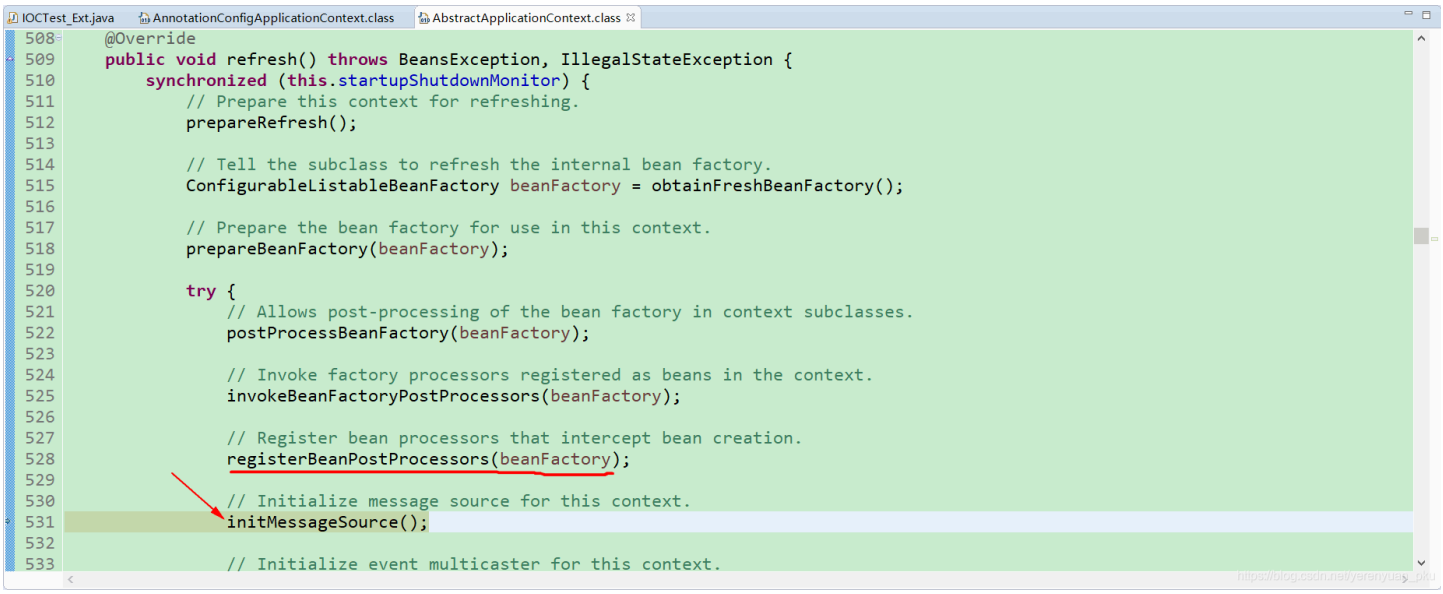
Spring注解驱动开发第44讲——Spring IOC容器创建源码解析(四)之初始化MessageSource组件

目录一览

写在前面

在上一讲中，我们已经搞清楚了如下registerBeanPostProcessors方法所做的事情，它无非就是来注册BeanPostProcessor的。



然后，我们让程序运行到以上第531行代码（即initMessageSource方法）处。顾名思义，该方法是来 初始化 MessageSource组件的。对于Spring MVC而言，该方法主要是来做国际化功能的，如消息绑定、消息解析等。

接下来，我们就得来研究研究initMessageSource方法里面究竟做了些什么事了。

初始化MessageSource组件

获取BeanFactory

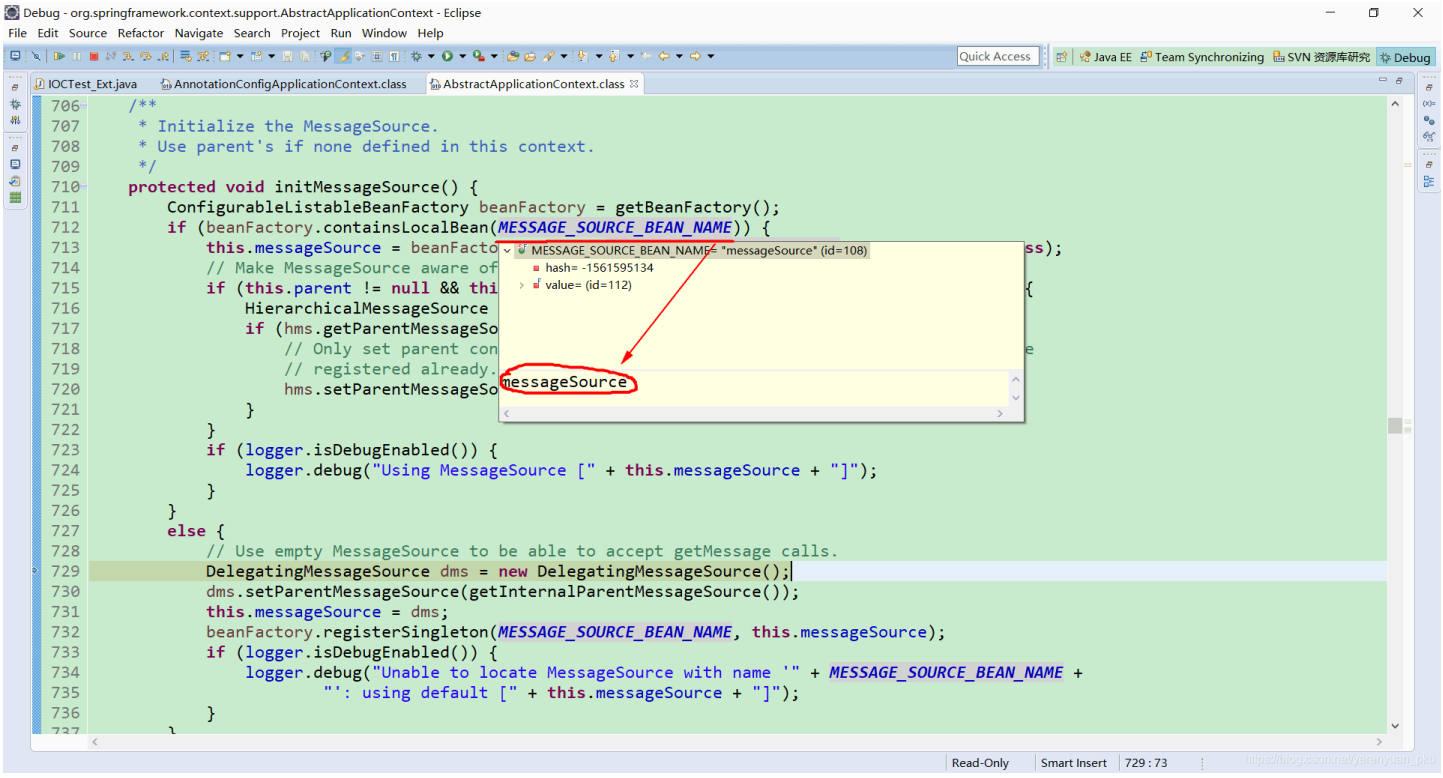
按下 F5 快捷键进入到initMessageSource方法里面，如下图所示，可以看到一开始是先来获取BeanFactory的。

```java
    * Use parent's if none defined in this context.
    */
   protected void initMessageSource() {
       ConfigurableListableBeanFactory beanFactory = getBeanFactory();
       if (beanFactory.containsLocalBean(MESSAGE_SOURCE_BEAN_NAME)) {
           this.messageSource = beanFactory.getBean(MESSAGE_SOURCE_BEAN_NAME, MessageSource.class);
           // Make MessageSource aware of parent MessageSource.
           if (this.parent != null && this.messageSource instanceof HierarchicalMessageSource) {
               HierarchicalMessageSource hms = (HierarchicalMessageSource) this.messageSource;
               if (hms.getParentMessageSource() == null) {
                   // Only set parent context as parent MessageSource if no parent MessageSource
                   // registered already.
                   hms.setParentMessageSource(getInternalParentMessageSource());
               }
           }
           if (logger.isDebugEnabled()) {
               logger.debug("Using MessageSource [" + this.messageSource + "]");
           }
       }
       else {
           // Use empty MessageSource to be able to accept getMessage calls.
           DelegatingMessageSource dms = new DelegatingMessageSource();
           dms.setParentMessageSource(getInternalParentMessageSource());
           this.messageSource = dms;
           beanFactory.registerSingleton(MESSAGE_SOURCE_BEAN_NAME, this.messageSource);
           if (logger.isDebugEnabled()) {
               logger.debug("Unable to locate MessageSource with name '" + MESSAGE_SOURCE_BEAN_NAME +
                       "': using default [" + this.messageSource + "]");
           }
       }
   }
```

而这个BeanFactory，我们之前早就准备好了。

看容器中是否有id为messageSource，类型是MessageSource的组件

按下 F6 快捷键让程序继续往下运行，会发现有一个判断，即判断BeanFactory中是否有一个id为messageSource的组件。我为什么会这么说呢，你只要看一下常量 MESSAGE_SOURCE_BEAN_NAME 的值就知道了，如下图所示，该常量的值就是messageSource。



若有，则赋值给this.messageSource

如果有的话，那么会从BeanFactory中获取到id为messageSource，类型是MessageSource的组件，并将其赋值给 this.messageSource 。这可以从下面这行代码看出。

```java
    * Use parent's if none defined in this context.
    */
   protected void initMessageSource() {
       ConfigurableListableBeanFactory beanFactory = getBeanFactory();
       if (beanFactory.containsLocalBean(MESSAGE_SOURCE_BEAN_NAME)) {
           this.messageSource = beanFactory.getBean(MESSAGE_SOURCE_BEAN_NAME, MessageSource.class);
           // Make MessageSource aware of parent MessageSource.
           if (this.parent != null && this.messageSource instanceof HierarchicalMessageSource) {
               HierarchicalMessageSource hms = (HierarchicalMessageSource) this.messageSource;
               if (hms.getParentMessageSource() == null) {
                   // Only set parent context as parent MessageSource if no parent MessageSource
                   // registered already.
                   hms.setParentMessageSource(getInternalParentMessageSource());
               }
           }
           if (logger.isDebugEnabled()) {
               logger.debug("Using MessageSource [" + this.messageSource + "]");
           }
       }
       else {
           // Use empty MessageSource to be able to accept getMessage calls.
           DelegatingMessageSource dms = new DelegatingMessageSource();
           dms.setParentMessageSource(getInternalParentMessageSource());
           this.messageSource = dms;
           beanFactory.registerSingleton(MESSAGE_SOURCE_BEAN_NAME, this.messageSource);
           if (logger.isDebugEnabled()) {
               logger.debug("Unable to locate MessageSource with name '" + MESSAGE_SOURCE_BEAN_NAME +
                       "': using default [" + this.messageSource + "]");
           }
       }
   }
```

很显然，容器刚开始创建的时候，肯定是还没有的，所以程序会来到下面的else语句中。

若没有，则创建一个DelegatingMessageSource类型的组件，并把创建好的组件注册在容器中

如果没有的话，那么Spring自己会创建一个DelegatingMessageSource类型的对象，即MessageSource类型的组件。

那么问题来了，这种MessageSource类型的组件有啥作用呢？我们不妨查看一下MessageSource接口的源码，如下图所示，它里面定义了很多重载的getMessage方法，该方法可以从配置文件（特别是国际化配置文件）中取出某一个key所对应的值。

```java
public interface MessageSource {

    /**
     * Try to resolve the message. Return default message if no message was found.
     * @param code the code to lookup up, such as 'calculator.noRateSet'. Users of
     * this class are encouraged to base message names on the relevant fully
     * qualified class name, thus avoiding conflict and ensuring maximum clarity.
     * @param args an array of arguments that will be filled in for params within
     * the message (params look like "{0}", "{1,date}", "{2,time}" within a message),
     * or {@code null} if none.
     * @param defaultMessage a default message to return if the lookup fails
     * @param locale the locale in which to do the lookup
     * @return the resolved message if the lookup was successful;
     * otherwise the default message passed as a parameter
     * @see java.text.MessageFormat
     */
    String getMessage(String code, Object[] args, String defaultMessage, Locale locale);

    /**
     * Try to resolve the message. Treat as an error if the message can't be found.
     * @param code the code to lookup up, such as 'calculator.noRateSet'
     * @param args an array of arguments that will be filled in for params within
     * the message (params look like "{0}", "{1,date}", "{2,time}" within a message),
     * or {@code null} if none.
     * @param locale the locale in which to do the lookup
     * @return the resolved message
     * @throws NoSuchMessageException if the message wasn't found
     * @see java.text.MessageFormat
     */
    String getMessage(String code, Object[] args, Locale locale) throws NoSuchMessageException;

    /**
```


也就是说，这种MessageSource类型的组件的作用一般是取出国际化配置文件中某个key所对应的值，而且还能按照区域信息获取哟~

紧接着，把创建好的MessageSource类型的组件注册到容器中，所执行的是下面这行代码。

IOCTest_Ext.java | AnnotationConfigApplicationContext.class | AbstractApplicationContext.class

```
709     */
710    protected void initMessageSource() {
711        ConfigurableListableBeanFactory beanFactory = getBeanFactory();
712        if (beanFactory.containsLocalBean(MESSAGE_SOURCE_BEAN_NAME)) {
713            this.messageSource = beanFactory.getBean(MESSAGE_SOURCE_BEAN_NAME, MessageSource.class);
714            // Make MessageSource aware of parent MessageSource.
715            if (this.parent != null && this.messageSource instanceof HierarchicalMessageSource) {
716                HierarchicalMessageSource hms = (HierarchicalMessageSource) this.messageSource;
717                if (hms.getParentMessageSource() == null) {
718                    // Only set parent context as parent MessageSource if no parent MessageSource
719                    // registered already.
720                    hms.setParentMessageSource(getInternalParentMessageSource());
721                }
722            }
723            if (logger.isDebugEnabled()) {
724                logger.debug("Using MessageSource [" + this.messageSource + "]");
725            }
726        }
727        else {
728            // Use empty MessageSource to be able to accept getMessage calls.
729            DelegatingMessageSource dms = new DelegatingMessageSource();
730            dms.setParentMessageSource(getInternalParentMessageSource());
731            this.messageSource = dms;
732            beanFactory.registerSingleton(MESSAGE_SOURCE_BEAN_NAME, this.messageSource);
733            if (logger.isDebugEnabled()) {
734                logger.debug("Unable to locate MessageSource with name '" + MESSAGE_SOURCE_BEAN_NAME +
735                        "': using default [" + this.messageSource + "]");
736            }
737        }
738    }
739
740    /**
```

那么，我们以后想获取国际化配置文件中的值的时候，就可以直接自动注入这个MessageSource类型的组件了，然后调用它的getMessage方法就行了，并且还能按照区域信息获取哟😄