IoC容器工作原理

主讲老师： 源码学院fox老师

1.创建beanFactory

容器底层用DefaultListableBeanFactory，即实现了BeanDefinitionRegistry，又实现了BeanFactory

java配置:

```
AnnotationConfigApplicationContext context =
      new AnnotationConfigApplicationContext(AppConfig.class);
```

this()方法---->调父类无参构造器

```
public GenericApplicationContext() {
   this.beanFactory = new DefaultListableBeanFactory();
}
```

在容器启动之前就创建beanFactory

xml配置:

```
ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("Spring.xml");
```

调refresh()后，即容器启动过程中创建beanFactory

```
ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

> refreshBeanFactory(); // AbstractApplicationContext#refreshBeanFactory
```

先创建DefaultListableBeanFactory实例，然后 解析xml配置文件，注册bean到beanFactory，最后再将beanFactory赋值给容器

```
DefaultListableBeanFactory beanFactory = createBeanFactory();
beanFactory.setSerializationId(getId());
// 定制beanFactory，设置参数
customizeBeanFactory(beanFactory);
// 注册spring的xml配置的bean到beanFactory，此时容器还未指定beanbeanFactory
loadBeanDefinitions(beanFactory);
// 给容器指定beanFactory
synchronized (this.beanFactoryMonitor) {
   this.beanFactory = beanFactory;
}
```

2.注册bean

核心原理:

通过调用registerBeanDefinition方法将bean的beanName----beanDefinition注册到beanFactory

```
DefaultListableBeanFactory#registerBeanDefinition  //  实现了
BeanDefinitionRegistry
> beanDefinitionMap.put(beanName, beanDefinition);  // 缓存beanDefinition
```

代码示例:

```
// 拿到工厂  实现了 BeanDefinitionRegistry
DefaultListableBeanFactory beanFactory =
context.getDefaultListableBeanFactory();

//创建一个beanDefinition
RootBeanDefinition beanDefinition = new RootBeanDefinition(User.class);

// 注册
beanFactory.registerBeanDefinition("user",beanDefinition);
```

java配置:

容器启动过程中, 会调用ConfigurationClassPostProcessor#postProcessBeanDefinitionRegistry 解析注解，注册bean

```
invokeBeanFactoryPostProcessors(beanFactory); // 解析注解，注册bean

> ConfigurationClassPostProcessor#postProcessBeanDefinitionRegistry
```

ConfigurationClassPostProcessor#postProcessBeanDefinitionRegistry 中有两个很重要的方法

```
// 解析配置类  @ComponentScan (bean注册到容器) @Import @ImportResource @Bean
parser.parse(candidates);

// 注册bean到容器
// 注册实现了ImportSelector的bean
// 方法bean注册到容器
// @ImportResource("spring.xml") 配置的bean注册到容器
// 实现ImportBeanDefinitionRegistrar的bean 注册到容器
this.reader.loadBeanDefinitions(configClasses);
```

解析注解

```
parser.parse(candidates);
>ConfigurationClassParser#processConfigurationClass

>ConfigurationClassParser#doProcessConfigurationClass
```

ConfigurationClassParser#doProcessConfigurationClass 会处理@ComponentScan，@Import，@ImportResource，@Bean

@ComponentScan会将@Component修饰的bean注入到容器

```
ClassPathBeanDefinitionScanner#doScan
>
// 找到@Component修饰的类的beanDefiniton集合
Set<BeanDefinition> candidates = findCandidateComponents(basePackage);
// 注册bean
registerBeanDefinition(definitionHolder, this.registry);
```

注册bean

```
this.reader.loadBeanDefinitions(configClasses)

>ConfigurationClassBeanDefinitionReader#loadBeanDefinitionsForConfigurationClass
```

```
if (configClass.isImported()) {
   //  implements ImportSelector 的bean 注册
   registerBeanDefinitionForImportedConfigurationClass(configClass);
}
for (BeanMethod beanMethod : configClass.getBeanMethods()) {
   // 方法bean 注册到容器
   loadBeanDefinitionsForBeanMethod(beanMethod);
}

// @ImportResource("spring.xml") 配置的bean注册到容器
loadBeanDefinitionsFromImportedResources(configClass.getImportedResources());
//  实现 ImportBeanDefinitionRegistrar的 bean 注册到容器
loadBeanDefinitionsFromRegistrars(configClass.getImportBeanDefinitionRegistrars(
));
```

xml配置：

```
ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

> refreshBeanFactory(); // AbstractApplicationContext#refreshBeanFactory
```

将beanFactory赋值给容器之前 解析xml，并且注册bean

```
DefaultListableBeanFactory beanFactory = createBeanFactory();
beanFactory.setSerializationId(getId());
// 定制beanFactory，设置参数
customizeBeanFactory(beanFactory);
// 注册spring的xml配置的bean到beanFactory，此时容器还未指定beanbeanFactory
loadBeanDefinitions(beanFactory);
// 给容器指定beanFactory
synchronized (this.beanFactoryMonitor) {
   this.beanFactory = beanFactory;
}
```

解析xml

loadBeanDefinitions(beanFactory) 最终会调
DefaultBeanDefinitionDocumentReader#doRegisterBeanDefinitions

```
// 注册spring的xml配置的bean到beanFactory，此时容器还未指定beanbeanFactory
```

```
loadBeanDefinitions(beanFactory);

>DefaultBeanDefinitionDocumentReader#doRegisterBeanDefinitions

preProcessXml(root);
parseBeanDefinitions(root, this.delegate);// 解析xml
postProcessXml(root);

>DefaultBeanDefinitionDocumentReader#parseDefaultElement

else if (delegate.nodeNameEquals(ele, BEAN_ELEMENT)) {
    // 解析bean
    processBeanDefinition(ele, delegate);
}

>DefaultBeanDefinitionDocumentReader#processBeanDefinition
// 注册最后修饰后的实例beanDefinition
BeanDefinitionReaderUtils.registerBeanDefinition(bdHolder,
getReaderContext().getRegistry());
```

注册bean

BeanDefinitionReaderUtils#registerBeanDefinition 中会注册bean

```
// 注册beanDefiniton
registry.registerBeanDefinition(beanName, definitionHolder.getBeanDefinition());
```

3.创建bean实例

准备

容器启动过程中会实例化非懒加载单例bean,通过AbstractBeanFactory创建bean的实例

```
AbstractApplicationContext#refresh

// Instantiate all remaining (non-lazy-init) singletons.
// 实例化所有剩余的(非懒加载)单例。
> finishBeanFactoryInitialization(beanFactory);
> AbstractApplicationContext#getBean(String)
```

调用context.getBean(name)获取bean实例，实际会去调用AbstractBeanFactory的getBean()方法。

```
AbstractApplicationContext#getBean(String)
>AbstractBeanFactory#getBean(String)
>AbstractBeanFactory#doGetBean
```

调用AbstractBeanFactory#doGetBean， 先从singletonObjects中去获取bean

```
//转换对应的beanName
// 1.带&前缀的去掉前缀
// 2.从aliasMap中找name对应id，bean没有配id就用name
final String beanName = transformedBeanName(name);
```

```
//先从缓存singletonObjects中找,没有则去创建
//处理循环依赖的问题,比如A->B，B->A
Object sharedInstance = getSingleton(beanName);


//判断单例
if (mbd.isSingleton()) {
    sharedInstance = getSingleton(beanName, () -> {
        return createBean(beanName, mbd, args);
    });
    bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);
}
```

调用AbstractAutowireCapableBeanFactory#doCreateBean 返回bean

```
AbstractAutowireCapableBeanFactory#createBean(String, RootBeanDefinition,
Object[])
// 返回bean实例
>Object beanInstance = doCreateBean(beanName, mbdToUse, args);
```

实例化bean

调用AbstractAutowireCapableBeanFactory#doCreateBean

```
/**
* 第2次调用后置处理器
* 创建bean实例，并将实例放在包装类BeanWrapper中返回
* 1.通过工厂方法创建bean实例
* 2.通过构造方法自动注入创建bean实例
* 3.通过无参构造器创建bean实例
*/
instanceWrapper = createBeanInstance(beanName, mbd, args);
final Object bean = instanceWrapper.getWrappedInstance();
```

填充bean

```
// 填充bean 设置属性  InstantiationAwareBeanPostProcessors
// 第5次，第6次调用后置处理器    注入依赖
populateBean(beanName, mbd, instanceWrapper);
```

初始化bean

```
// 初始化bean
exposedObject = initializeBean(beanName, exposedObject, mbd);
```

返回bean

```
// 返回类型判断 FactoryBean BeanFactroy
bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);
>
//  涉及FactoryBean的判断，直接返回普通bean的条件  类型是否是FactoryBean || name首字符是否是&
if (!(beanInstance instanceof FactoryBean)
||BeanFactoryUtils.isFactoryDereference(name){
    return beanInstance;
}
```