spring学习笔记—容器扩展点

|  |  |
| --- | --- |
|  | 🢂 内容概览 |
|  | Why：此文档用来做什么？它存在的意义是什么？为解决什么问题？   |  | | --- | |  |   What：当前包含了那些内容？   |  | | --- | |  |   How：此文档应如何参考？   |  | | --- | |  |   Who：此文档适用于那些人员阅读参考？   |  | | --- | |  | |

目录

[1 容器扩展点 3](#_Toc478746841)

[1.1 定制Bean：BeanPostProcessor 3](#_Toc478746842)

[1.1.1 示例：自定义BeanPostProcessor 3](#_Toc478746843)

[1.1.2 示例：RequiredAnnotationBeanPostProcessor 5](#_Toc478746844)

[1.2 定制元数据：BeanFactoryPostProcessor 5](#_Toc478746845)

[1.2.1 示例：PropertyPlaceHolderConfigurer 5](#_Toc478746846)

[1.2.2 示例：PropertyOverrideConfigurer 6](#_Toc478746847)

[1.3 定制Bean初始化逻辑：FactoryBean 6](#_Toc478746848)

# 容器扩展点

为了实现对IOC容器的定制，通常情况下，并不需要重写ApplicationContext的实现类，只需要实现几个定制接口；

## 定制Bean：BeanPostProcessor

|  |
| --- |
|  |

1. BeanPostProcessor接口用于在spring容器完成对每个Bean的实例化、配置和初始化后，定制bean的操作；
2. 可以定义多个BeanPostProcessor，并且按照指定顺序执行；顺序控制，需实现Ordered接口；
3. BeanPostProcessor只在当前容器生效，如果一个应用存在多个容器，比如存在层次的容器（spring mvc以spring容器作为父容器），BeanPostProcessor也只会在自己所在容器中生效；
4. BeanPostProcessor可以探测到bean中定义的各种初始化方法作为回调函数；包括：InitializingBean接口的afterPropertiesSet()，通过xml指定的init-method以及使用@PostConstruct注解的方法；
5. ApplicationContext会自动探测并注册实现了BeanPostProcessor的Bean，并在bean初始化后调用这些BeanPostProcessor；这使得BeanPostProcessor可以向普通bean一样定义；
6. 使用编程方式注册：可以通过ConfigurableFactoryBean的setBeanPostProcessor()方法注册BeanPostProcessor，但是通过该方法注册后，调用顺序只由注册先后顺序决定，不由Ordered接口决定；使用该种方式注册的BeanPostProcessor比自动探测的BeanPostProcessor先执行；
7. 无法使用AOP：BeanPostProcessor是直接在容器启动时被加载，然后通过BeanPostProcessor处理后续的Bean；并且AOP的自动代理功能本身也是用BeanPostProcessor方式实现，所以无论是BeanPostProcessor以及所依赖的Bean都无法使用自动代理；

### 示例：自定义BeanPostProcessor

1. BeanPostProcessor定义：定义了一个BeanPostProcessor，在每个Bean初始化后，打印Bean的名称；

|  |
| --- |
|  |

1. 配置文件：因为使用自动探测，故BeanPostProcessor不需要定义name属性；

|  |
| --- |
|  |

1. 启动：使用groovy和java混编，spring调用groovy定义的bean；

|  |
| --- |
|  |

1. 结果：

|  |
| --- |
|  |

### 示例：RequiredAnnotationBeanPostProcessor

使spring支持注解配置方式；

## 定制元数据：BeanFactoryPostProcessor

|  |
| --- |
|  |

|  |
| --- |
|  |

1. BeanFactoryPostProcessor用于在IOC容器初始化所有其它Bean之前，修改元数据；
2. 可以同时配置多个BeanFactoryPostProcessor，其调用顺序由Ordered接口决定；自定义BeanFactoryPostProcessor如果需要控制顺序，则需要实现Ordered接口；
3. BeanFactoryPostProcessor只在当前容器生效，如果一个应用存在多个容器，比如存在层次的容器（spring mvc以spring容器作为父容器），BeanFactoryPostProcessor也只会在自己所在容器中生效；
4. 配置好的BeanFactoryPostProcessor将会自动被容器探测并在合适的时候执行；spring本身提供了许多BeanFactoryPostProcessor，比如：PropertyOverrideConfigurer，PropertyPlaceholderConfigurer，自定义属性编辑器等；
5. 注意：在BeanFactoryPostProcessor中依赖其它Bean，违背了bean生命周期，可能造成二义性；比如：bean本身需要使用BeanFactoryPostProcessor定制行为，同时又被BeanFactoryPostProcessor依赖，形成悖论；

### 示例：PropertyPlaceHolderConfigurer

1. 该类用于将xml中的变量替换为properties文件中的属性，从而减少xml文件的复杂度；最常用的地方是使用jdbc.properties文件中的属性替换数据源中的配置；

|  |
| --- |
| Xml配置    jdbc.properties文件内容 |

1. 通过context命名空间的标签，可以简化上述配置，多个配置文件以逗号分隔；

|  |
| --- |
| **<context:property-placeholder location**=**"classpath:com/foo/jdbc.properties"/>** |

### 示例：PropertyOverrideConfigurer

TODO

|  |
| --- |
|  |

## 定制Bean初始化逻辑：FactoryBean

|  |
| --- |
|  |

1. 通过实现org.springframework.beans.factory.FactoryBean，能够定制特定bean的初始化逻辑；比如：ProxyFactoryBean，能够为Bean创建动态代理；
2. 接口提供3个方法：
   1. getObject()：创建一个实例对象并返回，该对象可以被共享，取决于是singleton还是prototype；
   2. isSingleton()：返回true，如果返回singletong对象；
   3. getObjectType()：返回被创建对象的类型；
   4. 使用getBean()加&,表示返回FacotryBean对象本身；不加，则表示返回对象；