spring学习笔记—Bean的生命周期管理

|  |  |
| --- | --- |
|  | 🢂 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 如何管理Bean? 3](#_Toc478810375)

[1.1 如何指定Bean的生命周期回调函数？ 3](#_Toc478810376)

[1.1.1 初始化回调函数的实现方式有哪些？ 3](#_Toc478810377)

[1.1.2 销毁回调函数的实现方式有哪些？ 3](#_Toc478810378)

[1.1.3 如何指定默认回调函数？ 3](#_Toc478810379)

[1.1.4 回调函数的执行顺序是怎样的？ 4](#_Toc478810380)

[1.1.5 LifeCycle和LifecycleProcessor 4](#_Toc478810381)

[1.2 如何在非web项目中优雅关闭spring容器？ 5](#_Toc478810382)

[1.3 Aware接口 5](#_Toc478810383)

[1.3.1 ApplicationContextAware 5](#_Toc478810384)

[1.3.2 BeanNameAware 6](#_Toc478810385)

[1.3.3 其它Aware接口 6](#_Toc478810386)

# 如何管理Bean?

Spring使用BeanPostProcessor去处理所有生命周期相关的回调函数；如果需要定制bean的生命周期，则需要自己实现BeanPostProcessor；

## 如何指定Bean的生命周期回调函数？

指定生命周期回调函数包括如下3种方式：

* 1. 方式1：定制bean的生命周期，可以通过实现InitializingBean和DisposableBean两个接口实现；容器会调用afterPropertiesSet()和destory()方法来定制初始化和销毁动作；
  2. 方式2：使用@PostConstruct和@PreDestory注解定制bean的生命周期是一种最佳实践，相比实现接口来说，使用注解能够减少与spring接口的耦合；
  3. 方式3：使用xml配置文件中的init-method和destory-method；

1. 通过实现LifeCycle接口，能够定制bean的启动、停止操作；

### 初始化回调函数的实现方式有哪些？

1. 实现InitializingBean接口；
2. 使用@PostConstruct注解；（推荐）

3）在xml中指定init-method；

### 销毁回调函数的实现方式有哪些？

1. 实现DisposableBean接口；
2. 使用@PreDestroy注解；（推荐）

3）在xml中指定destory-method；

### 如何指定默认回调函数？

|  |
| --- |
|  |

### 回调函数的执行顺序是怎样的？

1. 如果对于bean同时使用了多种方式定制生命周期，且每种方式调用的方法不同，则会按照顺序调用所有方法；如果方法相同，则只会调用一次；
2. 初始化调用顺序如下：
   1. @PostConstruct注解指定方法；
   2. InitializingBean的afterPropertiesSet()方法；
   3. 通过init-method或者default-init-method配置的init方法；
3. 销毁调用顺序如下：
   1. @PreDestory注解指定方法；
   2. DisposableBean的destory()方法；
   3. 通过destory-method或者default-destory-method配置的destory方法；

### LifeCycle和LifecycleProcessor

1. LifeCycle接口用于定制bean在容器收到启动、停止信号时的行为；
2. LifecycleProcessor作为LifeCycle子接口，用于定制容器刷新和关闭时，bean的行为；
3. SmartLifecycle接口作为LifeCycle，用于定制容器自动启动或者停止时，bean的行为；区别于收到启动、停止信号的动作；
4. Phase接口，用于指定实现类执行的先后顺序，当启动时，容器优先调用phased值最小的方法，销毁时按照相反的顺序进行；比如：当涉及到依赖关系时，比如使用depend-on配置，则在初始化时，需要先调用被依赖bean的初始化方法；在销毁时，需要先调用依赖bean的销毁方法；当启动时，容器优先调用phased值最小的方法，销毁时按照相反的顺序进行；

|  |
| --- |
| **public interface** SmartLifecycle **extends** Lifecycle, Phased {  **boolean** isAutoStartup();  **void** stop(Runnable callback);  } |

1. 任何实现SmartLifecycle接口的类，都需要调用stop方法中的Runnable对象的run方法，因而可以实现异步调用stop动作；比如：实现类DefaultLifecycleProcessor接口调用stop直到超时后，再调用下一个stop方法，默认超时时间为30s；可以通过xml进行超时时间配置：

|  |
| --- |
|  |

## 如何在非web项目中优雅关闭spring容器？

|  |
| --- |
|  |

## Aware接口

### ApplicationContextAware

1. 当一个实例实现了ApplicationContextAware接口以后，容器将会把ApplicationContext实例通过setter方法注入给实例；
2. 该种方法允许在程序中以编程方式访问spring上下文，但是会增加与spring的耦合；

|  |
| --- |
|  |

### BeanNameAware

1. 当一个实例实现了BeanNameAware接口，IOC容器将会提供指定name的Bean实例；
2. 该回调函数将会在bean调用生命周期回调函数之前调用；

|  |
| --- |
|  |

### 其它Aware接口

|  |  |
| --- | --- |
| **Aware接口** | **描述** |
| ApplicationContextAware | 注入spring上下文对象 |
| ApplicationEventPublisherAware | 围绕spring上下文的事件发布器 |
| BeanClassLoaderAware | 加载Bean的类加载器 |
| BeanFactoryAware | 注入BeanFactoryAware |
| BeanNameAware | 注入指定名称的Bean |
| BootstrapContextAware | 注入资源加载器BootstrapContext |
| LoadTimeWeaverAware | 加载class过程中，织入class定义 |
| MessageSourceAware | 配置资源解析策略，用以支持参数化和国际化 |
| NotificationPublisherAware | Spring JMX通知发布器 |
| PortletConfigAware | PortletConfig对象 |
| PortletContextAware | PortletContext 对象 |
| ResourceLoaderAware | 偏底层的资源加载器 |
| ServletConfigAware | ServletConfig对象 |
| ServletContextAware | ServletContext对象 |

注：

上面的Aware接口会增加与spring容器的耦合，所以建议只在基础Bean中使用，用于直接访问spring上下文的场景；