# Spring源码深度解析与注解驱动开发

## 文档结构

### 1.1目录视图

**Spring注解驱动开发**

1. 容器

·AnnotationConfigApplicationContext

·组件添加

·组件赋值

·组件注入

·AOP

·声明式事务

1. 扩展原理

·BeanFactoryPostProcessor

·BeanDefinitionRegistryPostProcessor

·ApplicationListener

·Spring容器创建过程

1. Web

·Servlet3.0

·异步请求

## 第2节 组件注册

### 2.1 @Configuration&@Bean给容器中注册组件

|  |
| --- |
| *<?***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"**>  <**bean id="person" class="com.byf.bean.Person"**>  <**property name="name" value="张三"**></**property**>  <**property name="age" value="20"**></**property**>  </**bean**> </**beans**> |
| ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext(**"beans.xml"**); Person person = (Person) applicationContext.getBean(**"person"**); System.***out***.println(person); |

|  |
| --- |
| @Configuration **public class** PersonConfig {  @Bean(**"lisi"**)  **public** Person person(){  **return new** Person(**"李四"**, 21);  } } |
| ApplicationContext applicationContext =  **new** AnnotationConfigApplicationContext(PersonConfig.**class**); Person person = (Person) applicationContext.getBean(**"lisi"**); System.***out***.println(person); |

### 2.2 @ComponentScan-自动扫描组件&指定扫描规则

|  |
| --- |
| @Configuration @ComponentScan(value = **"com.byf"**, includeFilters = {  @ComponentScan.Filter(type = FilterType.***ANNOTATION***, classes = {Controller.**class**}) },useDefaultFilters = **false**) @ComponentScans(value = {@ComponentScan(value = **"com.byf"**, includeFilters = {  @ComponentScan.Filter(type = FilterType.***ANNOTATION***, classes = {Service.**class**}) },useDefaultFilters = **false**)}) **public class** PersonConfig {  @Bean(**"lisi"**)  **public** Person person(){  **return new** Person(**"李四"**, 21);  } } |
| **public class** IOCTest {  @Test  **public void** testIOC(){  ApplicationContext applicationContext = **new** AnnotationConfigApplicationContext(PersonConfig.**class**);  String[] beanNames = applicationContext.getBeanDefinitionNames();  **for**(String name : beanNames){  System.***out***.println(name);  }  } } |
| **personConfig**  **bookController**  **bookService**  **lisi** |

### 2.3自定义TypeFilter指定过滤规则

|  |
| --- |
| @Configuration @ComponentScans(value = {@ComponentScan(value = **"com.byf"**, includeFilters = {  @ComponentScan.Filter(type = FilterType.***ANNOTATION***, classes = {Controller.**class**}),  @ComponentScan.Filter(type = FilterType.***ASSIGNABLE\_TYPE***, classes = {BookService.**class**}),  @ComponentScan.Filter(type = FilterType.***CUSTOM***, value = {MyTypeFilter.**class**}) },useDefaultFilters = **false**)}) *//FilterType.ANNOTATION：根据注解类注入 //FilterType.ASSIGNABLE\_TYPE: 根据class名注入 //FilterType.ASPECTJ: 根据ASPECTJ表达式 //FilterType.REGEX：根据正则表达式 //FilterType.CUSTOM：根据自定义规则 /\*@ComponentScans(value = {@ComponentScan(value = "com.byf", excludeFilters = {  @ComponentScan.Filter(type = FilterType.ANNOTATION, classes = {Service.class}) })})\*/* **public class** PersonConfig {  @Bean(**"lisi"**)  **public** Person person(){  **return new** Person(**"李四"**, 21);  } } |

|  |
| --- |
| **public class** MyTypeFilter **implements** TypeFilter {  @Override  **public boolean** match(MetadataReader metadataReader, MetadataReaderFactory metadataReaderFactory) **throws** IOException {  AnnotationMetadata annotationMetadata = metadataReader.getAnnotationMetadata();  ClassMetadata classMetadata = metadataReader.getClassMetadata();  Resource resource = metadataReader.getResource();  String name = classMetadata.getClassName();  System.***out***.println(**"-->"** + name);  **if** (**"er"**.equals(name.substring(name.length()-2,name.length()))){  **return true**;  }  **return false**;  } } |
| **-->com.byf.AppTest**  **-->com.byf.bean.IOCTest**  **-->com.byf.App**  **-->com.byf.bean.Person**  **-->com.byf.config.MyTypeFilter**  **-->com.byf.controller.BookController**  **-->com.byf.dao.BookDao**  **-->com.byf.service.BookService**  **org.springframework.context.annotation.internalConfigurationAnnotationProcessor**  **.....**  **personConfig**  **myTypeFilter**  **bookController**  **lisi** |

### 2.4@Scope-设置组件作用域

|  |
| --- |
| @Configuration **public class** MainConfig2 {  *// 默认是单实例的  /\*\*  \** ***@see*** *ConfigurableBeanFactory#SCOPE\_PROTOTYPE  \** ***@see*** *ConfigurableBeanFactory#SCOPE\_SINGLETON  \* prototype：多实例  \* singleton：单实例（默认值）：ioc容器启动会调用方法创建对象放到ioc容器中。  \* 以后每次获取就是直接从容器（map.get()）中拿  \* request：同一个请求创建一个实例  \* session：同一个session创建一个实例  \*/* @Scope(value = **"prototype"**)  @Bean(**"person"**)  **public** Person person(){  System.***out***.println(**"给容器添加Person..."**);  **return new** Person(**"李四"**, 21);  } } |

|  |
| --- |
| @Test **public void** testIOC2(){  ApplicationContext applicationContext = **new** AnnotationConfigApplicationContext(MainConfig2.**class**);  System.***out***.println(**"ioc容器创建完成...."**);  Object p1 = applicationContext.getBean(**"person"**);  Object p2 = applicationContext.getBean(**"person"**);  System.***out***.println(p1 == p2);  */\*String[] beanNames = applicationContext.getBeanDefinitionNames();  for(String name : beanNames){  System.out.println(name);  }   Object p1 = applicationContext.getBean("person");  Object p2 = applicationContext.getBean("person");  System.out.println(p1 == p2);\*/* } |
| ioc容器创建完成....  给容器添加Person...  给容器添加Person...  false |

### 2.5@Lazy-bean懒加载

|  |
| --- |
| @Configuration **public class** MainConfig2 {  *// 默认是单实例的  /\*\*  \** ***@see*** *ConfigurableBeanFactory#SCOPE\_PROTOTYPE  \** ***@see*** *ConfigurableBeanFactory#SCOPE\_SINGLETON  \* prototype：多实例  \* singleton：单实例（默认值）：ioc容器启动会调用方法创建对象放到ioc容器中。  \* 以后每次获取就是直接从容器（map.get()）中拿  \* request：同一个请求创建一个实例  \* session：同一个session创建一个实例  \*  \* 懒加载：  \* 单实例bean：默认在容器启动的时候创建  \* 懒加载：容器启动的时候不创建对象，第一次使用（获取）Bean创建对象，并初始化  \*/  // @Scope(value = "prototype")* @Bean(**"person"**)  @Lazy  **public** Person person(){  System.***out***.println(**"给容器添加Person..."**);  **return new** Person(**"李四"**, 21);  } } |
| @Test **public void** testIOC2(){  ApplicationContext applicationContext = **new** AnnotationConfigApplicationContext(MainConfig2.**class**);  System.***out***.println(**"ioc容器创建完成...."**);  Object p1 = applicationContext.getBean(**"person"**);  Object p2 = applicationContext.getBean(**"person"**);  System.***out***.println(p1 == p2);  */\*String[] beanNames = applicationContext.getBeanDefinitionNames();  for(String name : beanNames){  System.out.println(name);  }   Object p1 = applicationContext.getBean("person");  Object p2 = applicationContext.getBean("person");  System.out.println(p1 == p2);\*/* } |
| ioc容器创建完成....  给容器添加Person...  true |