spring学习笔记—依赖注入

|  |  |
| --- | --- |
|  | 🢂 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 依赖注入 4](#_Toc478742161)

[1.1 spring注入方式有哪些？ 4](#_Toc478742162)

[1.1.1 构造注入 4](#_Toc478742163)

[1.1.2 setter注入 4](#_Toc478742164)

[1.1.3 静态工厂方法注入 4](#_Toc478742165)

[1.2 如何选择注入方式？ 5](#_Toc478742166)

[1.3 Bean的注入过程是怎样的？ 5](#_Toc478742167)

[1.3.1 如何解决环形依赖？ 6](#_Toc478742168)

[1.4 注入配置详解 6](#_Toc478742169)

[1.4.1 直接注入值 6](#_Toc478742170)

[1.4.1.1 普通方式 6](#_Toc478742171)

[1.4.1.2 p标签 7](#_Toc478742172)

[1.4.1.3 映射java.util.Properties对象 7](#_Toc478742173)

[1.4.2 注入引用对象 7](#_Toc478742174)

[1.4.3 子容器包装父容器bean 7](#_Toc478742175)

[1.4.4 内部bean 8](#_Toc478742176)

[1.4.5 集合 8](#_Toc478742177)

[1.4.5.1 合并父类bean的集合 9](#_Toc478742178)

[1.4.5.2 强类型集合 10](#_Toc478742179)

[1.4.6 空值和空字符串 11](#_Toc478742180)

[1.4.7 P标签 11](#_Toc478742181)

[1.4.8 C标签 12](#_Toc478742182)

[1.4.9 嵌套属性名 12](#_Toc478742183)

[1.4.10 depends-on属性 13](#_Toc478742184)

[1.4.11 延迟初始化 13](#_Toc478742185)

[1.5 什么是自动装配？ 13](#_Toc478742186)

[1.5.1 自动装配的优点： 13](#_Toc478742187)

[1.5.2 自动装配的缺点 14](#_Toc478742188)

[1.5.3 设置bean不参与自动装配 14](#_Toc478742189)

[1.6 如何解决singleton依赖proto域Bean的问题？ 14](#_Toc478742190)

[1.6.1 lookup方法注入 15](#_Toc478742191)

[1.7 如何重写bean的方法？ 16](#_Toc478742192)

# 依赖注入

1. 注入解耦对象的创建和使用，使得对象在使用关联对象时，不必关注和维护其创建过程；
2. 依赖注入通常有：构造注入、setter注入、静态工厂方法；

## spring注入方式有哪些？

### 构造注入

1. 通过调用含参构造器向bean中注入依赖的bean，每个参数都是一个依赖；
2. 对应构造参数为基础数据类型，可以直接注入值；
3. 如果不指定顺序，则按照配置文件中参数顺序进行注入；适用于存在多个类型相同的参数的场景；
4. 可以在配置时指定绑定的参数的名称，但是需要在编译时开启debug模式，或者通过@ConstructorProperties标签指定参数名；

### setter注入

setter注入是在调用bean对象的无参构造器或者无参工厂方法后，调用setter方法进行的注入；

### 静态工厂方法注入

静态工厂方法注入能够替代构造注入；允许对私有构造器的bean进行注入；静态工厂方法注入对象的类型并不一定要和实际包含对象的类型一致；

|  |
| --- |
|  |

## 如何选择注入方式？

1. 构造注入和setter注入可以同时使用，构造注入适合于强制注入的场景，而setter适合于选择性注入的场景；通过在setter方法上添加@required注解能够标识属性为必需注入的属性；
2. Spring团队倡导优先使用构造注入，因为构造注入能够将组件变为不可变对象，同时确保必需注入的属性非空；同时，如果构造注入的参数很多，也可以用来判断类的职责是否过多，可以推动重构进行；
3. Setter注入仅在需要进行选择性注入的场景下使用，能够为对象提供默认值；setter注入的好处是能够将注入延迟，不必在创建bean对象时就进行注入；

## Bean的注入过程是怎样的？

1. 通过元数据创建和初始化ApplicationContext；
2. 每个bean注入的对象都以属性、构造器参数、静态工厂方法参数形式进行注入；
3. 每个属性或者构造器参数可以被定义为一个值的集合或者一个对容器中bean对象的引用；
4. 值类型的属性或者构造器参数将被IOC自动转换为java内建类型；比如：int,long,String,boolean等；
5. spring会在容器创建的时候就对所有bean的配置进行校验，但bean属性的注入直到其真正被创建后，才会注入；
6. singleton域或者预初始化的bean对象都会在容器创建时，被初始化；其它bean将会延迟在被用到时才创建和初始化；
7. 一个bean对象的创建会引起一个bean对象图的创建，包括直接依赖的bean以及bean属性依赖的bean；

### 如何解决环形依赖？

1. 使用构造注入有可能产生环形依赖的场景，IOC会在运行时检测环形依赖，如果存在，则抛出BeanCurrentlyInCreationException异常；
2. 避免环形依赖导致的问题，可以使用setter注入，setter注入允许存在环形注入；
3. 构造注入产生环形依赖的场景是：两个对象在创建时，都依赖彼此，形成矛盾，故不允许；
4. setter注入产生环形依赖的场景是：两个对象互相调用彼此的方法，这种情况属于双向关联关系，允许存在；

## 注入配置详解

通过xml注入主要通过<property>和<constructor-arg>两个标签；

### 直接注入值

通过在xml配置文件中写入字符串值的方式注入；spring会根据默认约定将字符串转换为具体的bean对象；

#### 普通方式

|  |
| --- |
|  |

#### p标签

|  |
| --- |
|  |

#### 映射java.util.Properties对象

|  |
| --- |
|  |

### 注入引用对象

|  |
| --- |
|  |

### 子容器包装父容器bean

当系统中同时存在父容器和子容器时，可以通过parent属性包装父容器中的bean到子容器中使用；子容器中的同名bean相当于是一个代理；

|  |
| --- |
|  |

### 内部bean

可以直接在<properties>标签中定义<bean>，称为内部bean；内部bean仅可以被外层bean使用，不需要id/name/scope属性；

|  |
| --- |
|  |

### 集合

list/map/set/props标签对应List/Map/Set/Properties类型集合；集合中的value可以嵌套取值，类型包括：bean | ref | idref | list | set | map | props | value | null；

|  |
| --- |
|  |

#### 合并父类bean的集合

bean配置中abstract属性为true，表示当前bean的配置是一个抽象配置，仅用于抽取存放某一类bean的公共配置；当子类中的集合merge属性为true时，子bean会覆盖父bean中同名的元素值；

|  |
| --- |
|  |

#### 强类型集合

|  |
| --- |
|  |

### 空值和空字符串

1. 空字符串

|  |
| --- |
|  |

1. 空值

|  |
| --- |
|  |

### P标签

p标签用来简化<property>标签的配置；

**注：**当属性值为引用时，需要在属性后面跟-ref，表明是引用名；使用p标签能够简化xml配置，但是会和以ref结尾的bean冲突；需要事先约定好命名规范；

|  |
| --- |
|  |

### C标签

C标签用来简化<constructor-arg>的配置;

|  |
| --- |
|  |

### 嵌套属性名

|  |
| --- |
|  |

### depends-on属性

1. 有些场景下，依赖关系并不会通过配置文件显式给出，但是一个bean的初始化需要等待某个bean初始化完成后才能进行；depends-on属性提供了对该功能的支持，能够强制约束bean在初始化一个或多个指定的bean后，进行初始化；
2. 在容器销毁时，会先销毁带有depends-on配置的bean，因而能够控制销毁对象的顺序；

|  |
| --- |
|  |

### 延迟初始化

1. 默认情况下，ApplicationContext启动时只会创建并配置singleton域的bean，作为初始化流程的一部分；预初始化能够在一开始检查出配置和周边环境的问题；
2. 在有些场景下，可能会要求bean延迟到第一次使用时在进行初始化，比如：某个占用资源的大对象，只在真正使用时候，才去创建，避免过早的占用资源；
3. lazy-init属性用来指定bean是否延迟初始化；
4. 当指定lazy-init属性的bean被未指定该属性的singleton域的bean依赖，则lazy-init属性无效；
5. 可以通过在<beans>标签中指定default-lazy-init属性来指定其中所有bean默认的lazy-init属性；

|  |
| --- |
|  |

## 什么是自动装配？

### 自动装配的优点：

1. 减少注入配置参数；
2. 能够在对象演进过程中方便更新；
3. 装配模式分为4种：

|  |  |
| --- | --- |
| **模式** | **描述** |
| no | 默认，不进行自动注入； |
| byName | 通过属性名称注入； |
| byType | 通过属性类型注入；适合一种类型仅有一个bean的情况；   1. 当前类型在容器中超过1个bean时，报错； 2. 容器中没有当前类型bean时，无操作； |
| constructor | 构造注入，类似于byType；  如果没有和构造参数类型相同的bean存在，则报错； |

### 自动装配的缺点

1. 自动注入会被明确依赖关系的精确注入覆盖；
2. 不能注入基础数据类型，如：数值类型、字符串等；
3. 注入关系不明确，容易引起二义性；
4. 不便于通过工具生成文档；

解决措施：

1. 放弃使用自动装配，使用精确注入代替；
2. 避免使用注入，将autowire-candidate属性设置为false;
3. 通过primary属性指定一个基础候选bean；
4. 实现更加细粒度的注入控制；

### 设置bean不参与自动装配

通过将<bean>中的autowire-candidate属性设置为false，能够使当前bean不参与自动装配；也可以通过<beans>中的default-autowire-candidates指定一个或多个注入匹配的模式字符串；

设置bean不参与自动装配，指的是bean不能被当做自动装配的候选，但bean本身装配时，仍然可以使用自动装配；

## 如何解决singleton依赖proto域Bean的问题？

问题：正常情况下，对于相同生命周期的bean，在注入时，直接使用被依赖bean作为依赖bean的属性；但是，对于生命周期不同的bean，比如：一个singlton的bean在每一次方法调用时，都会使用一个非singleton（prototype）的bean；而通过属性注入的bean，只有一个；

解决方法：

1. 放弃部分注入特性，方法执行时，每次通过ApplicationContext的getBean(beanName)方法获取prototype的对象；缺点：增加了和spring框架的耦合；

|  |
| --- |
|  |

1. 方法注入：

方法注入的使用动机：<https://spring.io/blog/2004/08/06/method-injection/>

### lookup方法注入

lookup方法注入，允许重写容器中bean的方法，并返回指定bean作为结果；spring通过使用CGLIB库的字节码生成技术来动态生成重写父类方法的子类；

|  |
| --- |
| 注：lookup方法注入，使用注意事项   1. 被重写的类和方法不能够被标记为final类型； 2. 单元测试时，需要通过stub自己实现抽象方法； 3. lookup方法不能够作用于工厂方法或标记为@Bean的方法；因为容器无法控制bean实例的创建，因而，也就不能创建bean的子类； 4. 使用方法注入的bean不能被序列化； |

示例：

|  |
| --- |
|  |

**注：**上述示例中，如果command的scope为singleton，则commandManager的createCommand方法每次返回的都是同一个bean；

## 如何重写bean的方法？

spring允许对被管理bean中的方法进行重写，使用其它方法进行替换；

示例：

|  |
| --- |
| 待替换方法的bean：  重写方法的bean，需要实现org.springframework.beans.factory.support.MethodReplacer  接口    xml配置： |