spring学习笔记—基于JavaConfig的配置方式

|  |  |
| --- | --- |
|  | 🢂 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 基于Java的容器化配置 3](#_Toc478755600)

[1.1 基本概念 3](#_Toc478755601)

[1.2 AnnotationConfigApplicationContext 3](#_Toc478755602)

[1.3 AnnotationConfigWebApplicationConext 4](#_Toc478755603)

[1.4 @Bean 5](#_Toc478755604)

[1.5 @Configuration 6](#_Toc478755605)

[1.5.1 lookup配置 6](#_Toc478755606)

[1.6 @Import 6](#_Toc478755607)

[1.7 @ImportResource 6](#_Toc478755608)

[1.8 @Profile 6](#_Toc478755609)

# 基于Java的容器化配置

## 基本概念

1. @Bean：与xml文件中的<bean>扮演相同的角色；用于实例化、配置和初始化一个需要被IOC容器托管的对象；通常@Bean和@Configuration搭配使用；
2. @Configuration：标识一个类的基本目的是为了定义Bean；@Configuration同时允许其内部配置的bean相互调用；

基本使用示例：

|  |
| --- |
|  |

等价于xml的如下配置：

|  |
| --- |
|  |

建议：

@Bean可以脱离@Configuration单独使用，此时，类内部定义的Bean不可以依赖；建议使用基于JavaConfig的配置方式；

## AnnotationConfigApplicationContext

1. AnnotationConfigApplicationContext可以使用@Configuration作为输入，为@Configuration以及其包含的@Bean生成Bean定义；

|  |
| --- |
|  |

1. 可以使用@Component和JSR-330定义的Bean作为输入；

|  |
| --- |
|  |

1. 通过register方法注册配置；

|  |
| --- |
|  |

1. 通过scan方法自动扫描配置；

|  |
| --- |
|  |

|  |
| --- |
|  |

等价于：

|  |
| --- |
|  |

## AnnotationConfigWebApplicationConext

AnnotationConfigWebApplicationContext是WebApplicationContext的变体；能够通过ContextLoaderListener或者DispatchServlet进行配置，配置示例如下：

|  |
| --- |
|  |

## @Bean

1. @Bean用于在@Configuration或者@Component中定义bean对象；
2. @Bean标记于方法上，相当于xml配置中的<bean/>；
3. @Bean支持生命周期回调函数；比如：@PostConstruct，@PreDestory；
4. 支持各类Aware接口；
5. 使用@Bean声明BeanPostProcessor、BeanFactoryPostProcessor、FactoryBean等时，需要将方法声明为static，因为这些Bean先于配置类初始化。

## @Configuration

用于标记一个类为spring配置类；

### lookup配置

用于singlton域依赖prototype域的场景；

|  |
| --- |
| @Bean   @Scope("prototype")  **public** AsyncCommand asyncCommand() {  AsyncCommand command = **new** AsyncCommand();  *// inject dependencies here as required*  **return** command;  }  @Bean  **public** CommandManager commandManager() {  *// return new anonymous implementation of CommandManager with command() overridden // to return a new prototype Command object*  **return new** CommandManager() {  **protected** Command createCommand() { **return** asyncCommand(); }  }  } |

## @Import

用于加载其它配置类，从而在初始化容器时，减少加载配置类的数量；

## @ImportResource

加载xml配置文件；

## @Profile

1. 用于区分执行环境，比如：开发环境和线上环境使用不同的配置类；
2. 基于@Conditional注解实现；
3. @Conditional对应的Condition实现类，会在@Bean注册之前执行；