

2018

# Java EE框架 ---Spring

Java EE framework --Spring

王磊

研发部

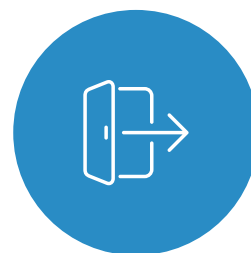
# CONTENTS



创建项目



创建接口



创建通知



织入通知

# 一、创建JavaWeb项目

- 1、新建Java Project
- 2、引入 最小Spring jar包
- 3、构建路径

springAOP

> src

> JRE System Library [JavaSE-1.8]

> Referenced Libraries

> JUnit 4

lib

commons-logging-1.2.jar

log4j-1.2.17.jar

spring-aop-4.3.8.RELEASE.jar

spring-aspects-4.3.8.RELEASE.jar

spring-beans-4.3.8.RELEASE.jar

spring-context-4.3.8.RELEASE.jar

spring-core-4.3.8.RELEASE.jar

spring-expression-4.3.8.RELEASE.jar

spring-jdbc-4.3.8.RELEASE.jar

spring-tx-4.3.8.RELEASE.jar

spring-web-4.3.8.RELEASE.jar

Log4j日志jar包

## 二、创建package，引入log4j配置文件

- 1、创建com.xk.aop包
- 2、创建com.xk.log包
- 3、引入log4j.properties配置文件

springAOP

src

com.xk.aop

com.xk.log

log4j.properties

JRE System Library [JavaSE-1.8]

Referenced Libraries

JUnit 4

lib

```
1 # priority :debug<info<warn<error
2 #you cannot specify every priority with different file for log4j
3 log4j.rootLogger=debug,stdout,info,debug,warn,error
4 #console
5 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
6 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
7 log4j.appender.stdout.layout.ConversionPattern= [%d{yyyy-MM-dd HH:mm:ss a}]:%p %l%m
8 #info log
9 log4j.logger.info=info
10 log4j.appender.info=org.apache.log4j.DailyRollingFileAppender
11 log4j.appender.info.DatePattern='_''yyyy-MM-dd''.log'
12 log4j.appender.info.File=./src/com/xk/log/info.log
13 log4j.appender.info.Append=true
14 log4j.appender.info.Threshold=INFO
15 log4j.appender.info.layout=org.apache.log4j.PatternLayout
16 log4j.appender.info.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss a} [Thread: %t]
17 #debug log
18 log4j.logger.debug=debug
19 log4j.appender.debug=org.apache.log4j.DailyRollingFileAppender
20 log4j.appender.debug.DatePattern='_''yyyy-MM-dd''.log'
21 log4j.appender.debug.File=./src/com/xk/log/debug.log
```

### • 三、编写接口文件 •

- 1、编写ServiceInterface接口文件
- 2、在该接口文件中编写空方法insertMySQLDb ( )

```
package com.xk.aop;  
public interface ServiceInterface {  
    public void insertMySQLDb();  
}
```

## • 四、编写接口文件的实现类文件Service •

- 1、编写ServiceInterface接口文件的实现类文件Service
- 2、在该实现类文件中重写insertMySQLDb ( ) 方法

```
package com.xk.aop;  
public class Service implements ServiceInterface {  
    @Override  
    public void insertMySQLDb() {  
        System.out.println("执行业务逻辑，信息录入MySQL数据库中...");  
    }  
}
```

## • 五、编写前置通知类 •

- 1、通知类分为：前置通知，后置通知，环绕通知，异常通知
- 2、在编写前置通知类之前，必须导入spring jar包
- 3、前置通知类必须要实现MethodBeforeAdvice接口
- 4、在该文件中可以编写与业务逻辑无关的其他内容（比如：写日志，开启事务等）

```
package com.xk.aop;
import java.lang.reflect.Method;
public class MyMethodBeforeAdvice implements MethodBeforeAdvice {
    private static Logger logger = Logger.getLogger(Test.class);
    @Override
    public void before(Method method, Object[] args, Object target) throws Throwable {
        System.out.println("在前置通知中，已经写入日志信息");
        logger.debug("This is debug message.");
    }
}
```

## 六、编写applicationContext.xml配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.springframework.org/schema/beans"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.3.xsd ">
<!-- 配置被代理对象 -->
<bean id="service" class="com.xk.aop.Service"></bean>
<!-- 配置前置通知 -->
<bean id="myMethodBeforeAdvice" class="com.xk.aop.MyMethodBeforeAdvice"></bean>
<!-- 配置代理对象 -->
<bean id="proxyFactoryBean" class="org.springframework.aop.framework.ProxyFactoryBean">
  <!-- 代理的接口集合 -->
  <property name="proxyInterfaces">
    <list>
      <value>com.xk.aop.ServiceInterface</value>
    </list>
  </property>
  <!--配置通知集合 -->
  <property name="interceptorNames">
    <list>
      <value>myMethodBeforeAdvice</value>
    </list>
  </property>
  <!-- 配置代理了哪些对象 -->
  <property name="target" ref="service"></property>
</bean>
</beans>
```



代理对象，无需手动编写



## • 七、编写测试文件 •

```
package com.xk.aop;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Run {
    @Test
    public void test() {
        ApplicationContext ac = new ClassPathXmlApplicationContext
            ("applicationContext.xml");
        ServiceInterface s = (ServiceInterface) ac.getBean("proxyFactoryBean");
        s.insertMySQLDb();
    }
}
```

## 八、进行junit单元测试

[2018-11-24 17:07:41 下午]:DEBUG org.springframework.aop.framework.JdkDyna  
在前置通知中，已经写入日志信息

[2018-11-24 17:07:41 下午]:DEBUG com.xk.aop.MyMethodBeforeAdvice.before  
执行业务逻辑，信息录入MySQL数据库中...

springAOP

src

com.xk.aop

MyMethodBeforeAdvice.java

Run.java

Service.java

ServiceInterface.java

com.xk.log

debug.log

error.log

info.log

warn.log

applicationContext.xml

log4j.properties

JRE System Library [JavaSE-1.8]

Referenced Libraries

JUnit 4

```
448 DEBUG:Could not find key 'spring.liveBeansView.mbeanDomain' in any property source
449 2018-11-24 17:07:41 下午 [Thread: main][ Class:org.springframework.beans.factory.support
450 DEBUG:Returning cached instance of singleton bean 'proxyFactoryBean'
451 2018-11-24 17:07:41 下午 [Thread: main][ Class:org.springframework.beans.factory.support
452 DEBUG:Returning cached instance of singleton bean 'myMethodBeforeAdvice'
453 2018-11-24 17:07:41 下午 [Thread: main][ Class:org.springframework.aop.framework.ProxyFa
454 DEBUG:Advice has changed; recaching singleton instance
455 2018-11-24 17:07:41 下午 [Thread: main][ Class:org.springframework.aop.framework.JdkDyna
456 DEBUG:Creating JDK dynamic proxy: target source is SingletonTargetSource for target ob
457 2018-11-24 17:07:41 下午 [Thread: main][ Class:org.junit.Test >> Method: com.xk.aop.MyMe
458 DEBUG:This is debug message.
```

## 九、增加新需求（业务逻辑后从Oracle读取数据）

新的需求：在正常执行完业务逻辑之后，要求从Oracle数据库中读取数据

1、编写新接口

```
package com.xk.aop;  
  
public interface ServiceInterface2 {  
    public void getDateFromOracle();  
}
```

## 九、增加新需求（业务逻辑后从Oracle读取数据）

2、改写Service服务类，让该类实现两个接口

```
package com.xk.aop;  
public class Service implements ServiceInterface, ServiceInterface2 {  
    @Override  
    public void insertMySQLDb() {  
        System.out.println("执行业务逻辑，信息录入MySQL数据库中...");  
    }  
    @Override  
    public void getDateFromOracle() {  
        System.out.println("从Oracle数据库中读取信息...");  
    }  
}
```


新增部分

## 九、增加新需求（业务逻辑后从Oracle读取数据）

新的需求：在正常执行完业务逻辑之后，要求从Oracle数据库中读取数据

3、编辑配置文件，增加新的接口

```
<!-- 代理的接口集合 -->
<property name="proxyInterfaces">
  <list>
    <value>com.xk.aop.ServiceInterface</value>
    <value>com.xk.aop.ServiceInterface2</value>
  </list>
</property>
```




新增信息

## 九、增加新需求（业务逻辑后从Oracle读取数据）

新的需求：在正常执行完业务逻辑之后，要求从Oracle数据库中读取数据

4、改写junit单元测试类

```
package com.xk.aop;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Run {
    @Test
    public void test() {
        ApplicationContext ac = new ClassPathXmlApplicationContext
            ("applicationContext.xml");
        ServiceInterface s = (ServiceInterface) ac.getBean("proxyFactoryBean");
        s.insertMySQLDb();
        ((ServiceInterface2)s).getDateFromOracle();
    }
}
```



注意：一个类实现两个接口，可以将该类在这两个接口中任意转换。

## 十、单元测试

新的需求：在正常执行完业务逻辑之后，要求从Oracle数据库中读取数据

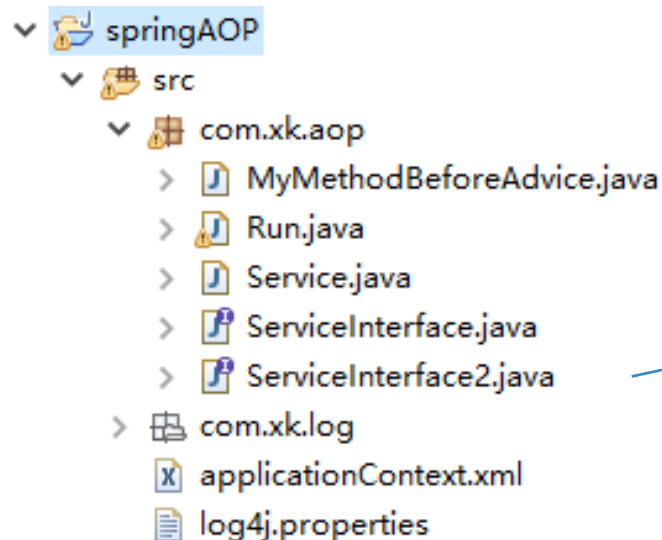
### 5、进行单元测试

[2018-11-24 17:26:41 下午]:DEBUG org.springframework.aop.framework.JdkDy  
在前置通知中，已经写入日志信息

[2018-11-24 17:26:41 下午]:DEBUG com.xk.aop.MyMethodBeforeAdvice.before  
执行业务逻辑，信息录入MySQL数据库中...

在前置通知中，已经写入日志信息

[2018-11-24 17:26:41 下午]:DEBUG com.xk.aop.MyMethodBeforeAdvice.before  
从Oracle数据库中读取信息...



项目整体架构

# 十一、增加新需求（业务逻辑后关闭资源）

新的需求：在正常执行完业务逻辑之后，要求关闭资源

1、编写新类，实现后置通知接口

src

com.xk.aop

- > MyAfterReturningAdvice.java
- > MyMethodBeforeAdvice.java
- > Run.java
- > Service.java
- > ServiceInterface.java
- > ServiceInterface2.java

```
package com.xk.aop;
import java.lang.reflect.Method;
public class MyAfterReturningAdvice implements AfterReturningAdvice {
    @Override
    public void afterReturning(Object returnValue, Method method,
                               Object[] args, Object target)
                               throws Throwable {
        System.out.println("关闭连接资源...");
    }
}
```



## 十一、增加新需求（业务逻辑后关闭资源）

2、修改applicationContext.xml配置文件

增加以下内容：

(1) 配置后置通知 (2) 配置通知集合

```
<!-- 配置被代理对象 -->
<bean id="service" class="com.xk.aop.Service"></bean>
<!-- 配置前置通知 -->
<bean id="myMethodBeforeAdvice" class="com.xk.aop.MyMethodBeforeAdvice"></bean>
<!-- 配置后置通知 -->
<bean id="myAfterReturningAdvice" class="com.xk.aop.MyAfterReturningAdvice"/>
<!-- 配置代理对象 -->
<bean id="proxyFactoryBean" class="org.springframework.aop.framework.ProxyFactoryBean">
    <!-- 代理的接口集合 -->
    <property name="proxyInterfaces">
        <list>
            <value>com.xk.aop.ServiceInterface</value>
            <value>com.xk.aop.ServiceInterface2</value>
        </list>
    </property>
    <!--配置通知集合-->
    <property name="interceptorNames">
        <list>
            <value>myMethodBeforeAdvice</value>
            <value>myAfterReturningAdvice</value>
        </list>
    </property>
</bean>
```

## 十一、增加新需求（业务逻辑后关闭资源）

### 3、junit测试类无需更改，进行单元测试

[2018-11-24 19:35:30 下午]:DEBUG org.springframework.aop.framework.JdkDynamicAopProxy  
在前置通知中，已经写入日志信息

[2018-11-24 19:35:30 下午]:DEBUG com.xk.aop.MyMethodBeforeAdvice.before  
执行业务逻辑，信息录入MySQL数据库中...

关闭连接资源...

在前置通知中，已经写入日志信息

[2018-11-24 19:35:30 下午]:DEBUG com.xk.aop.MyMethodBeforeAdvice.before  
从Oracle数据库中读取信息...

关闭连接资源...

## 十二、织入环绕通知

需求：织入环绕通知

1、增加新类MyMethodInterceptor，继承MethodInterceptor类

```
package com.xk.aop;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
public class MyMethodInterceptor implements MethodInterceptor {
    @Override
    public Object invoke(MethodInvocation arg0) throws Throwable {
        System.out.println("方法调用之前... (环绕通知)");
        Object obj = arg0.proceed();
        System.out.println("方法调用之后... (环绕通知)");
        return obj;
    }
}
```

## 十二、织入环绕通知

### 2、编辑applicationContext.xml配置文件

```
<!-- 配置后置通知 -->
<bean id="myAfterReturningAdvice" class="com.xk.aop.MyAfterReturningAdvice"/>

<!--配置通知集合 -->
<property name="interceptorNames">
    <list>
        <value>myMethodBeforeAdvice</value>
        <value>myAfterReturningAdvice</value>
        <value>myMethodInterceptor</value>
    </list>
</property>
```

## 十二、织入环绕通知

### 3、进行单元测试，观察结果

[2018-11-24 20:04:54 下午]:DEBUG org.springframework.aop.framework.JdkDy  
在前置通知中，已经写入日志信息

[2018-11-24 20:04:54 下午]:DEBUG com.xk.aop.MyMethodBeforeAdvice.before

方法调用之前... (环绕通知)

执行业务逻辑，信息录入MySQL数据库中...

方法调用之后... (环绕通知)

关闭连接资源...

在前置通知中，已经写入日志信息

[2018-11-24 20:04:54 下午]:DEBUG com.xk.aop.MyMethodBeforeAdvice.before

方法调用之前... (环绕通知)

从Oracle数据库中读取信息...

方法调用之后... (环绕通知)

关闭连接资源...

## 十三、织入异常通知

需求：织入异常通知

注：异常通知为标识性通知，没有任何方法。实现该异常接口的类，必须要有以下方法：

(1) `public void afterThrowing(Throwable throwable);`

(2) `public void afterThrowing(Method m, Object[] os, Object target, Exception throwable);`

第一个方法只需要接收一个参数：需要抛出的异常。

第二个方法接收异常、被调用的方法、参数以及目标对象。

## 十三、织入异常通知

需求：织入异常通知

1、增加新类MyThrowsAdvice，继承ThrowsAdvice接口

```
package com.xk.aop;
import java.lang.reflect.Method;
public class MyThrowsAdvice implements ThrowsAdvice {
    public void afterThrowing(Method m, Object[] os, Object target,
                             Exception throwable) {
        System.out.println("出错了...异常通知被执行");
    }
}
```

## 十三、织入异常通知

### 2、编辑applicationContext.xml配置文件

```
<!-- 配置异常通知 -->
<bean id="myThrowsAdvice" class="com.xk.aop.MyThrowsAdvice"/>

<!--配置通知集合 -->
<property name="interceptorNames">
    <list>
        <value>myMethodBeforeAdvice</value>
        <value>myAfterReturningAdvice</value>
        <value>myMethodInterceptor</value>
        <value>myThrowsAdvice</value>
    </list>
</property>
```



## 十三、织入异常通知

### 3、人为设置错误

```
package com.xk.aop;
public class Service implements ServiceInterface, ServiceInterface2 {
    @Override
    public void insertMySQLDb() {
        System.out.println("执行业务逻辑，信息录入MySQL数据库中...");
    }
    @Override
    public void getDateFromOracle() {
        System.out.println("从Oracle数据库中读取信息...");
        int x = 13/0;
    }
}
```

## 十三、织入异常通知

### 4、进行单元测试，观察结果

[2018-11-24 20:29:36 下午]:DEBUG org.springframework.aop.framework.adapt  
在前置通知中，已经写入日志信息

[2018-11-24 20:29:36 下午]:DEBUG com.xk.aop.MyMethodBeforeAdvice.before  
方法调用之前... (环绕通知)

执行业务逻辑，信息录入MySQL数据库中...

方法调用之后... (环绕通知)

关闭连接资源...

[2018-11-24 20:29:36 下午]:DEBUG org.springframework.aop.framework.adapt  
在前置通知中，已经写入日志信息

[2018-11-24 20:29:36 下午]:DEBUG com.xk.aop.MyMethodBeforeAdvice.before  
方法调用之前... (环绕通知)

从Oracle数据库中读取信息...

[2018-11-24 20:29:36 下午]:DEBUG org.springframework.aop.framework.adapt  
出错了...异常通知被执行

# • 总结 •

- 1、定义接口
- 2、编写被对象
- 3、编写前置通知（前置通知会在目标方法调用之前被调用）
- 4、配置文件applicationContext.xml
  - 4.1 配置被代理对象
  - 4.2 配置通知
  - 4.3 配置代理对象（是proxyFactoryBean对象的实例）
    - 4.3.1 配置代理接口
    - 4.3.2 织入通知
    - 4.3.3 配置被代理对象
- 5、运行并观察结果