1 Spring IoC 和 AOP

Spring 框架 Java 开发的行业标准

Spring 全家桶

Web：Spring Web MVC/Spring MVC、Spring Web Flux

持久层：Spring Data/ Spring Data JPA、Spring Data Redis、Spring Data MongoDB

安全校验：Spring Security

构建工程脚手架：Spring Boot

微服务：Spring Cloud

IoC 是 Spring 全家桶各个功能模块的基础，创建对象的容器。

AOP 也是以 IoC 为基础，AOP 是面向切面编程，抽象化的面向对象

1、打印日志

2、事务

3、权限处理

1.1 IoC

控制反转，将对象的创建进行反转，常规情况下，对象都是开发者手动创建的，使用 IoC 开发者不再需要创建对象，而是由 IoC 容器根据需求自动创建项目所需要的对象。

不用 IoC：所有对象开发者自己创建

使用 IoC：对象不用开发者创建，而是交给 Spring 框架来完成

1、pom.xml

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.15</version>
</dependency>
```

基于 XML 和基于注解

基于 XML：开发者把需要的对象在 XML 中进行配置，Spring 框架读取这个配置文件，根据配置文件的内容来创建对象

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"
```

```
xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:p="http://www.springframework.org/schema/p"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

    <bean class="com.southwind.ioc.DataConfig" id="config">
        <property name="driverName" value="Driver"></property>
        <property name="url" value="localhost:8080"></property>
        <property name="username" value="root"></property>
        <property name="password" value="root"></property>
```

```
    </bean>

</beans>
```

```
package com.southwind.ioc;

import
org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXm
lApplicationContext;

public class Test {
    public static void main(String[] args) {
//        DataConfig dataConfig = new
DataConfig();
//        dataConfig.setDriverName("Driver");
//
dataConfig.setUrl("localhost:3306/dbname");
//        dataConfig.setUsername("root");
//        dataConfig.setPassword("root");
        ApplicationContext context = new
ClassPathXmlApplicationContext("spring.xml");

 System.out.println(context.getBean("config"));
    }
}
```

基于注解

1、配置类

用一个 Java 类来替代 XML 文件，把在 XML 中配置的内容放到配置类中。

```
package com.southwind.configuration;

import com.southwind.ioc.DataConfig;
import
org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configur
ation;

@Configuration
public class BeanConfiguration {

    @Bean(value = "config")
    public DataConfig dataConfig(){
        DataConfig dataConfig = new
DataConfig();
        dataConfig.setDriverName("Driver");

 dataConfig.setUrl("localhost:3306/dbname");
        dataConfig.setUsername("root");
        dataConfig.setPassword("root");
        return dataConfig;
    }
}
```

```
ApplicationContext context = new
AnnotationConfigApplicationContext(BeanConfigur
ation.class);
System.out.println(context.getBean("config"));
```

2、扫包+注解

更简单的方式，不再需要依赖于 XML 或者配置类，而是直接将 bean 的创建交给目标类，在目标类添加注解来创建

```
package com.southwind.ioc;

import lombok.Data;
import
org.springframework.beans.factory.annotation.Va
lue;
import
org.springframework.stereotype.Component;

@Data
@Component
public class DataConfig {
    @Value("localhost:3306")
    private String url;
    @Value("Driver")
    private String driverName;
    @Value("root")
    private String username;
    @Value("root")
    private String password;
}
```

```
ApplicationContext context = new
AnnotationConfigApplicationContext("com.southwi
nd.ioc");
System.out.println(context.getBean(DataConfig.c
lass));
```

自动创建对象，完成依赖注入

```
package com.southwind.ioc;

import lombok.Data;
import
org.springframework.beans.factory.annotation.Au
towired;
import
org.springframework.beans.factory.annotation.Va
lue;
import
org.springframework.stereotype.Component;

@Data
@Component
public class GlobalConfig {
    @Value("8080")
    private String port;
    @Value("/")
    private String path;
    @Autowired
    private DataConfig dataConfig;
}
```

@Autowired 通过类型进行注入，如果需要通过名称取值，通过 @Qualifier 注解完成名称的映射

```
package com.southwind.ioc;

import lombok.Data;
import
org.springframework.beans.factory.annotation.Au
towired;
```

```
import
org.springframework.beans.factory.annotation.Qu
alifier;
import
org.springframework.beans.factory.annotation.Va
lue;
import
org.springframework.stereotype.Component;

@Data
@Component
public class GlobalConfig {
    @Value("8080")
    private String port;
    @Value("/")
    private String path;
    @Autowired
    @Qualifier("config")
    private DataConfig config;
}
```

```
package com.southwind.ioc;

import lombok.Data;
import
org.springframework.beans.factory.annotation.Va
lue;
import
org.springframework.stereotype.Component;

@Data
@Component("config")
public class DataConfig {
```

```
    @Value("localhost:3306")
    private String url;
    @Value("Driver")
    private String driverName;
    @Value("root")
    private String username;
    @Value("root")
    private String password;
}
```

1.2 AOP

面向切面编程，是一种抽象化的面向对象编程，对面向对象编程的一种补充，底层使用动态代理机制来实现

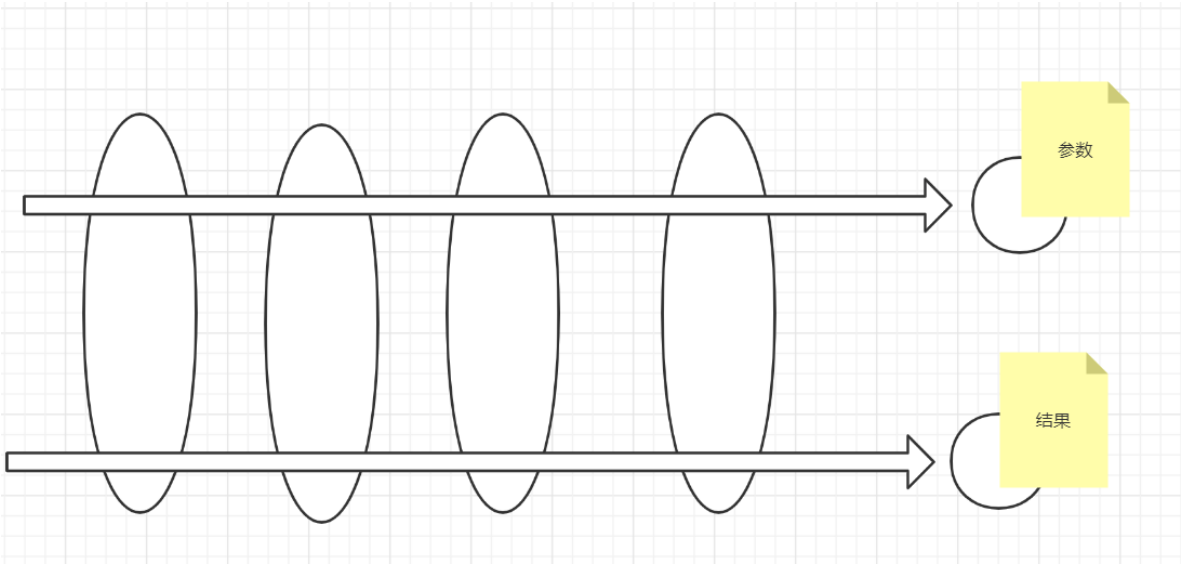
1、打印日志

2、事务

3、权限处理

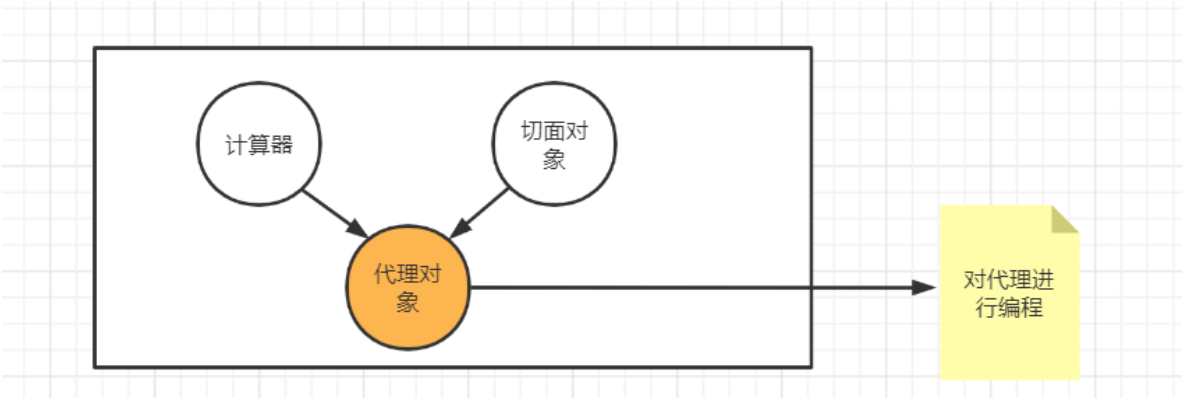
打印日志

业务代码和打印日志耦合起来



计算器方法中，日志和业务混合在一起，AOP 要做的就是将日志代码全部抽象出去统一进行处理，计算器方法中只保留核心的业务代码。

做到核心业务和非业务代码的解耦合



1、创建切面类

```
package com.southwind.aop;

import org.aspectj.lang.JoinPoint;
import
org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import
org.springframework.stereotype.Component;

import java.util.Arrays;

@Component
@Aspect
public class LoggerAspect {

    @Before("execution(public int
com.southwind.aop.CalImpl.*(..))")
    public void before(JoinPoint joinPoint){
```

```
        String name =
joinPoint.getSignature().getName();
        System.out.println(name+"方法的参数是"+
Arrays.toString(joinPoint.getArgs()));
    }

    @AfterReturning(value = "execution(public
int com.southwind.aop.CalImpl.*(..))",returning
= "result")
    public void afterReturning(JoinPoint
joinPoint,Object result){
        String name =
joinPoint.getSignature().getName();
        System.out.println(name+"方法的结果
是"+result);
    }
}
```

2、实现类添加 @Component 注解

```
package com.southwind.aop;

import
org.springframework.stereotype.Component;

@Component
public class CalImpl implements Cal {
    @Override
    public int add(int num1, int num2) {
        int result = num1 + num2;
        return result;
    }
```

```
    @Override
    public int sub(int num1, int num2) {
        int result = num1 - num2;
        return result;
    }

    @Override
    public int mul(int num1, int num2) {
        int result = num1 * num2;
        return result;
    }

    @Override
    public int div(int num1, int num2) {
        int result = num1 / num2;
        return result;
    }
}
```

3、配置自动扫包，开启自动生成代理对象

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"
```

```
xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:p="http://www.springframework.org/schema/p"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

    <!-- 自动扫包 -->
    <context:component-scan base-package="com.southwind.aop">
</context:component-scan>

    <!-- 开启自动生成代理 -->
    <aop:aspectj-autoproxy></aop:aspectj-autoproxy>

</beans>
```

4、使用

```
package com.southwind.aop;

import
org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXm
lApplicationContext;

public class Test {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("spring.xml");
        Cal bean = context.getBean(Cal.class);
        System.out.println(bean.add(9, 8));
        System.out.println(bean.sub(9, 8));
        System.out.println(bean.mul(9, 8));
        System.out.println(bean.div(9, 8));
    }
}
```