|  |
| --- |
| Bean的作用域、生命周期 |

* Bean的作用域

 　　Spring 3中为Bean定义了5中作用域，分别为singleton（单例）、prototype（原型）、request、session和global session，5种作用域说明如下：

1. singleton：单例模式，Spring IoC容器中只会存在一个共享的Bean实例，无论有多少个Bean引用它，始终指向同一对象。Singleton作用域是Spring中的缺省作用域，也可以显示的将Bean定义为singleton模式，配置为：
   * **<bean id="userDao" class="com.ioc.UserDaoImpl" scope="singleton"/>**
2. prototype:**原型模式，**每次通过Spring容器获取prototype定义的bean时，容器都将创建一个新的Bean实例，每个Bean实例都有自己的属性和状态，而singleton全局只有一个对象。

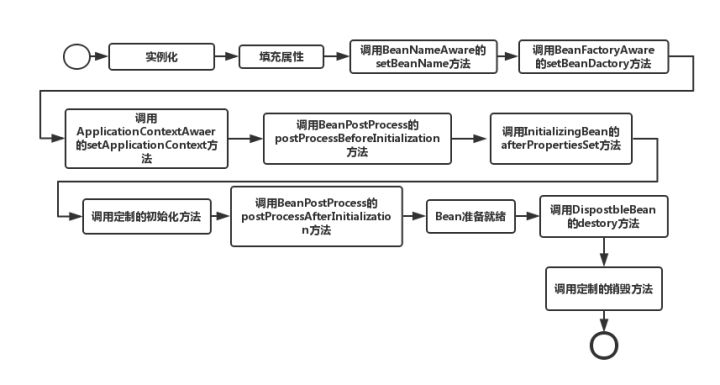
根据经验，对有状态的bean使用prototype作用域，而对无状态的bean使用singleton作用域。

1. request：在一次Http请求中，容器会返回该Bean的同一实例。而对不同的Http请求则会产生新的Bean，而且该bean仅在当前Http Request内有效。
   * <bean id="loginAction" class="com.cnblogs.Login" scope="request"/>,针对每一次Http请求，Spring容器根据该bean的定义创建一个全新的实例，且该实例仅在当前Http请求内有效，而其它请求无法看到当前请求中状态的变化，当当前Http请求结束，该bean实例也将会被销毁。
2. session：在一次Http Session中，容器会返回该Bean的同一实例。而对不同的Session请求则会创建新的实例，该bean实例仅在当前Session内有效。
   * <bean id="userPreference" class="com.ioc.UserPreference" scope="session"/>,同Http请求相同，每一次session请求创建新的实例，而不同的实例之间不共享属性，且实例仅在自己的session请求内有效，请求结束，则实例将被销毁。
3. global Session：在一个全局的Http Session中，容器会返回该Bean的同一个实例，仅在使用portlet context时有效。

* Bean的生命周期

　　经过如上对Bean作用域的介绍，接下来将在Bean作用域的基础上讲解Bean的生命周期。

　　Spring容器可以管理singleton作用域下Bean的生命周期，在此作用域下，Spring能够精确地知道Bean何时被创建，何时初始化完成，以及何时被销毁。而对于prototype作用域的Bean，Spring只负责创建，当容器创建了Bean的实例后，Bean的实例就交给了客户端的代码管理，Spring容器将不再跟踪其生命周期，并且不会管理那些被配置成prototype作用域的Bean的生命周期。Spring中Bean的生命周期的执行是一个很复杂的过程，读者可以利用Spring提供的方法来定制Bean的创建过程。**Spring容器在保证一个bean实例能够使用之前会做很多工作：**



1.Spring对Bean进行实例化（相当于程序中的new Xx()）

2.Spring将值和Bean的引用注入进Bean对应的属性中, 也就是IOC注入

3.如果Bean实现了BeanNameAware接口，Spring将Bean的ID传递给setBeanName()方法  
**（实现BeanNameAware主要是为了通过Bean的引用来获得Bean的ID，一般业务中是很少有用到Bean的ID的**）

4.如果Bean实现了BeanFactoryAware接口，Spring将调用setBeanDactory(BeanFactory)方法**并把BeanFactory容器实例作为参数传入。**  
**（实现BeanFactoryAware 主要目的是为了获取Spring容器（为了获得容器，从而调用容器里的东西），如Bean通过Spring容器发布事件等）**传递的是Spring工厂本身（可以用这个方法获取到其他Bean）

5. **（可以用这个方法获取到其他Bean，项目中用到了，传回class）**如果Bean实现了ApplicationContextAwaer接口，Spring容器将调用setApplicationContext(ApplicationContext)方法， **(作用与BeanFactory类似都是为了获取Spring容器，如Bean通过Spring容器发布事件等，项目是调用容器里为Handler的bean。applicationContext**.getBeansOfType(EventHandler.**class**);

**取得所有的事件处理的classbean做进一步的消费者事件处理2），不同的是Spring容器在调用setApplicationContext方法时会把它自己作为setApplicationContext 的参数传入，而Spring容器在调用setBeanDactory前需要程序员自己指定（注入）setBeanDactory里的参数BeanFactory )** 传入Spring上下文，该方式同样可以实现步骤4，但比4更好，因为ApplicationContext是BeanFactory的子接口，有更多的实现方法.（如果应用Spring的工厂也就是BeanFactory的话去掉这一步就Ok了）

6.如果Bean实现了BeanPostProcess接口，**Spring将调用它们的postProcessBeforeInitialization（预初始化）方法**  
**（作用是在Bean实例创建成功后对进行增强处理，如对Bean进行修改，增加某个功能）**

7.如果Bean实现了**InitializingBean接口**，Spring将调用它们的afterPropertiesSet方法，作用与在配置文件中对**Bean使用init-method声明初始化的作用一样，都是在Bean的全部属性设置成功后执行的初始化方法**。

8.如果Bean实现了BeanPostProcess接口，**Spring将调用它们的postProcessAfterInitialization（后初始化）方法  
（作用与6的一样，只不过6是在Bean初始化前执行的，而这个是在Bean初始化后执行的，时机不同 )**

9.经过以上的工作后，**Bean准备就绪，**将一直驻留在应用上下文中**给应用使用，**直到应用上下文被销毁.

**当Bean不再需要时，会经过清理阶段。**

10.如果Bean实现了DispostbleBean接口，Spring将调用它的destory方法，11.如果在配置文件中对Bean使用destory-method属性，作用跟10是都一样，都是在Bean实例销毁前执行的方法。



**1. 实例化Bean**

对于**BeanFactory**容器，当客户向容器请求一个尚未初始化的bean时，或初始化bean的时候需要注入另一个尚未初始化的依赖时，容器就会调用createBean进行实例化。   
对于**ApplicationContext**容器，当容器启动结束后，便实例化所有的bean。   
容器通过获取BeanDefinition对象中的信息进行实例化。并且这一步仅仅是简单的实例化，并未进行依赖注入。   
实例化对象被包装在BeanWrapper对象中，BeanWrapper提供了设置对象属性的接口，从而避免了使用反射机制设置属性。

**2. 设置对象属性（依赖注入）**

实例化后的对象被封装在BeanWrapper对象中，并且此**时对象仍然是一个原生的状态，并没有进行依赖注入。**   
紧接着，Spring根据BeanDefinition中的信息进行依赖注入。   
并且通过BeanWrapper提供的设置属性的接口完成依赖注入。

**3. 注入Aware接口**

紧接着，Spring会检测该对象是否实现了xxxAware接口，**并将相关的xxxAware实例注入给bean。（自动装配过程）**

**4. BeanPostProcessor**

当经过上述几个步骤后，bean对象已经被正确构造，**但如果你想要对象被使用前再进行一些自定义的处理，就可以通过BeanPostProcessor接口实现**。   
该接口提供了两个函数：

* postProcessBeforeInitialzation( Object bean, String beanName )   
  当前正在初始化的bean对象会被传递进来，我们就可以对这个bean作任何处理。   
  **这个函数会先于InitialzationBean执行，因此称为前置处理。**所有Aware接口的注入就是在这一步完成的。
* postProcessAfterInitialzation( Object bean, String beanName )   
  当前正在初始化的bean对象会被传递进来，我们就可以对这个bean作任何处理。   
  **这个函数会在InitialzationBean完成后执行，因此称为后置处理。**

**5. InitializingBean与init-method**

当BeanPostProcessor的前置处理完成后就会进入本阶段。   
**InitializingBean接口只有一个函数：**

* **afterPropertiesSet()//项目中用到这个。则在注册bean时，同时执行**

这一阶段也可以在**bean正式构造完成前增加我们自定义的逻辑**，但它与前置处理不同，由于该函数并不会把当前bean对象传进来，因此在这一步没办法处理对象本身，只能增加一些额外的逻辑。   
若要使用它，我们需要让bean实现该接口，并把要增加的逻辑写在该函数中。然后Spring会在前置处理完成后检测当前bean是否实现了该接口，并执行afterPropertiesSet函数。

当然，Spring为了降低对客户代码的侵入性，给bean的配置提供了init-method属性，该属性指定了在这一阶段需要执行的函数名。Spring便会在初始化阶段执行我们设置的函数。init-method本质上仍然使用了InitializingBean接口（执行的初始化方法）。

**6. DisposableBean和destroy-method**

和init-method一样，通过给destroy-method指定函数，就可以在bean销毁前执行指定的逻辑。