





1.Bean的建立

Spring对Bean进行

实例化（相当

于程序中的new Xx()）

2.属性注入

Spring将值和Bean的引用注入进Bean对应的属性中

3.BeanNameAware的setBeanName方法：org.springframework.beans.BeanNameAware

如果Bean实现了BeanNameAware接口，Spring将Bean的ID传递给setBeanName()方法（实现BeanNameAware清主要是为了通过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.BeanFactoryAware的setBeanFactory方法,传入工厂自身

如果Bean实现了BeanFactoryAware接口，Spring将调用setBeanFactory(BeanFactory bf)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，如Bean通过Spring容器发布事件等）

5.如果Bean实现了ApplicationContextAwaer接口，Spring容器将调用setApplicationContext(ApplicationContext ctx)方法，把y应用上下文作为参数传入.(作用与BeanFactory类似都是为了获取Spring容器，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 )

6.BeanPostPrococeesor的postProcessBeforeInitialization方法 前置处理

如果Bean实现了BeanPostProcessor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.InitializingBean接口的afterPropertiesSet方法

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。

添加：调用<bean>的init-method属性指定的初始化方法 检查是否配置有自定义init-method，指定初始化后的调用方法

8.BeanPostProcessor的postProcessAfterInitialization方法 后置处理

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 )

添加执行InstantiationAwareBeanPostProcessor的postProcessAfterInitialization方法

注册必要的Destruction相关回调接口

使用中

9.经过以上的工作后，Bean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

容器初始化成功，执行正常调用后开始销毁容器

10.DispostbleBean接口的destroy方法

如果Bean实现了DispostbleBean接口，Spring将调用它的destory方法，作用与在配置文件中对Bean使用destory-method属性的作用一样，都是在Bean实例销毁前执行的方法。

添加：调用<bean>的destroy-method属性指定的初始化方法检查是否配置有自定义destroy-method，指定销毁对象前的方法

二、各种接口方法分类

Bean的完整生命周期经历了各种方法调用，这些方法可以划分为以下几类：

1、Bean自身的方法　　：　　这个包括了Bean本身调用的方法和通过配置文件中<bean>的init-method和destroy-method指定的方法

2、Bean级生命周期接口方法　　：　　这个包括了BeanNameAware、BeanFactoryAware、InitializingBean和DiposableBean这些接口的方法

3、容器级生命周期接口方法　　：　　这个包括了InstantiationAwareBeanPostProcessor 和 BeanPostProcessor 这两个接口实现，一般称它们的实现类为“后处理器”。

4、工厂后处理器接口方法　　：　　这个包括了AspectJWeavingEnabler, ConfigurationClassPostProcessor, CustomAutowireConfigurer等等非常有用的工厂后处理器　　接口的方法。工厂后处理器也是容器级的。在应用上下文装配配置文件之后立即调用。

被依赖的类先加载：

如果B依赖于A，A先被加载。

Constructor A

Constructor B

BeanNameAware B

BeanFactoryAware

ApplicationContextAware B

BeanPostProcessor Before: b前置处理

Initializing Bean B

BeanPostProcessor After b

BeanNameAware A

BeanFactoryAware A

Application Context Aware A

BeanPostProcessor Before: a

Initializing Bean A

BeanPostProcessor After a

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。

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()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。

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。

销毁方法未调用：debug模式jvm直接就关了，spring容器还没来得及销毁对象

解决：((ClassPathXmlApplicationContext)applicationContext).close();

单例（singleton）的bean才可以手动销毁

prototype类型的Bean完成实例化之后就由调用方去管理后续流程了，IoC容器不再管理；

五类生命周期：singleton（默认）,prototype,request,session,global session