

# J2EE高级开发框架

## 一、Spring框架简介

### 1、Spring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

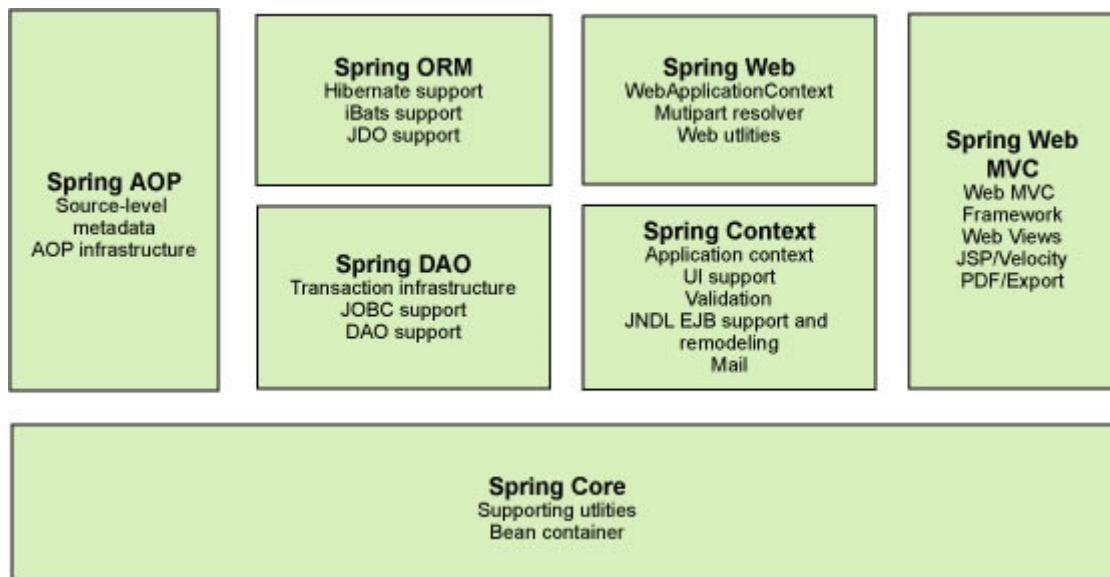
### 2、Spring优势

- 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：



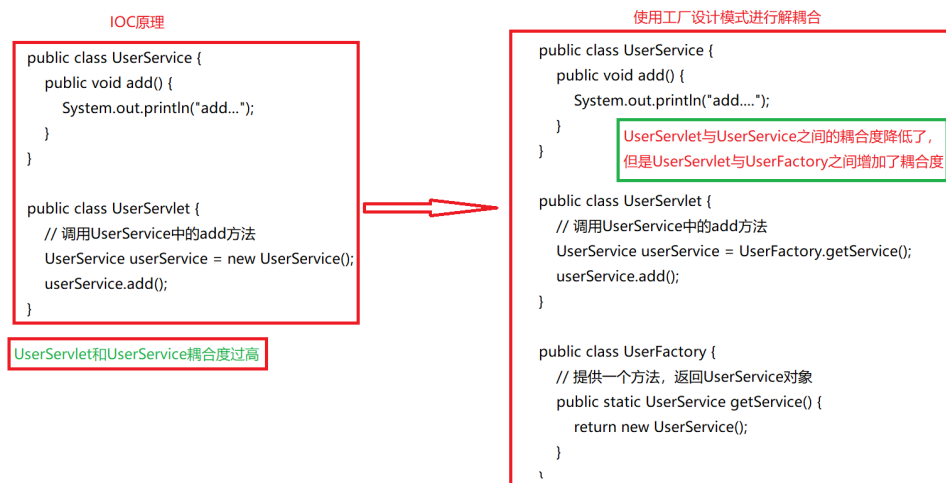
说明：Spring框架模块（组件）可以单独存在和使用，也可以和其他一个或多个模块联合使用。各个模块的功能如下：

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.println("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

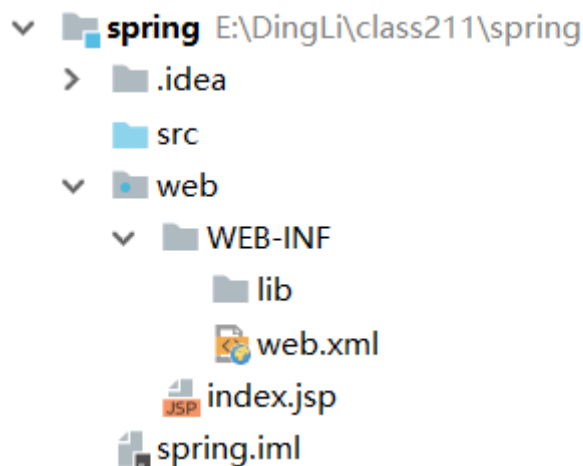
[Spring框架下载](#)

下载后解压目录如下：

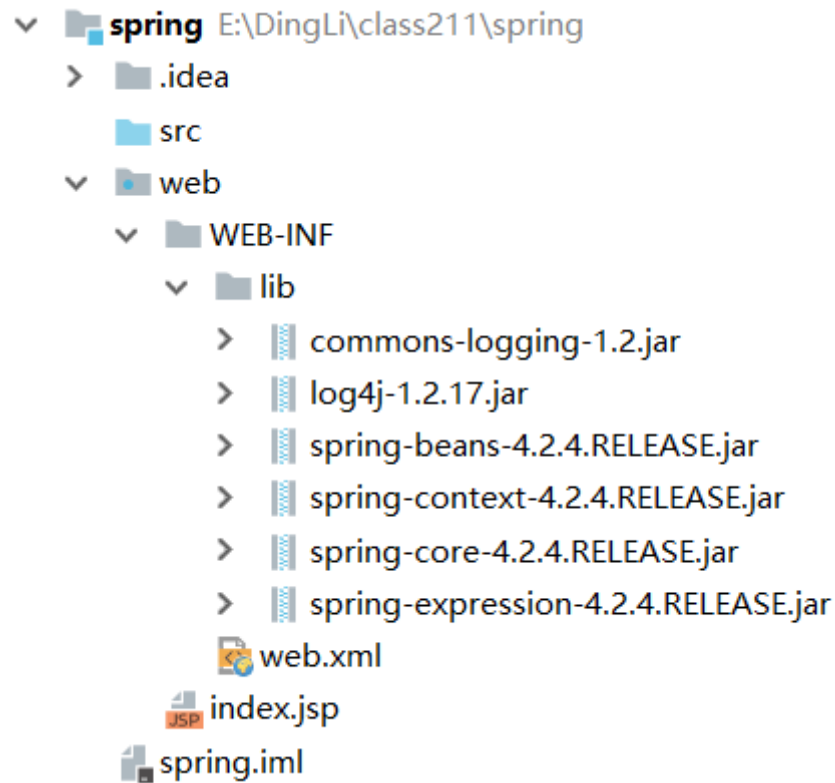
docs	2021-02-28 22:15	文件夹	
libs	2021-02-28 22:15	文件夹	
schema	2021-02-28 22:15	文件夹	
license.txt	2015-12-17 0:43	文本文档	15 KB
notice.txt	2015-12-17 0:43	文本文档	1 KB
readme.txt	2015-12-17 0:43	文本文档	1 KB

### 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

```
1 package com.hbnu.pojo;
2
3 /**
4  * @author 陈迪凯
5  * @date 2021-03-02 16:28
6  */
7 public class User {
8
9     public void add() {
10         System.out.println("ioc test...");
11     }
12 }
```

#### 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

```
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 </beans>
```

#### 6. 编写测试类，测试ioc

```
1 package com.hbnu.pojo;
2
```

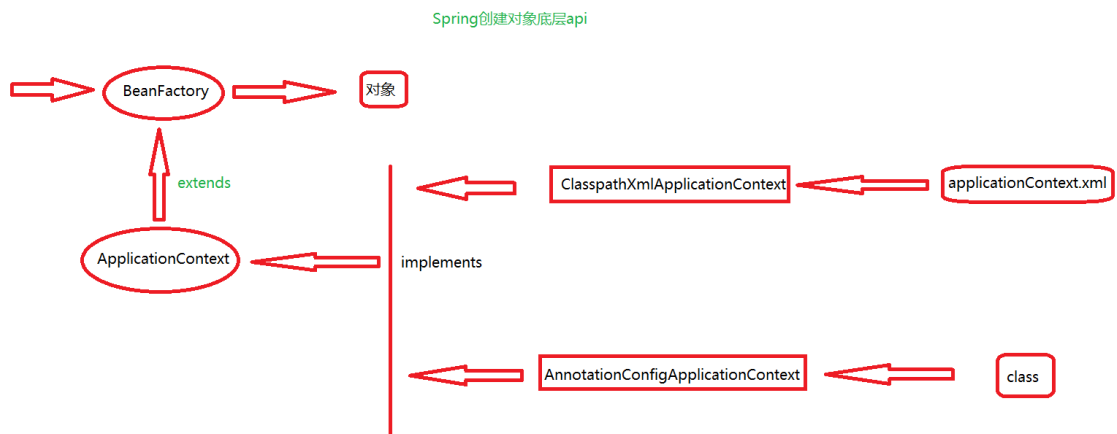
```

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
16         ApplicationContext applicationContext = new
            ClassPathXmlApplicationContext("applicationContext.xml");
17
18         User user = (User) applicationContext.getBean("user");
19
20         user.add();
21     }
22 }

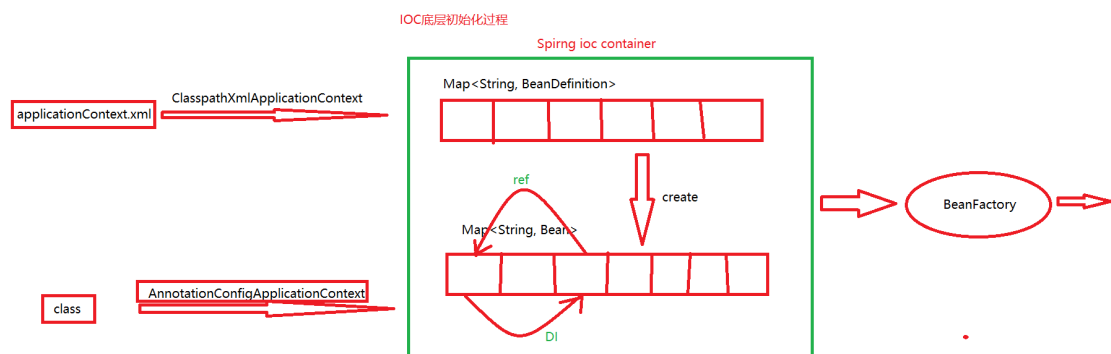
```

### 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

```
1 package com.hbnu.pojo;
2
3 /**
4  * @author 陈迪凯
5  * @date 2021-03-02 16:28
6  */
7 public class User {
8
9     public void add() {
10         System.out.println("ioc test...");
11     }
12 }
```

- 创建配置文件

```
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 </beans>
```

- 测试

```
1 package com.hbnu.pojo;
2
3 import org.junit.Test;
4 import org.springframework.context.ApplicationContext;
5 import
6 org.springframework.context.support.ClassPathXmlApplicationContext;
7
8 /**
9  * @author 陈迪凯
10  * @date 2021-03-02 16:30
11  */
12 public class IocTest {
13
14     @Test
15     public void testIoc() {
16         // 解析配置文件
17         ApplicationContext applicationContext = new
18             ClassPathXmlApplicationContext("applicationContext.xml");
19
20         User user = (User) applicationContext.getBean("user");
21     }
22 }
```

```

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-
12        method="getUser"></bean>
13 </beans>

```

- 测试

```

1 package com.hbnu.pojo;
2
3 import org.junit.Test;
4 import org.springframework.context.ApplicationContext;
5 import
6 org.springframework.context.support.ClassPathXmlApplicationContext;
7
8 /**
9  * @author 陈迪凯
10  * @date 2021-03-02 16:30
11  */
12 public class IocTest {
13
14     @Test
15     public void testIoc() {
16         // 解析配置文件
17     }
18 }

```

```

16     ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
17
18     User user = (User) applicationContext.getBean("factory");
19
20     user.add();
21 }
22 }

```

### 3、通过实例工厂管理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 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
13    <!-- 通过实例工厂管理Bean对象 -->
14    <bean id="factory" class="com.hbnu.pojo.UserFactory"></bean>
15    <bean id="user" factory-bean="factory" factory-method="getUser">
</bean>
16 </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
16          ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
17
18          User user = (User) applicationContext.getBean("user");
19
20          user.add();
21      }
22  }

```

## 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 <constructor-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容器中只会存在一个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

```
1 package com.hbnu.pojo;
2
3 /**
4  * @author 陈迪凯
5  * @date 2021-03-02 16:28
6  */
7 public class User {
8
9     private String username;
10
11     public User(String username) {
12         this.username = username;
13     }
14
15     public void add() {
16         System.out.println("properties injected..." + username);
17     }
18 }
19
```

- 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-
12method="getUser"></bean>-->
13
14    <!-- 通过实例工厂管理Bean对象 -->
15    <!--<bean id="factory" class="com.hbnu.pojo.UserFactory"></bean>
16    <bean id="user" factory-bean="factory" factory-method="getUser">
17    </bean> -->
```

```

17      <!-- 1、通过有参构造函数注入属性 -->
18      <bean id="user" class="com.hbnu.pojo.User">
19          <!--
20              name属性: 类中的属性名
21              value属性: 属性值
22          -->
23          <constructor-arg name="username" value="chendikai"/>
24      </bean>
25 </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
16         ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
17
18         User user = (User) applicationContext.getBean("user");
19
20         user.add();
21     }
22 }

```

## 2、通过set方法注入属性

- 修改User类

```

1 package com.hbnu.pojo;
2
3 /**
4  * @author 陈迪凯
5  * @date 2021-03-02 16:28
6  */
7 public class User {
8
9     private String username;
10
11     public void setUsername(String username) {
12         this.username = username;
13     }
14
15     public void add() {
16         System.out.println("properties injected..." + username);

```

```

17     }
18 }

```

- 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-
12         method="getUser"></bean>-->
13
14     <!-- 通过实例工厂管理Bean对象 -->
15     <!-- <bean id="factory" class="com.hbnu.pojo.UserFactory"></bean>
16         <bean id="user" factory-bean="factory" factory-method="getUser">
17         </bean> -->
18
19     <!-- 1、通过有参构造函数注入属性 -->
20     <!--
21     <bean id="user" class="com.hbnu.pojo.User">
22         <!-->
23         <!--name属性: 类中的属性名
24         <!--value属性: 属性值
25         <!-->
26         <!--<constructor-arg name="username" value="chendikai"/>
27     </bean>
28     -->
29
30     <!-- 2、通过set方法注入属性 -->
31     <bean id="user" class="com.hbnu.pojo.User">
32         <property name="username" value="chendikai"></property>
33     </bean>
34 </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
16         ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
17
18         User user = (User) applicationContext.getBean("user");
19
20         user.add();
21     }
22 }

```

### 3、注入类类型属性

- 编写UserDao类

```

1 package com.hbnu.pojo;
2
3 /**
4  * @author 陈迪凯
5  * @date 2021-03-09 15:45
6  */
7 public class UserDao {
8     public void printUserDao() {
9         System.out.println("UserDao.....printUserDao");
10    }
11 }

```

- 编写UserService类

```

1 package com.hbnu.pojo;
2
3 /**
4  * @author 陈迪凯
5  * @date 2021-03-09 15:46
6  */
7 public class UserService {
8     private UserDao userDao;
9
10    public void setUserDao(UserDao userDao) {
11        this.userDao = userDao;
12    }
13
14    public void printUserService() {
15        System.out.println("UserService.....printUserService");
16        userDao.printUserDao();
17    }
18 }
19

```

- 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
13     <!-- 通过实例工厂管理Bean对象 -->
14     <!-- <bean id="factory" class="com.hbnu.pojo.UserFactory"></bean>
15         <bean id="user" factory-bean="factory" factory-method="getUser">
</bean> -->
16
17     <!-- 1、通过有参构造函数注入属性 -->
18     <!--
19     <bean id="user" class="com.hbnu.pojo.User">
20         &lt;!--&dash;
21         name属性: 类中的属性名
22         value属性: 属性值
23         &dash;&gt;
24         <constructor-arg name="username" value="chendikai"/>
25     </bean>
26     -->
27
28     <!-- 2、通过set方法注入属性 -->
29     <!--
30     <bean id="user" class="com.hbnu.pojo.User">
31         <property name="username" value="chendikai"></property>
32     </bean>
33     -->
34
35     <!-- 3、注入类类型属性 -->
36     <bean id="userDao" class="com.hbnu.pojo.UserDao"></bean>
37     <bean id="userService" class="com.hbnu.pojo.UserService">
38         <property name="userDao" ref="userDao"></property>
39     </bean>
40 </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

```

```

16     ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
17
18     UserService userService = (UserService)
applicationContext.getBean("userService");
19
20     userService.printUserService();
21 }
22 }

```

#### 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

```

1  package com.hbnu.pojo;
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 2021-03-09 16:23
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 void setArr(String[] arr) {
18          this.arr = arr;
19      }
20
21      public void setList(List<String> list) {
22          this.list = list;
23      }
24
25      public void setMap(Map<String, String> map) {
26          this.map = map;
27      }
28
29      public void setProperties(Properties properties) {
30          this.properties = properties;
31      }
32
33      public void printDataType() {
34          System.out.println("arr:" + arr);
35          System.out.println("list:" + list);
36          System.out.println("map:" + map);
37          System.out.println("properties:" + properties);
38      }
39  }

```

- 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-
12method="getUser"></bean>-->
13
14    <!-- 通过实例工厂管理Bean对象 -->
15    <!-- <bean id="factory" class="com.hbnu.pojo.UserFactory"></bean>
16    <bean id="user" factory-bean="factory" factory-method="getUser">
17</bean> -->
18
19    <!-- 1、通过有参构造函数注入属性 -->
20    <!--
21    <bean id="user" class="com.hbnu.pojo.User">
22        &lt;t;!&ndash;
23        name属性：类中的属性名
24        value属性：属性值
25        &ndash;&gt;
26        <constructor-arg name="username" value="chendikai"/>
27    </bean>
28    -->
29
30    <!-- 2、通过set方法注入属性 -->
31    <!--
32    <bean id="user" class="com.hbnu.pojo.User">
33        <property name="username" value="chendikai"></property>
34    </bean>
35    -->
36
37    <!-- 3、注入类类型属性 -->
38    <!--
39    <bean id="userDao" class="com.hbnu.pojo.UserDao"></bean>
40    <bean id="userService" class="com.hbnu.pojo.UserService">
41        <property name="userDao" ref="userDao"></property>
42    </bean>
43    -->
44
45    <!-- 4、注入复杂数据类型 -->
46    <bean id="dataType" class="com.hbnu.pojo.DataType">
47        <!-- 4.1 注入数组类型属性 -->
48        <property name="arr">
49            <list>
50                <value>铠</value>
51                <value>小乔</value>
52                <value>姐己</value>
53            </list>
54        </property>
55
56        <!-- 4.2 注入list集合类型 -->
57        <property name="list">

```



```

56         <list>
57             <value>张三丰</value>
58             <value>张翠山</value>
59             <value>张无忌</value>
60         </list>
61     </property>
62
63     <!-- 4.3 注入map集合类型 -->
64     <property name="map">
65         <map>
66             <entry key="username" value="陈迪凯"></entry>
67             <entry key="gender" value="男"></entry>
68             <entry key="address" value="湖北黄石"></entry>
69         </map>
70     </property>
71
72     <!-- 4.4 注入properties类型 -->
73     <property name="properties">
74         <props>
75             <prop key="driverClass">com.mysql.cj.jdbc.Driver</prop>
76             <prop key="url">jdbc:mysql:///hbnu</prop>
77             <prop key="username">root</prop>
78             <prop key="password">chendikai</prop>
79         </props>
80     </property>
81 </bean>
82 </beans>

```

- 测试

```

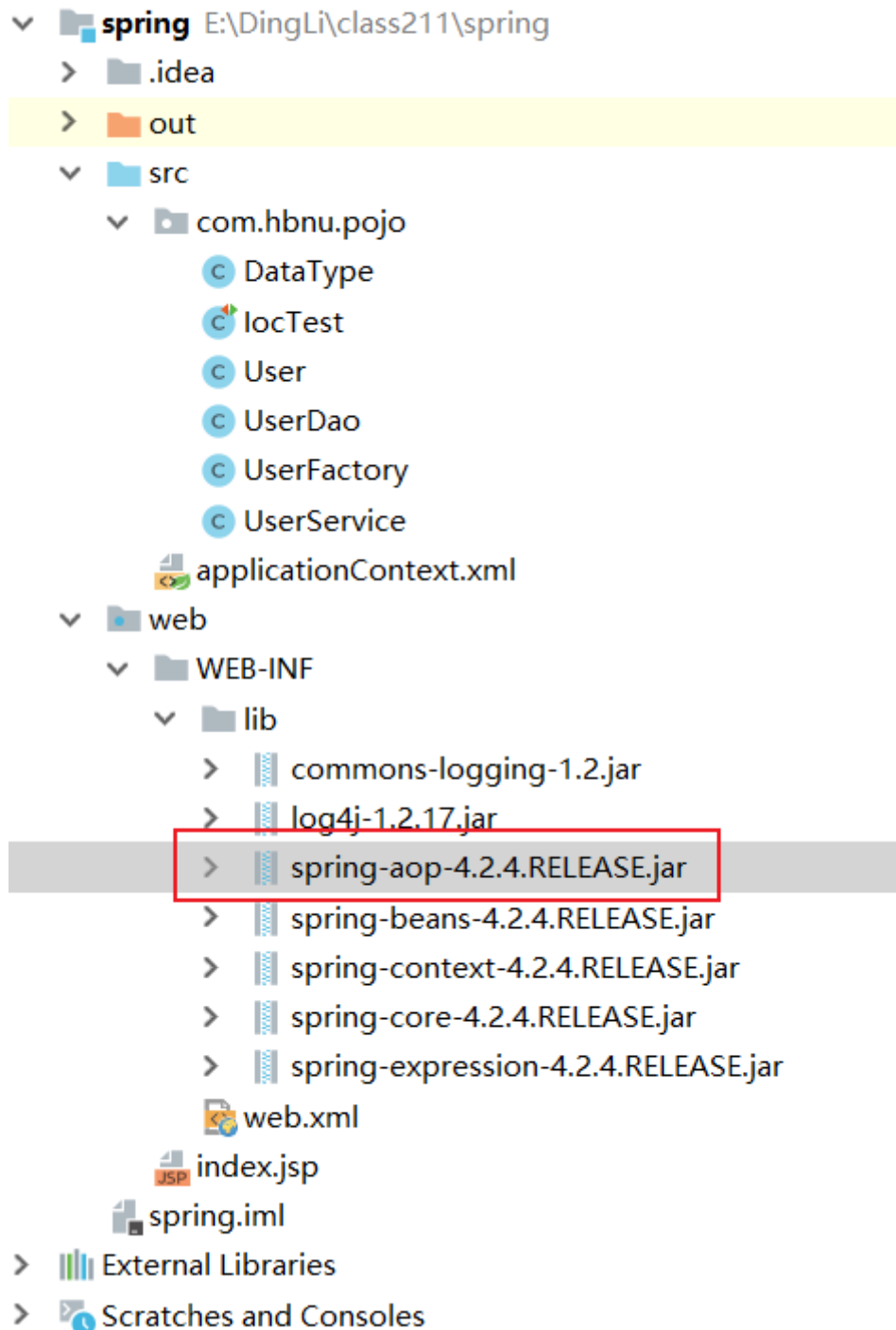
1  package com.hbnu.pojo;
2
3  import org.junit.Test;
4  import org.springframework.context.ApplicationContext;
5  import
org.springframework.context.support.ClassPathXmlApplicationContext;
6
7  /**
8   * @author 陈迪凯
9   * @date 2021-03-02 16:30
10  */
11 public class IocTest {
12
13     @Test
14     public void testIoc() {
15         // 解析配置文件
16         ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
17
18         DataType dataType = (DataType)
applicationContext.getBean("dataType");
19
20         dataType.printDataType();
21     }
22 }

```

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、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       xmlns:aop="http://www.springframework.org/schema/aop"
5       xmlns:context="http://www.springframework.org/schema/context"
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
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     <!-- 配置包扫描注解 -->
15     <context:component-scan base-package="com.hbnu.annotation">
16     </context:component-scan>
17 </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

```
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 @Component(value = "userDao")
10 public class UserDao {
11     public void printUserDao() {
12         System.out.println("UserDao.....printUserDao.....");
13     }
14 }
```

- 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 2021-03-09 17:21
9  */
10 @Component(value = "userService")
11 public class UserService {
12
13     // @Autowired
14     @Resource(name = "userDao")
15     private UserDao userDao;
16
17     public void printUserService() {
18         System.out.println("UserService.....printUserService.....");
19         userDao.printUserDao();
20     }
21 }
```

- 测试

```
1 package com.hbnu.annotation;
2
3 import org.junit.Test;
4 import org.springframework.context.ApplicationContext;
5 import
6 org.springframework.context.support.ClassPathXmlApplicationContext;
7
8 /**
9  * @author 陈迪凯
10  * @date 2021-03-09 16:57
11  */
```

```
11 public class AnnotationTest {
12
13     @Test
14     public void testAnnotation() {
15         ApplicationContext applicationContext = new
16         ClassPathXmlApplicationContext("annotation.xml");
17
18         UserService userService = (UserService)
19         applicationContext.getBean("userService");
20
21         userService.printUserService();
22     }
23 }
```