Spring Boot 源码学习笔记

笔记本:	工作笔记		
创建时间:	2020/11/19 19:52	**更新时间:**	2020/11/19 21:20
作者:	438842220@qq.com		
URL:	about:blank		

前言

目的：

我认为学习源码的主要原因在于理解框架的执行过程，提高工作效率和读代码的能力。次要原因在于理解框架原理，学习框架中代码的写法。

spring框架的目标是通过对bean的管理（主要IOC和AOP）来提高程序员的工作效率，因此从宏观来看，spring是一个复杂的bean容器。（参考 https://blog.csdn.net/zhangcongyi420/article/details/89419715 ，aop更简单了，代理模式就实现了。）

springboot是建立在spring基础上的框架，提供了一系列的操作（读application.xx文件，注解，@component）等节省了配置成本。

方法论：

二八原则：看最重要的代码，重点掌握实现流程，而不苛求细节实现。

框架式思维：建立整体的流程图，一步步细化。

目的性：除了对整体流程的理解，还有关键功能的实现。

springboot的整体流程

启动入口：

SpringApplication.run()

listeners.starting();

prepareEnvironment();这一部分主要是准备了environment配置信息。

createApplicationContext();创建了ApplicationContext也就是BeanFactory。在这里会判断具体的环境，大多数情况是servlet环境并实例化对应写死的类AnnotationConfigServletWebServerApplicationContext。而在这个类构造函数中，会new好AnnotatedBeanDefinitionReader和ClassPathBeanDefinitionScanner类供日后使用。

prepareContext();会逐个调用initializer.initialize()。发送contextPrepared()，并调用load()加载sources

refreshContext();这是个大方法，下面细说。

afterRefresh();这部分代码是空的。

listeners.running(context);一看就懂。

callRunners(context, applicationArguments);调用ApplicationRunner&CommandLineRunner。

refreshContext()调了几层包装之后，来到了下面的代码段：

```
synchronized (this.startupShutdownMonitor) {
    // Prepare this context for refreshing.
    // 容器状态设置，初始化属性设置，检查必备属性是否存在
    prepareRefresh();

    // Tell the subclass to refresh the internal bean factory.
```

```
            // 设置beanFactory序列化ID，获取beanFactory
            ConfigurableListableBeanFactory beanFactory =
obtainFreshBeanFactory();


            // Prepare the bean factory for use in this context.
            // 设置beanFactory的一些属性，添加后置处理器，设置忽略的自动
装配接口，注册一些组件
            prepareBeanFactory(beanFactory);

            try {
            // Allows post-processing of the bean factory in context
subclasses.
            // 子类重写以在beanFactory完成创建后做进一步配置
            postProcessBeanFactory(beanFactory);

            // Invoke factory processors registered as beans in the
context.
            // 调用beandefinitionregistrypostprocessor实现向容器内添加
bean的定义，
            // 调用beanfactorypostprocessor实现向容器内bean的定义添加属
性
            invokeBeanFactoryPostProcessors(beanFactory);

            // Register bean processors that intercept bean creation.
            // 找到beanpostprocessor的实现
            // 排序后注册进容器内
            registerBeanPostProcessors(beanFactory);

            // Initialize message source for this context.
            // 初始国际化相关
            initMessageSource();

            // Initialize event multicaster for this context.
            // 初始事件广播器
            initApplicationEventMulticaster();

            // Initialize other special beans in specific context
subclasses.
            // 创建web容器
            onRefresh();

            // Check for listener beans and register them.
            // 添加容器内事件监听器到事件广播器中
            // 派发早期事件
            registerListeners();

            // Instantiate all remaining (non-lazy-init) singletons.
            // 初始化单例bean
            finishBeanFactoryInitialization(beanFactory);

            // Last step: publish corresponding event.
```

```
            // 初始化生命周期处理器，调用生命周期处理器onrefresh方法
            finishRefresh();
        }
```

至此整体流程全部梳理完毕。

部分关键内容的处理

commandlinerunner:

callrunners

这里面有两种：ApplicationRunner/CommandLineRunner，这两种，在获取命令行传值方面有一些不同后面一个有排序，然后依次执行。

Aware:

实现它就可以获取spring的一些内部属性。

原理是判断bean是否实现aware接口（ApplicationContextAwareProcessor），如果有就会注入对应类。

属性配置:

spring的属性配置方式非常多。经常用到的内容有：命令行/application.xx（但其实有很多种其他的，只是不太常用，比如操作系统配置，JAVA系统配置等）这些配置会在run.prepareEnvironment()中验证并注入environment，见下。

Environment解析:

run.prepareEnvironment()

getOrCreateEnvironment()创建 1

configureEnvironment()主要处理非application系列的参数

listeners.environmentPrepared()发送事件

ConfigFileApplicationListener会监听事件

RandomValuePropertySource.addToEnvironment(environment)会处理random相关的

Loader(environment, resourceLoader).load()会处理application相关的。在读取默认的文件时，会有如下方法。

addActiveProfiles()

addIncludedProfiles();这两个会把profile加到profiles里，再做循环

loaded.add();这个方法会把属性添加到array，并调用foreach方法。逐一处理。

listener原理:

ApplicationEventMulticaster广播

```
for (final ApplicationListener<?> listener :
getApplicationListeners(event,type))
invokeListener(listener,event);
```

如何获得listeners集合:

registerListeners

```
getApplicationEventMulticaster().addApplicationListener(listener)
```

将this.applicationListeners添加到广播器

而this.applicationListeners的内容如何来的？分散在之前的各个步骤。

```
getBeanNamesForType(ApplicationListener.class, true, false);
```

获取ApplicationListener.class类，并在后面加入到广播器

Component怎么被感知的？

createApplicationContext()

```
BeanUtils.instantiateClass(contextClass);
```

AnnotationConfigServletWebServerApplicationContext()

```
public AnnotationConfigServletWebServerApplicationContext() {
this.reader = new AnnotatedBeanDefinitionReader(this);
this.scanner = new ClassPathBeanDefinitionScanner(this);
```

```
        }
        scanner.scan()
        doscan()

    bean初始化流程
        getbean 获取beandefinition，遍历
        dogetbean 处理别名，确认是否正在创建or已经创建
        getsingleton 执行参数的getobject方法，跳到createbean
        createbean 做resolve的预处理，override的预处理
        resolvebeforeinstantiation
applyBeanPostProcessorsBeforeInstantiation和
applyBeanPostProcessorsAfterInitialization
        docreatebean 先createbeaninstance，然后
applyMergedBeanDefinitionPostProcessors
        createbeaninstance 做一些特殊情况的处理，否则走下面的方法
        instantiatebean getInstantiationStrategy()（实现代理模式）
        instantiate 反射创建bean
        populatebean 处理autoware和
InstantiationAwareBeanPostProcessors
        initializebean 初始化完毕。
invokeAwareMethods&applyBeanPostProcessorsBeforeInitialization&invokeInitMethods
```