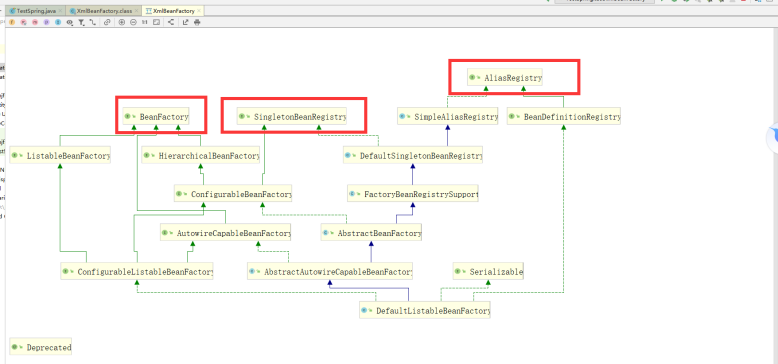
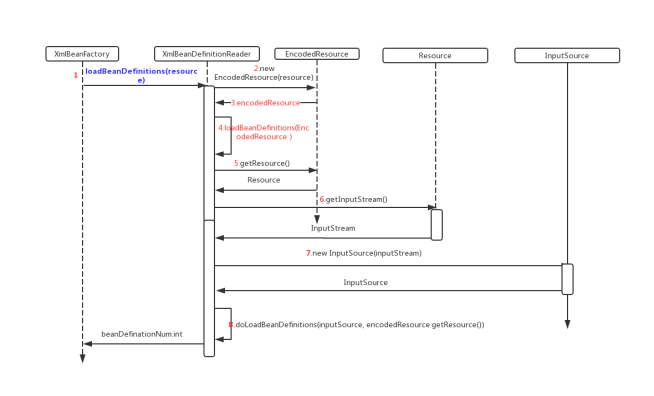
spring源码剖析

# BeanFactory

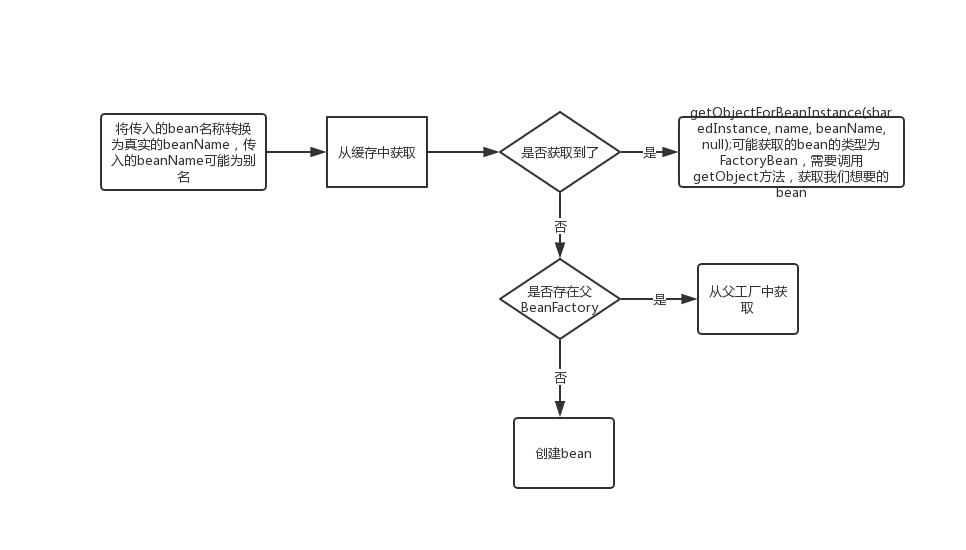
## 类图



## 时序图



## getBean的流程图



# 创建单例bean

## DefaultSingletonBeanRegistry

|  |
| --- |
| public Object getSingleton(String beanName, ObjectFactory<?> singletonFactory) {  Assert.notNull(beanName, "Bean name must not be null");  synchronized (this.singletonObjects) {  Object singletonObject = this.singletonObjects.get(beanName);  if (singletonObject == null) {  if (this.singletonsCurrentlyInDestruction) {  throw new BeanCreationNotAllowedException(beanName,  "Singleton bean creation not allowed while singletons of this factory are in destruction " +  "(Do not request a bean from a BeanFactory in a destroy method implementation!)");  }  if (logger.isDebugEnabled()) {  logger.debug("Creating shared instance of singleton bean '" + beanName + "'");  }  beforeSingletonCreation(beanName);  boolean newSingleton = false;  boolean recordSuppressedExceptions = (this.suppressedExceptions == null);  if (recordSuppressedExceptions) {  this.suppressedExceptions = new LinkedHashSet<>();  }  try {  singletonObject = singletonFactory.getObject();  newSingleton = true;  }  catch (IllegalStateException ex) {  // Has the singleton object implicitly appeared in the meantime ->  // if yes, proceed with it since the exception indicates that state.  singletonObject = this.singletonObjects.get(beanName);  if (singletonObject == null) {  throw ex;  }  }  catch (BeanCreationException ex) {  if (recordSuppressedExceptions) {  for (Exception suppressedException : this.suppressedExceptions) {  ex.addRelatedCause(suppressedException);  }  }  throw ex;  }  finally {  if (recordSuppressedExceptions) {  this.suppressedExceptions = null;  }  afterSingletonCreation(beanName);  }  if (newSingleton) {  addSingleton(beanName, singletonObject);  }  }  return singletonObject;  }  } |



## 存储bean

|  |
| --- |
| ***/\*\*  \* Add the given singleton object to the singleton cache of this factory.  \* <p>To be called for eager registration of singletons.  \* @param beanName the name of the bean  \* @param singletonObject the singleton object  \*/* protected void** addSingleton(**String beanName**, **Object singletonObject**) {  **synchronized** (**this**.**singletonObjects**) {  **this**.**singletonObjects**.**put**(**beanName**, **singletonObject**);  **this**.**singletonFactories**.**remove**(**beanName**);  **this**.**earlySingletonObjects**.**remove**(**beanName**);  **this**.**registeredSingletons**.**add**(**beanName**);  } } |

## 存储单例的map



# AbstractAutowireCapableBeanFactory

## createBean

|  |
| --- |
| ***/\*\*  \* Central method of this class: creates a bean instance,  \* populates the bean instance, applies post-processors, etc.  \* @see #*doCreateBean  *\*/* @Override protected Object** createBean(**String beanName**, **RootBeanDefinition mbd**, **@Nullable Object**[] **args**)  **throws BeanCreationException** {   **if** (**logger**.**isTraceEnabled**()) {  **logger**.**trace**(**"Creating instance of bean '" + beanName + "'"**);  }  **RootBeanDefinition mbdToUse = mbd**;   *// Make sure bean class is actually resolved at this point, and  // clone the bean definition in case of a dynamically resolved Class  // which cannot be stored in the shared merged bean definition.* **Class<?> resolvedClass = resolveBeanClass**(**mbd**, **beanName**);  **if** (**resolvedClass != null && !mbd**.**hasBeanClass**() **&& mbd**.**getBeanClassName**() **!= null**) {  **mbdToUse = new RootBeanDefinition**(**mbd**);  **mbdToUse**.**setBeanClass**(**resolvedClass**);  }   *// Prepare method overrides.* **try** {  **mbdToUse**.**prepareMethodOverrides**();  }  **catch** (**BeanDefinitionValidationException ex**) {  **throw new BeanDefinitionStoreException**(**mbdToUse**.**getResourceDescription**(),  **beanName**, **"Validation of method overrides failed"**, **ex**);  }   **try** {  *// Give BeanPostProcessors a chance to return a proxy instead of the target bean instance.* **Object bean = resolveBeforeInstantiation**(**beanName**, **mbdToUse**);  **if** (**bean != null**) {  **return bean**;  }  }  **catch** (**Throwable ex**) {  **throw new BeanCreationException**(**mbdToUse**.**getResourceDescription**(), **beanName**,  **"BeanPostProcessor before instantiation of bean failed"**, **ex**);  }   **try** {  // 真正的创建bean  **Object beanInstance = doCreateBean(beanName, mbdToUse, args);**  **if** (**logger**.**isTraceEnabled**()) {  **logger**.**trace**(**"Finished creating instance of bean '" + beanName + "'"**);  }  **return beanInstance**;  }  **catch** (**BeanCreationException | ImplicitlyAppearedSingletonException ex**) {  *// A previously detected exception with proper bean creation context already,  // or illegal singleton state to be communicated up to DefaultSingletonBeanRegistry.* **throw ex**;  }  **catch** (**Throwable ex**) {  **throw new BeanCreationException**(  **mbdToUse**.**getResourceDescription**(), **beanName**, **"Unexpected exception during bean creation"**, **ex**);  } } |

## doCreateBean

|  |
| --- |
| ***/\*\*  \* Actually create the specified bean. Pre-creation processing has already happened  \* at this point, e.g. checking {@code postProcessBeforeInstantiation} callbacks.  \* <p>Differentiates between default bean instantiation, use of a  \* factory method, and autowiring a constructor.  \* @param beanName the name of the bean  \* @param mbd the merged bean definition for the bean  \* @param args explicit arguments to use for constructor or factory method invocation  \* @return a new instance of the bean  \* @throws* BeanCreationException *if the bean could not be created  \* @see #*instantiateBean  *\* @see #*instantiateUsingFactoryMethod  *\* @see #*autowireConstructor  *\*/* protected Object** doCreateBean(**final String beanName**, **final RootBeanDefinition mbd**, **final @Nullable Object**[] **args**)  **throws BeanCreationException** {   *// Instantiate the bean.* **BeanWrapper instanceWrapper = null**;  **if** (**mbd**.**isSingleton**()) {  **instanceWrapper = this**.**factoryBeanInstanceCache**.**remove**(**beanName**);  }  **if** (**instanceWrapper == null**) {  **instanceWrapper = createBeanInstance**(**beanName**, **mbd**, **args**);  }  **final Object bean = instanceWrapper**.**getWrappedInstance**();  **Class<?> beanType = instanceWrapper**.**getWrappedClass**();  **if** (**beanType != NullBean**.**class**) {  **mbd**.**resolvedTargetType = beanType**;  }   *// Allow post-processors to modify the merged bean definition.* **synchronized** (**mbd**.**postProcessingLock**) {  **if** (**!mbd**.**postProcessed**) {  **try** {  **applyMergedBeanDefinitionPostProcessors**(**mbd**, **beanType**, **beanName**);  }  **catch** (**Throwable ex**) {  **throw new BeanCreationException**(**mbd**.**getResourceDescription**(), **beanName**,  **"Post-processing of merged bean definition failed"**, **ex**);  }  **mbd**.**postProcessed = true**;  }  }   *// Eagerly cache singletons to be able to resolve circular references  // even when triggered by lifecycle interfaces like BeanFactoryAware.* **boolean earlySingletonExposure =** (**mbd**.**isSingleton**() **&& this**.**allowCircularReferences &&  isSingletonCurrentlyInCreation**(**beanName**));  **if** (**earlySingletonExposure**) {  **if** (**logger**.**isTraceEnabled**()) {  **logger**.**trace**(**"Eagerly caching bean '" + beanName +  "' to allow for resolving potential circular references"**);  }  **// 解决循环依赖问题**  **addSingletonFactory**(**beanName**, () -> **getEarlyBeanReference**(**beanName**, **mbd**, **bean**));  }   *// Initialize the bean instance.* **Object exposedObject = bean**;  **try** {  **// 注入属性**  **populateBean(beanName, mbd, instanceWrapper);**  **// 调用init-method方法（先调用初始化前的后置处理器方法，再调用init-method，再调用初始化后的后置处理器方法）**  **exposedObject = initializeBean**(**beanName**, **exposedObject**, **mbd**);  }  **catch** (**Throwable ex**) {  **if** (**ex instanceof BeanCreationException && beanName**.**equals**(((**BeanCreationException**) **ex**).**getBeanName**())) {  **throw** (**BeanCreationException**) **ex**;  }  **else** {  **throw new BeanCreationException**(  **mbd**.**getResourceDescription**(), **beanName**, **"Initialization of bean failed"**, **ex**);  }  }   **if** (**earlySingletonExposure**) {  **Object earlySingletonReference = getSingleton**(**beanName**, **false**);  **if** (**earlySingletonReference != null**) {  **if** (**exposedObject == bean**) {  **exposedObject = earlySingletonReference**;  }  **else if** (**!this**.**allowRawInjectionDespiteWrapping && hasDependentBean**(**beanName**)) {  **String**[] **dependentBeans = getDependentBeans**(**beanName**);  **Set<String> actualDependentBeans = new LinkedHashSet<>**(**dependentBeans**.**length**);  **for** (**String dependentBean : dependentBeans**) {  **if** (**!removeSingletonIfCreatedForTypeCheckOnly**(**dependentBean**)) {  **actualDependentBeans**.**add**(**dependentBean**);  }  }  **if** (**!actualDependentBeans**.**isEmpty**()) {  **throw new BeanCurrentlyInCreationException**(**beanName**,  **"Bean with name '" + beanName + "' has been injected into other beans [" +  StringUtils**.*collectionToCommaDelimitedString*(**actualDependentBeans**) **+  "] in its raw version as part of a circular reference, but has eventually been " +  "wrapped. This means that said other beans do not use the final version of the " +  "bean. This is often the result of over-eager type matching - consider using " +  "'getBeanNamesOfType' with the 'allowEagerInit' flag turned off, for example."**);  }  }  }  }   *// Register bean as disposable.* **try** {  **registerDisposableBeanIfNecessary**(**beanName**, **bean**, **mbd**);  }  **catch** (**BeanDefinitionValidationException ex**) {  **throw new BeanCreationException**(  **mbd**.**getResourceDescription**(), **beanName**, **"Invalid destruction signature"**, **ex**);  }   **return exposedObject**; } |

# BeanPostProcessor体系

## BeanPostProcessor

### 适用场景

* Springmvc的注解

## InstantiationAwareBeanPostProcessor

### 使用场景

* Aop
* Autowire注解

## SmartInstantiationAwareBeanPostProcessor

### 作用

循环依赖的扩展

## MergedBeanDefinitionPostProcessor

# Aware体系

## BeanFactoryAware

## ApplicationContextAware

BeanNameAware

# Spring集成到web中

## 配置web.xml

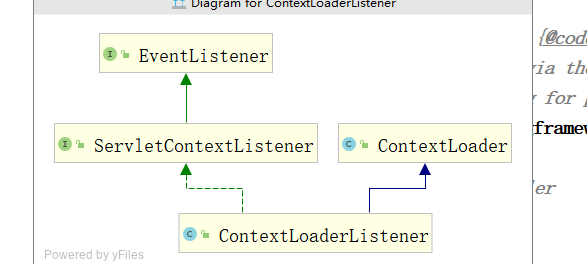
|  |
| --- |
| <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:application.xml</**param-value**> </**context-param**> <**listener**>  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**> </**listener**> |

## ContextLoaderListener

### 作用

用来初始化spring容器

### 类结构

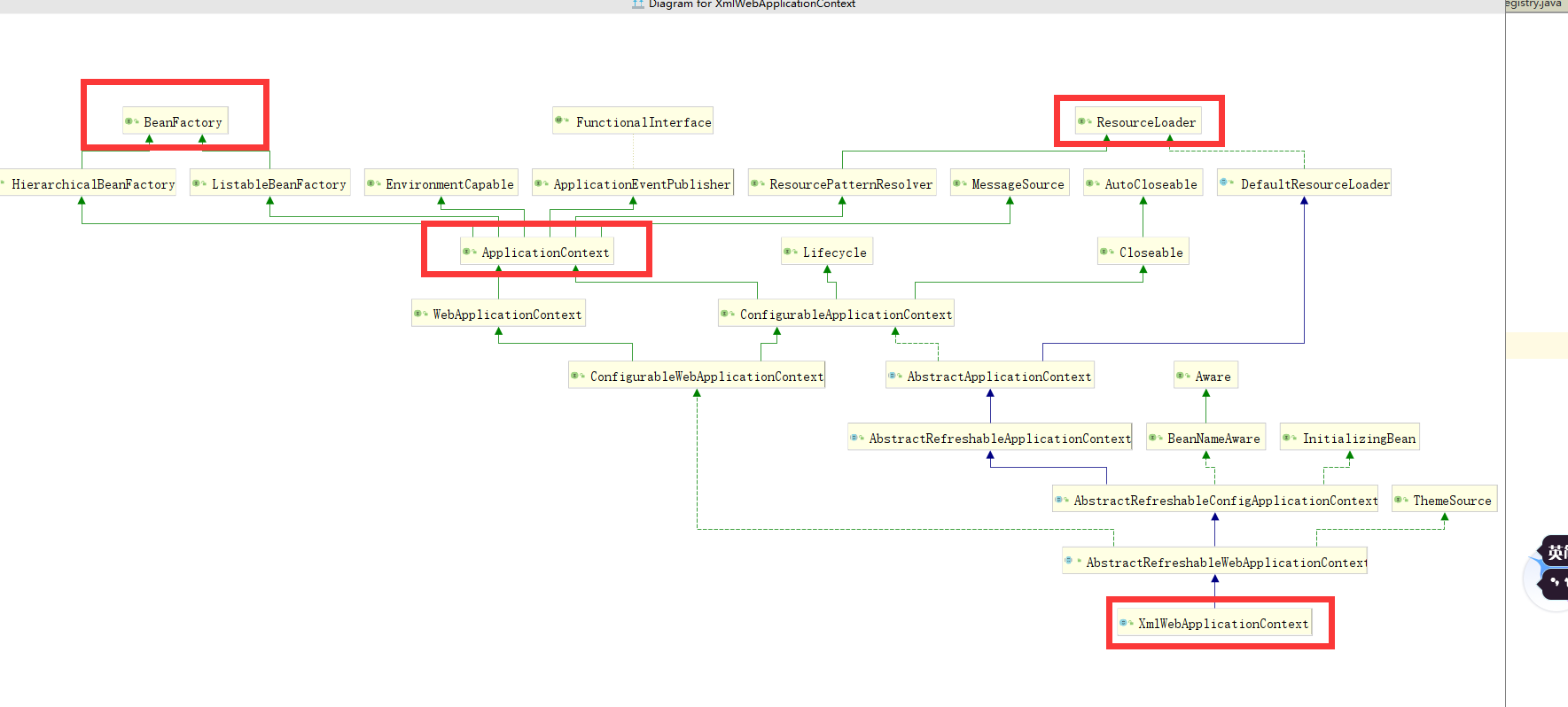


### 核心代码

|  |
| --- |
| ***/\*\*  \* Initialize Spring's web application context for the given servlet context,  \* using the application context provided at construction time, or creating a new one  \* according to the "{@link #CONTEXT\_CLASS\_PARAM contextClass}" and  \* "{@link #CONFIG\_LOCATION\_PARAM contextConfigLocation}" context-params.  \* @param servletContext current servlet context  \* @return the new WebApplicationContext  \* @see #*ContextLoader*(*WebApplicationContext*)  \* @see #CONTEXT\_CLASS\_PARAM  \* @see #CONFIG\_LOCATION\_PARAM  \*/* public WebApplicationContext** initWebApplicationContext(**ServletContext servletContext**) {  **if** (**servletContext**.**getAttribute**(**WebApplicationContext**.***ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE***) **!= null**) {  **throw new IllegalStateException**(  **"Cannot initialize context because there is already a root application context present - " +  "check whether you have multiple ContextLoader\* definitions in your web.xml!"**);  }   **servletContext**.**log**(**"Initializing Spring root WebApplicationContext"**);  **Log logger = LogFactory**.*getLog*(**ContextLoader**.**class**);  **if** (**logger**.**isInfoEnabled**()) {  **logger**.**info**(**"Root WebApplicationContext: initialization started"**);  }  **long startTime = System**.*currentTimeMillis*();   **try** {  *// Store context in local instance variable, to guarantee that  // it is available on ServletContext shutdown.* **if** (**this**.**context == null**) {  // 创建上下文(XmlWebApplicationContext)  **this**.**context = createWebApplicationContext**(**servletContext**);  }  **if** (**this**.**context instanceof ConfigurableWebApplicationContext**) {  **ConfigurableWebApplicationContext cwac =** (**ConfigurableWebApplicationContext**) **this**.**context**;  **if** (**!cwac**.**isActive**()) {  *// The context has not yet been refreshed -> provide services such as  // setting the parent context, setting the application context id, etc* **if** (**cwac**.**getParent**() **== null**) {  *// The context instance was injected without an explicit parent ->  // determine parent for root web application context, if any.* **ApplicationContext parent = loadParentContext**(**servletContext**);  **cwac**.**setParent**(**parent**);  }  // 刷新容器（主要创建bean，还有依赖注入）  **configureAndRefreshWebApplicationContext**(**cwac**, **servletContext**);  }  }  // 将spring容器设置为父容器  **servletContext**.**setAttribute**(**WebApplicationContext**.***ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE***, **this**.**context**);   **ClassLoader ccl = Thread**.*currentThread*().**getContextClassLoader**();  **if** (**ccl == ContextLoader**.**class**.**getClassLoader**()) {  ***currentContext* = this**.**context**;  }  **else if** (**ccl != null**) {  ***currentContextPerThread***.**put**(**ccl**, **this**.**context**);  }   **if** (**logger**.**isInfoEnabled**()) {  **long elapsedTime = System**.*currentTimeMillis*() **- startTime**;  **logger**.**info**(**"Root WebApplicationContext initialized in " + elapsedTime + " ms"**);  }   **return this**.**context**;  }  **catch** (**RuntimeException | Error ex**) {  **logger**.**error**(**"Context initialization failed"**, **ex**);  **servletContext**.**setAttribute**(**WebApplicationContext**.***ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE***, **ex**);  **throw ex**;  } } |

## XmlWebApplicationContext

### 类结构

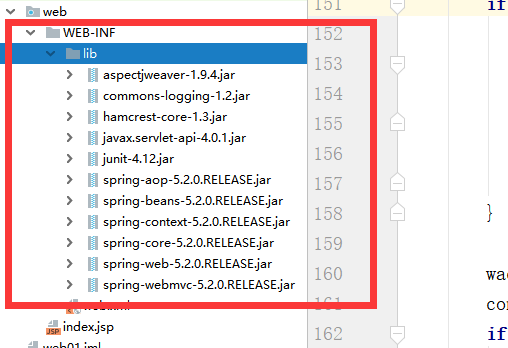


### 核心代码

|  |
| --- |
| ***/\*\*  \* 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**.**setEnvironment**(**getEnvironment**());  **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**);  // 解析bean（将bean从xml中解析出来，封装到BeanDefination中）  **loadBeanDefinitions**(**beanDefinitionReader**); } |

## 注意

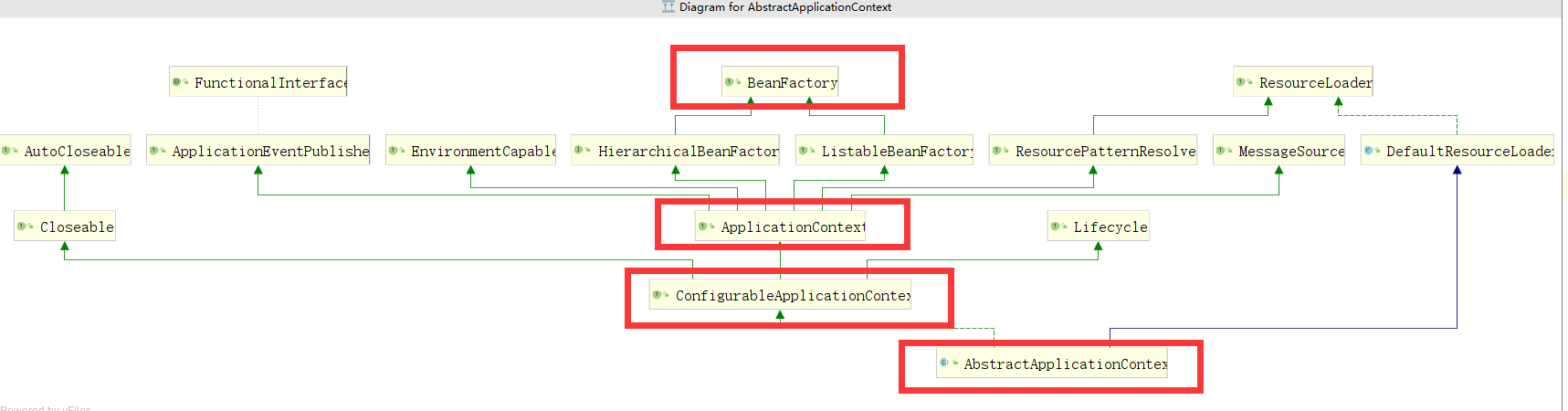
如果不是maven项目，一定要将jar包放到WEB-INF下面



# ApplicationContext家族

## AbstractApplicationContext

### 结构



### 核心代码

|  |
| --- |
| ***/\*\*  \* Return the list of statically specified ApplicationListeners.  \*/* public Collection<ApplicationListener<?>>** getApplicationListeners() {  **return this**.**applicationListeners**; }  **@Override 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**) {  **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**();  }  } } |