**Spring5 源码学习**

1. **Spring5 介绍**
   1. **Spring5 学习资源**

Spring5 框架使用说明文档：

<https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/>

源码获取地址：

<https://github.com/spring-projects/spring-framework>

github-wiki 介绍：

<https://github.com/spring-projects/spring-framework/wiki>

<https://github.com/spring-projects/spring-framework/wiki/Build-from-Source>

gradle 教程：<https://www.w3cschool.cn/gradle/>

导入到 eclipse

<https://github.com/spring-projects/spring-framework/blob/master/import-into-eclipse.md>

导入到 IDEA

<https://github.com/spring-projects/spring-framework/blob/master/import-into-idea.md>

spring 各种组件的使用指南

<https://spring.io/guides>

各版本新特性介绍

## 2.1 spring 框架模块介绍

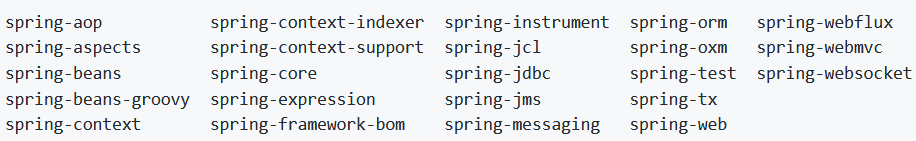
源码获取了解：

<https://github.com/spring-projects/spring-framework>

从 Spring-Framework-Artifacts 了解：

<https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-Artifacts>

从源码中可以看到 spring-framework 的模块构成，有 20+个模块



每个模块的用途，在模块的源码中有介绍，如 spring-core 模块的说明在：

[spring-framework](https://github.com/spring-projects/spring-framework)/[spring-core](https://github.com/spring-projects/spring-framework/tree/master/spring-core)/[src](https://github.com/spring-projects/spring-framework/tree/master/spring-core/src)/[main](https://github.com/spring-projects/spring-framework/tree/master/spring-core/src/main)/[java](https://github.com/spring-projects/spring-framework/tree/master/spring-core/src/main/java)/**overview.html**

根据特性划分为：

 [Core technologies:](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html) dependency injection, events, resources, i18n, validation, data binding, type conversion, SpEL, AOP.

spring-core spring-jcl

spring-beans

spring-context

spring-expression spring-aop spring-aspects

[Testing:](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/testing.html) mock objects, TestContext framework, Spring MVC Test, WebTestClient.

spring-test

 [Data Access:](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/data-access.html) transactions, DAO support, JDBC, ORM, Marshalling XML.

spring-jdbc spring-tx spring-orm spring-context-indexer spring-oxm

 [Spring MVC](https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html) and [Spring WebFlux](https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html) web frameworks.

spring-webflux spring-webmvc

spring-messaging

spring-websocket

spring-web

 [Integration:](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/integration.html) remoting, JMS, JCA, JMX, email, tasks, scheduling, cache.

spring-context-support spring-jms

 [Languages:](https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/languages.html) Kotlin, Groovy, dynamic languages.

spring-beans-groovy

## 2.2 spring5 特性介绍

官网介绍：

<https://spring.io/projects/spring-framework>

<https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/>

|  |  |
| --- | --- |
| [Core](https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/core.html#spring-core) | IoC container, Events, Resources, i18n, Validation, Data Binding, Type Conversion, SpEL, AOP. |
| [Testing](https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/testing.html#testing) | Mock objects, TestContext framework, Spring MVC Test, WebTestClient. |
| [Data Access](https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/data-access.html#spring-data-tier) | Transactions, DAO support, JDBC, ORM, Marshalling XML. |
| [Web Servlet](https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/web.html#spring-web) | Spring MVC, WebSocket, SockJS, STOMP messaging. |
| [Web Reactive](https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/web-reactive.html#spring-webflux) | Spring WebFlux, WebClient, WebSocket. |
| [Integration](https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/integration.html#spring-integration) | Remoting, JMS, JCA, JMX, Email, Tasks, Scheduling, Cache. |
| [Languages](https://docs.spring.io/spring/docs/5.0.8.RELEASE/spring-framework-reference/languages.html#languages) | Kotlin, Groovy, Dynamic languages. |

## spring5 新特性

主要的新特性是 Spring WebFlux

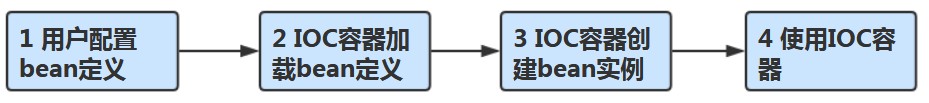
其他的一些详细更新请查阅下面链接：

<https://github.com/spring-projects/spring-framework/wiki/What%27s-New-in-Spring-Framework-5.x> <https://github.com/spring-projects/spring-framework/wiki/Upgrading-to-Spring-Framework-5.x> <https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-5-FAQ> <https://github.com/spring-projects/spring-framework/wiki/Spring-Framework-Versions>

# IOC

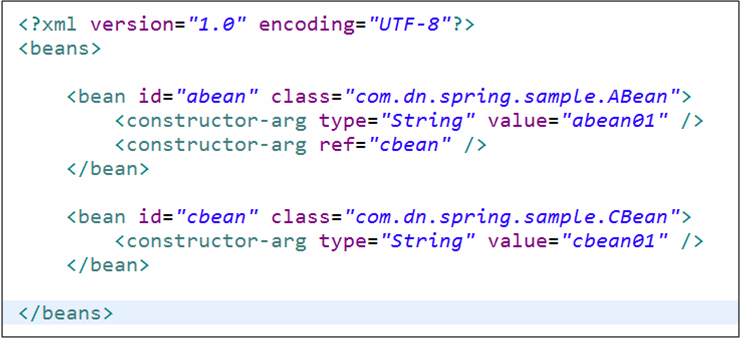
## IOC 是什么

为用户提供创建、管理、获取它们的类实例的容器。让用户在需要类对象时，只需向 IOC 容器要，进而达到与具体类解耦。为其他的高级功能提供基础。

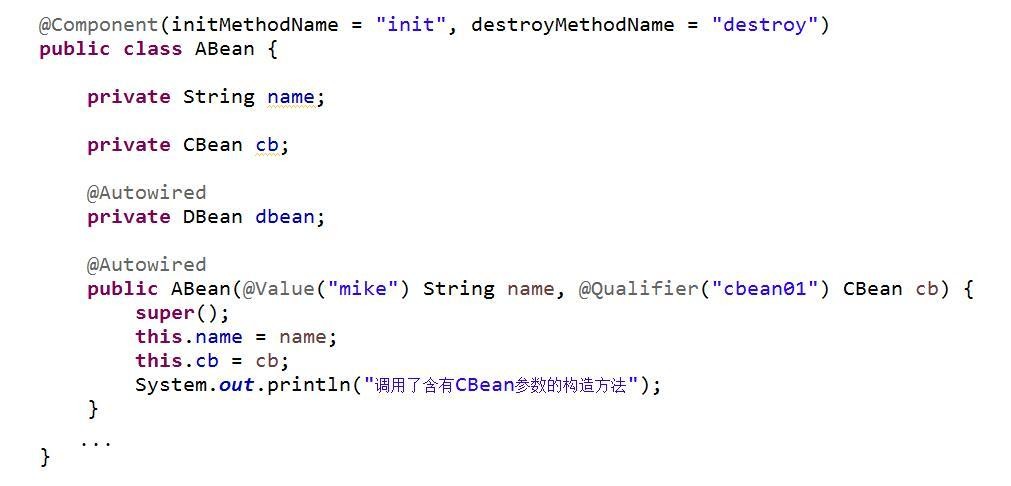
IOC 需要完成哪些工作，才能被我们使用？

我们使用 spring 时有几种方式来配置 bean 定义？

### Xml



* + - **Annotation**



* + - **Java based**



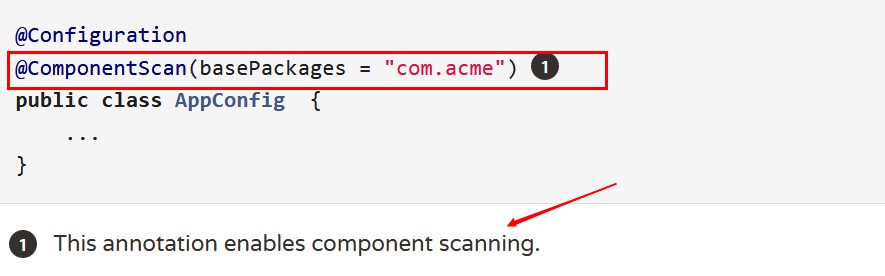
这三种方式，我们可以如何来告诉 spring 我们的 bean 定义配置，即我们可以如何来开始使用 spring?

### Xml

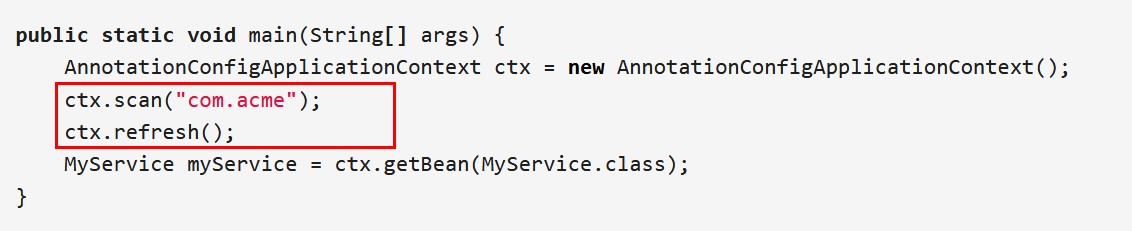
* + - **Annotation**

Xml 方式指定扫描的基础包

注解方式指定扫描的基础包



也可在代码中通过 API 直接指定



### Java based

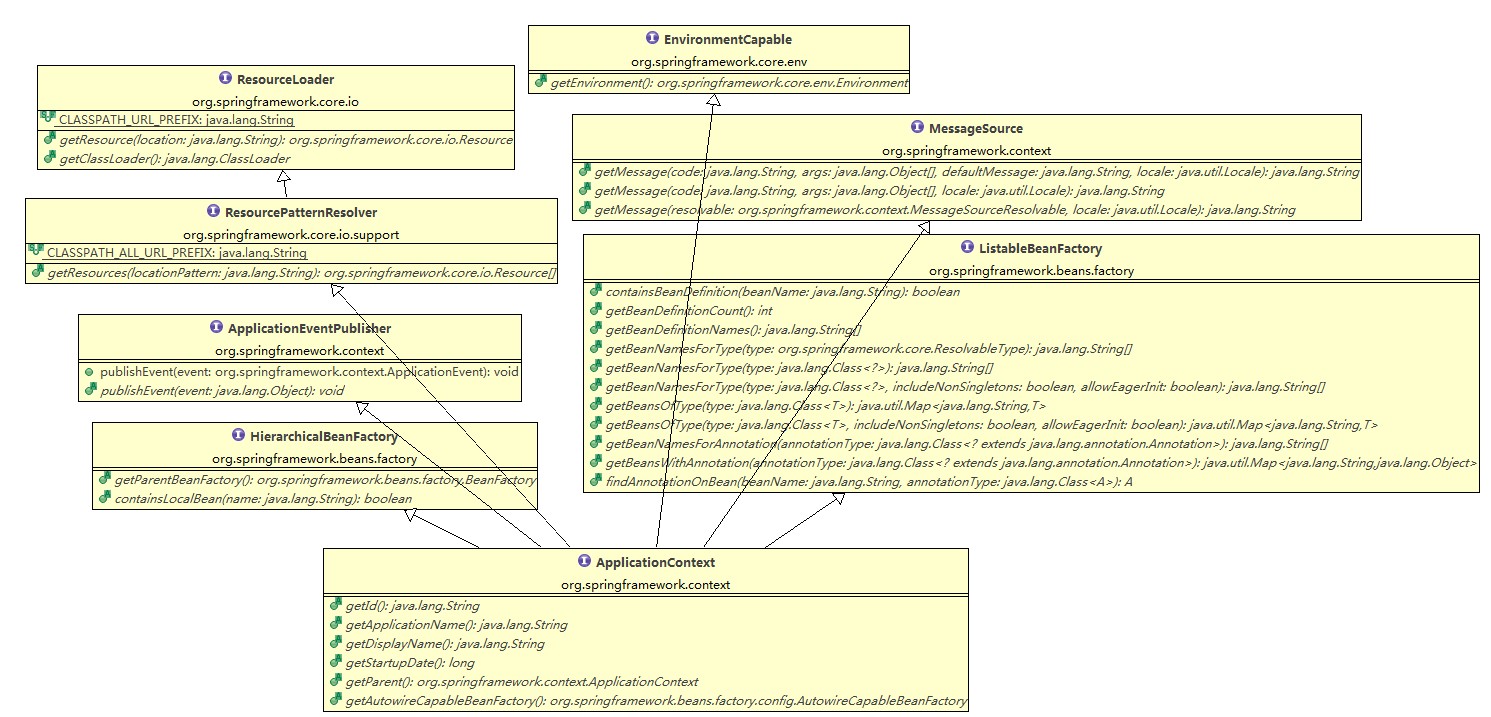
## 入口 ApplicationContext

* + 1. **ApplicationContext 是什么**

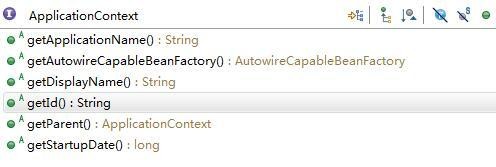
首先来了解 ApplicationContext 都是什么（即它负有哪些角色、责任）。它通过继承很多接口而肩负很多角色：



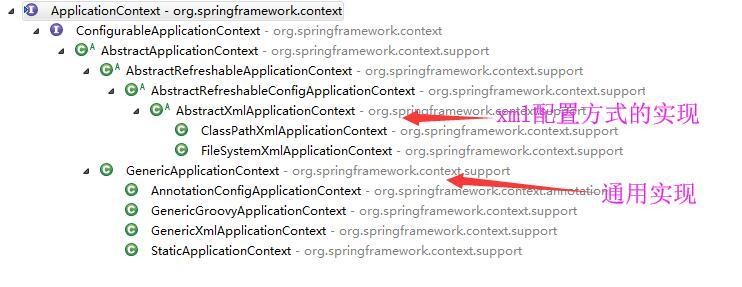
每个角色都有些什么职责（方法）

请仔细看图了解各接口定义的功能方法：

再来了解 ApplicationContext 自己中定义的方法：



## ApplicationContext 的子实现

接下来了解一下 ApplicationContext 的子：

从 AbstarctApplicationContext 之后分为两类：xml 配置方式的实现和通用实现。它们的基本使用示例如下：

@Configuration

**public class** MikeApplication {

**public static void** main(String[] args) {

ApplicationContext context = **new** ClassPathXmlApplicationContext("application.xml"); CombatService cs = context.getBean(CombatService.**class**);

cs.doInit(); cs.combating();

ApplicationContext context1 = **new** FileSystemXmlApplicationContext("e:/study/application.xml"); cs = context1.getBean(CombatService.**class**);

cs.doInit(); cs.combating();

context1 = **new** GenericXmlApplicationContext("file:e:/study/application.xml");

cs = context1.getBean(CombatService.**class**); cs.doInit();

cs.combating();

// 注解的方式

ApplicationContext context2 = **new** AnnotationConfigApplicationContext(MikeApplication.**class**); CombatService cs2 = context2.getBean(CombatService.**class**);

cs2.combating();

System.***out***.println("------------------------------------------------------");

GenericApplicationContext context3 = **new** GenericApplicationContext();

**new** XmlBeanDefinitionReader(context3).loadBeanDefinitions("classpath:application.xml");

**new** ClassPathBeanDefinitionScanner(context3).scan("com.study.mike.spring.service");

// 一定要刷新

context3.refresh();

cs2 = context3.getBean(CombatService.**class**); cs2.combating();

Abean ab = context3.getBean(Abean.**class**);

ab.doSomething();

}

@Bean

**public** CombatService getCombatService() {

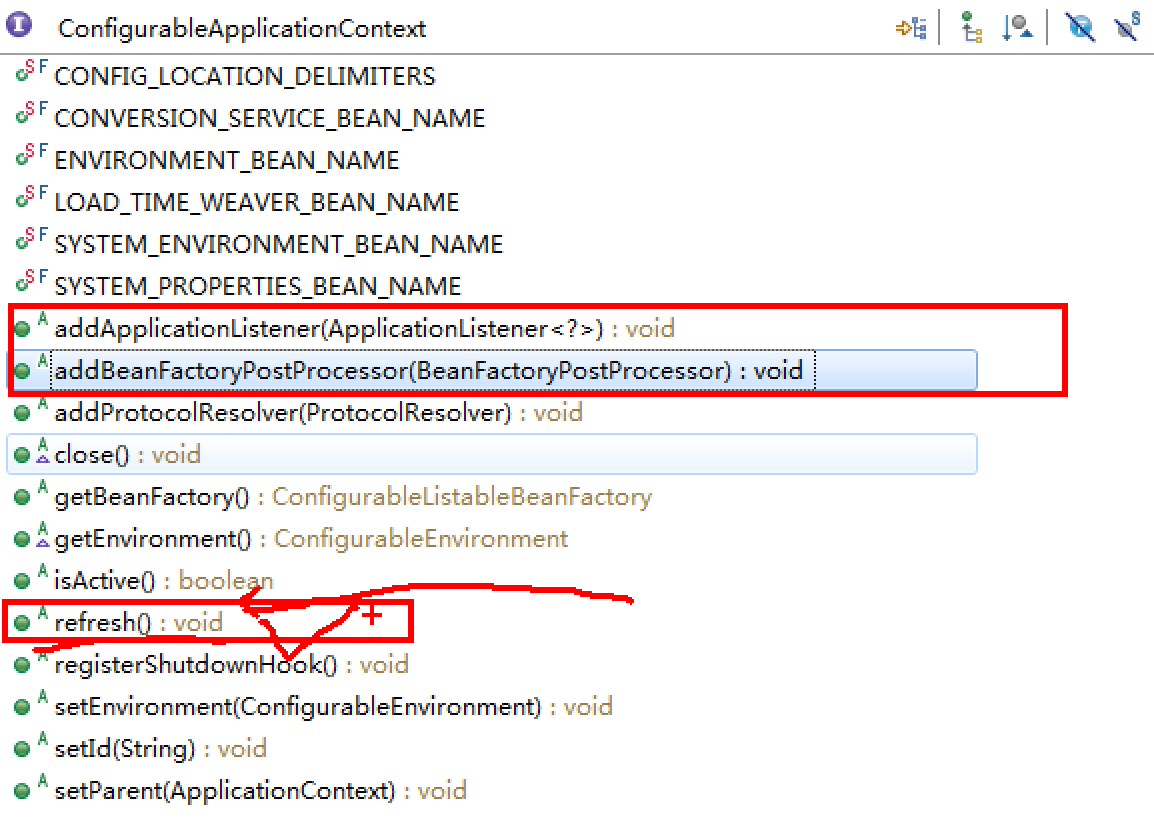
**return new** CombatService(120);

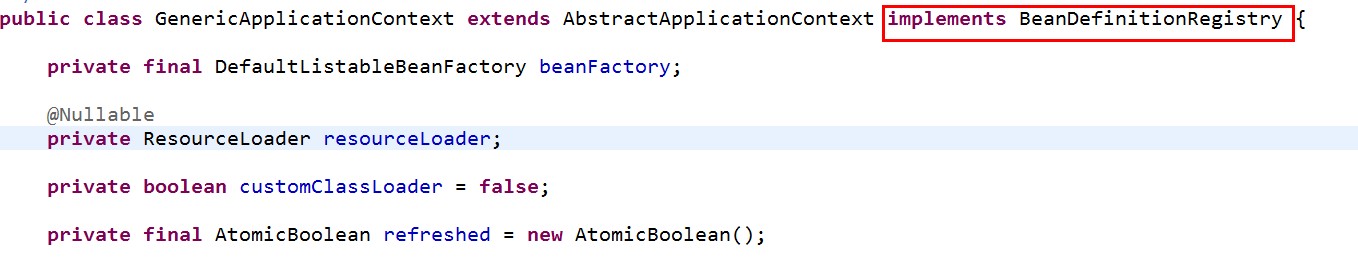
}

}

接下来，可以打开每个子去了解它们分别加入了什么、实现了什么。

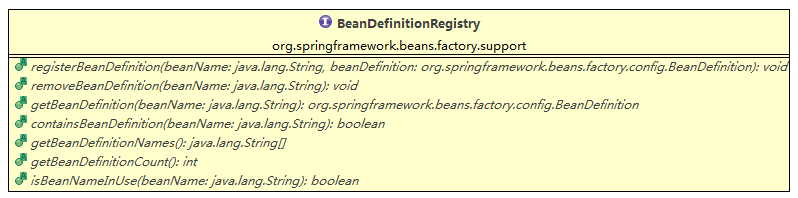
### ConfigurableApplicationContext 加入了什么：

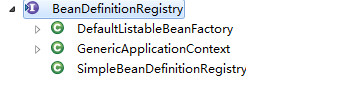


* + - * **AbstarctApplicationContext 提供了哪些基本的实现。**
      * **AbstractRefreshableApplicationContext 提供了哪些基本的实现。**
      * **Xml 这条线往，了解具体实现的构造方法、构造方法的源码是怎样的。通用的实现来学习了解一下。**
      * **GenericApplicationContext**

**它实现了 BeanDefinitionRegistry 接口，该接口定义了 bean 定义信息的注册行为。即我们可以直接往**

**GenericApplicationContext 中注册 bean 定义。了解一下 BeanDefinitionRegistry 中定义的行为：**

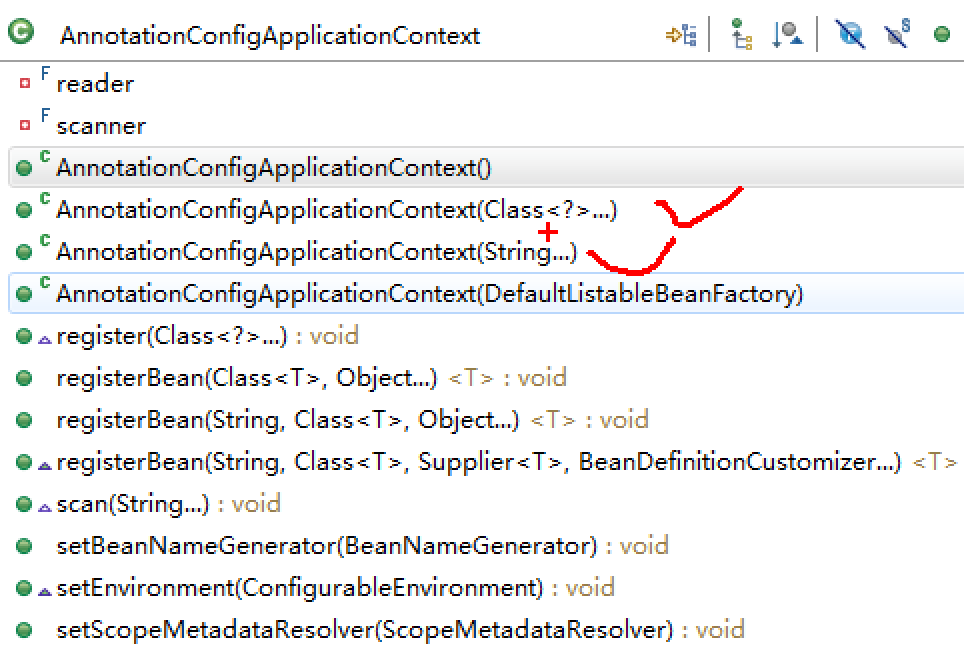


**都有谁实现了 BeanDefinitionRegistry 接口：**

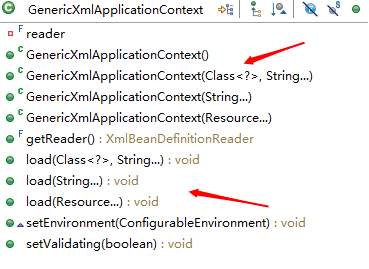
**GenericApplicationContext 中是不是持有 DefaultListableBeanFactory? GenericApplicationContext 的 bean 定义注册是不是委托给了持有的 DefaultListableBeanFactory？ 请看它的源码。**

**接下来请了解 GenericApplicationContext 的两个子**

* + - * **AnnotationConfigApplicationContext 了解它的构造方法、register 方法。看它的源码。**



* + - * **GenericXmlApplicationContext 了解它的构造方法、load 方法，看源码。**



## Bean 定义加载、解析

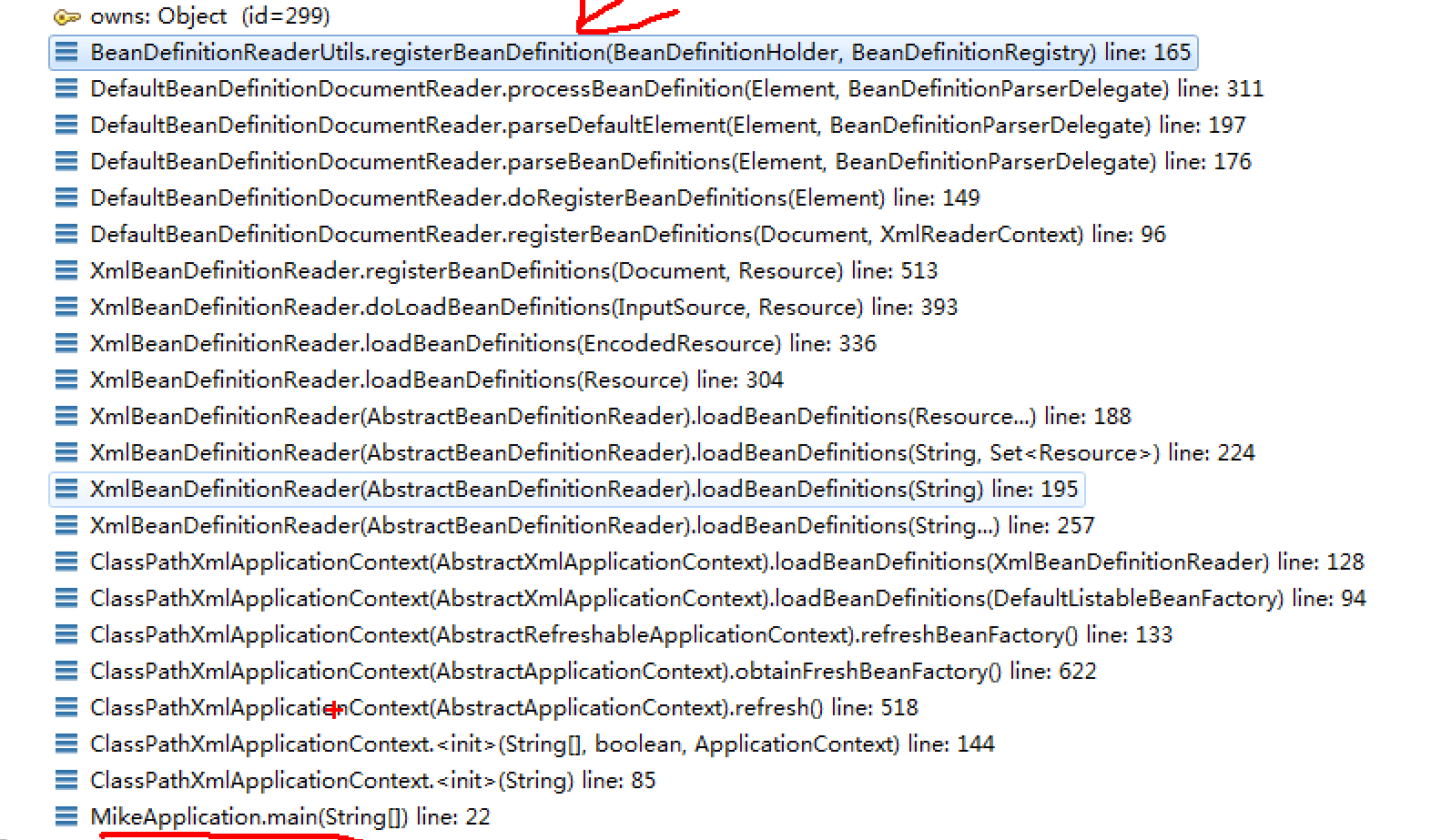
ApplicationContext 如何加载、解析 Bean 定义的。读源码，我们不光是了解一下这个过程，更重要的是看它是如何设计接口、类来配合解决这个问题的，以及有哪些扩展点、灵活之处。

## Xml Bean 配置的加载与解析

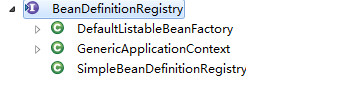
 xml 配置方式的 bean 定义获取过程是怎样的？



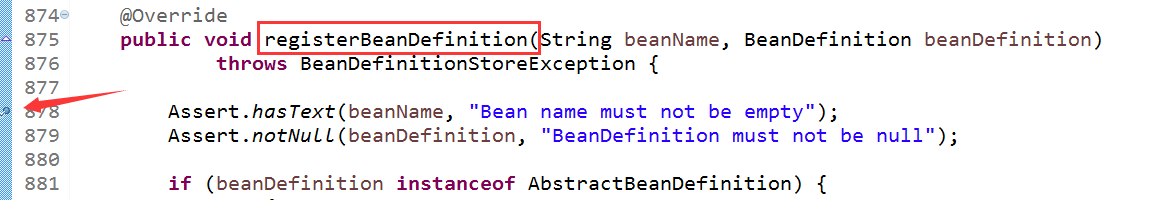
通过艰辛之旅，我们得到了这个。



在这里我们也可以有捷径获得这个调用栈，如果你够细心，会联想：我们这里解析 xml 是为了获取 bean

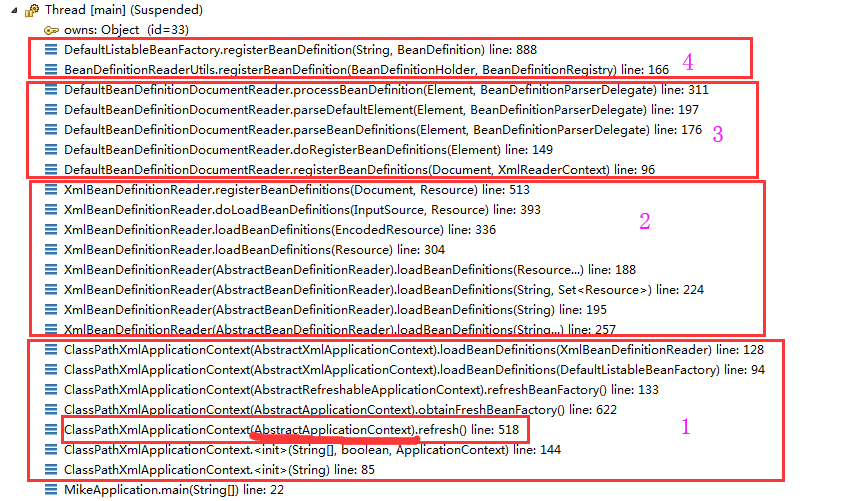
定义信息，注册到 IOC 容器中，也就是说一定会调用 IOC 容器的注册方法。这个在哪里？

在 DefaultListableBeanFactory 中的注册方法中打上断点，试试。



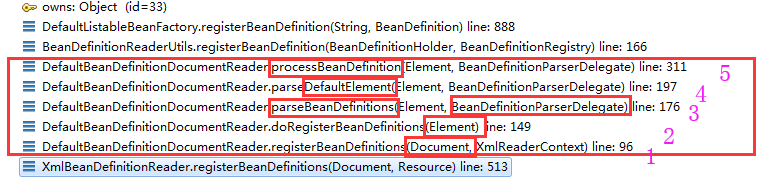


从这个调用栈上我们可以看到主要工作是由如下四部分完成

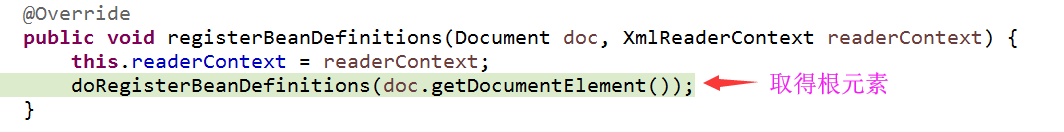


### 3.3.1.1 Xml 元素解析

从第三部分的调用栈上，我们可以看到如下这 4 处变化



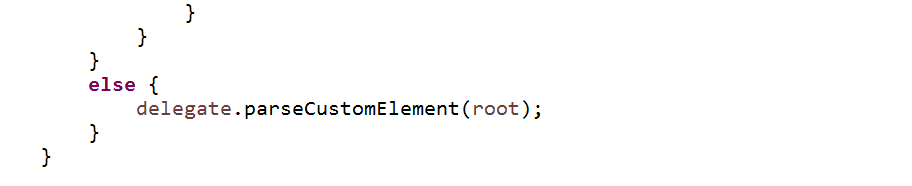
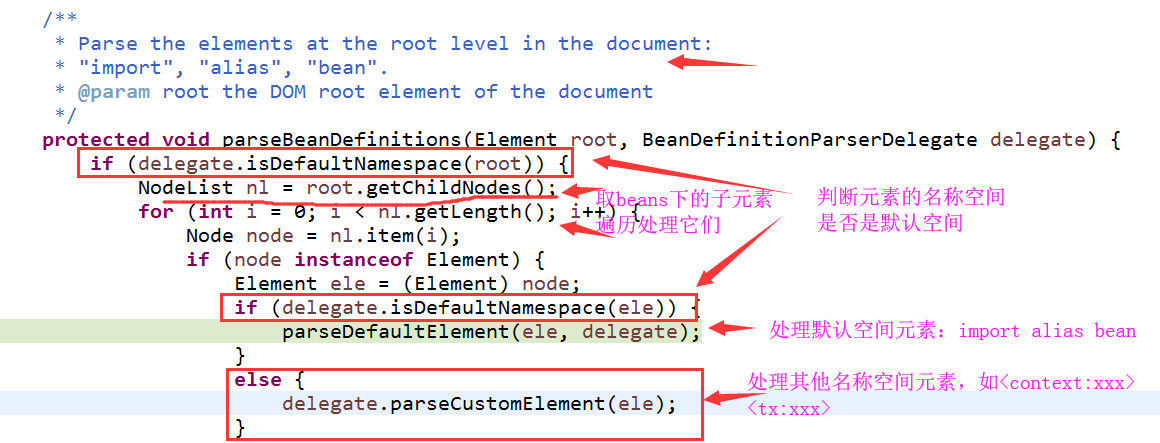
1  2 Document  Element



1、看 doRegisterBeanDefinitions 方法：

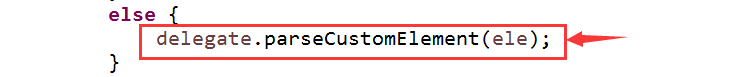




2、接下来看在 parseBeanDefinitions 方法中是如何来处理里面的元素的：

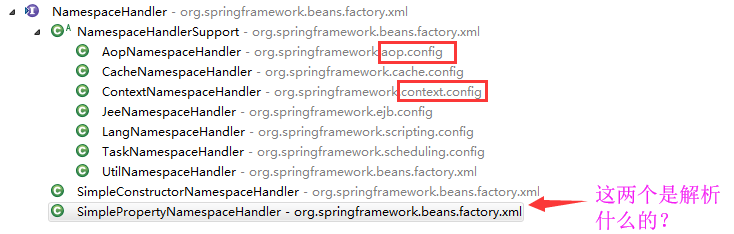
### 我们重点关心的是它是如何处理其他名称空间元素的，因为这里是个变化点：其他模块所需要的标签各异，表示的信息也不同，它也不知道其他模块会有哪些标签。 它是如何做到以不变应万变的？ 看下面的 xml 配置示例：



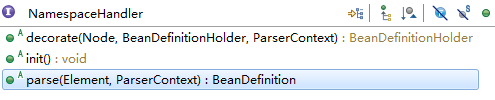
就来看 parseCustomElement：

它的代码：



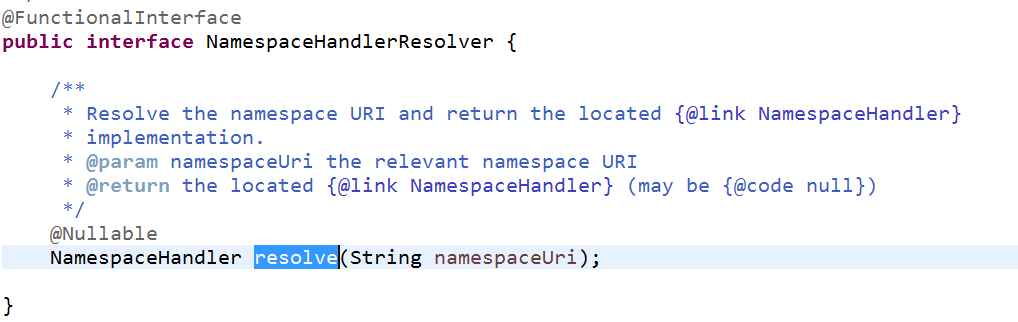
先来看一下 NamespaceHandler 的继承体系：

它中定义的方法：



请详细看 NamespaceHandler 的接口注释，方法注释了解两个方法的用法。

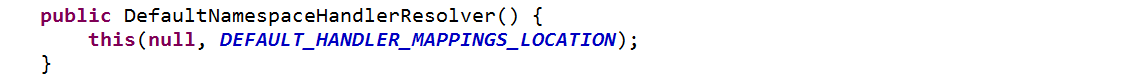
问题：名称空间对应的处理器在不同的模块实现，这里是如何加载到的？如事务处理的根本就不在现在的 这里。那就要去看图中 2 这条语句的方法调用了：

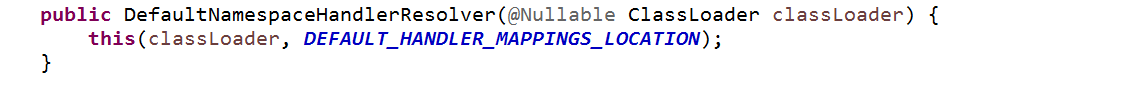
进入 resolve 方法

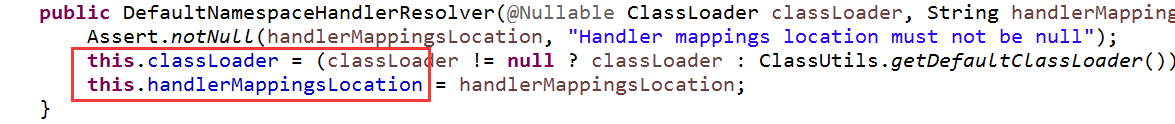
它是一个接口，那这里用的是它的什么实现类对象呢？

this.readerContext.getNamespaceHandlerResolver()方法。我们直接来看下它有什么实现（ctrl+t）

只有一个实现，那太好了，就去看它。

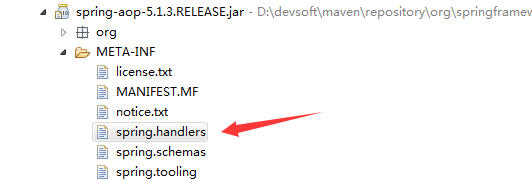




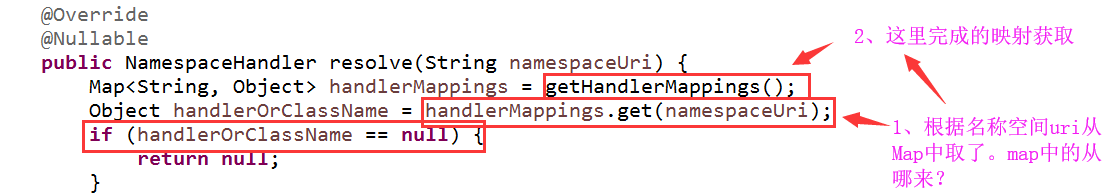


我们看到有两个属性 classLoader, handlerMappingsLocation。从 handlerMappingsLocation 这个名字能知道这是处理器与名称空间的映射的配置所在的地址。从前两个构造方法，我们看到它给入了一个常量的地址值：

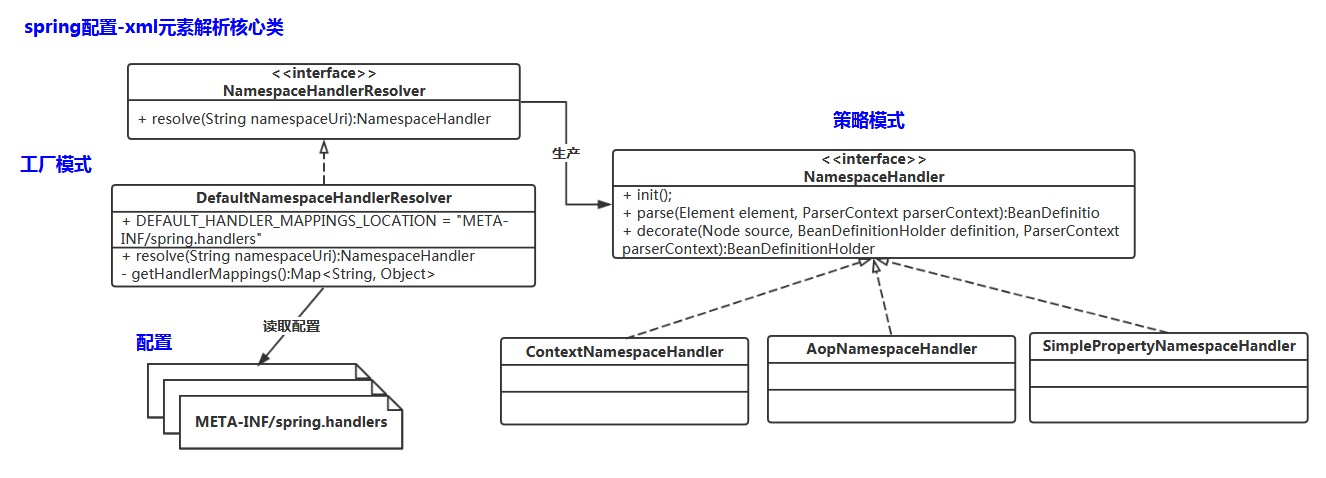


可大胆推测它是要到类目录下去找这个文件。请你看下我们的 spring 的模块 jar 包下有没有这个文件。

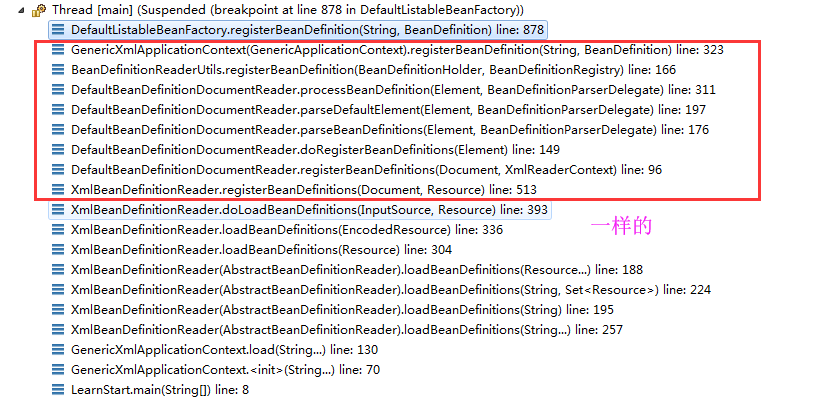


接下来来看下 resolve 方法：



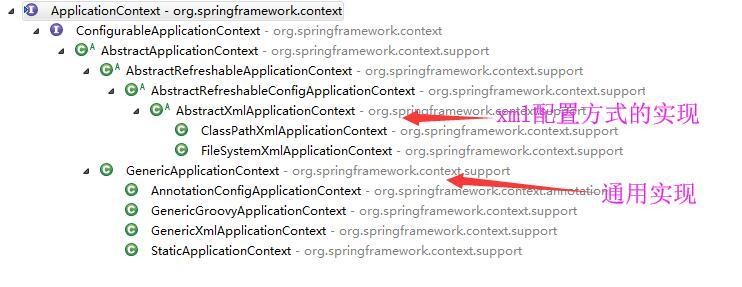


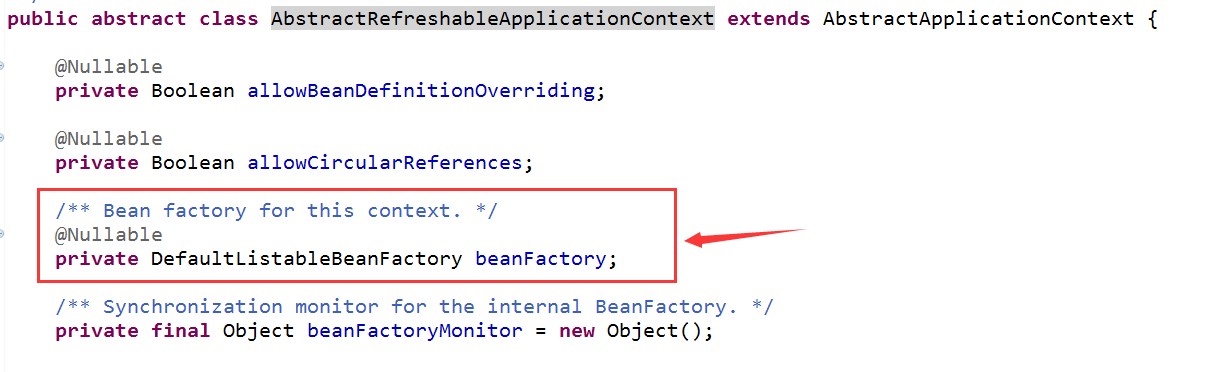
通用实现中的 GenericXmlApplicationContext 是否也是使用的这些 API

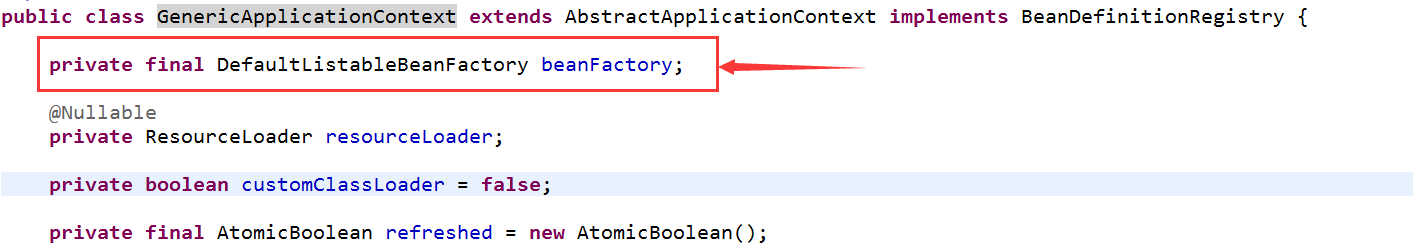


## BeanFactory

在ApplicationContext 的两类实现中，我们都可以看到 ApplicationContext 中持有DefaultListableBeanFactory：

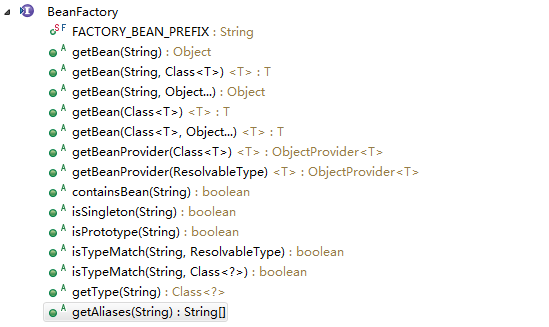




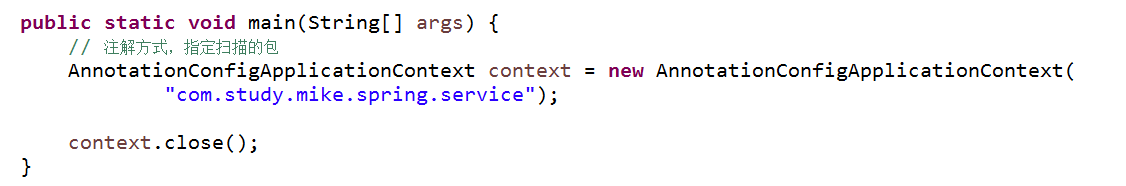


来看看 ApplicationContext 和 BeaFatory 的关系

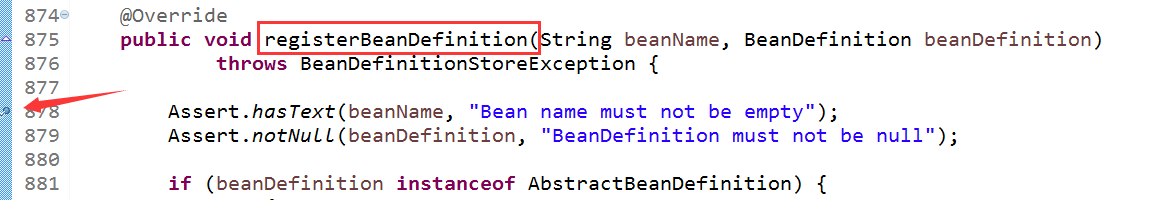


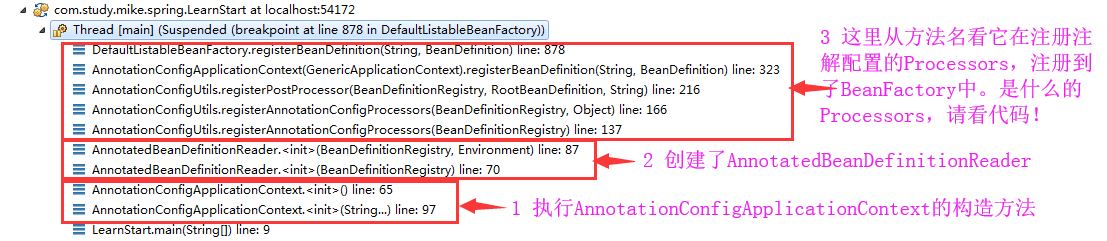


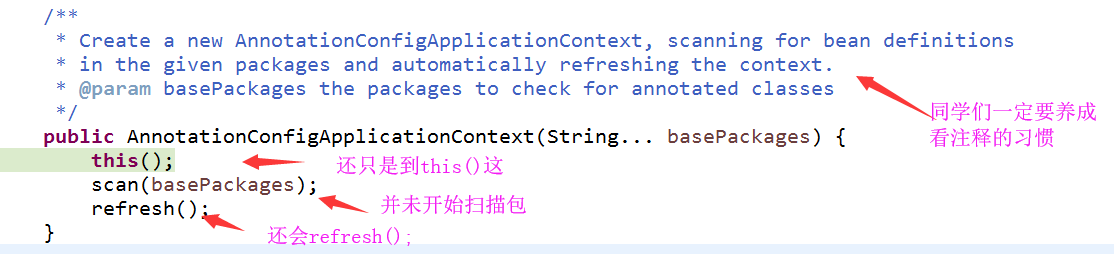
## Annotation Bean 配置的加载与解析

开始：

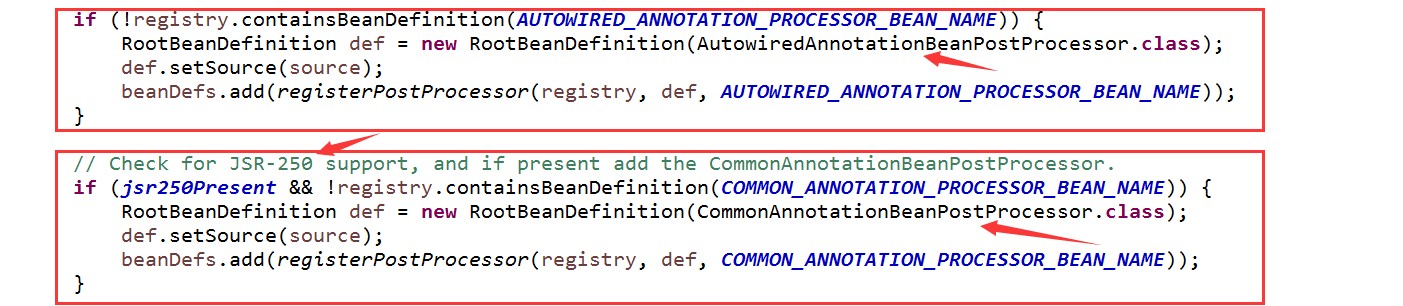
断点还是在 DefaultListableBeanFactory 中的注册方法上：

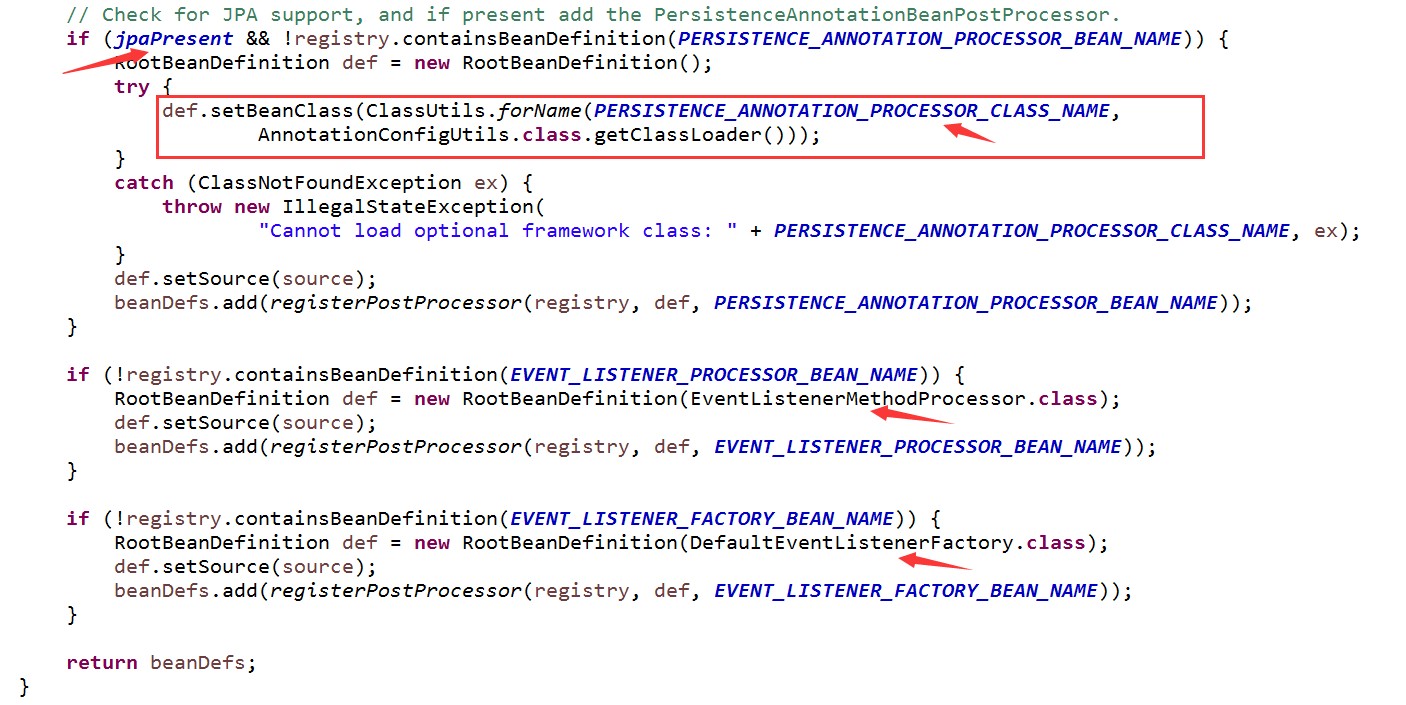


一调试运行，我就可以得到这样一个调用栈：

在这个调用栈中，我们并发没有看到它做包扫描的相关工作。从下往上看这个执行栈，点第 2 个，我们来看看代码：

它现在是在做一些初始化的准备处理，从这里我们获知，它做了 registerAnnotationConfigProcessors。从名字上理解就是注册了一些注解配置的处理器。到底是一些什么 processors，点方法进去看看：

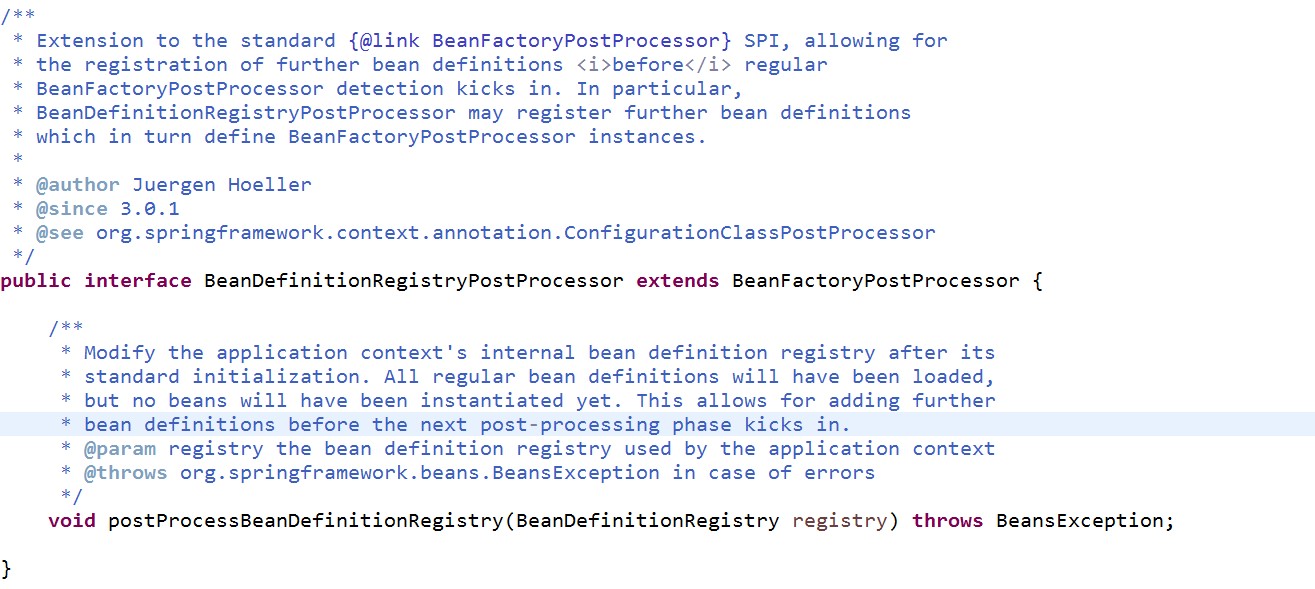




它注册了很多的 processor，都是些什么 Processor？点第一个的类名进去看看。

### BeanFactoryPostProcessor



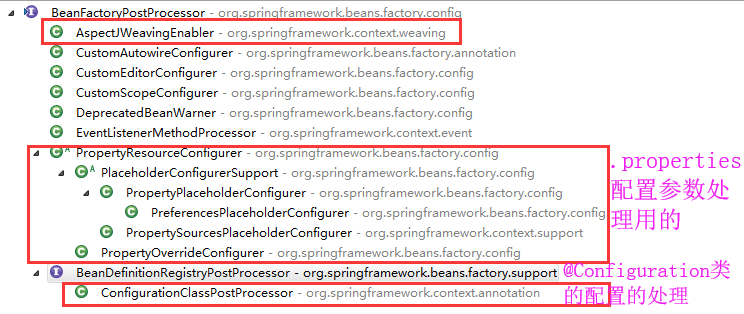


BeanDefinitionRegistryPostProcessor 扩展了 BeanFactoryPostProcessor，增加了 BeanDefinitionRegistry 位置的

处理，即它可以提前对注册好的 BeanDefinitionRegistry 进行前置处理。

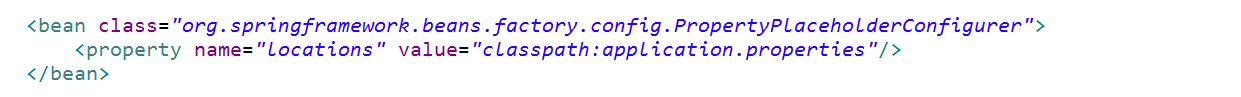


来看看 BeanFactoryPostProcessor 有哪些实现：



这是 springIOC 中给我们提供的又一个【扩展点】，让我们可以在 beanFactory 开始创建 Bean 实例前对

beanFactory 进行一些处理。我们可以来试试：



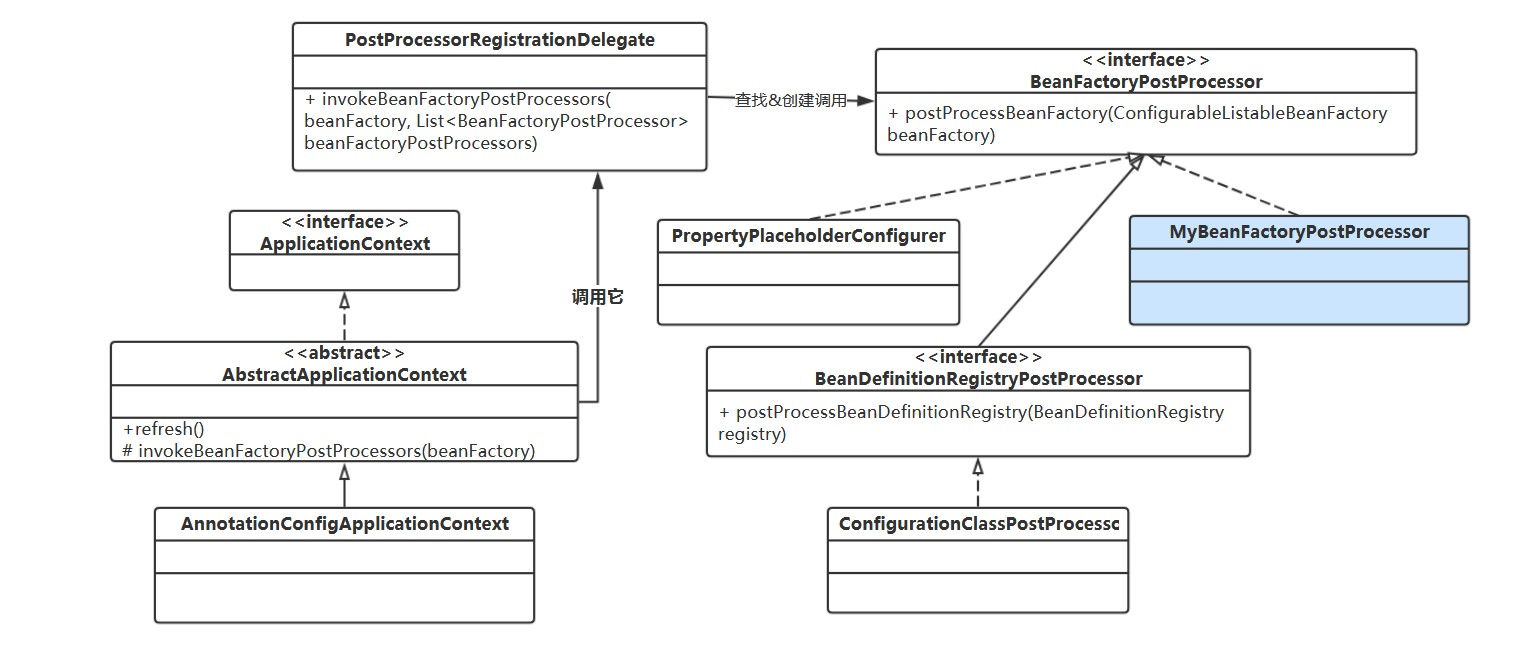
自己实现一个 BeanFactoryPostProcessor：



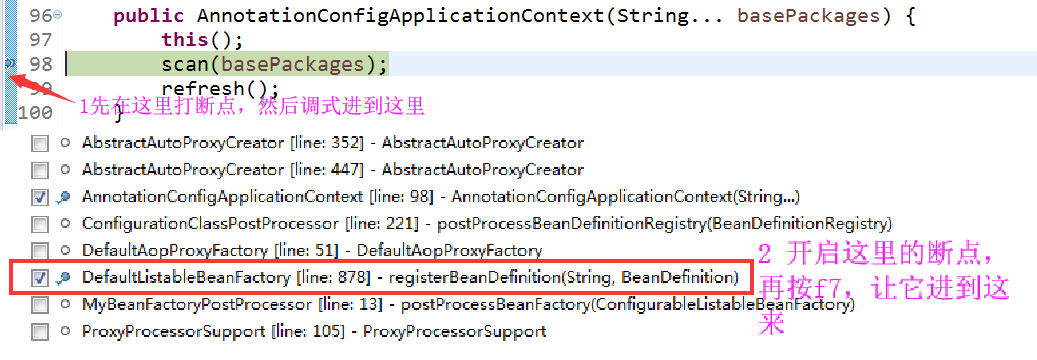
在哪里创建并执行的这个 MyBeanFactoryPostProcessor？ PostProcessorRegistrationDelegate.*invokeBeanFactoryPostProcessors*

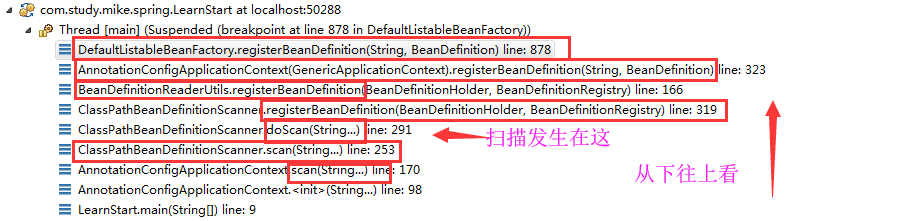
想要看其他的 BeanFactoryPostProcessor 都干了写什么，怎么看？ 在它里面的实现方法上打断点。

请把 BeanFactoryPostProcessor 加到类图中，它是重要一员。



看了这么久了，还没看注解定义的 bean 扫描！！！

* + - 1. **包扫描获取 Bean 定义**怎么看？

这样我们就可以得到调用栈了：



组件索引请阅读：

<https://docs.spring.io/spring/docs/5.1.3.RELEASE/spring-framework-reference/core.html#beans-scanning-index>

再来看下执行扫描的方法：



