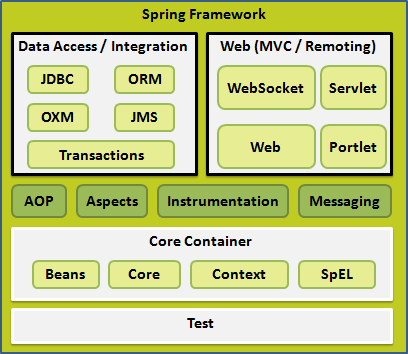
# Spring概述



Spring的核心模块：

**Spring-core**：spring框架的基础组成，包括IOC和依赖注入

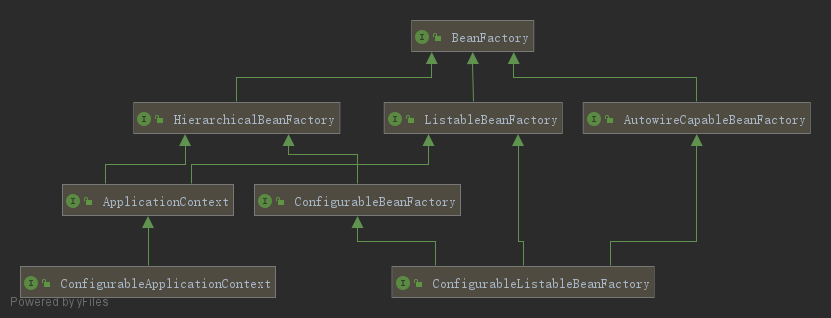
**Spring-beans**：工厂模式，提供了BeanFactory管理bean

**Context**：spring运行时容器，上下文配置，对第三方包的支持

**Spring-expression**：spring表达式

## Spring IOC 容器

IOC 即控制反转又称DI（依赖注入），负责管理对象（bean）的生命周期以及对象之间的关系。



**ListableBeanFactory：**提供了获取多个bean的接口，BeanFactory提供的是获取单个bean方法

**HierarchicalBeanFactory：**实现了bean的分层

**AutowireCapableBeanFactory：**自动装载bean

## 实例化ApplicationContext

Spring通过 ClassPathXmlApplicationContext 或 ClassPathXmlApplicationContext 来加载配置文件，连两个类的核心方法一样。

ClassPathXmlApplicationContext的构造方法，所有含参数的构造方法最终都是调用该构造方

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

// 根据提供的路径，处理成配置文件数组  
 this.setConfigLocations(configLocations);  
 if (refresh) {  
 this.refresh();  
 }

**refresh()（核心方法）**

// **AbstractApplicationContext.java**

public void refresh() throws BeansException, IllegalStateException {

// 加锁，防止refresh()未结束时，再次启动或者销毁  
 synchronized(this.startupShutdownMonitor) {

// 1.记录容器的启动，标记状态为“已启动”  
 this.prepareRefresh();

// 2.将配置文件解析为Bean，并注册到BeanFactory中，

// 只是将信息提取出来，保存到了注册中心，还未初始化  
 ConfigurableListableBeanFactory beanFactory = this.obtainFreshBeanFactory();

// 3. 设置BeanFactory的类加载器，添加BeanPostProcessor,  
 this.prepareBeanFactory(beanFactory);  
 try {

/\*\*

\* Bean如果实现了BeanFactoryPostProcessor接口，

\* 在容器初始化以后，Spring会调用postProcessFactory方法

\*/

// 4. Bean已经完成加载注册，给子类提供扩展  
 this.postProcessBeanFactory(beanFactory);

// 5. 调用BeanFactoryPostProcess实现类的

// postprocessBeanFactory方法  
 this.invokeBeanFactoryPostProcessors(beanFactory);

// 6. 注册BeanPostProcesser的实现类  
 this.registerBeanPostProcessors(beanFactory);

// 7. 国际化，初始化ApplicationContext的MessageSource  
 this.initMessageSource();

// 8. 初始化当前ApplicationContext的事件广播器  
 this.initApplicationEventMulticaster();

// 9. 在初始化singleton bean之前初始化一些特殊的bean  
 this.onRefresh();

// 10. 注册事件监听器（需要实现ApplicationListener接口）  
 this.registerListeners();

// 11. 初始化所有的singleton bean(lazy-init的除外)  
 this.finishBeanFactoryInitialization(beanFactory);

// 12. 广播事件，初始化完成  
 this.finishRefresh();  
 } catch (BeansException var9) {  
 if (this.logger.isWarnEnabled()) {  
 this.logger.warn("Exception encountered during context initialization - cancelling refresh attempt: " + var9);  
 }

// 13. 销毁初始化的singleton的bean，防止资源占用  
 this.destroyBeans();  
 this.cancelRefresh(var9);  
 throw var9;  
 } finally {

// 重置内省缓存  
 this.resetCommonCaches();  
 }  
  
 }  
}

### 刷新beanFactory之前的准备

#### prepareRefresh()

// **AbstractApplicationContext.java**

protected void prepareRefresh() {

// 记录启动时间，设置属性  
 this.startupDate = System.currentTimeMillis();  
 this.closed.set(false);  
 this.active.set(true);  
 if (this.logger.isDebugEnabled()) {  
 if (this.logger.isTraceEnabled()) {  
 this.logger.trace("Refreshing " + this);  
 } else {  
 this.logger.debug("Refreshing " + this.getDisplayName());  
 }  
 }

// 初始化spring上下文的占位符资源  
 this.initPropertySources();

// 校验xml文件  
 this.getEnvironment().validateRequiredProperties();  
 if (this.earlyApplicationListeners == null) {  
 this.earlyApplicationListeners = new LinkedHashSet(this.applicationListeners);  
 } else {  
 this.applicationListeners.clear();  
 this.applicationListeners.addAll(this.earlyApplicationListeners);  
 }  
  
 this.earlyApplicationEvents = new LinkedHashSet();  
}

### 更新beanFactory并加载bean到beanFactory中：

#### configurableListableBeanFactory()

// AbstractApplicationContext.java

protected ConfigurableListableBeanFactory obtainFreshBeanFactory() {  
 this.refreshBeanFactory();  
 return this.getBeanFactory();  
}

// AbstractRefreshableApplicationContext.java

protected final void refreshBeanFactory() throws BeansException {

// ApplictionContext 中加载过BeanFactory，销毁所有bean，

// 关闭BeanFactory  
 if (this.hasBeanFactory()) {  
 this.destroyBeans();  
 this.closeBeanFactory();  
 }  
 try {

// 初始化 DefaultListableBeanFactory  
 DefaultListableBeanFactory beanFactory = this.createBeanFactory();  
 beanFactory.setSerializationId(this.getId());

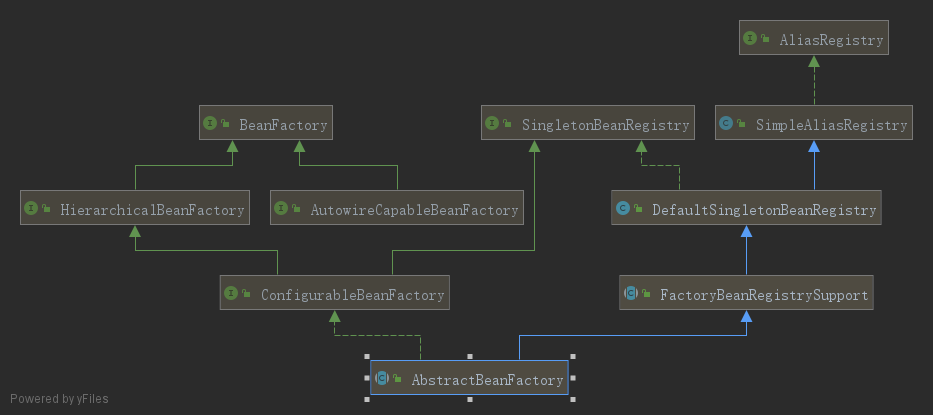
// 设置bean是否允许覆盖,是否允许循环引用  
 this.customizeBeanFactory(beanFactory);

// 加载bean到BeanFactory中  
 this.loadBeanDefinitions(beanFactory);  
 synchronized(this.beanFactoryMonitor) {  
 this.beanFactory = beanFactory;  
 }  
 } catch (IOException var5) {  
 throw new ApplicationContextException("I/O error parsing bean definition source for " + this.getDisplayName(), var5);  
 }  
}

为什么选择实例化DefaultListableBeanFactory?

public class DefaultListableBeanFactory extends AbstractAutowireCapableBeanFactory implements ConfigurableListableBeanFactory, BeanDefinitionRegistry, Serializable

DefaultListableBeanFactory是ConfigurableListableBeanFactory的唯一实现类，而ConfigurableListableBeanFactory继承了BeanFactory的所有子接口，而且通过继承AbstractAutowireCapableBeanFactory（AbstractBeanFactory子类），基本成为功能最全的BeanFactroy



**BeanDefintion**

public interface BeanDefinition extends AttributeAccessor, BeanMetadataElement {

// 默认提供的作用域  
 String SCOPE\_SINGLETON = "singleton";  
 String SCOPE\_PROTOTYPE = "prototype";

// 设置父bean，继承父Bean的配置信息  
 void setParentName(@Nullable String var1);  
 String getParentName();

// 设置/获取bean类的名称，通过反射生成实例  
 void setBeanClassName(@Nullable String var1);  
 @Nullable  
 String getBeanClassName();

// 设置/获取bean的作用域  
 void setScope(@Nullable String var1);  
 @Nullable  
 String getScope();

// 设置/获取懒加载  
 void setLazyInit(boolean var1);  
 boolean isLazyInit();

// 设置/获取所有依赖的bean(depend-on=””设置的值)  
 void setDependsOn(@Nullable String... var1);  
 @Nullable  
 String[] getDependsOn();

// 设置是否可以注入到其他bean中  
 void setAutowireCandidate(boolean var1);  
 boolean isAutowireCandidate();

// 同一接口的多实现，不指定名字时，spring会优选选择primary为

// true的

void setPrimary(boolean var1);  
 boolean isPrimary();

// 如果bean采用工厂方法生成，指定工厂名称  
 void setFactoryBeanName(@Nullable String var1);  
 @Nullable  
 String getFactoryBeanName();

// 设置工厂方法名称  
 void setFactoryMethodName(@Nullable String var1);  
 @Nullable  
 String getFactoryMethodName();

// 构造器参数  
 ConstructorArgumentValues getConstructorArgumentValues();  
 default boolean hasConstructorArgumentValues() {  
 return !this.getConstructorArgumentValues().isEmpty();  
 }  
  
 MutablePropertyValues getPropertyValues();  
 void setDescription(@Nullable String var1);

// ……… //  
}

#### 加载bean:loadBeanDefinitions(DefaultListableBeanFactory beanFactory);

// AbstractXmlApplicationContext

protected void loadBeanDefinitions(DefaultListableBeanFactory beanFactory) throws BeansException, IOException {

// 给beanFactory实例化一个XmlRe  
 XmlBeanDefinitionReader beanDefinitionReader = new XmlBeanDefinitionReader(beanFactory);  
 beanDefinitionReader.setEnvironment(this.getEnvironment());  
 beanDefinitionReader.setResourceLoader(this);  
 beanDefinitionReader.setEntityResolver(new ResourceEntityResolver(this));  
 this.initBeanDefinitionReader(beanDefinitionReader);

// 加载bean  
 this.loadBeanDefinitions(beanDefinitionReader);  
}

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

// 分支1  
 Resource[] configResources = this.getConfigResources();  
 if (configResources != null) {  
 reader.loadBeanDefinitions(configResources);  
 }

// 分支2  
 String[] configLocations = this.getConfigLocations();  
 if (configLocations != null) {  
 reader.loadBeanDefinitions(configLocations);  
 }  
}

// 分支1和分支2最终都会走到该方法

public int loadBeanDefinitions(Resource... resources) throws BeanDefinitionStoreException {  
 Assert.notNull(resources, "Resource array must not be null");  
 int count = 0;  
 Resource[] var3 = resources;  
 int var4 = resources.length;  
  
 for(int var5 = 0; var5 < var4; ++var5) {  
 Resource resource = var3[var5];  
 count += this.loadBeanDefinitions((Resource)resource);  
 }  
 return count;  
}

// XmlBeanDefinitionReader.java

public int loadBeanDefinitions(EncodedResource encodedResource) {  
 // 把配置文件资源放在ThrealLocal中：resourcesCurrentlyBeingLoaded  
 Set<EncodedResource> currentResources = (Set)this.resourcesCurrentlyBeingLoaded.get();  
 if (currentResources == null) {  
 currentResources = new HashSet(4);  
 this.resourcesCurrentlyBeingLoaded.set(currentResources);  
 }  
 if (!((Set)currentResources).add(encodedResource)) {  
 …………  
 } else {  
 int var5;  
 try {  
 InputStream inputStream = encodedResource.getResource().getInputStream();  
 try {  
 InputSource inputSource = new InputSource(inputStream);  
 if (encodedResource.getEncoding() != null) {  
 inputSource.setEncoding(encodedResource.getEncoding());  
 }

// 核心  
 var5 = this.doLoadBeanDefinitions(inputSource, encodedResource.getResource());  
 } finally {  
 inputStream.close();  
 }  
 } catch (IOException var15) {  
 …………  
 } finally {  
 ((Set)currentResources).remove(encodedResource);  
 if (((Set)currentResources).isEmpty()) {  
 this.resourcesCurrentlyBeingLoaded.remove();  
 }  
 }  
 return var5;  
 }  
}

// XmlBeanDefinitionReader.java

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

// 将xml文件转换为Document对象  
 Document doc = this.doLoadDocument(inputSource, resource);

// 注册bean并记录数量  
 int count = this.registerBeanDefinitions(doc, resource);  
 if (this.logger.isDebugEnabled()) {  
 this.logger.debug("Loaded " + count + " bean definitions from " + resource);  
 }  
 return count;  
 } catch (BeanDefinitionStoreException var5) {  
 throw var5;  
 } catch (SAXParseException var6) {  
 …………;  
 }  
}

#### 1.2.2.3 注册bean：resgiterBeanDefinitions (Resource resource)

// DefaultBeanDefinitionDocumentReader.java

public void registerBeanDefinitions(Document doc, XmlReaderContext readerContext) {  
 this.readerContext = readerContext;  
 this.doRegisterBeanDefinitions(doc.getDocumentElement());  
}

public int registerBeanDefinitions(Document doc, Resource resource) throws BeanDefinitionStoreException {  
 BeanDefinitionDocumentReader documentReader =

this.createBeanDefinitionDocumentReader();

// 统计之前加载了多少了个bean  
 int countBefore = this.getRegistry().getBeanDefinitionCount();  
 documentReader.registerBeanDefinitions(doc, this.createReaderContext(resource));  
 return this.getRegistry().getBeanDefinitionCount() - countBefore;  
}

public void registerBeanDefinitions(Document doc, XmlReaderContext readerContext) {  
 this.readerContext = readerContext;  
 this.doRegisterBeanDefinitions(doc.getDocumentElement());  
}

#### 1.2.2.4 注册bean：doRegisterBeanDefinitions (Element root)

protected void doRegisterBeanDefinitions(Element root) {

// 实例化一个bean的解析器委托类，解析bean的定义

// <beans />的内部是可以定义<beans />，  
 BeanDefinitionParserDelegate parent = this.delegate;  
 this.delegate = this.createDelegate(this.getReaderContext(), root, parent);

// 是否是默认的命名空间

// spring的默认空间xmlns，xmlns:[prefix]

// prefix:命名空间的别名  
 if (this.delegate.isDefaultNamespace(root)) {  
 String profileSpec = root.getAttribute("profile");  
 if (StringUtils.hasText(profileSpec)) {  
 String[] specifiedProfiles = StringUtils.tokenizeToStringArray(profileSpec, ",; ");  
 if (!this.getReaderContext().getEnvironment().acceptsProfiles(specifiedProfiles)) {  
 if (this.logger.isDebugEnabled()) {  
 ………  
 }  
 return;  
 }  
 }  
 }  
 this.preProcessXml(root);

// 将DOM解析为bean  
 this.parseBeanDefinitions(root, this.delegate);  
 this.postProcessXml(root);  
 this.delegate = parent;  
}

#### 1.2.2.5 解析bean：parseBeanDefinitions(Element root,BeanDefinitionPareserDelegate delegate)

// DefaultBeanDefinitionDocumentReader.java

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)) {

// 1. 解析默认命名下的元素

// <impoort/> <alias/> <bean/> <beans/>  
 this.parseDefaultElement(ele, delegate);  
 } else {

// 2. 解析其他命名下的元素,例如<mvc/> <aop/>等  
 delegate.parseCustomElement(ele);  
 }  
 }  
 }  
 } else {  
 delegate.parseCustomElement(root);  
 }  
}

// 1. 解析默认命名下的元素

private void parseDefaultElement(Element ele, BeanDefinitionParserDelegate delegate) {  
 if (delegate.nodeNameEquals(ele, "import")) {

// 处理<import/>  
 this.importBeanDefinitionResource(ele);  
 } else if (delegate.nodeNameEquals(ele, "alias")) {

// 处理<alias/>  
 this.processAliasRegistration(ele);  
 } else if (delegate.nodeNameEquals(ele, "bean")) {

// 处理<bean/>  
 this.processBeanDefinition(ele, delegate);  
 } else if (delegate.nodeNameEquals(ele, "beans")) {

// 处理<beans/>  
 this.doRegisterBeanDefinitions(ele);  
 }  
}

#### 1.2.2.6 处理配置文件中的bean：processBeanDefinition(Element ele, BeanDefinitionParserDelegate delegate)

// DefaultBeanDefinitionDocumentReader.java

protected void processBeanDefinition(Element ele, BeanDefinitionParserDelegate delegate) {

// 1. 获取bean的信息封装进BeanDefinitionHolder  
 BeanDefinitionHolder bdHolder = delegate.parseBeanDefinitionElement(ele);  
 if (bdHolder != null) {

// 2. 解析自定义的属性  
 bdHolder = delegate.decorateBeanDefinitionIfRequired(ele, bdHolder);  
 try {

// 3. 注册bean  
 BeanDefinitionReaderUtils.registerBeanDefinition(bdHolder, this.getReaderContext().getRegistry());  
 } catch (BeanDefinitionStoreException var5) {  
 this.getReaderContext().error("Failed to register bean definition with name '" + bdHolder.getBeanName() + "'", ele, var5);  
 }

// 4. 发送注册事件  
 this.getReaderContext().fireComponentRegistered(new BeanComponentDefinition(bdHolder));  
 }  
}

public BeanDefinitionHolder parseBeanDefinitionElement(Element ele) {  
 return this.parseBeanDefinitionElement(ele, (BeanDefinition)null);  
}

public BeanDefinitionHolder parseBeanDefinitionElement(Element ele, @Nullable BeanDefinition containingBean) {  
 String id = ele.getAttribute("id");  
 String nameAttr = ele.getAttribute("name");  
 List<String> aliases = new ArrayList();  
 if (StringUtils.hasLength(nameAttr)) {

// 分割定义的name作为别名  
 String[] nameArr = StringUtils.tokenizeToStringArray(nameAttr, ",; ");  
 aliases.addAll(Arrays.asList(nameArr));  
 }  
 String beanName = id;

// 未指定id，用第一个别名作为beanName  
 if (!StringUtils.hasText(id) && !aliases.isEmpty()) {  
 beanName = (String)aliases.remove(0);  
 if (this.logger.isTraceEnabled()) {  
 this.logger.trace("No XML 'id' specified - using '" + beanName + "' as bean name and " + aliases + " as aliases");  
 }  
 }

// 检查beanName有没有重名  
 if (containingBean == null) {  
 this.checkNameUniqueness(beanName, aliases, ele);  
 }

// 根据<bean/>表情内定义的信息创建beanDefinition  
 AbstractBeanDefinition beanDefinition = this.parseBeanDefinitionElement(ele, beanName, containingBean);

// 如果没有定义id和name，回根据全限定类名生成beanName

// com.demo.service.Impl.Demo#i（i从零开始）  
 if (beanDefinition != null) {  
 if (!StringUtils.hasText(beanName)) {  
 try {  
 if (containingBean != null) {  
 beanName = BeanDefinitionReaderUtils.generateBeanName(beanDefinition, this.readerContext.getRegistry(), true);  
 } else {  
 beanName = this.readerContext.generateBeanName(beanDefinition);  
 String beanClassName = beanDefinition.getBeanClassName();  
 if (beanClassName != null && beanName.startsWith(beanClassName) && beanName.length() > beanClassName.length() && !this.readerContext.getRegistry().isBeanNameInUse(beanClassName)) {  
 aliases.add(beanClassName);  
 }  
 }  
  
 if (this.logger.isTraceEnabled()) {  
 this.logger.trace("Neither XML 'id' nor 'name' specified - using generated bean name [" + beanName + "]");  
 }  
 } catch (Exception var9) {  
 this.error(var9.getMessage(), ele);  
 return null;  
 }  
 }  
  
 String[] aliasesArray = StringUtils.toStringArray(aliases);  
 return new BeanDefinitionHolder(beanDefinition, beanName, aliasesArray);  
 } else {  
 return null;  
 }  
}

##### 1.2.2.6.1 实例化BeanDefinition：parseBeanDefinitionElement(Element ele, String beanName, BeanDefinition containingBean)

// BeanDefinitionParserDelegate.class

@Nullable  
public AbstractBeanDefinition parseBeanDefinitionElement(Element ele, String beanName, BeanDefinition containingBean) {

// 根据beanName实例化BeanEntry放入LinkedList<Entry>中   
 this.parseState.push(new BeanEntry(beanName));

// 设置class和parent属性  
 String className = null;  
 if (ele.hasAttribute("class")) {  
 className = ele.getAttribute("class").trim();  
 }  
 String parent = null;  
 if (ele.hasAttribute("parent")) {  
 parent = ele.getAttribute("parent");  
 }  
 try {

// 实例化BeanDefinition  
 AbstractBeanDefinition bd = this.createBeanDefinition(className, parent);

// 设置BeanDefinition的属性，singleton，lazy-init等  
 this.parseBeanDefinitionAttributes(ele, beanName, containingBean, bd);  
 bd.setDescription(DomUtils.getChildElementValueByTagName(ele, "description"));  
 this.parseMetaElements(ele, bd);  
 this.parseLookupOverrideSubElements(ele, bd.getMethodOverrides());  
 this.parseReplacedMethodSubElements(ele, bd.getMethodOverrides());  
 this.parseConstructorArgElements(ele, bd);  
 this.parsePropertyElements(ele, bd);  
 this.parseQualifierElements(ele, bd);  
 bd.setResource(this.readerContext.getResource());  
 bd.setSource(this.extractSource(ele));  
 AbstractBeanDefinition var7 = bd;  
 return var7;  
 } catch (ClassNotFoundException var13) {  
 …………  
 } finally {  
 this.parseState.pop();  
 }  
  
 return null;  
}

protected AbstractBeanDefinition createBeanDefinition(@Nullable String className, @Nullable String parentName) throws ClassNotFoundException {  
 return BeanDefinitionReaderUtils.createBeanDefinition(parentName, className, this.readerContext.getBeanClassLoader());  
}

public static AbstractBeanDefinition createBeanDefinition(  
 @Nullable String parentName, @Nullable String className, @Nullable ClassLoader classLoader) throws ClassNotFoundException {  
  
 GenericBeanDefinition bd = new GenericBeanDefinition();  
 bd.setParentName(parentName);  
 if (className != null) {  
 if (classLoader != null) {  
 bd.setBeanClass(ClassUtils.*forName*(className, classLoader));  
 }  
 else {  
 bd.setBeanClassName(className);  
 }  
 }  
 return bd;  
}

##### 1.2.2.6.3 注册bean：registerBeanDefinition(BeanDefinitionHolder definitionHolder,BeanDefinitonRegistry registry)

public static void registerBeanDefinition(  
 BeanDefinitionHolder definitionHolder, BeanDefinitionRegistry registry) throws BeanDefinitionStoreException {

//把BeanDefinition放入Map中，Map<String, BeanDefinition>  
 String beanName = definitionHolder.getBeanName();  
 registry.registerBeanDefinition(beanName, definitionHolder.getBeanDefinition());

// 注册别名  
 String[] aliases = definitionHolder.getAliases();  
 if (aliases != null) {  
 for (String alias : aliases) {  
 registry.registerAlias(beanName, alias);  
 }  
 }  
}

## 准备bean工厂，以便在上下文中使 PrepareBeanFactory(ConfigurableListableBeanFactory beanFactory)

protected void prepareBeanFactory(ConfigurableListableBeanFactory beanFactory) {  
 // 设置BeanFactory的类加载器

beanFactory.setBeanClassLoader(getClassLoader());

// 表达式解析器  
 beanFactory.setBeanExpressionResolver(new StandardBeanExpressionResolver(beanFactory.getBeanClassLoader()));

// 属性注册器  
 beanFactory.addPropertyEditorRegistrar(new ResourceEditorRegistrar(this, getEnvironment()));

// 实现了Aware接口的beans在初始化的时候，processor负责回调  
 beanFactory.addBeanPostProcessor(new ApplicationContextAwareProcessor(this));

// 取消以下接口的自动注入，在ApplicationContextAwareProcessors中

// 完成了这几个接口的实现  
 beanFactory.ignoreDependencyInterface(EnvironmentAware.class);

beanFactory.ignoreDependencyInterface(EmbeddedValueResolverAware.class);  
 beanFactory.ignoreDependencyInterface(ResourceLoaderAware.class);

beanFactory.ignoreDependencyInterface(ApplicationEventPublisherAware.class);  
 beanFactory.ignoreDependencyInterface(MessageSourceAware.class);

beanFactory.ignoreDependencyInterface(ApplicationContextAware.class);  
  
 // BeanFactory interface not registered as resolvable type in a plain factory.  
 // MessageSource registered (and found for autowiring) as a bean.  
 beanFactory.registerResolvableDependency(BeanFactory.class, beanFactory);  
 beanFactory.registerResolvableDependency(ResourceLoader.class, this);  
 beanFactory.registerResolvableDependency(ApplicationEventPublisher.class, this);  
 beanFactory.registerResolvableDependency(ApplicationContext.class, this);  
  
 // Register early post-processor for detecting inner beans as ApplicationListeners.  
 beanFactory.addBeanPostProcessor(new ApplicationListenerDetector(this));  
  
 // Detect a LoadTimeWeaver and prepare for weaving, if found.  
 if (beanFactory.containsBean(*LOAD\_TIME\_WEAVER\_BEAN\_NAME*)) {  
 beanFactory.addBeanPostProcessor(new LoadTimeWeaverAwareProcessor(beanFactory));  
 // Set a temporary ClassLoader for type matching.  
 beanFactory.setTempClassLoader(new ContextTypeMatchClassLoader(beanFactory.getBeanClassLoader()));  
 }  
  
 // Register default environment beans.  
 if (!beanFactory.containsLocalBean(*ENVIRONMENT\_BEAN\_NAME*)) {  
 beanFactory.registerSingleton(*ENVIRONMENT\_BEAN\_NAME*, getEnvironment());  
 }  
 if (!beanFactory.containsLocalBean(*SYSTEM\_PROPERTIES\_BEAN\_NAME*)) {  
 beanFactory.registerSingleton(*SYSTEM\_PROPERTIES\_BEAN\_NAME*, getEnvironment().getSystemProperties());  
 }  
 if (!beanFactory.containsLocalBean(*SYSTEM\_ENVIRONMENT\_BEAN\_NAME*)) {  
 beanFactory.registerSingleton(*SYSTEM\_ENVIRONMENT\_BEAN\_NAME*, getEnvironment().getSystemEnvironment());  
 }  
}