Spring源码分析解读

## 如何查看源码

Spring源码下载https://github.com/spring-projects/spring-framework/tags?after=v3.1.0.RC1

## 源代码结构组织

Build-spring-framework是整个Spring源代码的构建目录，里面是项目的构建脚本，如果要自己动手构建Spring，可以进入这个目录使用ANT进行构建。

l  org.springframework.context是IoC容器的源代码目录

l  org.springframework.aop是AOP实现的源代码目录

l  org.springframework.jdbc是JDBC的源代码部分

l  org.springframework.orm是O/R Mapping对应的源代码实现部分

# SpringIOC源码分析

## IOC初始化

1、 XmlBeanFactory(屌丝IOC)的整个流程

2、 FileSystemXmlApplicationContext 的IOC容器流程

1、高富帅IOC解剖

2、 设置资源加载器和资源定位

3、AbstractApplicationContext的refresh函数载入Bean定义过程：

4、AbstractApplicationContext子类的refreshBeanFactory()方法：

5、AbstractRefreshableApplicationContext子类的loadBeanDefinitions方法：

6、AbstractBeanDefinitionReader读取Bean定义资源：

7、资源加载器获取要读入的资源：

8、XmlBeanDefinitionReader加载Bean定义资源：

9、DocumentLoader将Bean定义资源转换为Document对象：

10、XmlBeanDefinitionReader解析载入的Bean定义资源文件：

11、DefaultBeanDefinitionDocumentReader对Bean定义的Document对象解析：

12、BeanDefinitionParserDelegate解析Bean定义资源文件中的<Bean>元素：

13、BeanDefinitionParserDelegate解析<property>元素：

14、解析<property>元素的子元素：

15、解析<list>子元素：

16、解析过后的BeanDefinition在IoC容器中的注册：

17、DefaultListableBeanFactory向IoC容器注册解析后的BeanDefinition：

## IOC体系

BeanFactory

         Spring Bean的创建是典型的工厂模式，这一系列的Bean工厂，也即IOC容器为开发者管理对象间的依赖关系提供了很多便利和基础服务，在Spring中有许多的IOC容器的实现供用户选择和使用，其相互关系如下：



## BeanFactory

 BeanFactory定义了 IOC 容器的最基本形式，并提供了 IOC 容器应遵守的的最基本的接口，也就是Spring IOC 所遵守的最底层和最基本的编程规范。在  Spring 代码中， BeanFactory 只是个接口，并不是 IOC容器的具体实现，但是 Spring 容器给出了很多种实现，如 DefaultListableBeanFactory 、 XmlBeanFactory 、ApplicationContext 等，都是附加了某种功能的实现。

|  |
| --- |
| **public** **interface** BeanFactory {  //这里是对FactoryBean的转义定义，因为如果使用bean的名字检索FactoryBean得到的对象是工厂生成的对象，    //如果需要得到工厂本身，需要转义  //转义符“&”用来获取FactoryBean本身  String ***FACTORY\_BEAN\_PREFIX*** = "&";  //根据bean的名字进行获取bean的实例，这是IOC最大的抽象方法  Object getBean(String name) **throws** BeansException;  //根据bean的名字和Class类型进行获取Bean的实例,和上面方法不同的是，bean名字和Bean 的class类型不同时候会爆出异常  <T> T getBean(String name, Class<T> requiredType) **throws** BeansException;  <T> T getBean(Class<T> requiredType) **throws** BeansException;  Object getBean(String name, Object... args) **throws** BeansException;  //检测这个IOC容器中是否含有这个Bean  **boolean** containsBean(String name);  //判断这个Bean是不是单利  **boolean** isSingleton(String name) **throws** NoSuchBeanDefinitionException;  //判断这个Bean是不是原型  **boolean** isPrototype(String name) **throws** NoSuchBeanDefinitionException;  //查询指定的bean的名字和Class类型是不是指定的Class类型  **boolean** isTypeMatch(String name, Class targetType) **throws** NoSuchBeanDefinitionException;  //这里对得到bean实例的Class类型  Class<?> getType(String name) **throws** NoSuchBeanDefinitionException;  //这里得到bean的别名，如果根据别名检索，那么其原名也会被检索出来  String[] getAliases(String name);  } |

## BeanDefinition

|  |
| --- |
| 这个接口，可以理解为xml bean元素的数据载体。通过对比xml bean标签的属性列表和BeanDefinition的属性列表一看便知。  我的理解，是解析XML的过程，就是 xml <bean>元素内容 转换为BeanDefinition对象的过程。而且这个接口，支持层级，对应对象的继承。  有一个类BeanDefinitionHolder,BeanDefinitionHolder，根据名称或者别名持有beanDefinition，它承载了name和BeanDefinition的映射信息。  BeanWarpper:  提供对标准javabean的分析和操作方法：单个或者批量获取和设置属性值，获取属性描述符，查询属性的可读性和可写性等。支持属性的嵌套设置，深度没有限制。  AbstractRefreshableApplicationContext的refreshBeanFactory()这个方法 |

|  |
| --- |
| **protected** **final** **void** refreshBeanFactory() **throws** BeansException {  **if** (hasBeanFactory()) {  destroyBeans();  closeBeanFactory();  }  **try** {  DefaultListableBeanFactory beanFactory = createBeanFactory();//创建IOC容器  beanFactory.setSerializationId(getId());  customizeBeanFactory(beanFactory);  loadBeanDefinitions(beanFactory);//载入loadBeanDefinitions  **synchronized** (**this**.beanFactoryMonitor) {  **this**.beanFactory = beanFactory;  }  }  **catch** (IOException ex) {  **throw** **new** ApplicationContextException("I/O error parsing bean definition source for " + getDisplayName(), ex);  }  }  **public abstract class AbstractXmlApplicationContext extends AbstractRefreshableConfigApplicationContext { 实现**  /\*\*  \* Loads the bean definitions via an XmlBeanDefinitionReader.  \* **@see** org.springframework.beans.factory.xml.XmlBeanDefinitionReader  \* **@see** #initBeanDefinitionReader  \* **@see** #loadBeanDefinitions  \*/  @Override  **protected** **void** loadBeanDefinitions(DefaultListableBeanFactory beanFactory) **throws** BeansException, IOException {  // Create a new XmlBeanDefinitionReader for the given BeanFactory.  XmlBeanDefinitionReader beanDefinitionReader = **new** XmlBeanDefinitionReader(beanFactory);  // Configure the bean definition reader with this context's  // resource loading environment.  beanDefinitionReader.setResourceLoader(**this**);  beanDefinitionReader.setEntityResolver(**new** ResourceEntityResolver(**this**));  // Allow a subclass to provide custom initialization of the reader,  // then proceed with actually loading the bean definitions.  initBeanDefinitionReader(beanDefinitionReader);  loadBeanDefinitions(beanDefinitionReader);  }  先调用本类里面的loadBeanDefinitions  **protected** **void** loadBeanDefinitions(XmlBeanDefinitionReader reader) **throws** BeansException, IOException {  Resource[] configResources = getConfigResources();  **if** (configResources != **null**) {  reader.loadBeanDefinitions(configResources);  }  String[] configLocations = getConfigLocations();  **if** (configLocations != **null**) {  reader.loadBeanDefinitions(configLocations);  }  }  委托给reader.loadBeanDefinitions(configLocation);    XmlBeanDefinitionReader 通过XmlBeanDefinitionReader来读取。下面看一下XmlBeanDefinitionReader这个方法，但其实并不在这个类实现这个方法，而是在它的基类里面AbstractBeanDefinitionReader  **public** **int** loadBeanDefinitions(String... locations) **throws** BeanDefinitionStoreException {  Assert.*notNull*(locations, "Location array must not be null");  **int** counter = 0;  **for** (String location : locations) {  counter += loadBeanDefinitions(location);  }  **return** counter;  }  进入到loadBeanDefinitions  **public** **int** loadBeanDefinitions(EncodedResource encodedResource) **throws** BeanDefinitionStoreException {  Assert.*notNull*(encodedResource, "EncodedResource must not be null");  **if** (logger.isInfoEnabled()) {  logger.info("Loading XML bean definitions from " + encodedResource.getResource());  }  Set<EncodedResource> currentResources = **this**.resourcesCurrentlyBeingLoaded.get();  **if** (currentResources == **null**) {  currentResources = **new** HashSet<EncodedResource>(4);  **this**.resourcesCurrentlyBeingLoaded.set(currentResources);  }  **if** (!currentResources.add(encodedResource)) {  **throw** **new** BeanDefinitionStoreException(  "Detected cyclic loading of " + encodedResource + " - check your import definitions!");  }  **try** {  InputStream inputStream = encodedResource.getResource().getInputStream();//获取IO  **try** {  InputSource inputSource = **new** InputSource(inputStream);  **if** (encodedResource.getEncoding() != **null**) {  inputSource.setEncoding(encodedResource.getEncoding());  }  **return** doLoadBeanDefinitions(inputSource, encodedResource.getResource());//这个方法从流中读取  }  **finally** {  inputStream.close();  }  }  **catch** (IOException ex) {  **throw** **new** BeanDefinitionStoreException(  "IOException parsing XML document from " + encodedResource.getResource(), ex);  }  **finally** {  currentResources.remove(encodedResource);  **if** (currentResources.isEmpty()) {  **this**.resourcesCurrentlyBeingLoaded.remove();  }  }  }  进入到doLoadBeanDefinitions Resource IO封装  **protected** **int** doLoadBeanDefinitions(InputSource inputSource, Resource resource)  **throws** BeanDefinitionStoreException {  **try** {  **int** validationMode = getValidationModeForResource(resource);  Document doc = **this**.documentLoader.loadDocument(  inputSource, getEntityResolver(), **this**.errorHandler, validationMode, isNamespaceAware());  **return** registerBeanDefinitions(doc, resource); //解析XML  }  **catch** (BeanDefinitionStoreException ex) {  **throw** ex;  }  **catch** (SAXParseException ex) {  **throw** **new** XmlBeanDefinitionStoreException(resource.getDescription(),  "Line " + ex.getLineNumber() + " in XML document from " + resource + " is invalid", ex);  }  **catch** (SAXException ex) {  **throw** **new** XmlBeanDefinitionStoreException(resource.getDescription(),  "XML document from " + resource + " is invalid", ex);  }  **catch** (ParserConfigurationException ex) {  **throw** **new** BeanDefinitionStoreException(resource.getDescription(),  "Parser configuration exception parsing XML from " + resource, ex);  }  **catch** (IOException ex) {  **throw** **new** BeanDefinitionStoreException(resource.getDescription(),  "IOException parsing XML document from " + resource, ex);  }  **catch** (Throwable ex) {  **throw** **new** BeanDefinitionStoreException(resource.getDescription(),  "Unexpected exception parsing XML document from " + resource, ex);  }  }  进入到registerBeanDefinitions  /\*\*  \* Register the bean definitions contained in the given DOM document.  \* Called by <code>loadBeanDefinitions</code>.  \* <p>Creates a new instance of the parser class and invokes  \* <code>registerBeanDefinitions</code> on it.  \* **@param** doc the DOM document  \* **@param** resource the resource descriptor (for context information)  \* **@return** the number of bean definitions found  \* **@throws** BeanDefinitionStoreException in case of parsing errors  \* **@see** #loadBeanDefinitions  \* **@see** #setDocumentReaderClass  \* **@see** BeanDefinitionDocumentReader#registerBeanDefinitions  \*/  **public** **int** registerBeanDefinitions(Document doc, Resource resource) **throws** BeanDefinitionStoreException {  // Read document based on new BeanDefinitionDocumentReader SPI.  BeanDefinitionDocumentReader documentReader = createBeanDefinitionDocumentReader();  **int** countBefore = getRegistry().getBeanDefinitionCount();  documentReader.registerBeanDefinitions(doc, createReaderContext(resource));  **return** getRegistry().getBeanDefinitionCount() - countBefore;  }  documentReader.registerBeanDefinitionsXML解析  /\*\*  \* Parses bean definitions according to the "spring-beans" DTD.  \* <p>Opens a DOM Document; then initializes the default settings  \* specified at <code>&lt;beans&gt;</code> level; then parses  \* the contained bean definitions.  \*/  **public** **void** registerBeanDefinitions(Document doc, XmlReaderContext readerContext) {  **this**.readerContext = readerContext;  logger.debug("Loading bean definitions");  Element root = doc.getDocumentElement();  BeanDefinitionParserDelegate delegate = createHelper(readerContext, root);  preProcessXml(root);  parseBeanDefinitions(root, delegate);  postProcessXml(root);  }  **-----遍历节点**  **protected** **void** parseBeanDefinitions(Element root, BeanDefinitionParserDelegate delegate) {  **if** (delegate.isDefaultNamespace(root)) {  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)) {  parseDefaultElement(ele, delegate); //默认解析  }  **else** {  delegate.parseCustomElement(ele);  }  }  }  }  **else** {  delegate.parseCustomElement(root);  }  }  ---判断解析类  **private** **void** parseDefaultElement(Element ele, BeanDefinitionParserDelegate delegate) {  **if** (delegate.nodeNameEquals(ele, ***IMPORT\_ELEMENT***)) {  importBeanDefinitionResource(ele);//import类型  }  **else** **if** (delegate.nodeNameEquals(ele, ***ALIAS\_ELEMENT***)) {  processAliasRegistration(ele);//别名方式  }  **else** **if** (delegate.nodeNameEquals(ele, ***BEAN\_ELEMENT***)) {  processBeanDefinition(ele, delegate);//bean解析方式  }  } |

## Bean的解析方式

进入到 AbstractBeanDefinition beanDefinition = parseBeanDefinitionElement(ele, beanName, containingBean); 使用反射初始化类

|  |
| --- |
| **public** AbstractBeanDefinition parseBeanDefinitionElement(  Element ele, String beanName, BeanDefinition containingBean) {  **this**.parseState.push(**new** BeanEntry(beanName));  String className = **null**;  **if** (ele.hasAttribute(***CLASS\_ATTRIBUTE***)) {  className = ele.getAttribute(***CLASS\_ATTRIBUTE***).trim();  }  **try** {  String parent = **null**;  **if** (ele.hasAttribute(***PARENT\_ATTRIBUTE***)) {  parent = ele.getAttribute(***PARENT\_ATTRIBUTE***);  }  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());  parseConstructorArgElements(ele, bd);  parsePropertyElements(ele, bd);  parseQualifierElements(ele, bd);  bd.setResource(**this**.readerContext.getResource());  bd.setSource(extractSource(ele));  **return** bd;  }  **catch** (ClassNotFoundException ex) {  error("Bean class [" + className + "] not found", ele, ex);  }  **catch** (NoClassDefFoundError err) {  error("Class that bean class [" + className + "] depends on not found", ele, err);  }  **catch** (Throwable ex) {  error("Unexpected failure during bean definition parsing", ele, ex);  }  **finally** {  **this**.parseState.pop();  }  **return** **null**;  }  进入到AbstractBeanDefinition bd = createBeanDefinition(className, parent);  **protected** AbstractBeanDefinition createBeanDefinition(String className, String parentName)  **throws** ClassNotFoundException {  **return** BeanDefinitionReaderUtils.*createBeanDefinition*(  parentName, className, **this**.readerContext.getBeanClassLoader());  }  进入到BeanDefinitionReaderUtils.*createBeanDefinition*  **public** **static** AbstractBeanDefinition createBeanDefinition(  String parentName, String className, ClassLoader classLoader) **throws** ClassNotFoundException {  GenericBeanDefinition bd = **new** GenericBeanDefinition();  bd.setParentName(parentName);  **if** (className != **null**) {  **if** (classLoader != **null**) {  bd.setBeanClass(ClassUtils.*forName*(className, classLoader));//使用java反射机制初始化  }  **else** {  bd.setBeanClassName(className);  }  }  **return** bd;  } |

## Bean生命周期分析

1. spring对bean进行实例化,默认bean是单例  
   2)spring对bean进行依赖注入  
   3)如果bean实现了BeanNameAware接口,spring将bean的id传给setBeanName()方法  
   4)如果bean实现了BeanFactoryAware接口,spring将调用setBeanFactory方法,将BeanFactory实例传进来  
   5)如果bean实现了ApplicationContextAware()接口,spring将调用setApplicationContext()方法将应用上下文的引用传入  
   6) 如果bean实现了BeanPostProcessor接口,spring将调用它们的postProcessBeforeInitialization接口方法  
   7) 如果bean实现了InitializingBean接口,spring将调用它们的afterPropertiesSet接口方法,类似的如果bean使用了init-method属性声明了初始化方法,改方法也会被调用  
   8)如果bean实现了BeanPostProcessor接口,spring将调用它们的postProcessAfterInitialization接口方法  
   9)此时bean已经准备就绪,可以被应用程序使用了,他们将一直驻留在应用上下文中,直到该应用上下文被销毁  
   10)若bean实现了DisposableBean接口,spring将调用它的distroy()接口方法。同样的,如果bean使用了destroy-method属性声明了销毁方法,则该方法被调用

|  |
| --- |
| 1. public class UserEntity 2. implements BeanFactoryAware, BeanNameAware, InitializingBean, DisposableBean, ApplicationContextAware { 3. private String userName; 4. private Integer age = null; 5. public UserEntity() { 6. System.*out*.println("无惨构造函数....."); 7. } 8. public UserEntity(String userName, Integer age) { 9. System.*out*.println("我是有参构造函数 userName:" + userName + ",age:" + age); 10. this.userName = userName; 11. this.age = age; 12. } 13. public String getUserName() { 14. return userName; 15. } 16. public void setUserName(String userName) { 17. this.userName = userName; 18. } 19. public Integer getAge() { 20. return age; 21. } 22. public void setAge(Integer age) { 23. this.age = age; 24. } 25. @Override 26. public String toString() { 27. return "UserEntity [userName=" + userName + ", age=" + age + "]"; 28. } 29. // bean容器销毁 30. public void destroy() throws Exception { 31. System.*out*.println("destroy() 销毁bean"); 32. } 33. // afterPropertiesSet方法，初始化bean的时候执行，可以针对某个具体的bean进行配置 34. public void afterPropertiesSet() throws Exception { 35. System.*out*.println("afterPropertiesSet"); 36. } 37. // 如果bean实现了BeanNameAware接口,spring将bean的id传给setBeanName()方法 38. public void setBeanName(String name) { 39. System.*out*.println("setBeanName() set name:" + name); 40. } 41. // 如果bean实现了BeanNameAware接口,spring将bean的id传给setBeanName()方法 42. public void setBeanFactory(BeanFactory beanFactory) throws BeansException { 43. System.*out*.println("setBeanFactory()"); 44. } 45. // 获取Spring容器上下文 46. public void setApplicationContext(ApplicationContext applicationContext) throws BeansException { 47. System.*out*.println("获取上下文...."); 48. } 49. } 50. --------------- |
| System.***out***.println("开始初始化容器....");  ClassPathXmlApplicationContext app = **new** ClassPathXmlApplicationContext("spring.xml");  System.***out***.println("容器初始化成功....");  UserEntity userEntity = (UserEntity) app.getBean("userEntity01");  System.***out***.println("关闭容器...");  app.registerShutdownHook(); |

# SpringAop源码分析

常用单词

BeanFactory (兵fa ke te rui)

BeanDefinition(兵 fu 来xin)