J2EE高级开发框架

一、Spring框架简介

1、Sping框架概述

- Spring框架是一个免费的、开源的、轻量级的应用程序开发框架，其目的是为了简化企业级应用程序的开发，降低开发
- Spring框架提供了AOP和IOC应用，可以使组件之间的耦合度降至最低（解耦合），其目的使为了系统日后维护和升级
- Spring提供了一整套的解决方案，开发者除了可以使用其自身的功能外，还可以整合第三方框架和技术进行联合使用，开发中可以自由选择哪种技术进行实现

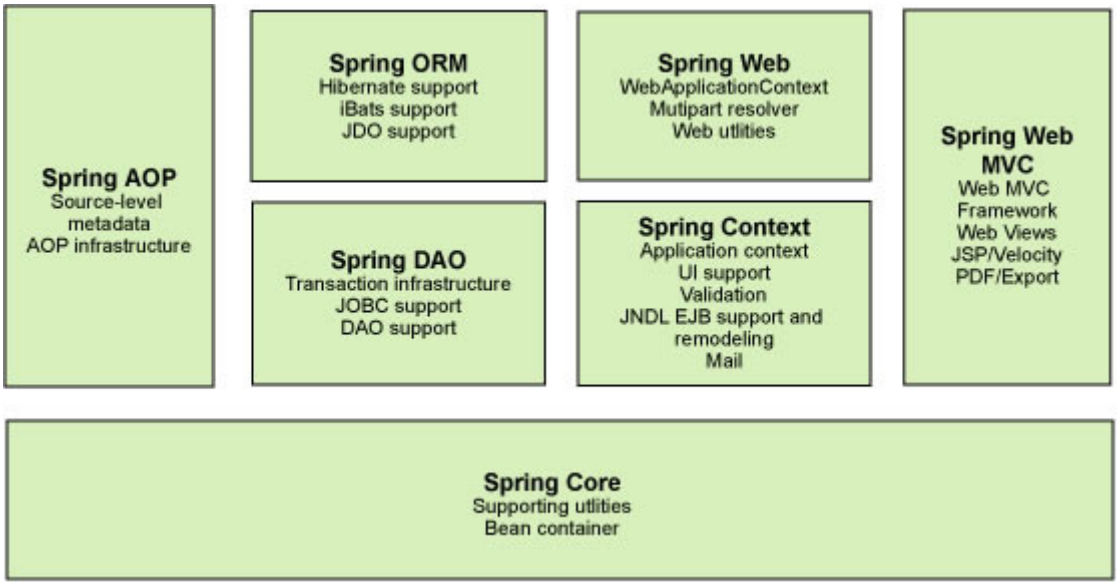
2、Spring优势

- 方便解耦，简化开发
- AOP支持
- 声明式事务的支持
- 降低J2EE中api的使用难度

用一句话概括：Spring的本质使管理软件中的对象，即对象的创建和维护对象之间的关系

二、Spring框架的架构

Spring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模块构件中核心容器之上，核心容器包含了创建、存储以及管理bean的方式，Spring架构图如下：



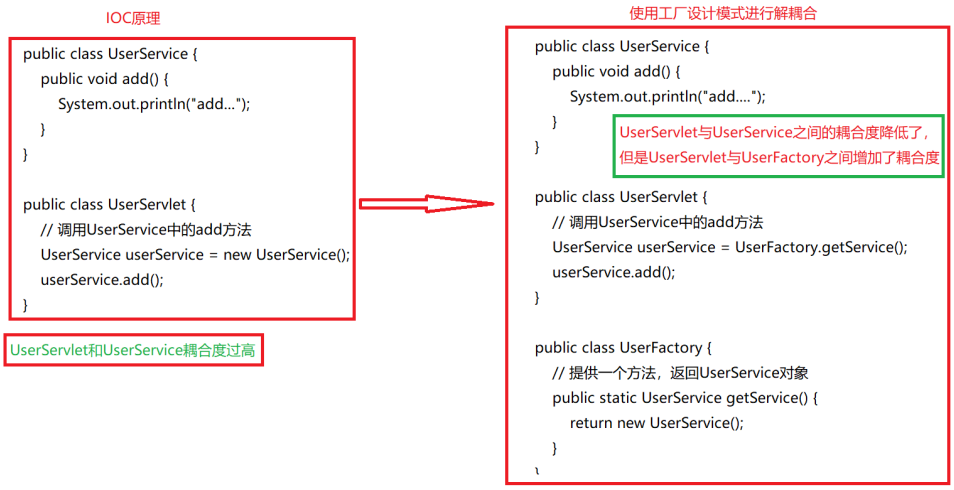
说明：Spring框架模块（组件）可以单独存在和使用，也可以和其他一个或多个模块联合使用。各个模块的功能如下：

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。

三、控制反转（IOC）

传统方式创建对象需要通过关键字new进行创建，Spring框架中，对象的创建不再使用new关键字，而是把对象的创建、存储和管理交给Spring容器，IOC可以实现这个功能，IOC通过配置文件或者是通过注解可以实现对象的创建和管理，IOC底层使用到的技术：xml配置文件、dom4j解析xml配置文件、工厂设计模式、反射。

1、IOC底层原理



上述发展中还存在耦合度问题

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;
    }
}
```

2、IOC入门案例

1. **下载Spring框架**

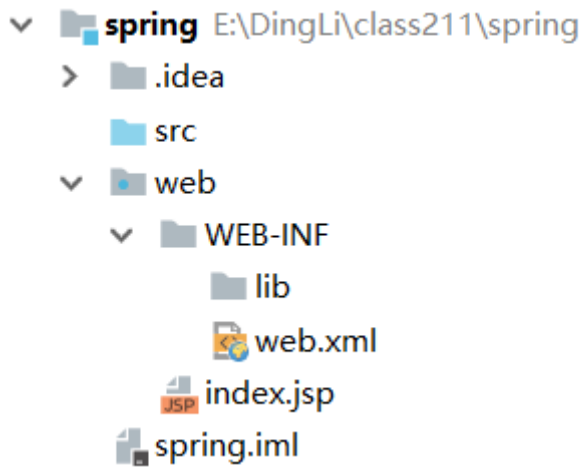
   Spring框架下载

   下载后解压目录如下：

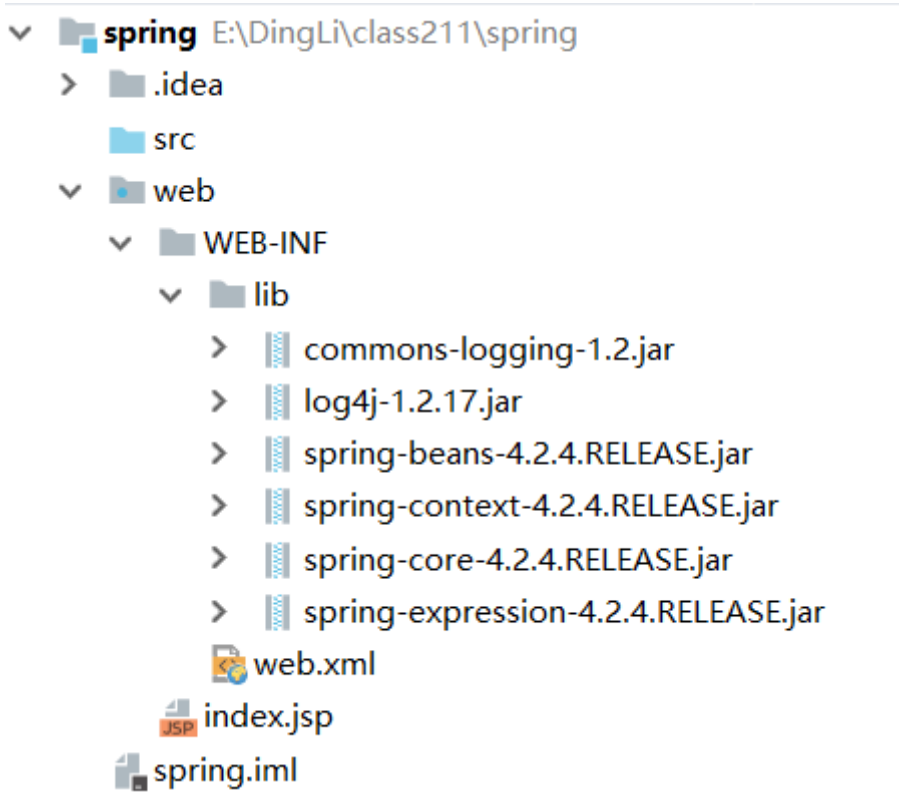
docs	2021-02-28 22:15	文件夹	
libs	2021-02-28 22:15	文件夹	
schema	2021-02-28 22:15	文件夹	
license.txt	2015-12-17 0:43	文本文档	15 KB
notice.txt	2015-12-17 0:43	文本文档	1 KB
readme.txt	2015-12-17 0:43	文本文档	1 KB

2. **创建简单web项目**

   项目结构如下：



3. **导入jar包**



4. **创建普通类User.java**

```java
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:28
 */
public class User {

    public void add() {
        System.out.println("ioc test...");
    }
}
```

5. **编写配置文件**

说明：配置文件的路径和名称没有严格要求，官方建议配置文件放到src目录下，官方建议名称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">
    <!-- 配置需要生成对象的类的信息 -->
    <bean id="user" class="com.hbnu.pojo.User"/>
</beans>
```

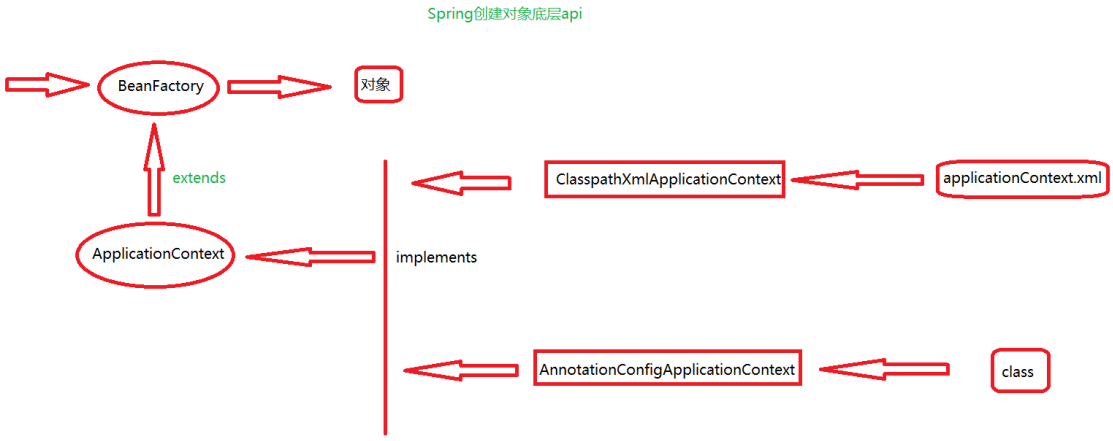
6. **编写测试类，测试ioc**

```java
package com.hbnu.pojo;

```

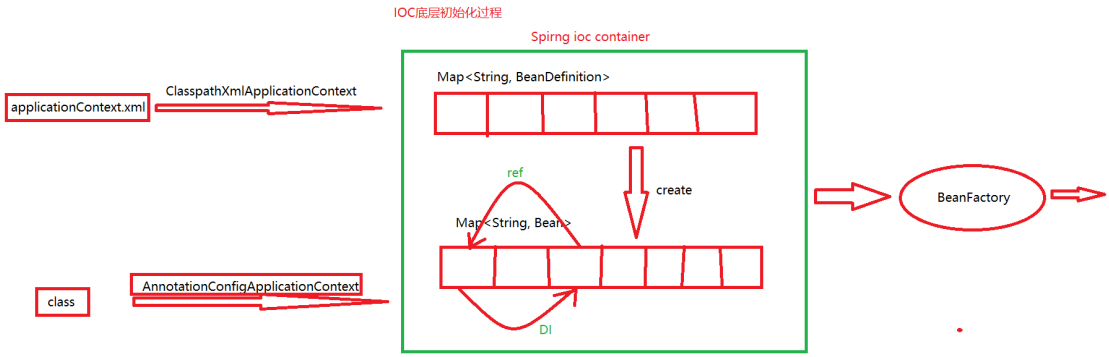
```java
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:30
 */
public class IocTest {

    @Test
    public void testIoc() {
        // 解析配置文件
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");

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

        user.add();
    }
}
```

3、IOC底层API分析

- api分析



- IOC底层初始化过程



四、Spring中Bean管理方式

Spring中Bean对象的管理：通过类的无参构造函数、通过静态工厂（了解）、通过实例工厂（了解）

1、通过类的无参构造函数管理Bean对象

- 创建User类

```
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:28
 */
public class User {

    public void add() {
        System.out.println("ioc test...");
    }
}
```

- 创建配置文件

```
<?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">
    <!-- 配置需要生成对象的类的信息 -->
    <bean id="user" class="com.hbnu.pojo.User"/>
</beans>
```

- 测试

```
package com.hbnu.pojo;

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

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:30
 */
public class IocTest {

    @Test
    public void testIoc() {
        // 解析配置文件
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");

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

```
19
20          user.add();
21      }
22  }
```

2、通过静态工厂管理Bean对象

- 创建工厂类

```
1   package com.hbnu.pojo;
2
3   /**
4    * @author 陈迪凯
5    * @date 2021-03-09 14:36
6    */
7   public class UserFactory {
8
9       public static User getUser() {
10          return new User();
11      }
12  }
```

- 修改配置文件

```
1   <?xml version="1.0" encoding="utf-8" ?>
2   <beans xmlns="http://www.springframework.org/schema/beans"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="
5           http://www.springframework.org/schema/beans
6           http://www.springframework.org/schema/beans/spring-beans.xsd">
7       <!-- 配置需要生成对象的类的信息 -->
8       <!--<bean id="user" class="com.hbnu.pojo.User"/>-->
9
10      <!-- 通过静态工厂管理Bean对象 -->
11      <bean id="factory" class="com.hbnu.pojo.UserFactory" factory-
    method="getUser"></bean>
12  </beans>
```

- 测试

```
1   package com.hbnu.pojo;
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 2021-03-02 16:30
10   */
11  public class IocTest {
12
13      @Test
14      public void testIoc() {
15          // 解析配置文件
```

```
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

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

        user.add();
    }
}
```

3、通过实例工厂管理Bean对象

- 创建工厂类

```
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-09 14:36
 */
public class UserFactory {

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

- 修改配置文件

```
<?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">
    <!-- 配置需要生成对象的类的信息 -->
    <!--<bean id="user" class="com.hbnu.pojo.User"/>-->

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

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

- 测试

```
package com.hbnu.pojo;

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

```

```java
/**
 * @author 陈迪凯
 * @date 2021-03-02 16:30
 */
public class IocTest {

    @Test
    public void testIoc() {
        // 解析配置文件
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

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

        user.add();
    }
}
```

五、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：

- singleton：单例模式，Spring容器中只会有一个Bean对象
- protoType：多例模式，Spring容器中会有多个Bean对象
- request：WEB项目中，Spring创建Bean对象后，将Bean对象存入request域中
- session：WEB项目中，Spring创建Bean对象后，将Bean对象存入session域中
- globalSession：WEB项目中，基于Porlet环境（Java的WEB环境组件），如果没有Porlet环境，那么跟session一样

六、属性注入

创建对象的同时，给类中的属性赋值。属性注入的方式：通过有参构造函数注入属性（重点）、通过set方法注入属性（重点）、通过接口注入属性（了解，Spring不支持这种注入）。

通过有参构造函数注入属性

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


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

通过set方法注入属性

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

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

通过接口注入属性

```
public interface Write {
    public void write(String username);
}

public class User implements Write {
    private String username;

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

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

1、通过有参构造注入属性

- 修改User类

```java
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:28
 */
public class User {

    private String username;

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

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

```

- 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"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- 配置需要生成对象的类的信息 -->
    <!--<bean id="user" class="com.hbnu.pojo.User"/>-->

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

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

```

```
    <!-- 1、通过有参构造函数注入属性 -->
    <bean id="user" class="com.hbnu.pojo.User">
        <!--
        name属性：类中的属性名
        value属性：属性值
        -->
        <constructor-arg name="username" value="chendikai"/>
    </bean>
</beans>
```

- 测试

```
package com.hbnu.pojo;

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

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:30
 */
public class IocTest {

    @Test
    public void testIoc() {
        // 解析配置文件
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");

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

        user.add();
    }
}
```

2、通过set方法注入属性

- 修改User类

```
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:28
 */
public class User {

    private String username;

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

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

```
    }
}
```

- 修改配置文件

```
<?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">
    <!-- 配置需要生成对象的类的信息 -->
    <!--<bean id="user" class="com.hbnu.pojo.User"/>-->

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

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

    <!-- 1、通过有参构造函数注入属性 -->
    <!--
    <bean id="user" class="com.hbnu.pojo.User">
        &lt;!&ndash;
        name属性：类中的属性名
        value属性：属性值
        &ndash;&gt;
        <constructor-arg name="username" value="chendikai"/>
    </bean>
    -->

    <!-- 2、通过set方法注入属性 -->
    <bean id="user" class="com.hbnu.pojo.User">
        <property name="username" value="chendikai"></property>
    </bean>
</beans>
```

- 测试

```
package com.hbnu.pojo;

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

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:30
 */
public class IocTest {

    @Test
```

```
    public void testIoc() {
        // 解析配置文件
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");

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

        user.add();
    }
}
```

3、注入类类型属性

- 编写UserDao类

```
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-09 15:45
 */
public class UserDao {
    public void printUserDao() {
        System.out.println("UserDao......printUserDao");
    }
}
```

- 编写UserService类

```
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-09 15:46
 */
public class UserService {
    private UserDao userDao;

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

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

```

- 修改配置文件

```
<?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">
    <!-- 配置需要生成对象的类的信息 -->
    <!--<bean id="user" class="com.hbnu.pojo.User"/>-->

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

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

    <!-- 1、通过有参构造函数注入属性 -->
    <!--
    <bean id="user" class="com.hbnu.pojo.User">
        &lt;!&ndash;
        name属性：类中的属性名
        value属性：属性值
        &ndash;&gt;
        <constructor-arg name="username" value="chendikai"/>
    </bean>
    -->

    <!-- 2、通过set方法注入属性 -->
    <!--
    <bean id="user" class="com.hbnu.pojo.User">
        <property name="username" value="chendikai"></property>
    </bean>
    -->

    <!-- 3、注入类类型属性 -->
    <bean id="userDao" class="com.hbnu.pojo.UserDao"></bean>
    <bean id="userService" class="com.hbnu.pojo.UserService">
        <property name="userDao" ref="userDao"></property>
    </bean>
</beans>
```

- 测试

```
package com.hbnu.pojo;

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

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:30
 */
public class IocTest {

    @Test
    public void testIoc() {
        // 解析配置文件
```

```
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

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

        userService.printUserService();
    }
}
```

4、注入复杂数据类型

复杂数据类型包括：数组、list集合、map集合、Properties类型。

- 创建DataType类

```
package com.hbnu.pojo;

import java.util.List;
import java.util.Map;
import java.util.Properties;

/**
 * @author 陈迪凯
 * @date 2021-03-09 16:23
 */
public class DataType {
    private String[] arr;
    private List<String> list;
    private Map<String, String> map;
    private Properties properties;

    public void setArr(String[] arr) {
        this.arr = arr;
    }

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

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

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

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

- 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"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- 配置需要生成对象的类的信息 -->
    <!--<bean id="user" class="com.hbnu.pojo.User"/>-->

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

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

    <!-- 1、通过有参构造函数注入属性 -->
    <!--
    <bean id="user" class="com.hbnu.pojo.User">
        &lt;!&ndash;
        name属性：类中的属性名
        value属性：属性值
        &ndash;&gt;
        <constructor-arg name="username" value="chendikai"/>
    </bean>
    -->

    <!-- 2、通过set方法注入属性 -->
    <!--
    <bean id="user" class="com.hbnu.pojo.User">
        <property name="username" value="chendikai"></property>
    </bean>
    -->

    <!-- 3、注入类类型属性 -->
    <!--
    <bean id="userDao" class="com.hbnu.pojo.UserDao"></bean>
    <bean id="userService" class="com.hbnu.pojo.UserService">
        <property name="userDao" ref="userDao"></property>
    </bean>
    -->

    <!-- 4、注入复杂数据类型 -->
    <bean id="dataType" class="com.hbnu.pojo.DataType">
        <!-- 4.1 注入数组类型属性 -->
        <property name="arr">
            <list>
                <value>铠</value>
                <value>小乔</value>
                <value>妲己</value>
            </list>
        </property>

        <!-- 4.2 注入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>

        <!-- 4.3 注入map集合类型 -->
        <property name="map">
            <map>
                <entry key="username" value="陈迪凯"></entry>
                <entry key="gender" value="男"></entry>
                <entry key="address" value="湖北黄石"></entry>
            </map>
        </property>

        <!-- 4.4 注入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>
```

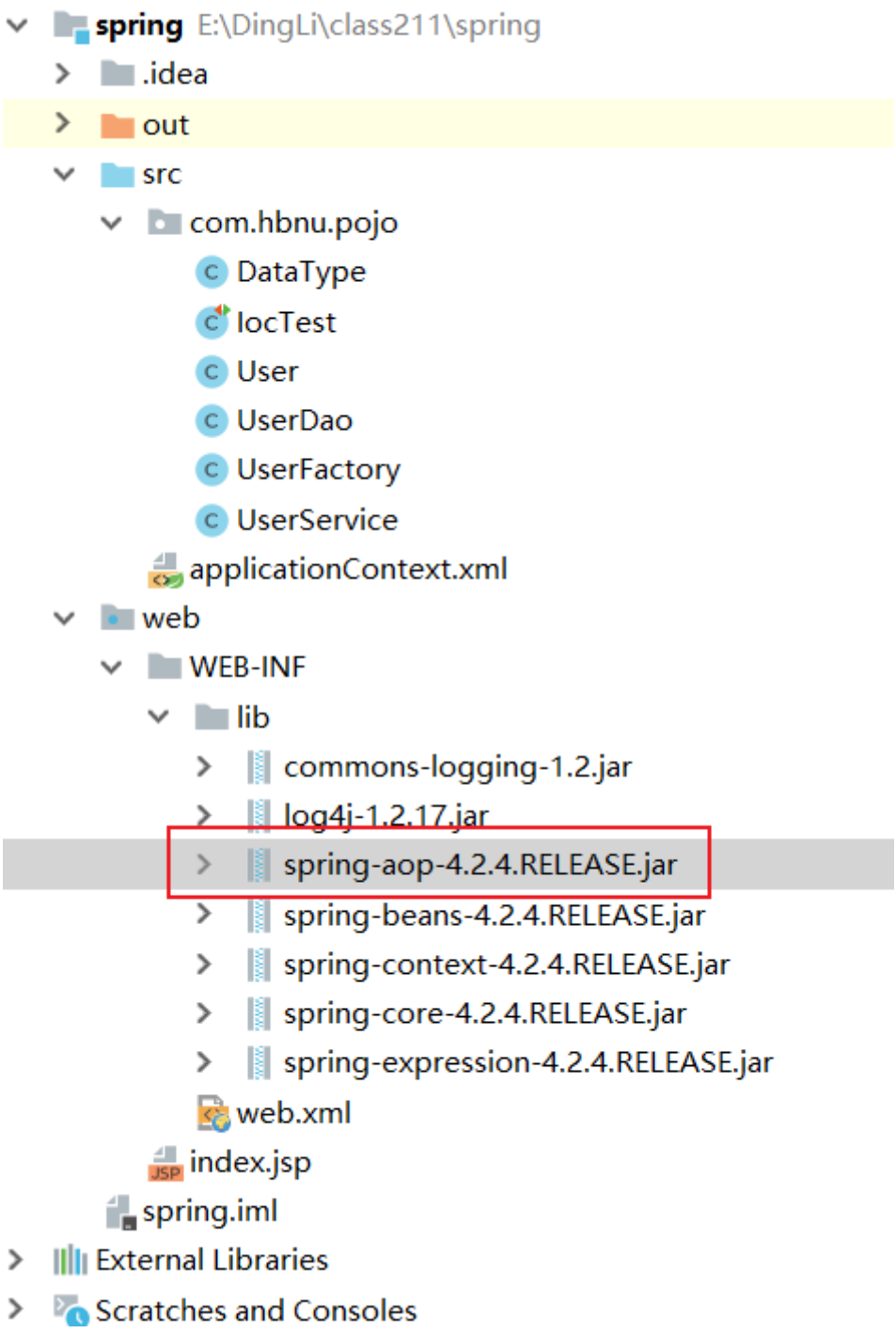
- 测试

```
package com.hbnu.pojo;

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

/**
 * @author 陈迪凯
 * @date 2021-03-02 16:30
 */
public class IocTest {

    @Test
    public void testIoc() {
        // 解析配置文件
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");

        DataType dataType = (DataType)
applicationContext.getBean("dataType");

        dataType.printDataType();
    }
}
```

说明：IOC：控制反转，负责对象的创建；DI：依赖注入，给类中的属性注入值。

七、Spring注解开发

1、导入jar包



2、创建配置文件annotation.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">
```

```
13
14      <!-- 配置包扫描注解 -->
15      <context:component-scan base-package="com.hbnu.annotation">
    </context:component-scan>
16  </beans>
```

3、通过注解生成Bean对象

- 创建User类

```
1  package com.hbnu.annotation;
2
3  import org.springframework.stereotype.Component;
4
5  /**
6   * @author 陈迪凯
7   * @date 2021-03-09 16:56
8   */
9  @Component(value = "user")
10 public class User {
11     public void add() {
12         System.out.println("User......add......");
13     }
14 }
```

- 测试

```
1  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 2021-03-09 16:57
10  */
11 public class AnnotationTest {
12
13     @Test
14     public void testAnnotation() {
15         ApplicationContext applicationContext = new
   ClassPathXmlApplicationContext("annotation.xml");
16
17         User user = (User) applicationContext.getBean("user");
18
19         user.add();
20     }
21 }
```

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

4、通过注解注入属性

- 创建UserDao

```
package com.hbnu.annotation;

import org.springframework.stereotype.Component;

/**
 * @author 陈迪凯
 * @date 2021-03-09 17:19
 */
@Component(value = "userDao")
public class UserDao {
    public void printUserDao() {
        System.out.println("UserDao......printUserDao......");
    }
}
```

- 创建UserService

```
package com.hbnu.annotation;

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

/**
 * @author 陈迪凯
 * @date 2021-03-09 17:21
 */
@Component(value = "userService")
public class UserService {

    // @Autowired
    @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
 */
```

```
11 public class AnnotationTest {
12
13     @Test
14     public void testAnnotation() {
15         ApplicationContext applicationContext = new
   ClassPathXmlApplicationContext("annotation.xml");
16
17         UserService userService = (UserService)
   applicationContext.getBean("userService");
18
19         userService.printUserService();
20     }
21 }
```

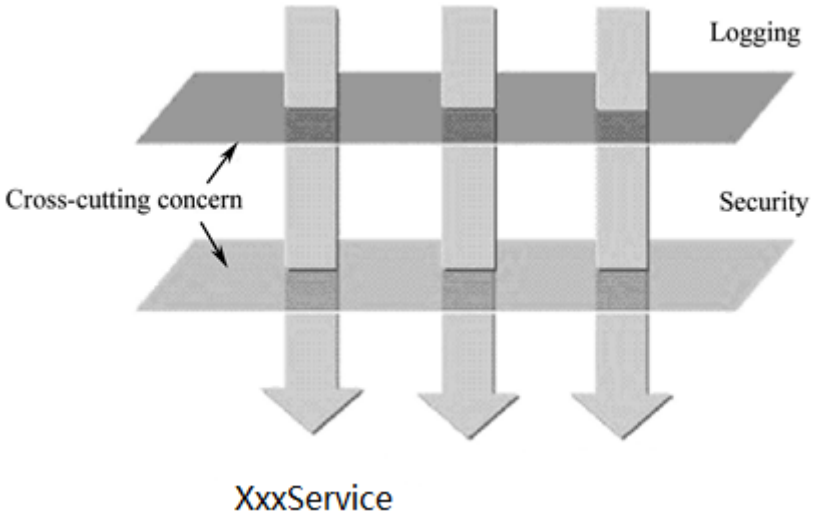
八、Spring AOP编程

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，它是对面向对象（OOP）编程的一个补充和完善，我们的面向对象编程在实际项目中，可以看作是一个静态过程（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），而面向切面是一个一个的动态过程（对象运行时织入一些功能）

面向切面应用案例示意图：



1.2、AOP演变过程

- 纵向机制

纵向抽取机制

```
public class UserService {
    // 添加用户业务
    public void insertUser() {
        // 添加用户代码逻辑
        // 添加日志代码逻辑
    }
}

***添加用户时，需要一个日志记录，记录什么时间添加了哪个用户
```

```
public class LogRecord {
    // 添加日志功能模块
    public void writeLog() {
        // 添加日志代码逻辑
    }
}

public class UserService extends LogRecord {
    // 添加用户业务
    public void insertUser() {
        // 添加用户代码逻辑

        super.writeLog();
    }
}
```

- 横向机制

AOP：横向抽取机制，底层采用动态代理机制实现

```
public inteface UserDao {
    public void insertUser();
}
```

第一种情况：针对有接口的情况，AOP底层采用的是JDK动态代理

implements

AOP：帮我们创建一个接口实现类的代理对象，这个对象不是真正对象，这个对象与UserService的实例对象是平级的，这个代理对象实现与UserService相同的功能

```
public class UserService implements UserDao {
    public void insertUser() {
        // 添加用户代码逻辑
    }
}
```

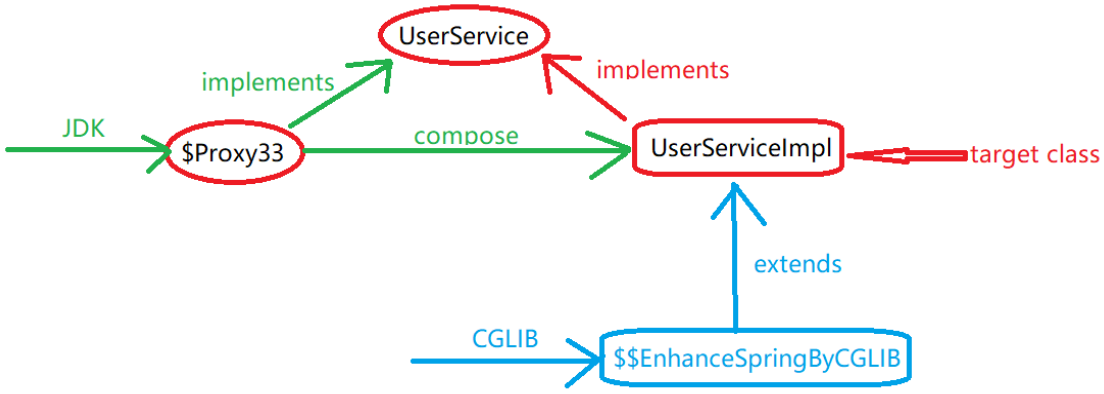
第二种情况：针对没有接口，AOP底层采用CGLIB动态代理

```
public class UserService {
    // 添加用户业务
    public void insertUser() {
        // 添加用户代码逻辑
    }
}
```

AOP：底层采用动态代理机制创建子类对象的代理对象，功能的扩展通过子类的代理对象完成

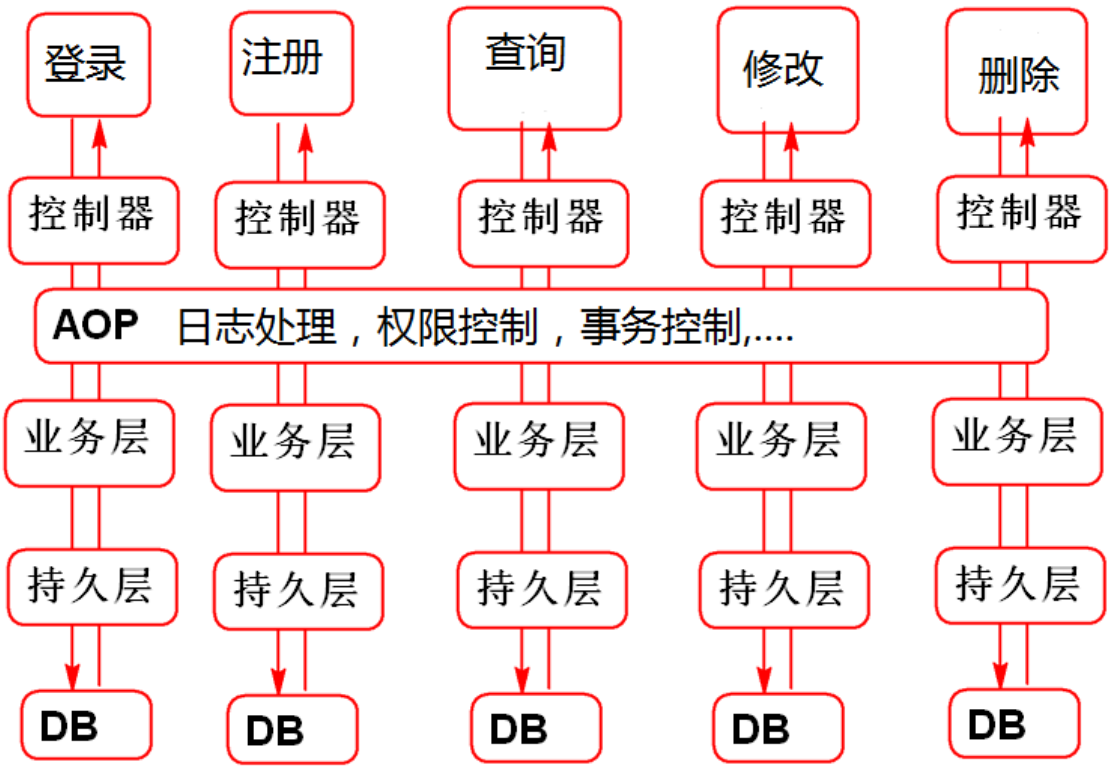
1.3、AOP代理机制

- 假如目标对象（被代理对象）实现了接口，则AOP底层采用JDK动态代理机制为我们的目标对象创建代理对象（目标类和代理类实现相同的接口）
- 假如目标对象（被代理对象）未实现接口，则AOP底层采用CGLIB动态代理机制为我们的目标对象创建代理对象（创建的代理类继承了目标类对象）



- AOP在实际项目中的应用场景

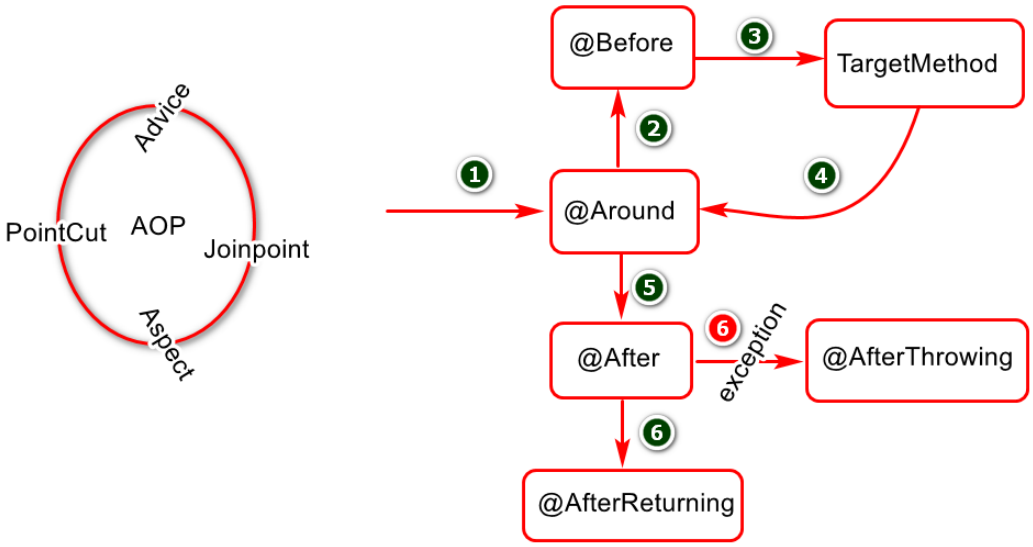
  AOP一般应用于：日志管理、权限管理、事务管理、缓存管理等



2、AOP相关术语

- 切面（Aspect）：横切面对象，就是一个具体的类对象（借助@Aspect可以声明切面）
- 连接点（JoinPoint）：程序执行过程中特点的点，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）：对连接点内容的一个定义，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）：在切面的某个连接点上执行的动作（扩展功能）
  - 前置通知（Before）：在目标方法之前执行
  - 返回通知（AfterReturning）：在目标方法之后执行
  - 异常通知（AfterThrowing）：在目标方法发生异常时执行
  - 后置通知（After）：在目标方法之后执行，肯定会执行，相当于try...catch...finally中的finally
  - 环绕通知（Around）：在目标方法之前和之后都会执行

  AOP通知执行过程：

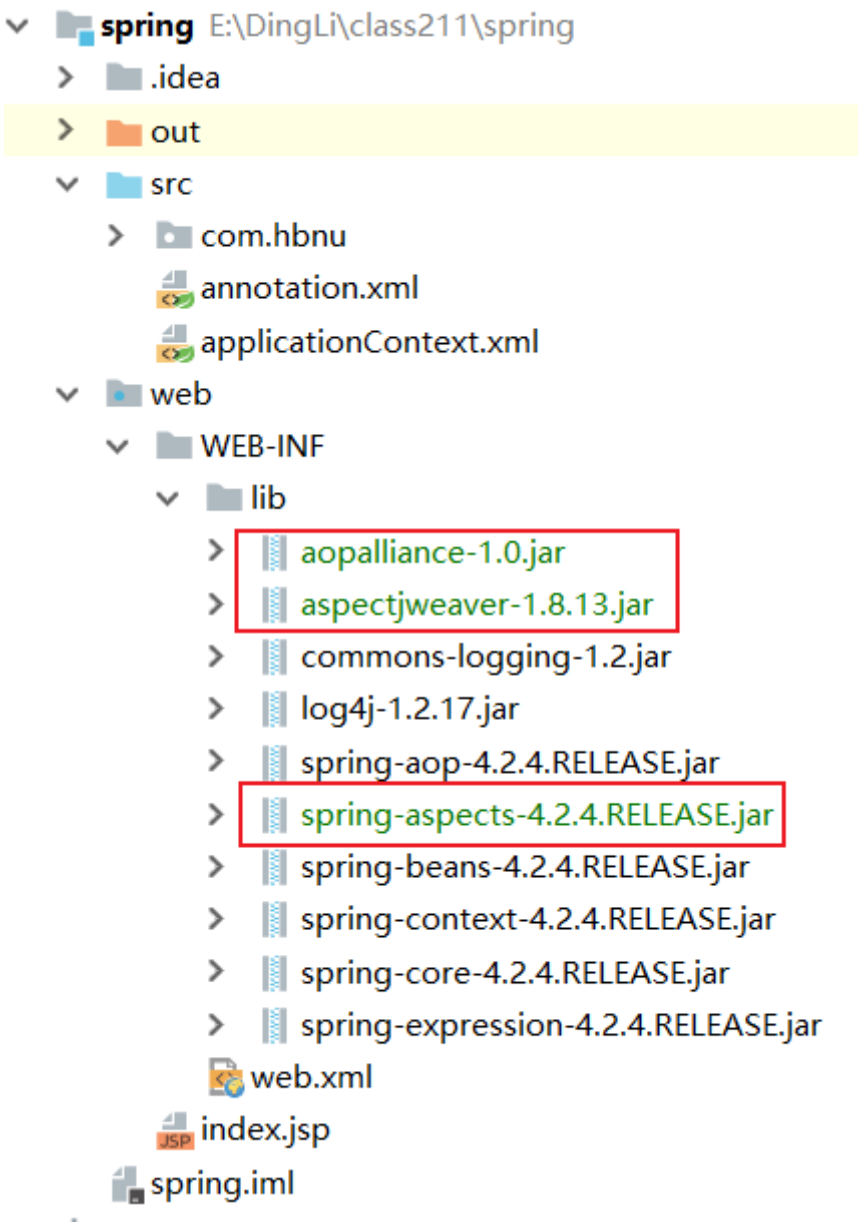


3、Spring AOP基础编程

在Spring中使用AOP，需要用到AspectJ实现，AspectJ不是Spring的一部分，它是面向切面编程的一个框架，和Spring一起来实现AOP操作，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的注解方式

3.1、基于AspectJ的xml配置

- 导入jar包



- 创建配置文件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

```java
package com.hbnu.aop;

/**
 * @author 陈迪凯
 * @date 2021-03-16 16:32
 */
public class UserService {

    public void insertUser() {
        System.out.println("添加用户......");
    }
}
```

- 创建增强类LogRecord

```java
package com.hbnu.aop;

import org.aspectj.lang.ProceedingJoinPoint;

/**
 * @author 陈迪凯
 * @date 2021-03-16 16:34
 */
public class LogRecord {

    public void before1() {
        System.out.println("在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
<?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>
            <!-- 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

```
package com.hbnu.aop;

import com.hbnu.pojo.User;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * @author 陈迪凯
 * @date 2021-03-16 16:45
 */
public class AOPTest {

    @Test
    public void testAop() {
        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，常用的AOP切入点表达式及说明

指示符	描述
bean	用于匹配指定bean id的类中方法的执行
within	用于匹配指定包名下类方法的执行
execution	用于进行细粒度方法匹配，执行具体的方法
@annotation	用于匹配指定注解修饰的方法执行

1. 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")：指定一个类中所有方法
   - bean("*Service")：指定所有以Service结尾的类中所有方法

   说明：bean表达式内部的对象是由Spring容器管理的，bean表达式内部的名字必须是spring容器某个bean的id

2. 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")：指定具体的类，对类中的所有方法进行增强
   - within("com.hbnu.aop.*")：对指定包下所有的类中的方法进行匹配
   - within("com.hbnu.aop..*")：对指定包下说有的子目录中类方法进行匹配

3. 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

   常用表达式格式：execution(<访问修饰符>?<返回值类型><方法名>(参数)<异常>)

   - execution(* com.hbnu.aop.UserService.insertUser)
   - executioin(* com.hbnu.aop.UserService.*(..))
   - execution(* *.*(..))
   - execution(* *.add*(..))

4. @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)：对@RequestLog修饰的方法进行增强

   说明：RequestLog是我们自定义的注解，当我们使用@RequestLog注解修饰方法是，AOP底层会对被修饰的方法进行增强

3.3、基于AspectJ的注解方式

- 创建被增强类UserService

```java
package com.hbnu.aop;

/**
 * @author 陈迪凯
 * @date 2021-03-16 16:32
 */
public class UserService {

    public void insertUser() {
        System.out.println("添加用户......");
    }
}
```

- 创建增强类LogRecord

```java
package com.hbnu.aop;

import org.aspectj.lang.ProceedingJoinPoint;

/**
 * @author 陈迪凯
 * @date 2021-03-16 16:34
 */
public class LogRecord {

    public void before1() {
        System.out.println("在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
<?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.Aspect;
import org.aspectj.lang.annotation.Before;

/**
 * @author 陈迪凯
 * @date 2021-03-16 16:34
 */
@Aspect
public class LogRecord {

    @Before("execution(* com.hbnu.aop.UserService.*(..))")
    public void before1() {
        System.out.println("在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 com.hbnu.pojo.User;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
```

```
 * @author 陈迪凯
 * @date 2021-03-16 16:45
 */
public class AOPTest {

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

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

        userService.insertUser();
    }
}
```