## J2EE高级开发框架

## 一、Spring框架简介

- Spring框架是一个免费的、开源的、轻量级应用程序开发框架,其目的是为了简化企业级应用程序的开发,降低开发者的开发难度
- Spirng框架提供了AOP和IOC应用,其目的是将应程序之间的模块进行解耦合
- Spring框架提供了一整套解决方案,开发者可以使用其自身的功能,也可以看整合第三方优秀框架或技术,开发中具体使用哪种技术可以自由选择

## 二、Spring框架优势

- 方便解耦,简化开发
- AOP编程支持
- 声明式事务支持
- 降低J2EE中api的使用难度
- 方便整合其他优秀框架

一句话概括: Spring框架的本质是管理软件中的对象,即对象的创建和维护对象之间的关系

## 三、Spring框架的架构

Spring框架最初的目的是为了整合一切优秀资源,然后对外提供统一的服务。Spring模块构建在核心容器之上,核心容器定义了创建、存储和管理bean的方式,如图所示:

# Spring AOP Source-level metadata AOP infrastructure



Spring ORM

Hibernate support

iBats support

JDO support

# Spring Web WebApplicationContext Mutipart resolver Web utilities







说明: Spring模块每一个可以单独存在和使用,也可以联合一个或多个模块进行开发。各个模块的功能如下:

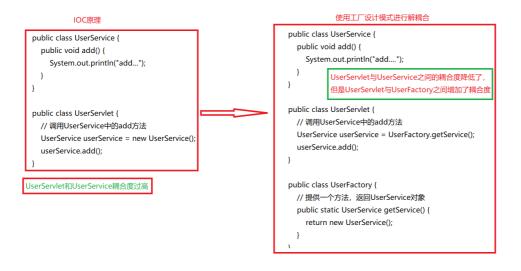
| 模块                         | 说明  |
|----------------------------|---|
| 核心容<br>器<br>Spring<br>Core | 核心容器,提供Spring框架的基本功能。核心容器的主要组件是BeanFactory,它是工厂模式的实现。BeanFactory 使用控制反转(IOC)模式,将应用程序的配置和依赖性规范与实际的应用程序代码分开。  |
| Spring<br>Context          | Spring上下文,是一个配置文件,向 Spring 框架提供上下文信息。Spring 上下文包括企业服务,例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。   |
| Spring<br>AOP              | 通过配置管理特性,Spring AOP 模块直接将面向切面的编程功能集成到了 Spring 框架中。可以很容易地使 Spring框架管理的任何对象支持AOP。Spring AOP模块为基于 Spring 的应用程序中的对象提供了事务管理服务。通过使用 Spring AOP,就可以将声明性事务管理集成到应用程序中。 |
| Spring<br>DAO              | JDBC DAO 抽象层提供了有意义的异常层次结构,可用该结构来管理异常处理和不同数据库供应商抛出的错误消息。异常层次结构简化了错误处理,并且极大地降低了需要编写的异常代码数量(例如打开和关闭连接)。Spring DAO 的面向JDBC 的异常遵从通用的 DAO 异常层次结构。                   |
| Spring<br>ORM              | Spring 框架插入了若干个 ORM 框架,从而提供了 ORM 的对象关系工具,其中包括JDO、Hibernate和iBatis SQL<br>Map。所有这些都遵从 Spring 的通用事务和 DAO 异常层次结构   |
| Spring<br>Web              | Web上下文模块建立在应用程序上下文模块之上,为基于 Web 的应用程序提供了上下文。所以Spring 框架支持与<br>Jakarta Struts的集成。Web模块还简化了处理多部分请求以及将请求参数绑定到域对象的工作。  |
| Spring<br>MVC框<br>架        | MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口,MVC 框架变成为高度可配置的,MVC 容纳了大量视图技术,其中包括 JSP、Velocity、Tiles、iText 和 POI。  |

## 四、Spring框架的IOC (重点)

## 1、IOC概述

IOC(控制反转),传统方式对象的创建需要通过关键字new进行,在Spring框架中,不再使用关键字new创建对象,而是将对象的创建、存储和管理交给Spring,IOC通过xml配置文件或者注解实现这种功能。IOC底层使用到的技术:xml配置文件、dom4i解析xml文件、工厂设计模式、反射。

## 2、IOC底层原理分析



#### 上述过程仍有耦合度



## 3、IOC入门案例

1. 导入jar包

## Spring框架下载

创建普通的web项目,导入Spring框架核心jar包,项目结构如下:

Scratches and Consoles

- ▼ Image:\DingLi\class210\spring idea .idea > out src > com.hbnu.entity 🌄 applicationContext.xml web WEB-INF V III lib commons-logging-1.2.jar log4j-1.2.17.jar > spring-beans-4.2.4.RELEASE.jar spring-context-4.2.4.RELEASE.jar > spring-core-4.2.4.RELEASE.jar spring-expression-4.2.4.RELEASE.jar keb.xml 🚚 index.jsp a spring.iml External Libraries
- 2. 创建普通类User.java

```
package com.hbnu.entity;
 1
 2
 3
    * @author 陈迪凯
 4
    * @date 2021-03-01 10:03
 5
 6
    */
 7
    public class User {
 8
        public void add() {
 9
            System.out.println("IOC test.....");
10
        }
11 }
```

## 3. 创建配置文件

Spring核心配置文件路径和名称没有固定要求,官方建议:配置文件放到src目录下,配置文件名称applicationContext.xml

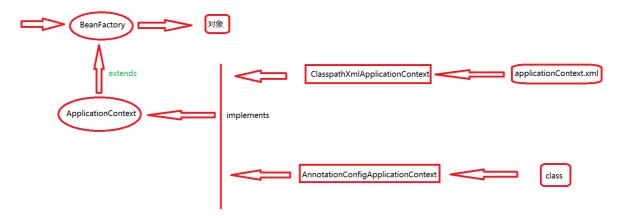
```
1 <?xml version="1.0" encoding="UTF-8" ?>
2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="
 4
 5
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd">
 6
 7
        <!-- 配置bean
        id属性:必须唯一
 8
9
        class属性: 需要交给spring管理的类的全路径
10
        <bean id="user" class="com.hbnu.entity.User" />
11
12
    </beans>
```

## 4. 测试IOC

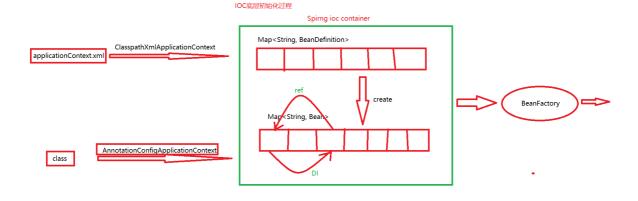
```
1
   package com.hbnu.entity;
 2
   import org.junit.Test;
    import org.springframework.context.ApplicationContext;
    import
    org.springframework.context.support.ClassPathXmlApplicationContext;
 6
   /**
7
 8
    * @author 陈迪凯
    * @date 2021-03-01 10:11
9
10
    */
    public class IOCTest {
11
12
13
        @Test
        public void userTest() {
14
15
            // 1、加载spring核心配置文件
            ApplicationContext applicationContext = new
16
    ClassPathXmlApplicationContext("applicationContext.xml");
17
18
            /// 2、获取bean对象
19
            User user = (User) applicationContext.getBean("user");
20
21
            user.add();
22
        }
```

## 4、IOC底层api

Spring创建对象底层api



## 5、IOC初始化过程



## 五、Spring中的Bean管理

Spring中Bean管理的方式:通过无参构造函数(<mark>重点</mark>)、通过静态工厂(了解)、通过实例工厂(<mark>了解</mark>)

## 1、通过无参构造函数

• 创建User类

```
package com.hbnu.entity;
2
3
4
    * @author 陈迪凯
    * @date 2021-03-01 10:03
 5
    */
    public class User {
7
8
        public void add() {
9
            System.out.println("IOC test.....");
10
        }
11
   }
```

• 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
1
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 2
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
           xsi:schemaLocation="
 5
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd">
 6
 7
        <!-- 配置bean
       id属性:必须唯一
 8
 9
        class属性: 需要交给spring管理的类的全路径
10
11
        <bean id="user" class="com.hbnu.entity.User"/>
12
```

## 2、通过静态工厂创建Bean对象

• 创建工厂

```
1
    package com.hbnu.entity;
2
 3
    /**
 4
    * @author 陈迪凯
    * @date 2021-03-08 8:14
 6
    */
7
   public class UserFactory {
8
9
        public static User getUser() {
10
            return new User();
11
        }
12
   }
```

• 配置文件

```
1 <?xml version="1.0" encoding="UTF-8" ?>
 2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="
 4
 5
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd">
 6
 7
       <!-- 配置bean
 8
       id属性:必须唯一
9
       class属性: 需要交给spring管理的类的全路径
        -->
10
        <!--<bean id="user" class="com.hbnu.entity.User"/>-->
11
12
        <!-- 通过静态工厂管理Bean对象 -->
13
        <bean id="factory" class="com.hbnu.entity.UserFactory" factory-</pre>
14
    method="getUser"></bean>
15
    </beans>
```

• 测试

```
package com.hbnu.entity;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
```

```
5
   import
    org.springframework.context.support.ClassPathXmlApplicationContext;
 6
   /**
7
8
    * @author 陈迪凯
9
    * @date 2021-03-01 10:11
10
    */
11
    public class IOCTest {
12
13
        @Test
14
        public void userTest() {
15
            // 1、加载spring核心配置文件
16
            ApplicationContext applicationContext = new
    ClassPathXmlApplicationContext("applicationContext.xml");
17
            /// 2、获取bean对象
18
19
            User user = (User) applicationContext.getBean("factory");
20
21
           user.add();
22
        }
23
    }
```

## 3、通过实例工厂创建Bean对象

• 创建工厂类

```
1
    package com.hbnu.entity;
 2
    /**
 3
    * @author 陈迪凯
 4
    * @date 2021-03-08 8:14
 5
    */
 6
 7
    public class UserFactory {
 8
 9
        public User getUser() {
10
            return new User();
11
        }
12
   }
```

• 配置文件

```
1
    <?xml version="1.0" encoding="UTF-8" ?>
 2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
           xsi:schemaLocation="
            http://www.springframework.org/schema/beans
 5
 6
            http://www.springframework.org/schema/beans/spring-beans.xsd">
 7
        <!-- 配置bean
        id属性:必须唯一
 8
 9
        class属性: 需要交给spring管理的类的全路径
10
11
        <!--<bean id="user" class="com.hbnu.entity.User"/>-->
12
13
        <!-- 通过静态工厂管理Bean对象 -->
        <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-
14
    method="getUser"></bean>-->
15
```

```
16<!-- 通过实例工厂管理Bean对象 -->17<bean id="factory" class="com.hbnu.entity.UserFactory"></bean>18<bean id="user" factory-bean="factory" factory-method="getUser"></bean>19</bean>
```

```
package com.hbnu.entity;
2
   import org.junit.Test;
   import org.springframework.context.ApplicationContext;
    import
    org.spring framework.context.support.Class {\tt PathXmlApplicationContext};\\
6
8
    * @author 陈迪凯
9
    * @date 2021-03-01 10:11
10
11
   public class IOCTest {
12
13
        @Test
14
        public void userTest() {
15
            // 1、加载spring核心配置文件
16
            ApplicationContext applicationContext = new
    ClassPathXmlApplicationContext("applicationContext.xml");
17
18
            /// 2、获取bean对象
19
            User user = (User) applicationContext.getBean("user");
20
21
            user.add();
22
        }
23
    }
```

## 六、Spring中Bean标签常见属性

| 属性名称                | 描述  |  |
|---------------------|---|--|
| id                  | 是一个 Bean 的唯一标识符,Spring 容器对 Bean 的配置和管理都通过该属性完成  |  |
| name                | Spring 容器同样可以通过此属性对容器中的 Bean 进行配置和管理,name 属性中可以为 Bean 指定多个名称,每个名称之间用逗号或分号隔开                         |  |
| class               | 该属性指定了 Bean 的具体实现类,它必须是一个完整的类名,使用类的全限定名   |  |
| scope               | 用于设定 Bean 实例的作用域,其属性值有 singleton(单例)、prototype(原型)、request、session 和 global Session。其默认值是 singleton |  |
| constructor-<br>arg | <br><br><br><br>  |  |
| property            | <br><br>bean>元素的子元素,用于调用 Bean 实例中的 Set 方法完成属性赋值,从而完成依赖注入。该元素的 name 属性指定 Bean 实例中的相应属性名              |  |
| ref                 | <pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>            |  |
| value               | <pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>            |  |
| list                | 用于封装 List 或数组类型的依赖注入  |  |
| set                 | 用于封装 Set 类型属性的依赖注入  |  |
| map                 | 用于封装 Map 类型属性的依赖注入  |  |
| entry               | <map> 元素的子元素,用于设置一个键值对。其 key 属性指定字符串类型的键值,ref 或 value 子元素指定其值</map>                                 |  |

## Scope属性:

1. singleton: 单例的, Spring容器只会创建一个Bean对象

- 2. prototype:多例的,Spring容器创建多个Bean对象
- 3. request: 在WEB项目中, Spring创建Bean对象, 将Bean对象存入request域
- 4. session: 在WEB项目中, Spring创建Bean对象, 将Bean对象存入session域
- 5. globalSession:在WEB项目中,基于Porlet环境(基于java的web组件)创建Bean,如果没有Porlet环境,那么就跟session一样

## 七、属性注入

创建对象的过程中,给类中的属性设置值。属性注入有三种方式:通过有参构造函数、通过set方法(<mark>重点</mark>)、通过接口注入(Spring框架不支持)

#### 第一种:通过有参构造函数

```
public class User {
    private String username;
    public User(String username) {
        this.username = username;
    }
}
User user = new User("chendikai");
```

## 第二种、通过set方法注入属性值

```
public class User {
    private String username;
    public void setUsername(String username) {
        this.username = username;
    }
}
User user = new User();
user.setUsername("chendikai");
```

## 第三种、通过接口注入属性

```
public interface UserInterface {
   public void del(String name);
}

public class User implements UserInterfact {
   private String username;
   public void del(String name) {
      this.username = name;
   }
}

User user = new User();
   user.del("chendikai");
```

#### 1、通过构造函数注入属性

• 创建User类

```
1
    package com.hbnu.entity;
 2
 3
 4
     * @author 陈迪凯
 5
    * @date 2021-03-01 10:03
 6
     */
 7
    public class User {
 8
        private String username;
 9
        public User(String username) {
10
11
            this.username = username;
        }
12
13
14
        public void add() {
            System.out.println("add....." + username);
15
16
    }
17
```

## • 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
1
2
   <beans xmlns="http://www.springframework.org/schema/beans"</pre>
3
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="
4
           http://www.springframework.org/schema/beans
5
6
           http://www.springframework.org/schema/beans/spring-beans.xsd">
7
       <!-- 配置bean
       id属性:必须唯一
8
9
       class属性:需要交给spring管理的类的全路径
```

```
10
11
        <!--<bean id="user" class="com.hbnu.entity.User"/>-->
12
13
        <!-- 通过静态工厂管理Bean对象 -->
14
        <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-
    method="getUser"></bean>-->
15
        <!-- 通过实例工厂管理Bean对象 -->
16
17
        <!--<bean id="factory" class="com.hbnu.entity.UserFactory"></bean>
        <bean id="user" factory-bean="factory" factory-method="getUser">
18
    </bean>-->
19
        <!-- 属性注入 -->
20
21
        <!-- 通过有参构造函数注入属性 -->
        <bean id="user" class="com.hbnu.entity.User">
22
            <constructor-arg name="username" value="chendikai">
23
    </constructor-arg>
24
        </bean>
25
    </beans>
```

```
package com.hbnu.entity;
 2
   import org.junit.Test;
 3
    import org.springframework.context.ApplicationContext;
    org.spring framework.context.support.Class {\tt PathXmlApplicationContext};\\
 6
    /**
7
 8
    * @author 陈迪凯
    * @date 2021-03-01 10:11
9
10
    */
   public class IOCTest {
11
12
13
        @Test
14
        public void userTest() {
            // 1、加载spring核心配置文件
15
16
            ApplicationContext applicationContext = new
    ClassPathXmlApplicationContext("applicationContext.xml");
17
            // 2、获取bean对象
18
19
            User user = (User) applicationContext.getBean("user");
20
21
            user.add();
22
        }
23
    }
```

## 2、通过set方法注入属性

创建User类

```
6
 7
    public class User {
 8
        private String username;
 9
10
        public void setUsername(String username) {
11
            this.username = username;
12
        }
13
        public void add() {
14
15
            System.out.println("add....." + username);
16
        }
17
    }
```

#### • 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
 1
 2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
          xsi:schemaLocation="
            http://www.springframework.org/schema/beans
 5
 6
            http://www.springframework.org/schema/beans/spring-beans.xsd">
 7
        <!-- 配置bean
 8
        id属性: 必须唯一
 9
        class属性:需要交给spring管理的类的全路径
10
11
        <!--<bean id="user" class="com.hbnu.entity.User"/>-->
12
13
        <!-- 通过静态工厂管理Bean对象 -->
        <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-
14
    method="getUser"></bean>-->
15
16
        <!-- 通过实例工厂管理Bean对象 -->
        <!--<bean id="factory" class="com.hbnu.entity.UserFactory"></bean>
17
        <bean id="user" factory-bean="factory" factory-method="getUser">
18
    </bean>-->
19
20
        <!-- 属性注入 -->
        <!-- 通过有参构造函数注入属性 -->
21
22
        <!--
        <bean id="user" class="com.hbnu.entity.User">
23
24
           <!&ndash;
25
            name属性: 类中的属性名
26
            value属性: 注入的值
27
            –>
28
            <constructor-arg name="username" value="chendikai">
    </constructor-arg>
29
        </bean>
30
        -->
31
        <!-- 通过set方法注入属性 -->
        <bean id="user" class="com.hbnu.entity.User">
32
33
            cproperty name="username" value="zhangsanfeng"></property>
34
        </bean>
35
    </beans>
```

## • 测试

```
1 package com.hbnu.entity;
2
   import org.junit.Test;
   import org.springframework.context.ApplicationContext;
4
    org.spring framework.context.support.Class {\tt PathXmlApplicationContext};\\
 6
    /**
7
8
    * @author 陈迪凯
    * @date 2021-03-01 10:11
9
10
11
   public class IOCTest {
12
13
        @Test
        public void userTest() {
14
15
           // 1、加载spring核心配置文件
            ApplicationContext applicationContext = new
16
    ClassPathXmlApplicationContext("applicationContext.xml");
17
18
            // 2、获取bean对象
19
            User user = (User) applicationContext.getBean("user");
20
21
            user.add();
22
        }
23
   }
```

## 3、注入对象属性

• 创建UserDao类

```
package com.hbnu.entity;
2
 3
4
    * @author 陈迪凯
    * @date 2021-03-08 9:31
6
    */
7
   public class UserDao {
8
        public void printUserDao() {
9
           System.out.println("UserDao.....");
10
11
  }
```

• 创建UserService类

```
1
    package com.hbnu.entity;
 2
 3
   /**
    * @author 陈迪凯
 4
 5
    * @date 2021-03-08 9:33
    */
 6
    public class UserService {
 7
 8
        private UserDao userDao;
 9
10
        public void setUserDao(UserDao userDao) {
11
            this.userDao = userDao;
12
        }
13
```

```
public void printUserService() {
    System.out.println("UserService...");
    userDao.printUserDao();
}
```

## • 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
 1
 2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="
 4
 5
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd">
 6
        <!-- 配置bean
 7
        id属性: 必须唯一
 8
 9
        class属性: 需要交给spring管理的类的全路径
        -->
10
        <!--<bean id="user" class="com.hbnu.entity.User"/>-->
11
12
13
        <!-- 通过静态工厂管理Bean对象 -->
        <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-
14
    method="getUser"></bean>-->
15
        <!-- 通过实例工厂管理Bean对象 -->
16
17
        <!--<bean id="factory" class="com.hbnu.entity.UserFactory"></bean>
        <bean id="user" factory-bean="factory" factory-method="getUser">
18
    </bean>-->
19
20
        <!-- 属性注入 -->
21
        <!-- 通过有参构造函数注入属性 -->
22
        <!--
23
        <bean id="user" class="com.hbnu.entity.User">
            <!&ndash;
24
25
            name属性: 类中的属性名
26
            value属性: 注入的值
27
            –>
            <constructor-arg name="username" value="chendikai">
28
    </constructor-arg>
29
        </bean>
30
        -->
        <!-- 通过set方法注入属性 -->
31
        <!--<bean id="user" class="com.hbnu.entity.User">
32
33
            cproperty name="username" value="zhangsanfeng"></property>
34
        </bean>-->
35
        <bean id="user" class="com.hbnu.entity.UserDao"></bean>
36
37
        <bean id="userService" class="com.hbnu.entity.UserService">
38
39
            cproperty name="userDao" ref="user"></property>
40
        </bean>
41
    </beans>
```

## • 测试

```
1 package com.hbnu.entity;
```

```
2
    import org.junit.Test;
 4
    import org.springframework.context.ApplicationContext;
 5
    import
    org.springframework.context.support.ClassPathXmlApplicationContext;
 6
    /**
7
8
    * @author 陈迪凯
9
    * @date 2021-03-01 10:11
10
   public class IOCTest {
11
12
13
        @Test
14
        public void userTest() {
            // 1、加载spring核心配置文件
15
16
            ApplicationContext applicationContext = new
    ClassPathXmlApplicationContext("applicationContext.xml");
17
18
            // 2、获取bean对象
19
            UserService userService = (UserService)
    applicationContext.getBean("userService");
20
21
            userService.printUserService();
22
        }
23
   }
```

#### 4、注入复杂数据类型

数组、集合 (list、map) 、Properties

创建DataType类

```
package com.hbnu.entity;
2
 3
   import java.util.List;
   import java.util.Map;
    import java.util.Properties;
 6
    /**
7
8
     * @author 陈迪凯
9
    * @date 2021-03-08 10:12
10
    */
11
   public class DataType {
12
        private String[] arr;
13
        private List<String> list;
14
        private Map<String, String> map;
15
        private Properties properties;
16
17
        public String[] getArr() {
18
            return arr;
19
        }
20
21
        public void setArr(String[] arr) {
22
           this.arr = arr;
23
        }
24
        public List<String> getList() {
25
26
            return list;
```

```
27
28
29
        public void setList(List<String> list) {
            this.list = list;
30
31
32
33
        public Map<String, String> getMap() {
34
             return map;
        }
35
36
37
        public void setMap(Map<String, String> map) {
38
            this.map = map;
39
        }
40
41
        public Properties getProperties() {
42
             return properties;
43
        }
44
        public void setProperties(Properties properties) {
45
46
            this.properties = properties;
47
        }
48
49
        public void test() {
            System.out.println("arr:" + arr);
50
51
             System.out.println("list:" + list);
             System.out.println("map:" + map);
52
53
             System.out.println("properties:" + properties);
54
        }
55
    }
```

## • 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 2
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 3
 4
           xsi:schemaLocation="
 5
            http://www.springframework.org/schema/beans
 6
            http://www.springframework.org/schema/beans/spring-beans.xsd">
        <!-- 配置bean
 7
        id属性:必须唯一
 8
 9
        class属性: 需要交给spring管理的类的全路径
10
        <!--<bean id="user" class="com.hbnu.entity.User"/>-->
11
12
13
        <!-- 通过静态工厂管理Bean对象 -->
        <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-
14
    method="getUser"></bean>-->
15
16
        <!-- 通过实例工厂管理Bean对象 -->
        <!--<bean id="factory" class="com.hbnu.entity.UserFactory"></bean>
17
18
        <bean id="user" factory-bean="factory" factory-method="getUser">
    </bean>-->
19
        <!-- 属性注入 -->
20
        <!-- 通过有参构造函数注入属性 -->
21
22
        <!--
        <bean id="user" class="com.hbnu.entity.User">
23
```

```
24
           &1t;!–
25
           name属性: 类中的属性名
26
           value属性: 注入的值
27
           –>
28
           <constructor-arg name="username" value="chendikai">
    </constructor-arg>
29
       </bean>
30
       -->
31
       <!-- 通过set方法注入属性 -->
32
       <!--<bean id="user" class="com.hbnu.entity.User">
           cproperty name="username" value="zhangsanfeng"></property>
33
34
       </bean>-->
35
       <!--<bean id="user" class="com.hbnu.entity.UserDao"></bean>
36
37
       <bean id="userService" class="com.hbnu.entity.UserService">
38
39
           cproperty name="userDao" ref="user"></property>
40
       </bean>-->
41
       <bean id="dataType" class="com.hbnu.entity.DataType">
42
           <!-- 1、数组类型 -->
43
           cproperty name="arr">
44
45
               st>
                  <value>铠</value>
46
47
                  <value>妲己</value>
                  <value>小乔</value>
48
               </list>
49
50
           </property>
51
52
           <!-- 2、list集合类型 -->
53
           property name="list">
54
              st>
55
                  <value>张三丰</value>
56
                  <value>张翠山</value>
57
                  <value>张无忌</value>
58
               </list>
59
           </property>
60
           <!-- 3、map集合类型 -->
61
62
           cproperty name="map">
63
               <map>
                  <entry key="name" value="陈迪凯"></entry>
64
                  <entry key="gender" value="男"></entry>
65
                  <entry key="address" value="湖北黄石"></entry>
66
67
               </map>
68
           69
70
           <!-- 4、Properties类型 -->
71
           roperty name="properties">
72
               ops>
73
                  74
                   key="url">jdbc:mysql:///hbnu
                   key="username">root>
75
76
                  prop key="password">chendikai</prop>
77
               </props>
78
           79
       </bean>
80
    </beans>
```

```
package com.hbnu.entity;
2
   import org.junit.Test;
 3
    import org.springframework.context.ApplicationContext;
    org.spring framework.context.support.Class {\tt PathXmlApplicationContext};\\
 6
7
 8
    * @author 陈迪凯
9
    * @date 2021-03-01 10:11
10
    public class IOCTest {
11
12
13
        @Test
        public void userTest() {
14
15
            // 1、加载spring核心配置文件
16
            ApplicationContext applicationContext = new
    ClassPathXmlApplicationContext("applicationContext.xml");
17
            // 2、获取bean对象
18
19
            DataType dataType = (DataType)
    applicationContext.getBean("dataType");
20
21
            dataType.test();
22
        }
23
    }
```

## 八、Spring中注解开发

## 1、注解开发案例

• 项目结构

导入aop的jar包

```
spring E:\DingLi\class210\spring
    idea .idea
    out
    src
    com.hbnu
          annotation
       entity
       🌄 applicationContext.xml
    web
    WEB-INF
       V III lib
            commons-logging-1.2.jar
          log4j-1.2.17.jar
          > || spring-aop-4.2.4.RELEASE.jar
          spring-beans-4.2.4.RELEASE.jar
            spring-context-4.2.4.RELEASE.jar
           spring-core-4.2.4.RELEASE.jar
            spring-expression-4.2.4.RELEASE.jar
          🎅 web.xml
       🚚 index.jsp
    a spring.iml
External Libraries
 Scratches and Consoles
```

## • 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
 1
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 2
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
           xmlns:aop="http://www.springframework.org/schema/aop"
 5
           xmlns:context="http://www.springframework.org/schema/context"
           xsi:schemaLocation="
 6
 7
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
 8
 9
            http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/beans/spring-aop.xsd
10
            http://www.springframework.org/schema/context
11
12
            http://www.springframework.org/schema/context/spring-
    context.xsd">
13
        <!-- 开启注解扫描 -->
14
        <context:component-scan base-package="com.hbnu.annotation">
15
    </context:component-scan>
16
    </beans>
```

• 创建User类

```
package com.hbnu.annotation;
 3
   import org.springframework.stereotype.Component;
4
5
    * @author 陈迪凯
6
    * @date 2021-03-08 10:45
    */
8
9
   @Component(value = "user")
10 public class User {
        public void add() {
11
12
            System.out.println("annotation....add...");
13
       }
14
   }
```

```
package com.hbnu.annotation;
2
3
   import org.junit.Test;
   import org.springframework.context.ApplicationContext;
4
    org.springframework.context.support.ClassPathXmlApplicationContext;
 6
    /**
7
8
    * @author 陈迪凯
9
    * @date 2021-03-08 10:51
10
11
  public class AnnotationTest {
12
        @Test
13
14
        public void annoTest() {
            ApplicationContext applicationContext = new
15
    ClassPathXmlApplicationContext("annotation.xml");
16
17
            User user = (User) applicationContext.getBean("user");
18
19
            user.add();
        }
20
21
   }
```

在Spring中创建对象的注解有四个: @Component、@Controller(WEB层)、@Repository (数据层)、@Service (业务层)

## 2、注解注入类类型

创建UserDao

```
package com.hbnu.annotation;

import org.springframework.stereotype.Component;

import javax.xml.crypto.KeySelector;

/**

author 陈迪凯

adate 2021-03-08 11:05
```

```
10 */
11 @Component(value = "userDao")
12 public class UserDao {
13    public void add() {
14        System.out.println("userdao...add...");
15    }
16 }
```

• 创建UserService

```
1
    package com.hbnu.annotation;
 2
 3
    import org.springframework.beans.factory.annotation.Autowired;
 4
    import org.springframework.stereotype.Component;
 5
 6
 7
     * @author 陈迪凯
 8
     * @date 2021-03-08 11:06
 9
    @Component(value = "userService")
10
11
    public class UserService {
12
        // @Autowired
13
        @Resource(name = "userDao")
14
15
        private UserDao userDao;
16
        public void printUserService() {
17
18
            System.out.println("UserService.....");
19
            userDao.add();
20
        }
21
    }
```

## • 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
 1
 2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:aop="http://www.springframework.org/schema/aop"
 4
 5
           xmlns:context="http://www.springframework.org/schema/context"
           xsi:schemaLocation="
 6
 7
            http://www.springframework.org/schema/beans
 8
            http://www.springframework.org/schema/beans/spring-beans.xsd
 9
            http://www.springframework.org/schema/aop
10
            http://www.springframework.org/schema/beans/spring-aop.xsd
11
            http://www.springframework.org/schema/context
12
            http://www.springframework.org/schema/context/spring-
    context.xsd">
13
14
        <!-- 开启注解扫描 -->
15
        <context:component-scan base-package="com.hbnu.annotation">
    </context:component-scan>
16
    </beans>
```

• 测试

```
1 package com.hbnu.annotation;
```

```
2
    import org.junit.Test;
4
    import org.springframework.context.ApplicationContext;
 5
    import
    org.springframework.context.support.ClassPathXmlApplicationContext;
 6
    /**
7
    * @author 陈迪凯
8
9
    * @date 2021-03-08 10:51
10
   public class AnnotationTest {
11
12
13
        @Test
14
        public void annoTest() {
15
            ApplicationContext applicationContext = new
    ClassPathXmlApplicationContext("annotation.xml");
16
17
            UserService userService = (UserService)
    applicationContext.getBean("userService");
18
19
            userService.printUserService();
20
        }
21 }
```

## 3、注解和xml配置文件混合使用

• 创建一个UserDao

```
package com.hbnu.annotation;
2
 3
   import org.springframework.stereotype.Component;
4
   /**
 5
    * @author 陈迪凯
6
7
    * @date 2021-03-09 17:19
8
   public class UserDao {
9
10
        public void printUserDao() {
11
            System.out.println("UserDao.....printUserDao.....");
        }
12
13
   }
```

• 创建一个UserService

```
package com.hbnu.annotation;
2
 3
   import org.springframework.beans.factory.annotation.Autowired;
 4
   import org.springframework.stereotype.Component;
6
   import javax.annotation.Resource;
7
8
9
     * @author 陈迪凯
     * @date 2021-03-09 17:21
10
11
12
```

```
13
    public class UserService {
14
15
        private UserDao userDao:
16
17
        public void printUserService() {
18
            System.out.println("UserService.....printUserService.....");
19
            userDao.printUserDao();
20
        }
21
    }
```

## • 配置spring核心配置文件

```
1
    <?xml version="1.0" encoding="utf-8" ?>
 2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
           xmlns:aop="http://www.springframework.org/schema/aop"
 5
           xmlns:context="http://www.springframework.org/schema/context"
           xsi:schemaLocation="
 6
            http://www.springframework.org/schema/beans
 7
            http://www.springframework.org/schema/beans/spring-beans.xsd
 8
 9
            http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop.xsd
10
            http://www.springframework.org/schema/context
11
            http://www.springframework.org/schema/context/spring-
12
    context.xsd">
13
14
        <!-- 配置包扫描注解 -->
15
        <context:component-scan base-package="com.hbnu.annotation">
    </context:component-scan>
16
17
        <!-- 配置bean对象 -->
        <bean id="userDao" class="com.hbnu.annotation.UserDao"></bean>
18
        <bean id="userService" class="com.hbnu.annotation.UserService">
19
    </bean>
20
    </beans>
```

## 修改UserService

```
package com.hbnu.annotation;
 2
 3
    import org.springframework.beans.factory.annotation.Autowired;
 4
    import org.springframework.stereotype.Component;
    import javax.annotation.Resource;
 7
 8
    /**
 9
     * @author 陈迪凯
    * @date 2021-03-09 17:21
10
11
     */
    public class UserService {
12
13
14
        @Resource(name = "userDao")
15
        private UserDao userDao;
16
17
        public void printUserService() {
            System.out.println("UserService.....printUserService.....");
18
```

```
19     userDao.printUserDao();
20    }
21 }
```

```
package com.hbnu.annotation;
 1
 2
    import org.junit.Test;
 3
   import org.springframework.context.ApplicationContext;
    org.springframework.context.support.ClassPathXmlApplicationContext;
 6
 7
 8
    * @author 陈迪凯
    * @date 2021-03-09 16:57
 9
10
11
    public class AnnotationTest {
12
13
        @Test
        public void testAnnotation() {
14
            ApplicationContext applicationContext = new
15
    ClassPathXmlApplicationContext("annotation.xml");
16
17
            UserService userService = (UserService)
    applicationContext.getBean("userService");
18
19
            userService.printUserService();
20
        }
    }
21
```

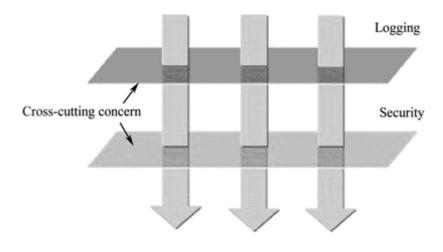
## 九、Spring中的AOP编程

## 1、AOP简介

## 1.1、AOP概述

AOP是软件设计领域中的面向切面编程,它是面向对象(OOP)编程的补充和完善,在实际项目中,我们可以将面向对象理解为一个静态过程(例如一个系统有哪些功能模块、一个模块有哪些对象,一个对象有哪些属性),而面向切面编程时一个一个的动态过程(比在对象运行时织入一些功能)。

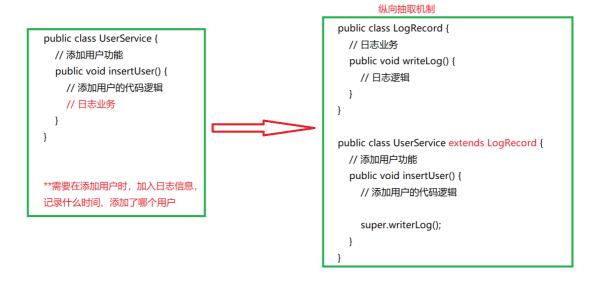
## 面向切面案例图:



XxxService

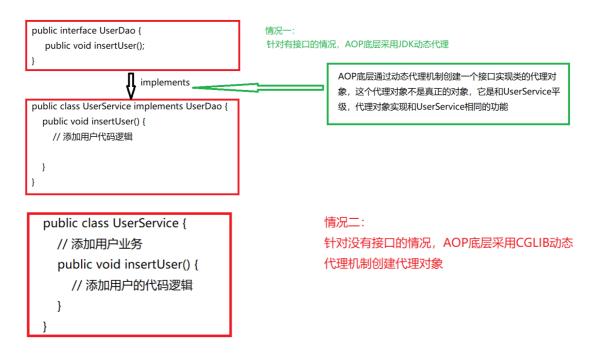
## 1.2、AOP演变过程

• 纵向机制



• 横向机制

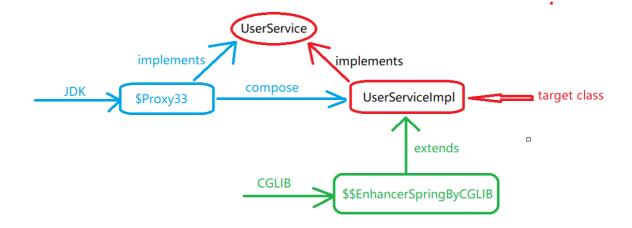
AOP: 横向抽取机制, 底层采用动态代理机制实现



// AOP底层创建一个UserServic子类的代理对象,未使用接口的情况,AOP底层采用CGLIB动态代理机制创建代理对象

## 1.3、AOP代理机制

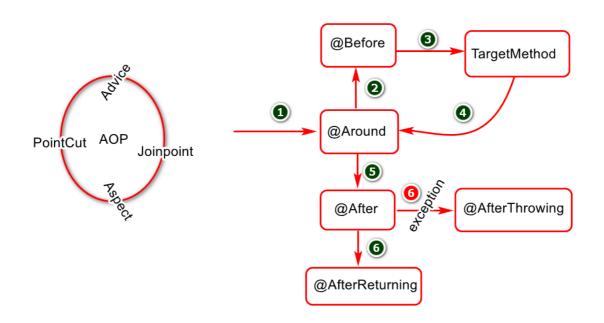
- 假如目标对象(被代理对象)实现了接口,AOP底层采用JKD动态代理机制为目标对象创建代理对象(目标类和代理类实现共同的接口)
- 假如目标对象(被代理对象)未实现接口,AOP底层采用CGLIB动态代理机制为目标对象创建代理 对象(默认创建的代理类会继承了目标对象类型)



## 2、AOP相关术语

- 切面(Aspect):横切面对象,一般为具体的类对象(可以借助@Aspect声明)
- 连接点 (JoinPoint): 程序执行过程中的某个特定的点,一般指被拦截到的方法,也可以理解为类 里面可以被增强的方法,这些方法就被称为连接点
- 切入点 (PointCut): 对连接点内容的一种定义,一般可以理解为多个连接点的结合,也可以理解为类里面实际被增强的方法。
- 通知 (Advice): 在切面的某个特定连接点上执行的动作 (扩展功能)
  - 。 前置通知 (Before): 在被增强方法之前执行
  - 。 返回通知 (AfterReturning): 在被增强方法之后执行
  - 。 异常通知 (AfterThrowing): 在被增强方法发生异常时执行
  - 。 后置通知(After): 在被增强方法之后执行,肯定会执行的,相当于try...catcht...finally中的 finally
  - 环绕通知 (Around) : 在被增强方法之前和之后都执行

## AOP通知执行过程:

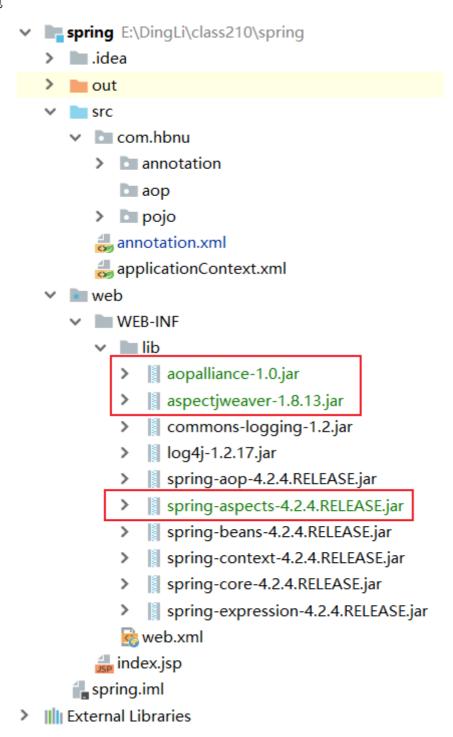


## 3、AOP编程基础

在Spring中要使用AOP编程,需要使用AspectJ实现,AspectJ不是我们Spring中的一部分,AspectJ是一个面向切面编程的框架,Spring要实现AOP编程,需要和AspectJ一起使用,使用AspectJ实现AOP有两种方式:一种是基于AspectJ的xml配置、一种是基于AspectJ的注解

## 3.1、基于AspectJ的xml配置

导入jar包



• Spring核心配置文件aopContext.xml

```
7
            http://www.springframework.org/schema/beans
 8
            http://www.springframework.org/schema/beans/spring-beans.xsd
 9
            http://www.springframework.org/schema/aop
10
            http://www.springframework.org/schema/aop/spring-aop.xsd
11
            http://www.springframework.org/schema/context
12
            http://www.springframework.org/schema/context/spring-
    context.xsd">
13
14
15
    </beans>
```

• 创建被增强类UserService

```
package com.hbnu.aop;
 1
2
    /**
 3
    * @author 陈迪凯
 4
 5
    * @date 2021-03-15 10:13
 6
    */
   public class UserService {
7
 8
        public void insertUser() {
9
            System.out.println("添加用户代码逻辑.....");
10
        }
11
12
   }
```

• 创建增强类LogRecord

```
1
    package com.hbnu.aop;
 2
 3
   import org.aspectj.lang.ProceedingJoinPoint;
 4
    /**
 5
 6
    * @author 陈迪凯
    * @date 2021-03-15 10:15
 7
 8
    */
9
    public class LogRecord {
10
        public void before1() {
11
12
            System.out.println("被增强方法之前执行");
13
        }
14
        public void around(ProceedingJoinPoint proceedingJoinPoint) throws
15
    Throwable {
16
            System.out.println("被增强方法之前执行.....");
17
18
            // 执行被增强方法
            proceedingJoinPoint.proceed();
19
20
            System.out.println("被增强方法之后执行.....");
21
        }
22
23
    }
```

• 修改核心配置文件

```
1 | <?xml version="1.0" encoding="utf-8" ?>
```

```
2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
           xmlns:aop="http://www.springframework.org/schema/aop"
           xmlns:context="http://www.springframework.org/schema/context"
 5
 6
           xsi:schemaLocation="
 7
            http://www.springframework.org/schema/beans
 8
            http://www.springframework.org/schema/beans/spring-beans.xsd
 9
            http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop.xsd
10
11
            http://www.springframework.org/schema/context
12
            http://www.springframework.org/schema/context/spring-
    context.xsd">
13
        <!-- 将增强类和被增强类交给Spring容器进行管理 -->
14
        <bean id="userService" class="com.hbnu.aop.UserService"></bean>
15
        <bean id="logRecord" class="com.hbnu.aop.LogRecord"></bean>
16
17
        <!-- 配置AOP -->
18
19
        <aop:config>
20
            <!-- 配置切入点 -->
            <aop:pointcut id="pointcut1" expression="execution(*)</pre>
21
    com..hbnu.aop.UserService.*(..))">
                                               </aop:pointcut>
22
            <!-- 配置切面 -->
23
24
            <aop:aspect ref="logRecord">
25
                <!-- 配置通知 -->
                <aop:before method="before1" pointcut-ref="pointcut1">
26
    </aop:before>
27
28
                <aop:around method="around" pointcut-ref="pointcut1">
    </aop:around>
29
            </aop:aspect>
30
        </aop:config>
31
32
    </beans>
```

```
1
    package com.hbnu.aop;
 2
 3
    import org.junit.Test;
    import org.springframework.context.ApplicationContext;
 4
    import
    org.springframework.context.support.ClassPathXmlApplicationContext;
 6
    /**
 7
     * @author 陈迪凯
 8
 9
    * @date 2021-03-15 10:25
10
    */
11
    public class AOPTest {
12
13
        @Test
14
        public void aopTest() {
            ApplicationContext applicationContext = new
15
    ClassPathXmlApplicationContext("aopContext.xml");
16
```

```
UserService userService = (UserService)
applicationContext.getBean("userService");

userService.insertUser();

}
```

## 3.2、切入点表达式增强

Spring通过切入点表达式定义具体的切入点,常用AOP切入点表达式定义及说明

| 指示符         | 描述                 |
|-------------|--------------------|
| bean        | 用于匹配指定bean id的方法执行 |
| within      | 用于配置指定包名下的类方法执行    |
| execution   | 用于进行细粒度的方法匹配执行     |
| @annotation | 用于对指定注解的方法进行匹配执行   |

## • bean表达式应用增强

bean表达式应用于类级别,实现粗粒度的切入点表达式

- o bean("userService"): 指一个类中的所有方法
- o bean("\*Service"): 指定所有以Service为后缀的类

说明: bean表达式内部的对象是又Spring容器管理的对象, bena里面的表达式是容器中对象的id

## • within表达式增强

within表达式应用于类级别,实现粗粒度的切入点表达式

- o within("com.hbnu.aop.UserService"): 指定类,只能指定一个类
- o within("com.hbnu.aop.\*"): 指定包名下的所有类
- o within("com.hbnu.aop..\*"): 指定包名下子目录下的所有类

## • execution表达式增强

execution表达式应用于方法级别,实现细粒度的切入点表达式

常用格式: execution(<访问修饰符>?<返回类型><方法名>(参数)<异常>)

- execution(\* com.hbnu.aop.UserService.insertUser(..))
- execution(\* com.hbnu.aop.UserService.\*(..))
- o execution(\* \*.\*(..))
- execution(\* com.hbnu.aop.\*Service.\*(..))

#### • @annotation表达式增强

@annotation表达式应用于方法级别,实现细粒度的切入点表达式

o @annotation(com.hbnu.RequestLog): 指定一个需要实现增强功能的方法

说明: RequestLog是我们自定义的一个注解

## 3.3、基于AspectJ的注解方式

• 创建被增强类

```
1
    package com.hbnu.aop;
 2
 3
 4
    * @author 陈迪凯
     * @date 2021-03-15 10:13
 5
 6
 7
    public class UserService {
        public void insertUser() {
 8
9
            System.out.println("添加用户代码逻辑.....");
10
        }
11
12
   }
```

## • 创建增强类

```
package com.hbnu.aop;
 1
 2
 3
    import org.aspectj.lang.ProceedingJoinPoint;
 4
 5
    /**
 6
    * @author 陈迪凯
    * @date 2021-03-15 10:15
7
 8
9
    public class LogRecord {
10
11
        public void before1() {
            System.out.println("被增强方法之前执行");
12
13
14
15
        public void around(ProceedingJoinPoint proceedingJoinPoint) throws
    Throwable {
16
           System.out.println("被增强方法之前执行.....");
17
18
            // 执行被增强方法
19
            proceedingJoinPoint.proceed();
20
21
           System.out.println("被增强方法之后执行.....");
22
        }
23
    }
```

## • 修改配置文件

```
<?xml version="1.0" encoding="utf-8" ?>
 1
 2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
           xmlns:aop="http://www.springframework.org/schema/aop"
           xmlns:context="http://www.springframework.org/schema/context"
 5
           xsi:schemaLocation="
 6
 7
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
 8
 9
            http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop.xsd
10
11
            http://www.springframework.org/schema/context
12
            http://www.springframework.org/schema/context/spring-
    context.xsd">
13
```

```
14
        <!-- 将增强类和被增强类交给Spring容器进行管理 -->
15
        <bean id="userService" class="com.hbnu.aop.UserService"></bean>
16
        <bean id="logRecord" class="com.hbnu.aop.LogRecord"></bean>
17
18
19
        <!&ndash; 配置AOP &ndash;&gt;
20
        <aop:config>
21
            <!&ndash; 配置切入点 &ndash;&gt;
22
            <aop:pointcut id="pointcut1" expression="execution(*)</pre>
    com.hbnu.aop.UserService.*(..))"></aop:pointcut>
23
24
            <!&ndash; 配置切面 &ndash;&gt;
25
            <aop:aspect ref="logRecord">
26
                <!&ndash; 配置通知 &ndash;&gt;
27
                <aop:before method="before1" pointcut-ref="pointcut1">
    </aop:before>
28
29
                <aop:around method="around" pointcut-ref="pointcut1">
    </aop:around>
30
            </aop:aspect>
31
        </aop:config>
32
        -->
33
34
        <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
35
36
    </beans>
```

#### • 修改增强类

```
1
    package com.hbnu.aop;
 2
    import org.aspectj.lang.ProceedingJoinPoint;
 3
    import org.aspectj.lang.annotation.Around;
    import org.aspectj.lang.annotation.Before;
 5
6
 7
    /**
 8
    * @author 陈迪凯
    * @date 2021-03-15 10:15
 9
    */
10
11
    @Aspect
12
    public class LogRecord {
13
        @Before("execution(* com.hbnu.aop.UserService.*(..))")
14
15
        public void before1() {
16
            System.out.println("被增强方法之前执行");
17
        }
18
        @Around("execution(* com.hbnu.aop.UserService.*(..))")
19
20
        public void around(ProceedingJoinPoint proceedingJoinPoint) throws
    Throwable {
21
            System.out.println("被增强方法之前执行.....");
22
23
            // 执行被增强方法
            proceedingJoinPoint.proceed();
24
25
26
            System.out.println("被增强方法之后执行.....");
27
        }
```

```
28 }
```

```
package com.hbnu.aop;
 2
 3
   import org.junit.Test;
   import org.springframework.context.ApplicationContext;
    import
    org.spring framework.context.support.Class {\tt PathXmlApplicationContext};\\
 6
   /**
7
    * @author 陈迪凯
8
    * @date 2021-03-15 10:25
9
10
   public class AOPTest {
11
12
13
        @Test
14
        public void aopTest() {
15
            ApplicationContext applicationContext = new
    ClassPathXmlApplicationContext("aopContext.xml");
16
17
            UserService userService = (UserService)
    applicationContext.getBean("userService");
18
            userService.insertUser();
19
20
        }
21
    }
```