# 尚马教育 JAVA 课程

# Spring容器

文档编号：C03

创建日期： 2017-07-07

最后修改日期：2019-10-18

版 本 号：V3.5

电子版文件名：尚马教育-第三阶段-3.spring容器.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-19 | 张元林 | 初始版本 | V1.0 |
| 2018-08-08 | 王绍成 | 版本更新 | V2.0 |
| 2019-10-17 | 冯勇涛 | 课件格式以及课程深度加深 | V3.0 |
| 2021-01-20 | 冯勇涛 | 调整课程顺序，优化课程 | V3.5 |

目录

[尚马教育 JAVA 课程 1](#_Toc27093)

[Spring容器 1](#_Toc9058)

[1. spring介绍 3](#_Toc24628)

[1.1. 认识spring 3](#_Toc24510)

[1.2. Spring的好处 3](#_Toc10186)

[1.3. Spring核心服务 4](#_Toc413)

[1.3.1. IOC 4](#_Toc3186)

[1.3.2. DI 5](#_Toc22647)

[1.3.3. AOP 5](#_Toc20108)

[2. Spring容器对象结构图 8](#_Toc16083)

[3. Spring快速入门1 8](#_Toc5022)

[3.1. 版本选择 9](#_Toc3278)

[3.2. 下载jar包 9](#_Toc16694)

[3.3. 编写代码并添加xml配置文件 9](#_Toc28673)

[3.4. 初始化spring容器对象 10](#_Toc15309)

[3.5. 注意点 11](#_Toc23379)

[4. Spring快速入门2 12](#_Toc4526)

[4.1. 添加mysql驱动与Druid连接池包 12](#_Toc5830)

[4.2. 修改xml配置文件配置DataSource 12](#_Toc6707)

[4.3. 初始化spring容器对象 12](#_Toc30270)

[4.4. 注意点 12](#_Toc5080)

[5. bean标签属性详解 13](#_Toc14280)

[6. IOC控制反转 14](#_Toc24561)

[6.1. Bean标签配置 14](#_Toc19704)

[6.2. 工厂实例化bean 14](#_Toc12553)

[6.2.1. 静态工厂实例化bean 14](#_Toc12372)

[6.2.2. 实例工厂实例化bean 15](#_Toc26927)

[6.3. 包扫描 16](#_Toc11998)

[7. DI依赖注入 17](#_Toc4031)

[7.1. Set注入 17](#_Toc3908)

[7.2. 构造器注入 18](#_Toc17003)

[7.3. 集合注入 19](#_Toc2881)

[7.4. 内部bean注入 20](#_Toc25487)

[7.5. 自动装配 20](#_Toc29990)

[8. 类中常用注解 21](#_Toc6160)

[9. Xml中常用标签 21](#_Toc29361)

[10. Spring测试环境搭建 22](#_Toc31070)

[10.1. 环境准备 22](#_Toc16282)

[10.2. 编写测试类 22](#_Toc17483)

|  |
| --- |
| 知识点：  知识点1：认识IOC控制反转思想，并熟练使用spring的IOC配置  知识点2：认识DI依赖注入思想，熟练使用。 |

## spring介绍

### 认识spring

Spring是一个开源的控制反转(Inversion of Control ,IoC)和面向切面(AOP)的容器框架。帮助开发人员分离组件之间的依赖关系，它的主要目的是简化企业开发。

Spring 利用了一些现有的技术，如ORM框架、日志框架、Quartz，线程池，任务 等。

### Spring的好处

在项目中引入spring立即可以带来下面的好处:

* 降低组件之间的耦合度,实现软件各层之间的解耦。

|  |
| --- |
| Dao  Controller  Service |

* 可以使用容器提供的众多服务，如：事务管理服务、消息服务等等。当我们使用容器管理事务时，开发人员就不再需要手工控制事务。
* 容器提供单例模式支持，开发人员不再需要自己编写实现代码。
* 容器提供了AOP技术，利用它很容易实现如权限拦截、运行期监控等功能。
* 容器提供的众多辅助类，使用这些类能够加快应用的开发，如：JdbcTemplate、 RedisTemplate、RestTemplate等。
* Spring对于主流的应用框架提供了集成支持，如：集成Hibernate 、Struts 、JPA等，这样更便于应用的开发。

### Spring核心服务

核心服务提供Spring框架的基本功能Bean Container。Spring以bean的方式组织和管理Java应用中的各个组件及其关系,Spring使用BeanFactory来产生和管理Bean，它是工厂模式的实现。BeanFactory使用控制反转(IoC)模式将应用的配置和依赖性规范与实际的应用程序代码分开。

Spring AOP 模块直接将面向切面的编程思想集成到了 Spring框架中。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，很简单就可以将声明性事务管理集成到应用程序中。

#### IOC

控制反转：

|  |
| --- |
| public class PersonServiceBean {  private PersonDao personDao ~~= new PersonDaoImpl()~~;  public void save(Person person){  personDao.save(person);  }  } |

PersonDaoImpl 是在应用内部创建及维护的。所谓控制反转就是应用本身不负责依赖对象的创建及维护，依赖对象的创建及维护是由外部容器负责的。这样控制权就由应用转移到了外部容器，控制权的转移就是所谓反转。

#### DI

依赖注入理解：当我们把依赖对象交给外部容器负责创建，那么PersonServiceBean 类可以改成如下：

|  |
| --- |
| public class PersonServiceBean {  private PersonDao personDao ;  //通过构造器参数，让容器把创建好的依赖对象注入进PersonServiceBean，当然也可以使用setter方法进行注入。  public PersonServiceBean(PersonDao personDao){  this.personDao=personDao;  }  public void save(Person person){  personDao.save(person);  }  } |

所谓依赖注入就是指：在运行期间，由外部容器动态地将依赖对象注入到组件中。

#### AOP

AOP面向切面编程：在运行期间通过动态代理实现程序功能统一维护的一种技术，AOP是[OOP](https://baike.baidu.com/item/OOP" \t "https://baike.baidu.com/item/AOP/_blank)的延续，是软件开发中的一个热点，也是[Spring](https://baike.baidu.com/item/Spring" \t "https://baike.baidu.com/item/AOP/_blank)框架中的一个重要内容，利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的[耦合度](https://baike.baidu.com/item/%E8%80%A6%E5%90%88%E5%BA%A6/2603938" \t "https://baike.baidu.com/item/AOP/_blank)降低，提高程序的可重用性，同时提高了开发的效率。

以一个实际业务开发中非常常见的应用示例，日志:

|  |
| --- |
| 1. 业务介绍：进行整数的计算操作 |
| 1. 完成核心业务后，有了新的业务外的监控需求：在程序执行期间追踪正在发生的活动。 |
| 1. 利用传统编码方式实现核心业务和非核心业务要求： |
|  |
| 缺点：  代码混乱：越来越多的非业务需求(日志,验证等)代码的添加,使原有的业务方法急剧膨胀. 每个方法在处理核心逻辑的同时还必须兼顾其他辅助性的关注点。  代码分散: 以日志需求为例, 只是为了满足这个单一需求, 就需要在多个方法里多次重复相同的日志代码. 如果日志需求发生变化, 必须修改所有模块。 |
| 1. 使用AOP思路实现：   使用AOP利用一个代理将原业务对象包装起来, 然后用该代理对象取代原始对象，将日志处理的代码织入到原始对象的方法执行代码中: |
|  |

## Spring容器对象结构图

|  |
| --- |
|  |

## Spring快速入门1

|  |
| --- |
| 1. 选择合适的spring版本 2. 下载对应版本的jar包,并导入核心包到工程 3. 创建spring.xml配置文件 4. 实例化spring容器对象，测试bean的获取 |

### 版本选择

|  |
| --- |
|  |

带GA表示正式发布版本,说明是release版本。习惯选用非最新正式版。

可选择5.2版本。

### 下载jar包

Spring官网：https://spring.io/

下载地址：http://repo.spring.io/release/org/springframework/spring/

当前入门案例需要4个spring的核心jar包，以及依赖的common-logging.jar。

|  |
| --- |
|  |

### 编写代码并添加xml配置文件

|  |
| --- |
| **public class** SysuserServiceImpl **implements** ISysuserService {  **public** SysuserServiceImpl() {  System.***out***.print(**"SysuserServiceImpl constructor"**);  }  @Override  **public** Sysuser login(String uname, String upwd) {  System.***out***.println(**"login："** + uname + **"--"** + upwd);  **return null**;  } } |
| *<?***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"**>  <**bean id="sysuserService" class="com.javasm.sys.service.SysuserServiceImpl"**>  </**bean**> </**beans**> |

**注意点：**

注意点1：该文件必须有beans根标签，并且beans标签中必须引入beans.xsd模板定义文件；

注意点2：bean标签的class不能配置接口类,必须是一个可实例化的类。

注意点3：bean标签的id必须唯一，不可重复。

注意点4：xml配置文件放在src类路径下，名称随意。此处文件名spring.xml

### 初始化spring容器对象

|  |
| --- |
| @Test **public void** test2\_springUserService(){  ApplicationContext ac = **new** ClassPathXmlApplicationContext(**"spring.xml"**);  ISysuserService userService1 = (ISysuserService) ac.getBean(**"userService"**);  ISysuserService userService =ac.getBean(ISysuserService.**class**);  System.***out***.println(userService);  System.***out***.println(userService1);  System.***out***.println(userService==userService1);  userService.login(**"fyt"**,**"fyt"**); } |

### 注意点

注意点1：当前加载类路径下xml配置，使用ClassPathXmlApplicationContext创建spring容器。

注意点2：ApplicationContext接口中的三个getBean重载方法，两个按照bean的id获取对象，1个按照类型获取对象。

注意点3：三个getBean方法得到的对象，是同一个对象，即spring管理的bean对象默认单例状态。

注意点3：当执行new ClassPathXmlApplicationContext对象时，SysuserServiceImpl对象的构造器会立即执行。在该行代码处打断点，查看对象内数据：

|  |
| --- |
|  |

## Spring快速入门2

### 添加mysql驱动与Druid连接池包

|  |
| --- |
|  |

### 修改xml配置文件配置DataSource

|  |
| --- |
| <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"**>  <**property name="url" value="jdbc:mysql://127.0.0.1:3306/704b"**></**property**>  <**property name="driverClassName" value="com.mysql.jdbc.Driver"**></**property**>  <**property name="username" value="root"**></**property**>  <**property name="password" value="root"**></**property**>  <**property name="initialSize" value="5"**></**property**>  <**property name="minIdle" value="5"**></**property**>  </**bean**> |

### 初始化spring容器对象

|  |
| --- |
| @Test **public void** test3\_springDataSource(){  ApplicationContext ac = **new** ClassPathXmlApplicationContext(**"spring.xml"**);  DataSource ds = ac.getBean(DataSource.**class**);  System.***out***.println(ds);  ((ClassPathXmlApplicationContext) ac).close(); } |

### 注意点

注意点1：bean标签的 init-method属性用来指定bean实例化后立即执行的初始化方法；destroy-method属性指定spring容器销毁后执行的释放资源方法。

注意点2：只有显示执行了ApplicaiotnContext的close方法才能够触发destroy-method。

注意点3：**property标签的name属性值是DruidDataSource类中的成员变量名，value是赋值，等价于调用了set方法。配置bean标签与如下代码效果类似：**

|  |
| --- |
| @Test **public void** test3\_dataSource() **throws** SQLException {  DruidDataSource ds = **new** DruidDataSource();  ds.setUrl(**"jdbc:mysql://127.0.0.1:3306/704b"**);  ds.setDriverClassName(**"com.mysql.jdbc.Driver"**);  ds.setUsername(**"root"**);  ds.setPassword(**"root"**);  ds.setInitialSize(5);  ds.setMinIdle(5);  **try** {  ds.init();  } **catch** (SQLException e) {  e.printStackTrace();  }  System.***out***.println(ds.getConnection());  ds.close(); } |

## bean标签属性详解

|  |
| --- |
| Id属性:注册bean对象在spring容器中的标示名称。 |
| Scope属性:bean的作用域：  <**bean id="userService" class="com.javasm.sys.service.SysuserServiceImpl" scope="prototype"**></**bean**>  .singleton  在每个Spring IoC容器中一个bean定义只有一个对象实例。  .prototype  每次从容器获取bean都是新的对象。<bean id=“” class=“” scope=“prototype”/> |
| Init-method与Destory-method属性:bean的初始化和销毁方法  <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"**> |
| Lazy-init属性：bean对象延迟初始化  <**bean id="userService" class="com.javasm.sys.service.SysuserServiceImpl" lazy-init="true"**></**bean**> |

## IOC控制反转

IOC：控制反转，应用本身不负责依赖对象的创建及维护，依赖对象的创建及维护是由外部容器负责的。这样控制权就由应用转移到了外部容器，控制权的转移就是所谓反转。

### Bean标签配置

入门案例中即在xml中配置bean标签注册bean对象到spring容器。一般jar包中类需要注册进spring容器使用bean标签配置。

|  |
| --- |
| <**bean id="userService" class="com.javasm.sys.service.SysuserServiceImpl"**></**bean**> |

### 工厂实例化bean

构建对象的过程比较复杂的情况下使用工厂实例化bean,比如sqlSessionFactory对象的构建，如果需要把mybatis的sqlSessionFactory对象交给spring容器，可使用工厂实例化bean。

#### 静态工厂实例化bean

|  |
| --- |
| **public class** MySqlSessionFactoryBean {  *//静态工厂* **public static** SqlSessionFactory createSqlSessionFactory(){  **try** {  InputStream resourceAsStream = Resources.*getResourceAsStream*(**"mybatis-config.xml"**);  SqlSessionFactory f = **new** SqlSessionFactoryBuilder().build(resourceAsStream);  **return** f;  } **catch** (IOException e) {  e.printStackTrace();  }  **return null**;  } } |
| <**bean id="factoryBean" class="com.javasm.factory.MySqlSessionFactoryBean" factory-method="createSqlSessionFactory"**></**bean**> |

注意点：工厂方法是static修饰，表示静态工厂方法。

#### 实例工厂实例化bean

|  |
| --- |
| **public class** MySqlSessionFactoryBean {  *//实例方法* **public** SqlSessionFactory createSqlSessionFactory(){  **try** {  InputStream resourceAsStream = Resources.*getResourceAsStream*(**"mybatis-config.xml"**);  SqlSessionFactory f = **new** SqlSessionFactoryBuilder().build(resourceAsStream);  **return** f;  } **catch** (IOException e) {  e.printStackTrace();  }  **return null**;  } } |
| *<!--先实例化工厂对象-->* <**bean id="factoryBean" class="com.javasm.factory.MySqlSessionFactoryBean"**></**bean**> *<!--再调用实例工厂对象的方法，把方法的返回值注册到容器中-->* <**bean id="sqlSessionFactory" factory-bean="factoryBean" factory-method="createSqlSessionFactory"**></**bean**> |

注意点1：工厂方法没有static修饰，必须先实例化出工厂对象，再调用工厂对象的实例方法。

注意点2：**factory-bean属性值是工厂bean对象的id值，表示引用对象。**

### 包扫描

一般由开发人员自定义的类使用包扫描方式注册进spring容器。

|  |
| --- |
| <!-- 识别类上的@Controller，@Service，@Repository，@Component等等注解。-->  <context:component-scan base-package=*"com.javasm"*></context:component-scan> |
| @Controller用于标注控制层组件;  @Service用于标注业务层组件;  @Repository用于标注数据访问组件，即DAO组件;  @Component泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注; |
| @Component(**"userController"**) **public class** SysuserController {} |
| @Service **public class** SysuserServiceImpl **implements** ISysuserService {} |
| @Component **public class** AsyncManager {} |

注意点1：context:component-scan不仅仅识别这四个注解，后续学习其它。

注意点2：通过注解方式注册的bean，默认id是类名首字母小写;可以自定义id

注意点3：这四个注解当前是没有区别，效果都是注册bean到spring容器，后续springMVC框架有区别，我们需要有好的代码习惯，分别注解各层的bean对象。

注意点4：不能注解到接口类上。

## DI依赖注入

依赖注入：指组件之间依赖关系由容器在运行期决定，形象的说，即由容器动态的将某个依赖关系注入到组件之中。具体使用方式如下：

|  |
| --- |
| Set注入：调用bean对象中的set方法注入依赖值。  构造器注入：实例化bean对象时，调用有参构造注入依赖值。  集合注入：bean对象中的Array，List，Map等集合属性注入值。  内部bean注入：本质仍然是set注入或构造器注入。  自动装配：基于包扫描bean配置。 |

### Set注入

|  |
| --- |
| **public class** SysuserController {  **private** ISysuserService **sysuserService**;  **private** String **str**;  **public void** setSysuserService(ISysuserService sysuserService) {  **this**.**sysuserService** = sysuserService;  }  **public void** setStr(String str) {  **this**.**str** = str;  } |
| <**bean id="userController" class="com.javasm.sys.controller.SysuserController"**>  <**property name="sysuserService" ref="userService"**></**property**>  <**property name="str" value="admin"**></**property**> </**bean**> |

注意点1：property标签的ref与value两个属性的选择，如果需要注入的值是容器中的bean对象，则使用ref指定引用bean的id。如果注入的值是简单类型（String，Date，Integer等），则使用value指定值。

注意点2：另一种set注入方法：引入p命名空间，在bean节点中以“p:属性名=属性值”的方式为属性注入值。

|  |
| --- |
| **public class** SysuserServiceImpl **implements** ISysuserService {  **private** ISysuserMapper **um**;  **private** String **uname**;  } |
| <**bean id="sysuserService" class="com.javasm.sys.service.SysuserServiceImpl" p:uname="fyt" p:um-ref="userMapper"**> </**bean**> <**bean id="userMapper" class="com.javasm.sys.mapper.SysuserMapperImpl"**> </**bean**> |

### 构造器注入

|  |
| --- |
| **public class** SysuserController {  **private** ISysuserService **sysuserService**;  **private** String **str**;  **private** Integer **uage**;  **public** SysuserController(ISysuserService sysuserService,String str)  {  System.***out***.println(**" SysuserController init-------"**);  **this**.**sysuserService** = sysuserService;  **this**.**str** = str;  }  } |
| <**bean id="userController" class="com.javasm.sys.controller.SysuserController"**>  <**constructor-arg name="sysuserService" ref="userService"**></**constructor-arg**>  <**constructor-arg name="str" value="张三"**></**constructor-arg**> </**bean**> |

注意点1：**constructor-arg标签中的name属性表示形参名；index属性表示形参索引号。**

### 集合注入

|  |
| --- |
| **public class** SysroleController {  **private** String **strAttr**;  **private** Integer **intAttr**;  **private** List<String> **strList**;  **private** List<Sysuser> **usersList**;  **private** Integer[] **intsArray**;  **private** Map<String,Integer> **strMaps**;  //TODO setter,getter生成  } |
| <**bean id="sysroleController" class="com.javasm.sys.controller.SysroleController"**>  <**property name="strAttr" value="stringValue"**></**property**>  <**property name="intAttr" value="100"**></**property**>  <**property name="strList"**>  <**list**>  <**value**>老贾</**value**>  <**value**></**value**>  <**value**>ccc</**value**>  </**list**>  </**property**>  <**property name="usersList"**>  <**list**>  <**bean class="com.javasm.sys.entity.Sysuser"**>  <**constructor-arg index="0" value="admin"**></**constructor-arg**>  <**constructor-arg index="1" value="admin"**></**constructor-arg**>  <**constructor-arg index="2" value="11111"**></**constructor-arg**>  </**bean**>  <**bean class="com.javasm.sys.entity.Sysuser"**>  <**constructor-arg index="0" value="admin2"**></**constructor-arg**>  <**constructor-arg index="1" value="admin2"**></**constructor-arg**>  <**constructor-arg index="2" value="2222"**></**constructor-arg**>  </**bean**>  </**list**>  </**property**>  <**property name="intsArray"**>  <**array**>  <**value**>111</**value**>  <**value**>222</**value**>  <**value**>333</**value**>  </**array**>  </**property**>  <**property name="strMaps"**>  <**map**>  <**entry key="a" value="111"**></**entry**>  <**entry key="b" value="222"**></**entry**>  </**map**>  </**property**> </**bean**> |

### 内部bean注入

以上7.3案例中**usersList属性的注入即内部bean注入，当前bean对象不能被其他对象引用。**

### 自动装配

通过注解方式来进行DI注入；一般与包扫描结合使用。

|  |
| --- |
| <context:component-scan base-package=*"com.javasm"*></context:component-scan> |
| @Component **public class** SysuserController {  @Resource  **private** ISysuserService **sysuserService**;  } |
| 如果没有指定name属性，并且按照默认的名称找不到依赖对象时， @Resource注解会回退到按类型装配。但一旦指定了name属性，就只能按名称进行装配。 |

注意点1：在java代码中使用@Autowired或@Resource注解方式进行装配，这两个注解的区别是：

@Autowired 默认按类型装配，类型匹配不上，再按照形参名称装配。

@Resource默认按形参名称装配，当找不到与名称匹配的bean才会按类型装配。

注意点2：Resource注解可以指定名称@Resource(name="userService"),指定后则只能按照名称进行装配，一般没有必要。

## 类中常用注解

|  |
| --- |
| @Scope用于指定scope作用域的（用在类或方法上）,等价于bean标签的scope属性  @PostConstruct用于指定初始化方法（用在方法上），等价于bean标签的init-method属性  @PreDestory用于指定销毁方法（用在方法上），等价于bean标签的destory-method属性 |

## Xml中常用标签

1. context:property-placeholder标签，导入类路径下的properties文件到spring容器。

定义dao.xml文件：

|  |
| --- |
| <**context:property-placeholder location="classpath:jdbc.properties" ignore-unresolvable="true"**></**context:property-placeholder**> <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"** >  <**property name="driverClassName" value="${jdbc.driver}"**></**property**>  <**property name="url" value="${jdbc.url}"**></**property**>  <**property name="username" value="${jdbc.username}"**></**property**>  <**property name="password" value="${jdbc.password}"**></**property**> </**bean**> |

注意点：context:property-placeholder标签的ignore-unresolvable属性是当引入了多个properties文件时必须指定的属性。

2.import标签，导入其它spring风格的xml文件

在Spring.xml总配置中通过import标签包含dao.xml：

|  |
| --- |
| <import resource="classpath:dao.xml"></import> |

## Spring测试环境搭建

### 环境准备

|  |
| --- |
| 加入Junit4.0 jar包  加入spring-test-xxx.jar  使用@RunWith(SpringJUnit4ClassRunner.class)注解描述当前测试用例依赖启动器  使用@ContextConfiguration(locations = { “classpath:applicationContext.xml” }) 配置spring配置文件 |

### 编写测试类

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.class)  @ContextConfiguration(locations = { "classpath:applicationContext.xml" })  public class TestAnnotation {  @Resource  private PersonDao personDao;  @Test  public void testAnnotation(){  personDao.add();  }  } |