基础知识

beandefinition 包含什么

1. Class 文件
2. scope
3. lazy

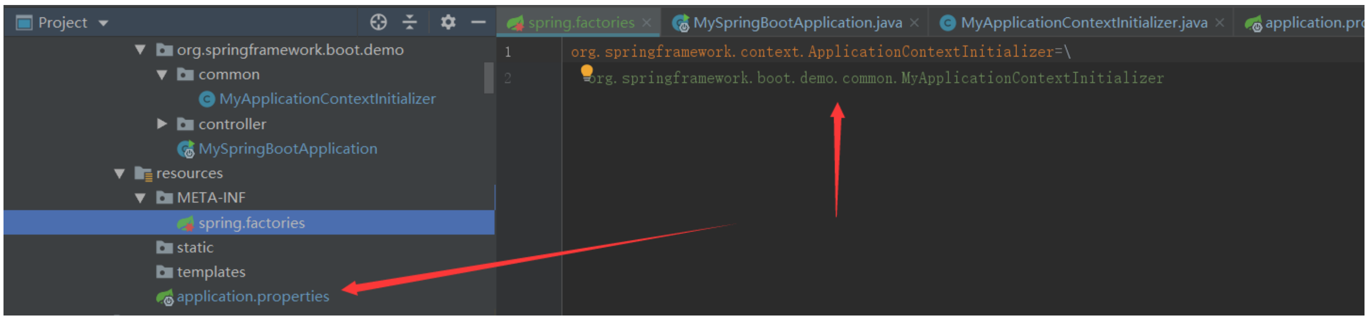
如何启动的 lancher

1. 详见jar包接口中的 main-class 和 start-class
2. 对应jvm的classloader

容器启动

ApplicationContextInitializer

```
1.  容器刷新之前会调用该类的 initialize 方法， 并将context 传入进去，
2.  通常用于 根据上下文环境注册属性元 或者激活配置文件

是用方法

1. application.addInitializers(new MyApplicationContextInitializer());
2. 配置文件中配置

context.initializer.classes=org.springframework.boot.demo.common.MyApplicationCont
extInitializer
3. SpringBoot的SPI扩展---META-INF/spring.factories
```



```
org.springframework.context.ApplicationContextInitializer=\
org.springframework.boot.demo.common.MyApplicationContextInitializer
```

refresh 流程关键节点

springboot PrepareContext

1. setEnvironment
2. applyInitializers
3. logStartupInfo
4. load 加载并注册主类 beandefinition

invokeBeanFactoryPostProcessors

ConfigurableListableBeanFactory

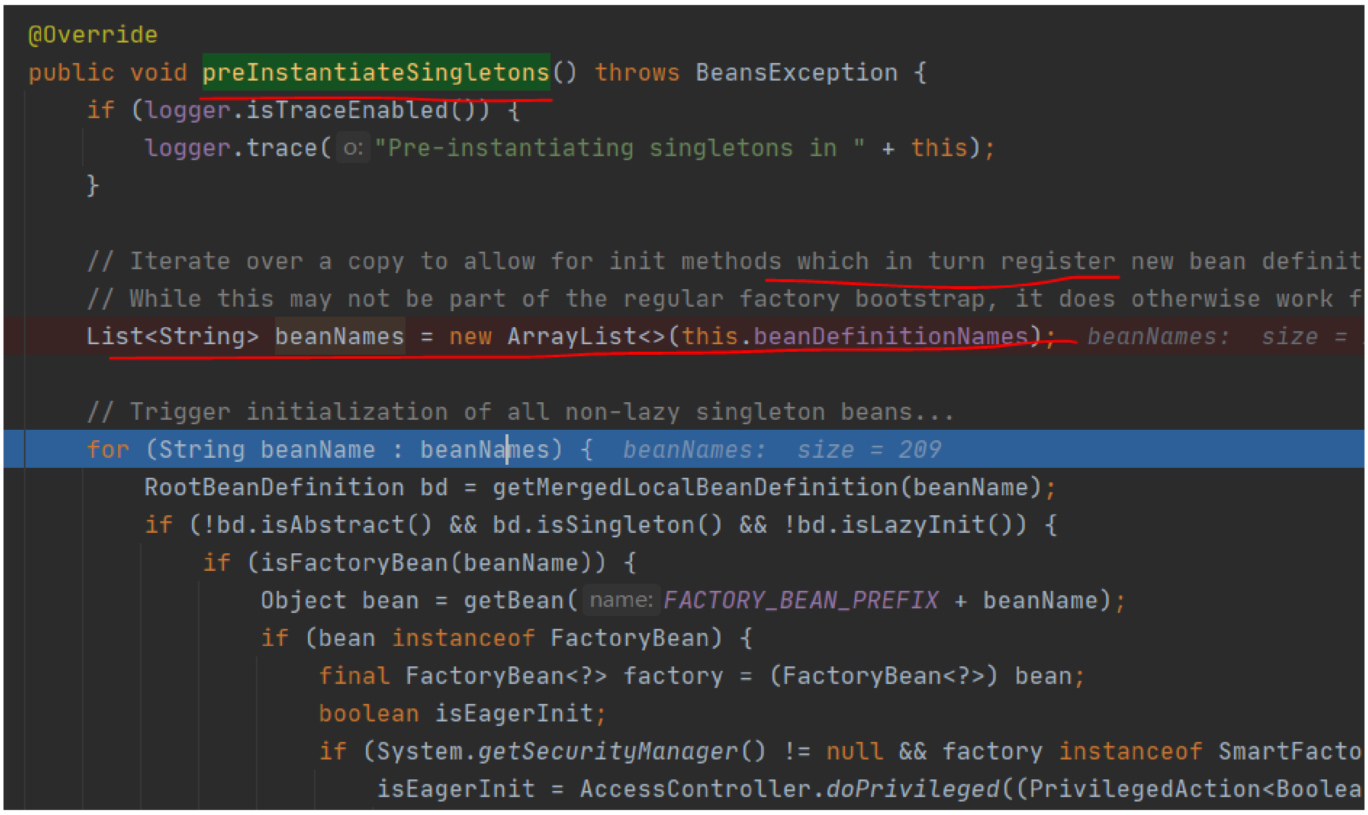
```
    核心功能
        1. 加载跟配置文件 或者与配置相关的注解  相关的bean处理 流程
        2. 准备 bean definition
        3. 可以用来做部分bean的初始化(一般是框架相关的，比如
EventListenerMethodProcessor,  一般会初始化其余依赖的bean)

    核心实现接口
        1.  BeanFactoryPostProcessor#postProcessBeanFactory  主要作用是 加载所有
bean definition，但是不初始化
        2.  BeanDefinitionRegistryPostProcessor#postProcessBeanDefinitionRegistry
在 1 中加载的 bean definition 之前，先进行一些操作
             以ConfigurationClassPostProcessor#processConfigBeanDefinitions 为例
             会处理， resource, componet, importSelector 等等
             见 ConfigurationClassPostProcessor#processConfigBeanDefinitions ->
ConfigurationClassParser#doProcessConfigurationClass
                ConfigurationClassParser#processImports()

    关于main
        1. 上述的接口都是由Main 启动
        2. 并且 Main 也被声明称 beandefinition ,
        3. beanFactoryPostprocessor 和 beandefinitionRefistrypostProcessor 会对
Main进行处理， 解析 上面的注解
        4. 特别是 Main 上的 SpringBootApplication, Enable***, 等等
        5. ConfigurationClassParser#processImports() 是有递归操作的，比如解析到一个
Configurarion, 里面包含了 ImportSerect,  会对 importselector 返回的Class继续以
Configuration 进行解析



    相关回调接口或实现
        BeanDefinitionRegistryPostProcessor
        BeanFactoryPostProcessor
        ConfigurationClassPostProcessor
                ImportSelector
                Configuration
```

finishBeanFactoryInitialization

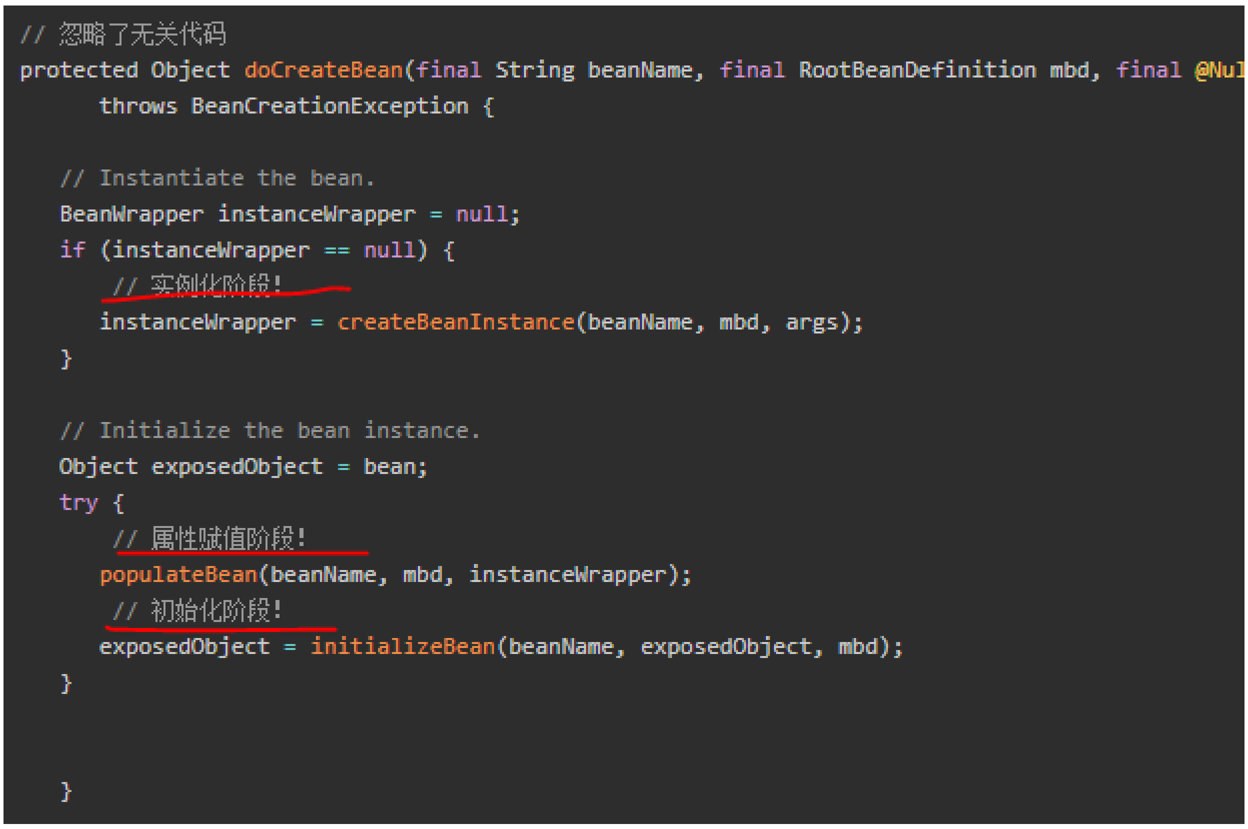
https://www.jianshu.com/p/1dec08d290c1

```
核心功能
    1. 管理bean的生命周期 ()
        实例化：是对象创建的过程。比如使用构造方法new对象，为对象在内存中分配空间。
        设置属性：如果属性是依赖注入的其他bean，走一遍getBean方法
        初始化：调用aware方法、bean后置处理器的初始化前方法、初始化方法、bean后置处理器
```

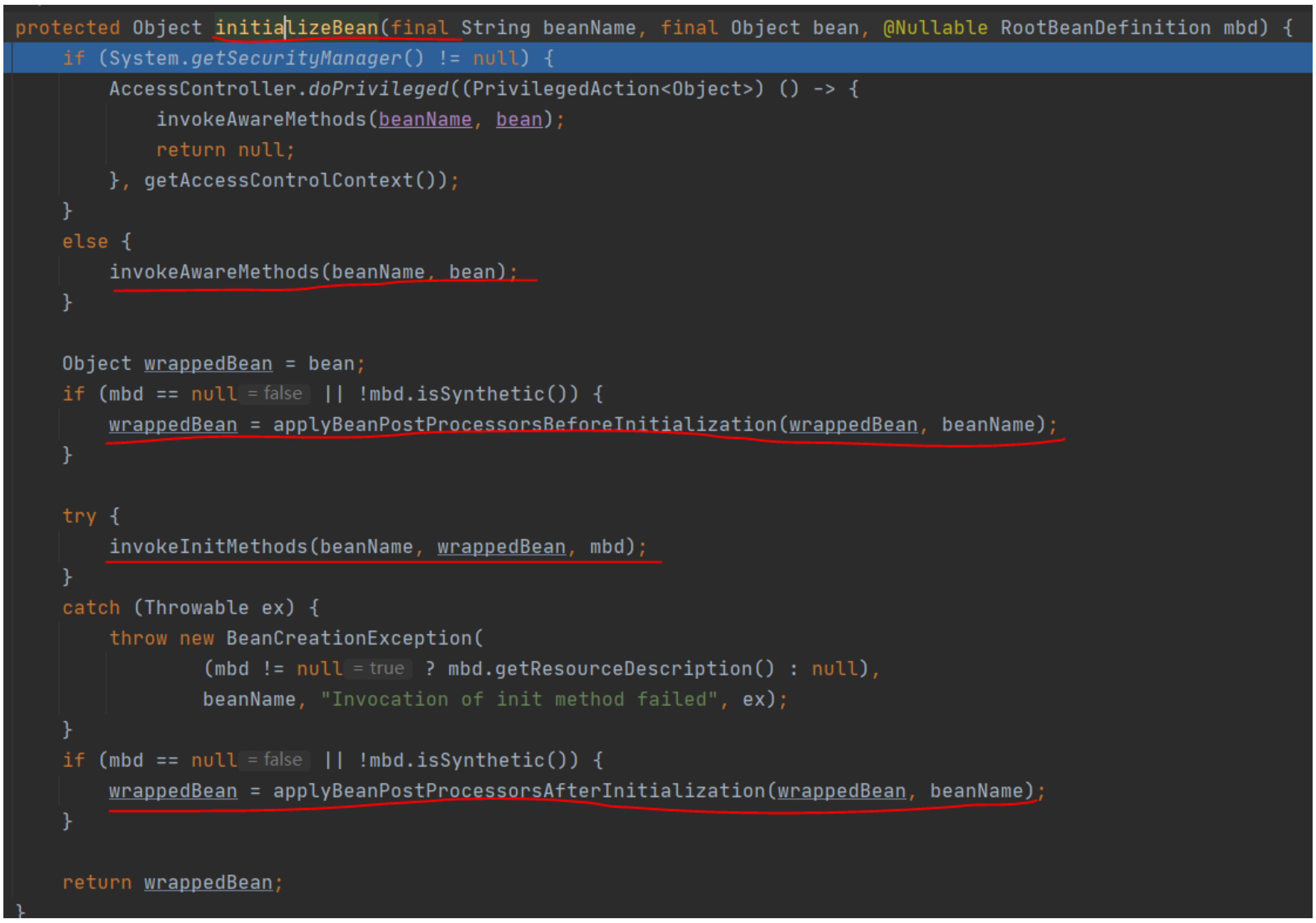
```
的初始化后方法


    2. 执行 bean 初始化相应的钩子函数

核心相关接口
    BeanPostProcessor
    FactoryBean


核心流程

    DefaultListableBeanFactory#preInstantiateSingletons
    AbstractBeanFactory#doGetBean
    AbstractBeanFactory#getObjectForBeanInstance
    FactoryBeanRegistrySupport#getObjectFromFactoryBean
```



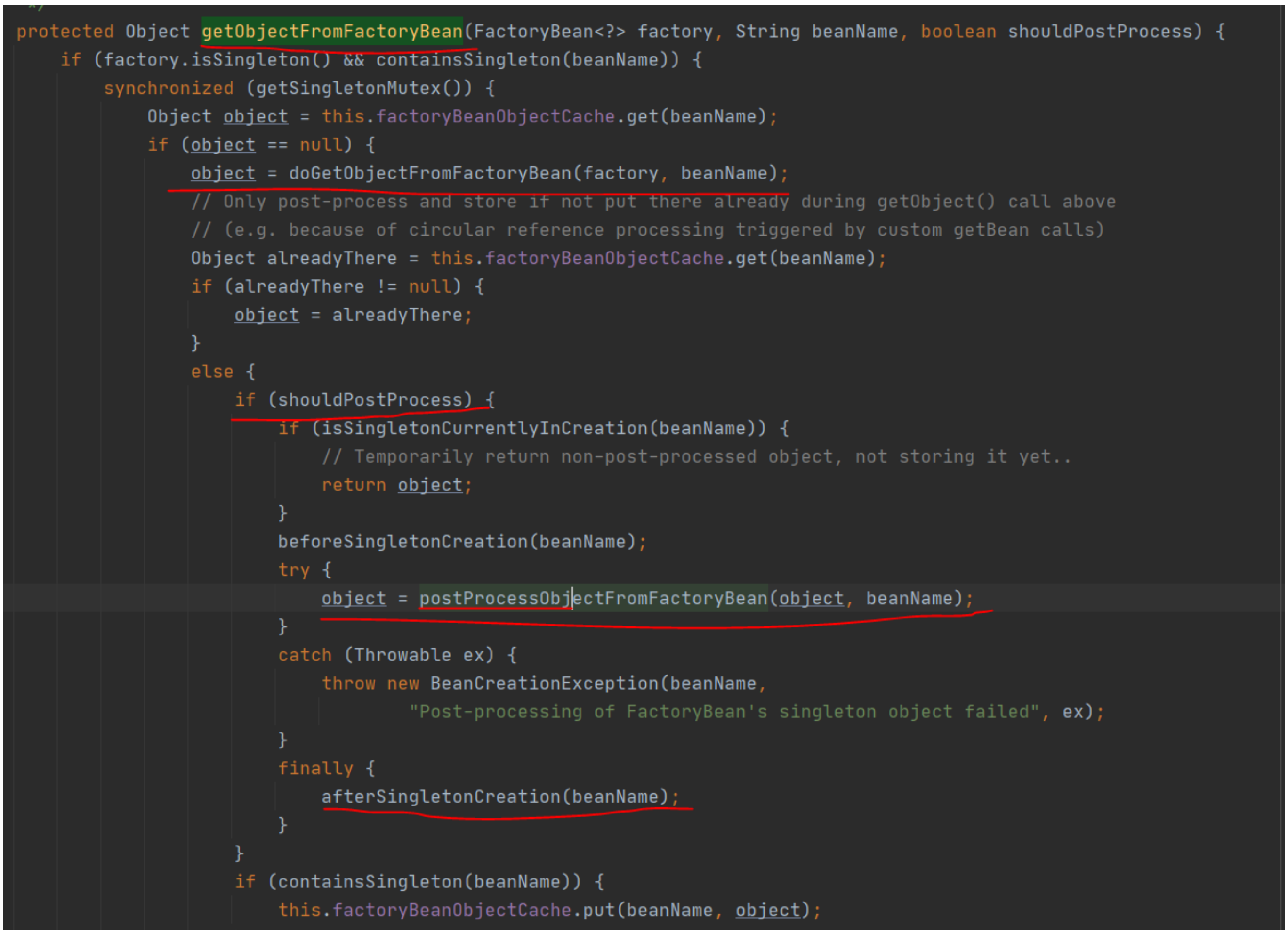
```
@Override
public void preInstantiateSingletons() throws BeansException {
    if (logger.isTraceEnabled()) {
        logger.trace( o: "Pre-instantiating singletons in " + this);
    }

    // Iterate over a copy to allow for init methods which in turn register new bean definit
    // While this may not be part of the regular factory bootstrap, it does otherwise work f
    List<String> beanNames = new ArrayList<>(this.beanDefinitionNames);  beanNames:  size =
    
    // Trigger initialization of all non-lazy singleton beans...
    for (String beanName : beanNames) {  beanNames:  size = 209
        RootBeanDefinition bd = getMergedLocalBeanDefinition(beanName);
        if (!bd.isAbstract() && bd.isSingleton() && !bd.isLazyInit()) {
            if (isFactoryBean(beanName)) {
                Object bean = getBean( name: FACTORY_BEAN_PREFIX + beanName);
                if (bean instanceof FactoryBean) {
                    final FactoryBean<?> factory = (FactoryBean<?>) bean;
                    boolean isEagerInit;
                    if (System.getSecurityManager() != null && factory instanceof SmartFacto
                        isEagerInit = AccessController.doPrivileged((PrivilegedAction<Boolea
```



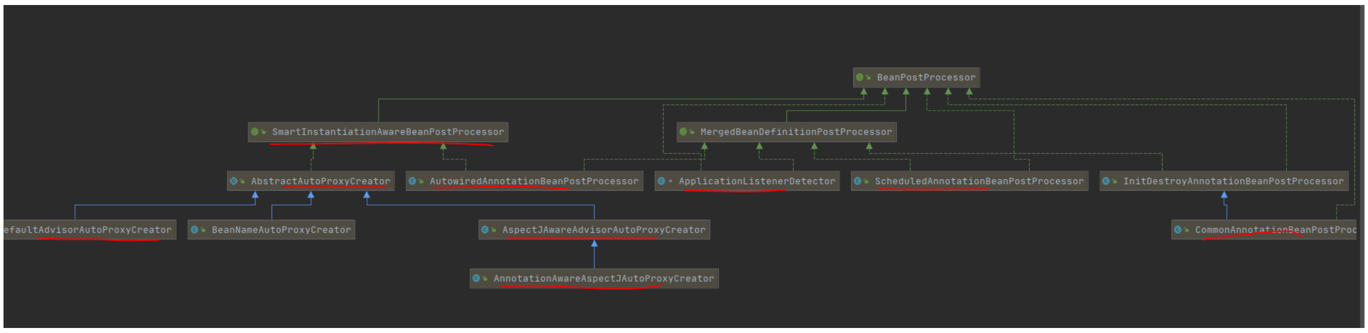
```
// 忽略了无关代码
protected Object doCreateBean(final String beanName, final RootBeanDefinition mbd, final @Nul
        throws BeanCreationException {

    // Instantiate the bean.
    BeanWrapper instanceWrapper = null;
    if (instanceWrapper == null) {
        // 实例化阶段!
        instanceWrapper = createBeanInstance(beanName, mbd, args);
    }

    // Initialize the bean instance.
    Object exposedObject = bean;
    try {
        // 属性赋值阶段!
        populateBean(beanName, mbd, instanceWrapper);
        // 初始化阶段!
        exposedObject = initializeBean(beanName, exposedObject, mbd);
    }


    }
```



```java
protected Object initializeBean(final String beanName, final Object bean, @Nullable RootBeanDefinition mbd) {
    if (System.getSecurityManager() != null) {
        AccessController.doPrivileged((PrivilegedAction<Object>) () -> {
            invokeAwareMethods(beanName, bean);
            return null;
        }, getAccessControlContext());
    }
    else {
        invokeAwareMethods(beanName, bean);
    }

    Object wrappedBean = bean;
    if (mbd == null = false  || !mbd.isSynthetic()) {
        wrappedBean = applyBeanPostProcessorsBeforeInitialization(wrappedBean, beanName);
    }

    try {
        invokeInitMethods(beanName, wrappedBean, mbd);
    }
    catch (Throwable ex) {
        throw new BeanCreationException(
                (mbd != null = true  ? mbd.getResourceDescription() : null),
                beanName, "Invocation of init method failed", ex);
    }
    if (mbd == null = false  || !mbd.isSynthetic()) {
        wrappedBean = applyBeanPostProcessorsAfterInitialization(wrappedBean, beanName);
    }

    return wrappedBean;
}
```



```java
protected Object getObjectFromFactoryBean(FactoryBean<?> factory, String beanName, boolean shouldPostProcess) {
    if (factory.isSingleton() && containsSingleton(beanName)) {
        synchronized (getSingletonMutex()) {
            Object object = this.factoryBeanObjectCache.get(beanName);
            if (object == null) {
                object = doGetObjectFromFactoryBean(factory, beanName);
                // Only post-process and store if not put there already during getObject() call above
                // (e.g. because of circular reference processing triggered by custom getBean calls)
                Object alreadyThere = this.factoryBeanObjectCache.get(beanName);
                if (alreadyThere != null) {
                    object = alreadyThere;
                }
                else {
                    if (shouldPostProcess) {
                        if (isSingletonCurrentlyInCreation(beanName)) {
                            // Temporarily return non-post-processed object, not storing it yet..
                            return object;
                        }
                        beforeSingletonCreation(beanName);
                        try {
                            object = postProcessObjectFromFactoryBean(object, beanName);
                        }
                        catch (Throwable ex) {
                            throw new BeanCreationException(beanName,
                                    "Post-processing of FactoryBean's singleton object failed", ex);
                        }
                        finally {
                            afterSingletonCreation(beanName);
                        }
                    }
                    if (containsSingleton(beanName)) {
                        this.factoryBeanObjectCache.put(beanName, object);
```


BeanPostProcessor
SmartInstantiationAwareBeanPostProcessor
MergedBeanDefinitionPostProcessor
AbstractAutoProxyCreator
AutowiredAnnotationBeanPostProcessor
ApplicationListenerDetector
ScheduledAnnotationBeanPostProcessor
InitDestroyAnnotationBeanPostProcessor
efaultAdvisorAutoProxyCreator
BeanNameAutoProxyCreator
AspectJAwareAdvisorAutoProxyCreator
CommonAnnotationBeanPostProc
AnnotationAwareAspectJAutoProxyCreator

```
```

BeanPostProcessor

```
但是BeanPostProcessor只能在初始化后（注意初始化不包括init方法）执行一些操作
```

IOC

bean的生命周期

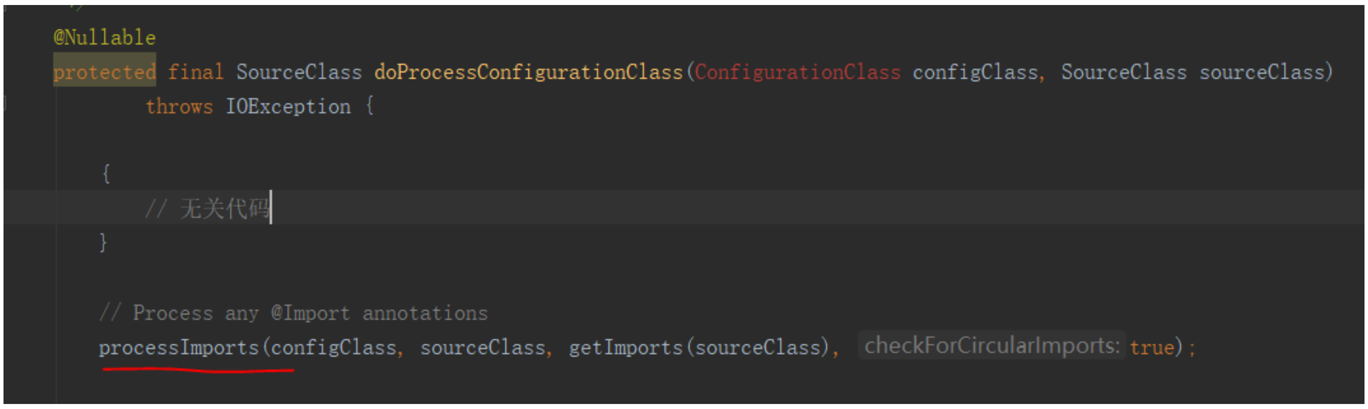
DI

核心

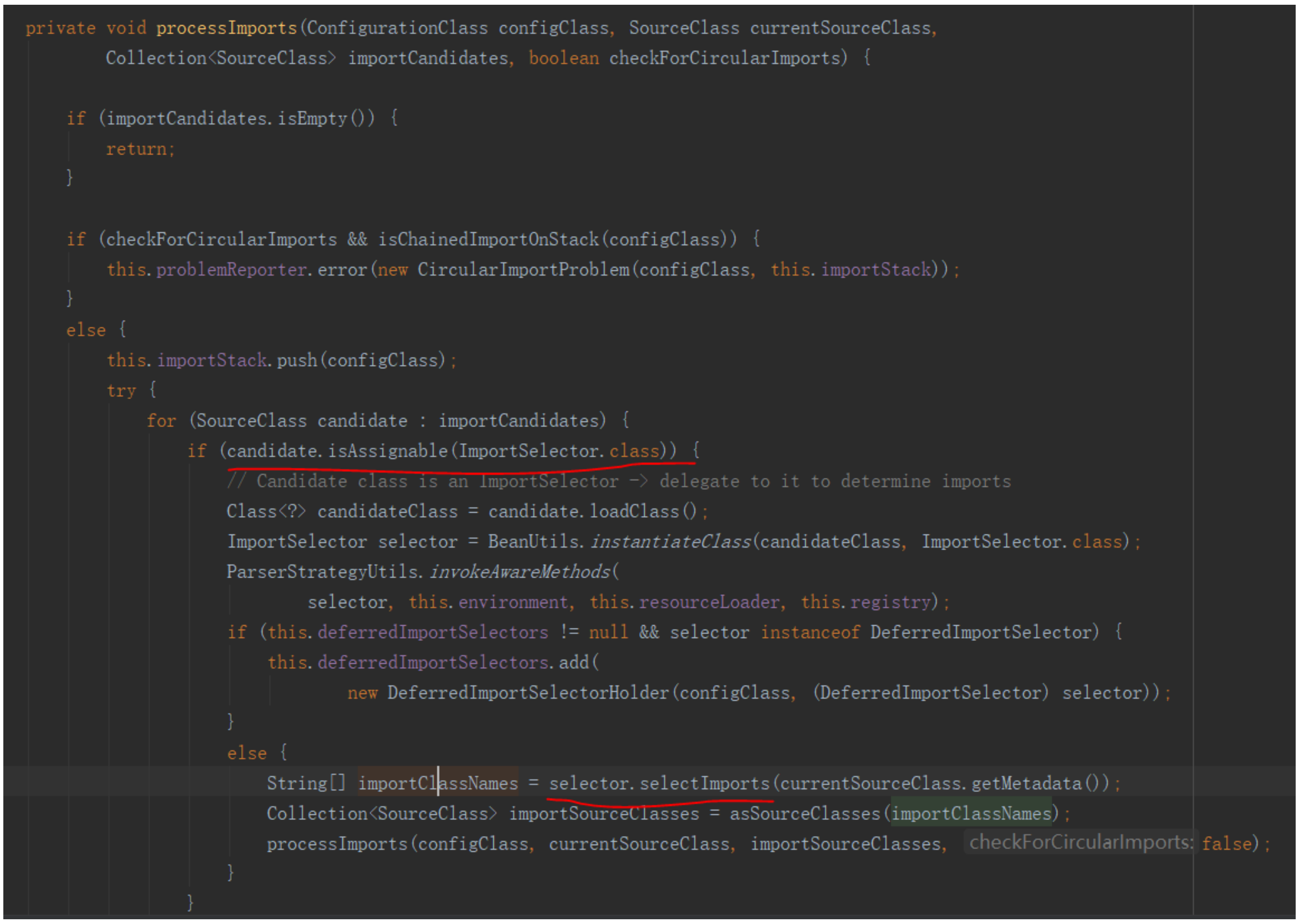
```
ImportSelector接口的返回值会递归进行解析，把解析到的类全名按照@Configuration进行处理
```

Import 与 ImportSelector

```
org.springframework.context.annotation.ConfigurationClassParser#doProcessConfigurationClass
```



```
@Nullable
protected final SourceClass doProcessConfigurationClass(ConfigurationClass configClass, SourceClass sourceClass)
        throws IOException {

    {
        // 无关代码
    }

    // Process any @Import annotations
    processImports(configClass, sourceClass, getImports(sourceClass), checkForCircularImports: true);
```



```java
private void processImports(ConfigurationClass configClass, SourceClass currentSourceClass,
        Collection<SourceClass> importCandidates, boolean checkForCircularImports) {

    if (importCandidates.isEmpty()) {
        return;
    }

    if (checkForCircularImports && isChainedImportOnStack(configClass)) {
        this.problemReporter.error(new CircularImportProblem(configClass, this.importStack));
    }
    else {
        this.importStack.push(configClass);
        try {
            for (SourceClass candidate : importCandidates) {
                if (candidate.isAssignable(ImportSelector.class)) {
                    // Candidate class is an ImportSelector -> delegate to it to determine imports
                    Class<?> candidateClass = candidate.loadClass();
                    ImportSelector selector = BeanUtils.instantiateClass(candidateClass, ImportSelector.class);
                    ParserStrategyUtils.invokeAwareMethods(
                            selector, this.environment, this.resourceLoader, this.registry);
                    if (this.deferredImportSelectors != null && selector instanceof DeferredImportSelector) {
                        this.deferredImportSelectors.add(
                                new DeferredImportSelectorHolder(configClass, (DeferredImportSelector) selector));
                    }
                    else {
                        String[] importClassNames = selector.selectImports(currentSourceClass.getMetadata());
                        Collection<SourceClass> importSourceClasses = asSourceClasses(importClassNames);
                        processImports(configClass, currentSourceClass, importSourceClasses, checkForCircularImports: false);
                    }
                }
```

BeanDefinition 与 FactoryBean

https://blog.csdn.net/forezp/article/details/83896098

```
设置definition 为  FeignClientFactoryBean bean,
BeanDefinitionBuilder definition =
BeanDefinitionBuilder.genericBeanDefinition(FeignClientFactoryBean.class);


进行初始化的时候，通过 FactoryBean.getObject() 获取 bean 对象。
```

Import 与 ImportBeanDefinitionRegistrar

```
示例
    RibbonClientConfigurationRegistrar
    MapperScannerRegistrar
```

Import 与 Configuration

```

```

```
示例
    Configuration
```