第四章：Spring Bean 基础

小马哥 · mercyblitz

极客时间

扫码试看/订阅

《小马哥讲 Spring 核心编程思想》视频课程

Spring Bean 基础

1. 定义 Spring Bean
2. BeanDefinition 元信息
3. 命名 Spring Bean
4. Spring Bean 的别名
5. 注册 Spring Bean
6. 实例化 Spring Bean
7. 初始化 Spring Bean
8. 延迟初始化 Spring Bean
9. 销毁 Spring Bean 如线程池
10. 垃圾回收 Spring Bean
11. 面试题精选

定义 Spring Bean

- 什么是 BeanDefinition?
- BeanDefinition 是 Spring Framework 中定义 Bean 的配置元信息接口，包含：
  - Bean 的类名
    全限定名
  - Bean 行为配置元素，如作用域、自动绑定的模式，生命周期回调等
  - 其他 Bean 引用，又可称作合作者（collaborators）或者依赖（dependencies）
  - 配置设置，比如 Bean 属性（Properties）

BeanDefinition 元信息

- BeanDefinition 元信息

属性（Property）	说明
Class	Bean 全类名，必须是具体类，不能用抽象类或接口
Name	Bean 的名称或者 ID
Scope	Bean 的作用域（如：singleton、prototype 等）
Constructor arguments	Bean 构造器参数（用于依赖注入）
Properties	Bean 属性设置（用于依赖注入） Setter 注入/参数注入
Autowiring mode	Bean 自动绑定模式（如：通过名称 byName）
Lazy initialization mode	Bean 延迟初始化模式（延迟和非延迟）
Initialization method	Bean 初始化回调方法名称
Destruction method	Bean 销毁回调方法名称

BeanDefinition 元信息

- BeanDefinition 构建
  - 通过 BeanDefinitionBuilder
  - 通过 AbstractBeanDefinition 以及派生类

命名 Spring Bean

- Bean 的名称 自定义或者利用自动生成

  每个 Bean 拥有一个或多个标识符（identifiers），这些标识符在 Bean 所在的容器必须是唯一的。通常，一个 Bean 仅有一个标识符，如果需要额外的，可考虑使用别名（Alias）来扩充。

  bean 标识符在所在容器(如 BeanFactory)是唯一的，不是在整个应用中唯一

  在基于 XML 的配置元信息中，开发人员可用 id 或者 name 属性来规定 Bean 的 标识符。通常 Bean 的 标识符由字母组成，允许出现特殊字符。如果要想引入 Bean 的别名的话，可在 name 属性使用半角逗号（“,”）或分号（“;”）来间隔。

  Bean 的 id 或 name 属性并非必须制定，如果留空的话，容器会为 Bean 自动生成一个唯一的名称。Bean 的命名尽管没有限制，不过官方建议采用驼峰的方式，更符合 Java 的命名约定。

命名 Spring Bean

- Bean 名称生成器（BeanNameGenerator）不需要过度关注实现细节，了解一下位置和基本使用方略
- 由 Spring Framework 2.0.3 引入，框架内建两种实现：

  DefaultBeanNameGenerator：默认通用 BeanNameGenerator 实现

- AnnotationBeanNameGenerator：基于注解扫描的 BeanNameGenerator 实现，起始于 Spring Framework 2.5，关联的官方文档：

  With component scanning in the classpath, Spring generates bean names for unnamed components, following the rules described earlier: essentially, taking the simple class name and turning its initial character to lower-case. However, in the (unusual) special case when there is more than one character and both the first and second characters are upper case, the original casing gets preserved. These are the same rules as defined by java.beans.Introspector.decapitalize (which Spring uses here).

Spring Bean 的别名

- Bean 别名（Alias）的价值
  - 复用现有的 BeanDefinition 必须要已有名字
  - 更具有场景化的命名方法，比如：

```
<alias name="myApp-dataSource" alias="subsystemA-dataSource"/>
<alias name="myApp-dataSource" alias="subsystemB-dataSource"/>
```

bean 别名用法与正常 bean 相同

注册 Spring Bean

- BeanDefinition 注册
  - XML 配置元信息
    - **<bean name=”...” ... />**
  - Java 注解配置元信息
    - **@Bean**
    - **@Component**
    - **@Import**
  - Java API 配置元信息
    - 命名方式：**BeanDefinitionRegistry#registerBeanDefinition(String,BeanDefinition)**
    - 非命名方式：**BeanDefinitionReaderUtils#registerWithGeneratedName(AbstractBeanDefinition,BeanDefinitionRegistry)**
    - 配置类方式：**AnnotatedBeanDefinitionReader#register(Class...)**

注册 Spring Bean

- 外部单例对象注册
  - Java API 配置元信息
    - **SingletonBeanRegistry#registerSingleton**

实例化 Spring Bean

- Bean 实例化（Instantiation）
  - 常规方式　参见 src/main/resources/META-INF/bean-instantiation-context.xml
    - 通过构造器（配置元信息：XML、Java 注解和 Java API ）
    - 通过静态工厂方法（配置元信息：XML 和 Java API ）
    - 通过 Bean 工厂方法（配置元信息：XML和 Java API ）
    - 通过 **FactoryBean**（配置元信息：XML、Java 注解和 Java API ）
  - 特殊方式
    - 通过 **ServiceLoaderFactoryBean**（配置元信息：XML、Java 注解和 Java API ）
    - 通过 **AutowireCapableBeanFactory#createBean(java.lang.Class, int, boolean)**
    - 通过 **BeanDefinitionRegistry#registerBeanDefinition(String,BeanDefinition)**

初始化 Spring Bean

- Bean 初始化（Initialization）
  - @PostConstruct 标注方法
  - 实现 InitializingBean 接口的 afterPropertiesSet() 方法
  - 自定义初始化方法
    - XML 配置：<bean init-method=”init” ... />
    - Java 注解：@Bean(initMethod=”init”)
    - Java API：AbstractBeanDefinition#setInitMethodName(String)

思考：假设以上三种方式均在同一 Bean 中定义，那么这些方法的执行顺序是怎样？

初始化顺序：@PostConstruct --> afterPropertiesSet() --> 自定义初始化方法

延迟初始化 Spring Bean

- Bean 延迟初始化（Lazy Initialization）
  - XML 配置：<bean lazy-init=”true” ... />
  - Java 注解：@Lazy(true)

思考：当某个 Bean 定义为延迟初始化，那么，Spring 容器返回的对象与非延迟的对象存在怎样的差异？

销毁 Spring Bean

此处销毁是指: 在 gc 之后，销毁 bean

- Bean 销毁（Destroy）
  - @PreDestroy 标注方法
  - 实现 DisposableBean 接口的 destroy() 方法
  - 自定义销毁方法
    - XML 配置：<bean destroy=”destroy” ... />
    - Java 注解：@Bean(destroy=”destroy”)
    - Java API：AbstractBeanDefinition#setDestroyMethodName(String)

思考：假设以上三种方式均在同一 Bean 中定义，那么这些方法的执行顺序是怎样？

垃圾回收 Spring Bean

- Bean 垃圾回收（GC）

1. 关闭 Spring 容器（应用上下文）
2. 执行 GC
3. Spring Bean 覆盖的 finalize() 方法被回调

面试题

沙雕面试题 - 如何注册一个 Spring Bean?

答：通过 BeanDefinition 和外部单体对象来注册

regisyty SingletonBeanRegistry

面试题

996 面试题 - 什么是 Spring BeanDefinition?

答：回顾“定义 Spring Bean” 和 “BeanDefinition 元信息”

面试题

劝退面试题 - Spring 容器是怎样管理注册 Bean

答：答案将在后续专题章节详细讨论，如：IoC 配置元信息读取和解析、依赖查找和注入以及 Bean 生命周期等。

极客时间

扫码试看/订阅

《小马哥讲 Spring 核心编程思想》视频课程