**春天的杰作Spring源码分析**

主讲：Cat 老师

北京动力节点教育科技有限公司  
2020 • 北京

动力节点•版权所有•禁止传播

## 快速回顾

1. 源码环境构建
2. IOC容器源码剖析

spring-framework-5.2.6.RELEASE

spring-framework-5.1.8.RELEASE

比如：spring-framework-5.1.8.RELEASE

5：主版本号，当功能模块有较大更新或整体架构有巨大变化，主版本号会更新；

1：次版本号，次版本表示只是局部的一些变动；

8：修改版本号，一般是bug的修改或者是小的变动；

RELEASE：希腊字母版本号，用于标注当前版本处于软件的哪个开发阶段；

关于开发阶段有很多希腊字母版本号：

DEV: 开发版本（Development）

Alpha: 内部版本；

Beta: 测试版；

Snapshot: 快照版，代表不稳定尚处于开发中的版本；

Demo: 演示版；

Enhance: 增强版；

Free: 自由版；

Full Version: 完整版，即正式版；

Final: 最终版，正式版；

LTS: 长期维护版本(Long Term Support)，一般为18个月；

Pro(professional): 专业版；

Plus: 加强版；

Retail: 零售版；

Shareware: 共享版，虽然不要求注册但一般有功能限制；

Trial: 试用版（一般有时间或者功能限制）

Standard: 标准版

Ultimate: 旗舰版

Upgrade: 升级版

M: 里程碑版(M即Milestone），具有一些全新功能或具有里程碑意义的版本；

RC: 即将作为正式版发布(Release Candidat)

Release: 发行版，代表稳定的版本；

GA: 代表广泛可用的稳定版(General Availability)

SR: 修正版（Senior，中文意为资深的）

## IOC源码分析

1. ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfig.class);

2、new AnnotationConfigApplicationContext(SpringConfig.class);--> {

1、this();-->{

//spring内部加载了5个BeanDefinition：就是对bean的描述;

1、this.reader = new AnnotatedBeanDefinitionReader(this);

//向includeFilters放置了@Compment注解

2、this.scanner = new ClassPathBeanDefinitionScanner(this);

}

2、register(componentClasses); {

//把componentClasses(SpringConfig)变成bean的描述：BeanDefinition;

//然后把BeanDefinition包装成BeanDefinitionHolder;

//然后取出BeanDefinitionHolder里面的BeanDefinition;

//注册到Map<String, BeanDefinition> beanDefinitionMap中;

}

3、refresh();//参考下面的具体步骤

}

3、refresh(); {

//该方法总共有12个步骤，重点看第5步和第11步，宏观上的步骤是：

//根据项目中的bean（@Service、@Compment...注解的bean）得到BeanDefinition

//把BeanDefinition放入（注册）beanDefinitionMap中

//把beanDefinitionMap里面的BeanDefinition拿出来，依次创建所对应的对象；

}

}

## refresh()方法12个步骤分析

*// 1、预准备刷新上下文*

prepareRefresh();

*// 2、告诉子类去刷新内部的bean factory，如果是xml方式开发，此步骤很重要，如果是注解方式开发，这一步没有做什么操作；*ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

*// 3、预处理BeanFactory，注册systemEnvironment，environment，systemProperties几个bean*prepareBeanFactory(beanFactory);

*//4、由子类覆盖该方法做额外的处理，默认spring框架什么也没有做*postProcessBeanFactory(beanFactory);

*//5、调用各种BeanFactoryPostProcessor，注解开发的时候，解析和扫描注解得到BeanDefinition，然后放入到BeanDefinition的Map中，这一步非常重要（核心步骤）*

invokeBeanFactoryPostProcessors(beanFactory);

*//6、注册BeanPostProcessors，用于拦截Bean的创建，如果没有bean processors，此步骤什么也不做*registerBeanPostProcessors(beanFactory);  
  
*//7、实例化注册messageSource的bean，不同语言环境信息，国际化处理*initMessageSource();  
  
*//8、在上下文初始化注册applicationEventMulticaster的bean，应用事件广播*initApplicationEventMulticaster();

*//9、初始化其他的bean，默认情况下Spring框架什么也不做*onRefresh();  
  
*//10、在所有bean中查找listener bean并注册到消息广播中，没有的话就什么也不做；*registerListeners();  
  
*//11、实例化所有非延迟初始化的单例bean对象，这一步非常重要（核心步骤）*

finishBeanFactoryInitialization(beanFactory);  
  
*//12、完成刷新过程，通知生命周期处理器lifecycleProcessor刷新，同时发出ContextRefreshEvent通知相关事件*finishRefresh();

在以上的12个步骤中，只有第5步和第11步是非常重要的，是我们源码阅读重点需关注的，其他步骤可以略读，下面我们分别剖析一下这两个步骤，首先剖析第5步：

## IOC容器12步中的第5步剖析

*//调用BeanFactoryPostProcessor*

*invokeBeanFactoryPostProcessors(beanFactory);*

*1、调用BeanFactoryPostProcessors*

*PostProcessorRegistrationDelegate.invokeBeanFactoryPostProcessors(beanFactory, getBeanFactoryPostProcessors());*

*2、调用PostProcessors，里面处理bean的定义  
invokeBeanDefinitionRegistryPostProcessors*(currentRegistryProcessors, registry);

*//3、BeanDefinition的注册*

postProcessor.postProcessBeanDefinitionRegistry(registry);

*4、处理配置类的bean的定义*processConfigBeanDefinitions(registry);

*5、解析每一个带有@Configuration注解的类*ConfigurationClassParser parser = new ConfigurationClassParser(...);

*6、对配置类中注解进行处理*parser.parse(candidates);

*7、加载bean的定义: 把BeanDefinition放入到一个Map中*this.reader.loadBeanDefinitions(configClasses);

*8、注册bean的定义*this.registry.registerBeanDefinition(beanName, beanDefToRegister);

*9、注册bean定义，即把beanDefinition放入Map中*this.beanDefinitionMap.put(beanName, beanDefinition);

以上是第5步的主要流程，大家可以基于该流程反复断点调试跟几轮代码，潜移默化中提升源码的自我阅读能力，此能力需要坚持一个过程和一段时间，非一蹴而就；

## IOC容器12步中的第11步剖析

*//1、初始化所有剩下的非延迟初始化的单例bean对象*finishBeanFactoryInitialization(beanFactory);

*//2、实例化所有剩下的非延迟加载的单例bean*beanFactory.preInstantiateSingletons();

*//3、获取bean，底层初始化bean对象，并对bean对象赋值和依赖注入*getBean(beanName);

*//4、获取一个bean实例*return doGetBean(name, null, null, false);

*//5、创建单例的bean实例*return createBean(beanName, mbd, args);

*//6、创建bean实例*Object beanInstance = doCreateBean(beanName, mbdToUse, args);

*//7、创建bean实例*instanceWrapper = createBeanInstance(beanName, mbd, args); {

*// TODO 实例化bean实例*return instantiateBean(beanName, mbd);

*//TODO 此处真正地创建了 bean 的实例对象，底层依然是反射*beanInstance = getInstantiationStrategy().instantiate(mbd, beanName, parent);

*//TODO 根据构造器实例化对象*return BeanUtils.*instantiateClass*(constructorToUse);

}

*//8、填充bean的属性值，给bean赋值和注入依赖，如果当前bean依赖了其他的bean，则会递归调用beanFactory.getBean()方法尝试获取目标bean*populateBean(beanName, mbd, instanceWrapper);

## Spring循环依赖

Spring bean的创建，本质上是一个对象的创建；

一个完整的对象包含两部分：当前对象和对象的属性所依赖的另一个对象；

对象实例化通过反射实现，而对象的属性实例化通过依赖注入实现；

Spring实例化bean是通过ApplicationContext.getBean()方法来进行的，如果要获取的对象又依赖了另一个对象，那么其首先会创建当前对象，然后通过递归调用ApplicationContext.getBean()方法来获取所依赖的对象，最后将获取到的对象注入到当前对象中；

前提：（HeBean 依赖 MyBean，MyBean依赖HeBean）

假设容器首先创建HeBean：

1、创建HeBean对象：反射 （此时MyBean还没有创建）

2、把“heBean”放入 singletonFactories的Map中；

this.singletonFactories.put(beanName, singletonFactory);

this.earlySingletonObjects.remove(beanName);

3、给heBean设置属性值（也就是给heBean的成员变量赋值）

4、当HeBean依赖了MyBean，而MyBean还没有创建，此时HeBean无法完成属性赋值，HeBean还是一个半成品，然后就跳到了：beanFactory.getBean(“myBean”); （递归到了获取bean的入口处）

5、开始创建MyBean对象

1、创建MyBean对象：反射 （此时HeBean是个半成品）

2、把“myBean”放入 ingletonFactories的Map

this.singletonFactories.put(beanName, singletonFactory);

this.earlySingletonObjects.remove(beanName);

3、给myBean设置属性值（也就是给MyBean的成员变量赋值）

4、MyBean此时是一个半成品，此时代码：

//TODO 解析候选者bean，会调beanFactory.getBean()方法获取bean，该方法内部会进行初始化

instanceCandidate = descriptor.resolveCandidate(autowiredBeanName, type, this);

并没有完成heBean属性设置然后就跳到了：beanFactory.getBean(“heBean”);（递归到了获取bean的入口处）

去入口处：

public Object resolveCandidate(String beanName, Class<?> requiredType, BeanFactory beanFactory) throws BeansException {

// TODO 根据bean名称获取bean对象

return beanFactory.getBean(beanName);

}

5、从这里获取heBean：

ObjectFactory<?> singletonFactory =  
 this.singletonFactories.get(beanName);

singletonObject = singletonFactory.getObject();

1. 把heBean放入map

this.earlySingletonObjects.put(beanName, singletonObject);

this.singletonFactories.remove(beanName);

7、回到入口处：

public Object resolveCandidate(String beanName, Class<?> requiredType, BeanFactory beanFactory) throws BeansException {

// TODO 根据bean名称获取bean对象

return beanFactory.getBean(beanName);

}

8、此时已经拿到了HeBean对象，反射调用：field.set(bean, value); 把MyBean对象的heBean属性赋值，此时MyBean对象就完全创建好了；

9、把myBean放入ioc容器：

this.singletonObjects.put(beanName, singletonObject);

this.singletonFactories.remove(beanName);

this.earlySingletonObjects.remove(beanName);

10、返回到 resolveCandidate(String beanName, Class<?> requiredType, BeanFactory beanFactory) 方法（相当于返回到创建HeBean的第4步，此时已经拿到mybean对象，通过反射调用field.set(bean, value);把heBean对象的mybean属性赋值，此时HeBean对象也就创建好了，放入ioc的map容器中）

循环依赖：

1、setter注入 支持

2、构造方法注入 不支持

3、bean的多例模式 不支持

## 基于注解的Spring IOC容器宏观总结

1、创建AnnotationConfigApplicationContext;

2、创建AnnotationConfigApplicationContext的时候会先创建DefaultListableBeanFactory;

3、创建好DefaultListableBeanFactory之后，解析和扫描类上的注解并将所有Bean封装成BeanDefinition对象，该BeanDefinition对象封装了Spring所有bean的相关定义描述信息；

4、对Spring配置的所有bean做实例化，根据BeanDefinition对象的信息，底层采用反射创建具体的bean对象；

5、将实例化的bean对象放入到一个Map中，后续应用程序就能从该Map中获取到bean对象，该Map也就是我们的ioc容器；