链滴

Spring AOP 切面执行顺序

作者：boolean-dev
原文链接：https://ld246.com/article/1601193800660
来源网站：链滴

Spring AOP 切面执行顺序

1. 概述

1.1 术语

Spring AOP 的相关术语：

- **Aspect：切面**，由一系列切点、增强和引入组成的模块对象，可定义优先级，从而影响增强和引的执行顺序。事务管理（Transaction management）在java企业应用中就是一个很好的切面样例。
- **Join point：接入点**，程序执行期的一个点，例如方法执行、类初始化、异常处理。 在Spring AO中，接入点始终表示方法执行。
- **Advice：增强**，切面在特定接入点的执行动作，包括 "around," "before" and "after"等多种类型包含Spring在内的许多AOP框架，通常会使用拦截器来实现增强，围绕着接入点维护着一个拦截器链。
- **Pointcut：切点**，用来匹配特定接入点的谓词（表达式），增强将会与切点表达式产生关联，并运在任何切点匹配到的接入点上。通过切点表达式匹配接入点是AOP的核心，Spring默认使用AspectJ切点表达式。
- **Introduction：引入**，为某个type声明额外的方法和字段。Spring AOP允许你引入任何接口以及的默认实现到被增强对象上。
- **Target object：目标对象**，被一个或多个切面增强的对象。也叫作被增强对象。既然Spring AOP用运行时代理（runtime proxies），那么目标对象就总是代理对象。
- **AOP proxy：AOP代理**，为了实现切面功能一个对象会被AOP框架创建出来。在Spring框架中AO代理的默认方式是：有接口，就使用基于接口的JDK动态代理，否则使用基于类的CGLIB动态代理。是我们可以通过设置proxy-target-class="true"，完全使用CGLIB动态代理。
- **Weaving：织入**，将一个或多个切面与类或对象链接在一起创建一个被增强对象。织入能发生在译时 （compile time ）(使用AspectJ编译器)，加载时（load time），或运行时（runtime） 。Spri g AOP默认就是运行时织入，可以通过枚举AdviceMode来设置。

1.2 简述

本次 Spring AOP 执行顺序主要是针对同一切入点的不同切面执行顺序。
Spring AOP 为定义面的执行顺序提供了两种方案：

- 实现 Ordered 接口
- 使用 @Order 接口

2. 示例

2.1 实现 Ordered 接口

注解类

First

```
@Documented
@Target({ElementType.TYPE,ElementType.METHOD})
@Retention(RUNTIME)
public @interface First {
}
```

Sencod

```
@Documented
@Target({ElementType.TYPE,ElementType.METHOD})
@Retention(RUNTIME)
public @interface Second {
}
```


切面

FirstAspect

```
@Slf4j
@Component
@Aspect
public class FirstAspect implements Ordered {

    @Pointcut("@annotation(com.booleandev.data.aop.First) || @within(com.booleandev.data.
op.First)")
    public void pointcut() {
    }

    @Around("pointcut()")
    public Object around(ProceedingJoinPoint pjp) {
        log.info("--------------> ann,first注解执行");

        try {
            return pjp.proceed();
        } catch (Throwable throwable) {
            throwable.printStackTrace();
            return null;
        }
    }

    @Override
    public int getOrder() {
        return 1;
```

```
    }
}
```

Second

```
@Slf4j
@Component
@Aspect
public class SecondAspect implements Ordered{

    @Pointcut("@annotation(com.booleandev.data.aop.Second) || @within(com.booleandev.da
a.aop.Second)")
    public void pointcut() {
    }

    @Around("pointcut()")
    public Object around(ProceedingJoinPoint pjp) {
        log.info("--------------> ann,Second执行");

        try {
            return pjp.proceed();
        } catch (Throwable throwable) {
            throwable.printStackTrace();
            return null;
        }
    }

    @Override
    public int getOrder() {
        return 2;
    }
}
```


切入点

```
@Slf4j
@Service
public class UserService{

    @Autowired
    private UserRepository userRepository;

    @First
    @Second
    public List<User> findAll() {
        log.info(entityManager.toString());
        return userRepository.findAll();
    }
}
```


结果

```
--------------> ann,first注解执行
--------------> ann,Second执行
```


2.2 使用 @Order 接口

注解类

First

```
@Documented
@Target({ElementType.TYPE,ElementType.METHOD})
@Retention(RUNTIME)
public @interface First {
}
```

Sencod

```
@Documented
@Target({ElementType.TYPE,ElementType.METHOD})
@Retention(RUNTIME)
public @interface Second {
}
```


切面

FirstAspect

```
@Slf4j
@Component
@Aspect
@Order(1)
public class FirstAspect {

    @Pointcut("@annotation(com.booleandev.data.aop.First) || @within(com.booleandev.data.
op.First)")
    public void pointcut() {
    }

    @Around("pointcut()")
    public Object around(ProceedingJoinPoint pjp) {
        log.info("--------------> ann,first注解执行");

        try {
            return pjp.proceed();
```

```
        } catch (Throwable throwable) {
            throwable.printStackTrace();
            return null;
        }
    }

}
```

SecondAspect

```
@Slf4j
@Component
@Aspect
@Order(2)
public class SecondAspect{

    @Pointcut("@annotation(com.booleandev.data.aop.Second) || @within(com.booleandev.da
a.aop.Second)")
    public void pointcut() {
    }

    @Around("pointcut()")
    public Object around(ProceedingJoinPoint pjp) {
        log.info("--------------> ann,Second执行");

        try {
            return pjp.proceed();
        } catch (Throwable throwable) {
            throwable.printStackTrace();
            return null;
        }
    }

}
```


切入点

```
@Slf4j
@Service
public class UserService{

    @Autowired
    private UserRepository userRepository;

    @First
    @Second
    public List<User> findAll() {
        log.info(entityManager.toString());
        return userRepository.findAll();
    }
}
```


结果

```
----------> ann,Second执行
----------> ann,first注解执行
```


3. 结论

- 切面执行顺序有两种方式
  - 实现 Ordered 接口
  - 使用 @Order注解
- 排序为顺序，数字越小，越先被执行
- 如果同时使用了注解和实现接口，则以接口的 order 为主

参考文档：
https://juejin.im/post/6844903969433583624
https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#aop-ataspectj-advice-ordering