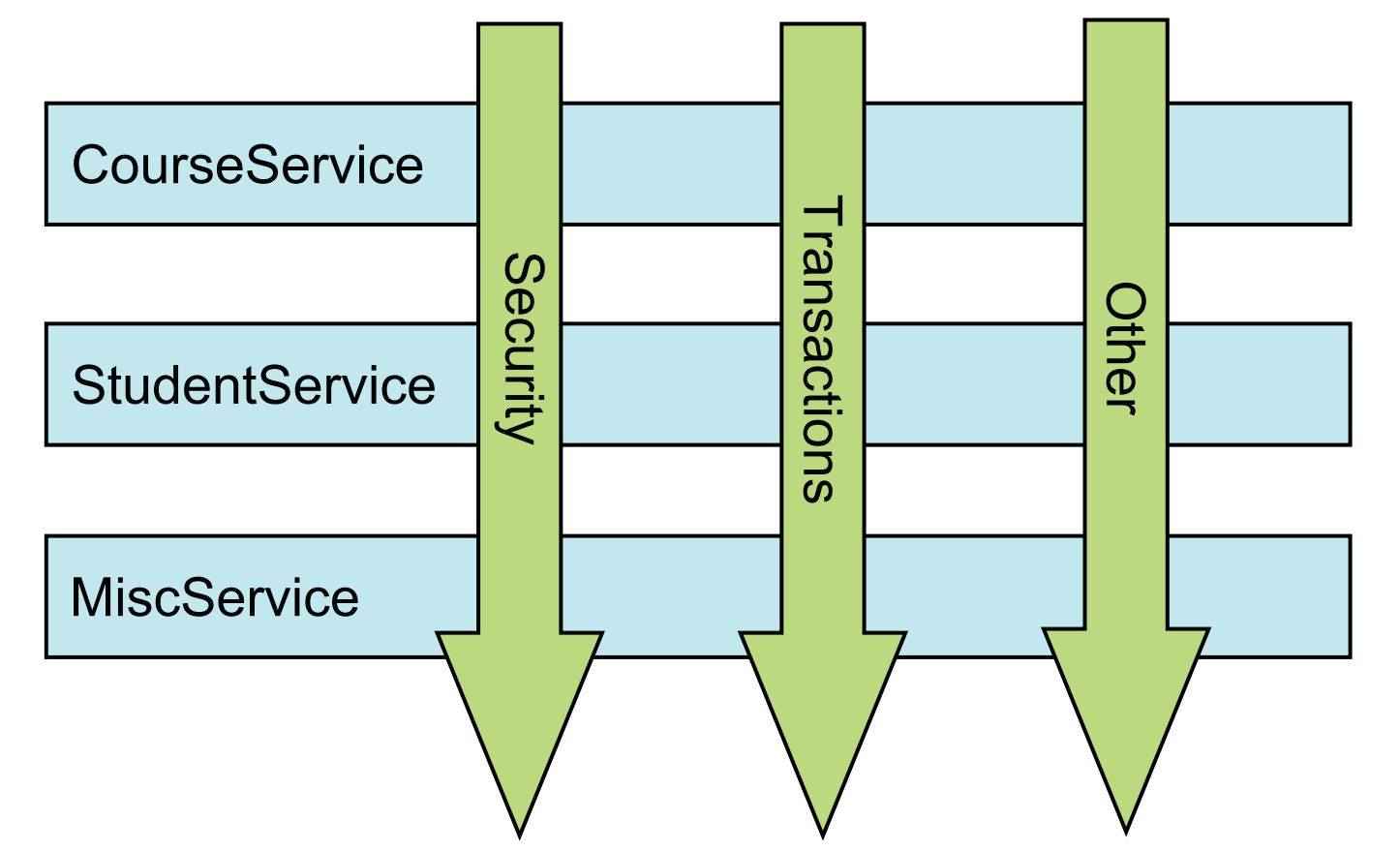
Spring AOP

# 介绍

AOP（Aspect Oriented Programming）即面向切面编程，是一种面向对象编程（Object-Oriented Programming、OOP）的补充，提供了另一种方法构建软件。面向对象编程（OOP）的核心单元是对象（Object），而面向切面编程（AOP）的核心单元是切面（aspect）。切面能够用来模块化那些贯穿多个类型和对象的关注点，比如事务管理、日志管理等。（在AOP术语中这些关注点被称为横切关注点（crosscutting）。）如下图所示：



# AOP概念

需求：统计一个方法的运行时间。步骤为：

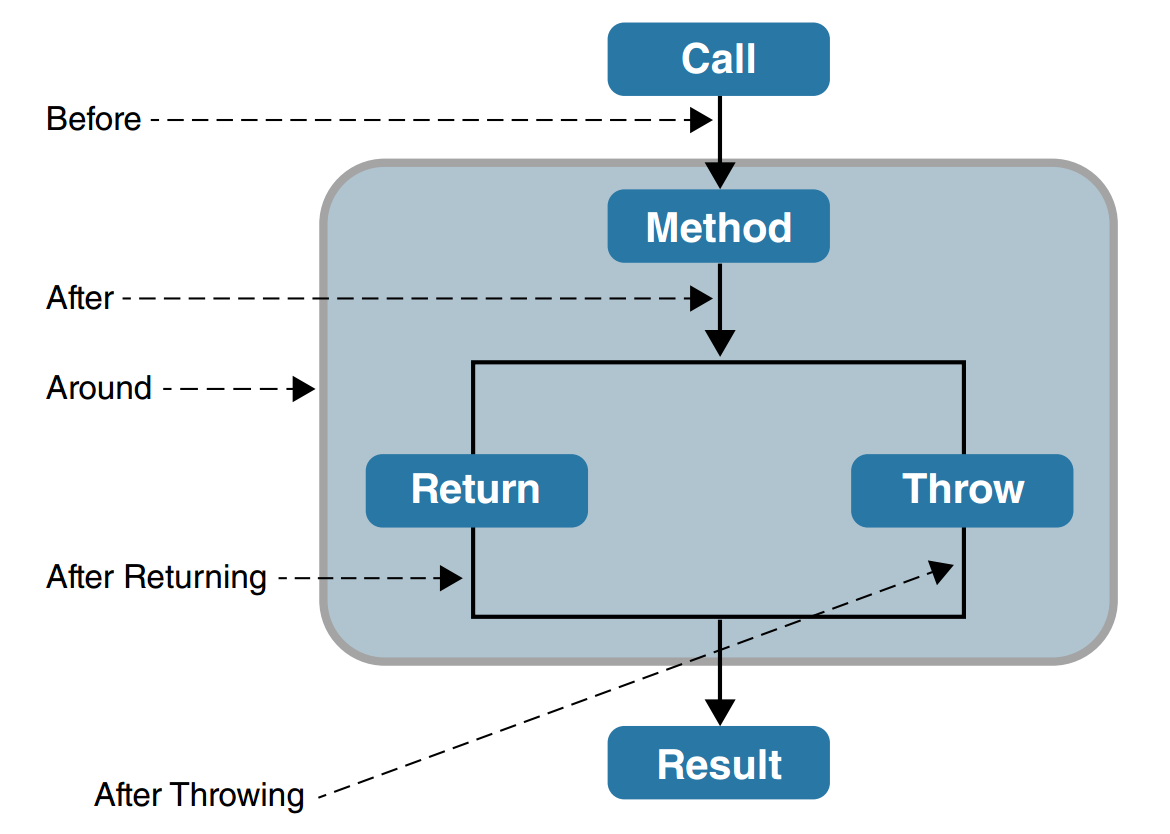
1. 在主方法执行之前记录当前系统时间，存储在变量startTime中。
2. 执行主方法
3. 主方法返回后，记录当前系统时间，存在变量endTime。
4. 使用endTime - startTime则为主方法执行时间elapsedTime。

涉及到的方面：主方法，统计主方法执行的代码。那么，我们怎么使用AOP在不修改方法主体的基础上统计方法的执行时间呢。首先，我们介绍Spring AOP的基本概念。AOP的核心概念有：通知（Advice）、切入点（point  
cuts）、连接点（join points）。当然还有其他的概念，如下：

* 通知（Advice）：定义切面的内容，以及何时应用切面。需求中执行方法执行的时间就是切面的内容，而何时则表示应该何时在主代码流程中插入这些代码。这些时间包括环绕（around），之前（before），之后（after）。我们将会在后面详细讨论通知的类型。
* 连接点（Join Points）：程序执行过程中的一个点。比如方法的调用，异常的抛出都可以作为一个连接点。在我们的需求中，连接点就是方法的执行。在Spring AOP中连接点只能是方法的执行。（想支持更多的连接点请参考AspectJ）
* 切入点（Pointcuts）：**用来匹配一个或多个的连接点**。比如，我们需要统计很多主方法执行（连接点）的时间。那么，如何选取这些连接点（比如一个包下面的方法）的概念就是切入点。如果通**知定义了何时和内容**，那么**切入点则定义了何地**。
* 切面（Aspect）：通知和切入点的合并。也就是说它表示整个对关注点进行模块化的过程，**也就是何时，何地，干什么**。
* 引入（Introduction）：为已存在的类（Class）添加额外的方法和属性。Spring AOP允许你添加额外的接口（interface）与实现到类中。
* 目标（Target）：被Spring AOP代理的对象，**也就是主方法所在的对象**。
* AOP代理（AOP proxy）：Spring为被代理对象创建的代理对象。在Spring AOP中，一个AOP代理会是一个Java动态代理或则CGLIB代理。
* 编织（Weaving）：把切面应用到目标对象生成一个新的代理对象的过程。这个过程可以在三个时期完成：编译时期（Compile time）, 加载时期（load time）, 运行时期（runtime）。Spring AOP支持在运行时期进行编制。

通知（Advice）的类型：

* Before：在主方法调用之前执行
* After：通知在主方法完成之后执行，不管主方法的调用结果如何
* After-returnning：通知在主方法正常返回后执行。比如在不抛出异常时正常返回
* After-throwing：通知在主方法抛出异常后执行
* Around：通知包装了主方法，提供在方法调用一直或之后提供一些功能



# 代码示例

首先，我们定义被代理的接口和实现：

|  |
| --- |
| #1. 被代理接口和实现  **public** **interface** MyBean {  **void** sayHello(String msg);  }  // 使用spring @Component注解，加载到spring上下文中  @Component  **public** **class** MyBeanImpl **implements** MyBean {  @Override  **public** **void** sayHello(String msg) {  System.***out***.println(msg);  }  } |
| #2切面  @Component  **public** **class** AdviceExampleAspect {  // 定义切点  @Pointcut("execution(\* com.imooc..\*Bean.\*(..))")  **public** **void** beanPointCut() {  }  @Before("beanPointCut()")  **public** **void** silenceCellPhone() {  System.***out***.println("手机静音！");  }  @After("beanPointCut()")  **public** **void** applause() {  System.***out***.println("鼓掌！鼓掌！");  }  } |
| #3. Spring配置  <aop:aspectj-autoproxy></aop:aspectj-autoproxy> |
| #4. JUnit测试切面  手机静音！  你好！  鼓掌！鼓掌！ |

# Spring AOP详解

前面，我们已经讨论过，Spring AOP主要由连接点、切入点和通知组成切面。而切入点是用来选择某一范围的连接点的，所以我们首先讨论如何定义切入点。这里要说明一点的就是：Spring AOP只支持一种Join point，就是方法的执行。所以，以下切入点（Pointcut）只能选取方法执行的连接点（Join Point）。

## 切入点（Pointcuts）

Spring AOP使用AspectJ的切入点表达式来声明切入点。以下是Spring AOP支持的AspectJ切入点：

| **AspectJ表达式** | **描述** |
| --- | --- |
| **execution** | 匹配方法执行的连接点，这是使用Spring AOP时主要使用的切入点 |
| **within** | 匹配特定**类型**中的连接点（在Spring AOP中则限制为**匹配类型**中的方法执行） |
| **this** | 匹配Spring AOP代理对象中的连接点（在Spring AOP中为方法的执行），注意匹配的是spring aop代理对象为指定的类型。this表达式必须使用完整的限定类名，不能使用通配符。 |
| **target** | 匹配目标对象中的连接点（在Spring AOP中为方法的执行），但是目标对象得为特定的类型。target表达式必须使用完整的限定类名，不能使用通配符。 |
| **args** | 匹配参数为特定类型实例的连接点（Spring AOP中为方法的执行） |
| **@target** | 匹配特定的连接点（Spring AOP中为方法执行），执行方法的类拥有指定类型的注解 |
| **@args** | 匹配特定的连接点（Spring AOP中为方法的执行），运行时传入的参数必须拥有特定  类型的注解 |
| **@within** | 用于匹配拥有特定注解的类型中的连接点 |
| **@annotation** | 用于匹配拥有特定注解的连接点（Spring AOP中为方法的执行） |
| **bean** | Spring AOP扩展的切入点，可以匹配特定的类名中的连接点（方法的执行） |

### 通配符（Wildcards）

在使用切入点表达式的时候，有些时候我们可以使用通配符：\*、..、+。

|  |  |
| --- | --- |
| **符号** | **含义** |
| .. | 在类型匹配时，匹配任何以.，以.结尾的包名。在方法定义时匹配任意数量的参数。 |
| + | 匹配给定类型的任意子类型。 |
| \* | 匹配数量的任意字符，除了\*字符。 |

### 类型（Type）指示符

通过类型来过滤方法，比如接口、类名或者是包名。Spring提供within切入点，使用方式如下。type name可以被替换为package name或者class name。**within(<type name>)** 。以下是一些例子：

* within(com.xyz.web..\*)：匹配com.xyz.web包下面的所有类中方法的执行，而且因为使用了..通配符，所以可以匹配com.xyz.web的所有子包。\*通配符匹配所有的类名，所以可以匹配所有类中方法的执行。
* with(com.xyz.web.\*)：匹配com.xyz.web包下面所有类中方法的执行。因为没有使用..通配符，所有只是匹配到web包，不包括子包。\*一样匹配所有的类名。
* with(com.xyz.service.AccountService)：匹配AccountService类下面所有方法的执行。
* with(com.xyz.interface.MyServiceInterface+)：匹配所有实现了MyServiceInterface`接口的类中的所有方法的执行。
* with(com.xyz.service.MyBaseService+)：匹配MyBaseService`类和它的子类。

### 方法（Method）指示符

|  |
| --- |
| 匹配特定方法的执行，可以使用execution关键字。execution表达式的格式如下：  execution(modifiers-pattern? ret-type-pattern declaring-type-pattern?name-pattern(param-pattern) throws-pattern?)  必选项：返回值类型（ret-type-pattern）、方法名（name-pattern）和参数（param-pattern）  修饰符（modifiers-pattern）：可以是public、protected或private，也可以使用\*匹配所有的修饰符。  返回值类型（ret-type-pattern）: 返回匹配特定的返回类型。大多数情况都是使用\*通配符匹配所有的返回类型。  方法名（name-pattern）：匹配执行方法的名称，可以使用\*通配符匹配任意数量够的字符。如果要匹配特定类中方法的执行，就必须指定类名（declaring-type-pa ttern）部分(这部分使用的格式参考4.1.2 类)  参数列表（param-pattern）：指定方法的参数必须满足的格式。()匹配没有参数的方法，(..)匹配任意数量的参数。当然你也可以使用\*匹配任意一个参数的类型，比如(\\*, String)匹配第二个参数为String类型，第一个参数为任意类型的情况。  异常列表（throws-pattern）：匹配全限定类名异常类型，如果有多个异常，使用,分割，比如throws java.lang.IllegalArgumentException, java.lang.ArrayIndex OutOfBoundsException。 |

示例如下：

* execution(public \* \*(..))：匹配所有的public方法的执行。
* execution(\* set\*(..))：匹配所有方法名以set开头的方法的执行。
* execution(\* com.xyz.service.AccountService.\*(..))：匹配AccountService接口下所有方法的执行。
* execution(\* com.xyz.service.\*.\*(..))：匹配包com.xyz.service下所有类（或接口）下的所有方法的执行。
* execution(\* com.xyz.service..\*.\*(..))：匹配包com.xyz.service和其子包中的类（或接口）下的所有方法的执行。
* execution(\* \*(.., String)：匹配所有最后一个参数为String的方法的执行。
* execution(\* ..Sample+.sampleGenericCollectionMethod(\*))：匹配任意以.Sample结尾的包，以及其子包中的sampleGenericCollection  
  Method方法的执行，且具有唯一的任意类型的参数。
* execution(\* \*(\*, String, ..)：匹配第一个参数为任意类型，第二个参数为String，后面可拥有任意个参数的方法的执行。

### 其他的切入点指示符

|  |
| --- |
| bean(\*Service)：所有bean名称以Service结尾的bean。  @annotation(org.springframework.transaction.annotation.Transactional)：匹配所有连接点（Spring AOP中方法的执行）拥有@Transaction注解在方法上。  this(com.xyz.service.AccountService)：匹配实现了AccountService接口的代理类中的所有连接点（Spring AOP中方法的执行）  target(com.xyz.service.AccountService)：匹配实现了AccountService接口的目标类的所有连接点（Spring AOP中方法的执行） |

### 组合多个切入点

很多时候可能一个切入点并不能满足我们的需求，这时候就需要组合使用切入点来限制匹配的切入点。在Spring AOP中可以使用and(&&)、or(||)和not(!)。既可以使用文字的形式也可以使用符号的形式。比如 ：

execution(\* concert.Performance.perform(..)) && within(concert.\*))

## 定义切面（@Aspect）

在Spring中可以使用XML和注解的方式来定义切面，本文将只讨论使用注解定义切面的方式。Spring支持使用AspectJ的注解@Aspect来定义切面。但是，如果我们要让Spring知道我们定义了一个\*\*切面\*\*的话，还必须把这个切面声明为一个\*\*Bean\*\*。所以我们使用@Component`注解，这样Spring就会识别它，并把它当做切面来看待了。

|  |
| --- |
| @Component  @Aspect  public class AdviceExampleAspect {} |

## 定义切点（@Pointcut）

切点（Pointcut）定义使用@Pointcut注解。注解中我们就可以使用4.1 切点中的表达式来选择连接点。定义切点时，可以组合多个切点。

|  |
| --- |
| @Pointcut("execution(public \* \*(..))")  private void anyPublicOperation() {}  @Pointcut("within(com.xyz.someapp.trading..\*)")  private void inTrading() {}  // 组合切点  @Pointcut("anyPublicOperation() && inTrading()")  private void tradingOperation() {} |

## 定义通知

我们已经定义了@Before和@After通知，也可以使用其他类型的通知。这里主要介绍如何使用@Around注解，定义环绕（Around）通知类型。

|  |
| --- |
| **public** **class** AdviceExampleAspect {  // 定义切点  @Pointcut("execution(\* com.imooc.beans..\*.\*(..))")  **public** **void** beanPointCut() {  }  @Around("beanPointCut()")  **public** **void** watchBean(ProceedingJoinPoint jp) {  **try** {  System.***out***.println("手机静音！");  jp.proceed();  System.***out***.println("鼓掌！鼓掌！");  } **catch** (Throwable e) {  System.***out***.println("投诉！投诉！");  }  }  }  说明：在Around通知类型中，我们可以定义具体方法运行前或者后的逻辑。 |

## 使用参数

# AspectJ问题列表

## call vs execution

call和execution都是匹配方法的执行。区别在于call表示方法被调用，execution表示方法真正执行。具体体现在两点：

第一，当和within或withincode一起使用时会有不同。call(void m()) && withincode(void m())会捕获任何递归调用。而execution(void m()) && withincode(void m())不会，因为方法中递归调用自己，也还是在方法的执行中。所以只捕获一次，就像execution(void m())一样。

第二，call连接点不会捕获super calls to non-static methods，关于这点，我也不是很明白。

## this() vs target()

this(Atype)匹配所有this instanceof Atype的情况。

target(Atype)匹配所有anObject instanceof Atype的情况。如果你在一个对象中调用一个方法，而且这个对象符合instanceof Atype，那么这就是一个合法的连接点。

execution( void Dog.bark() ) && this( thisObject ) && target( targetObject )：thisObject会是一个Dog实例，targetObject也会是一个Dog实例。因为，execution表示方法真正执行的时候，所以这两个对象都是一样的。

call( void Dog.bark() ) && this( thisObject ) && target( targetObject )：targetObject表示目标对象，所以还是Dog实例。但是，thisObject将会是任何一个调用dog.bark()方法的对象，所以可能是任何类型。因为，call表示方法在调用的时候。