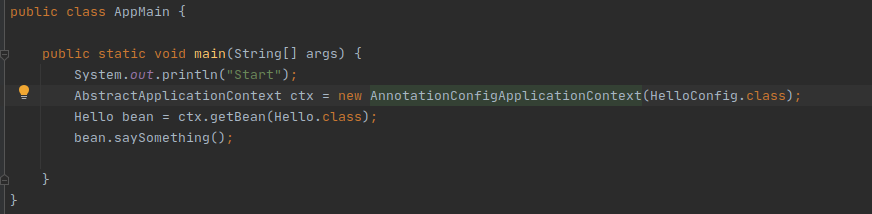
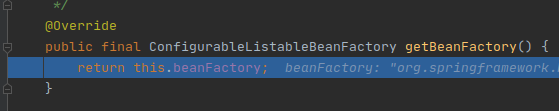
AbstractApplicationConext.getBean(Class<T>)

getBeanFactory().getBean()

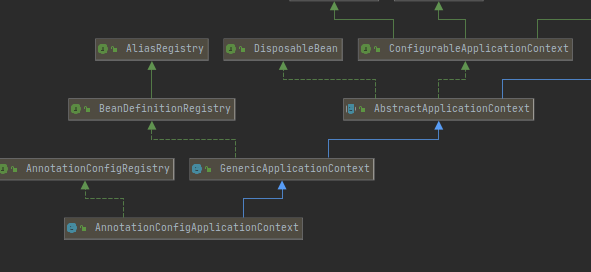
下图中，ctx对应的BeanFactory是如何创建出来的？



Debug, 看到getBeanFactory实际上调用的是GenericApplicationContext.

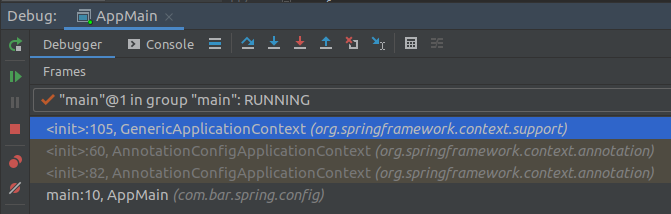


AnnotationConfigApplicationContext的父类是GenericApplicationContext.静态代码的时候，你就应该看到出BeanFactory源自哪里。源自GenericApplicationConext的成员变量。



下面就看这个成员变量是哪里来的？如何构造出来的。

new AnnotationConfigApplicationContext(HelloConfig.class);这个构造函数，其实调用了AnnotationConfigApplicationContext的默认构造函数，默认构造函数，虽然没明确写，肯定是调用了父类的默认构造函数，而父类的构造函数中就直接创建了DefaultListableBeanFactory。从下图的stack中可以证明这一点：



有了BeanFactory, （我们先不深入了解BeanFactory）, 后面是如何工作的，怎样把HelloConfig里的bean创建出来的？

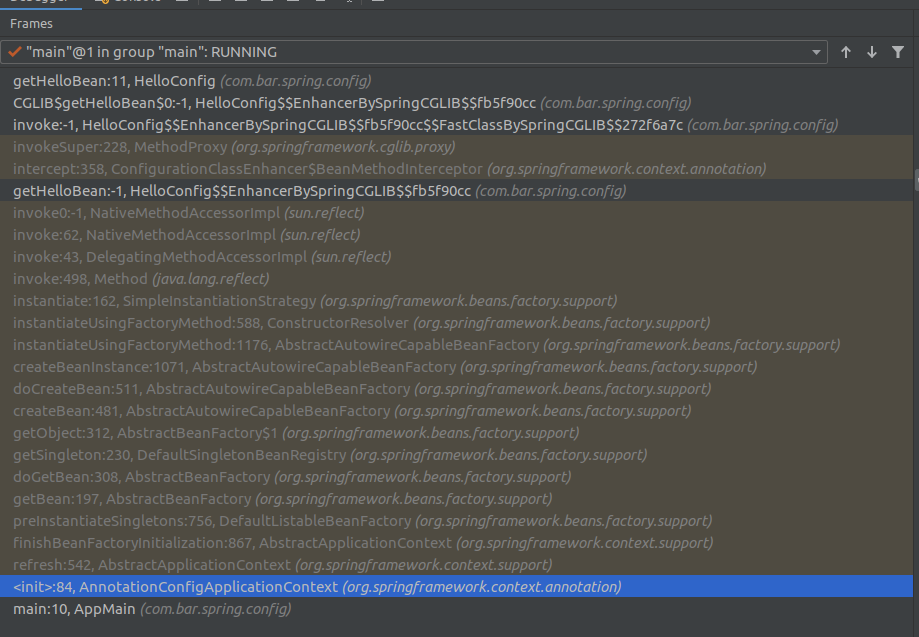
在AnnotationConfigApplicationContext的构造函数里，我们创建了两个成员变量: reader和scanner，都以this（registry）为参数。Reader (sanner)和AnnotationConfigApplicationContext的关系是互相直到对方的引用。但是Reader构造函数参数的类型是BeanDefintionRegitry, 所以只能访问这个接口中定义的方法。这就是解耦。

看名字理解，reader会把读取到的BeanDef注册到AnnotaionConfigAPplicationCOntext所实现的BeanDefinitionRegistry(interface)中。

AnnotaionConfigApplicationContext的成员变量reader:AnnotatedBeafnDefinitionReader, scanner:ClassPathBeanDefnitionScanner. 从名字上看，就很容易直到这两个类的用途。

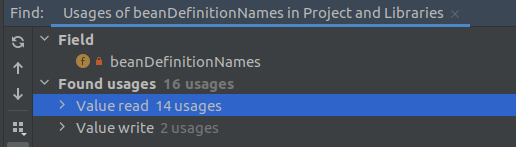
有点走错路，刚刚想找reader如何扫描的config，以为扫描的时候会load bean def或者创建bean。没找到，有点迷糊。又想是不是懒加载，在getBean的时候创建的Bean. 然后又看getBean。但是这个方向也是以午所获，都是按照自己的假设，去看。而我的假设是错的。经验不足？这种方式有点低效。

我给自己提的问题是怎样把HelloConfig里的bean创建出来？那么我们直接就在bean的构造函数里打断点，就可以看到stack。这才精准快速地找到我们想要的信息：



Bean的创建最上层的调用链是AnnotationConfigApplicationContext.constructor->refresh.

Field的引用分两类：read 和 write. 很多时候我们关注write, 这个field值的源头。



关键点： 什么时机调用的DefaultListableBeanFactory.registerBeanFactory. 在AbstrctApplicationContext.refresh->的时候。Refresh方法很核心，里面包含了一些BeanFactory的主要启动逻辑。

什么时候创建的bean?

AbstractApplicationContext.refresh -> finishBeanFactoryInitialization -> AbstractBeanFactory.getBean -> AbstractAutorwireCapableBeanFactory.createBean.

找到一个关键点。这个调用路径会把HelloConfig定义的bean definitions加载到DefaultLIstableBeanFactory(通过调用DefaultListableBeanFactory.registerBeanDefinition()).

AbstractApplicationContext.refresh -> … -> ConfigurationClassPostProcessor->processConfigBeanDefinitions->this.reader.loadBeanDefinitions(configClasses); //where configClasses contains HellConfig; reader’s type is ConfigurationClassBeanDefinitionReader.

Bean的一个老家终于找到：DefaultSingletonBeanRegistry

一个关键函数 AbstractBeanFactory.doGetBean. 根据Bean不同类型，查找或者创建Bean.