Spring IoC初始化源码分析

IoC(控制反转)，主要完成对象的创建和依赖的注入，由Spring IoC容器来实现，下面是Spring IoC的代码入口，spring.xml的配置如下：

*<bean id="teacher" class="com.fys.spring.basic.Teacher">*

*<property name="name" value="Jerry"/>*

*</bean>*

编写程序入口代码如下：

*FileSystemXmlApplicationContext context =*

*new FileSystemXmlApplicationContext("spring.xml");*

*Teacher teacher = (Teacher) context.getBean("teacher");*

# 启动过程分析

主程序是从FileSystemXmlApplicationContext的构造方法开始，

*public FileSystemXmlApplicationContext( String[] configLocations, boolean refresh,*

*@Nullable ApplicationContext parent)  
 super(parent);  
 setConfigLocations(configLocations);  
 if (refresh) {  
 refresh();  
 }  
}*

其核心方法是refresh，ApplicationContext通过refresh进行多次初始化操作，其核心如下：

*@Override*

*public void refresh() throws BeansException, IllegalStateException {*

*synchronized (this.startupShutdownMonitor) {*

*//前期准备工作，记录容器启动时间、标记已启动状态、处理配置文件的占位符*

*prepareRefresh();*

*//核心方法，将配置文件解析成Bean定义，注册到BeanFactory中，*

*在这一步Bean还没有初始化*

*ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();*

*// 设置BeanFactory的类加载器*

*prepareBeanFactory(beanFactory);*

*try {*

*// 添加Bean初始化涉及到的处理类，*

*包括BeanFactoryPostProcessor/BeanPostProcessor等*

*postProcessBeanFactory(beanFactory);*

*invokeBeanFactoryPostProcessors(beanFactory);*

*registerBeanPostProcessors(beanFactory);*

*//添加MessageSource、事件广播器、事件监听等*

*initMessageSource();*

*initApplicationEventMulticaster();*

*onRefresh();*

*registerListeners();*

*// 初始化所有的Singleton Beans*

*finishBeanFactoryInitialization(beanFactory);*

*// 收尾工作*

*inishRefresh();*

*.......}*

这个方法包含整个容器初始化过程，整个过程比较复杂，这里只分析两个阶段：

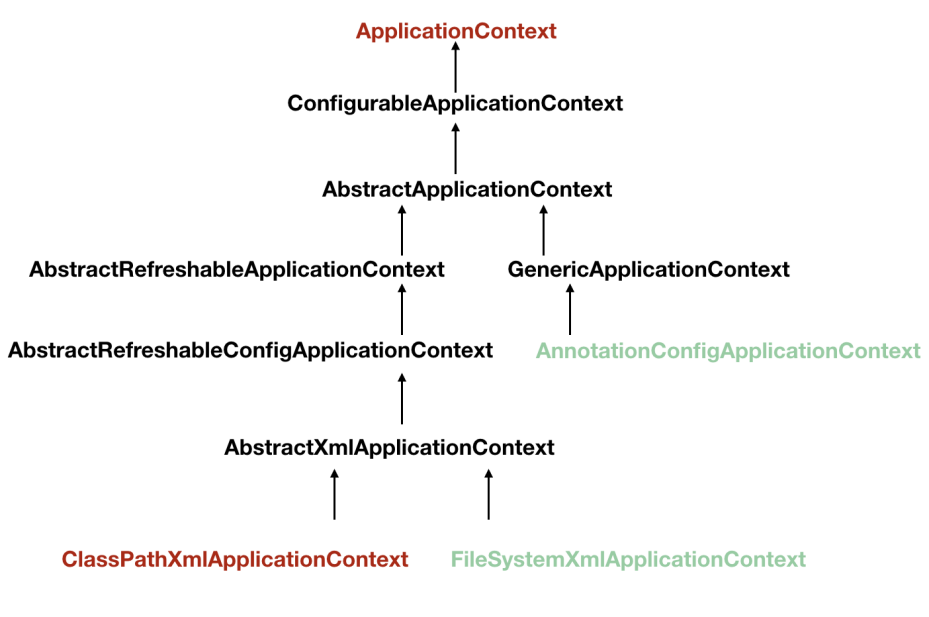
* *obtainFreshBeanFactory*，将配置文件解析为Bean定义，并注册到BeanFactory中
* *finishBeanFactoryInitialization*，初始化所有的Singleton Beans

# 核心概念及类

Spring IoC容器的初始化过程的核心是将配置文件解析为Bean定义，初始化Bean，并注册到BeanFactory中

## **ApplicationContext**

从示例中可以看出ApplicationContext封装了Spring IoC的管理，包括资源文件的定位、Bean的载入及注册三个基本过程，其底层接口为ApplicationContext，继承体系如下图所示：



Spring为了适应不同的使用场景，提供多个实现类，其中核心：ClassPathXmlApplication和FileSystemApplicationContext，基于XML文件配置；AnnotationConfigApplicationContext，基于注解来使用，不需要配置文件，采用Java配置类和注解来配置，是目前使用最广的方式。在第一部分已经分析了FileSystemApplicationContext的启动过程，Context核心在于Resource的不同，即将配置文件或者注解解析成Bean的定义。

## BeanDefiniton

在配置中定义的Bean会转换成BeanDefiniton存在Spring的BeanFactory中，Bean在代码层面上是BeanDefinition的实例，其保存了Bean指向的类、是否单例、是否懒加载及依赖关系等，其接口定义如下：



核心属性：

* SCOPE\_SINGLETON和SCOPE\_PROTOTYPE，默认提供Sington和ProtoType两种类型
* parentName: 父Bean，涉及到Bean继承
* beanClassName: bean的类名称
* Scope，bean的scope
* lazyInit，是否懒加载
* dependsOn，依赖的Bean
* autowireCandidate，设置该Bean是否可以注入到其他Bean中
* factoryBeanName，如果bean使用工厂方法生成，指定工厂名称
* constructorArgumentValues，构造器参数

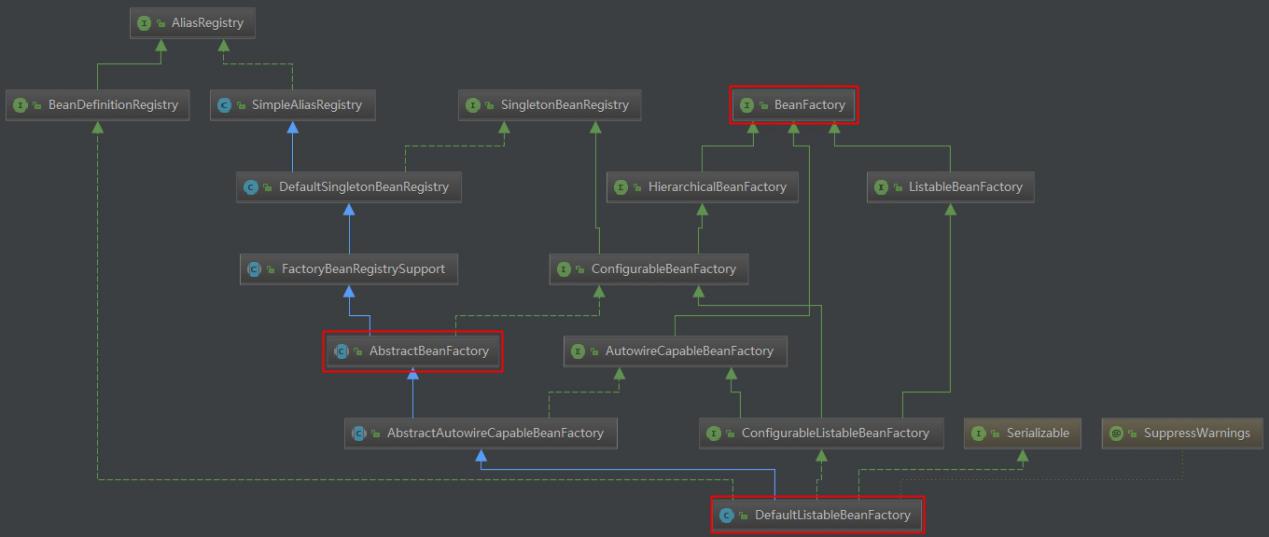
在BeanDefinition中包含很多信息。

## **BeanFactory**

BeanFactory是实现IoC机制的核心，实例的加载、创建和初始化过程都是在BeanFactory中完成，在接口中定义了各种方法：

* getBean，获取BeanFactory中存储的对象实例
* containersBean，判断BeanFactory中是否包含对应对象
* isSingleton，判断是否单例
* 其他

BeanFactory的实现类有多种，默认实现类为DefaultListableBeanFactory，如下图所示：



在DefaultListableBeanFactory中其对应属性如类图：



在DefaultListableBeanFactory中维护了两个核心值：

* beanDefinitionMap:Map<String,BeanDefinition>，Bean的定义
* resolvableDependencies:Map<Class<?>,Object>，Bean对应生成的对象Object

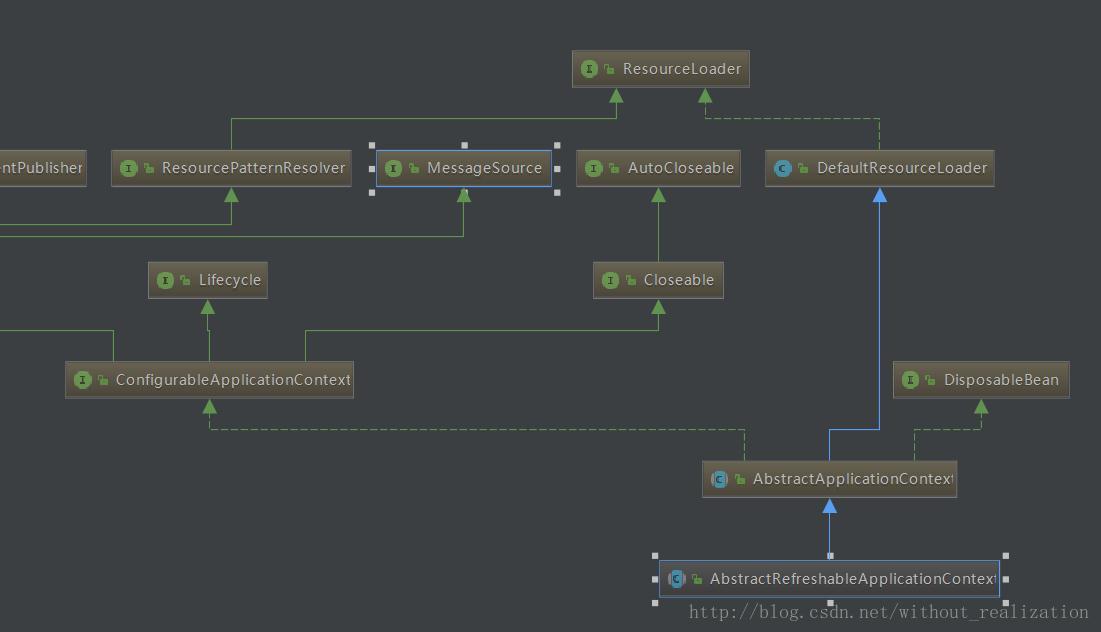
当需要依赖注入时BeanFactory根据以上的定义返回对应的Object

# Spring IoC容器初始化流程

Spring IoC容器的加载，大体上经过：资源文件定位、解析、注册和实例化这几个过程

## **资源文件定位**

在ApplicationContext实现类中完成，通过ResourceLoader#getResource读取外部资源：包括XML配置文件，类中bean定义等。Spring IoC将外部资源中解析为BeanDefinition信息，类图如下：



每一种资源都会对应资源Reader，以XML配置文件为例，其读取类为XmlBeanDefinitionReader，使用如下：

*protected void loadBeanDefinitions(DefaultListableBeanFactory beanFactory) {  
 // Create a new XmlBeanDefinitionReader for the given BeanFactory.  
 XmlBeanDefinitionReader beanDefinitionReader =*

*new XmlBeanDefinitionReader(beanFactory);  
 beanDefinitionReader.setEnvironment(this.getEnvironment());  
 beanDefinitionReader.setResourceLoader(this);  
 beanDefinitionReader.setEntityResolver(new ResourceEntityResolver(this));  
 initBeanDefinitionReader(beanDefinitionReader);  
 loadBeanDefinitions(beanDefinitionReader); //从Resource中读取BeanDefinition  
}*

## **加载**

读取配置的操作在XmlBeanDefinitionReader中完成，负责加载配置及解析，核心方法:

*loadBeanDefinitions*

其中的核心是读取XML文件，解析Document，并注册到BeanFactory中

*protected int doLoadBeanDefinitions(InputSource inputSource, Resource resource)*

*throws BeanDefinitionStoreException {*

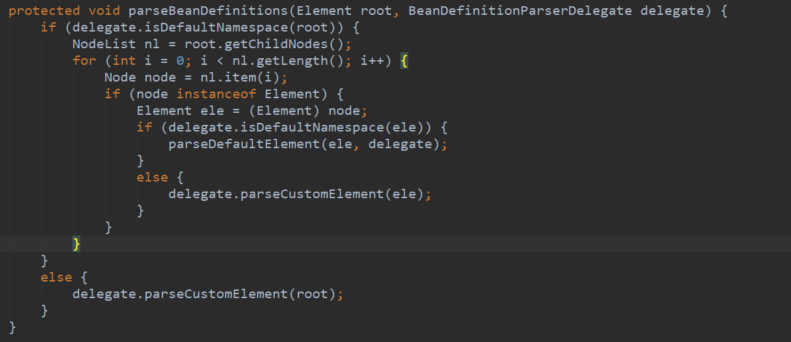
*Document doc = doLoadDocument(inputSource, resource);*

*int count = registerBeanDefinitions(doc, resource);*

*..*

*}*

对于Document的处理，DefaultBeanDefinitionDocumentReader如下：



解析其中的配置中的Bean标签，获取属性值，Bean标签的属性包括：



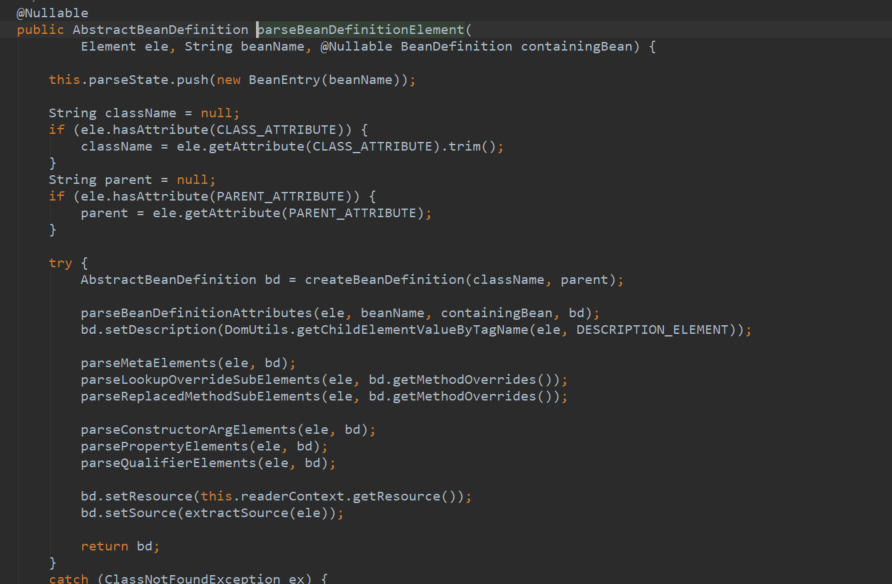
简单的定义：

*<bean id="teacher" class="com.fys.spring.basic.Teacher">*

*<property name="name" value="Jerry"/>*

*</bean>*

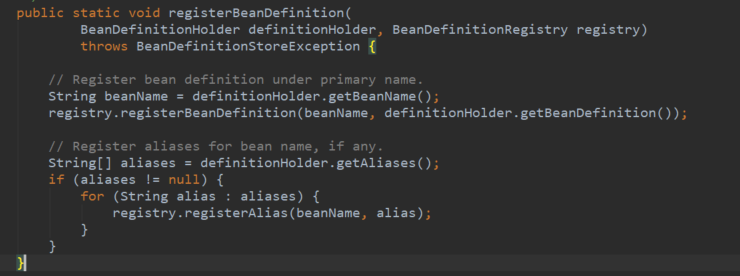
最终的调用如下：



获取BeanDefinition

## **注册Bean**

从配置中解析BeanDefinition后，注册Bean到BeanFactory中，BeanDefinitionReaderUitls执行如下：



BeanDefinitionRegistry是接口，BeanFactory继承该接口，其DefaultListableBeanFactory中执行如下：

*@Override*

*public void registerBeanDefinition(String beanName, BeanDefinition beanDefinition)*

*throws BeanDefinitionStoreException {*

*.....*

*this.beanDefinitionMap.put(beanName, beanDefinition);*

*this.beanDefinitionNames.add(beanName);*

*this.manualSingletonNames.remove(beanName);....}*

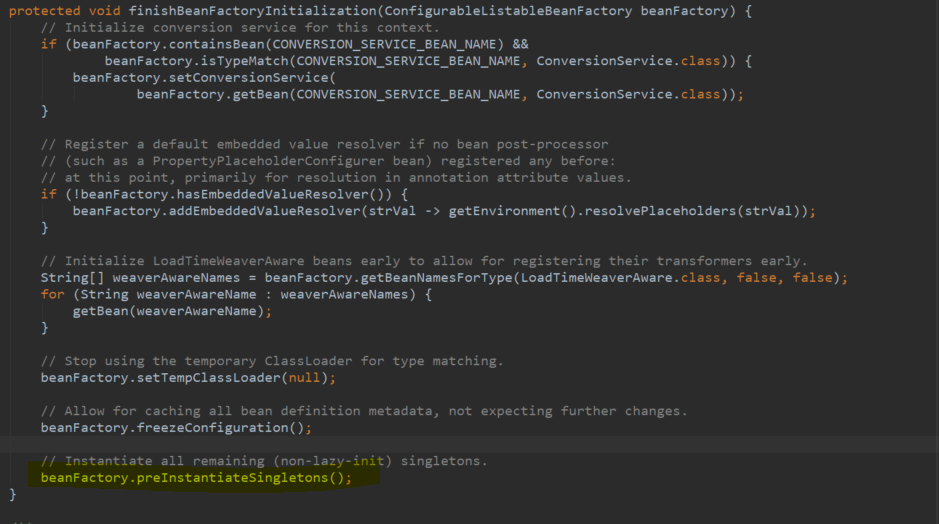
至此BeanDefinition注册到BeanFactory中，beanDefinitionMap:Map<String,BeanDefinition>

## **Bean实例化**

在前文中分析了其启动流程，最后一个步骤也是最重要的步骤是

*finishBeanFactoryInitialization(beanFactory);*

根据BeanDefinition对Bean进行实例化，负载初始化所有的Singleton Beans，执行方法如下：



通过DefaultListableBeanFactory#preInstantiateSingletons，实例化自定义的Bean



getBean的执行核心代码：

*if (mbd.isSingleton()) {  
 sharedInstance = getSingleton(beanName, () -> {  
 try {  
 return createBean(beanName, mbd, args); //beanName,参数等  
 }  
 bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);  
}*

进一步的调用为AbstractAutowireCapableBeanFactor#doCreateBean

*protected Object doCreateBean(final String beanName, final RootBeanDefinition mbd, final @Nullable Object[] args)*

*throws BeanCreationException {*

*// 初始化bean*

*BeanWrapper instanceWrapper = null;*

*..*

*instanceWrapper = createBeanInstance(beanName, mbd, args);*

*final Object bean = instanceWrapper.getWrappedInstance();*

*...*

*return bean;*

*}*

最终的实现是使用Java反射技术，

*// Need to determine the constructor...  
Constructor<?>[] ctors = determineConstructorsFromBeanPostProcessors(beanClass, beanName);  
if (ctors != null ||  
 mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE\_CONSTRUCTOR ||  
 mbd.hasConstructorArgumentValues() || !ObjectUtils.isEmpty(args)) {  
 return autowireConstructor(beanName, mbd, ctors, args); //有参构造函数  
}*

*return instantiateBean(beanName, mbd); //无参构造函数*

获取构造函数，并生成实例。其执行过程需要考虑依赖注入的populateBean等。

所有的Bean存放在BeanRegistry类中，其定义在对象中

*private final Map<String, Object> singletonObjects = new ConcurrentHashMap<>(256);*

http://www.importnew.com/27469.html