1. AOP概念

面向切面编程

1. AOP和OOP区别

AOP是面向切面编程

OOP是面向对象编程

1. JDK动态代理和CGLIB动态代理

JDK动态代理：JDK原生动态代理，代理对象针对的是接口

CGLIB动态代理：代理对象，可以是接口，也可以是实现类

1. AOP底层实现代码
   1. JDK动态代理

创建JDK代理工厂

public class UserProxy implements InvocationHandler {  
 //目标对象  
 private Object target;  
 public UserProxy(Object target) {  
 this.target = target;  
 }  
 public Object createProxy(){  
 ClassLoader loader = target.getClass().getClassLoader();  
 Class<?>[] interfaces = target.getClass().getInterfaces();  
 return Proxy.*newProxyInstance*(loader,interfaces,this);  
 }  
 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
 //增强方法  
 System.*out*.println("日志打印。。");  
 return method.invoke(target,args);  
 }  
}

Test

UserProxy proxy = new UserProxy(userService);  
UserService us = (UserService) proxy.createProxy();

* 1. CGLIB动态代理

创建CGLIB代理工厂

public class CglibFactory implements MethodInterceptor {  
 //代理对象  
 private Object target;  
 public CglibFactory(Object target) {  
 this.target = target;  
 }  
 //创建代理对象方法  
 public Object createProxy(){  
 //enhancer对象  
 Enhancer enhancer = new Enhancer();  
 //设置代理对象的class  
 enhancer.setSuperclass(target.getClass());  
 enhancer.setCallback(this);  
 //返回代理对象  
 return enhancer.create();  
 }  
 public Object intercept(Object proxy, Method method, Object[] args, MethodProxy methodProxy) throws Throwable {  
 System.*out*.println("记录日志。。");  
 return method.invoke(target,args);  
 }  
}

Test

CglibFactory proxy = new CglibFactory(userService);  
UserServiceImpl us = (UserServiceImpl) proxy.createProxy();

CglibFactory proxy = new CglibFactory(userService);  
UserService us = (UserService) proxy.createProxy();

1. 传统的springAOP实现
   1. 通过实现5个增强通知的接口

前置通知：org.springframework.aop.MethodBeforeAdvice

后置通知：org.springframework.aop.AfterReturningAdvice

环绕通知：org.aopalliance.intercept.MethodInterceptor

异常抛出通知：org.springframework.aop.ThrowsAdvice

引介通知：org.springframework.aop.IntroductionAdvisor

* 1. 通知代码

public class UserServiceHelper implements MethodBeforeAdvice, AfterReturningAdvice, MethodInterceptor {  
 public Object invoke(MethodInvocation invocation) throws Throwable {  
 System.*out*.println("执行前通知。。");  
 Object proceed = invocation.proceed();  
 System.*out*.println("执行后通知。。");  
 return proceed;  
 }  
 public void afterReturning(Object returnValue, Method method, Object[] args, Object target) throws Throwable {  
 System.*out*.println("后置通知。。");  
 }  
 public void before(Method method, Object[] args, Object target) throws Throwable {  
 System.*out*.println("前置通知。。");  
 }  
}

* 1. 配置文件的配置代码

<!--配置增强对象-->  
<bean id="userService" class="com.java.service.UserServiceImpl"></bean>  
<!--配置增强通知-->  
<bean id="helper" class="com.java.helper.UserServiceHelper"></bean>  
<!--配置切点-->  
<bean id="pointcut" class="org.springframework.aop.support.NameMatchMethodPointcut">  
 <!--配置需要增强方法-->  
 <property name="mappedNames">  
 <list>  
 <value>add</value>  
 </list>  
 </property>  
</bean>  
<!--配置切面-->  
<bean id="aspect" class="org.springframework.aop.support.DefaultPointcutAdvisor">  
 <!--配置切点-->  
 <property name="pointcut" ref="pointcut"></property>  
 <!--配置增强-->  
 <property name="advice" ref="helper"></property>  
</bean>  
<!--配置代理对象-->  
<bean id="proxy" class="org.springframework.aop.framework.ProxyFactoryBean">  
 <!--设置目标对象-->  
 <property name="target" ref="userService"></property>  
 <!--配置切面-->  
 <property name="interceptorNames">  
 <list>  
 <value>aspect</value>  
 </list>  
 </property>  
 <!--配置代理对象的接口-->  
 <property name="proxyInterfaces" value="com.java.service.UserService"></property>  
</bean>

Test

UserService userService = (UserService) applicationContext.getBean("proxy");

* 1. 引入aop命名空间

xmlns:aop="http://www.springframework.org/schema/aop"

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd

配置文件的配置代码

<!--false或者不写-默认使用JDK动态代理 true-使用CGLIB动态代理-->  
<aop:config>  
 <!--配置切入点-->  
 <aop:pointcut id="pointCut" expression="execution(\* com.java..\*(..))"/>  
 <!--配置切面-->  
 <aop:advisor advice-ref="aspect" pointcut-ref="pointCut"/>  
</aop:config>

Test

UserService userService = (UserService) applicationContext.getBean("userService");

1. 基于AspectjAOP实现

通知类

public class AspectUserServiceHelper {  
 public void before() {  
 System.*out*.println("前置通知。。");  
 }  
  
 public void after() {  
 System.*out*.println("后置通知。。");  
 }  
  
 public Object around(ProceedingJoinPoint pj) throws Throwable {  
 System.*out*.println("执行前通知。。");  
 Object value = pj.proceed();  
 System.*out*.println("执行后通知。。");  
 return value;  
 }  
}

配置文件配置信息

<aop:config>  
 <aop:pointcut id="pointCut" expression="execution(\* com.java..\*(..))"></aop:pointcut>  
 <aop:aspect ref="aspect">  
 <aop:before method="before" pointcut-ref="pointCut"/>  
 <aop:after-returning method="after" pointcut-ref="pointCut"/>  
 <aop:around method="around" pointcut-ref="pointCut"/>  
 </aop:aspect>  
</aop:config>