1 Spring

1.1 IoC

解决的是项目中所需要组件的创建问题

不再由开发者来创建项目中所需组件，而是由 Spring IoC 统一创建，开发者只需要获取相应组件使用即可

依赖注入

```java
package com.southwind.entity;

import lombok.Data;

@Data
public class Order {
    private Integer id;
    private String name;
}
```

```java
package com.southwind.entity;

import lombok.Data;

@Data
public class User {
    private Integer id;
    private String name;
    private Integer age;
    private Order order;
}
```

```xml
<bean class="com.southwind.entity.User"
id="user">
    <property name="id" value="1"></property>
    <property name="name" value="张三">
</property>
    <property name="age" value="22"></property>
    <property name="order" ref="order">
</property>
</bean>

<bean id="order"
class="com.southwind.entity.Order">
    <property name="id" value="1"></property>
    <property name="name" value="订单1">
</property>
</bean>
```

如果通过 Class 获取 bean，必须保证 IoC 容器中 bean 是单列模式，否则会报错。

基于 XML 方式，将 bean 信息配置在 XML 文件中，通过读取 XML 文件获取 bean 信息。

基于注解的方式来完成 bean 的注入

1、通过 XML 的方法

2、通过配置类的方式

```java
package com.southwind.configuration;

import com.southwind.entity.Order;
import com.southwind.entity.User;
import
org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configur
ation;

@Configuration
public class BeanConfiguration {

    @Bean
    public User user(){
        return new User();
    }

    @Bean
    public Order order(){
        return new Order();
    }

}
```

```xml
<?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: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">

    <bean class="com.southwind.entity.User"
id="user">
        <property name="id" value="1">
</property>
        <property name="name" value="张三">
</property>
        <property name="age" value="22">
</property>
        <property name="order" ref="order">
</property>
    </bean>
    <bean id="order2"
class="com.southwind.entity.Order">
        <property name="id" value="1">
</property>
        <property name="name" value="订单1">
</property>
    </bean>

    <bean id="order2"
class="com.southwind.entity.Order">
        <property name="id" value="2">
</property>
        <property name="name" value="订单2">
</property>
    </bean>

</beans>
```

```
package com.southwind;

import
com.southwind.configuration.BeanConfiguration;
import com.southwind.entity.Order;
import com.southwind.entity.User;
import
org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.Annotati
onConfigApplicationContext;
import
org.springframework.context.support.ClassPathXm
lApplicationContext;
```

```
public class Test {
    public static void main(String[] args) {
        //启动Spring
        ApplicationContext applicationContext =
new
AnnotationConfigApplicationContext("com.southwi
nd.configuration");
//        ApplicationContext applicationContext
= new
AnnotationConfigApplicationContext(BeanConfigur
ation.class);
        User bean =
applicationContext.getBean(User.class);
        System.out.println(bean);
    }
}
```

3、通过给目标类添加 @Component 进行自动扫描注入

1.2 AOP

面向切面编程，对面向对象编程的一种补充，解耦合

将业务代码和非业务代码进行分离

切面对象

```
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;

@Aspect
@Component
public class LoggerAspect {

    @Before("execution(public int
com.southwind.CalImpl.*(..))")
    public void before(JoinPoint joinPoint){
        //方法名
        String name =
joinPoint.getSignature().getName();
        //参数列表
        String args =
Arrays.toString(joinPoint.getArgs());
        System.out.println(name + "方法的参数是"
+ args);
    }

    @AfterReturning(value = "execution(public
int com.southwind.CalImpl.*(..))",returning =
"result")
    public void afterReturn(JoinPoint
joinPoint,Object result){
        String name =
joinPoint.getSignature().getName();
```

```
        System.out.println(name + "方法的结果是"
+ result);
    }

}
```

切面对象是真正实现类对象的代理，必须要依赖于实现类对象才可以运行

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

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

xmlns:context="http://www.springframework.org/s
chema/context"

xmlns:aop="http://www.springframework.org/schem
a/aop"

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

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

http://www.springframework.org/schema/beans/spr
ing-beans.xsd

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

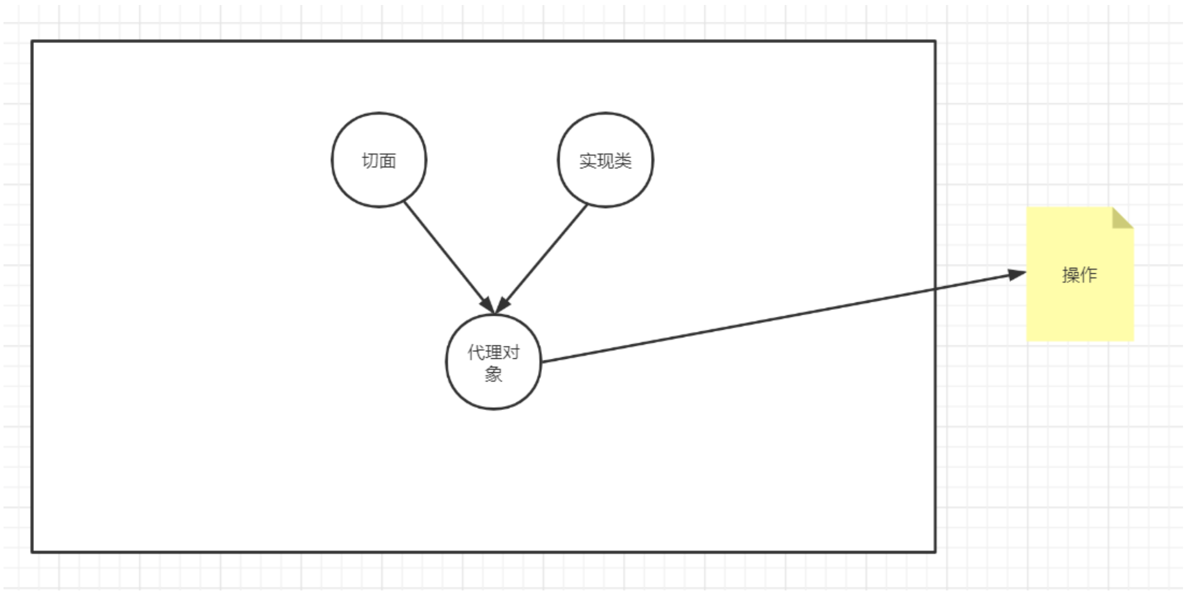
```
http://www.springframework.org/schema/context/spring-context.xsd

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

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

    <context:component-scan base-package="com.southwind"></context:component-scan>

    <aop:aspectj-autoproxy></aop:aspectj-autoproxy>

</beans>
```

1、实现类和切面必须注入到 IoC 中

2、切面必须声明为切面对象

3、必须设置自动代理

4、必须通过接口类来获取


切面
实现类
代理对象
操作