**Spring实现AOP的4种方式**

  先了解AOP的相关术语:  
1.通知(Advice):  
通知定义了切面是什么以及何时使用。描述了切面要完成的工作和何时需要执行这个工作。  
2.连接点(Joinpoint):  
程序能够应用通知的一个“时机”，这些“时机”就是连接点，例如方法被调用时、异常被抛出时等等。  
3.切入点(Pointcut)  
通知定义了切面要发生的“故事”和时间，那么切入点就定义了“故事”发生的地点，例如某个类或方法的名称，Spring中允许我们方便的用正则表达式来指定  
4.切面(Aspect)  
通知和切入点共同组成了切面：时间、地点和要发生的“故事”  
5.引入(Introduction)  
引入允许我们向现有的类添加新的方法和属性(Spring提供了一个方法注入的功能）  
6.目标(Target)  
即被通知的对象，如果没有AOP,那么它的逻辑将要交叉别的事务逻辑，有了AOP之后它可以只关注自己要做的事（AOP让他做爱做的事）  
7.代理(proxy)  
应用通知的对象，详细内容参见设计模式里面的代理模式  
8.织入(Weaving)  
把切面应用到目标对象来创建新的代理对象的过程，织入一般发生在如下几个时机:  
(1)编译时：当一个类文件被编译时进行织入，这需要特殊的编译器才可以做的到，例如AspectJ的织入编译器  
(2)类加载时：使用特殊的ClassLoader在目标类被加载到程序之前增强类的字节代码  
(3)运行时：切面在运行的某个时刻被织入,SpringAOP就是以这种方式织入切面的，原理应该是使用了JDK的动态代理技术  
  
Spring提供了4种实现AOP的方式：  
1.经典的基于代理的AOP  
2.@AspectJ注解驱动的切面  
3.纯POJO切面  
4.注入式AspectJ切面  
  
首先看经典的基于代理的AOP:  
Spring支持五种类型的通知：  
Before(前)  org.apringframework.aop.MethodBeforeAdvice  
After-returning(返回后) org.springframework.aop.AfterReturningAdvice  
After-throwing(抛出后) org.springframework.aop.ThrowsAdvice  
Arround(周围) org.aopaliance.intercept.MethodInterceptor  
Introduction(引入) org.springframework.aop.IntroductionInterceptor  
  
值的说明的是周围通知，他是由AOP Alliance中的接口定义的而非Spring,周围通知相当于前通知、返回后通知、抛出后通知的结合（传说中的完全体？好吧，我看日和看多  
  
了）还有引入通知怎么玩我还没搞清楚，等心无杂念的时候玩玩  
  
这东西怎么玩？这么几个步骤：  
1.创建通知：实现这几个接口，把其中的方法实现了  
2.定义切点和通知者：在Spring配制文件中配置这些信息  
3.使用ProxyFactoryBean来生成代理  
  
具体做法。。。大晚上的就举个睡觉的例子吧:  
  
首先写一个接口叫Sleepable,这是一个牛X的接口,所有具有睡觉能力的东西都可以实现该接口（不光生物，包括关机选项里面的休眠）  
  
package test.spring.aop.bean  
  
public interface Sleepable{  
   
    void sleep();   
}  
  
然后写一个Human类，他实现了这个接口  
  
package test.spring.aop.bean  
  
public Human implements Sleepable{  
     
   /\*这人莫非跟寡人差不多？  
    \*除了睡觉睡的比较好之外其余的什么也不会做？\*/  
   public void sleep(){  
      System.out.println("睡觉了！梦中自有颜如玉！");  
   }  
  
}  
  
  
好了，这是主角，不过睡觉前后要做些辅助工作的，最基本的是脱穿衣服，失眠的人还要吃安眠药什么的,但是这些动作与纯粹的睡觉这一“业务逻辑”是不相干的，如果把  
  
这些代码全部加入到sleep方法中，是不是有违单一职责呢？，这时候我们就需要AOP了。  
  
编写一个SleepHelper类，它里面包含了睡觉的辅助工作,用AOP术语来说它就应该是通知了，我们需要实现上面的接口  
  
package test.spring.aop.bean;  
  
import java.lang.reflect.Method;  
  
import org.springframework.aop.AfterReturningAdvice;  
import org.springframework.aop.MethodBeforeAdvice;  
  
public class SleepHelper implements MethodBeforeAdvice,AfterReturningAdvice{  
  
    public void before(Method mtd, Object[] arg1, Object arg2)  
            throws Throwable {  
        System.out.println("通常情况下睡觉之前要脱衣服！");  
    }  
  
    public void afterReturning(Object arg0, Method arg1, Object[] arg2,  
            Object arg3) throws Throwable {  
        System.out.println("起床后要先穿衣服！");  
    }  
      
}  
  
然后在spring配置文件中进行配置:  
<bean id="sleepHelper" class="test.spring.aop.bean.SleepHelper">  
</bean>  
  
OK!现在创建通知的工作就完成了.  
  
第二步是进行配置，这是很令人蛋疼的操作，尤其是这么热的天，Spring又把东西的名字起的见鬼的长！它为啥不能像usr这种风格呢？  
  
首先要做的是配置一个切点,据说切点的表示方式在Spring中有好几种，但是常用的只有两种：1.使用正则表达式 2.使用AspectJ表达式 AspectJ我不是很熟悉(我也是熟悉  
  
党 or 精通党？),我还是习惯用正则表达式  
  
Spring使用org.springframework.aop.support.JdkRegexpMethodPointcut来定义正则表达式切点  
<bean id="spleepPointcut" class="org.springframework.aop.support.JdkRegexpMethodPointcut">  
  <property name="pattern" value=".\*sleep"/>  
</bean>  
  
pattern属性指定了正则表达式，它匹配所有的sleep方法  
  
切点仅仅是定义了故事发生的地点，还有故事发生的时间以及最重要的故事的内容,就是通知了，我们需要把通知跟切点结合起来，我们要使用的通知者是:  
org.springframework.aop.support.DefaultPointcutAdvisor  
  
<bean id="sleepHelperAdvisor" class="org.springframework.aop.support.DefaultPointcutAdvisor">  
     <property name="advice" ref="sleepHelper"/>  
     <property name="pointcut" ref="sleepPointcut"/>  
</bean>  
  
切入点和通知都配置完成，接下来该调用ProxyFactoryBean产生代理对象了  
  
<bean id="humanProxy" class="org.springframework.aop.framework.ProxyFactoryBean">  
     <property name="target" ref="human"/>  
     <property name="interceptorNames" value="sleepHelperAdvisor" />  
     <property name="proxyInterfaces" value="test.spring.aop.bean.Sleepable" />  
</bean>  
  
ProxyFactoryBean是一个代理，我们可以把它转换为proxyInterfaces中指定的实现该interface的代理对象:  
  
import org.springframework.aop.framework.ProxyFactoryBean;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
import test.spring.aop.bean.Sleepable;  
  
  
public class Test {  
  
    public static void main(String[] args){  
        ApplicationContext appCtx = new ClassPathXmlApplicationContext("applicationContext.xml");  
        Sleepable sleeper = (Sleepable)appCtx.getBean("humanProxy");  
        sleeper.sleep();  
    }  
}  
  
程序运行产生结果：  
通常情况下睡觉之前要脱衣服！  
睡觉啦~梦中自有颜如玉!  
起床后要先穿衣服！  
  
OK!这是我们想要的结果，但是上面这个过程貌似有点复杂，尤其是配置切点跟通知,Spring提供了一种自动代理的功能，能让切点跟通知自动进行匹配，修改配置文件如下:  
 <bean id="sleepHelper" class="test.spring.aop.bean.SleepHelper">  
  </bean>  
  <bean id="sleepAdvisor" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">  
    <property name="advice" ref="sleepHelper"/>  
    <property name="pattern" value=".\*sleep"/>  
  </bean>  
  <bean id="human" class="test.spring.aop.bean.Human">  
  </bean>  
  <bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"/>  
  
执行程序：  
public class Test {  
  
    public static void main(String[] args){  
        ApplicationContext appCtx = new ClassPathXmlApplicationContext("applicationContext.xml");  
        Sleepable sleeper = (Sleepable)appCtx.getBean("human");  
        sleeper.sleep();  
    }  
}  
成功输出结果跟前面一样!  
只要我们声明了org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator(我勒个去的，名太长了)就能为方法匹配的bean自动创建代理！  
  
但是这样还是要有很多工作要做,有更简单的方式吗?有！  
  
一种方式是使用AspectJ提供的注解:  
  
package test.mine.spring.bean;  
  
import org.aspectj.lang.annotation.AfterReturning;  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Before;  
import org.aspectj.lang.annotation.Pointcut;  
@Aspect  
public class SleepHelper {  
  
    public SleepHelper(){  
          
    }  
      
    @Pointcut("execution(\* \*.sleep())")  
    public void sleeppoint(){}  
      
    @Before("sleeppoint()")  
    public void beforeSleep(){  
        System.out.println("睡觉前要脱衣服!");  
    }  
      
    @AfterReturning("sleeppoint()")  
    public void afterSleep(){  
        System.out.println("睡醒了要穿衣服！");  
    }  
      
}  
  
用@Aspect的注解来标识切面,注意不要把它漏了，否则Spring创建代理的时候会找不到它,@Pointcut注解指定了切点，@Before和@AfterReturning指定了运行时的通知，注  
  
意的是要在注解中传入切点的名称  
  
然后我们在Spring配置文件上下点功夫,首先是增加AOP的XML命名空间和声明相关schema  
命名空间:  
xmlns:aop="http://www.springframework.org/schema/aop"  
schema声明:  
http://www.springframework.org/schema/aop  
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd  
  
然后加上这个标签:  
<aop:aspectj-autoproxy/> 有了这个Spring就能够自动扫描被@Aspect标注的切面了  
  
最后是运行，很简单方便了：  
public class Test {  
  
    public static void main(String[] args){  
        ApplicationContext appCtx = new ClassPathXmlApplicationContext("applicationContext.xml");  
        Sleepable human = (Sleepable)appCtx.getBean("human");  
        human.sleep();  
    }  
}  
  
下面我们来看最后一种常用的实现AOP的方式:使用Spring来定义纯粹的POJO切面  
  
前面我们用到了<aop:aspectj-autoproxy/>标签,Spring在aop的命名空间里面还提供了其他的配置元素:  
<aop:advisor> 定义一个AOP通知者  
<aop:after> 后通知  
<aop:after-returning> 返回后通知  
<aop:after-throwing> 抛出后通知  
<aop:around> 周围通知  
<aop:aspect>定义一个切面  
<aop:before>前通知  
<aop:config>顶级配置元素，类似于<beans>这种东西  
<aop:pointcut>定义一个切点  
  
我们用AOP标签来实现睡觉这个过程:  
代码不变，只是修改配置文件,加入AOP配置即可:  
<aop:config>  
    <aop:aspect ref="sleepHelper">  
    <aop:before method="beforeSleep" pointcut="execution(\* \*.sleep(..))"/>  
    <aop:after method="afterSleep" pointcut="execution(\* \*.sleep(..))"/>  
    </aop:aspect>  
</aop:config>  
  
完！  
  
OK~~基本上就这么多了吧，要想用好还得多折腾折腾，另外玩玩AspectJ~不就是个玩！