关于注册 bean 到容器

我们开发的类，如果想注册到 spring 容器，让 spring 来完成实例化，常用方式如下：

1. xml 中通过 bean 节点来配置；

2. 使用@Service、@Controller、@Conponent 等注解；

其实，除了以上方式，spring 还支持我们通过代码来将指定的类注册到 spring 容器中，也就是今天我们要实践的主要内容，接下来就从 spring 源码开始，先学习源码再动手实战；

本章概要

本章由以下几部分组成：

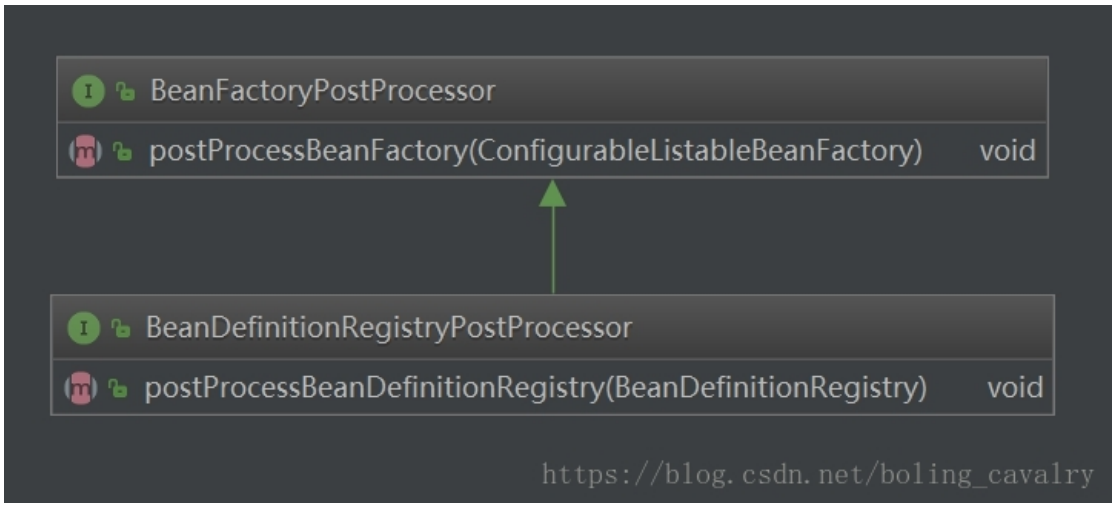
1. 了解 BeanDefinitionRegistryPostProcessor 接口；

2. 分析 spring 容器如何使用 BeanDefinitionRegistryPostProcessor 接口；

3. 实战，开发 BeanDefinitionRegistryPostProcessor 接口的实现类，验证通过代码注册 bean 的功能；

了解 BeanDefinitionRegistryPostProcessor 接口

实现注册 bean 功能的关键是 BeanDefinitionRegistryPostProcessor 接口，来看看这接口的继承关系，如下图：



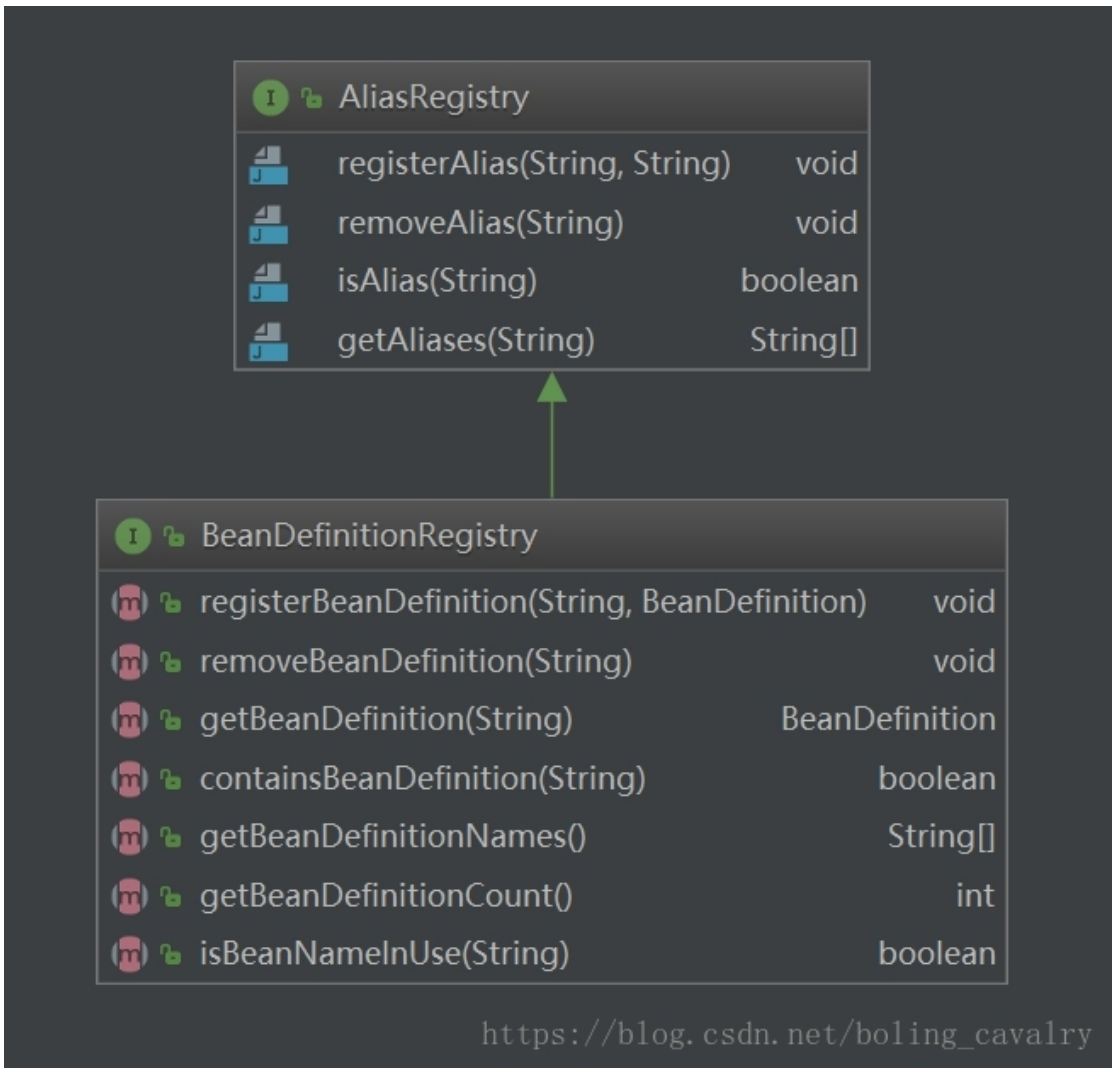
BeanDefinitionRegistryPostProcessor 继承了 BeanFactoryPostProcessor 接口，关于 BeanFactoryPostProcessor 我们在上一章《spring4.1.8 扩展实战之五：改变 bean 的定义(BeanFactoryPostProcessor 接口)》已做了详细的分析和实战，知道 BeanFactoryPostProcessor 的实现类在其 postProcessBeanFactory 方法被调用时，可以对 bean 的定义进行控制，因此 BeanDefinitionRegistryPostProcessor 的实现类一共要实现以下两个方法：

1. void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException：

该方法的实现中，主要用来对 bean 定义做一些改变，这些在上一章《spring4.1.8 扩展实战之五：改变 bean 的定义(BeanFactoryPostProcessor 接口)》有详细说明；

2. void postProcessBeanDefinitionRegistry(BeanDefinitionRegistry registry) throws BeansException：

该方法用来注册更多的 bean 到 spring 容器中，详细观察入参 BeanDefinitionRegistry 接口，看看这个参数能带给我们什么能力：

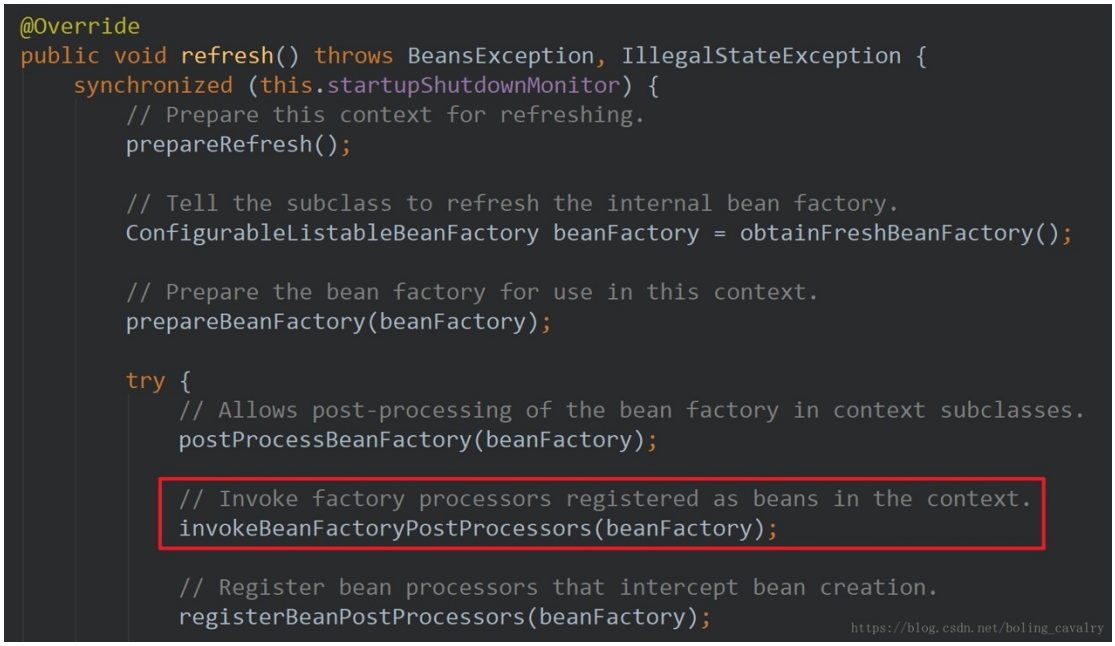


从上图可以看到，为了能让我们通过代码将 bean 注册到 spring 环境，BeanDefinitionRegistry 提供了丰富的方法来操作 bean 定义，判断、注册、反注册等方法都准备好了，我们在编写 postProcessBeanDefinitionRegistry 方法的内容时，就能直接使用入参 registry 的这些方法来完成判断和注册、反注册等操作；

分析 spring 容器如何使用 BeanDefinitionRegistryPostProcessor 接口

来看看 BeanDefinitionRegistryPostProcessor 接口的实现类，是在哪里被 spring 容器使用的：

1. 如下图所示，红框中的 invokeBeanFactoryPostProcessors 方法用来找出所有 beanFactory 后置处理器，并且调用这些处理器来改变 bean 的定义：



2. 打开 invokeBeanFactoryPostProcessors 方法，如下所示，实际操作是委托 PostProcessorRegistrationDelegate 去完成的：

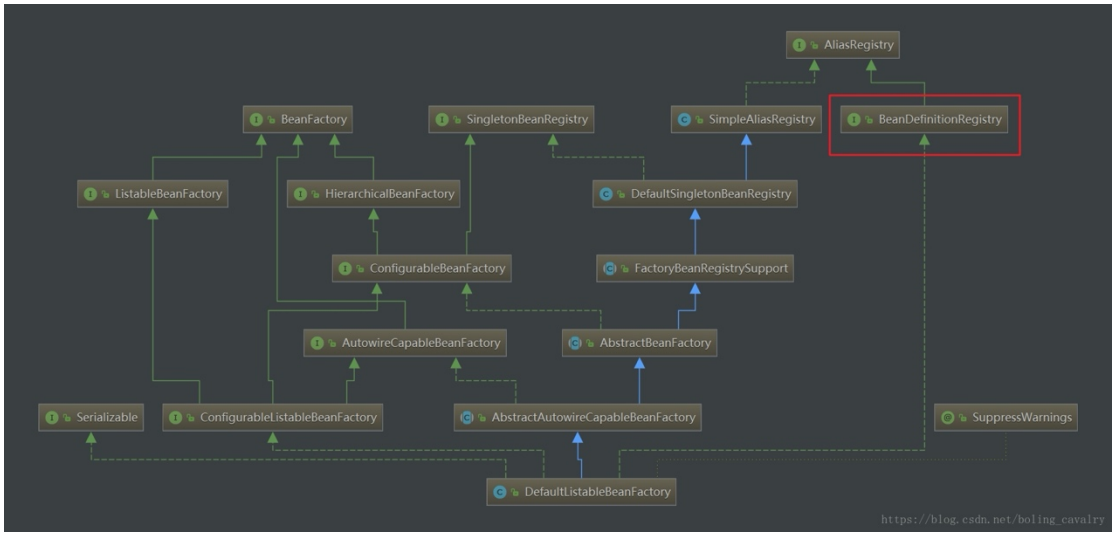
```
protected void invokeBeanFactoryPostProcessors(ConfigurableListableBeanFactory beanFactory) {

PostProcessorRegistrationDelegate.invokeBeanFactoryPostProcessors(beanFactory, getBeanFactoryPostProcessors());
    }
```

3. 继续看 PostProcessorRegistrationDelegate 类的 invokeBeanFactoryPostProcessors 方法，该方法内容太丰富，我们只看重点，第一个重点如下图红框所示，当前的 beanFactory 是否实现了接口 BeanDefinitionRegistry：

```java
public static void invokeBeanFactoryPostProcessors(
        ConfigurableListableBeanFactory beanFactory, List<BeanFactoryPostProcessor> beanFactoryPostProcessors) {

    // Invoke BeanDefinitionRegistryPostProcessors first, if any.
    Set<String> processedBeans = new HashSet<~>();

    if (beanFactory instanceof BeanDefinitionRegistry) {
        BeanDefinitionRegistry registry = (BeanDefinitionRegistry) beanFactory;
        List<BeanFactoryPostProcessor> regularPostProcessors = new LinkedList<~>();
        List<BeanDefinitionRegistryPostProcessor> registryPostProcessors =
                new LinkedList<~>();

        for (BeanFactoryPostProcessor postProcessor : beanFactoryPostProcessors) {
            if (postProcessor instanceof BeanDefinitionRegistryPostProcessor) {
                BeanDefinitionRegistryPostProcessor registryPostProcessor =
                        (BeanDefinitionRegistryPostProcessor) postProcessor;
                registryPostProcessor.postProcessBeanDefinitionRegistry(registry);
                registryPostProcessors.add(registryPostProcessor);
            }
            else {
                regularPostProcessors.add(postProcessor);
            }
        }
```

https://blog.csdn.net/boling_cavalry

为了搞清楚这个问题，我们应该看看当前 beanFactory 的继承和实现，以 springboot 中的应用为例，当前 beanFactory 的类型是 DefaultListableBeanFactory，来看看它的类图：



从上图红框可见，beanFactory 实现了 BeanDefinitionRegistry 接口，因此我们的关注点是 if 条件满足后的执行逻辑；

4. 继续看 PostProcessorRegistrationDelegate 类的 invokeBeanFactoryPostProcessors 方法，以下片段就是操作 BeanDefinitionRegistryPostProcessor 的核心逻辑：

```
boolean reiterate = true;
while (reiterate) {
    reiterate = false;
    //查出所有实现了BeanDefinitionRegistryPostProcessor接口的bean名称
    postProcessorNames = beanFactory.getBeanNamesForType(BeanDefinitionRegistryPostProcessor.class, true, false);
    for (String ppName : postProcessorNames) {
        //前面的逻辑中，已经对实现了PriorityOrdered和Ordered的bean都处理过了，因此通过processedBeans过滤，processedBeans中没有的才会在此处理
        if (!processedBeans.contains(ppName)) {
            //根据名称和类型获取bean
            BeanDefinitionRegistryPostProcessor pp = beanFactory.getBean(ppName, BeanDefinitionRegistryPostProcessor.class);
            //把已经调用过postProcessBeanDefinitionRegistry方法的bean全部放在registryPostProcessors中
            registryPostProcessors.add(pp);
            //把已经调用过postProcessBeanDefinitionRegistry方法的bean的名称全部放在processedBeans中
            processedBeans.add(ppName);
            //执行此bean的postProcessBeanDefinitionRegistry方法
            pp.postProcessBeanDefinitionRegistry(registry);
            //改变退出while的条件
            reiterate = true;
        }
    }
}

//registryPostProcessors中保存了所有执行过postProcessBeanDefinitionRegistry方法的bean,
//现在再来执行这些bean的postProcessBeanFactory方法
invokeBeanFactoryPostProcessors(registryPostProcessors, beanFactory);
//regularPostProcessors中保存的是所有入参中带来的BeanFactoryPostProcessor实现类，并且这里面已经剔除了BeanDefinitionRegistryPostProcessor的实现类，现在要让这些bean执行postProcessBeanFactory方法
invokeBeanFactoryPostProcessors(regularPostProcessors, beanFactory);

```

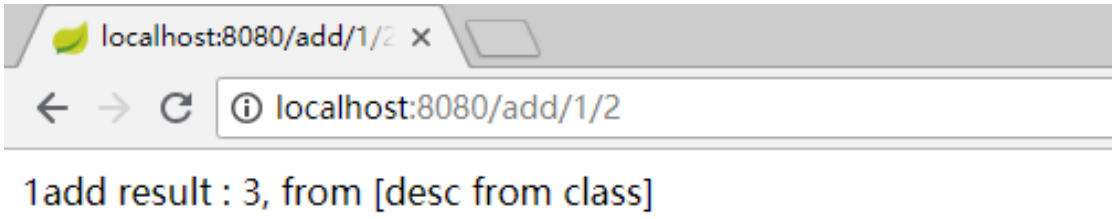
如上述代码所示，所有实现了 BeanDefinitionRegistryPostProcessor 接口的 bean，其 postProcessBeanDefinitionRegistry 方法都会调用，然后再调用其 postProcessBeanFactory 方法，这样一来，我们如果自定义了 BeanDefinitionRegistryPostProcessor 接口的实现类，那么我们开发的 postProcessBeanDefinitionRegistry 和 postProcessBeanFactory 方法都会被执行一次；

到这里，我们的源码学习部分就完成了，接下来看开始实战吧；

启动应用，在启动日志中可以看到 CustomizeBeanDefinitionRegistryPostProcessor 的方法被调用时的堆栈情况：

```
2018-08-30 18:55:40.323  INFO 14880 --- [           main] c.b.c.util.Utils                         : 复制
************************************************************
java.lang.Thread.getStackTrace() 1,556 <-
com.bolingcavalry.customizebeandefinitionregistrypostprocessor.util.Utils.printTrack() 20 <-
com.bolingcavalry.customizebeandefinitionregistrypostprocessor.registrypostprocessor.CustomizeBeanDefiniti
org.springframework.context.support.PostProcessorRegistrationDelegate.invokeBeanDefinitionRegistryPostProc
org.springframework.context.support.PostProcessorRegistrationDelegate.invokeBeanFactoryPostProcessors() 12
org.springframework.context.support.AbstractApplicationContext.invokeBeanFactoryPostProcessors() 687 <-
org.springframework.context.support.AbstractApplicationContext.refresh() 525 <-
org.springframework.boot.context.embedded.EmbeddedWebApplicationContext.refresh() 122 <-
org.springframework.boot.SpringApplication.refresh() 693 <-
org.springframework.boot.SpringApplication.refreshContext() 360 <-
org.springframework.boot.SpringApplication.run() 303 <-
org.springframework.boot.SpringApplication.run() 1,118 <-
org.springframework.boot.SpringApplication.run() 1,107 <-
com.bolingcavalry.customizebeandefinitionregistrypostprocessor.Customizebeandefinitionregistrypostprocessc
************************************************************
2018-08-30 18:55:40.542  INFO 14880 --- [           main] c.b.c.util.Utils                         : execu
************************************************************
java.lang.Thread.getStackTrace() 1,556 <-
com.bolingcavalry.customizebeandefinitionregistrypostprocessor.util.Utils.printTrack() 20 <-
com.bolingcavalry.customizebeandefinitionregistrypostprocessor.registrypostprocessor.CustomizeBeanDefiniti
org.springframework.context.support.PostProcessorRegistrationDelegate.invokeBeanFactoryPostProcessors() 28
org.springframework.context.support.PostProcessorRegistrationDelegate.invokeBeanFactoryPostProcessors() 12
org.springframework.context.support.AbstractApplicationContext.invokeBeanFactoryPostProcessors() 687 <-
org.springframework.context.support.AbstractApplicationContext.refresh() 525 <-
```

在浏览器输入：http://localhost:8080/add/1/2，如下图可以看到网页正常响应，controller 可以正常使用 calculateService 实例：



https://blog.csdn.net/boling_cavalry

去掉 CustomizeBeanDefinitionRegistryPostProcessor 的注释 @Component，重启应用，再去访问 http://localhost:8080/add/1/2，可以看到网页提示错误如下图：

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Aug 30 19:00:01 CST 2018
There was an unexpected error (type=Internal Server Error, status=500).
No message available

https://blog.csdn.net/boling_cavalry