### JAVA第三阶段—DAY06-JAVA案例

1. AOP入门案例

掌握AOP的基本思想和配置过程，重点讲解pointcut的expression配置

引入依赖

      <dependency>  
          <groupId>org.springframework</groupId>  
          <artifactId>spring-aop</artifactId>  
          <version>5.2.8.RELEASE</version>  
      </dependency>  
​  
      <dependency>  
          <groupId>org.aspectj</groupId>  
          <artifactId>aspectjrt</artifactId>  
          <version>1.9.5</version>  
      </dependency>  
​  
      <dependency>  
          <groupId>org.aspectj</groupId>  
          <artifactId>aspectjweaver</artifactId>  
          <version>1.9.5</version>  
      </dependency>

### 编写通知类

/\*\*  
\* 日志输出通知类  
\*/  
public class LogAdvise {  
​  
  public void beforeLog(){  
      System.out.println("方法开始执行！");  
  }  
​  
  public void afterLog(){  
      System.out.println("方法后置执行！");  
  }  
​  
  public void afterReturning(){  
      System.out.println("方法返回了数据");  
  }  
​  
  public void afterThrowing(){  
      System.out.println("方法抛出了异常");  
  }  
​  
  public void around(ProceedingJoinPoint joinPoint) throws Throwable {  
      System.out.println("around方法名：" + joinPoint.getSignature().getName());  
      System.out.println("around --前置");  
      //原来方法  
      joinPoint.proceed();  
      System.out.println("around --后置");  
  }  
}

### AOP的配置

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xmlns:context="http://www.springframework.org/schema/context"  
      xmlns:aop="http://www.springframework.org/schema/aop"  
      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-aop.xsd">  
​  
  <!--配置包的扫描-->  
  <context:component-scan base-package="com.blb.aop\_demo"></context:component-scan>  
  <!--配置通知类-->  
  <bean id="logAdvise" class="com.blb.aop\_demo.util.LogAdvise"></bean>  
  <!--配置切面-->  
  <aop:config>  
      <!--配置切入点-->  
      <aop:pointcut id="pc" expression="execution(\* com.blb.aop\_demo.service.\*Service.\*(..))"/>  
      <!--配置切面 ref是通知类的bean-->  
      <aop:aspect id="aspect1" ref="logAdvise">  
          <!--前置通知 method是对应的通知方法 pointcut-ref是切入点-->  
          <aop:before method="beforeLog" pointcut-ref="pc"></aop:before>  
          <!--后置-->  
          <aop:after method="afterLog" pointcut-ref="pc"></aop:after>  
          <!--后置返回-->  
          <aop:after-returning method="afterReturning" pointcut-ref="pc"></aop:after-returning>  
          <!--后置抛异常-->  
          <aop:after-throwing method="afterThrowing" pointcut-ref="pc"></aop:after-throwing>  
          <!--环绕-->  
          <aop:around method="around" pointcut-ref="pc"></aop:around>  
      </aop:aspect>  
  </aop:config>  
</beans>

### 测试

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("spring-aop.xml");  
GoodsService goodsService = context.getBean(GoodsService.class);  
goodsService.queryGoods();  
goodsService.queryGoodsById(1);  
goodsService.createGoods();  
goodsService.updateGoods();  
goodsService.deleteGoodsById(1);

1. 注解配置案例

配置类

@ComponentScan(basePackages = "com.blb.aop\_demo")  
@Configuration  
//启动AspectJ的注解配置  
@EnableAspectJAutoProxy  
public class AopConfig {  
}

日志切面

/\*\*  
\* 日志切面  
\*/  
@Aspect  
@Component  
public class LogAspect {  
​  
  //配置切入点  
  @Pointcut("execution(\* com.blb.aop\_demo.service.\*Service.\*(..))")  
  public void pointcut(){  
  }  
​  
  //配置通知方法  
  @Before("pointcut()")  
  public void beforeLog(){  
      System.out.println("这是前置的通知方法！！");  
  }  
}  
​

测试

AnnotationConfigApplicationContext context2 = new     
                            AnnotationConfigApplicationContext(AopConfig.class);  
GoodsService goodsService = context2.getBean(GoodsService.class);  
goodsService.queryGoods();  
goodsService.queryGoodsById(1);  
goodsService.createGoods();  
goodsService.updateGoods();  
goodsService.deleteGoodsById(1);

1. 日志跟踪案例

讲解AOP在实际项目中的一些应用，比如日志、事务、权限、缓存的处理等

/\*\*

\* Log4j日志输出切面

\*/

@Aspect

@Component

public class Log4jAspect {

//创建日志对象

private Logger logger = Logger.getLogger(Log4jAspect.class);

//给所有的service类的所有方法加日志跟踪

@Pointcut("execution(\* com.blb.aop\_demo.service.\*Service.\*(..))")

public void logPointcut(){

}

//配置环绕通知

@Around("logPointcut()")

public Object aroundLog(ProceedingJoinPoint point) throws Throwable {

//记录方法执行前时间

long start = System.currentTimeMillis();

//打印方法名称

if(logger.isDebugEnabled()){

logger.debug("当前执行方法：" + point.getSignature().getName());

}

//打印参数

Object[] args = point.getArgs();

for(Object arg : args){

if(logger.isDebugEnabled()){

logger.debug("参数："+arg);

}

}

//打印返回值

Object result = point.proceed();

if(logger.isDebugEnabled()){

logger.debug("方法返回值：" +result);

}

//打印执行时间

long end = System.currentTimeMillis();

if(logger.isDebugEnabled()){

logger.debug("方法执行时间：" +(end - start));

}

return result;

}

}