# Spring源码-Bean的实例化

接下来我们看看Bean的实例化处理

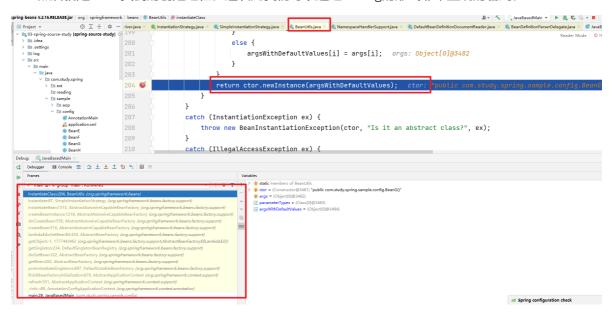
### — BeanDefinition

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

```
return createBean(beanName, mbd, args);
                    }
                                                           这块的Lambda表达式
                    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);
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) {
                                                          保证单例的创建
              if (this.singletonsCurrentlyInDestruction) {
                  throw new <a href="mailto:BeanCreationNotAllowedException">BeanCreationNotAllowedException</a>(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<>();
              }
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;
        }
```

// CLEATE DEAL THETAILE.

```
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

```
| tactory | support | DefaultSingletonBeanRegistry | SingletonObjects | AvapaBasedMain | DefaultSingletonBeanRegistry | SingletonSeanRegistry | SingletonStrategy,java | SimpleInstantiationStrategy,java | BeanUtils.java | NamespaceHandlerSupport.java | DefaultSingletonBeanRegistry | DefaultSingletonBeanRegistry | National | SimpleInstantiationStrategy,java | BeanUtils.java | NamespaceHandlerSupport.java | DefaultSingletonBeanRegistry | DefaultSingletonStategy, Java | DefaultSingletonBeanRegistry | DefaultSingletonBeanRegi
```

#### 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 🗴 👊 BeanUtilis.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 ->
                       \ensuremath{/\!/} 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) {
```

## 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

```
03-spring-source-study [spring-source-study] D:
> in .idea
> in .settings
> in log
                              9
                                    @Component
                                    @ImportResource("classpath:com/study/spring/sample/config/application.xml")
  ∨ 🗎 main
                                    public class BeanG {
      > 🖿 ext
                                        @Autowired
      > sample
> aop
> config
                                        private BeanF beanf;
                            15
                                        public void dog() {
           application.xml
                                             System.out.println("----");
                            18
                                             this.beanf.do1();
                            19
            20
                                        @PreDestroy
                                        public void destroy(){
        > 🖿 factory
        > 🛅 jta
                                             System.out.println(this+" 销毁了...");
        d LearnStart
```

然后我们在测试的案例中显示的调用 close 方法

```
public static void main(String[] args) {
    AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(...basePackages: "com.stude
    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.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
                         8 🕨 🖯
                                   public static void main(String[] args) {
                          9
                                       AnnotationConfigApplicationContext context = new AnnotationConfigApplicationConte
                        10
           🏭 applic

© BeanE
                                       BeanG bg = context.getBean(BeanG.class);
           @ Beanf
                                      bq.doq();
           © BeanG
                                       // 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();
         di
factory
                         15
                                       context.stop();
        > 🛅 jta
> 🛅 tx
                                       context.close();
         ContrsDemo
                         18
    .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。

```
// Publish shutdown event.
   publishEvent(new ContextClosedEvent( source: this));
                                                           发布事件--》打印日志的原因
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);
                   从BeanFactory中销毁对应的单例Bean
}
// 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...
```

### 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()

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));
```

### 对应的在web项目中就有对应的调用



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。