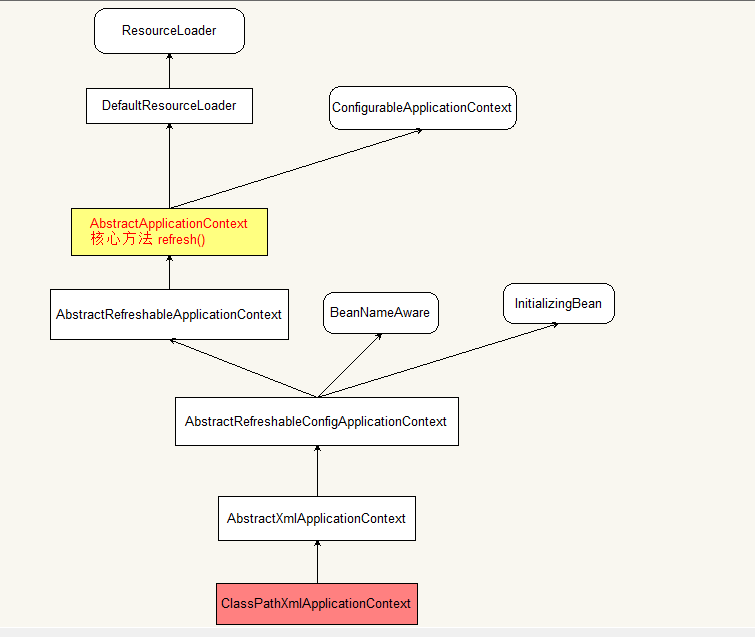
# ClassPathXmlApplicationContext



调用spring容器的类

**public** ClassPathXmlApplicationContext(  
 String[] configLocations, **boolean** refresh, **@Nullable** ApplicationContext parent)  
 **throws** BeansException {  
  
 **super**(parent);  
 setConfigLocations(configLocations);  
 **if** (refresh) {

// spring容器显示的核心方法  
 **refresh();**  
 }  
}

**1 refresh()**

refresh()方法的具体实现在AbstractApplicationContext类中

**abstract class** AbstractApplicationContext

**@Override  
public void refresh()** **throws** BeansException, IllegalStateException {  
 **synchronized** (**this**.**startupShutdownMonitor**) {

prepareRefresh();*ss to refresh the internal bean factory.*

***// 核心方法，xml解析，转换成BeanDefinitation，存入容器***ConfigurableListableBeanFactory beanFactory = **obtainFreshBeanFactory();**

prepareBeanFactory(beanFactory);  
 **try** {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) {  
 **if** (**logger**.isWarnEnabled()) {  
 **logger**.warn(**"Exception encountered during context initialization - "** +  
 **"cancelling refresh attempt: "** + ex);  
 }  
 *// Destroy already created singletons to avoid dangling resources.* destroyBeans();  
 *// Reset 'active' flag.* cancelRefresh(ex);  
 *// Propagate exception to caller.* **throw** ex;  
 }  
 **finally** {  
 *// Reset common introspection caches in Spring's core, since we  
 // might not ever need metadata for singleton beans anymore...* resetCommonCaches();  
 }  
 }  
}

**2 obtainFreshBeanFactory()**

**bean工厂刷新核心方法**

**abstract class** AbstractApplicationContext

obtainFreshBeanFactory();

具体代码

**protected** ConfigurableListableBeanFactory obtainFreshBeanFactory() {  
 **refreshBeanFactory();**  
 **return** getBeanFactory();  
}

**3 refreshBeanFactory();**

**abstract class AbstractRefreshableApplicationContext**

refreshBeanFactory();  
具体代码

**@Override  
protected final void** refreshBeanFactory() **throws** BeansException {  
 // 判断beanfacotry实例是否存在，如果存在清空关闭

**if** (hasBeanFactory()) {  
 destroyBeans();  
 closeBeanFactory();  
 }  
 **try** {  
 DefaultListableBeanFactory beanFactory = createBeanFactory();  
 beanFactory.setSerializationId(getId());  
 customizeBeanFactory(beanFactory);

// 核心加载bean类，bean的创建是在该类引导下进行的  
 **loadBeanDefinitions(beanFactory);** **synchronized** (**this**.**beanFactoryMonitor**) {  
 **this**.**beanFactory** = beanFactory;  
 }  
 }  
 **catch** (IOException ex) {  
 **throw new** ApplicationContextException(**"I/O error parsing bean definition source for "** + getDisplayName(), ex);  
 }  
}

**4 loadBeanDefinitions(beanFactory)**

**abstract class** AbstractXmlApplicationContext

*/\*\*  
 \* 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 {  
 ***// 创建构建xml配置文件的工具对象.*XmlBeanDefinitionReader**XmlBeanDefinitionReader beanDefinitionReader = **new** XmlBeanDefinitionReader(beanFactory);  
  
 *// Configure the bean definition reader with this context's  
 // resource loading environment.* beanDefinitionReader.setEnvironment(**this**.getEnvironment());

// 设置资源加载器  
 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);

// 核心方法  
  **loadBeanDefinitions(beanDefinitionReader);**}

**5 reader.loadBeanDefinitions(configLocations)**

**abstract class AbstractXmlApplicationContext** **extends** AbstractRefreshableConfigApplicationContext

/\*\*

使用上一个XmlBeanDefinitionReader 创建的对象，读取并解析xml配置文件

\*\*/

**protected void** loadBeanDefinitions(XmlBeanDefinitionReader reader) **throws** BeansException, IOException {

// 该出的config文件的设置在ClassPathXmlApplicationContext中

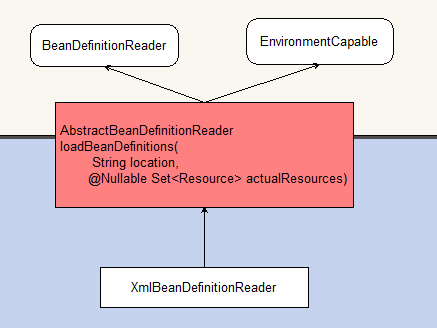
// setConfigLocations(configLocations); 设置  
 Resource[] configResources = getConfigResources();  
 **if** (configResources != **null**) {  
 reader.loadBeanDefinitions(configResources);  
 }

// 如果xml的config资源存在，那么就调用XmlBeanDefinitionReader 的loadBeanDefinitions方法进行处理  
 String[] configLocations = getConfigLocations();  
 **if** (configLocations != **null**) {  
  **reader.loadBeanDefinitions(configLocations);** }  
}

**6 loadBeanDefinitions(String location, @Nullable Set<Resource> actualResources)**

**abstract class AbstractBeanDefinitionReader implements** BeanDefinitionReader, EnvironmentCapable

**public int** loadBeanDefinitions(String location, **@Nullable** Set<Resource> actualResources) **throws** BeanDefinitionStoreException {  
 ResourceLoader resourceLoader = getResourceLoader();  
 **if** (resourceLoader == **null**) {  
 **throw new** BeanDefinitionStoreException(  
 **"Cannot load bean definitions from location ["** + location + **"]: no ResourceLoader available"**);  
 }  
  
 **if** (resourceLoader **instanceof** ResourcePatternResolver) {  
 *// Resource pattern matching available.* **try** {  
 Resource[] resources = ((ResourcePatternResolver) resourceLoader).getResources(location);  
 **int** count = loadBeanDefinitions(resources);  
 **if** (actualResources != **null**) {  
 Collections.*addAll*(actualResources, resources);  
 }  
 **if** (**logger**.isTraceEnabled()) {  
 **logger**.trace(**"Loaded "** + count + **" bean definitions from location pattern ["** + location + **"]"**);  
 }  
 **return** count;  
 }  
 **catch** (IOException ex) {  
 **throw new** BeanDefinitionStoreException(  
 **"Could not resolve bean definition resource pattern ["** + location + **"]"**, ex);  
 }  
 }  
 **else** {  
 *// Can only load single resources by absolute URL.* Resource resource = resourceLoader.getResource(location);  
 **int** count = loadBeanDefinitions(resource);  
 **if** (actualResources != **null**) {  
 actualResources.add(resource);  
 }  
 **if** (**logger**.isTraceEnabled()) {  
 **logger**.trace(**"Loaded "** + count + **" bean definitions from location ["** + location + **"]"**);  
 }  
 **return** count;  
 }  
}



**7 loadBeanDefinitions(EncodedResource encodedResource)**

**XmlBeanDefinitionReader extends AbstractBeanDefinitionReader**

**public int** loadBeanDefinitions(EncodedResource encodedResource) **throws** BeanDefinitionStoreException {  
 Assert.*notNull*(encodedResource, **"EncodedResource must not be null"**);  
 **if** (**logger**.isTraceEnabled()) {  
 **logger**.trace(**"Loading XML bean definitions from "** + encodedResource);  
 }  
  
 Set<EncodedResource> currentResources = **this**.**resourcesCurrentlyBeingLoaded**.get();  
 **if** (currentResources == **null**) {  
 currentResources = **new** HashSet<>(4);  
 **this**.**resourcesCurrentlyBeingLoaded**.set(currentResources);  
 }  
 **if** (!currentResources.add(encodedResource)) {  
 **throw new** BeanDefinitionStoreException(  
 **"Detected cyclic loading of "** + encodedResource + **" - check your import definitions!"**);  
 }  
 **try** {

// 读取xml，将其转换成inputStream流  
 InputStream inputStream = encodedResource.getResource().getInputStream();  
 **try** {  
 InputSource inputSource = **new** InputSource(inputStream);  
 **if** (encodedResource.getEncoding() != **null**) {  
 inputSource.setEncoding(encodedResource.getEncoding());  
 }

// 核心加载方法，xml转成**BeanDefinitions的核心代码**  
 **return doLoadBeanDefinitions(inputSource, encodedResource.getResource());** }  
 **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();  
 }  
 }  
}

**8 dBeanDefinitions(InputSource inputSource, Resource resource)**

**protected int** doLoadBeanDefinitions(InputSource inputSource, Resource resource) **throws** BeanDefinitionStoreException {  
 **try** {

// 解析xml，doLoadDocument是处理bean规则，然后得到Document对象  
  **Document doc = doLoadDocument(inputSource, resource);**

// 将Document中解析的bean和依赖生成BeanDefinition，并将其

// 缓存到内存中，一般使用的是ConcurrentHashMap  
  **int count = registerBeanDefinitions(doc, resource);** **return** count;  
 }  
 **catch** (BeanDefinitionStoreException ex) {  
 **throw** ex;  
 }  
 }

**9 registerBeanDefinitions(Document doc, Resource resource)**

/\*\*

将xml解析的document对象解析成对应的bean封装类b**eanDefinitions**

\*\*/

**public int** registerBeanDefinitions(Document doc, Resource resource) **throws** BeanDefinitionStoreException {

// 实例化BeanDefinitionDocumentReader  
 BeanDefinitionDocumentReader documentReader = createBeanDefinitionDocumentReader();

**// BeanDefinitionRegistry获取已经注册bean的数量**  
 **int** countBefore = getRegistry().getBeanDefinitionCount();

// 处理bean和xml  
 **documentReader.registerBeanDefinitions(doc, createReaderContext(resource));** **return** getRegistry().getBeanDefinitionCount() - countBefore;  
}

**10 registerBeanDefinitions()**

**class DefaultBeanDefinitionDocumentReader implements BeanDefinitionDocumentReader**

/\*\*

解析xml和bean

\*\*/

**public void** registerBeanDefinitions(Document doc, XmlReaderContext readerContext) {  
 **this**.**readerContext** = readerContext;

// 解析dom元素  
 **doRegisterBeanDefinitions(doc.getDocumentElement());**  
}