SpringAOP

1. **AspectJ：java社区中最完整的AOP框架**

可以使用注解的方式获取xml配置文件的方式

导入jar包



加入aop命名空间

1. **注解方式**

**① 在xml文件中加入配置**

<aop:aspectj-autoproxy></aop:aspectj-autoproxy>

使@Aspect注解产生作用，自动创建代理对象

**② 把横切关注点的代码抽象到类中**

这个类首先要加入到ioc容器中，即加入@Component注解

还要加入注解证明这个bean是一个切面，**@Aspec**，加载上面注解前

如果需要声明注解**优先级**，在此注解前加入**@Order(1)**这个注解，值越小越优先

**③ 在类中声明各种通知**

声明一个方法，在方法前加入注解

@Before("execution

(public int aop.impl.ArithmeticCalculatorImpl.add(int, int))")

括号内是方法的全称，可以用\*代表所有，**..**代表任意数量

@Before("execution

(public int aop.impl.ArithmeticCalculatorImpl.\*(int, int))")

代表ArithmeticCalculatorImpl类下的所有公有int返回值的方法，参数为两个int类型

@Before("execution

(\* aop.impl.ArithmeticCalculatorImpl.\*(..))")

代表ArithmeticCalculatorImpl类下的所有方法，任意参数

* **@Before**：前置通知，在方法执行前执行
* **@After**：后置通知，在方法执行后执行（无论是否发生异常）
* **@AfterRunning**：返回通知，在方法正常结束后执行(没有异常)，可以访问到方法的结果
* **@AfterThrowing**：异常通知，在方法抛出异常之后
* **@Around**：环绕通知，围绕着方法执行，相当于动态代理的全过程，必须有返回值，返回值即代理方法的实际返回值，需要写一个参数ProceedingJoinPoint pjp

如果想要在方法中访问方法的细节（方法名，参数等），需要在方法里加入一个参数JoinPoint joinPoint可以通过这个对象来访问方法细节

当多个切面方法的execution表达式相同时，可以使用重用切面表达式

用**@Pointcut**注解声明一个切面表达式

@Pointcut("execution (public int aop.impl.ArithmeticCalculatorImpl.\*(int, int))")

**public** **void** declareJointPointExpression() {}

调用

@Before("declareJointPointExpression()")

相当于

@Before("execution (public int aop.impl.

ArithmeticCalculatorImpl.\*(int, int))")

如果在类的外部引用这个重用切面表达式时，类名.方法名调用

如果在包外需要写上带包的完整类名

@Before("AOPHelloWorld.sdeclareJointPointExpression()")

案例

@Order(1)

@Aspect

@Component

**public** **class** AOPHelloWorld {

/\*\*

\* 定义一个方法声明切面表达式，一般的，该方法中不需要添加其他代码

\* 使用@Pointcut注解声明一个切入表达式

\* 后面的其他通知就可以直接用方法名引用这个切入表达式

\*/

@Pointcut("execution (public int aop.impl.ArithmeticCalculatorImpl.\*(int, int))")

**public** **void** declareJointPointExpression() {}

/\*\*

\* 前置通知，在方法访问前被执行

\*

\* **@param** joinPoint

\*/

@Before("declareJointPointExpression()")

**public** **void** before(JoinPoint joinPoint) {

// 获取方法签名，获取方法名称

String methodName = joinPoint.getSignature().getName();

// 获取参数

Object[] objects = joinPoint.getArgs();

System.***out***.println("执行" + methodName + "方法，参数为" + Arrays.*asList*(objects));

}

/\*\*

\* 后置通知，方法结束后被执行，无论是否出异常都会被执行

\*

\* **@param** joinPoint

\*/

@After("declareJointPointExpression()")

**public** **void** after(JoinPoint joinPoint) {

System.***out***.println(joinPoint.getSignature().getName() + "方法执行结束");

}

/\*\*

\* 返回通知，方法正常结束后被访问，出现异常则不访问， 可以获取方法的返回值

\* 获取返回值方法：在注解中添加returning属性，在参数添加一个Object类型

\*/

@AfterReturning(value = "declareJointPointExpression()", returning = "result")

**public** **void** afterReturning(JoinPoint joinPoint, Object result) {

System.***out***.println(joinPoint.getSignature().getName() + "执行结果为" + result);

}

/\*\*

\* 如果出异常则通知，可以获取异常对象 只有抛出异常和参数异常类型一致时才会被执行

\*

\* **@param** joinPoint

\* **@param** ex

\*/

@AfterThrowing(value = "declareJointPointExpression()", throwing = "ex")

**public** **void** afterThrowing(JoinPoint joinPoint, Exception ex) {

System.***out***.println(joinPoint.getSignature().getName() + "方法执行出现异常：" + ex);

}

/\*

/\*\*

\* 环绕通知，代表整个动态代理，必须要有返回值，为代理方法的实际返回值 必须有一个参数ProceedingJoinPoint

\* pjp，可以获取方法的所有信息，并在内部执行方法，获取返回值

\*

\* @param pjp

\*

@Around("declareJointPointExpression()")

public Object around(ProceedingJoinPoint pjp) {

Object object = null;

// 前通知

// ...

try {

// 执行原方法

object = pjp.proceed();

// 结果通知

// ...

} catch (Throwable e) {

// 异常通知

// ...

e.printStackTrace();

}

// 后通知

// ...

return object;

}

\*/

}

1. **配置文件方式**

配置文件：

<!-- 配置bean -->

<bean id=*"arithmeticCalculator"* class=*"aop.xml.ArithmeticCalculatorImpl"*></bean>

<bean id=*"AOPHelloWorld"* class=*"aop.xml.AOPHelloWorld"*></bean>

<!-- AOP配置 -->

<aop:config>

<!-- **配置切面重用表达式** -->

<**aop:pointcut** expression=*"execution (\* aop.xml.ArithmeticCalculatorImpl.\*(..))"*

id=*"pointcut"*/>

**<!-- 配置切面及通知 指向切面的bean，设置优先级 -->**

<**aop:aspect** ref=*"AOPHelloWorld"* order=*"1"*>

**<!-- 前置通知，使用重用表达式 -->**

<**aop:before** method=*"before"* pointcut-ref=*"pointcut"*/>

<!-- 后置通知，使用重用表达式 -->

<**aop:after** method=*"after"* pointcut-ref=*"pointcut"*/>

<!-- 返回通知，设置返回对象名称 -->

<**aop:after-returning** method=*"afterReturning"* pointcut-ref=*"pointcut"* returning=*"result"*/>

<!-- 异常通知，设置异常对象名 -->

<**aop:after-throwing** method=*"afterThrowing"* pointcut-ref=*"pointcut"* throwing=*"ex"*/>

</aop:aspect>

</aop:config>

**类中的代码就变得很简单，只需要写上方法和实现，所有代理部分交给配置文件即可**

**public** **class** AOPHelloWorld {

**public** **void** before(JoinPoint joinPoint) {

// 获取方法签名，获取方法名称

String methodName = joinPoint.getSignature().getName();

// 获取参数

Object[] objects = joinPoint.getArgs();

System.***out***.println("执行" + methodName + "方法，参数为" + Arrays.*asList*(objects));

}

**public** **void** after(JoinPoint joinPoint) {

System.***out***.println(joinPoint.getSignature().getName() + "方法执行结束");

}

**public** **void** afterReturning(JoinPoint joinPoint, Object result) {

System.***out***.println(joinPoint.getSignature().getName() + "执行结果为" + result);

}

**public** **void** afterThrowing(JoinPoint joinPoint, Exception ex) {

System.***out***.println(joinPoint.getSignature().getName() + "方法执行出现异常：" + ex);

}

}