Spring，day03，主讲：汤小洋

一、AOP

1. 简介

1.1 概念

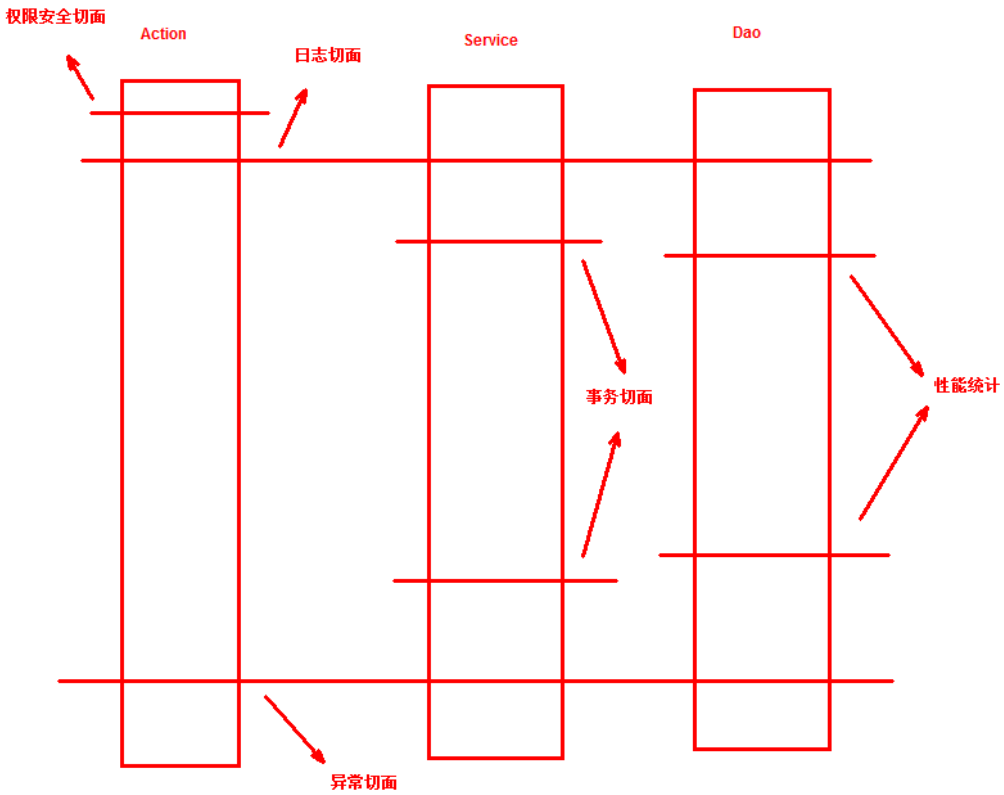
AOP：Aspect Oriented Programming面向切面编程，是OOP面向对象编程的一种补充

将程序中交叉业务逻辑（事务、日志）代码提取出来，封装成切面，由AOP容器在适当时机(位置)将封装的切面动态的织入到具体业务逻辑中

AOP不是Spring特有的

1.2 应用场合

适用于具有横切逻辑的场合，如事务管理、日志记录、性能监测、异常通知、访问控制等



1.3 作用

- 不改变原有代码的基础上动态添加新的功能
- 模块化

1.4 术语

- 连接点 Joinpoint

  程序执行的某个特定位置，如方法调用前、方法调用后、方法抛出异常时、方法调用前后等

- 切入点 Pointcut

  定位查找到需要的连接点，即切点

- 增强 Advice 也称为通知

  在切点上执行的一段程序代码，用来实现某些功能

- 目标对象 Target

  将执行增强处理的目标类

- 织入 Weaving

将增强添加到目标类具体切入点上的过程

- 代理 Proxy

  一个类被织入增强后，会产生一个代理类

- 切面 Aspect

  切点和增强的组合

- 引介 Introduction 也称为引入

2. 实现原理

2.1 回顾：代理模式

概念：为其他对象提供一种代理，以控制对这个对象的访问，起到中介的作用

通过代理对象访问目标对象，可以增强额外的操作，扩展目标对象的功能

分类：

- 静态代理

  代理类是程序员创建或工具生成

  所谓静态就是程序运行之前就已经存在代理类的字节码文件

  缺点：代理对象需要和目标对象实现相同的接口，如果接口增加方法，目标对象和代理对象都要维护

- 动态代理

  代理类是程序在运行期间由JVM根据反射等机制动态生成的，自动生成代理类和代理对象

  所谓动态就是指在程序运行前不存在代理类的字节码文件

动态代理的两种技术：

- jdk技术

```
Proxy.newProxyInstance(
    classLoader, //目标类的类加载器
    interfaces, //目标类的接口列表
    InvocationHandler //交叉业务逻辑
);
```

  特点：目标对象必须实现一个或多个接口，如果没有实现任何接口，则无法使用jdk的动态代理，此时可以使用cglib

- cglib技术，适用于无接口时使用

2.2 cglib技术

如果没有实现接口，则通过继承来实现的

步骤：

1. 添加jar包

```
<dependency>
  <groupId>cglib</groupId>
  <artifactId>cglib</artifactId>
  <version>${cglib.version}</version>
</dependency>
```

2. 用法

```
Enhancer.create(
    class, //目标类的类型
    InvocationHandler, //交叉业务逻辑
);
```

2.3 AOP原理

Spring AOP原理就是使用动态代理

- 对于实现接口的目标类，使用的是jdk动态代理
- 对于没有实现任何接口的目标类，使用的是cglib的动态代理

3.Spring AOP的配置方式

3.1 三种配置方式

- Spring AOP 1.x，使用ProxyFactoryBean手动代理
- Spring AOP 2.x，基本命名空间的配置
- Annotation，基于注解的配置（推荐）

3.2 Advice类型

Spring AOP支持5种类型的通知（增强）

通知类型	实现接口	描述
前置通知(增强)	MethodBeforeAdvice	在方法执行前添加功能

通知类型	实现接口	描述
后置通知	AfterReturningAdvice	在方法执行后添加功能
环绕通知	MethodInterceptor	在方法执行前后添加功能
异常通知	ThrowsAdvice	在方法抛出异常后添加功能
引入通知(了解)	IntroductionInterceptor	在目标类中添加新方法和属性

注：多个Advice之间不允许有耦合，即多个Advice之间不允许有业务交叉

二、Spring AOP 1.x

1. 基本用法

1.1 添加jar包

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aspects</artifactId>
</dependency>
```

1.2 配置Advice

定义增强类，实现相应的接口

```
<!-- 配置Advice -->
<bean class="aop04.advice.LogAdvice"/>
```

1.3 配置Pointcut

定义切入点，配置位置信息，指定哪些类的哪些方法需要被执行AOP

使用NameMatchMethodPointcutAdvisor根据方法名匹配切入点

Advisor是Pointcut和Advice的配置器，Pointcut+Advice=Advisor

```
 <!-- 配置Advisor，将Advice注入到Pointcut位置，织入的过程 -->
<bean
class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">
  <!-- 指定Advice -->
  <property name="advice" ref="logAdvice"/>
  <!--  配置Pointcut，指定匹配哪些方法 -->
  <property name="mappedNames">
    <list>
      <value>login</value>
      <value>logout</value>
    </list>
  </property>
</bean>
```

1.4 配置代理

使用ProxyFactoryBean配置代理

```
<bean id="userService"
class="org.springframework.aop.framework.ProxyFactoryBean">
  <property name="target" ref="userServiceTarget"/>  <!-- 目标类的实例 -->
  <property name="interfaces"> <!-- 目标类的接口列表 -->
    <list>
      <value>aop04.service.UserService</value>
    </list>
  </property>
  <property name="interceptorNames"> <!-- 交叉业务逻辑 -->
    <list>
      <value>logAdvisor</value>
    </list>
  </property>
</bean>
```

2. 练习

对结果进行缓存的计算器

- 如果已经执行过，则直接从缓存中查找结果
- 如果未执行过，则调用业务逻辑

  运行结果如下：

```
CalcServiceImpl.add
3
3
CalcServiceImpl.add
3
CalcServiceImpl.minus
-1.0
-1.0
3
```

三、Spring AOP 2.x

1. 简介

基于命名空间的配置，原理是使用后处理器，更简单

特点：

- 简化配置
- 非侵入性：编写通知时不需要实现任何接口
- 使用AspectJ表达式定义切点

2. 基本用法

2.1 配置Advice

定义增强类，不需要实现任何接口，但有多种写法：

写法	说明
public void 方法名(JoinPoint)	前置通知
public void 方法名(JoinPoint,Object)	后置通知
public void 方法名(JoinPoint,Exception)	异常通知
public Object 方法名(ProceedingJoinPoint)	环绕通知

2.2 配置Pointcut并织入

四、AspectJ表达式

1. 简介

切点表达式，一种表达式，用来定义切点位置

2. 用法

2.1 within

语法：within(包名.类名)

匹配该类中的所有方法

2.2 execution

匹配特定包中的特定类中特定返回值类型的特定参数的特定方法

语法：execution(表达式)

表达式：返回值类型 包名.类名.方法名(参数类型)

通配符：* 和 ..