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

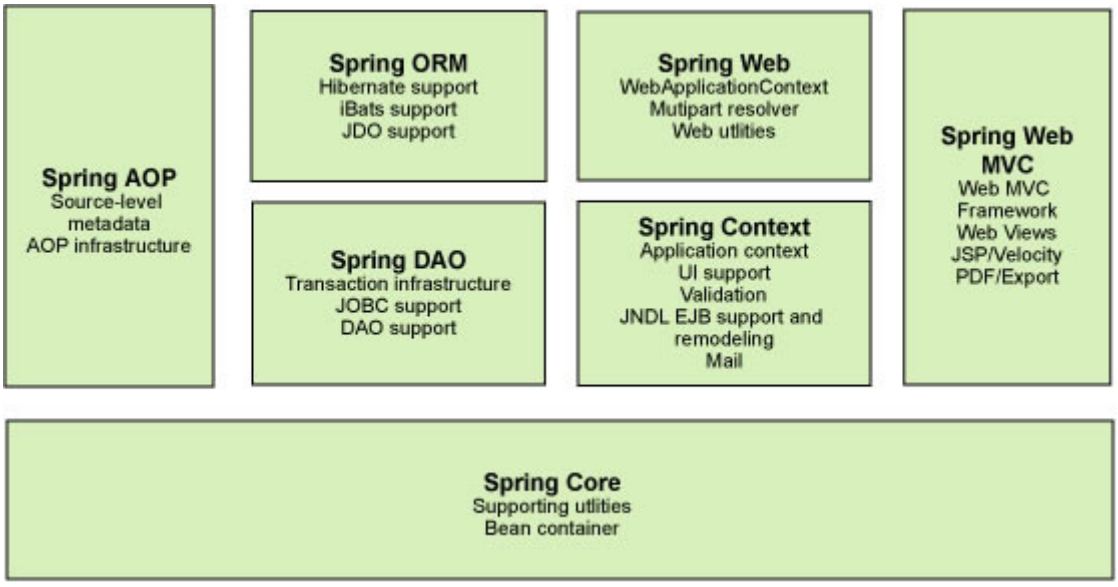
二、Spring框架优势

- 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：



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：

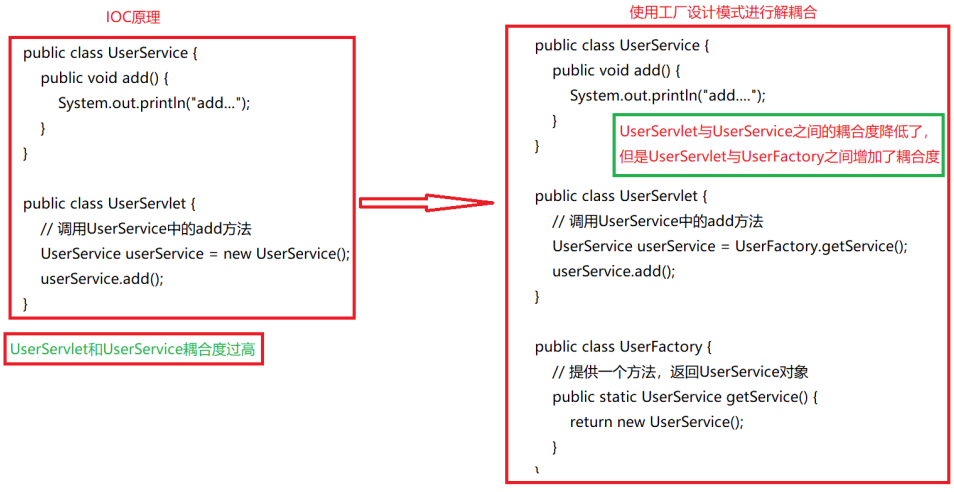
模块	说明
核心容器 Spring Core	核心容器，提供Spring框架的基本功能。核心容器的主要组件是BeanFactory，它是工厂模式的实现。BeanFactory 使用控制反转（IOC）模式，将应用程序的配置和依赖性规范与实际的应用程序代码分开。
Spring Context	Spring上下文，是一个配置文件，向 Spring 框架提供上下文信息。Spring 上下文包括企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。
Spring AOP	通过配置管理特性，Spring AOP 模块直接将面向切面的编程功能集成到了 Spring 框架中。可以很容易地使 Spring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 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 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 Map。所有这些都遵从 Spring 的通用事务和 DAO 异常层次结构
Spring 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 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 MVC框架	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、dom4j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

IOC原理

```
public class UserService {
    public void add() {
        System.out.pritnt("add....");
    }
}

public class UserServlet {
    // 生成UserService对象
}
```

第一步，创建xml配置文件，配置需要生成对象的类的信息

```
<bean id = "userService" class = "com.hbnu.pojo.UserService" />
```

第二步，创建工厂类，使用dom4j解析xml配置文件，再通过反射创建对象

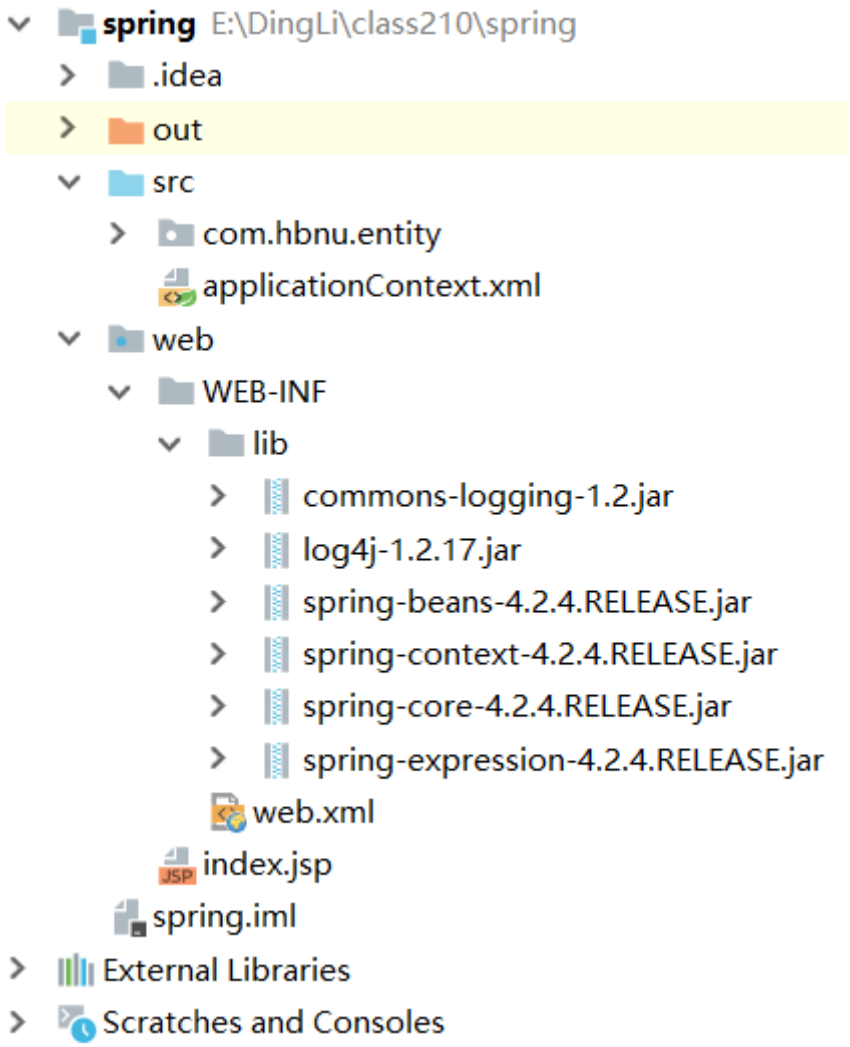
```
public class UserFactory {
    // 返回UserService对象
    public static UserService getService {
        // 使用dom4j技术解析xml文件，根据id值获取class属性值
        String classValue = "class属性值";

        // 通过反射获取对象
        Class clazz = Class.forName(classValue);

        // 通过字节码对象，创建UserService对象
        UserService userService = clazz.newInstance();

        return userService;
    }
}
```

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：



2. 创建普通类User.java

```java
package com.hbnu.entity;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:03
 */
public class User {
    public void add() {
        System.out.println("IOC test......");
    }
}
```

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

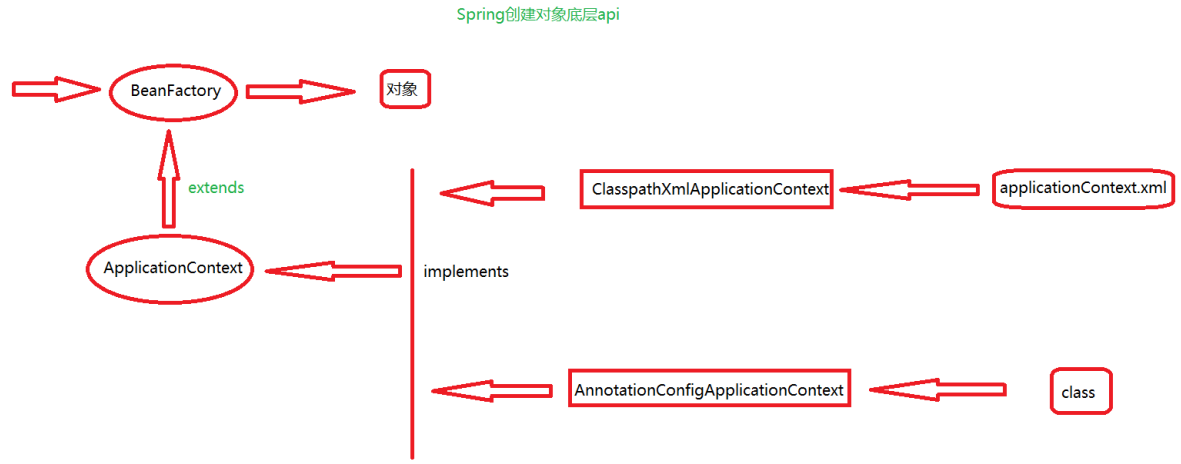
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">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

4. 测试IOC

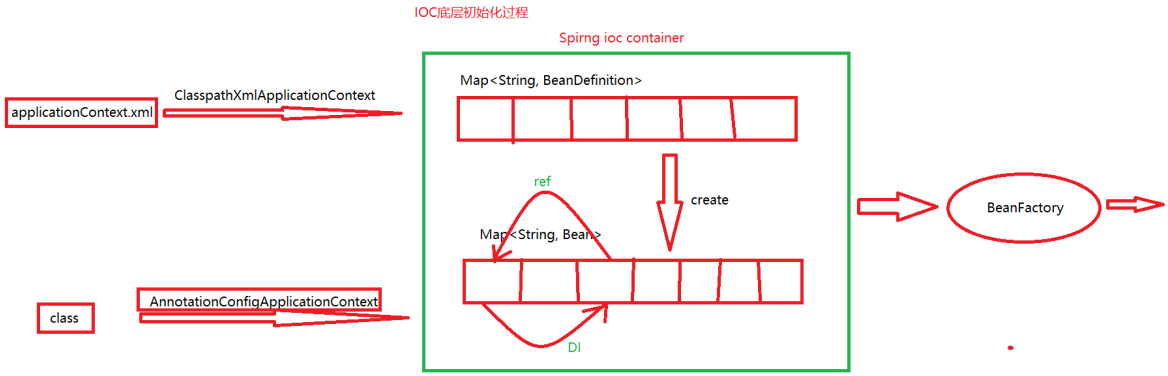
```java
package com.hbnu.entity;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:11
 */
public class IOCTest {

    @Test
    public void userTest() {
        // 1、加载spring核心配置文件
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");

        /// 2、获取bean对象
        User user = (User) applicationContext.getBean("user");

        user.add();
    }
```

```
}
```

4、IOC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（重点）、通过静态工厂（了解）、通过实例工厂（了解）

1、通过无参构造函数

- 创建User类

```java
package com.hbnu.entity;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:03
 */
public class User {
    public void add() {
        System.out.println("IOC test......");
    }
}
```

- 配置文件

```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"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- 配置bean
    id属性：必须唯一
    class属性：需要交给spring管理的类的全路径
    -->
    <bean id="user" class="com.hbnu.entity.User"/>
</beans>
```

2、通过静态工厂创建Bean对象

- 创建工厂

```java
package com.hbnu.entity;

/**
 * @author 陈迪凯
 * @date 2021-03-08 8:14
 */
public class UserFactory {

    public static User getUser() {
        return new User();
    }
}
```

- 配置文件

```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"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- 配置bean
    id属性：必须唯一
    class属性：需要交给spring管理的类的全路径
    -->
    <!--<bean id="user" class="com.hbnu.entity.User"/>-->

    <!-- 通过静态工厂管理Bean对象 -->
    <bean id="factory" class="com.hbnu.entity.UserFactory" factory-
method="getUser"></bean>
</beans>
```

- 测试

```java
package com.hbnu.entity;

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

```java
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:11
 */
public class IOCTest {

    @Test
    public void userTest() {
        // 1、加载spring核心配置文件
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

        /// 2、获取bean对象
        User user = (User) applicationContext.getBean("factory");

        user.add();
    }
}
```

3、通过实例工厂创建Bean对象

- 创建工厂类

```java
package com.hbnu.entity;

/**
 * @author 陈迪凯
 * @date 2021-03-08 8:14
 */
public class UserFactory {

    public User getUser() {
        return new User();
    }
}
```

- 配置文件

```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"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- 配置bean
    id属性：必须唯一
    class属性：需要交给spring管理的类的全路径
    -->
    <!--<bean id="user" class="com.hbnu.entity.User"/>-->

    <!-- 通过静态工厂管理Bean对象 -->
    <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-method="getUser"></bean>-->

```

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

- 测试

```
package com.hbnu.entity;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:11
 */
public class IOCTest {

    @Test
    public void userTest() {
        // 1、加载spring核心配置文件
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");

        /// 2、获取bean对象
        User user = (User) applicationContext.getBean("user");

        user.add();
    }
}
```

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-arg	<bean>元素的子元素，可以使用此元素传入构造参数进行实例化。该元素的 index 属性指定构造参数的序号（从 0 开始），type 属性指定构造参数的类型
property	<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	<property> 和 <constructor-arg> 等元素的子元素，该元素中的 bean 属性用于指定对 Bean 工厂中某个 Bean 实例的引用
value	<property> 和 <constractor-arg> 等元素的子元素，用于直接指定一个常量值
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

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（重点）、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 UserInterface {
    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

```java
package com.hbnu.entity;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:03
 */
public class User {
    private String username;

    public User(String username) {
        this.username = username;
    }

    public void add() {
        System.out.println("add......" + username);
    }
}
```

- 配置文件

```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"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- 配置bean
    id属性：必须唯一
    class属性：需要交给spring管理的类的全路径
```

```
    -->
    <!--<bean id="user" class="com.hbnu.entity.User"/>-->

    <!-- 通过静态工厂管理Bean对象 -->
    <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-method="getUser"></bean>-->

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

    <!-- 属性注入 -->
    <!-- 通过有参构造函数注入属性 -->
    <bean id="user" class="com.hbnu.entity.User">
        <constructor-arg name="username" value="chendikai"></constructor-arg>
    </bean>
</beans>
```

- 测试

```
package com.hbnu.entity;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:11
 */
public class IOCTest {

    @Test
    public void userTest() {
        // 1、加载spring核心配置文件
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

        // 2、获取bean对象
        User user = (User) applicationContext.getBean("user");

        user.add();
    }
}
```

2、通过set方法注入属性

- 创建User类

```
package com.hbnu.entity;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:03
```

```
 */
public class User {
    private String username;

    public void setUsername(String username) {
        this.username = username;
    }

    public void add() {
        System.out.println("add......" + username);
    }
}
```

- 配置文件

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

    <!-- 通过静态工厂管理Bean对象 -->
    <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-method="getUser"></bean>-->

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

    <!-- 属性注入 -->
    <!-- 通过有参构造函数注入属性 -->
    <!--
    <bean id="user" class="com.hbnu.entity.User">
        &lt;!&ndash;
        name属性：类中的属性名
        value属性：注入的值
        &ndash;&gt;
        <constructor-arg name="username" value="chendikai">
</constructor-arg>
    </bean>
    -->
    <!-- 通过set方法注入属性 -->
    <bean id="user" class="com.hbnu.entity.User">
        <property name="username" value="zhangsanfeng"></property>
    </bean>
</beans>
```

- 测试

```java
package com.hbnu.entity;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:11
 */
public class IOCTest {

    @Test
    public void userTest() {
        // 1、加载spring核心配置文件
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");

        // 2、获取bean对象
        User user = (User) applicationContext.getBean("user");

        user.add();
    }
}
```

3、注入对象属性

- 创建UserDao类

```java
package com.hbnu.entity;

/**
 * @author 陈迪凯
 * @date 2021-03-08 9:31
 */
public class UserDao {
    public void printUserDao() {
        System.out.println("UserDao......");
    }
}
```

- 创建UserService类

```java
package com.hbnu.entity;

/**
 * @author 陈迪凯
 * @date 2021-03-08 9:33
 */
public class UserService {
    private UserDao userDao;

    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }

```

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

- 配置文件

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

    <!-- 通过静态工厂管理Bean对象 -->
    <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-method="getUser"></bean>-->

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

    <!-- 属性注入 -->
    <!-- 通过有参构造函数注入属性 -->
    <!--
    <bean id="user" class="com.hbnu.entity.User">
        &lt;!&ndash;
        name属性：类中的属性名
        value属性：注入的值
        &ndash;&gt;
        <constructor-arg name="username" value="chendikai"></constructor-arg>
    </bean>
    -->
    <!-- 通过set方法注入属性 -->
    <!--<bean id="user" class="com.hbnu.entity.User">
        <property name="username" value="zhangsanfeng"></property>
    </bean>-->

    <bean id="user" class="com.hbnu.entity.UserDao"></bean>

    <bean id="userService" class="com.hbnu.entity.UserService">
        <property name="userDao" ref="user"></property>
    </bean>
</beans>
```

- 测试

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

```
2
3  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  */
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         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

```
1  package com.hbnu.entity;
2
3  import java.util.List;
4  import java.util.Map;
5  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
25     public List<String> getList() {
26         return list;
```

```
    }

    public void setList(List<String> list) {
        this.list = list;
    }

    public Map<String, String> getMap() {
        return map;
    }

    public void setMap(Map<String, String> map) {
        this.map = map;
    }

    public Properties getProperties() {
        return properties;
    }

    public void setProperties(Properties properties) {
        this.properties = properties;
    }

    public void test() {
        System.out.println("arr:" + arr);
        System.out.println("list:" + list);
        System.out.println("map:" + map);
        System.out.println("properties:" + properties);
    }
}
```

- 配置文件

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

    <!-- 通过静态工厂管理Bean对象 -->
    <!--<bean id="factory" class="com.hbnu.entity.UserFactory" factory-method="getUser"></bean>-->

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

    <!-- 属性注入 -->
    <!-- 通过有参构造函数注入属性 -->
    <!--
    <bean id="user" class="com.hbnu.entity.User">
```

```
        &lt;!&ndash;
        name属性：类中的属性名
        value属性：注入的值
        &ndash;&gt;
        <constructor-arg name="username" value="chendikai">
</constructor-arg>
    </bean>
    -->
    <!-- 通过set方法注入属性 -->
    <!--<bean id="user" class="com.hbnu.entity.User">
        <property name="username" value="zhangsanfeng"></property>
    </bean>-->

    <!--<bean id="user" class="com.hbnu.entity.UserDao"></bean>

    <bean id="userService" class="com.hbnu.entity.UserService">
        <property name="userDao" ref="user"></property>
    </bean>-->

    <bean id="dataType" class="com.hbnu.entity.DataType">
        <!-- 1、数组类型 -->
        <property name="arr">
            <list>
                <value>铠</value>
                <value>妲己</value>
                <value>小乔</value>
            </list>
        </property>

        <!-- 2、list集合类型 -->
        <property name="list">
            <list>
                <value>张三丰</value>
                <value>张翠山</value>
                <value>张无忌</value>
            </list>
        </property>

        <!-- 3、map集合类型 -->
        <property name="map">
            <map>
                <entry key="name" value="陈迪凯"></entry>
                <entry key="gender" value="男"></entry>
                <entry key="address" value="湖北黄石"></entry>
            </map>
        </property>

        <!-- 4、Properties类型 -->
        <property name="properties">
            <props>
                <prop key="driverClass">com.mysql.cj.jdbc.Driver</prop>
                <prop key="url">jdbc:mysql:///hbnu</prop>
                <prop key="username">root</prop>
                <prop key="password">chendikai</prop>
            </props>
        </property>
    </bean>
</beans>
```

- 测试

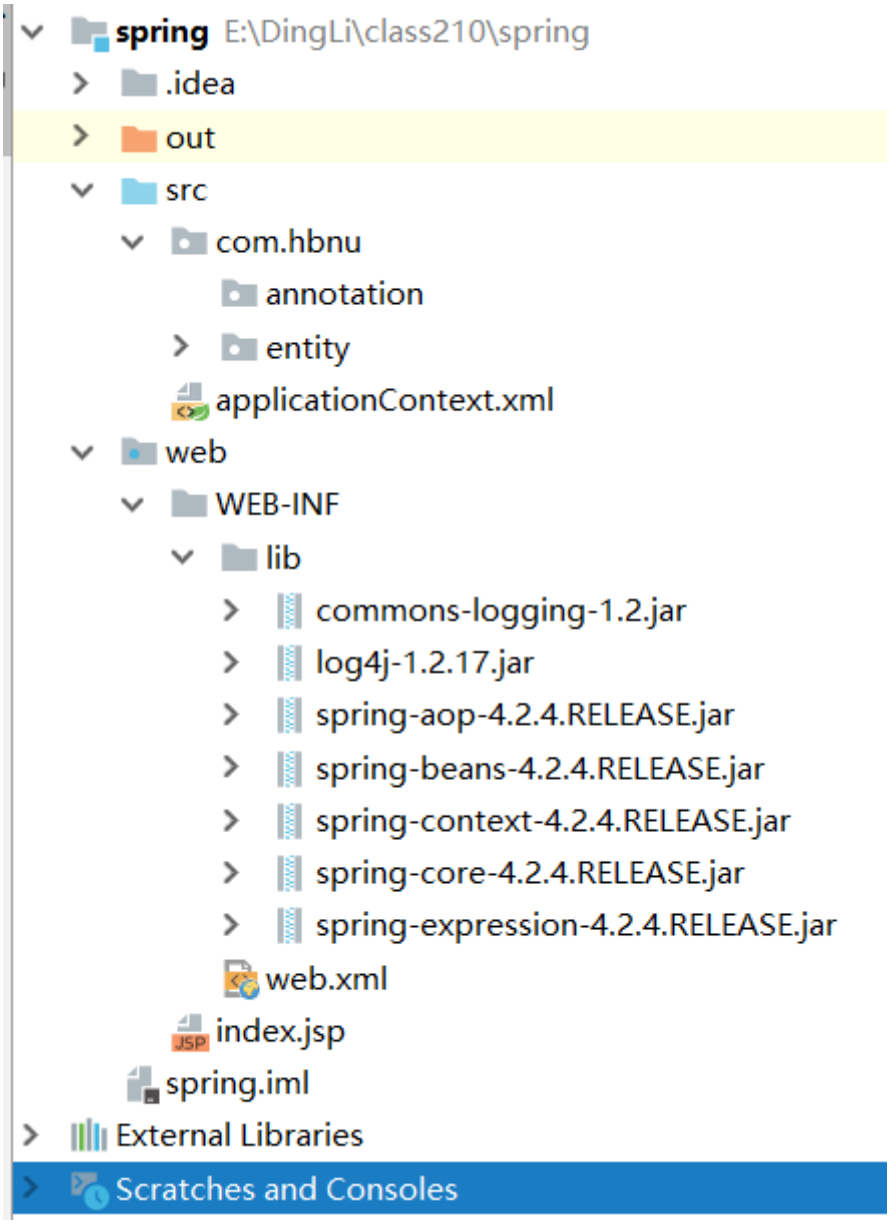
```
package com.hbnu.entity;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-01 10:11
 */
public class IOCTest {

    @Test
    public void userTest() {
        // 1、加载spring核心配置文件
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");

        // 2、获取bean对象
        DataType dataType = (DataType)
applicationContext.getBean("dataType");

        dataType.test();
    }
}
```

八、Spring中注解开发

1、注解开发案例

- 项目结构

  导入aop的jar包



- 配置文件

```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:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/beans/spring-aop.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 开启注解扫描 -->
    <context:component-scan base-package="com.hbnu.annotation">
</context:component-scan>
</beans>
```

- 创建User类

```java
package com.hbnu.annotation;

import org.springframework.stereotype.Component;

/**
 * @author 陈迪凯
 * @date 2021-03-08 10:45
 */
@Component(value = "user")
public class User {
    public void add() {
        System.out.println("annotation....add...");
    }
}
```

- 测试

```java
package com.hbnu.annotation;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-08 10:51
 */
public class AnnotationTest {

    @Test
    public void annoTest() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("annotation.xml");

        User user = (User) applicationContext.getBean("user");

        user.add();
    }
}
```

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

```java
package com.hbnu.annotation;

import org.springframework.stereotype.Component;

import javax.xml.crypto.KeySelector;

/**
 * @author 陈迪凯
 * @date 2021-03-08 11:05
```

```
 */
@Component(value = "userDao")
public class UserDao {
    public void add() {
        System.out.println("userdao....add...");
    }
}
```

- 创建UserService

```
package com.hbnu.annotation;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

/**
 * @author 陈迪凯
 * @date 2021-03-08 11:06
 */
@Component(value = "userService")
public class UserService {

    // @Autowired
    @Resource(name = "userDao")
    private UserDao userDao;

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

- 配置文件

```
<?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:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/beans/spring-aop.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 开启注解扫描 -->
    <context:component-scan base-package="com.hbnu.annotation">
</context:component-scan>
</beans>
```

- 测试

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

```
2
3  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 2021-03-08 10:51
10  */
11 public class AnnotationTest {
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

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

- 创建一个UserService

```
1  package com.hbnu.annotation;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Component;
5
6  import javax.annotation.Resource;
7
8  /**
9   * @author 陈迪凯
10  * @date 2021-03-09 17:21
11  */
12
```

```java
public class UserService {

    private UserDao userDao;

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

- 配置spring核心配置文件

```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:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 配置包扫描注解 -->
    <context:component-scan base-package="com.hbnu.annotation"></context:component-scan>

    <!-- 配置bean对象 -->
    <bean id="userDao" class="com.hbnu.annotation.UserDao"></bean>
    <bean id="userService" class="com.hbnu.annotation.UserService"></bean>
</beans>
```

- 修改UserService

```java
package com.hbnu.annotation;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

/**
 * @author 陈迪凯
 * @date 2021-03-09 17:21
 */
public class UserService {

    @Resource(name = "userDao")
    private UserDao userDao;

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

```
        userDao.printUserDao();
    }
}
```

- 测试

```
package com.hbnu.annotation;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-09 16:57
 */
public class AnnotationTest {

    @Test
    public void testAnnotation() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("annotation.xml");

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

        userService.printUserService();
    }
}
```

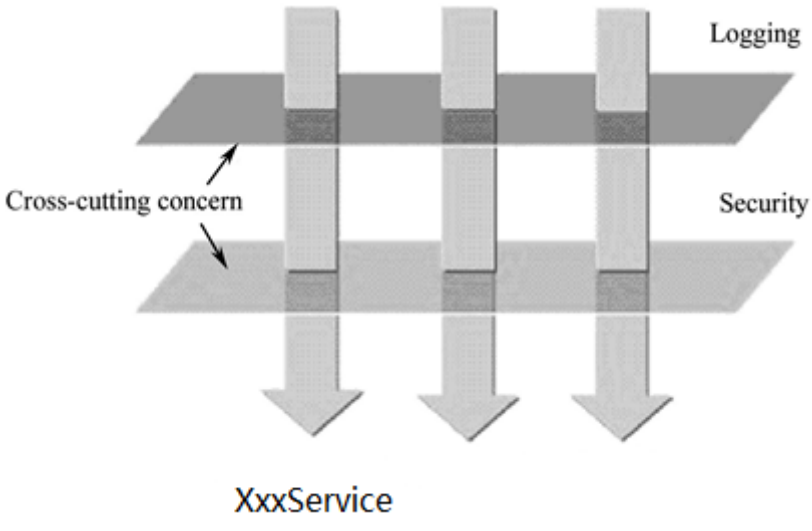
九、Spring中的AOP编程

1、AOP简介

1.1、AOP概述

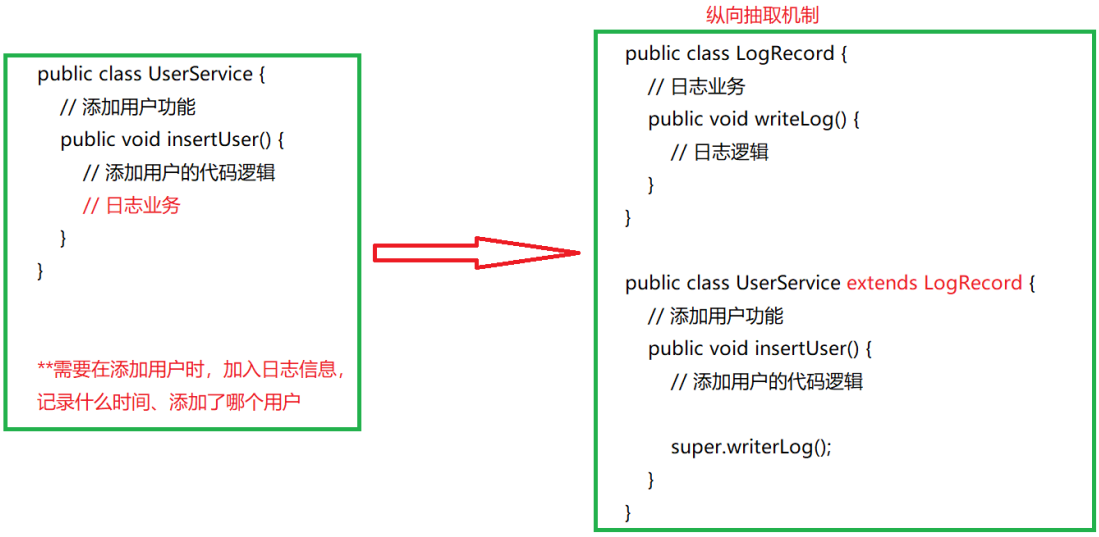
AOP是软件设计领域中的面向切面编程，它是面向对象（OOP）编程的补充和完善，在实际项目中，我们可以将面向对象理解为一个静态过程（例如一个系统有哪些功能模块、一个模块有哪些对象，一个对象有哪些属性），而面向切面编程时一个一个的动态过程（比在对象运行时织入一些功能）。

面向切面案例图：

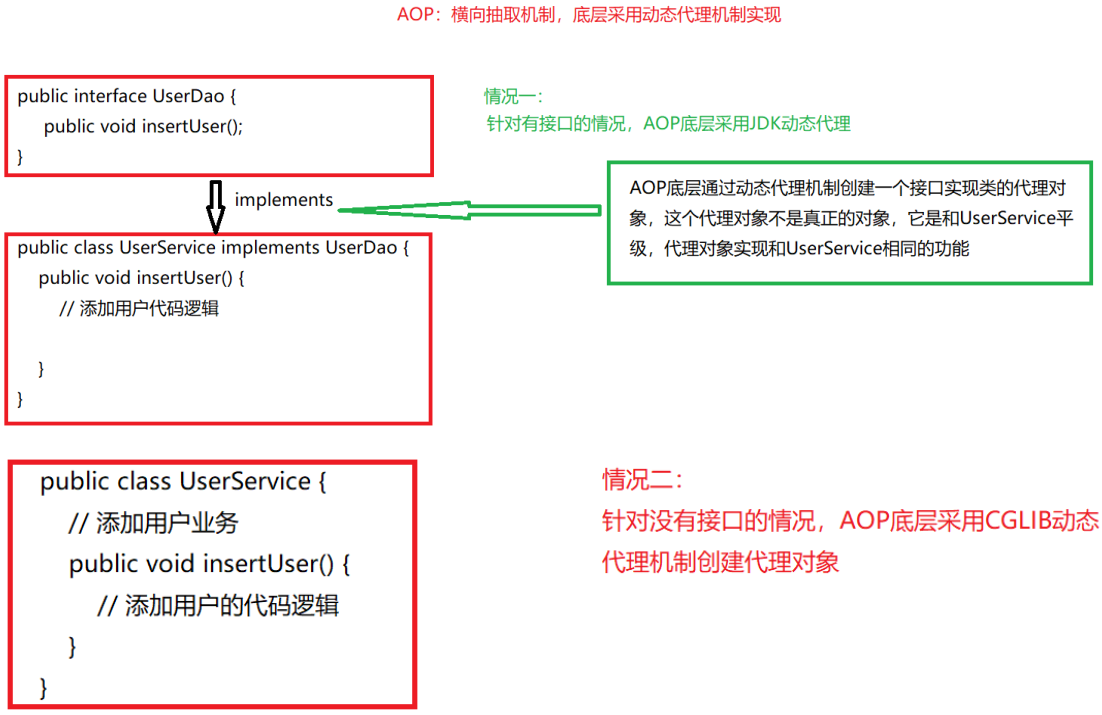


1.2、AOP演变过程

- 纵向机制



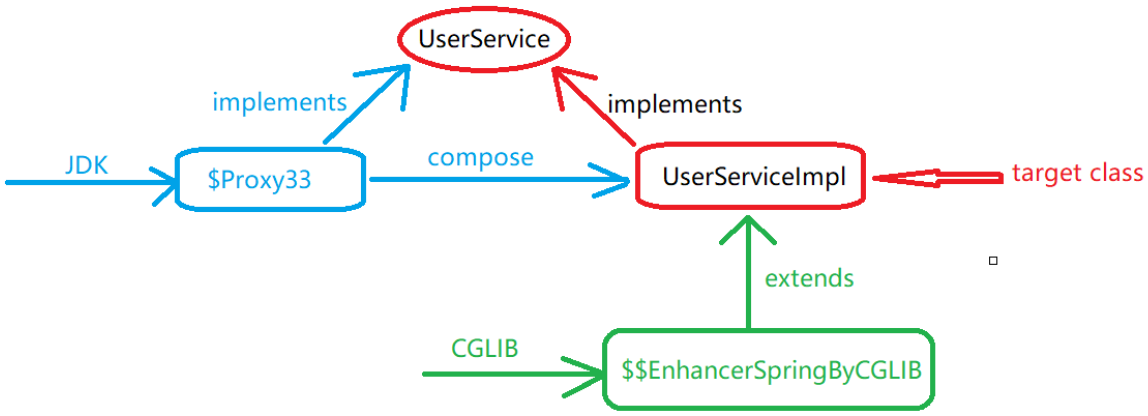
- 横向机制



// 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

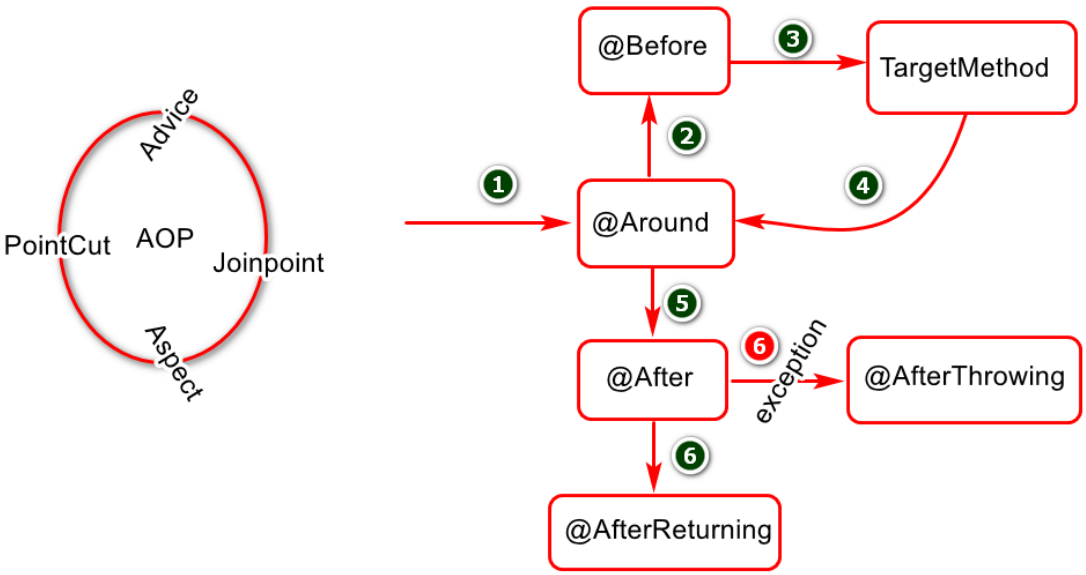
- 假如目标对象（被代理对象）实现了接口，AOP底层采用JKD动态代理机制为目标对象创建代理对象（目标类和代理类实现共同的接口）
- 假如目标对象（被代理对象）未实现接口，AOP底层采用CGLIB动态代理机制为目标对象创建代理对象（默认创建的代理类会继承了目标对象类型）



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：

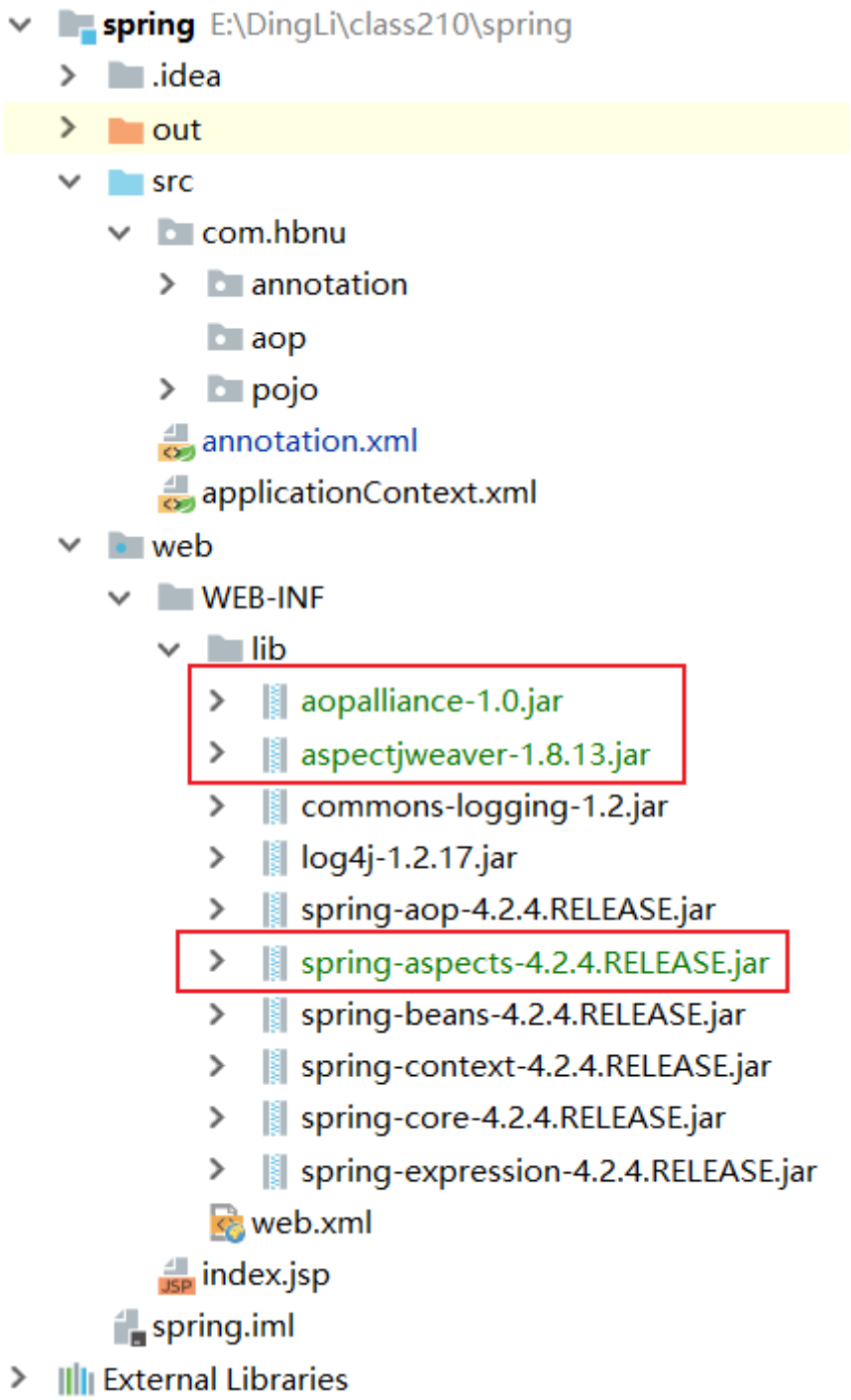


3、AOP编程基础

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

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

```
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">


</beans>
```

- 创建被增强类UserService

```
package com.hbnu.aop;

/**
 * @author 陈迪凯
 * @date 2021-03-15 10:13
 */
public class UserService {
    public void insertUser() {
        System.out.println("添加用户代码逻辑......");
    }

}
```

- 创建增强类LogRecord

```
package com.hbnu.aop;

import org.aspectj.lang.ProceedingJoinPoint;

/**
 * @author 陈迪凯
 * @date 2021-03-15 10:15
 */
public class LogRecord {

    public void before1() {
        System.out.println("被增强方法之前执行");
    }

    public void around(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
        System.out.println("被增强方法之前执行......");

        // 执行被增强方法
        proceedingJoinPoint.proceed();

        System.out.println("被增强方法之后执行......");
    }
}
```

- 修改核心配置文件

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

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 将增强类和被增强类交给Spring容器进行管理 -->
    <bean id="userService" class="com.hbnu.aop.UserService"></bean>
    <bean id="logRecord" class="com.hbnu.aop.LogRecord"></bean>

    <!-- 配置AOP -->
    <aop:config>
        <!-- 配置切入点 -->
        <aop:pointcut id="pointcut1" expression="execution(* com..hbnu.aop.UserService.*(..))">         </aop:pointcut>

        <!-- 配置切面 -->
        <aop:aspect ref="logRecord">
            <!-- 配置通知 -->
            <aop:before method="before1" pointcut-ref="pointcut1"></aop:before>

            <aop:around method="around" pointcut-ref="pointcut1"></aop:around>
        </aop:aspect>
    </aop:config>

</beans>
```

- 测试

```java
package com.hbnu.aop;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-15 10:25
 */
public class AOPTest {

    @Test
    public void aopTest() {
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext("aopContext.xml");

```

```
        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

  - bean("userService")：指一个类中的所有方法
  - 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

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

  - @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

```java
package com.hbnu.aop;

/**
 * @author 陈迪凯
 * @date 2021-03-15 10:13
 */
public class UserService {
    public void insertUser() {
        System.out.println("添加用户代码逻辑......");
    }

}
```

- 创建增强类

```java
package com.hbnu.aop;

import org.aspectj.lang.ProceedingJoinPoint;

/**
 * @author 陈迪凯
 * @date 2021-03-15 10:15
 */
public class LogRecord {

    public void before1() {
        System.out.println("被增强方法之前执行");
    }

    public void around(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
        System.out.println("被增强方法之前执行......");

        // 执行被增强方法
        proceedingJoinPoint.proceed();

        System.out.println("被增强方法之后执行......");
    }
}
```

- 修改配置文件

```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:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

```

```
    <!-- 将增强类和被增强类交给Spring容器进行管理 -->
    <bean id="userService" class="com.hbnu.aop.UserService"></bean>
    <bean id="logRecord" class="com.hbnu.aop.LogRecord"></bean>

    <!--
    &lt;!&ndash; 配置AOP &ndash;&gt;
    <aop:config>
        &lt;!&ndash; 配置切入点 &ndash;&gt;
        <aop:pointcut id="pointcut1" expression="execution(*
com.hbnu.aop.UserService.*(..))"></aop:pointcut>

        &lt;!&ndash; 配置切面 &ndash;&gt;
        <aop:aspect ref="logRecord">
            &lt;!&ndash; 配置通知 &ndash;&gt;
            <aop:before method="before1" pointcut-ref="pointcut1">
</aop:before>

            <aop:around method="around" pointcut-ref="pointcut1">
</aop:around>
        </aop:aspect>
    </aop:config>
    -->

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

</beans>
```

- 修改增强类

```
package com.hbnu.aop;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Before;

/**
 * @author 陈迪凯
 * @date 2021-03-15 10:15
 */
@Aspect
public class LogRecord {

    @Before("execution(* com.hbnu.aop.UserService.*(..))")
    public void before1() {
        System.out.println("被增强方法之前执行");
    }

    @Around("execution(* com.hbnu.aop.UserService.*(..))")
    public void around(ProceedingJoinPoint proceedingJoinPoint) throws
Throwable {
        System.out.println("被增强方法之前执行......");

        // 执行被增强方法
        proceedingJoinPoint.proceed();

        System.out.println("被增强方法之后执行......");
    }
```

```
}
```

- 测试

```
package com.hbnu.aop;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-15 10:25
 */
public class AOPTest {

    @Test
    public void aopTest() {
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("aopContext.xml");

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

        userService.insertUser();
    }
}
```