spring学习笔记—Scope介绍

|  |  |
| --- | --- |
|  | 🢂 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 Scope 3](#_Toc478742960)

[1.1 singleton 3](#_Toc478742961)

[1.1.1 与单例模式的区别 3](#_Toc478742962)

[1.2 prototype 3](#_Toc478742963)

[1.2.1 包含prototype bean的singleton bean 4](#_Toc478742964)

[1.3 如何初始化web应用？ 4](#_Toc478742965)

[1.3.1 初始化web应用 4](#_Toc478742966)

[1.3.2 生命周期长的bean使用生命周期短的bean 5](#_Toc478742967)

[1.3.3 选择代理方式 5](#_Toc478742968)

[1.4 定制scope 6](#_Toc478742969)

# Scope

|  |  |
| --- | --- |
| **Scope** | **描述** |
| singleton | 默认，每个容器中仅定义一个bean实例，由容器托管； |
| prototype | 每一次注入都新建一个bean实例； |
| request | web应用使用，bean生命周期贯穿http的一次请求；每次请求都会新建一个bean实例； |
| session | web应用使用，bean生命周期贯穿http session；每个session新建一个bean实例； |
| global session | web应用使用，bean生命周期贯穿http global session；一个global session使用一个； |
| application | web应用使用，bean生命周期贯穿ServletContext；一个context使用一个； |

## singleton

|  |
| --- |
|  |

### 与单例模式的区别

1. 单例模式硬编码，控制实例的数量；
2. singleton控制在spring容器中实例的数量，适合于无状态的bean的管理；

## prototype

1. 适合于有状态的bean的管理；
2. 每次注入都会新建一个；
3. spring并不管理prototype的全部生命周期，只负责bean的初始化、配置、组装和注入；bean实例的释放需要在调用代码中自行实现；bean配置的destory回调函数将不会被调用；

### 包含prototype bean的singleton bean

使用注入属性注入方式，将prototype bean注入到singleton bean中，则singleton只与一个prototype bean关联；因为注入只发生在singleton bean创建时这一次；如果singleton中的prototype bean需要每次注入都新建一个，则推荐使用方法注入实现；

|  |
| --- |
|  |

## 如何初始化web应用？

### 初始化web应用

1. 要使用web相关的scope需要一些初始化配置；
2. 配置依据使用不同的Servlet环境而不同；如果使用spring mvc，则web相关scope将由DispatcherServlet或者DispatcherPortlet自动维护；
3. 对于不使用spring mvc的web应用而言，需要配置org.springframework.web.context.request.RequestContextListener
4. 可以通过两种方式配置：
   1. WebApplicationInitializer实例，在程序中指定；
   2. 在web.xml中指定，可以配置监听器或者过滤器；

|  |
| --- |
|  |

### 生命周期长的bean使用生命周期短的bean

比如，需要将request bean注入到session bean中，为保证周期安全，需要创建request bean的AOP代理类，然后注入到session bean中，即可以代理被注入bean的操作，又可以在需要的时候，重新恢复被代理对象；示例如下：

|  |
| --- |
|  |

### 选择代理方式

* 1. spring默认使用基于CGLIB的方式创建动态代理，使用<aop:scoped-proxy/>

标签标记bean需要创建代理；

* 1. CGLIB只能代理public方法，非public方法无法代理；

也可以使用jdk原生的基于接口的动态代理，通过在标签中添加proxy-target-class =false但是要求被代理的bean必须实现至少一个接口，且接口中的方法符合代理需求；示例如下：

|  |
| --- |
|  |

## 定制scope

1. spring允许重写已有scope或者自定义新的scope;
2. 创建scope需要实现org.springframework.beans.factory.config.Scope；接口包含4个方法，用于从scope中创建、删除、销毁对象，
   1. Object get(String name, ObjectFactory objectFactory)：获取bean对象；
   2. Object remove(String name)：移除bean；
   3. **void** registerDestructionCallback(String name, Runnable destructionCallback) ：当scope中对象销毁时，调用；
   4. String getConversationId()：存放对话ID，标识每个scope；
3. 实现scope接口后，需要向IOC容器中注册scope；包括编程式和声明式两种方式；
4. 编程式使用方法void registerScope(String scopeName, Scope scope); 方法在ConfigurableBeanFactory 接口中声明,能够在大多数ApplicationContext实现类中使用；方法的第一个参数是scope的名称，第二个参数是scope的实现类；

|  |
| --- |
|  |

1. 声明式可以直接在xml文件中配置；

|  |
| --- |
|  |