Spring源码-Bean的实例化

接下来我们看看Bean的实例化处理

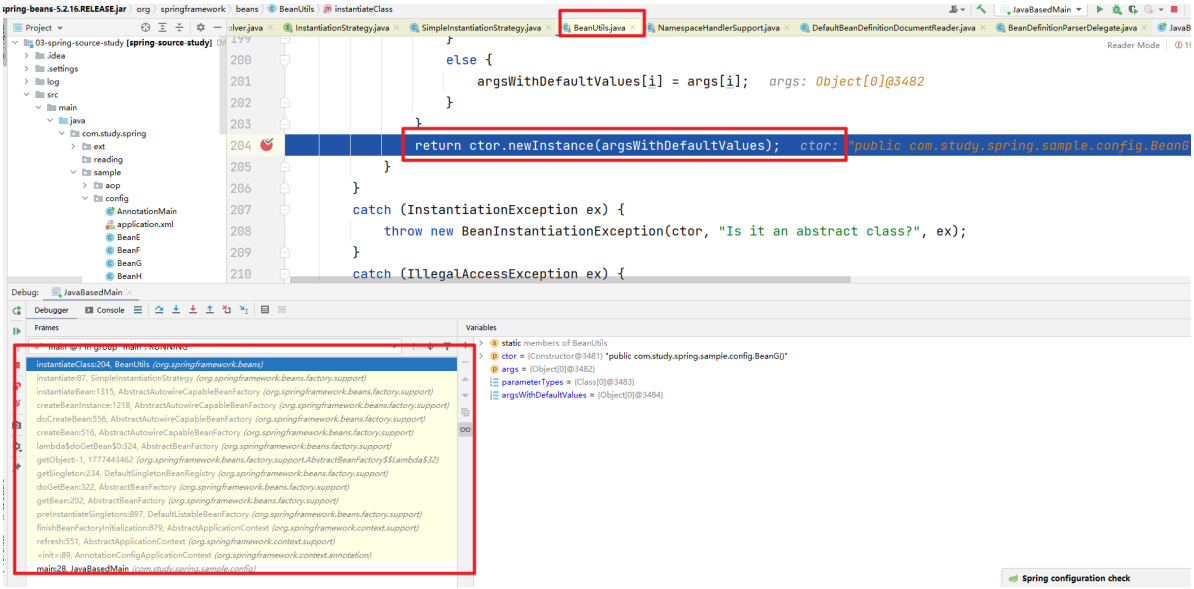
一、BeanDefinition

首先我们来看看BeanDefinition的存放位置。因为Bean对象的实例化肯定是BeanFactory基于对应的BeanDefinition的定义来实现的，所以在这个过程中BeanDefinition是非常重要的，前面的课程讲解已经完成了BeanDefinition的定义。同时根据前面refresh方法的讲解我们知道了BeanFactory的具体实现是 `DefaultListableBeanFactory`.所以BeanDefinition的相关信息是存储在 `DefaultListableBeanFactory` 的相关属性中的。

```
/** Map of bean definition objects, keyed by bean name. */
private final Map<String, BeanDefinition> beanDefinitionMap = new
ConcurrentHashMap<>(256);
```

二、Bean实例的创建过程

然后就是Bean实例的创建过程。这块儿我们可以通过Debug的形式非常直观的看到。



按照这种步骤一个个去分析就OK了。

三、单例对象

在创建单例对象的时候是如何保存单例的特性的？这块我们需要注意下面的代码

```
// create bean instance.
if (mbd.isSingleton()) {  mbd: "Root bean: class [com.study.spring.sample.config.Bean6
    sharedInstance = getSingleton(beanName, () -> {  sharedInstance: null   beanName:
        try {
            return createBean(beanName, mbd, args);
        }
        catch (BeansException ex) {
            // Explicitly remove instance from singleton cache: It might have been put
            // eagerly by the creation process, to allow for circular reference resolu
            // Also remove any beans that received a temporary reference to the bean.
            destroySingleton(beanName);
            throw ex;
        }
    });
    bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);
}
```

这块的Lambda表达式

然后进入到getSingleton方法中。

```
public Object getSingleton(String beanName, ObjectFactory<?> singletonFactory) {
    Assert.notNull(beanName, message: "Bean name 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 facto
                        "(Do not request a bean from a BeanFactory in a destroy method impl
            }
            if (logger.isDebugEnabled()) {
                logger.debug( o: "Creating shared instance of singleton bean '" + beanName +
            }
            beforeSingletonCreation(beanName);
            boolean newSingleton = false;
            boolean recordSuppressedExceptions = (this.suppressedExceptions == null);
            if (recordSuppressedExceptions) {
                this.suppressedExceptions = new LinkedHashSet<>();
            }
            try {
```

保证单例的创建

创建成功的单例对象会被缓存起来。在 addSingleton 方法中

```
            if (recordSuppressedExceptions) {
                for (Exception suppressedException : this.suppressedExceptions) {
                    ex.addRelatedCause(suppressedException);
                }
            }
            throw ex;
        }
        finally {
            if (recordSuppressedExceptions) {
                this.suppressedExceptions = null;
            }
            afterSingletonCreation(beanName);
        }
        if (newSingleton) {
            addSingleton(beanName, singletonObject);
        }
    }
    return singletonObject;
}
}
```

```
singletonObject – the singleton object
protected void addSingleton(String beanName, Object singletonObject) {
    synchronized (this.singletonObjects) {
        this.singletonObjects.put(beanName, singletonObject);
        this.singletonFactories.remove(beanName);
        this.earlySingletonObjects.remove(beanName);
        this.registeredSingletons.add(beanName);
    }
}
```

所以singletonObjects是缓存所有Bean实例的容器

```
factory > support > DefaultSingletonBeanRegistry > singletonObjects
AbstractAutowireCapableBeanFactory.java | AbstractBeanFactory.java | DefaultSingletonBeanRegistry.java | SimpleInstantiationStrategy.java | BeanUtils.java | NamespaceHandlerSupport.java
public class DefaultSingletonBeanRegistry extends SimpleAliasRegistry implements SingletonBeanRegistry {

    Maximum number of suppressed exceptions to preserve.
    private static final int SUPPRESSED_EXCEPTIONS_LIMIT = 100;

                                        缓存所有单例Bean的容器
    Cache of singleton objects: bean name to bean instance.
    private final Map<String, Object> singletonObjects = new ConcurrentHashMap<>( initialCapacity: 256);

    Cache of singleton factories: bean name to ObjectFactory.
    private final Map<String, ObjectFactory<?>> singletonFactories = new HashMap<>( initialCapacity: 16);
```

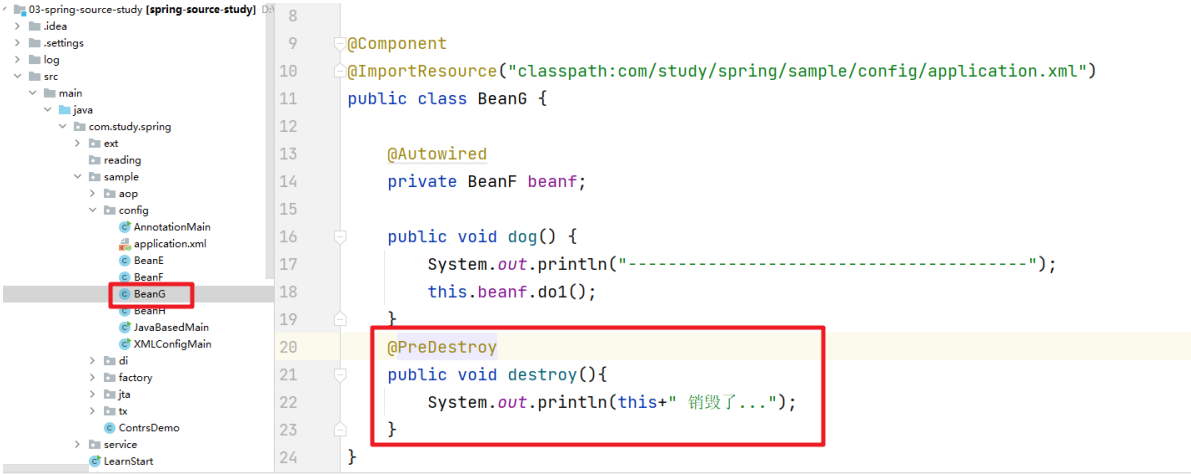
而具体创建单例Bean的逻辑会回调前面的Lambda表达式中的createBean方法

```
wireCapableBeanFactory.java | AbstractBeanFactory.java | DefaultSingletonBeanRegistry.java | SimpleInstantiationStrategy.java | BeanUtils.java | NamespaceHandlerSupport.java
        beforeSingletonCreation(beanName);
        boolean newSingleton = false;
        boolean recordSuppressedExceptions = (this.suppressedExceptions == null);
        if (recordSuppressedExceptions) {
            this.suppressedExceptions = new LinkedHashSet<>();
        }
        try {
            singletonObject = singletonFactory.getObject();
            newSingleton = true;
        }
        catch (IllegalStateException ex) {
            // Has the singleton object implicitly appeared in the meantime ->
            // if yes, proceed with it since the exception indicates that state.
            singletonObject = this.singletonObjects.get(beanName);
            if (singletonObject == null) {
                throw ex;
            }
        }
        catch (BeanCreationException ex) {
            if (recordSuppressedExceptions) {
```

```java
// Create bean instance.
if (mbd.isSingleton()) {  mbd: "Root bean: class [com.study.spring.sample.config.BeanG],
    sharedInstance = getSingleton(beanName, () -> {  sharedInstance: null  beanName: "
        try {
            return createBean(beanName, mbd, args);
        }
        catch (BeansException ex) {
            // Explicitly remove instance from singleton cache: It might have been put t
            // eagerly by the creation process, to allow for circular reference resoluti
            // Also remove any beans that received a temporary reference to the bean.
            destroySingleton(beanName);
            throw ex;
        }
    });
    bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);
}
```

四、单例对象的销毁

然后我们先来看下一个单例Bean对象的销毁过程。定义一个案例




```java
@Component
@ImportResource("classpath:com/study/spring/sample/config/application.xml")
public class BeanG {

    @Autowired
    private BeanF beanf;

    public void dog() {
        System.out.println("----------------------------------------");
        this.beanf.do1();
    }
    @PreDestroy
    public void destroy(){
        System.out.println(this+" 销毁了...");
    }
}
```

然后我们在测试的案例中显示的调用 `close` 方法

```java
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext( ...basePackages: "com.stu

        BeanG bg = context.getBean(BeanG.class);
        bg.dog();
        // 主动调用下面的方法 会触发对应的事件发布，触发对应的事件的监听执行
        context.start();
        context.stop();
        context.close();
    }
}
```

执行的时候可以看到相关的日志执行了。

```
sample
  aop
  config
    AnnotationMain
    application.xml
    BeanE
    BeanF
    BeanG
    BeanH
    JavaBasedMain
    XMLConfigMain
  di
  factory
  jta
  tx
    ContrsDemo
  service
    LearnStart
```

```java
8   public static void main(String[] args) {
9       AnnotationConfigApplicationContext context = new AnnotationConfigApplicationConte
10
11      BeanG bg = context.getBean(BeanG.class);
12      bg.dog();
13      // 主动调用下面的方法 会触发对应的事件发布，触发对应的事件的监听执行
14      context.start();
15      context.stop();
16      context.close();
17  }
18 }
```

```
Run: AnnotationMain
 .AnnotationConfigApplicationContext@4148db48, started on Tue Sep 20 14:56:53 CST 2022]
-----收到应用事件: org.springframework.context.event.ContextStoppedEvent[source=org.springframework.context.annotation
 .AnnotationConfigApplicationContext@4148db48, started on Tue Sep 20 14:56:53 CST 2022]
-----收到应用事件: org.springframework.context.event.ContextClosedEvent[source=org.springframework.context.annotation
 .AnnotationConfigApplicationContext@4148db48, started on Tue Sep 20 14:56:53 CST 2022]
com.study.spring.sample.config.BeanG@6b09fb41 销毁了...
```

进入到close方法中分析，比较核心的有两个位置。在doClose方法中。

```java
try {
    // Publish shutdown event.
    publishEvent(new ContextClosedEvent( source: this));
}
catch (Throwable ex) {
    logger.warn( o: "Exception thrown from ApplicationListener handling ContextClosedEvent", ex);
}

// Stop all Lifecycle beans, to avoid delays during individual destruction.
if (this.lifecycleProcessor != null) {
    try {
        this.lifecycleProcessor.onClose();
    }
    catch (Throwable ex) {
        logger.warn( o: "Exception thrown from LifecycleProcessor on context close", ex);
    }
}

// Destroy all cached singletons in the context's BeanFactory.
destroyBeans();

// Close the state of this context itself.
closeBeanFactory();

// Let subclasses do some final clean-up if they wish...
```

发布事件--》打印日志的原因

从BeanFactory中销毁对应的单例Bean

具体销毁的代码进入destroyBeans()中查看即可。

在doClose方法中有个提示。registerShutdownHook方法

```
Actually performs context closing: publishes a ContextClosedEvent and destroys the singletons in
the bean factory of this application context.
Called by both close() and a JVM shutdown hook, if any.
See Also: ContextClosedEvent, destroyBeans(), close(), registerShutdownHook()
```

```java
protected void doClose() {
    // Check whether an actual close attempt is necessary...
    if (this.active.get() && this.closed.compareAndSet( expect: false, update: true)) {
        if (logger.isDebugEnabled()) {
            logger.debug( o: "Closing " + this);
        }

        LiveBeansView.unregisterApplicationContext(this);

        try {
            // Publish shutdown event.
            publishEvent(new ContextClosedEvent( source: this));
        }
```

```java
    @Override
    public void registerShutdownHook() {
        if (this.shutdownHook == null) {
            // No shutdown hook registered yet.
            this.shutdownHook = new Thread(SHUTDOWN_HOOK_THREAD_NAME) {
                @Override
                public void run() {
                    synchronized (startupShutdownMonitor) {
                        doClose();
                    }
                }
            };
            Runtime.getRuntime().addShutdownHook(this.shutdownHook);
        }
    }
```

对应的在web项目中就有对应的调用

```java
    private void refreshContext(ConfigurableApplicationContext context) {
        if (this.registerShutdownHook) {
            try {
                context.registerShutdownHook();
            }
            catch (AccessControlException ex
                // Not allowed in some envir
            }
        }
        refresh((ApplicationContext) context);
    }
```

Usages of registerShutdownHook() — Results in 'Project and Libraries'

SpringApplication.java	399	context.registerShutdownHook();
ConfigurableApplicationContext.java	91	* {@link Thread#getName() Name} of the {@linkplain #registerShutdownHoo
ConfigurableApplicationContext.java	94	* @see #registerShutdownHook()
AbstractApplicationContext.java	974	* @see #registerShutdownHook()
AbstractApplicationContext.java	1000	* @see #registerShutdownHook()

这个就是Bean实例化的过程了，当然在实例化中的DI问题我们在下篇文章中重点分析。