配置通知类型

注意：AOP的通知顺序　前置通知-->最终通知-->后置通知 -->异常通知

1、类型•

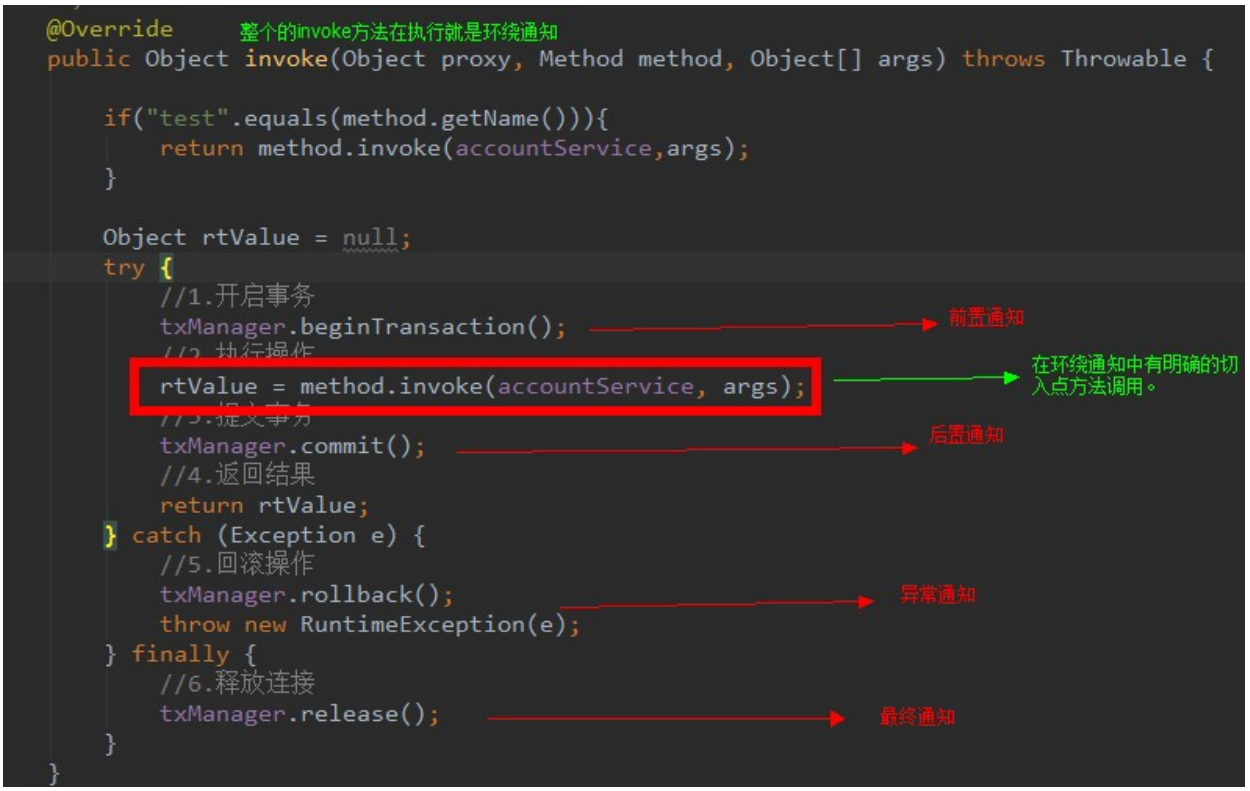
前置通知

后置通知

异常通知

最终通知

环绕通知



2、属性

method：指定通知中方法的名称。

pointct：定义切入点表达式

pointcut-ref：指定切入点表达式的引用

```
<!--配置AOP-->
<aop:config>

    <!-- 配置切入点表达式 ： 指明哪些方法使用这些通知方法
        id属性用于指定表达式的唯一标识。expression属性用于指定表达式内容
```

```
        此标签写在aop:aspect标签内部只能当前切面使用。
        它还可以写在aop:aspect外面，此时就变成了所有切面可用,根据约束一般写在前面
     -->
    <aop:pointcut id="pt1" expression="execution(* com.itheima.service.impl.*.*
(..))"></aop:pointcut>

    <!--配置切面 -->
    <aop:aspect id="logAdvice" ref="logger">
      //  配置前置通知：在切入点方法执行之前执行
        <aop:before method="beforePrintLog" pointcut-ref="pt1" ></aop:before>

      //配置后置通知：在切入点方法正常执行之后值。它和异常通知永远只能执行一个
        <aop:after-returning method="afterReturningPrintLog" pointcut-ref="pt1">
</aop:after-returning>

        //配置异常通知：在切入点方法执行产生异常之后执行。它和后置通知永远只能执行一
个
        <aop:after-throwing method="afterThrowingPrintLog" pointcut-ref="pt1">
</aop:after-throwing>

        // 配置最终通知：无论切入点方法是否正常执行它都会在其后面执行
        <aop:after method="afterPrintLog" pointcut-ref="pt1"></aop:after>

        //配置环绕通知 详细的注释请看Logger类中
        <aop:around method="aroundPringLog" pointcut-ref="pt1"></aop:around>
    </aop:aspect>
</aop:config>



/**
 * 环绕通知
 * 问题：
 *      当我们配置了环绕通知之后，切入点方法没有执行，而通知方法执行了。
 * 分析：
 *      通过对比动态代理中的环绕通知代码，发现动态代理的环绕通知有明确的切入点方法
调用，而我们的代码中没有。
 * 解决：
 *      Spring框架为我们提供了一个接口：ProceedingJoinPoint。该接口有一个方法
proceed()，此方法就相当于明确调用切入点方法。
 *      该接口可以作为环绕通知的方法参数，在程序执行时，spring框架会为我们提供该接
口的实现类供我们使用。
 *
 * spring中的环绕通知：
 *      它是spring框架为我们提供的一种可以在代码中手动控制增强方法何时执行的方式。
 */
public Object aroundPringLog(ProceedingJoinPoint pjp){
```

```
    Object rtValue = null;
    try{
        //得到方法执行所需的参数
        Object[] args = pjp.getArgs();

        System.out.println("Logger类中的aroundPringLog方法开始记录日志了。。。前
置");

        rtValue = pjp.proceed(args);//明确调用业务层方法（切入点方法）

        System.out.println("Logger类中的aroundPringLog方法开始记录日志了。。。后
置");

        return rtValue;
    }catch (Throwable t){
        System.out.println("Logger类中的aroundPringLog方法开始记录日志了。。。异
常");
        throw new RuntimeException(t);
    }finally {
        System.out.println("Logger类中的aroundPringLog方法开始记录日志了。。。最
终");
    }
}
```