义了一个编辑器接口Editor，接口中的edit()方法实现编辑功能。****ContentEditor类实现了Editor接口，其edit()方法提供了正文内容的编辑功能。博客系统需要博文编辑功能，包括博文基本信息（base），博文正文，敏感词过滤（filter）等。请使用代理模式实现博文编辑类****BlogEditor。**

**【实验步骤】**

1. **类图**

**画出类图。**

****

1. **实现代码**

**(1) 抽象主题角色**

public interface Editor {  
 public void edit();  
}

**(2) 真实主题角色**

**public class ContentEditor implements Editor {  
 @Override  
 public void edit() {  
 System.*out*.println("正文内容的编辑");  
 }  
}**

**(3) 代理主题角色**

**public class BlogEditor implements Editor {  
// 维护一个对真实主题的引用  
 private ContentEditor contentEditor = new ContentEditor();  
  
 @Override  
 public void edit() {  
 baseEdit();  
 contentEditor.edit();  
 filter();  
 }  
  
 public void baseEdit() {  
 System.*out*.println("博文基本信息编辑");  
 }  
  
 public void filter() {  
 System.*out*.println("敏感词过滤");  
 }  
}**

**(4)工具类及配置文件**

**XMLUtil类**

**public class XMLUtil {  
 //该方法用于从XML配置文件中提取具体类类名，并返回一个实例对象  
 public static Object getBean() {  
 try {  
 //创建DOM文档对象  
 DocumentBuilderFactory dFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = dFactory.newDocumentBuilder();  
 Document doc;  
 doc = builder.parse(new File("src//com//fancy//experiment13\_proxy//config.xml"));  
  
 //获取包含类名的文本结点  
 NodeList nl = doc.getElementsByTagName("className");  
 Node classNode = nl.item(0).getFirstChild();  
 String cName = classNode.getNodeValue();  
  
 //通过类名生成实例对象并将其返回  
 Class c = Class.*forName*(cName);  
 Object obj = c.newInstance();  
 return obj;  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
}**

**配置文件config.xml**

**<?xml version="1.0"?>  
<config>  
 <className>com.fancy.experiment13\_proxy.BlogEditor</className>  
</config>**

**(5)测试类**

**Test**

**package** design.proxy;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Editor editor;

editor = (Editor)XMLUtil.getBean();

editor.edit();

}

}

**【问题】**

1. 修改上面代码，采用动态代理模式实现。

实现逻辑：动态代理需要自定义请求处理类，继承InvocationHandle，并且实现invoke方法加上相应的处理逻辑，添加一个有参函数注入不同的真实主题对象；在客户端的调用中，使用Proxy.newProxyInstance来处理不同类型的主题对象，抽象主题和真实主题可能会有很多，但是自定义请求处理类通常只需要定义一类，如果处理逻辑相似，只是调用的主题对象不同的情况下。

实现代码：

抽象主题类：

public interface Editor {  
 public void edit();  
}

真实主题类：

public class ContentEditor implements Editor {  
 @Override  
 public void edit() {  
 System.*out*.println("正文内容的编辑");  
 }  
}

自定义请求处理程序类，实现InvocationHandler接口  
public class BlogEditor implements InvocationHandler {  
 private Object object;  
  
 public BlogEditor() {  
 }  
  
// 有参构造函数，注入真实代理对象  
 public BlogEditor(Object object) {  
 this.object = object;  
 }  
  
 public void baseEdit() {  
 System.*out*.println("博文基本信息编辑");  
 }  
  
 public void filter() {  
 System.*out*.println("敏感词过滤");  
 }  
// 在invoke()中调用真实主题对象的方法  
 @Override  
 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
 baseEdit(); //额外的处理  
 Object result = method.invoke(object, args); //调用真实主题对象的方法  
 filter(); //额外的处理  
 return result;  
 }  
}

客户端类：

public class Test {

public static void main(String[] args) {

InvocationHandler invocationHandler = null;

Editor contentEditor = new ContentEditor(); //具体主题类

invocationHandler = new BlogEditor(contentEditor); //在自定义请求处理程序类中放入具体主题类!!关键处理

Editor proxy = null; // 抽象主题类型的 动态创建的代理类实例

proxy = (Editor) Proxy.newProxyInstance(

Editor.class.getClassLoader(), //代理类（抽象主题）的类加载器

new Class[]{Editor.class}, //代理类（真实主题）的接口列表

invocationHandler //自定义请求处理程序类，实现了InvocationHandler接口

);

proxy.edit(); //直接就是抽象类中定义的方法，但实际会调用BlogEditor处理过的方法

}

}

2. 目前已有第三方提供的功能强大的编辑器插件（例如KindEditor）。假设有KindEditor类中的html()方法实现内容编辑功能，请问实现上述的博文编辑功能，需要使用那些模式。具体给出代码。

方式一：使用动态代理模式。只需要增加KindEditor为抽象主题类，然后再多加一个具体类，修改客户端代码就可以了，客户端需要重新调用不同的方式

方式二：外观模式+动态代理模式，不用修改客户端代码，也不用修改其他代码，更换了编辑内容的方式，采用外观模式，可以通过修改配置文件更换不同的方式，客户端依然是只调用edit方法

方式二的代码：

public class KindEditor implements Editor{  
 public void html(){  
 System.*out*.println("使用KindEditor插件的html方法编辑内容");  
 };  
  
 @Override  
 public void edit() {  
 html();  
 }  
}

<?xml version="1.0"?>  
<config>  
 <className>com.fancy.experiment13\_proxy.active\_proxy.KindEditor</className>  
</config>

public class XMLUtil {  
 public static Object getBean() {  
 try {  
 //创建DOM文档对象  
 DocumentBuilderFactory dFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = dFactory.newDocumentBuilder();  
 Document doc;  
 doc = builder.parse(new File("src//com//fancy//experiment13\_proxy//active\_proxy//config.xml"));  
  
 //获取包含类名的文本结点  
 NodeList nl = doc.getElementsByTagName("className");  
 Node classNode = nl.item(0).getFirstChild();  
 String cName = classNode.getNodeValue();  
  
 //通过类名生成实例对象并将其返回  
 Class c = Class.*forName*(cName);  
 Object obj = c.newInstance();  
 return obj;  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
}

public class Test {  
 public static void main(String[] args) {  
 InvocationHandler invocationHandler = null;  
  
 Editor realEditor = null;  
 realEditor = (Editor) XMLUtil.*getBean*();  
 invocationHandler = new BlogEditor(realEditor); //在自定义请求处理程序类中放入具体主题类!!关键处理  
 Editor proxy = null; // 抽象主题类型的 动态创建的代理类实例  
  
 proxy = (Editor) Proxy.*newProxyInstance*(  
 Editor.class.getClassLoader(), //代理类（抽象主题）的类加载器  
 new Class[]{Editor.class}, //代理类（真实主题）的接口列表  
 invocationHandler //自定义请求处理程序类，实现了InvocationHandler接口  
 );  
  
 proxy.edit(); //直接就是抽象类中定义的方法，但实际会调用BlogEditor处理过的方法  
  
 }  
}

**【总结】**

|  |  |
| --- | --- |
| 实验总结  （不少于100字） | **实验总结：**  在本次实验中，我深入理解了代理模式的结构与实现，并学会了如何将其应用于实际开发中。通过代理模式，我能够在不修改现有代码的情况下，增强或修改方法的功能，如在 KindEditor 类中插入博文基本信息编辑和敏感词过滤的操作。实验展示了如何使用动态代理模式创建代理对象，并通过外观模式简化客户端与系统内部复杂操作的交互，提供了更清晰的结构和更高的扩展性。通过这一过程，我不仅掌握了代理模式的基本应用，还理解了其在开发中如何扩展应用，如添加新的功能而不影响已有的系统代码，增强了我对设计模式的理解与实际应用能力。 |
| 学号 | 22154A206 |
| 姓名 | 冯艳芳 |
| 成绩 |  |