Spring IOC核心源码学习

2016/04/18 | 分类： [基础技术](http://www.importnew.com/cat/basic) | [0 条评论](http://www.importnew.com/19243.html#respond) | 标签： [IOC](http://www.importnew.com/tag/ioc), [SPRING](http://www.importnew.com/tag/spring)

分享到：**0**

原文出处： [Yikun](http://yikun.github.io/2015/05/29/Spring-IOC%E6%A0%B8%E5%BF%83%E6%BA%90%E7%A0%81%E5%AD%A6%E4%B9%A0/)

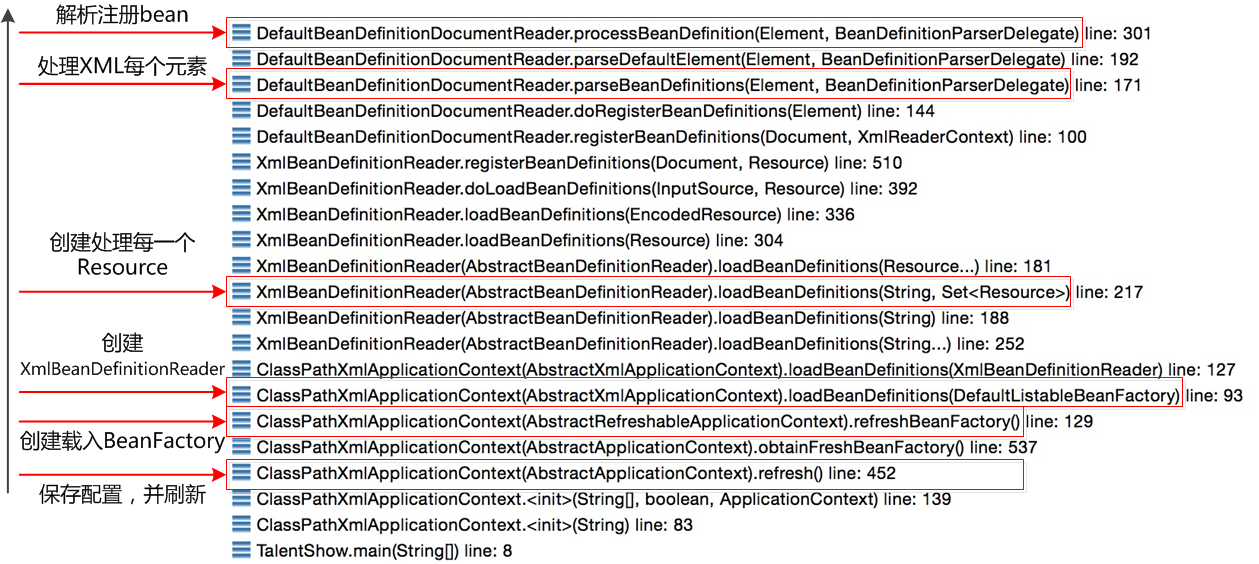
1. 初始化

大致单步跟了下Spring IOC的初始化过程，整个脉络很庞大，初始化的过程主要就是**读取XML资源，并解析，最终注册到Bean Factory中**：

[https://cloud.githubusercontent.com/assets/1736354/7897341/032179be-070b-11e5-9ecf-d7befc804e9d.png](https://cloud.githubusercontent.com/assets/1736354/7897341/032179be-070b-11e5-9ecf-d7befc804e9d.png)

在完成初始化的过程后，Bean们就在BeanFactory中蓄势以待地等调用了。下面通过一个具体的例子，来详细地学习一下初始化过程，例如当加载下面一个bean：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <bean id="XiaoWang" class="com.springstudy.talentshow.SuperInstrumentalist">      <property name="instruments">          <list>              <ref bean="piano"/>              <ref bean="saxophone"/>          </list>      </property>  </bean> |

加载时需要读取、解析、注册bean，这个过程具体的调用栈如下所示：  
[](https://cloud.githubusercontent.com/assets/1736354/7896285/8a488060-06e6-11e5-9ad9-4ddd3375984f.png)

下面对每一步的关键的代码进行详细分析：

1.1 准备

**保存配置位置，并刷新**  
在调用ClassPathXmlApplicationContext后，先会将配置位置信息保存到configLocations，供后面解析使用，之后，会调用AbstractApplicationContext的refresh方法进行刷新：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | public ClassPathXmlApplicationContext(String[] configLocations, boolean refresh,          ApplicationContext parent) throws BeansException {        super(parent);      // 保存位置信息，比如`com/springstudy/talentshow/talent-show.xml`      setConfigLocations(configLocations);      if (refresh) {          // 刷新          refresh();      }  }    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();          }          catch (BeansException ex) {              // Destroy already created singletons to avoid dangling resources.              destroyBeans();              // Reset 'active' flag.              cancelRefresh(ex);              // Propagate exception to caller.              throw ex;          }      }  } |

**创建载入BeanFactory**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | protected final void refreshBeanFactory() throws BeansException {      // ... ...      DefaultListableBeanFactory beanFactory = createBeanFactory();      // ... ...      loadBeanDefinitions(beanFactory);      // ... ...  } |

**创建XMLBeanDefinitionReader**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | protected void loadBeanDefinitions(DefaultListableBeanFactory beanFactory)       throws BeansException, IOException {      // Create a new XmlBeanDefinitionReader for the given BeanFactory.      XmlBeanDefinitionReader beanDefinitionReader = new XmlBeanDefinitionReader(beanFactory);      // ... ...      // Allow a subclass to provide custom initialization of the reader,      // then proceed with actually loading the bean definitions.      initBeanDefinitionReader(beanDefinitionReader);      loadBeanDefinitions(beanDefinitionReader);  } |

1.2 读取

**创建处理每一个resource**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public int loadBeanDefinitions(String location, Set<Resource> actualResources)       throws BeanDefinitionStoreException {      // ... ...      // 通过Location来读取Resource      Resource[] resources = ((ResourcePatternResolver) resourceLoader).getResources(location);      int loadCount = loadBeanDefinitions(resources);      // ... ...  }    public int loadBeanDefinitions(Resource... resources) throws BeanDefinitionStoreException {      Assert.notNull(resources, "Resource array must not be null");      int counter = 0;      for (Resource resource : resources) {          // 载入每一个resource          counter += loadBeanDefinitions(resource);      }      return counter;  } |

**处理XML每个元素**

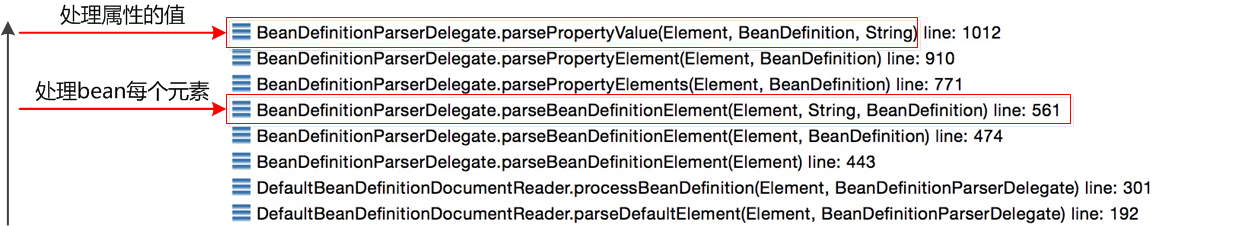
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | protected void parseBeanDefinitions(Element root, BeanDefinitionParserDelegate delegate) {      // ... ...      NodeList nl = root.getChildNodes();      for (int i = 0; i < nl.getLength(); i++) {          Node node = nl.item(i);          if (node instanceof Element) {              Element ele = (Element) node;              if (delegate.isDefaultNamespace(ele)) {                  // 处理每个xml中的元素，可能是import、alias、bean                  parseDefaultElement(ele, delegate);              }              else {                  delegate.parseCustomElement(ele);              }          }      }      // ... ...  } |

**解析和注册bean**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | protected void processBeanDefinition(Element ele, BeanDefinitionParserDelegate delegate) {      // 解析      BeanDefinitionHolder bdHolder = delegate.parseBeanDefinitionElement(ele);      if (bdHolder != null) {          bdHolder = delegate.decorateBeanDefinitionIfRequired(ele, bdHolder);          try {              // 注册              // Register the final decorated instance.              BeanDefinitionReaderUtils.registerBeanDefinition(                  bdHolder, getReaderContext().getRegistry());          }          catch (BeanDefinitionStoreException ex) {              getReaderContext().error("Failed to register bean definition with name '" +                      bdHolder.getBeanName() + "'", ele, ex);          }          // Send registration event.          getReaderContext().fireComponentRegistered(new BeanComponentDefinition(bdHolder));      }  } |

本步骤中，通过parseBeanDefinitionElement将XML的元素解析为BeanDefinition，然后存在BeanDefinitionHolder中，然后再利用BeanDefinitionHolder将BeanDefinition注册，实质就是把BeanDefinition的实例put进BeanFactory中，和后面将详细的介绍解析和注册过程。

1.3 解析

[](https://cloud.githubusercontent.com/assets/1736354/7896302/eae02bc6-06e6-11e5-941a-d1f59e3b363f.png)

**处理每个Bean的元素**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | public AbstractBeanDefinition parseBeanDefinitionElement(          Element ele, String beanName, BeanDefinition containingBean) {        // ... ...      // 创建beandefinition      AbstractBeanDefinition bd = createBeanDefinition(className, parent);        parseBeanDefinitionAttributes(ele, beanName, containingBean, bd);      bd.setDescription(DomUtils.getChildElementValueByTagName(ele, DESCRIPTION\_ELEMENT));        parseMetaElements(ele, bd);      parseLookupOverrideSubElements(ele, bd.getMethodOverrides());      parseReplacedMethodSubElements(ele, bd.getMethodOverrides());      // 处理“Constructor”      parseConstructorArgElements(ele, bd);      // 处理“Preperty”      parsePropertyElements(ele, bd);      parseQualifierElements(ele, bd);      // ... ...  } |

**处理属性的值**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | public Object parsePropertyValue(Element ele, BeanDefinition bd, String propertyName) {      String elementName = (propertyName != null) ?                      "<property> element for property '" + propertyName + "'" :                      "<constructor-arg> element";        // ... ...      if (hasRefAttribute) {      // 处理引用          String refName = ele.getAttribute(REF\_ATTRIBUTE);          if (!StringUtils.hasText(refName)) {              error(elementName + " contains empty 'ref' attribute", ele);          }          RuntimeBeanReference ref = new RuntimeBeanReference(refName);          ref.setSource(extractSource(ele));          return ref;      }      else if (hasValueAttribute) {      // 处理值          TypedStringValue valueHolder = new TypedStringValue(ele.getAttribute(VALUE\_ATTRIBUTE));          valueHolder.setSource(extractSource(ele));          return valueHolder;      }      else if (subElement != null) {      // 处理子类型（比如list、map等）          return parsePropertySubElement(subElement, bd);      }      // ... ...  } |

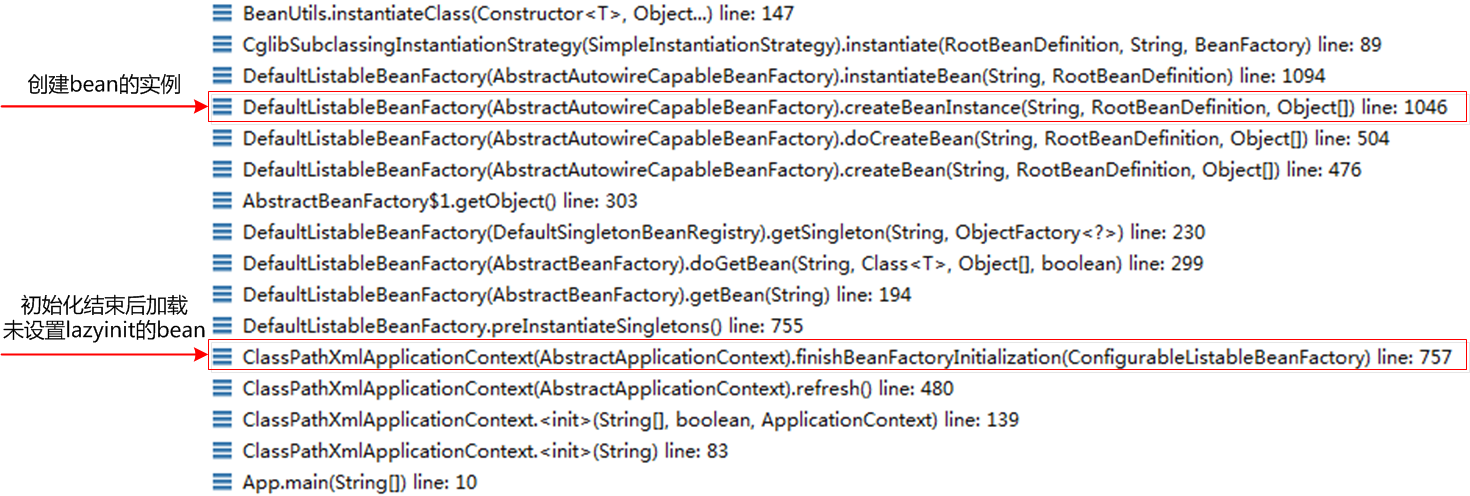
1.4 注册

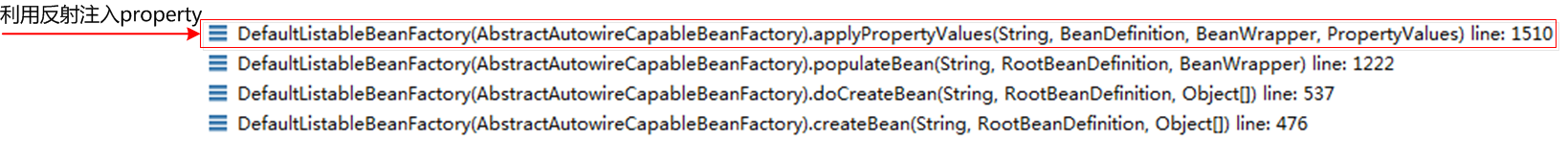
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | public static void registerBeanDefinition(          BeanDefinitionHolder definitionHolder, BeanDefinitionRegistry registry)          throws BeanDefinitionStoreException {        // Register bean definition under primary name.      String beanName = definitionHolder.getBeanName();      registry.registerBeanDefinition(beanName, definitionHolder.getBeanDefinition());        // Register aliases for bean name, if any.      String[] aliases = definitionHolder.getAliases();      if (aliases != null) {          for (String alias : aliases) {              registry.registerAlias(beanName, alias);          }      }  }    public void registerBeanDefinition(String beanName, BeanDefinition beanDefinition)          throws BeanDefinitionStoreException {        // ......        // 将beanDefinition注册      this.beanDefinitionMap.put(beanName, beanDefinition);        // ......  } |

注册过程中，最核心的一句就是：this.beanDefinitionMap.put(beanName, beanDefinition)，也就是说注册的实质就是以beanName为key，以beanDefinition为value，将其put到HashMap中。

2. 注入依赖

当完成初始化IOC容器后，如果bean没有设置lazy-init(延迟加载)属性，那么bean的实例就会在初始化IOC完成之后，及时地进行初始化。初始化时会先建立实例，然后根据配置利用反射对实例进行进一步操作，具体流程如下所示：  
[https://cloud.githubusercontent.com/assets/1736354/7929429/615570ea-0930-11e5-8097-ae982ef7709d.png](https://cloud.githubusercontent.com/assets/1736354/7929429/615570ea-0930-11e5-8097-ae982ef7709d.png)

**创建bean的实例**  
创建bean的实例过程函数调用栈如下所示：  
[](https://cloud.githubusercontent.com/assets/1736354/7929379/cec01bcc-092f-11e5-81ad-88c285f33845.png)

**注入bean的属性**  
注入bean的属性过程函数调用栈如下所示：  
[](https://cloud.githubusercontent.com/assets/1736354/7929381/db58350e-092f-11e5-82a4-caaf349291ea.png)

在创建bean和注入bean的属性时，都是在doCreateBean函数中进行的，我们重点看下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | protected Object doCreateBean(final String beanName, final RootBeanDefinition mbd,          final Object[] args) {      // Instantiate the bean.      BeanWrapper instanceWrapper = null;      if (mbd.isSingleton()) {          instanceWrapper = this.factoryBeanInstanceCache.remove(beanName);      }      if (instanceWrapper == null) {          // 创建bean的实例          instanceWrapper = createBeanInstance(beanName, mbd, args);      }        // ... ...        // Initialize the bean instance.      Object exposedObject = bean;      try {          // 初始化bean的实例，如注入属性          populateBean(beanName, mbd, instanceWrapper);          if (exposedObject != null) {              exposedObject = initializeBean(beanName, exposedObject, mbd);          }      }        // ... ...  } |

理解了以上两个过程，我们就可以自己实现一个简单的Spring框架了。于是，我根据自己的理解实现了一个简单的IOC框架[Simple Spring](https://github.com/Yikun/simple-spring)，有兴趣可以看看。