Spring注解驱动开发第16讲——面试官再问你BeanPostProcessor的执行流程，就把这篇文章甩给他！

写在前面

在前面的文章中，我们讲述了BeanPostProcessor的postProcessBeforeInitialization()方法和postProcessAfterInitialization()方法在bean初始化的前后调用。而且我们可以自定义类来实现BeanPostProcessor接口，并在postProcessBeforeInitialization()方法和postProcessAfterInitialization()方法中编写我们自定义的逻辑。

今天，我们来一起探讨下BeanPostProcessor的底层原理。

bean的初始化和销毁

我们知道BeanPostProcessor的postProcessBeforeInitialization()方法是在bean的初始化之前被调用；而postProcessAfterInitialization()方法是在bean初始化的之后被调用。并且bean的初始化和销毁方法我们可以通过如下方式进行指定。

（一）通过@Bean指定init-method和destroy-method

```java
@Bean(initMethod="init", destroyMethod="destroy")
public Car car() {
    return new Car();
}
```
AI写代码java运行

（二）通过让bean实现InitializingBean和DisposableBean这俩接口

```java
package com.meimeixia.bean;

import org.springframework.beans.factory.DisposableBean;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
public class Cat implements InitializingBean, DisposableBean {

    public Cat() {
        System.out.println("cat constructor...");
    }

    /**
     * 会在容器关闭的时候进行调用
     */
    @Override
    public void destroy() throws Exception {
        // TODO Auto-generated method stub
        System.out.println("cat destroy...");
    }

    /**
     * 会在bean创建完成，并且属性都赋好值以后进行调用
     */
    @Override
    public void afterPropertiesSet() throws Exception {
        // TODO Auto-generated method stub
        System.out.println("cat afterPropertiesSet...");
    }

}
```
AI写代码java运行

（三）使用JSR-250规范里面定义的@PostConstruct和@PreDestroy这俩注解

- @PostConstruct：在bean创建完成并且属性赋值完成之后，来执行初始化方法
- @PreDestroy：在容器销毁bean之前通知我们进行清理工作

```java
package com.meimeixia.bean;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

import org.springframework.stereotype.Component;

/**
 *
 * @author liayun
 *
 */
@Component
public class Dog {

    public Dog() {
        System.out.println("dog constructor...");
    }

    // 在对象创建完成并且属性赋值完成之后调用
    @PostConstruct
    public void init() {
        System.out.println("dog...@PostConstruct...");
    }

    // 在容器销毁（移除）对象之前调用
    @PreDestroy
    public void destory() {
        System.out.println("dog...@PreDestroy...");
    }

}
```

AI写代码java运行

（四）通过让bean实现BeanPostProcessor接口

```
package com.meimeixia.bean;

import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.core.Ordered;
import org.springframework.stereotype.Component;

/**
 * 后置处理器，在初始化前后进行处理工作
 * @author liayun
 *
 */
@Component // 将后置处理器加入到容器中，这样的话，Spring就能让它工作了
public class MyBeanPostProcessor implements BeanPostProcessor, Ordered {

    @Override
    public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
        // TODO Auto-generated method stub
        System.out.println("postProcessBeforeInitialization..." + beanName + "=>" + bean);
        return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {
        // TODO Auto-generated method stub
        System.out.println("postProcessAfterInitialization..." + beanName + "=>" + bean);
        return bean;
    }

    @Override
    public int getOrder() {
        // TODO Auto-generated method stub
        return 3;
    }

}
```

AI写代码java运行

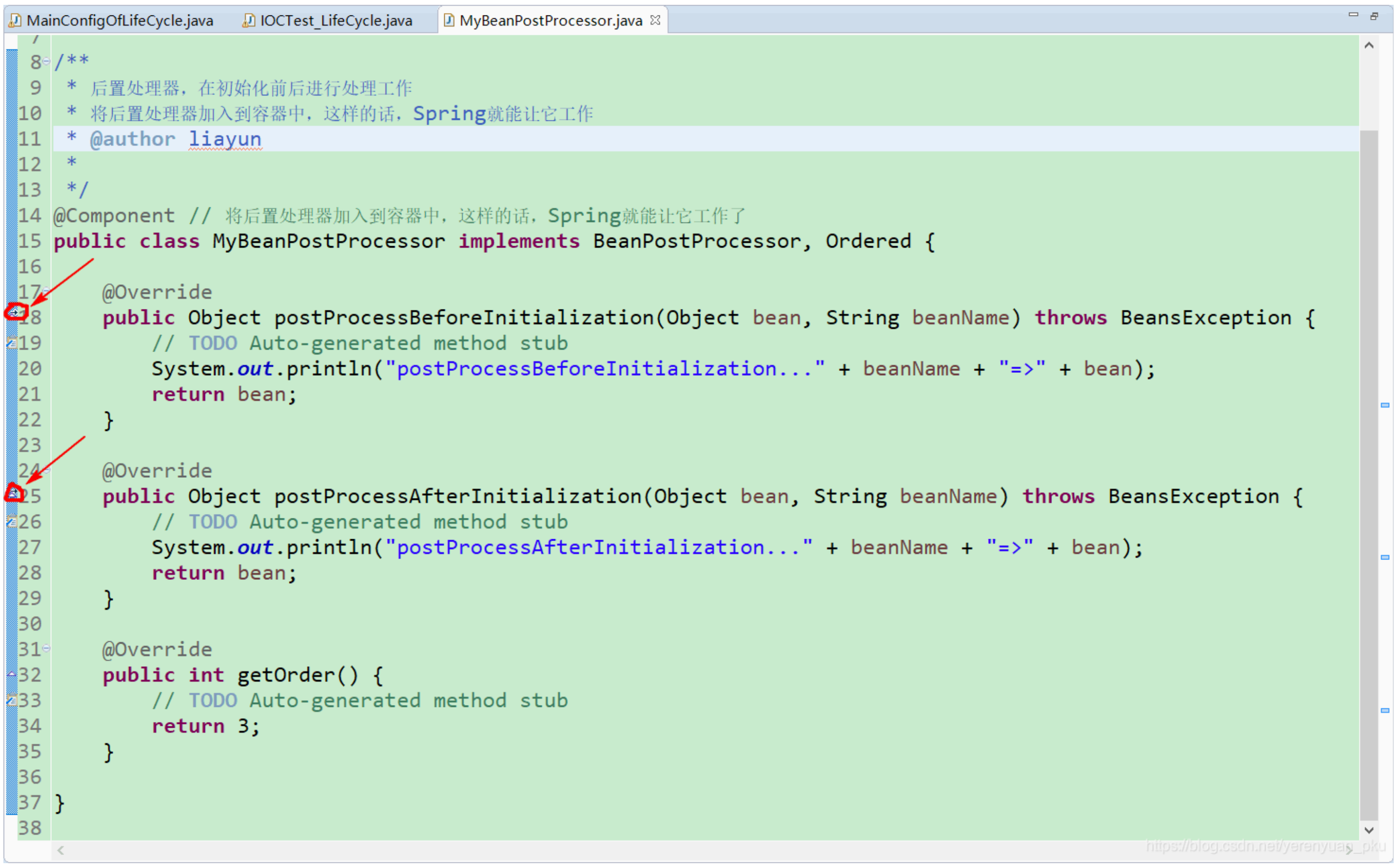
通过以上这四种方式，我们就可以对bean的整个生命周期 进行控制：

- bean的实例化：调用bean的构造方法，我们可以在bean的无参构造方法中执行相应的逻辑。
- bean的初始化：在初始化时，可以通过BeanPostProcessor的postProcessBeforeInitialization()方法和postProcessAfterInitialization()方法进行拦截，执行自定义的逻辑；通过@PostConstruct注解、InitializingBean和init-method来指定bean初始化前后执行的方法，在该方法中咱们可以执行自定义的逻辑。
- bean的销毁：可以通过@PreDestroy注解、DisposableBean和destroy-method来指定bean在销毁前执行的方法，在该方法中咱们可以执行自定义的逻辑。

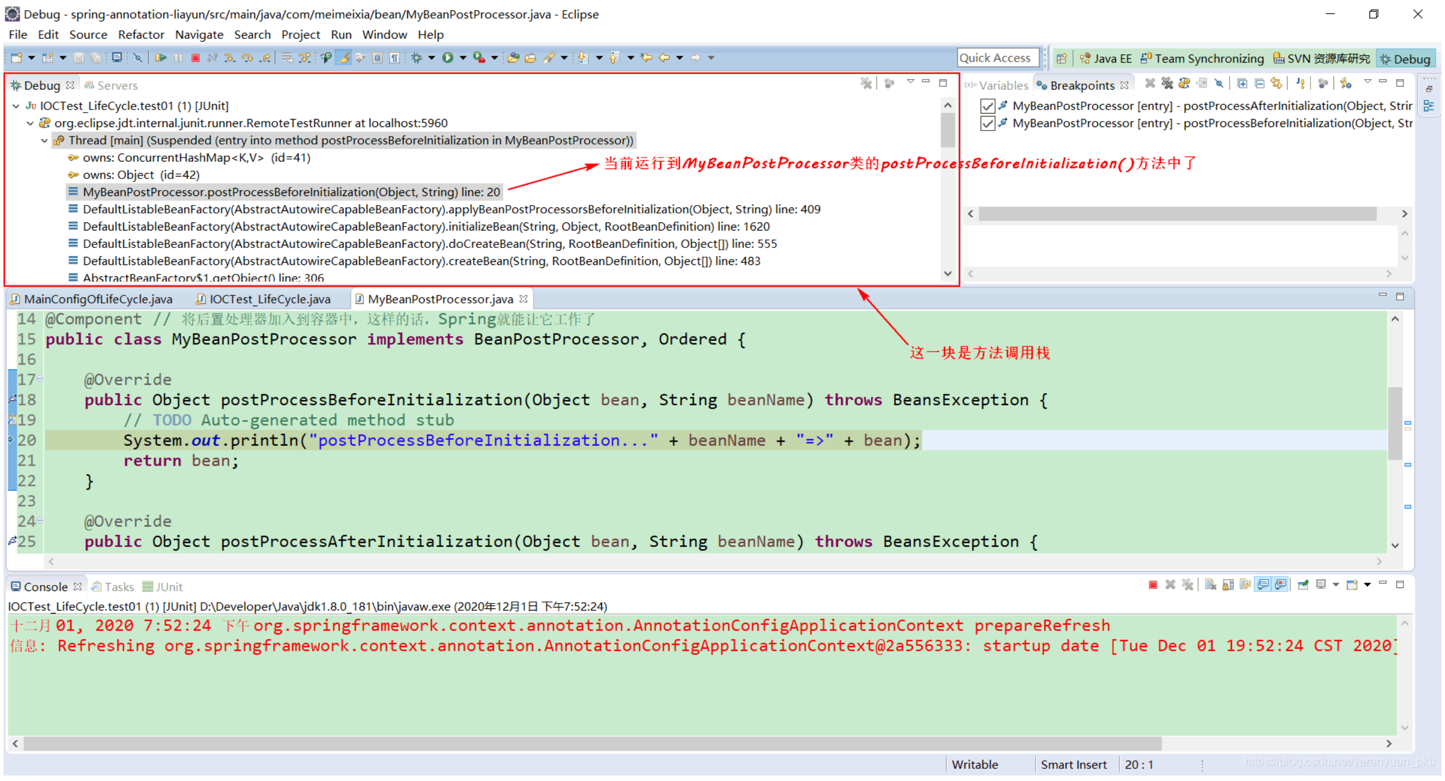
所以，通过上述四种方式，我们可以控制Spring中bean的整个生命周期。

BeanPostProcessor源码解析

如果想深刻理解BeanPostProcessor的工作原理，那么就不得不看下相关的源码，我们可以在MyBeanPostProcessor类的postProcessBeforeInitialization()方法和postProcessAfterInitialization()方法这两处打上断点来进行调试，如下所示。


```java
/**
 * 后置处理器，在初始化前后进行处理工作
 * 将后置处理器加入到容器中，这样的话，Spring就能让它工作
 * @author liayun
 *
 */
@Component // 将后置处理器加入到容器中，这样的话，Spring就能让它工作了
public class MyBeanPostProcessor implements BeanPostProcessor, Ordered {

    @Override
    public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
        // TODO Auto-generated method stub
        System.out.println("postProcessBeforeInitialization..." + beanName + "=>" + bean);
        return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {
        // TODO Auto-generated method stub
        System.out.println("postProcessAfterInitialization..." + beanName + "=>" + bean);
        return bean;
    }

    @Override
    public int getOrder() {
        // TODO Auto-generated method stub
        return 3;
    }

}
```

随后，我们以Debug的方式来运行IOCTest_LifeCycle类中的test01()方法，运行后的效果如下所示。


Debug - spring-annotation-liayun/src/main/java/com/meimeixia/bean/MyBeanPostProcessor.java - Eclipse

```
IOCTest_LifeCycle.test01 (1) [JUnit]
  org.eclipse.jdt.internal.junit.runner.RemoteTestRunner at localhost:5960
    Thread [main] (Suspended (entry into method postProcessBeforeInitialization in MyBeanPostProcessor))
      owns: ConcurrentHashMap<K,V>  (id=41)
      owns: Object  (id=42)
      MyBeanPostProcessor.postProcessBeforeInitialization(Object, String) line: 20
      DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).applyBeanPostProcessorsBeforeInitialization(Object, String) line: 409
      DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).initializeBean(String, Object, RootBeanDefinition) line: 1620
      DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).doCreateBean(String, RootBeanDefinition, Object[]) line: 555
      DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).createBean(String, RootBeanDefinition, Object[]) line: 483
      AbstractBeanFactory$1.getObject() line: 306
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Strir
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, Str
```

当前运行到MyBeanPostProcessor类的postProcessBeforeInitialization()方法中了

这一块是方法调用栈

```java
@Component // 将后置处理器加入到容器中，这样的话，Spring就能让它工作了
public class MyBeanPostProcessor implements BeanPostProcessor, Ordered {

    @Override
    public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
        // TODO Auto-generated method stub
        System.out.println("postProcessBeforeInitialization..." + beanName + "=>" + bean);
        return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {
```

```
IOCTest_LifeCycle.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2020年12月1日 下午7:52:24)
十二月 01, 2020 7:52:24 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh
信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@2a556333: startup date [Tue Dec 01 19:52:24 CST 2020]
```

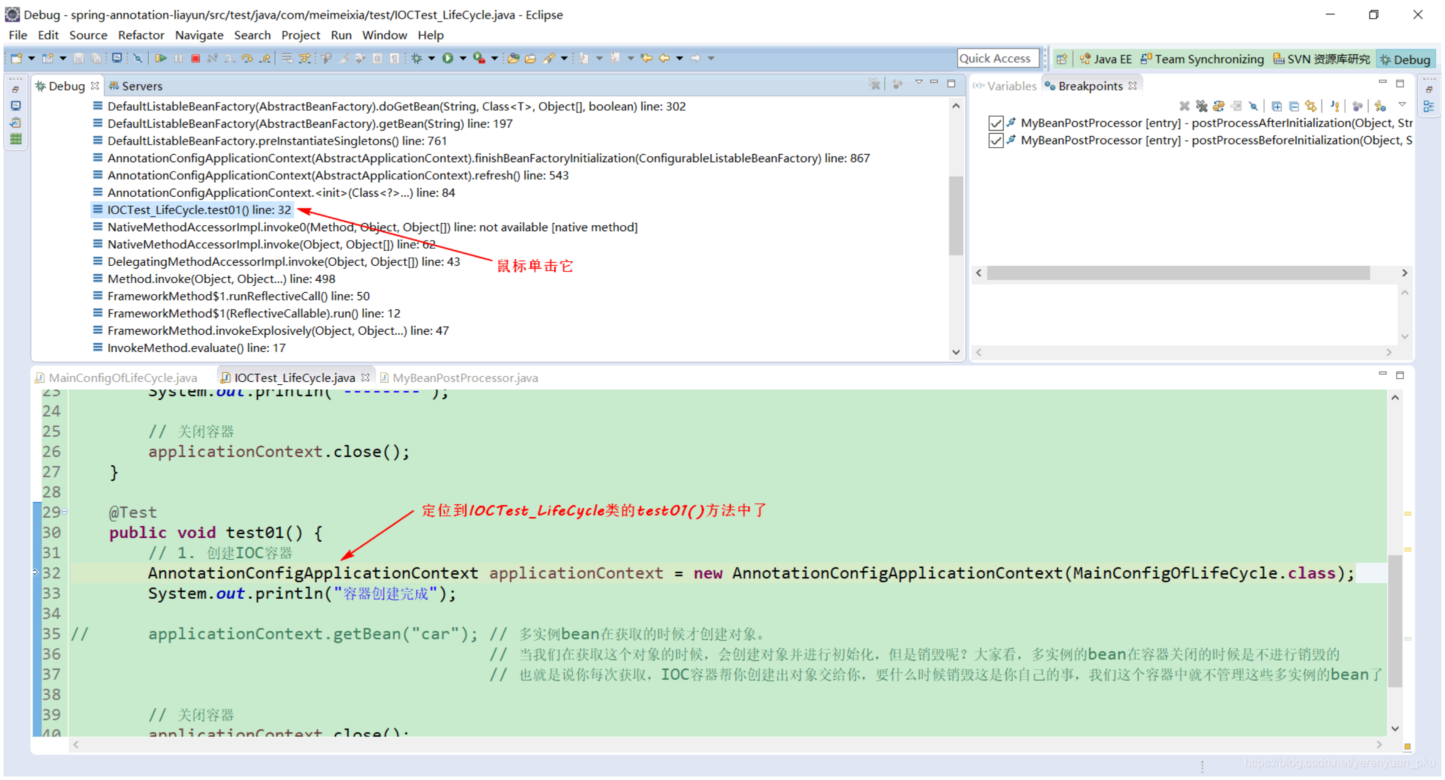
可以看到，程序已经运行到MyBeanPostProcessor类的postProcessBeforeInitialization()方法中了，而且在Eclipse 的左上角我们可以清晰的看到方法的调用栈。

通过这个方法调用栈，我们可以详细地分析从运行IOCTest_LifeCycle类中的test01()方法开始，到进入MyBeanPostProcessor类的postProcessBeforeInitialization()方法中的执行流程。只要我们在Eclipse的方法调用栈中找到IOCTest_LifeCycle类的test01()方法，依次分析方法调用栈中在该类的test01()方法上面位置的方法，即可了解整个方法调用栈的过程。要想定位方法调用栈中的方法，只需要在Eclipse的方法调用栈中单击相应的方法即可。

温馨提示：方法调用栈是先进后出的，也就是说，最先调用的方法会最后退出，每调用一个方法，JVM会将当前调用的方法放入栈的栈顶，方法退出时，会将方法从栈顶的位置弹出。

接下来，就跟随着笔者的脚步，一步一步来分析从运行IOCTest_LifeCycle类中的test01()方法开始，到进入MyBeanPostProcessor类的postProcessBeforeInitialization()方法中的执行流程。

第一步，我们在Eclipse的方法调用栈中，找到IOCTest_LifeCycle类的test01()方法并单击，此时Eclipse的主界面会定位到IOCTest_LifeCycle类的test01()方法中，如下所示。


Debug - spring-annotation-liayun/src/test/java/com/meimeixia/test/IOCTest_LifeCycle.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Team Synchronizing
SVN 资源库研究
Debug
Debug
Servers
Variables
Breakpoints
MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
鼠标单击它
定位到IOCTest_LifeCycle类的test01()方法中了

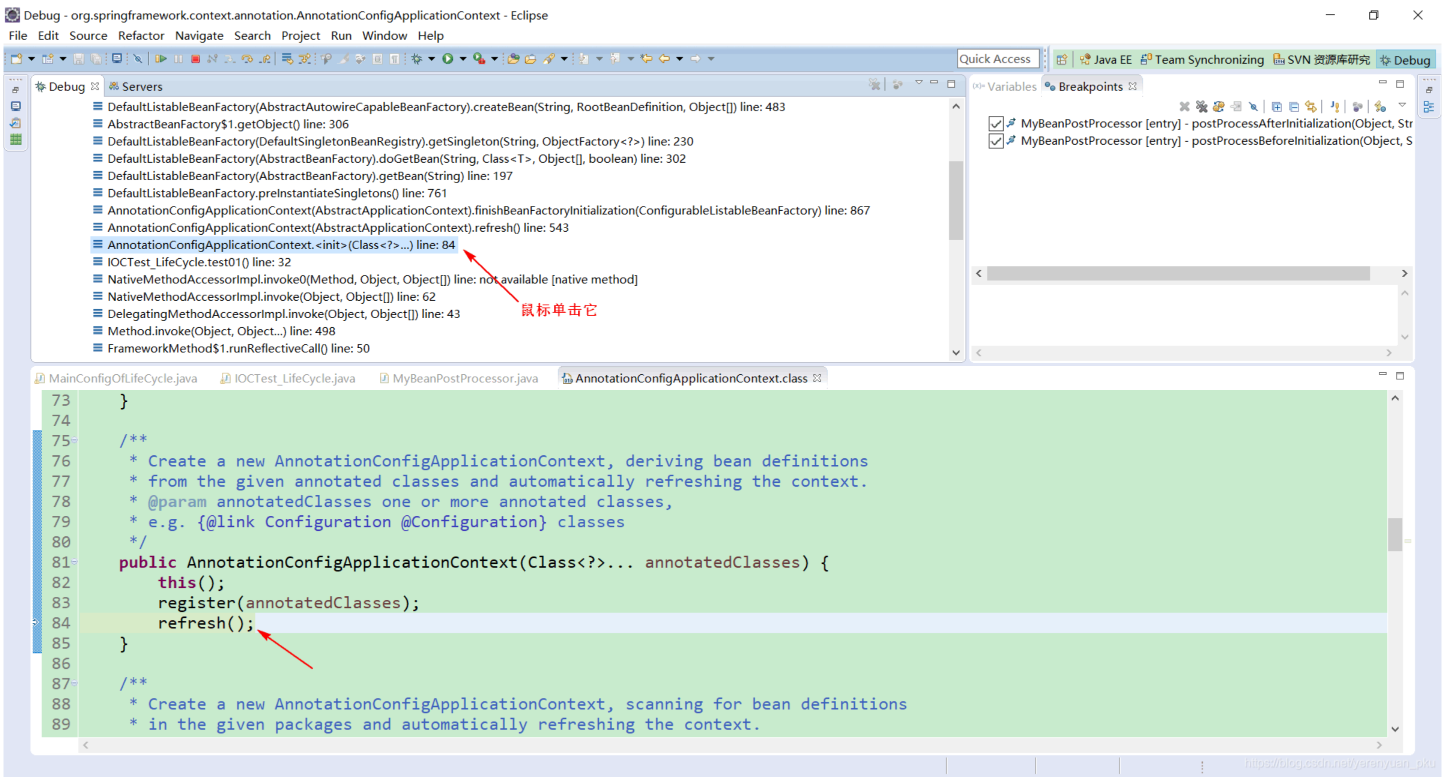
```
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
DefaultListableBeanFactory(AbstractBeanFactory).getBean(String) line: 197
DefaultListableBeanFactory.preInstantiateSingletons() line: 761
AnnotationConfigApplicationContext(AbstractApplicationContext).finishBeanFactoryInitialization(ConfigurableListableBeanFactory) line: 867
AnnotationConfigApplicationContext(AbstractApplicationContext).refresh() line: 543
AnnotationConfigApplicationContext.<init>(Class<?>...) line: 84
IOCTest_LifeCycle.test01() line: 32
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
Method.invoke(Object, Object...) line: 498
FrameworkMethod$1.runReflectiveCall() line: 50
FrameworkMethod$1(ReflectiveCallable).run() line: 12
FrameworkMethod.invokeExplosively(Object, Object...) line: 47
InvokeMethod.evaluate() line: 17
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java
        System.out.println("--------");

        // 关闭容器
        applicationContext.close();
    }

    @Test
    public void test01() {
        // 1. 创建IOC容器
        AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfLifeCycle.class);
        System.out.println("容器创建完成");

//        applicationContext.getBean("car"); // 多实例bean在获取的时候才创建对象。
                                             // 当我们在获取这个对象的时候，会创建对象并进行初始化，但是销毁呢？大家看，多实例的bean在容器关闭的时候是不进行销毁的
                                             // 也就是说你每次获取，IOC容器帮你创建出对象交给你，要什么时候销毁这是你自己的事，我们这个容器中就不管理这些多实例的bean了

        // 关闭容器
        applicationContext.close();
```

在IOCTest_LifeCycle类的test01()方法中，首先通过new实例对象的方式创建了一个IOC容器。

第二步，通过Eclipse的方法调用栈继续分析，单击IOCTest_LifeCycle类的test01()方法上面的那个方法，这时会进入AnnotationConfigApplicationContext类的构造方法中。


Debug - org.springframework.context.annotation.AnnotationConfigApplicationContext - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Team Synchronizing
SVN 资源库研究
Debug
Debug
Servers
Variables
Breakpoints
MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
AnnotationConfigApplicationContext.class
鼠标单击它

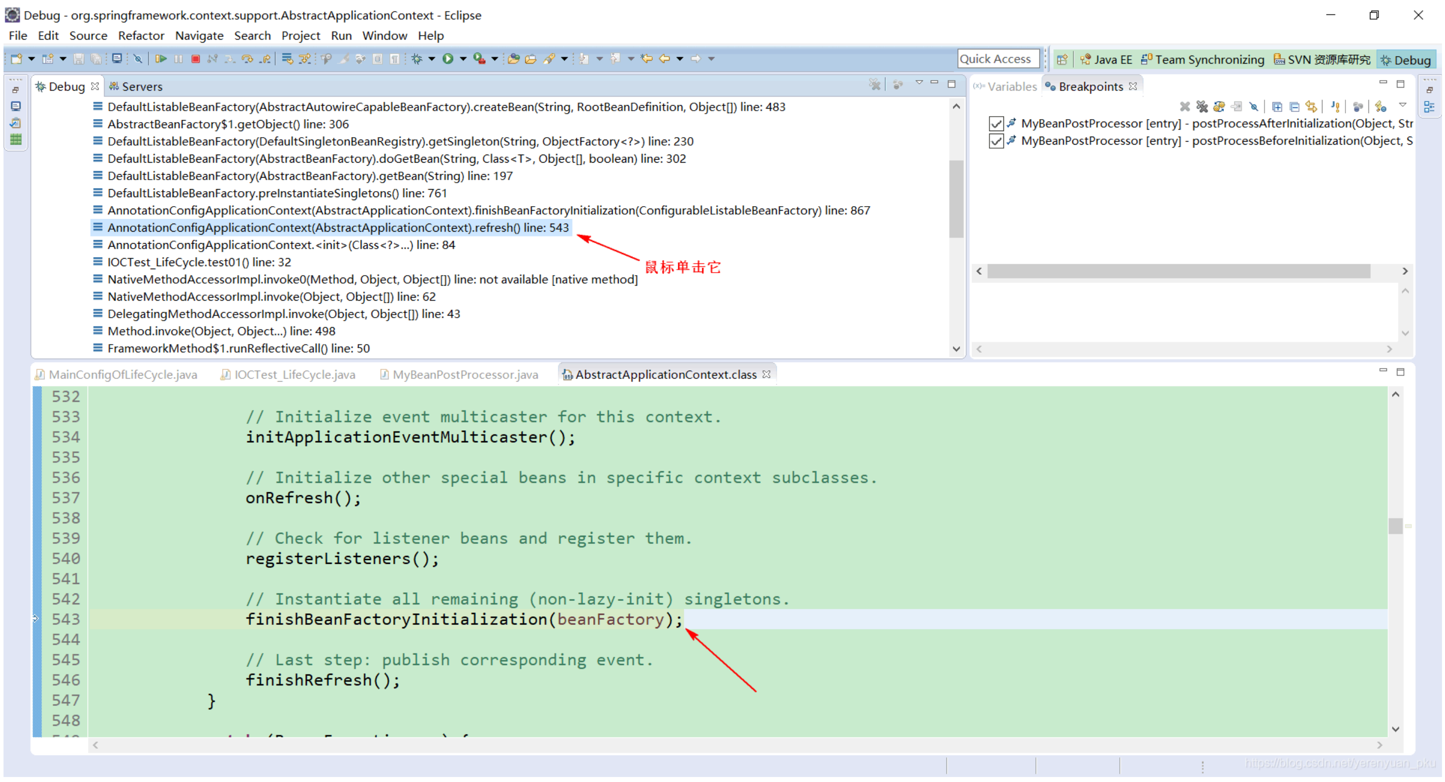
```
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).createBean(String, RootBeanDefinition, Object[]) line: 483
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
DefaultListableBeanFactory(AbstractBeanFactory).getBean(String) line: 197
DefaultListableBeanFactory.preInstantiateSingletons() line: 761
AnnotationConfigApplicationContext(AbstractApplicationContext).finishBeanFactoryInitialization(ConfigurableListableBeanFactory) line: 867
AnnotationConfigApplicationContext(AbstractApplicationContext).refresh() line: 543
AnnotationConfigApplicationContext.<init>(Class<?>...) line: 84
IOCTest_LifeCycle.test01() line: 32
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
Method.invoke(Object, Object...) line: 498
FrameworkMethod$1.runReflectiveCall() line: 50
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java
    }

    /**
     * Create a new AnnotationConfigApplicationContext, deriving bean definitions
     * from the given annotated classes and automatically refreshing the context.
     * @param annotatedClasses one or more annotated classes,
     * e.g. {@link Configuration @Configuration} classes
     */
    public AnnotationConfigApplicationContext(Class<?>... annotatedClasses) {
        this();
        register(annotatedClasses);
        refresh();
    }

    /**
     * Create a new AnnotationConfigApplicationContext, scanning for bean definitions
     * in the given packages and automatically refreshing the context.
```

可以看到，在AnnotationConfigApplicationContext类的构造方法中会调用refresh()方法。

第三步，我们继续跟进方法调用栈，如下所示，可以看到，方法的执行定位到AbstractApplicationContext类的refresh()方法中的如下那行代码处。


Debug - org.springframework.context.support.AbstractApplicationContext - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Team Synchronizing
SVN 资源库研究
Debug
Debug
Servers
Variables
Breakpoints
MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
AbstractApplicationContext.class
鼠标单击它

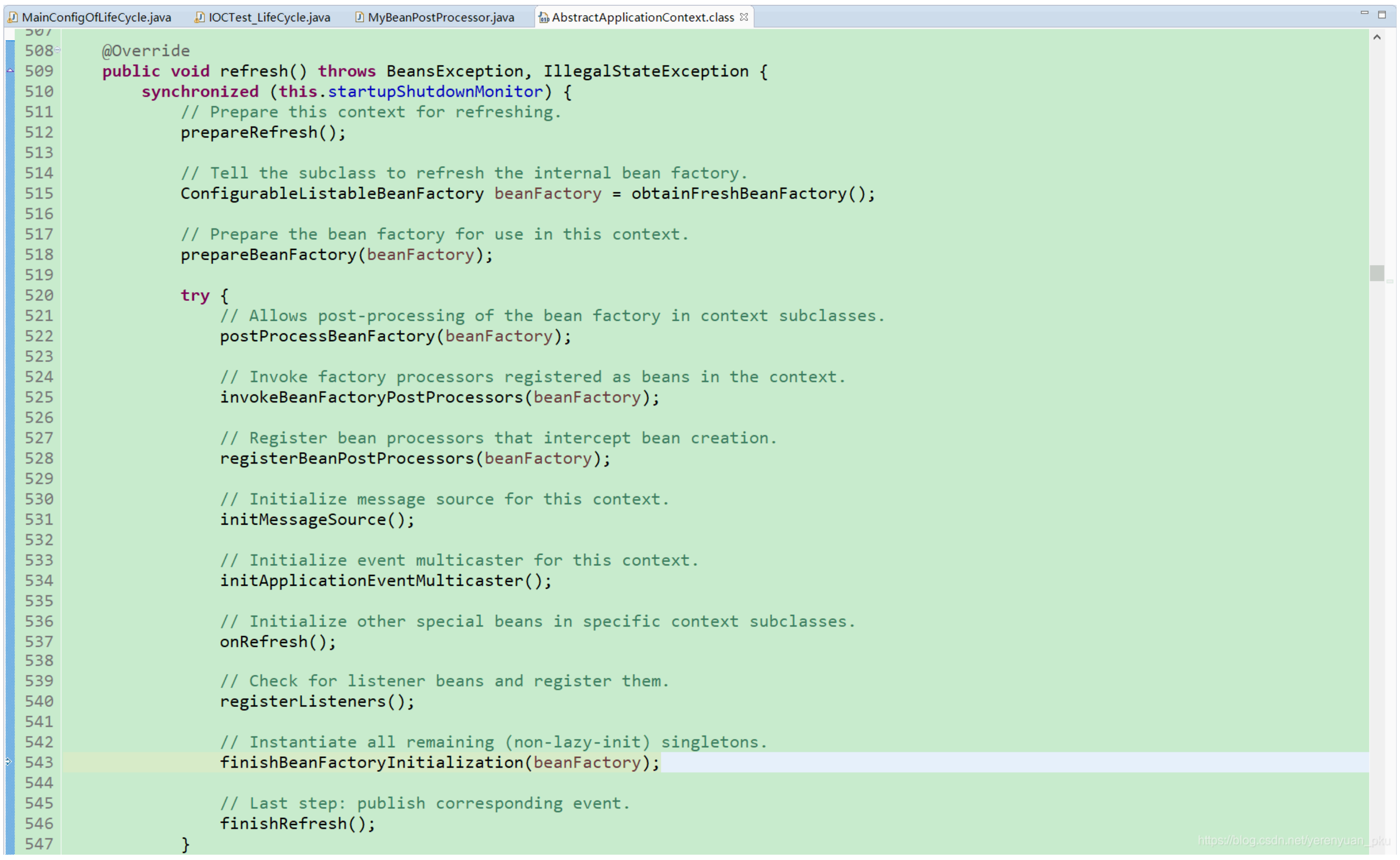
```
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).createBean(String, RootBeanDefinition, Object[]) line: 483
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
DefaultListableBeanFactory(AbstractBeanFactory).getBean(String) line: 197
DefaultListableBeanFactory.preInstantiateSingletons() line: 761
AnnotationConfigApplicationContext(AbstractApplicationContext).finishBeanFactoryInitialization(ConfigurableListableBeanFactory) line: 867
AnnotationConfigApplicationContext(AbstractApplicationContext).refresh() line: 543
AnnotationConfigApplicationContext.<init>(Class<?>...) line: 84
IOCTest_LifeCycle.test01() line: 32
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
Method.invoke(Object, Object...) line: 498
FrameworkMethod$1.runReflectiveCall() line: 50
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java

            // Initialize event multicaster for this context.
            initApplicationEventMulticaster();

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

            // Check for listener beans and register them.
            registerListeners();

            // Instantiate all remaining (non-lazy-init) singletons.
            finishBeanFactoryInitialization(beanFactory);

            // Last step: publish corresponding event.
            finishRefresh();
        }

```

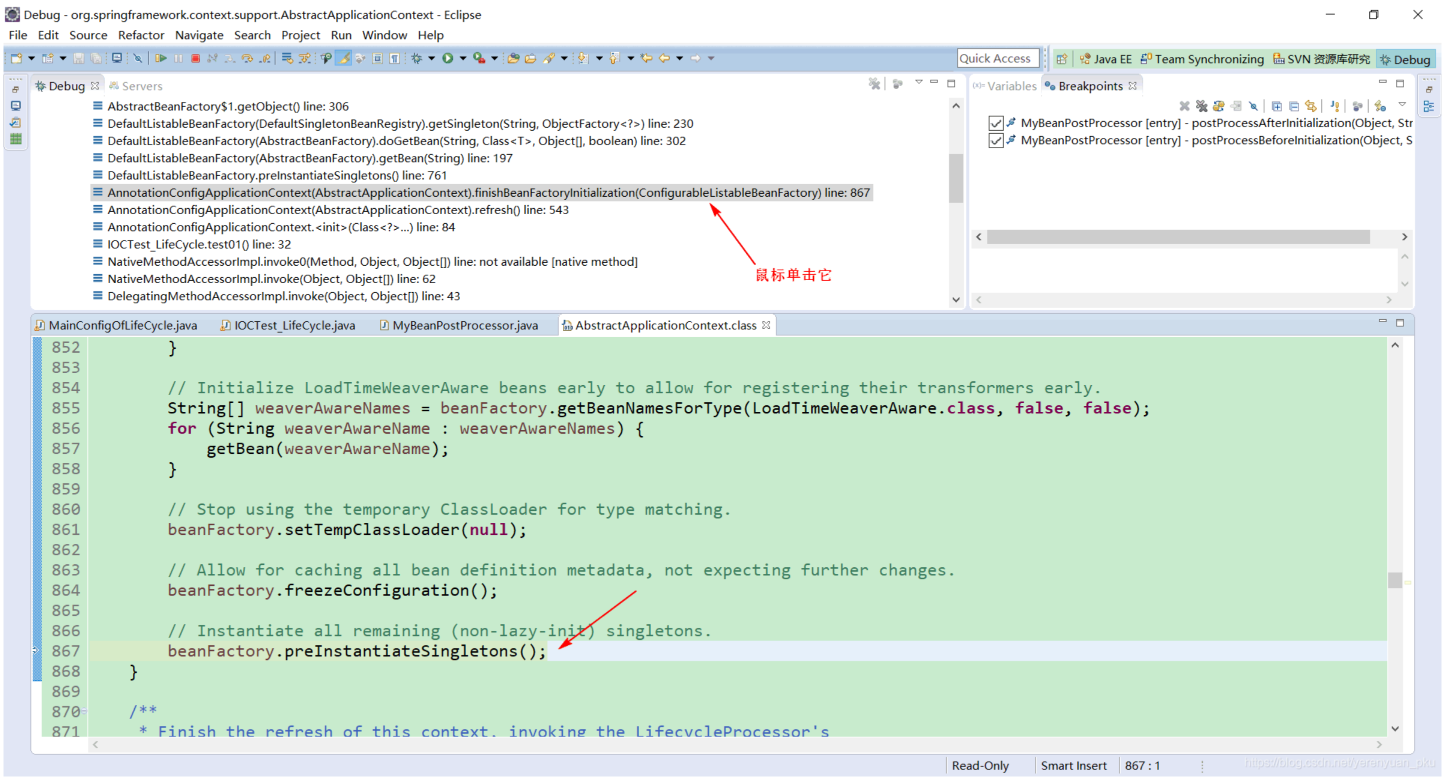
上面这行代码的作用就是初始化所有的（非懒加载的）单实例bean对象。

AbstractApplicationContext类中的refresh()方法有点长，我怕同学们看不清，所以又截了一张图，如下所示，可以清楚地看到refresh()方法里面调用了finishBeanFactoryInitialization()方法。



```java
    @Override
    public void refresh() throws BeansException, IllegalStateException {
        synchronized (this.startupShutdownMonitor) {
            // Prepare this context for refreshing.
            prepareRefresh();

            // Tell the subclass to refresh the internal bean factory.
            ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

            // Prepare the bean factory for use in this context.
            prepareBeanFactory(beanFactory);

            try {
                // Allows post-processing of the bean factory in context subclasses.
                postProcessBeanFactory(beanFactory);

                // Invoke factory processors registered as beans in the context.
                invokeBeanFactoryPostProcessors(beanFactory);

                // Register bean processors that intercept bean creation.
                registerBeanPostProcessors(beanFactory);

                // Initialize message source for this context.
                initMessageSource();

                // Initialize event multicaster for this context.
                initApplicationEventMulticaster();

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

                // Check for listener beans and register them.
                registerListeners();

                // Instantiate all remaining (non-lazy-init) singletons.
                finishBeanFactoryInitialization(beanFactory);

                // Last step: publish corresponding event.
                finishRefresh();
            }
```

第四步，我们继续跟进方法调用栈，如下所示，可以看到，方法的执行定位到AbstractApplicationContext类的finishBeanFactoryInitialization()方法中的如下那行代码处。


Debug - org.springframework.context.support.AbstractApplicationContext - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Team Synchronizing
SVN 资源库研究
Debug
Debug
Servers
Variables
Breakpoints
鼠标单击它
MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
AbstractApplicationContext.class
Read-Only
Smart Insert
867 : 1

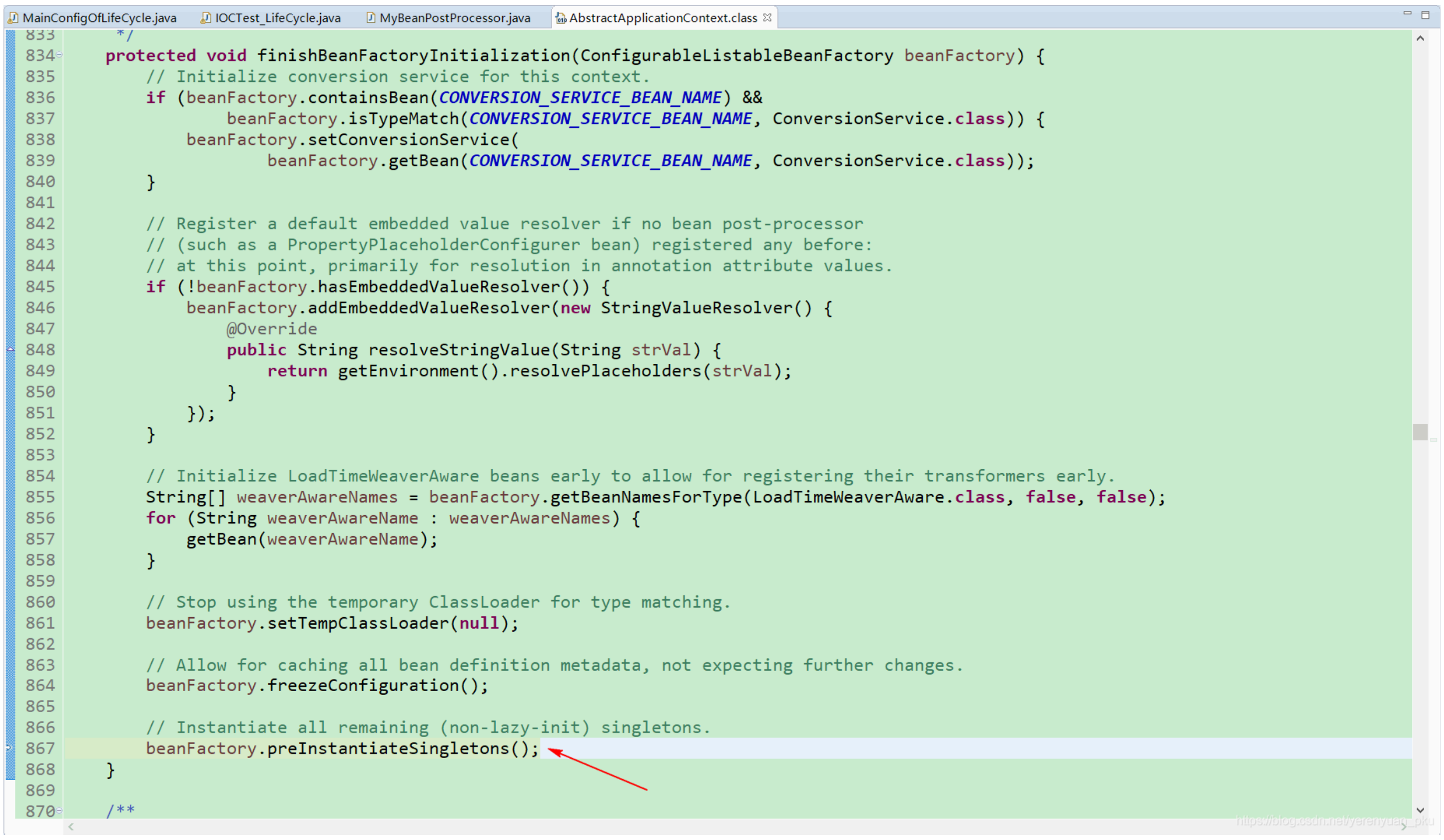
```
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
DefaultListableBeanFactory(AbstractBeanFactory).getBean(String) line: 197
DefaultListableBeanFactory.preInstantiateSingletons() line: 761
AnnotationConfigApplicationContext(AbstractApplicationContext).finishBeanFactoryInitialization(ConfigurableListableBeanFactory) line: 867
AnnotationConfigApplicationContext(AbstractApplicationContext).refresh() line: 543
AnnotationConfigApplicationContext.<init>(Class<?>...) line: 84
IOCTest_LifeCycle.test01() line: 32
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java
        }

        // Initialize LoadTimeWeaverAware beans early to allow for registering their transformers early.
        String[] weaverAwareNames = beanFactory.getBeanNamesForType(LoadTimeWeaverAware.class, false, false);
        for (String weaverAwareName : weaverAwareNames) {
            getBean(weaverAwareName);
        }

        // Stop using the temporary ClassLoader for type matching.
        beanFactory.setTempClassLoader(null);

        // Allow for caching all bean definition metadata, not expecting further changes.
        beanFactory.freezeConfiguration();

        // Instantiate all remaining (non-lazy-init) singletons.
        beanFactory.preInstantiateSingletons();
    }

    /**
     * Finish the refresh of this context, invoking the LifecycleProcessor's
```

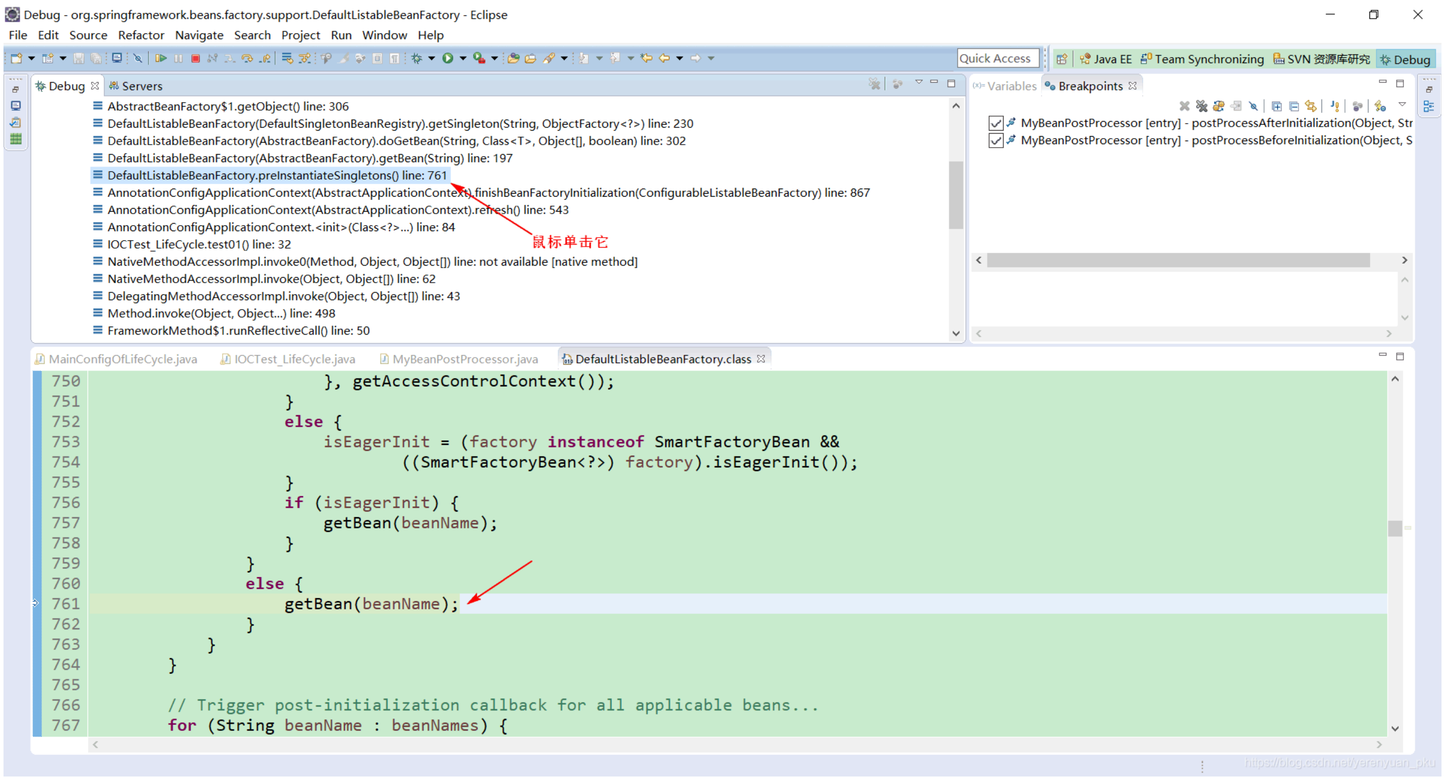
这行代码的作用同样是初始化所有的（非懒加载的）单实例bean。

AbstractApplicationContext类的finishBeanFactoryInitialization()方法同样有点长，我怕同学们看不清，所以又截了一张图，如下所示。



```java
     */
    protected void finishBeanFactoryInitialization(ConfigurableListableBeanFactory beanFactory) {
        // Initialize conversion service for this context.
        if (beanFactory.containsBean(CONVERSION_SERVICE_BEAN_NAME) &&
                beanFactory.isTypeMatch(CONVERSION_SERVICE_BEAN_NAME, ConversionService.class)) {
            beanFactory.setConversionService(
                    beanFactory.getBean(CONVERSION_SERVICE_BEAN_NAME, ConversionService.class));
        }

        // Register a default embedded value resolver if no bean post-processor
        // (such as a PropertyPlaceholderConfigurer bean) registered any before:
        // at this point, primarily for resolution in annotation attribute values.
        if (!beanFactory.hasEmbeddedValueResolver()) {
            beanFactory.addEmbeddedValueResolver(new StringValueResolver() {
                @Override
                public String resolveStringValue(String strVal) {
                    return getEnvironment().resolvePlaceholders(strVal);
                }
            });
        }

        // Initialize LoadTimeWeaverAware beans early to allow for registering their transformers early.
        String[] weaverAwareNames = beanFactory.getBeanNamesForType(LoadTimeWeaverAware.class, false, false);
        for (String weaverAwareName : weaverAwareNames) {
            getBean(weaverAwareName);
        }

        // Stop using the temporary ClassLoader for type matching.
        beanFactory.setTempClassLoader(null);

        // Allow for caching all bean definition metadata, not expecting further changes.
        beanFactory.freezeConfiguration();

        // Instantiate all remaining (non-lazy-init) singletons.
        beanFactory.preInstantiateSingletons();
    }

    /**
```

第五步，我们继续跟进方法调用栈，如下所示，可以看到，方法的执行定位到DefaultListableBeanFactory类的preInstantiateSingletons()方法的最后一个else分支调用的getBean()方法上。

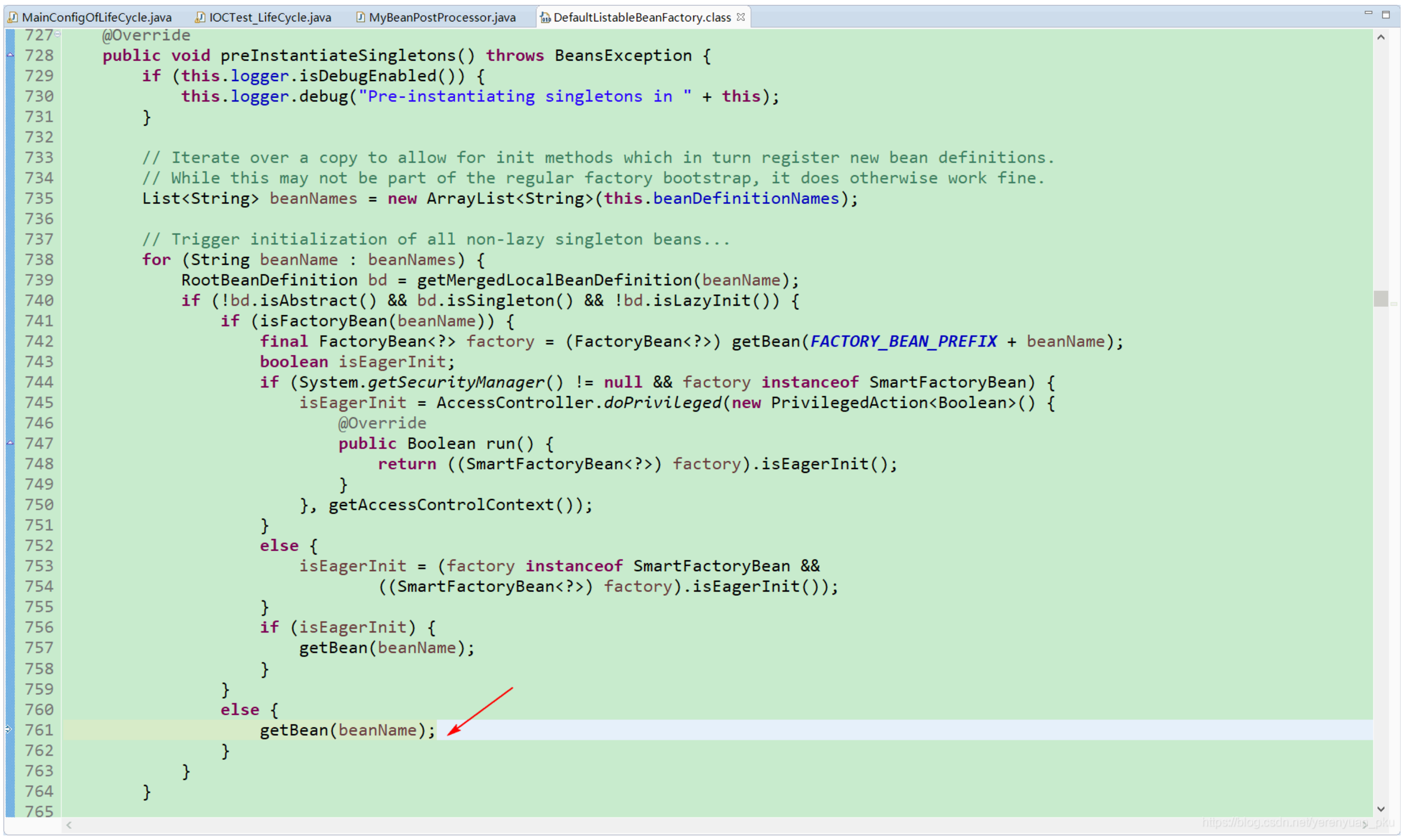

Debug - org.springframework.beans.factory.support.DefaultListableBeanFactory - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Team Synchronizing
SVN 资源库研究
Debug
Debug
Servers
Variables
Breakpoints
鼠标单击它
MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
DefaultListableBeanFactory.class

```
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
DefaultListableBeanFactory(AbstractBeanFactory).getBean(String) line: 197
DefaultListableBeanFactory.preInstantiateSingletons() line: 761
AnnotationConfigApplicationContext(AbstractApplicationContext).finishBeanFactoryInitialization(ConfigurableListableBeanFactory) line: 867
AnnotationConfigApplicationContext(AbstractApplicationContext).refresh() line: 543
AnnotationConfigApplicationContext.<init>(Class<?>...) line: 84
IOCTest_LifeCycle.test01() line: 32
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
Method.invoke(Object, Object...) line: 498
FrameworkMethod$1.runReflectiveCall() line: 50
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

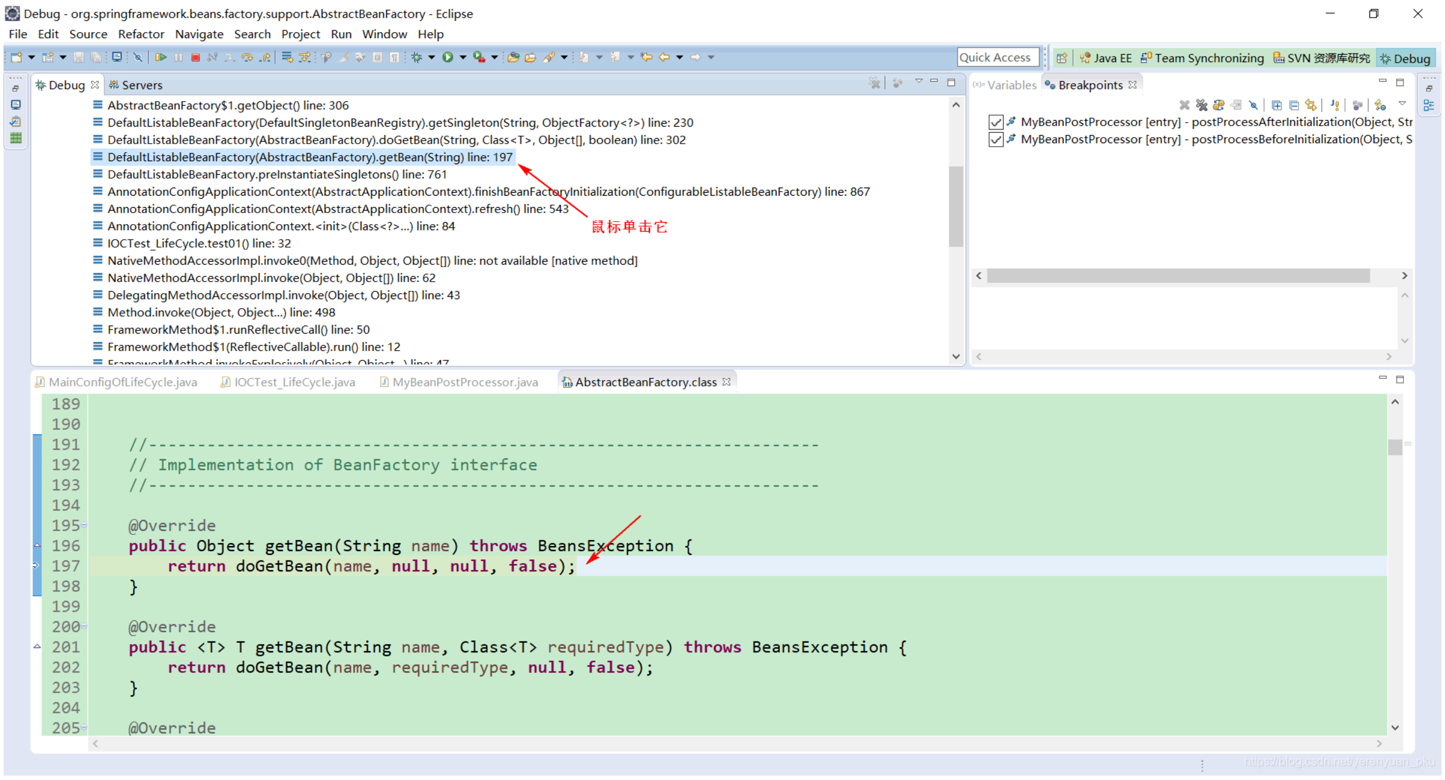
```java
                    }, getAccessControlContext());
                }
                else {
                    isEagerInit = (factory instanceof SmartFactoryBean &&
                            ((SmartFactoryBean<?>) factory).isEagerInit());
                }
                if (isEagerInit) {
                    getBean(beanName);
                }
            }
            else {
                getBean(beanName);
            }
        }
    }

    // Trigger post-initialization callback for all applicable beans...
    for (String beanName : beanNames) {
```

DefaultListableBeanFactory类的preInstantiateSingletons()方法同样是有点长，我怕同学们看不清，所以又截了一张图，如下所示。



```java
    @Override
    public void preInstantiateSingletons() throws BeansException {
        if (this.logger.isDebugEnabled()) {
            this.logger.debug("Pre-instantiating singletons in " + this);
        }

        // Iterate over a copy to allow for init methods which in turn register new bean definitions.
        // While this may not be part of the regular factory bootstrap, it does otherwise work fine.
        List<String> beanNames = new ArrayList<String>(this.beanDefinitionNames);

        // Trigger initialization of all non-lazy singleton beans...
        for (String beanName : beanNames) {
            RootBeanDefinition bd = getMergedLocalBeanDefinition(beanName);
            if (!bd.isAbstract() && bd.isSingleton() && !bd.isLazyInit()) {
                if (isFactoryBean(beanName)) {
                    final FactoryBean<?> factory = (FactoryBean<?>) getBean(FACTORY_BEAN_PREFIX + beanName);
                    boolean isEagerInit;
                    if (System.getSecurityManager() != null && factory instanceof SmartFactoryBean) {
                        isEagerInit = AccessController.doPrivileged(new PrivilegedAction<Boolean>() {
                            @Override
                            public Boolean run() {
                                return ((SmartFactoryBean<?>) factory).isEagerInit();
                            }
                        }, getAccessControlContext());
                    }
                    else {
                        isEagerInit = (factory instanceof SmartFactoryBean &&
                                ((SmartFactoryBean<?>) factory).isEagerInit());
                    }
                    if (isEagerInit) {
                        getBean(beanName);
                    }
                }
                else {
                    getBean(beanName);
                }
            }
        }

```

第六步，继续跟进方法调用栈，如下所示。


Debug - org.springframework.beans.factory.support.AbstractBeanFactory - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
鼠标单击它

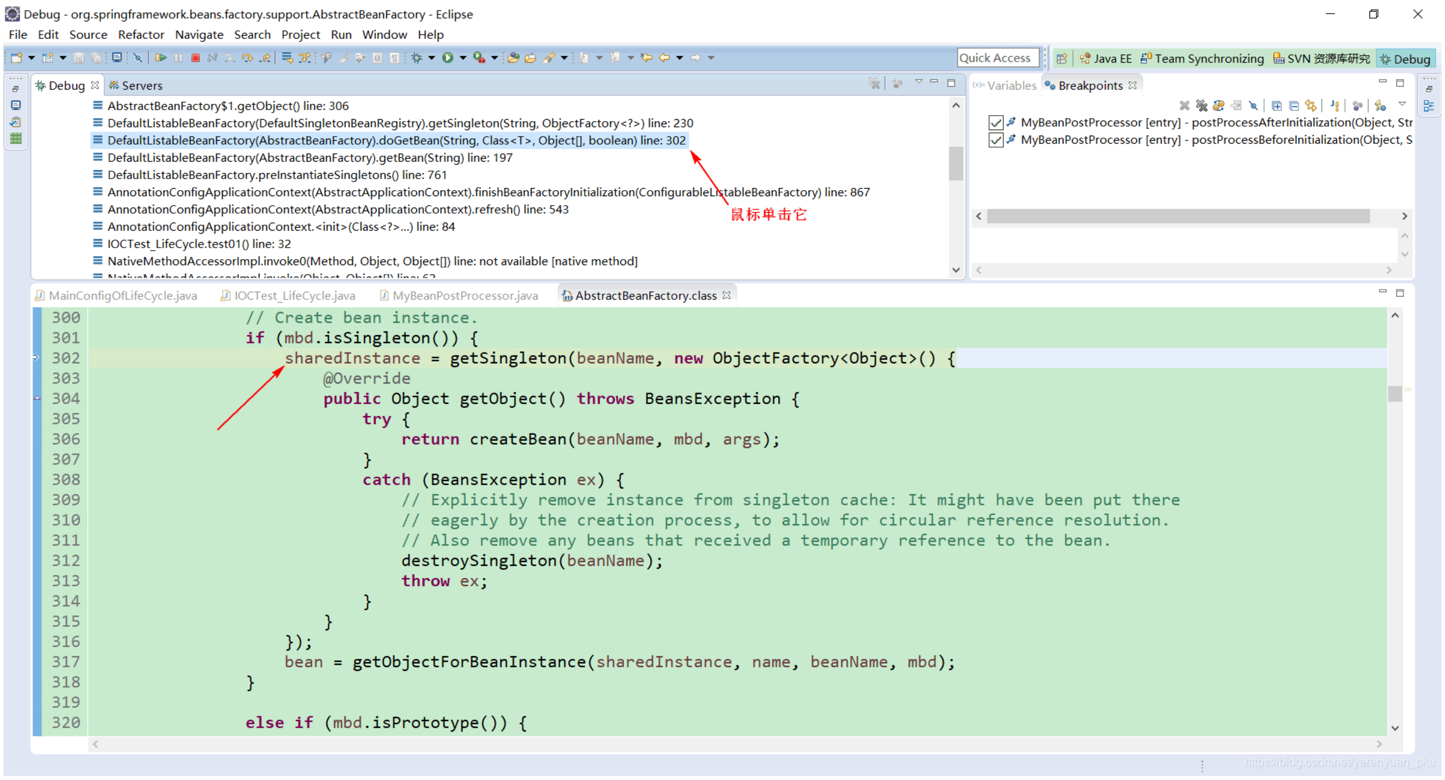
```
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
DefaultListableBeanFactory(AbstractBeanFactory).getBean(String) line: 197
DefaultListableBeanFactory.preInstantiateSingletons() line: 761
AnnotationConfigApplicationContext(AbstractApplicationContext).finishBeanFactoryInitialization(ConfigurableListableBeanFactory) line: 867
AnnotationConfigApplicationContext(AbstractApplicationContext).refresh() line: 543
AnnotationConfigApplicationContext.<init>(Class<?>...) line: 84
IOCTest_LifeCycle.test01() line: 32
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
Method.invoke(Object, Object...) line: 498
FrameworkMethod$1.runReflectiveCall() line: 50
FrameworkMethod$1(ReflectiveCallable).run() line: 12
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java

    //-----------------------------------------------------------------------
    // Implementation of BeanFactory interface
    //-----------------------------------------------------------------------

    @Override
    public Object getBean(String name) throws BeansException {
        return doGetBean(name, null, null, false);
    }

    @Override
    public <T> T getBean(String name, Class<T> requiredType) throws BeansException {
        return doGetBean(name, requiredType, null, false);
    }

    @Override
```

此时方法定位到AbstractBeanFactory类的getBean()方法中了，在getBean()方法中，又调用了doGetBean()方法。

第七步，继续跟进方法调用栈，如下所示，此时，方法的执行定位到AbstractBeanFactory类的doGetBean()方法中的如下那行代码处。

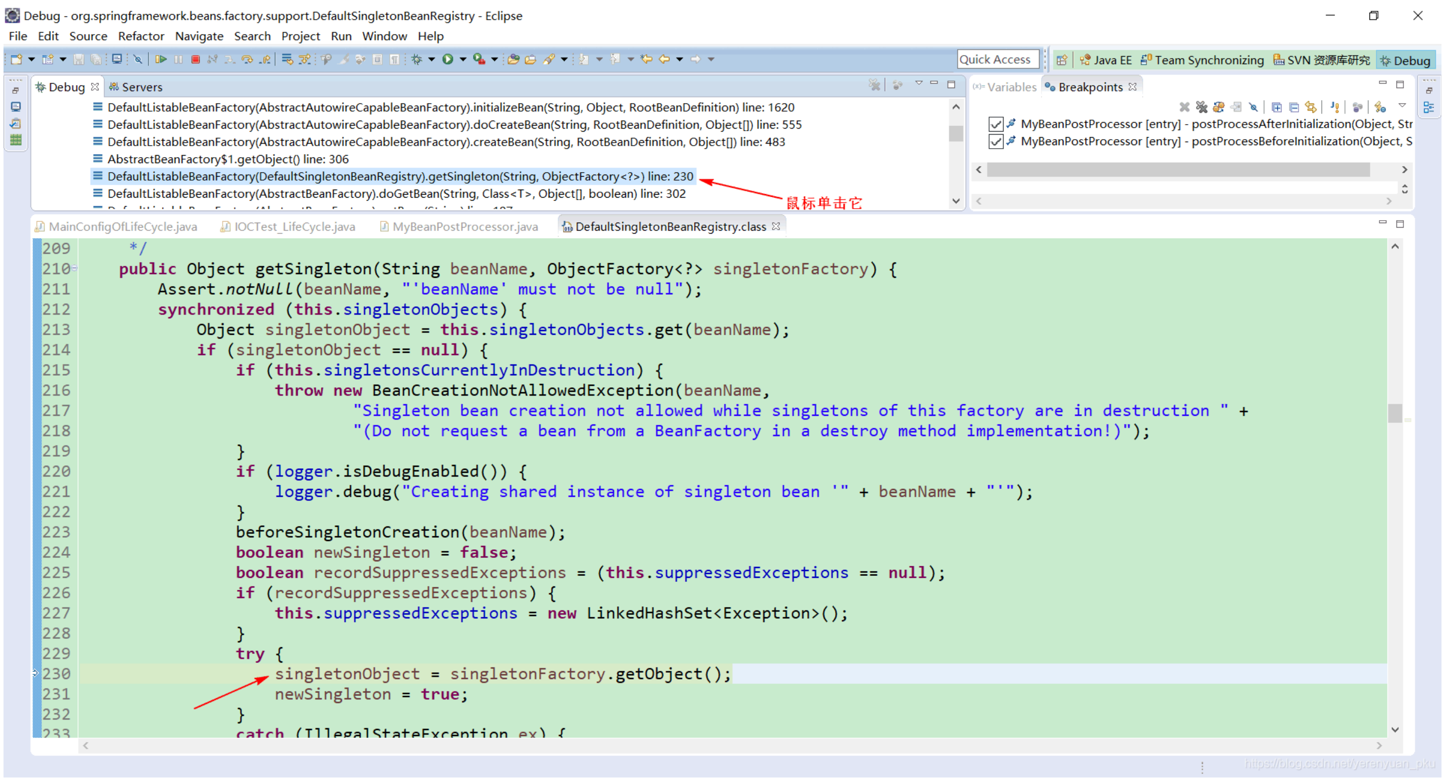

Debug - org.springframework.beans.factory.support.AbstractBeanFactory - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
鼠标单击它

```
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
DefaultListableBeanFactory(AbstractBeanFactory).getBean(String) line: 197
DefaultListableBeanFactory.preInstantiateSingletons() line: 761
AnnotationConfigApplicationContext(AbstractApplicationContext).finishBeanFactoryInitialization(ConfigurableListableBeanFactory) line: 867
AnnotationConfigApplicationContext(AbstractApplicationContext).refresh() line: 543
AnnotationConfigApplicationContext.<init>(Class<?>...) line: 84
IOCTest_LifeCycle.test01() line: 32
NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
```

```java
            // Create bean instance.
            if (mbd.isSingleton()) {
                sharedInstance = getSingleton(beanName, new ObjectFactory<Object>() {
                    @Override
                    public Object getObject() throws BeansException {
                        try {
                            return createBean(beanName, mbd, args);
                        }
                        catch (BeansException ex) {
                            // Explicitly remove instance from singleton cache: It might have been put there
                            // eagerly by the creation process, to allow for circular reference resolution.
                            // Also remove any beans that received a temporary reference to the bean.
                            destroySingleton(beanName);
                            throw ex;
                        }
                    }
                });
                bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);
            }

            else if (mbd.isPrototype()) {
```

可以看到，在Spring内部是通过getSingleton()方法来获取单实例bean的。

第八步，继续跟进方法调用栈，如下所示，此时，方法定位到DefaultSingletonBeanRegistry类的getSingleton()方法中的如下那行代码处。


Debug - org.springframework.beans.factory.support.DefaultSingletonBeanRegistry - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Team Synchronizing
SVN 资源库研究
Debug
Debug
Servers
Variables
Breakpoints
MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
DefaultSingletonBeanRegistry.class
鼠标单击它

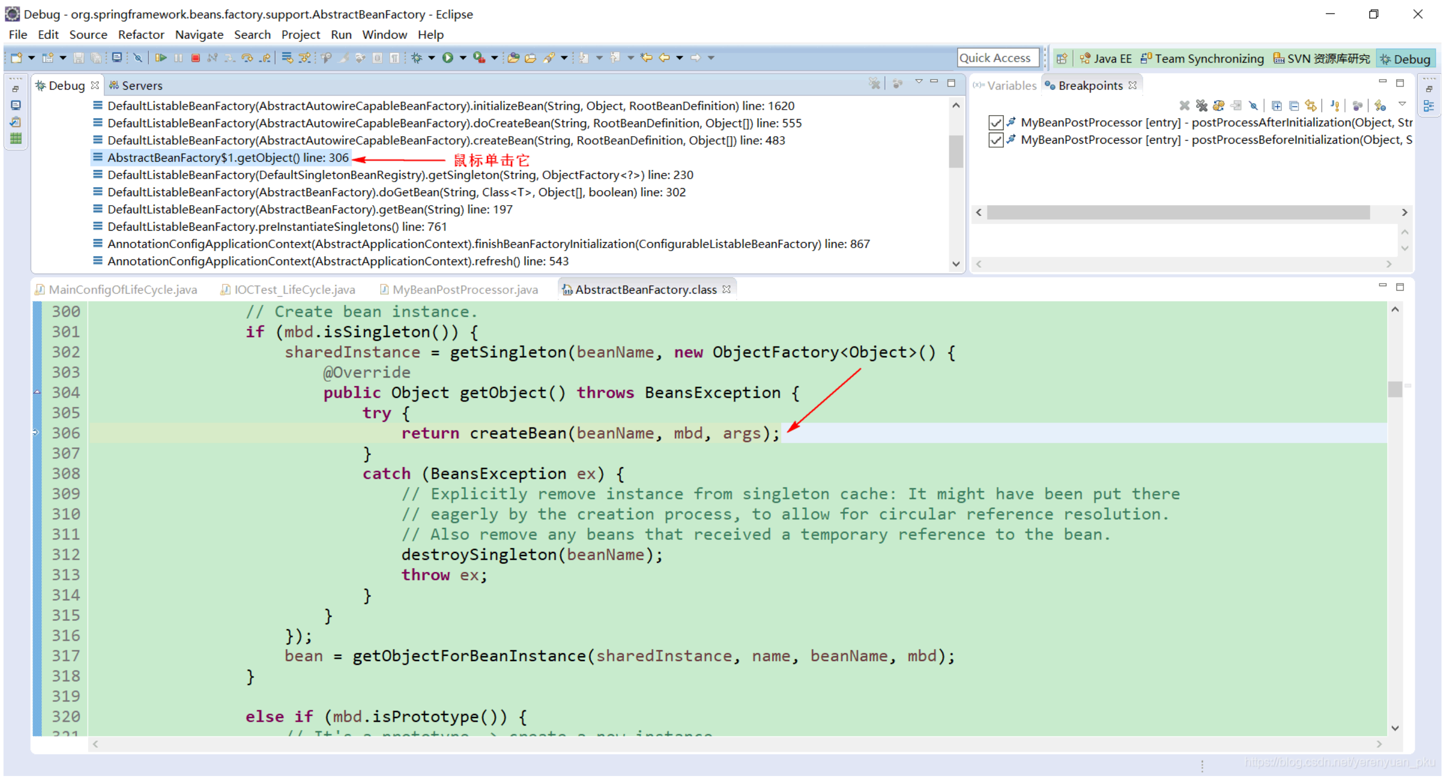
```
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).initializeBean(String, Object, RootBeanDefinition) line: 1620
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).doCreateBean(String, RootBeanDefinition, Object[]) line: 555
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).createBean(String, RootBeanDefinition, Object[]) line: 483
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java
	 */
	public Object getSingleton(String beanName, ObjectFactory<?> singletonFactory) {
		Assert.notNull(beanName, "'beanName' must not be null");
		synchronized (this.singletonObjects) {
			Object singletonObject = this.singletonObjects.get(beanName);
			if (singletonObject == null) {
				if (this.singletonsCurrentlyInDestruction) {
					throw new BeanCreationNotAllowedException(beanName,
							"Singleton bean creation not allowed while singletons of this factory are in destruction " +
							"(Do not request a bean from a BeanFactory in a destroy method implementation!)");
				}
				if (logger.isDebugEnabled()) {
					logger.debug("Creating shared instance of singleton bean '" + beanName + "'");
				}
				beforeSingletonCreation(beanName);
				boolean newSingleton = false;
				boolean recordSuppressedExceptions = (this.suppressedExceptions == null);
				if (recordSuppressedExceptions) {
					this.suppressedExceptions = new LinkedHashSet<Exception>();
				}
				try {
					singletonObject = singletonFactory.getObject();
					newSingleton = true;
				}
				catch (IllegalStateException ex) {
```

可以看到，在getSingleton()方法里面又调用了getObject()方法来获取单实例bean。

第九步，继续跟进方法调用栈，如下所示，此时，方法定位到AbstractBeanFactory类的doGetBean()方法中的如下那行代码处。


Debug - org.springframework.beans.factory.support.AbstractBeanFactory - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Team Synchronizing
SVN 资源库研究
Debug
Debug
Servers
Variables
Breakpoints
MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
AbstractBeanFactory.class
鼠标单击它

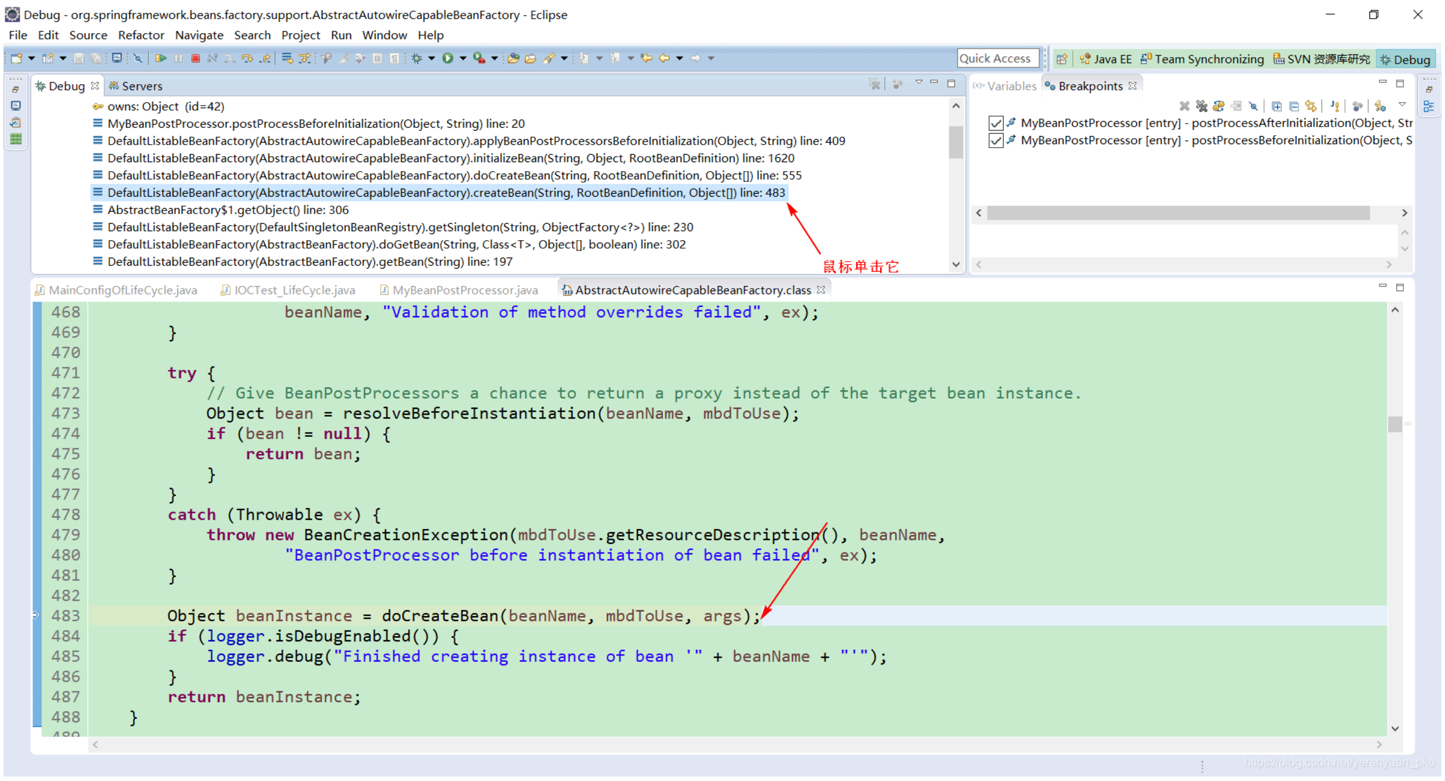
```
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).initializeBean(String, Object, RootBeanDefinition) line: 1620
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).doCreateBean(String, RootBeanDefinition, Object[]) line: 555
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).createBean(String, RootBeanDefinition, Object[]) line: 483
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
DefaultListableBeanFactory(AbstractBeanFactory).getBean(String) line: 197
DefaultListableBeanFactory.preInstantiateSingletons() line: 761
AnnotationConfigApplicationContext(AbstractApplicationContext).finishBeanFactoryInitialization(ConfigurableListableBeanFactory) line: 867
AnnotationConfigApplicationContext(AbstractApplicationContext).refresh() line: 543
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java
			// Create bean instance.
			if (mbd.isSingleton()) {
				sharedInstance = getSingleton(beanName, new ObjectFactory<Object>() {
					@Override
					public Object getObject() throws BeansException {
						try {
							return createBean(beanName, mbd, args);
						}
						catch (BeansException ex) {
							// Explicitly remove instance from singleton cache: It might have been put there
							// eagerly by the creation process, to allow for circular reference resolution.
							// Also remove any beans that received a temporary reference to the bean.
							destroySingleton(beanName);
							throw ex;
						}
					}
				});
				bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);
			}

			else if (mbd.isPrototype()) {
```

也就是说，当第一次获取单实例bean时，由于单实例bean还未创建，那么Spring会调用createBean()方法来创建单实例bean。

第十步，继续跟进方法调用栈，如下所示，可以看到，方法的执行定位到AbstractAutowireCapableBeanFactory类的createBean()方法中的如下那行代码处。

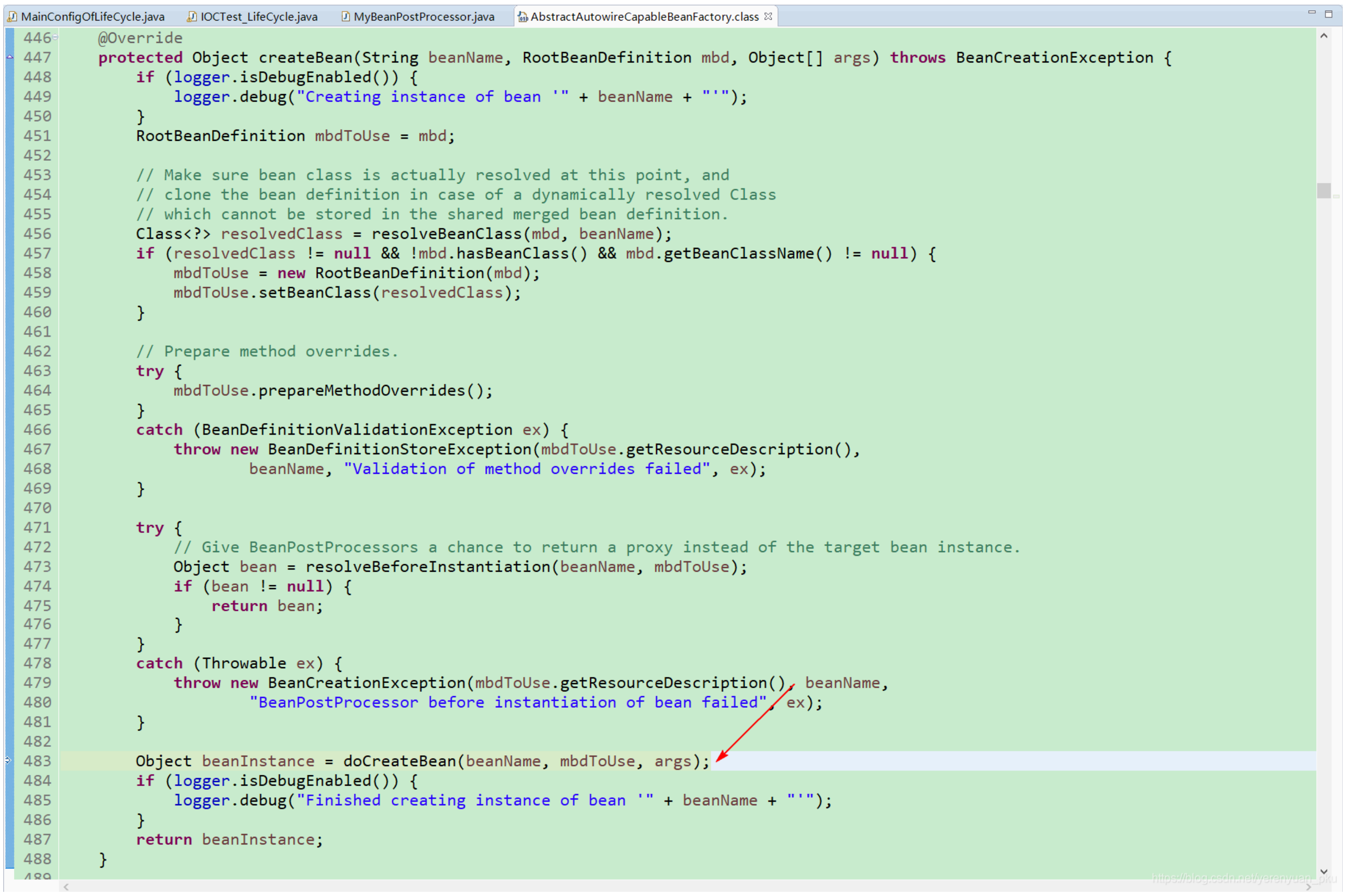

Debug - org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Team Synchronizing
SVN 资源库研究
Debug
Debug
Servers
Variables
Breakpoints
MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
AbstractAutowireCapableBeanFactory.class
鼠标单击它

```
owns: Object (id=42)
MyBeanPostProcessor.postProcessBeforeInitialization(Object, String) line: 20
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).applyBeanPostProcessorsBeforeInitialization(Object, String) line: 409
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).initializeBean(String, Object, RootBeanDefinition) line: 1620
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).doCreateBean(String, RootBeanDefinition, Object[]) line: 555
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).createBean(String, RootBeanDefinition, Object[]) line: 483
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
DefaultListableBeanFactory(AbstractBeanFactory).getBean(String) line: 197
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java
                    beanName, "Validation of method overrides failed", ex);
        }

        try {
            // Give BeanPostProcessors a chance to return a proxy instead of the target bean instance.
            Object bean = resolveBeforeInstantiation(beanName, mbdToUse);
            if (bean != null) {
                return bean;
            }
        }
        catch (Throwable ex) {
            throw new BeanCreationException(mbdToUse.getResourceDescription(), beanName,
                    "BeanPostProcessor before instantiation of bean failed", ex);
        }

        Object beanInstance = doCreateBean(beanName, mbdToUse, args);
        if (logger.isDebugEnabled()) {
            logger.debug("Finished creating instance of bean '" + beanName + "'");
        }
        return beanInstance;
    }
```

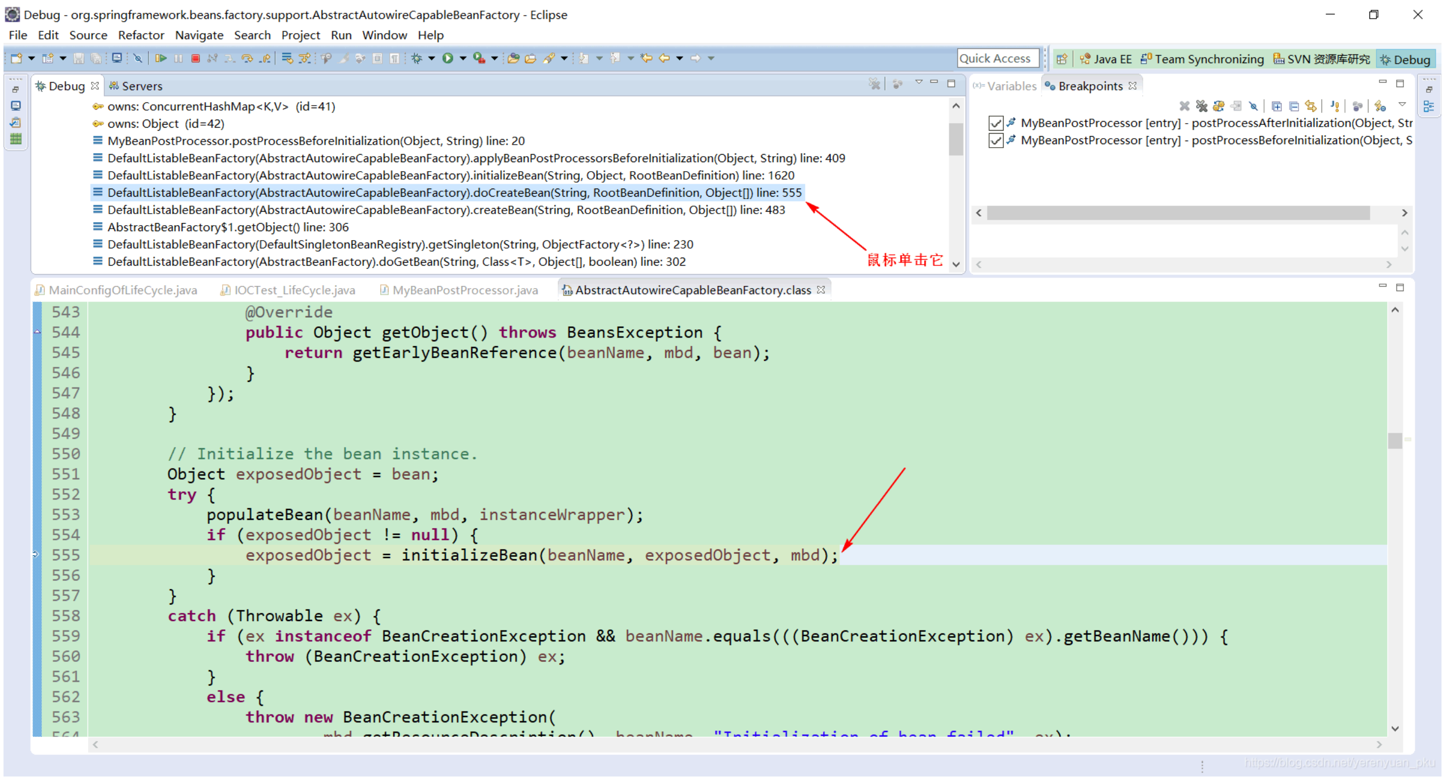
AbstractAutowireCapableBeanFactory类中的createBean()方法同样也是太长，不太方便观看，所以我就又截了一张图，如下所示。


MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
AbstractAutowireCapableBeanFactory.class

```java
    @Override
    protected Object createBean(String beanName, RootBeanDefinition mbd, Object[] args) throws BeanCreationException {
        if (logger.isDebugEnabled()) {
            logger.debug("Creating instance of bean '" + beanName + "'");
        }
        RootBeanDefinition mbdToUse = mbd;

        // Make sure bean class is actually resolved at this point, and
        // clone the bean definition in case of a dynamically resolved Class
        // which cannot be stored in the shared merged bean definition.
        Class<?> resolvedClass = resolveBeanClass(mbd, beanName);
        if (resolvedClass != null && !mbd.hasBeanClass() && mbd.getBeanClassName() != null) {
            mbdToUse = new RootBeanDefinition(mbd);
            mbdToUse.setBeanClass(resolvedClass);
        }

        // Prepare method overrides.
        try {
            mbdToUse.prepareMethodOverrides();
        }
        catch (BeanDefinitionValidationException ex) {
            throw new BeanDefinitionStoreException(mbdToUse.getResourceDescription(),
                    beanName, "Validation of method overrides failed", ex);
        }

        try {
            // Give BeanPostProcessors a chance to return a proxy instead of the target bean instance.
            Object bean = resolveBeforeInstantiation(beanName, mbdToUse);
            if (bean != null) {
                return bean;
            }
        }
        catch (Throwable ex) {
            throw new BeanCreationException(mbdToUse.getResourceDescription(), beanName,
                    "BeanPostProcessor before instantiation of bean failed", ex);
        }

        Object beanInstance = doCreateBean(beanName, mbdToUse, args);
        if (logger.isDebugEnabled()) {
            logger.debug("Finished creating instance of bean '" + beanName + "'");
        }
        return beanInstance;
    }
```

可以看到，Spring中创建单实例bean调用的是doCreateBean()方法。

第十一步，继续跟进方法调用栈，如下所示，此时，方法的执行已经定位到AbstractAutowireCapableBeanFactory类的doCreateBean()方法中的如下那行代码处了。


Debug - org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory - Eclipse

```
owns: ConcurrentHashMap<K,V>  (id=41)
owns: Object  (id=42)
MyBeanPostProcessor.postProcessBeforeInitialization(Object, String) line: 20
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).applyBeanPostProcessorsBeforeInitialization(Object, String) line: 409
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).initializeBean(String, Object, RootBeanDefinition) line: 1620
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).doCreateBean(String, RootBeanDefinition, Object[]) line: 555
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).createBean(String, RootBeanDefinition, Object[]) line: 483
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
```

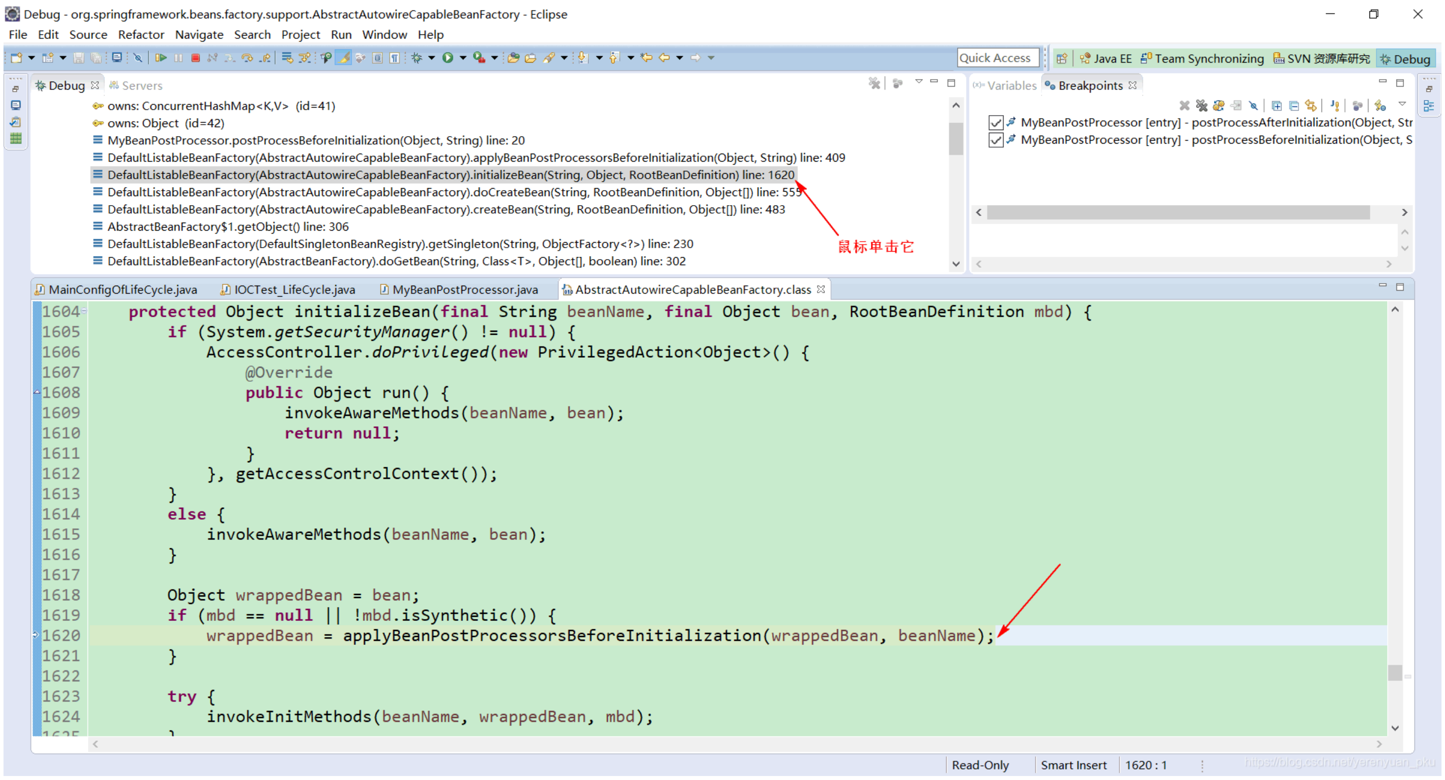
鼠标单击它

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java
                @Override
                public Object getObject() throws BeansException {
                    return getEarlyBeanReference(beanName, mbd, bean);
                }
            });
        }

        // Initialize the bean instance.
        Object exposedObject = bean;
        try {
            populateBean(beanName, mbd, instanceWrapper);
            if (exposedObject != null) {
                exposedObject = initializeBean(beanName, exposedObject, mbd);
            }
        }
        catch (Throwable ex) {
            if (ex instanceof BeanCreationException && beanName.equals(((BeanCreationException) ex).getBeanName())) {
                throw (BeanCreationException) ex;
            }
            else {
                throw new BeanCreationException(
```
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Team Synchronizing
SVN 资源库研究
Debug
Servers
Variables
Breakpoints
MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
AbstractAutowireCapableBeanFactory.class

在initializeBean()方法里面会调用一系列的后置处理器。

第十二步，继续跟进方法调用栈，如下所示，此时，方法的执行定位到AbstractAutowireCapableBeanFactory类的initializeBean()方法中的如下那行代码处。


Debug - org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory - Eclipse

```
owns: ConcurrentHashMap<K,V>  (id=41)
owns: Object  (id=42)
MyBeanPostProcessor.postProcessBeforeInitialization(Object, String) line: 20
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).applyBeanPostProcessorsBeforeInitialization(Object, String) line: 409
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).initializeBean(String, Object, RootBeanDefinition) line: 1620
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).doCreateBean(String, RootBeanDefinition, Object[]) line: 555
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).createBean(String, RootBeanDefinition, Object[]) line: 483
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
```

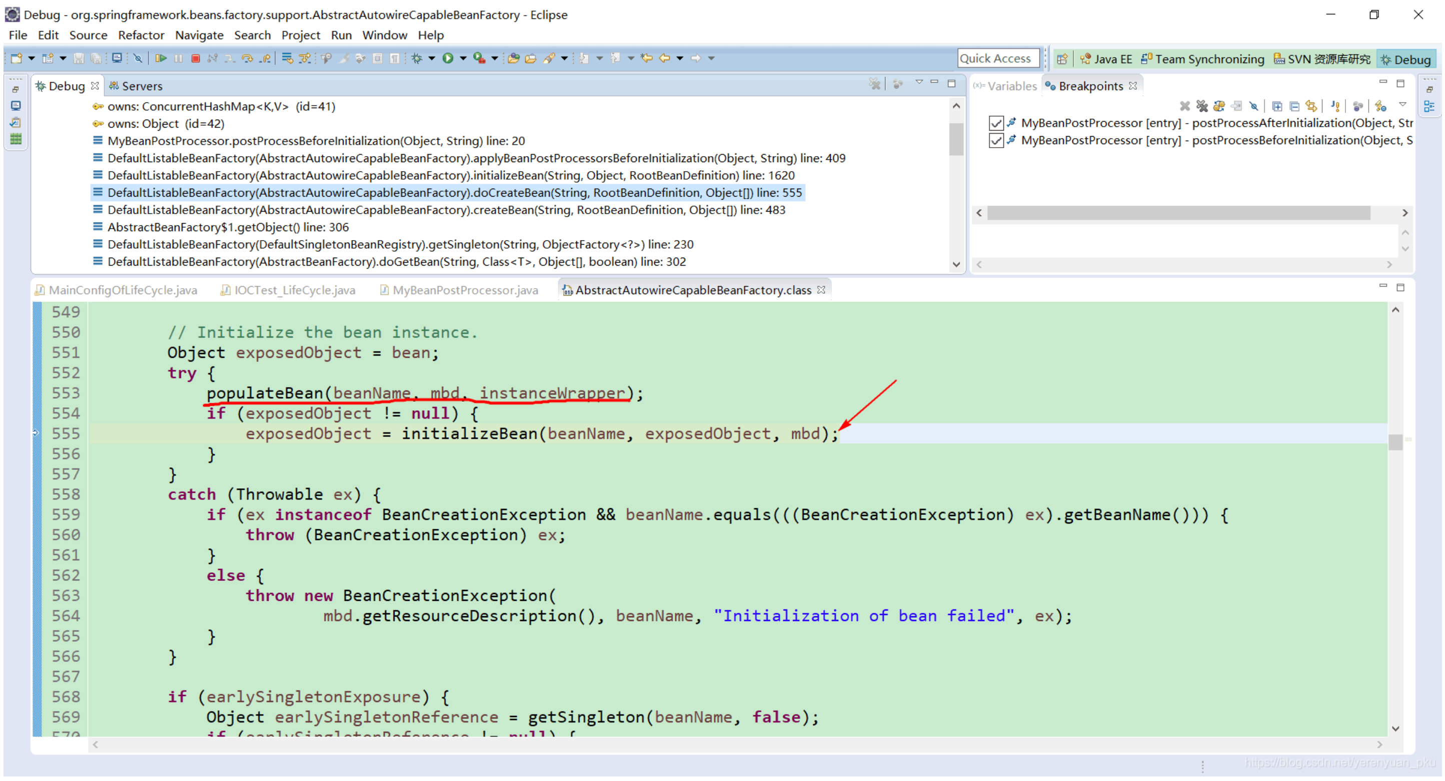
鼠标单击它

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

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

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

        try {
            invokeInitMethods(beanName, wrappedBean, mbd);
```

小伙伴们需要重点留意一下这个applyBeanPostProcessorsBeforeInitialization()方法。

回过头来我们再来看看AbstractAutowireCapableBeanFactory类的doCreateBean()方法中的如下这行代码。


Debug - org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory - Eclipse

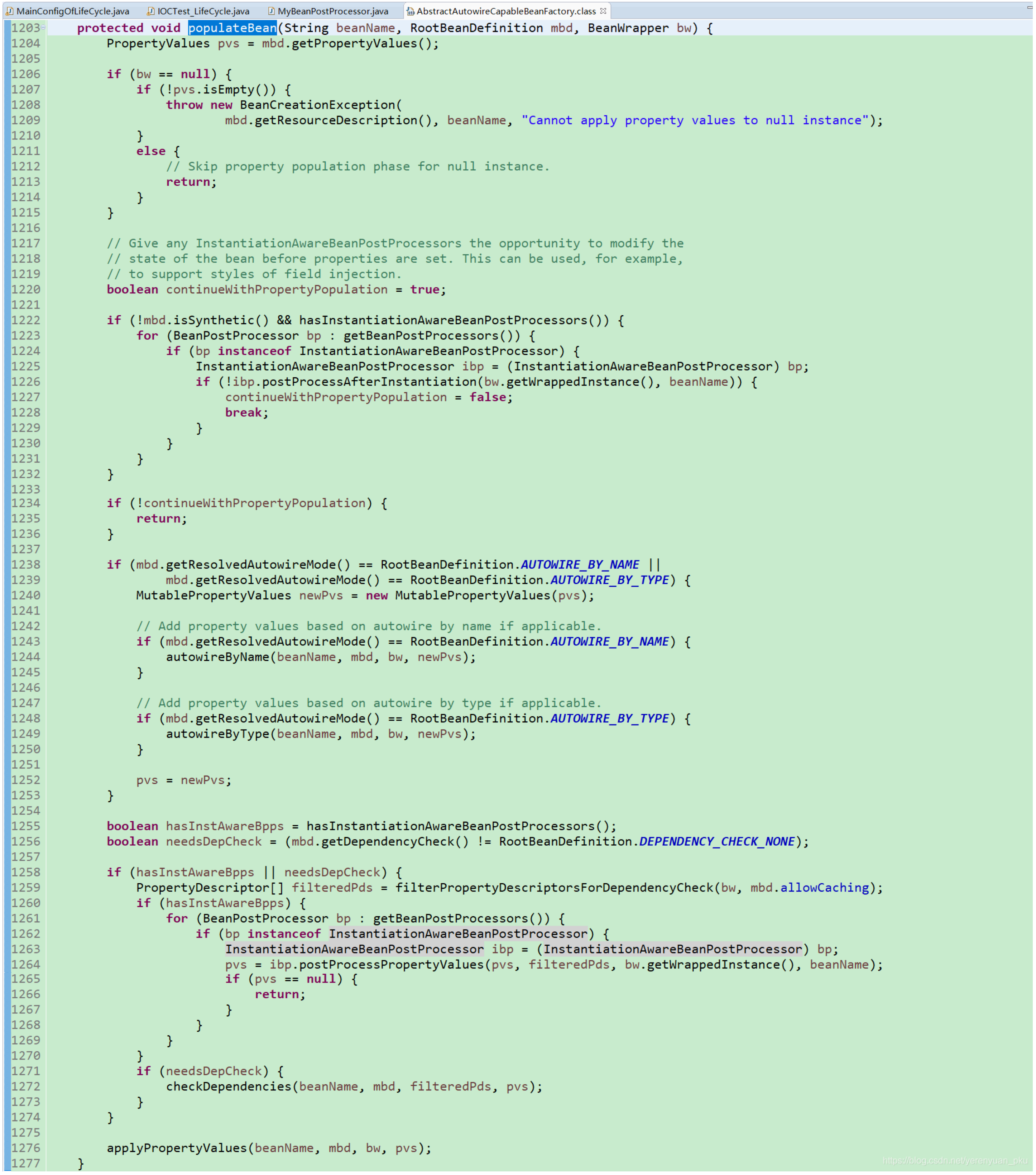
```
owns: ConcurrentHashMap<K,V>  (id=41)
owns: Object  (id=42)
MyBeanPostProcessor.postProcessBeforeInitialization(Object, String) line: 20
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).applyBeanPostProcessorsBeforeInitialization(Object, String) line: 409
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).initializeBean(String, Object, RootBeanDefinition) line: 1620
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).doCreateBean(String, RootBeanDefinition, Object[]) line: 555
DefaultListableBeanFactory(AbstractAutowireCapableBeanFactory).createBean(String, RootBeanDefinition, Object[]) line: 483
AbstractBeanFactory$1.getObject() line: 306
DefaultListableBeanFactory(DefaultSingletonBeanRegistry).getSingleton(String, ObjectFactory<?>) line: 230
DefaultListableBeanFactory(AbstractBeanFactory).doGetBean(String, Class<T>, Object[], boolean) line: 302
```

```
MyBeanPostProcessor [entry] - postProcessAfterInitialization(Object, Str
MyBeanPostProcessor [entry] - postProcessBeforeInitialization(Object, S
```

```java

        // Initialize the bean instance.
        Object exposedObject = bean;
        try {
            populateBean(beanName, mbd, instanceWrapper);
            if (exposedObject != null) {
                exposedObject = initializeBean(beanName, exposedObject, mbd);
            }
        }
        catch (Throwable ex) {
            if (ex instanceof BeanCreationException && beanName.equals(((BeanCreationException) ex).getBeanName())) {
                throw (BeanCreationException) ex;
            }
            else {
                throw new BeanCreationException(
                        mbd.getResourceDescription(), beanName, "Initialization of bean failed", ex);
            }
        }

        if (earlySingletonExposure) {
            Object earlySingletonReference = getSingleton(beanName, false);
```

没错，在以上initializeBean()方法中调用了后置处理器的逻辑，这我上面已经说到了。小伙伴们需要特别注意一下，在AbstractAutowireCapableBeanFactory类的doCreateBean()方法中，调用initializeBean()方法之前，还调用了一个populateBean()方法，我也在上图中标注出来了。

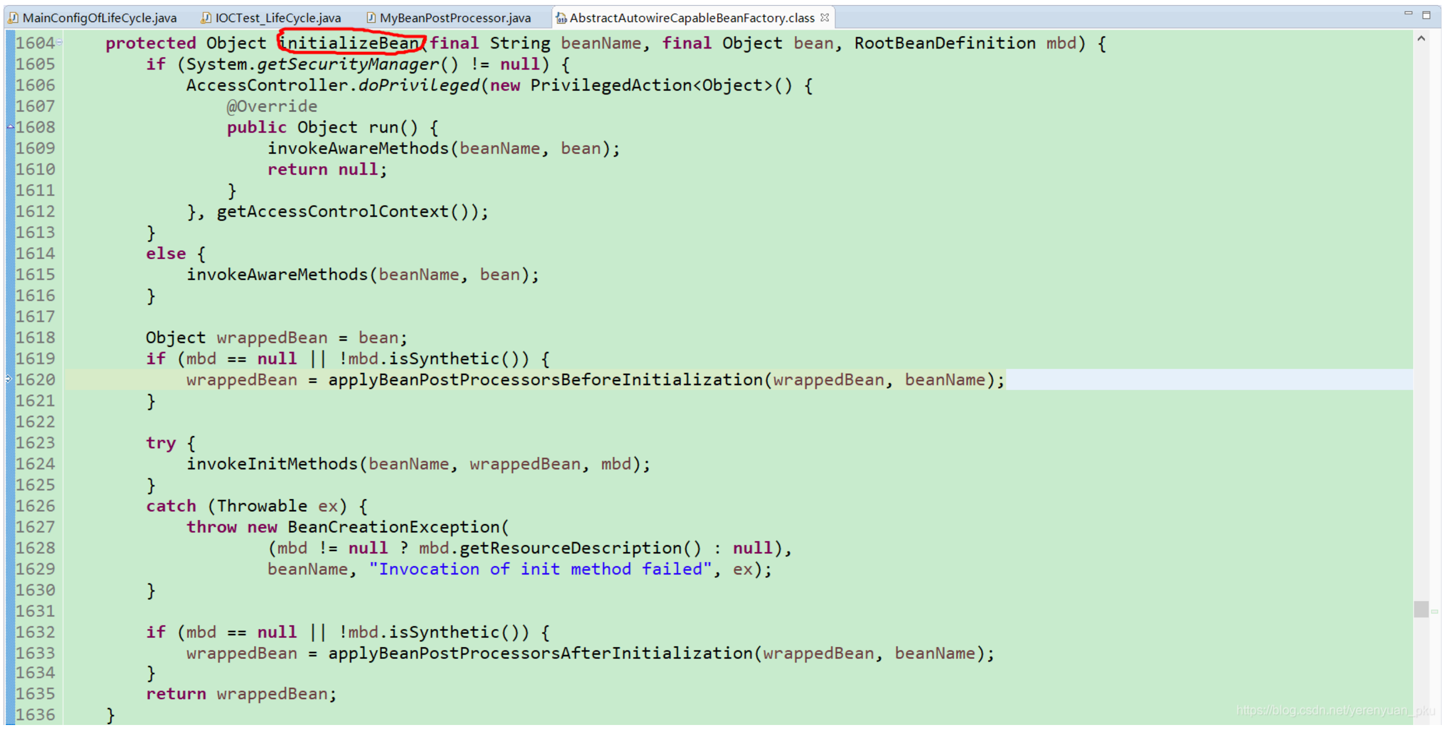
我们点进去这个populateBean()方法中，看下这个方法到底执行了哪些逻辑，如下所示。


MainConfigOfLifeCycle.java
IOCTest_LifeCycle.java
MyBeanPostProcessor.java
AbstractAutowireCapableBeanFactory.class

```
    protected void populateBean(String beanName, RootBeanDefinition mbd, BeanWrapper bw) {
        PropertyValues pvs = mbd.getPropertyValues();

        if (bw == null) {
            if (!pvs.isEmpty()) {
                throw new BeanCreationException(
                        mbd.getResourceDescription(), beanName, "Cannot apply property values to null instance");
            }
            else {
                // Skip property population phase for null instance.
                return;
            }
        }

        // Give any InstantiationAwareBeanPostProcessors the opportunity to modify the
        // state of the bean before properties are set. This can be used, for example,
        // to support styles of field injection.
        boolean continueWithPropertyPopulation = true;

        if (!mbd.isSynthetic() && hasInstantiationAwareBeanPostProcessors()) {
            for (BeanPostProcessor bp : getBeanPostProcessors()) {
                if (bp instanceof InstantiationAwareBeanPostProcessor) {
                    InstantiationAwareBeanPostProcessor ibp = (InstantiationAwareBeanPostProcessor) bp;
                    if (!ibp.postProcessAfterInstantiation(bw.getWrappedInstance(), beanName)) {
                        continueWithPropertyPopulation = false;
                        break;
                    }
                }
            }
        }

        if (!continueWithPropertyPopulation) {
            return;
        }

        if (mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE_BY_NAME ||
                mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE_BY_TYPE) {
            MutablePropertyValues newPvs = new MutablePropertyValues(pvs);

            // Add property values based on autowire by name if applicable.
            if (mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE_BY_NAME) {
                autowireByName(beanName, mbd, bw, newPvs);
            }

            // Add property values based on autowire by type if applicable.
            if (mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE_BY_TYPE) {
                autowireByType(beanName, mbd, bw, newPvs);
            }

            pvs = newPvs;
        }

        boolean hasInstAwareBpps = hasInstantiationAwareBeanPostProcessors();
        boolean needsDepCheck = (mbd.getDependencyCheck() != RootBeanDefinition.DEPENDENCY_CHECK_NONE);

        if (hasInstAwareBpps || needsDepCheck) {
            PropertyDescriptor[] filteredPds = filterPropertyDescriptorsForDependencyCheck(bw, mbd.allowCaching);
            if (hasInstAwareBpps) {
                for (BeanPostProcessor bp : getBeanPostProcessors()) {
                    if (bp instanceof InstantiationAwareBeanPostProcessor) {
                        InstantiationAwareBeanPostProcessor ibp = (InstantiationAwareBeanPostProcessor) bp;
                        pvs = ibp.postProcessPropertyValues(pvs, filteredPds, bw.getWrappedInstance(), beanName);
                        if (pvs == null) {
                            return;
                        }
                    }
                }
            }
            if (needsDepCheck) {
                checkDependencies(beanName, mbd, filteredPds, pvs);
            }
        }

        applyPropertyValues(beanName, mbd, bw, pvs);
    }
```

populateBean()方法同样是AbstractAutowireCapableBeanFactory类中的方法，它里面的代码比较多，但是逻辑非常简单，populateBean()方法做的工作就是为bean的属性赋值。也就是说，在Spring中会先调用populateBean()方法为bean的属性赋好值，然后再调用initializeBean()方法。

接下来，我们好好分析下initializeBean()方法，为了方便，我将Spring中AbstractAutowireCapableBeanFactory类的initializeBean()方法的代码特意提取出来了，如下所示。


MainConfigOfLifeCycle.java IOCTest_LifeCycle.java MyBeanPostProcessor.java AbstractAutowireCapableBeanFactory.class

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

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

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

        if (mbd == null || !mbd.isSynthetic()) {
            wrappedBean = applyBeanPostProcessorsAfterInitialization(wrappedBean, beanName);
        }
        return wrappedBean;
    }
```

在initializeBean()方法中，调用了invokeInitMethods()方法，代码行如下所示。

```java
invokeInitMethods(beanName, wrappedBean, mbd);
```
AI写代码java运行

invokeInitMethods()方法的作用就是执行 初始化方法 ，这些初始化方法包括我们之前讲的：**在XML配置文件的标签中使用init-method属性指定的初始化方法；在@Bean注解中使用initMehod属性指定的方法；使用@PostConstruct注解标注的方法；实现InitializingBean接口的方法等。**

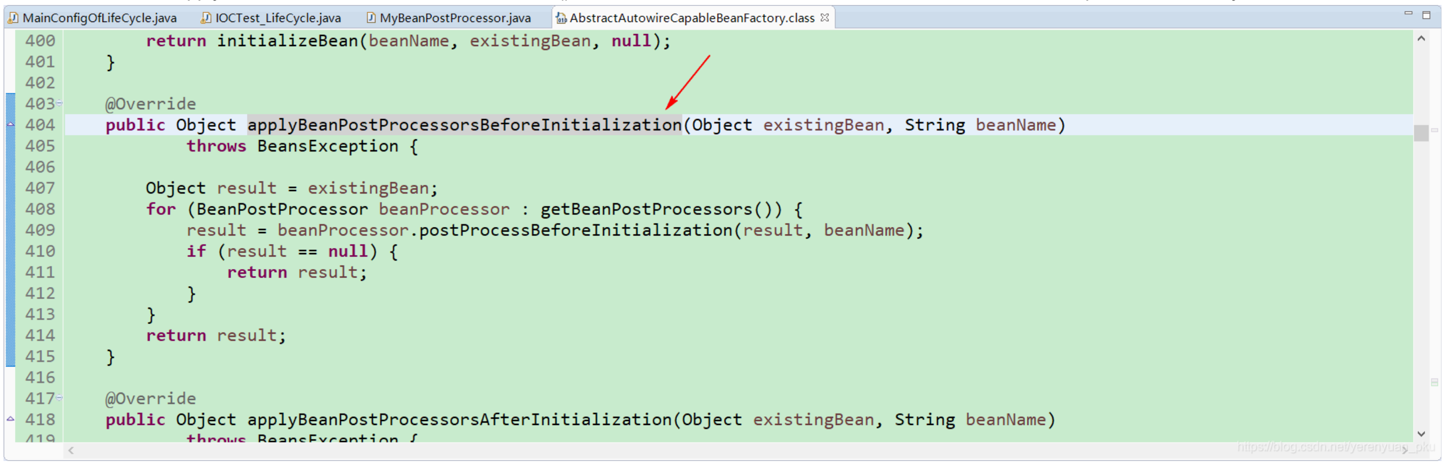
在调用invokeInitMethods()方法之前，Spring调用了applyBeanPostProcessorsBeforeInitialization()这个方法，代码行如下所示。

```java
wrappedBean = applyBeanPostProcessorsBeforeInitialization(wrappedBean, beanName);
```
AI写代码java运行

在调用invokeInitMethods()方法之后，Spring又调用了applyBeanPostProcessorsAfterInitialization()这个方法，如下所示。

```java
wrappedBean = applyBeanPostProcessorsAfterInitialization(wrappedBean, beanName);
```
AI写代码java运行

这里，我们先来看看applyBeanPostProcessorsBeforeInitialization()方法中具体执行了哪些逻辑，该方法位于AbstractAutowireCapableBeanFactory类中，源码如下所示。


MainConfigOfLifeCycle.java IOCTest_LifeCycle.java MyBeanPostProcessor.java AbstractAutowireCapableBeanFactory.class

```java
        return initializeBean(beanName, existingBean, null);
    }

    @Override
    public Object applyBeanPostProcessorsBeforeInitialization(Object existingBean, String beanName)
            throws BeansException {

        Object result = existingBean;
        for (BeanPostProcessor beanProcessor : getBeanPostProcessors()) {
            result = beanProcessor.postProcessBeforeInitialization(result, beanName);
            if (result == null) {
                return result;
            }
        }
        return result;
    }

    @Override
    public Object applyBeanPostProcessorsAfterInitialization(Object existingBean, String beanName)
            throws BeansException {
```

可以看到，在applyBeanPostProcessorsBeforeInitialization()方法中，会遍历所有BeanPostProcessor对象，然后依次执行所有BeanPostProcessor对象的postProcessBeforeInitialization()方法，一旦BeanPostProcessor对象的postProcessBeforeInitialization()方法返回null以后，则后面的BeanPostProcessor对象便不再执行了，而是直接退出for循环。这些都是我们看源码看到的。

看Spring源码，我们还看到了一个细节，**在Spring中调用initializeBean()方法之前，还调用了populateBean()方法来为bean的属性赋值，** 这在上面我也已经说过了。

经过上面的一系列的跟踪源码分析，我们可以将关键代码的调用过程使用如下伪代码表述出来。

```java
populateBean(beanName, mbd, instanceWrapper); // 给bean进行属性赋值
initializeBean(beanName, exposedObject, mbd)
{
    applyBeanPostProcessorsBeforeInitialization(wrappedBean, beanName);
    invokeInitMethods(beanName, wrappedBean, mbd); // 执行自定义初始化
    applyBeanPostProcessorsAfterInitialization(wrappedBean, beanName);
}
```

AI写代码java运行

也就是说，在Spring中，调用initializeBean()方法之前，调用了populateBean()方法为bean的属性赋值，为bean的属性赋好值之后，再调用initializeBean()方法进行初始化。

在initializeBean()中，调用自定义的初始化方法（即invokeInitMethods()）之前，调用了applyBeanPostProcessorsBeforeInitialization()方法，而在调用自定义的初始化方法之后，又调用了applyBeanPostProcessorsAfterInitialization()方法。至此，整个bean的初始化过程就这样结束了。