Spring源码分析

依赖反转是软件设计中一个重要的思想，工厂模式应用的就是这种思想。解决了对象之间的依赖关系。

spring framework的核心:IOC容器的实现

依赖反转:依赖的对象的获得反转了.

依赖倒置,依赖反转,依赖注入,控制反转,都是一个意思.

在面向对象系统中,对象封装了数据和对数据的处理,对象的依赖关系常常体现在对数据和方法的依赖上.这些依赖关系可以通过把对象的依赖注入交给

框架或IOC容器来完成.解耦代码的同时提高代码的可测试性.

依赖控制反转的实现:在spring中IOC容器是实现这个模式的载体,他可以在对象生成或初始化时直接将数据注入到对象中.也可以通过将对象引用

注入到对象数据域中的方式来注入对方法调用的依赖.这种依赖注入是可以递归的,对象被逐层注入.把对象的依赖关系有序的建立起来,简化了对象依赖关系的管理.简化了面向对象系统的复杂性.

如何反转对依赖的控制,把控制权从具体业务对象手中转交到平台或者框架中,是降低面向对象系统设计复杂性和提高面向对象系统可测试性的一个有效解决方案

他促进了IOC设计模式的发展,是IOC容器要解决的核心问题.

同时,也是产品化的IOC容器出现的推动力

spring提供了统一管理依赖的平台框架,而不是每一个对象都有自己的依赖,每次使用某一个对象时,就要创建一个对象.简单的对象还不算麻烦,当一个对象内部有多个数据域需要初始化,或者依赖的它的对象可能多个,或者存在递归或者深层次的调用,或者需要不同的初始化数据时,这时候对象依赖的管理尤为重要.否则后期将无力维护,并且BUG的查找困难,新需要难以开展.

在多种面向对象语言中,都已经使用了这种开发技巧,也就是设计模式.JAVA,C++等.

通过外部配置文件来提高对组件关系的管理水平,并且如果耦合关系需要变动,并不需要重新修改和编译JAVA源代码,符合面向对象的开闭原则,并且提高提高组件系统设计的灵活性,同时,如果结合OSGI的使用特性,还可以提高应用的动态部署能力.另外,在最新版的jdk中已经提供了模块化编程的支持.

# Spring IOC容器概述

Spring中主要两个容器系列:

1. 实现BeanFactory 接口的简单容器系列。实现了容器的基本功能.
2. ApplicationContext 应用上下文。 容器的高级形态存在.

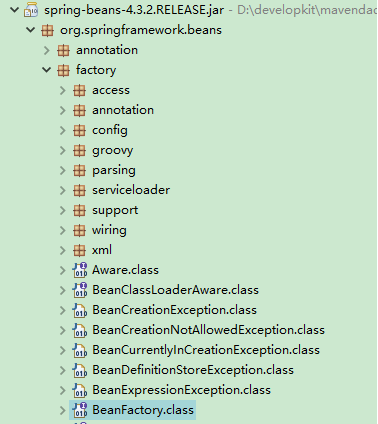
应用上下文在简单容器的基础上,增加了面向框架的特性,同时对应用环境做了许多适配.

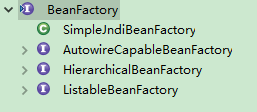
在spring提供的基本IOC容器的接口定义和实现的基础上,spring通过定义BeanDefinition来管理基于spring的应用中的各种对象以及它们的相互依赖关系.

Beandefinition抽象了Bean的定义,是让容器起作用的主要数据类型.

ICO容器是用来管理对象依赖关系的,对IOC容器来说,BeanDefinition就是对依赖反转模式中管理的对象依赖关系的数据抽象.

# BeanFactory结构

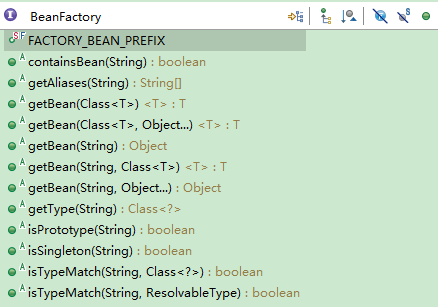




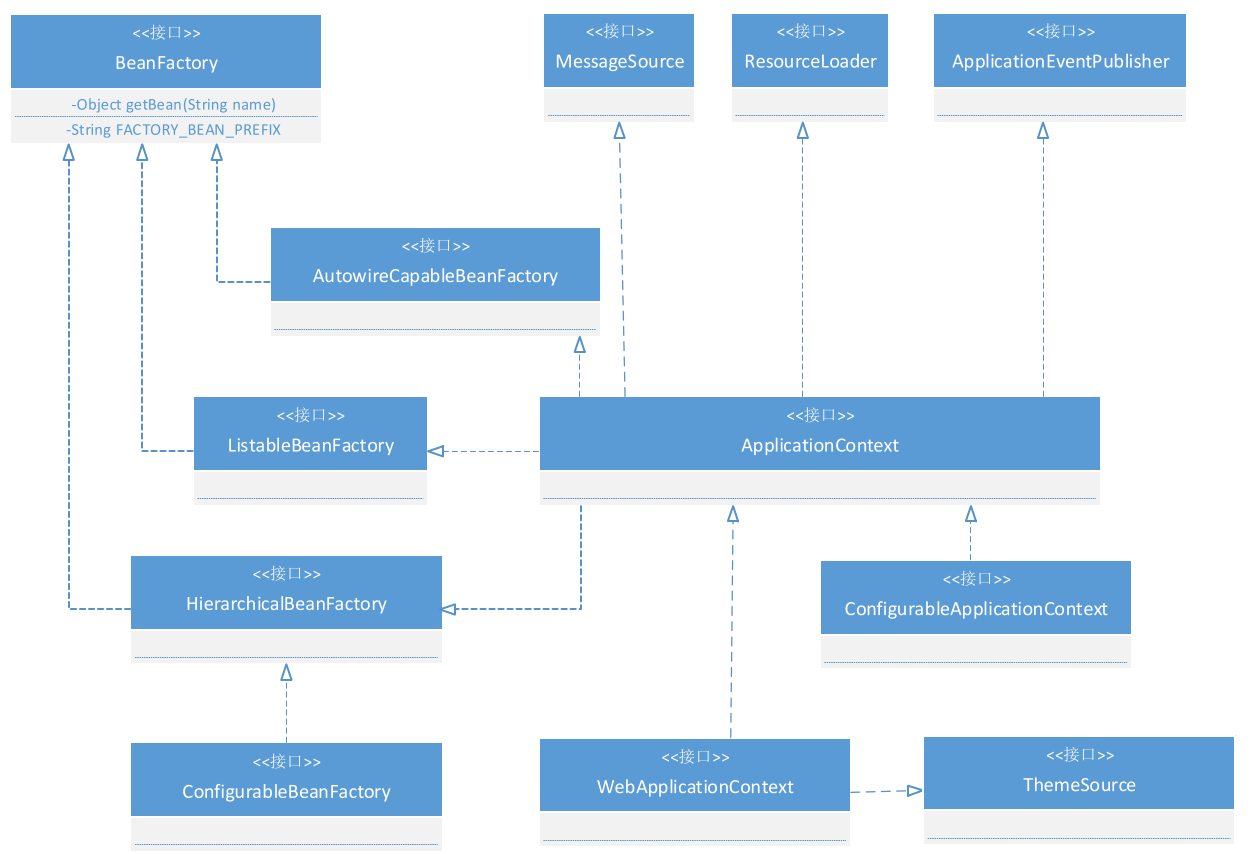
AutowireCapableBeanFactory 自动装载有能力的beanfactory

HierarchicalBeanFactory 分级的beanfactory

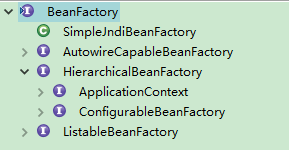
ListableBeanFactory 可列举的beanfactory



# IOC容器两大设计路线



## 基本实现



BeanFactory 🡪 HierarchicalBeanFactory 🡪 ConfigurableBeanFactory

* HierarchicalBeanFactory 具备双亲IOC容器的管理功能.getParentBeanFactory
* ConfigurableBeanFactory 具备设置双亲IOC容器 setParentBeanFactory

HierarchicalBeanFactory：分级工厂

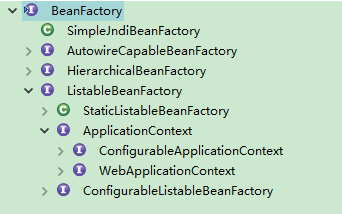
在继承了BeanFactory基本接口之后，增加了getParentBeanFactory的接口功能，使BeanFactory具备了双亲IOC容器的配置功能。

ConfigurableBeanFactory：可配置工厂

配置接口由大多数bean工厂实现。 除了org.springframework.beans.factory.BeanFactory接口中的bean factoryclient方法之外，还提供了配置bean工厂的功能。

设计路线: ConfigurableBeanFactory和HierarchicalBeanFactory的产品设计思想为 可配置,可管理.强调IOC容器的功能性实现.

## 高级功能



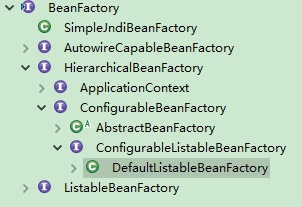
BeanFactory 🡪 ListableBeanFactory 🡪 ApplicationContext 🡪 WebApplicationContext

* ListableBeanFactory 细化BeanFactory的接口功能. 其中defaultListableBeanFactory默认实现的中间产品.可以基于该产品进行扩展开发.
* ApplicationContext 强调对应用环境的适配,在框架中实际使用的是WebApplicationContext

ApplicationContext 通过继承BeanFactory的ListableBeanFactory HierarchicalBeanFactory AutowireCapableBeanFactory 来获取基本功能,还通过继承MessageSource, ApplicationEventPublisher, ResourcePatternResolver 来获取高级功能.

设计路线: ApplicationContext 可直接使用, 强调产品性

## 具体IOC容器实现



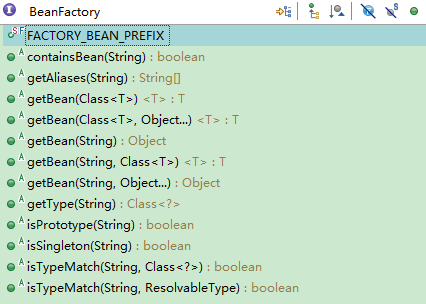
DefaultListableBeanFactory 实现了 ConfigurableBeanFactory 从而成为一个简单的IOC容器实现.

其他IOC容器,比如XmlBeanFactory 都是在DefaultListableBeanFactory 基础上扩展的.

XmlBeanFactory 已过时。从Spring 3.1开始，支持DefaultListableBeanFactory和XmlBeanDefinitionReader

对于高级需求，请考虑使用XmlBeanDefinitionReader中的DefaultListableBeanFactory。后者允许从多个XML资源中读取，并且在其实际的XML解析行为中具有高度可配置性。

# BeanFactory 的应用场景

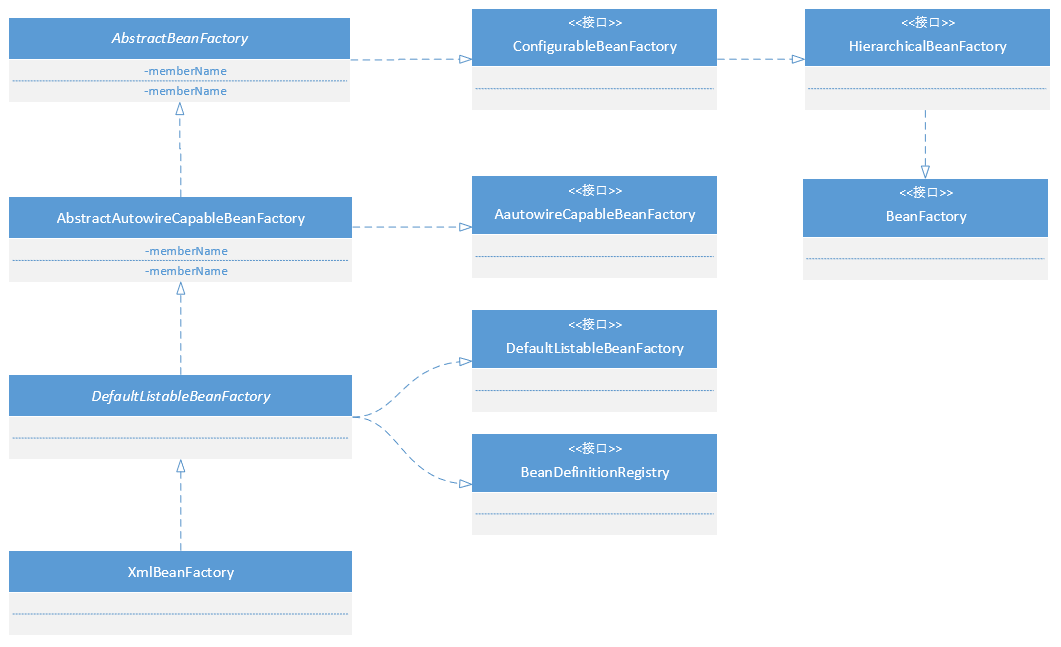


String FACTORY\_BEAN\_PREFIX = "&";

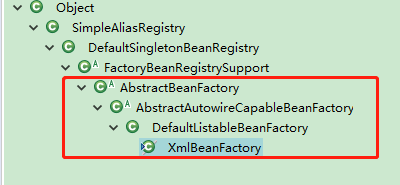
用来区分通过容器来获取BeanFactory产生的对象和获取BeanFactory本身

使用&时获取BeanFactory本身

# BeanFactory 容器设计原理

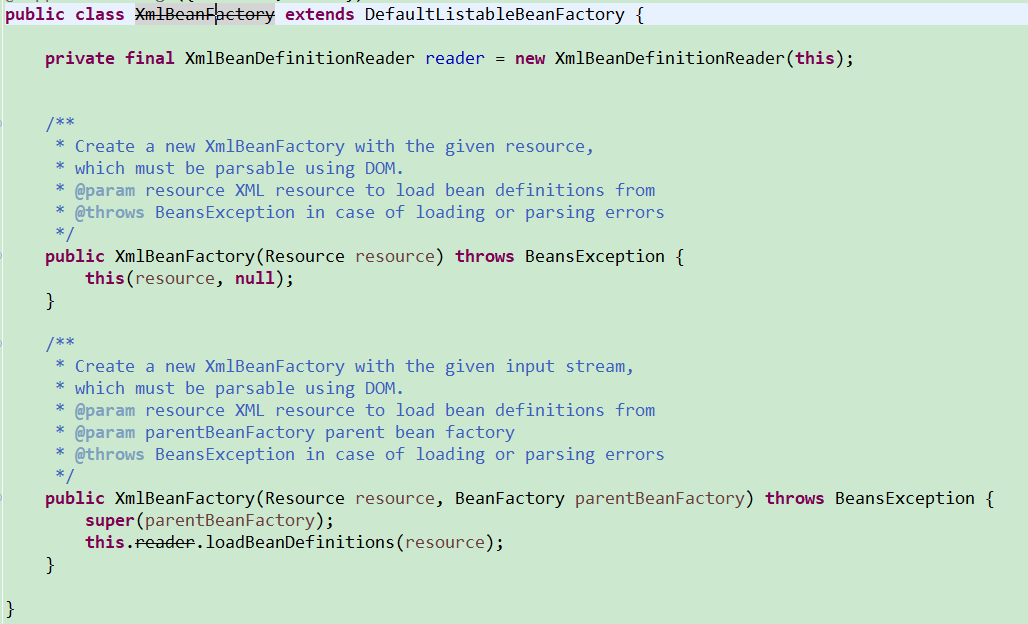


以XmlBeanFactory 为例进行说明



可以看出来, 在spring中 抽象类 为接口的下一层实现.

其中DefaultListableBeanFactory 作为一个默认的功能完整的IOC容器来使用。包含了基本IOC容器所具有的的重要功能。XmlBeanFactory通过继承DefaultListableBeanFactory,获取到了他的基本功能.并在其基础之上添加了读取解析XML新的功能.



如何实现XML读取功能?

通过以下接口和类来组合实现:

* XmlBeanDefinitionReader XML 文件处理
* Resource IO流操作封装接口

构造器初始化XmlBeanFactory时,传入Resource 接口

XmlBeanDefinitionReader初始化后调用reader.loadBeanDefinitions(resource);

载入beandefinitions ，完成IOC容器初始化。就可以直接使用该容器。

编程实现DefaultListableBeanFactory:

回调的核心:就是回调方将本身的this传递给对方.

|  |
| --- |
| ClassPathResource res = new ClassPathResource(“beans.xml”);  DefaultListableBeanFactory factory = new DefaultListableBeanFactory();  XmlBeanDefinitionReader reader = new XmlBeanDefinitionReader(factory);  Reader.loadBeanDefinitions(res); |

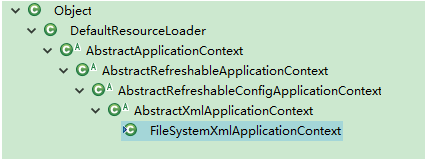
1. 创建IOC配置文件的抽象资源,这个抽象资源包含了BeanDefinition的定义信息
2. 创建一个BeanFactory , 这里使用DefaultListableBeanFactory
3. 创建一个载入BeanDefinition的读取器，这里使用XMLBeanDefinitionReader来载入XML文件形式的BeanDefinition，通过一个回调配置给BeanFactory
4. 从定义好的资源未知读入配置信息，具体的解析过程由XmlBeanDefinitionReader来完成。完成整个载入和注册Bean定义之后，需要的IOC容器就建立起来了。

# ApplicationContext 的应用场景

1. 支持不同的信息源，国际化的支持，多语言版本的应用提供服务。MessageSource。
2. 访问资源。ResourceLoader、Resource、ResourcePatternResolver。一般通过继承DefaultResourceLoader的子类，因为DefaultResourceLoader是AbstractApplicationContext的父类。
3. 支持应用事件。继承了接口ApplicationEventPublisher，在上下文中引入事件机制。这些事件和Bean的生命周期的结合为Bean的管理提供了便利。

# ApplicationContext容器的设计原理

以FileSystemXmlApplicationContext为例讲解

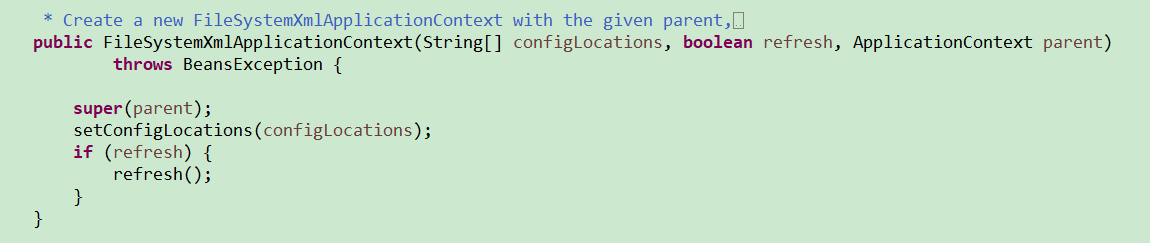


FileSystemXmlApplicationContext中ApplicationContext应用上下文的主要功能已经在FileSystemXmlApplicationContext的基类AbstractXmlApplicationContext中实现。所以作为一个具体的应用上下文，只需要实现和它自身设计相关的两个功能即可。

功能1：

容器的初始化和启动由以下方法来判定

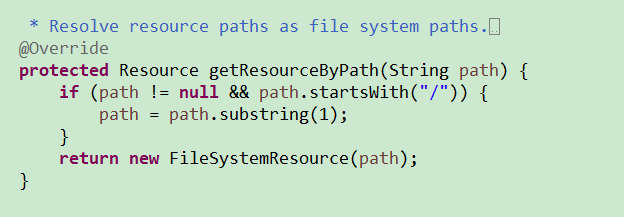
这个refresh()过程会牵涉IOC容器启动的一系列复杂操作，对于不同的容器实现，这些操作都是类似的，因此在基类中将它们封装好。所以，我们在FileSystemXmlApplicationContext的设计中看到的只是一个简单的调用。



功能2：

怎么从文件系统中加载XML的Bean定义资源有关。

通过从文件系统中加载XML这个过程，可以为在文件系统中读取以XML形式存在的BeanDefinition做准备，因为不同的应用上下文实现对应着不同的读取BeanDefinition的方式，在FileSystemXmlApplication 的代码实现如下



通过调用这个方法，可以得到FileSystemResource的资源定位

# IOC容器的初始化过程

IOC容器的初始化由refresh()方法来启动，这个方法标志着IOC容器的正式启动。这个启动包括BeanDefinition的Resource定位，载入和注册三个基本过程。

Spring将三个基本过程分开，并使用不同的模块来完成。通过这样的设计方式，可以让用户灵活的对三个过程进行裁剪或扩展，定义出最合适自己的IOC容器的初始化过程。

在IOC容器的初始化过程中，是不包含Bean以来注入的实现。

在Spring IOC的设计中，Bean定义的载入和依赖注入是两个独立的过程。

依赖注入一般发生在应用第一次通过getBean向容器索取Bean的时候。

但有一个例外是，设置Bean定义信息中lazying属性。

## BeanDefinition的资源定位

这个定位指的是BeanDefinition的资源定位，它由ResourceLoader通过统一的Resource接口来完成，这个Resources对各种形式的BeanDefinition的使用都提供了统一的接口。比如说FlieSystemResource和ClassPathResource

## BeanDefinition的载入

用户定义好的Bean表示成IOC容器内部的数据结构，而这个容器内部的数据结构就是Beandefinition。

Beandefinition就是POJO对象在IOC容器中的冲向，通过这个Beandefinition定义的数据结构，使IOC容器能够方便地对POJO对象也就是Bean进行管理。

## Beandefinition的注册

向IOC容器中注册Beandefinition，这个过程是通过调用BeandefinitionRegistry接口的实现来完成的。把载入过程中解析得到的Beandefinition向IOC容器进行注册。在IOC容器内部将Beandefinition注入到一个HashMap中，IOC容器就是通过这个HashMap来持有这些Beandefinition数据的。

# Beandefinition的resources定位

1. ClassPathResource

以编程方式使用DefaultListableBeanFactory时，首先定义一个Resource来定位容器使用的

Beandefinition。这时使用的是ClassPathResource，这意味着Spring会在类路径中寻找以文件形式存在的Beandefinition信息。

1. BeandefinitionReader

这里定义的Resource并不能由DefaultListableBeanFactory直接使用，spring通过BeandefinitionReader来对这些信息进行处理。在Application中已经为我们提供了加载不同resource的实现，而DefaultListableBeanFactory只是一个基本的IOC容器，需要为它配置特定的读取器 才能完成这些功能。

以FileSystemXmlApplicationContext为例，它通过继承AbstractApplicationContext具备了ResourceLoader读入以Resource定义的Beandefinition的能力。

因为AbstractApplicationContext的基类是DefaultResourceLoader。

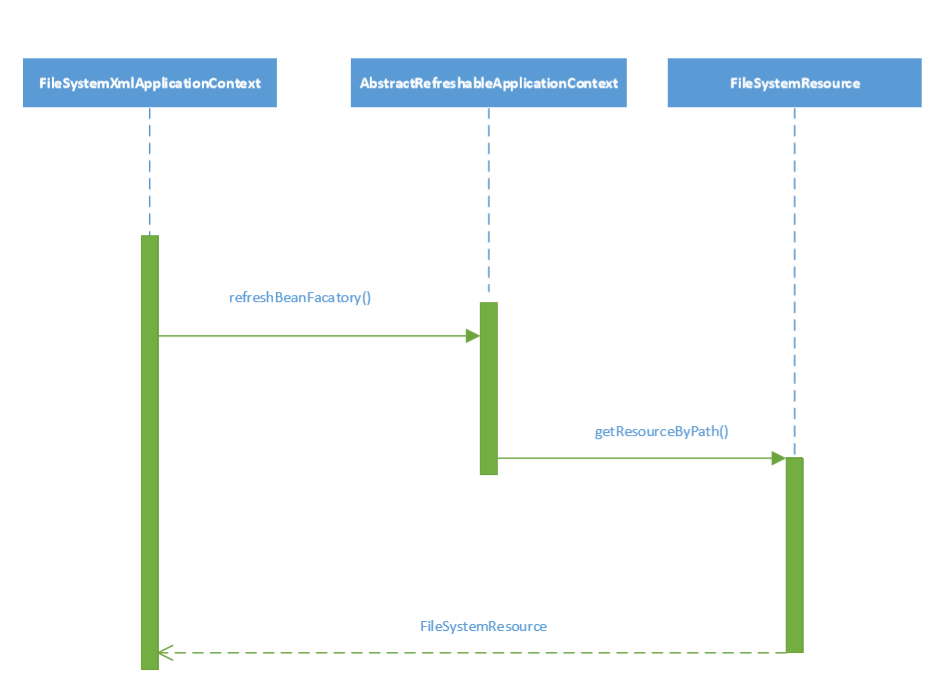
以此类推：

FileSystemXmlApplicationContext，ClassPathXmlApplicationContext以及XmlWebApplicationContext 分别提供不同放入Resource读入功能。

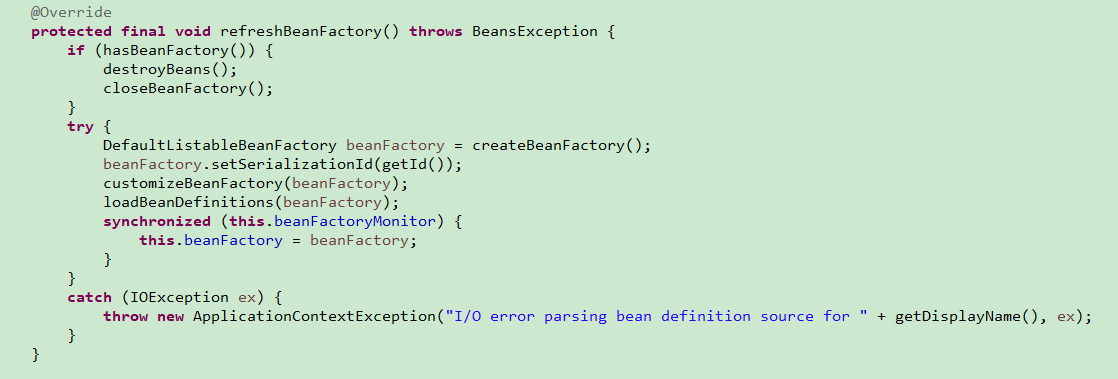


FileSystemXmlApplicationContext是在什么地方定义了BeanDefinition的度如其BeanDefinitionReader,从而完成BeanDefinition信息的读入？

关于读入器的配置，可以在FileSystemXmlApplicationContext的基类AbstractRefreshableApplicationContext中看看它时怎么实现的。

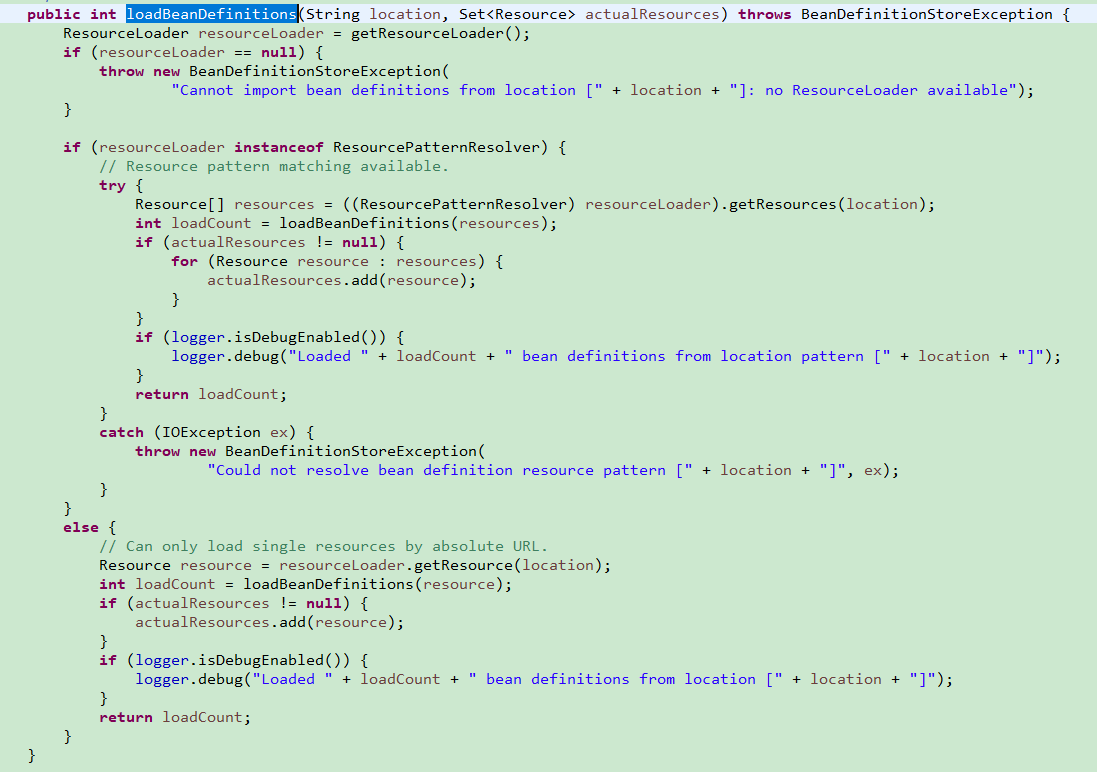


AbstractRefreshableApplicationContext的refreBeanFactory方法的实现，这个refrershBeanFactory被FileSystemXmlFactoryContext构造函数中的refresh调用。



通过createBeanFactory构建了一个IOC容器供ApplicationContext使用。这个容器就是我们前面提到的DefaultListableBeanFactory，同时，它启动了LoadBeanDefinitions来载入BeanDefinition。

具体的载入过程，是在XMLBeanDefinitionReader的基类中AbstractBeanDefinitionReader中可以看到这个载入过程的具体实现。代码如下

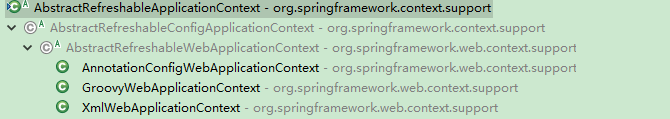


载入的启动，可以在AbstractRefreshableApplicationContext的loadBeanDefinitions方法中看到。因为有多种载入方式，虽然用的最多的是XML定义的形式，这里通过一个抽象函数把具体的实现委托给子类完成。

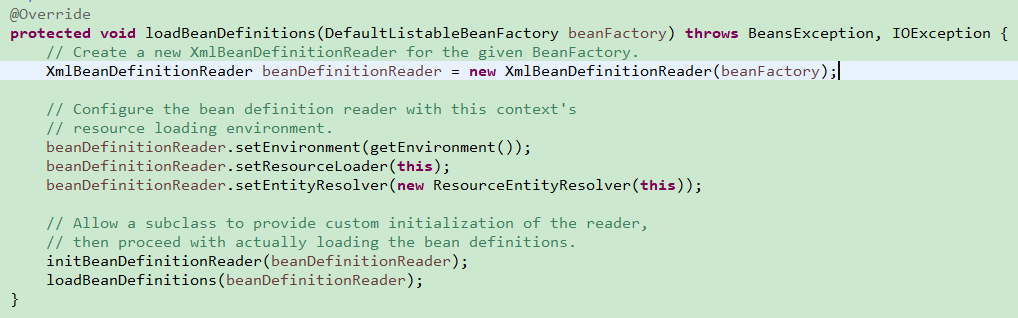
代码如下：

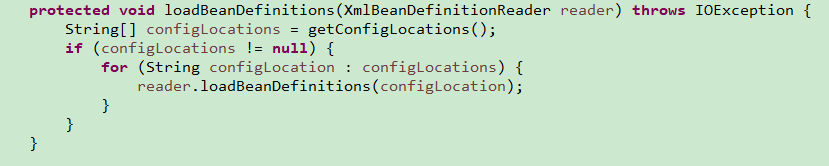


具体的实现子类有

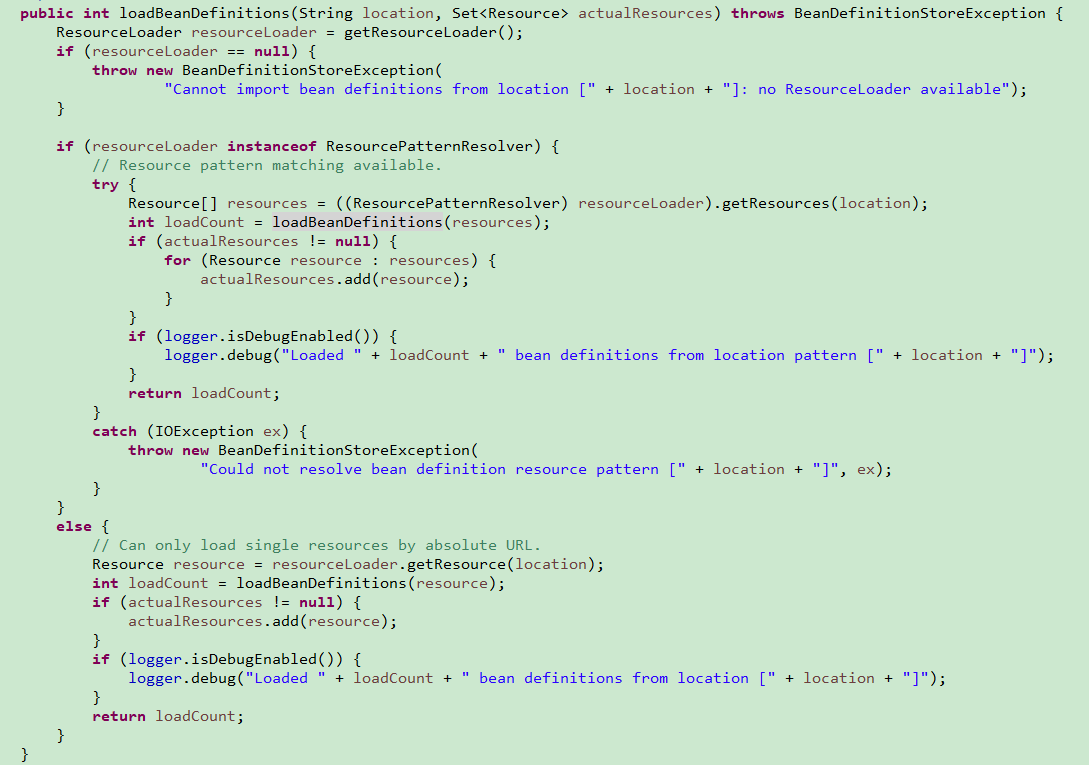


XmlWebApplicationContext的实现代码如下





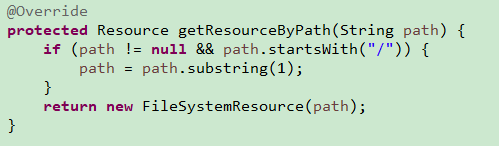
AbstractBeanDefinitionReader的调用代码如下



在上面Resource resource = resourceLoader.getResource(location); 是使用ResourceLoader接口的子类实现来完成的。默认实现DefaultResourceLoader的代码实现如下



最后在异常中返回调用getResourceByPath(location); 也就是FileSystemXmlApplicationContext中的



返回的是Resource接口，所以，根据局不同需要可以返回不同的Resource子类

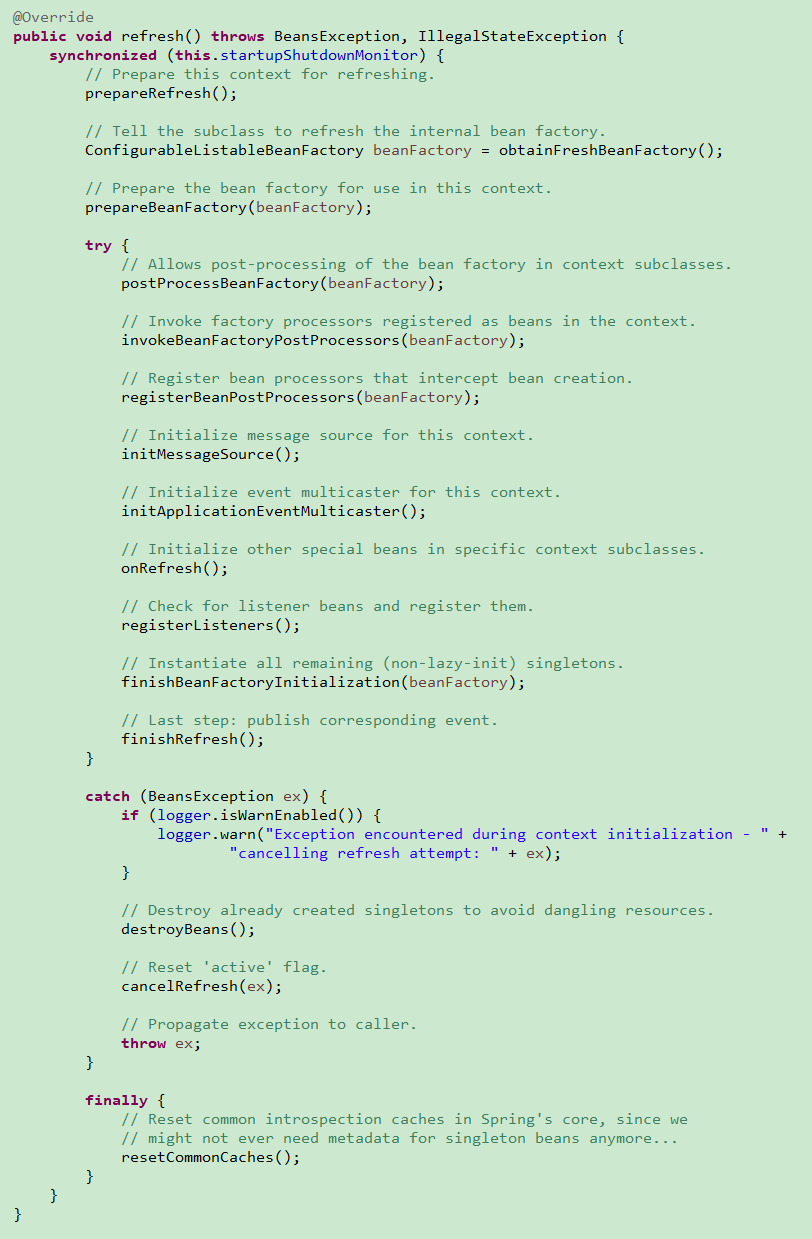
# Beandefinition的载入和解析

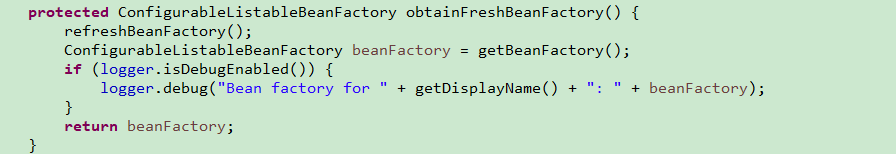
载入的过程就相当于把定义的BeanDefinition在IOC容器中转化成一个Spring内部表示的数据结构的过程。

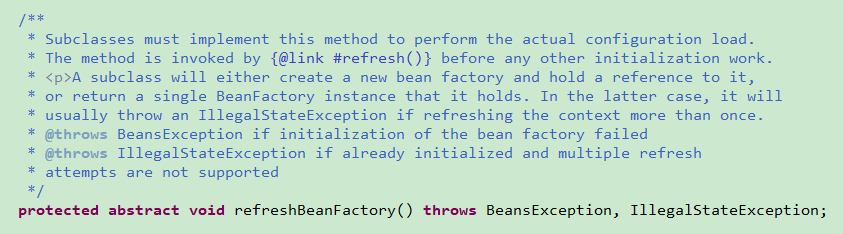
IOC容器对Bean的管理和依赖注入功能的实现，是通过对其持有的BeanDefinition进行各种相关的操作来完成的。

对容器的启动来说，refresh是一个重要的方法。该方法在AbstractApplicationContext类中找到，FilesystemXmlApplicationContext的基类。它详细描述了整个ApplicationContext的初始化过程。如BeanFactory的更新，MessageSource的PostProcess的注册。这里看起来更像是对ApplicationContext进行初始化的模板或执行提纲。

AbstractApplicationContext代码如下：





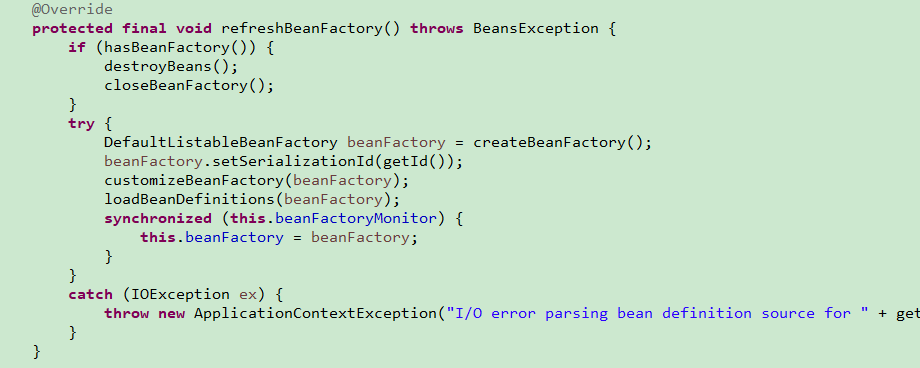


该抽象方法的中文注释说明：

子类必须实现此方法以执行实际的配置加载。该方法在任何其他初始化工作之前由refresh（）调用。

子类将创建一个新的bean工厂并保存对它的引用，或者返回它所拥有的单个BeanFactory实例。 在后一种情况下，如果多次刷新上下文，它通常会抛出IllegalStateException。

AbstractApplicationContext子类AbstractRefreshableApplicationContext的实现代码如下：

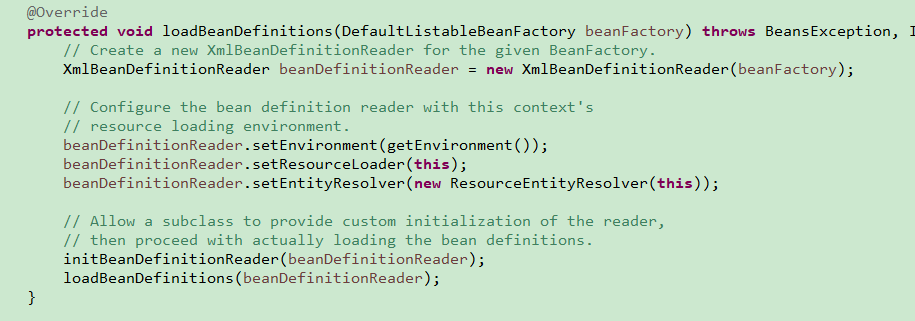


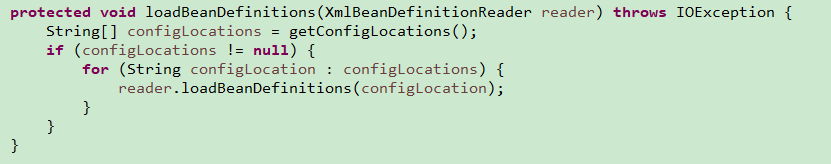
AbstractRefreshableApplicationContext中载入beandefintion接口

C:\Users\roseonly\AppData\Local\Temp\1562573934(1).png

AbstractRefreshableApplicationContext的子类

XmlWebApplicationContext的实现代码如下





AbstractBeanDefinitionReader的代码实现如下



AbstractBeanDefinitionReader实现了BeanDefinitionReader接口的loadBeanDefinitions(Resource resource)方法

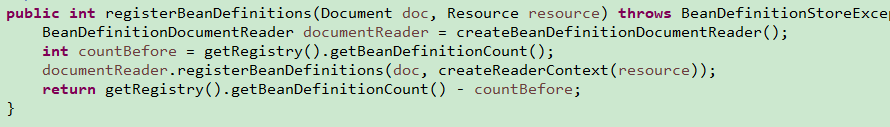


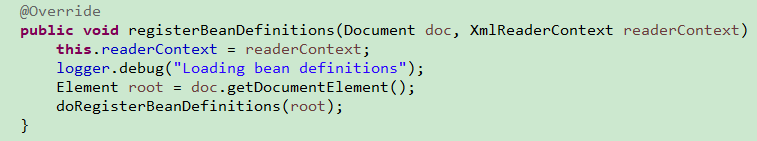
XmlBeanDefinitionReader重写了AbstractBeanDefinitionReader抽象类的loadBeanDefinitions(Resource resource)方法

XmlBeanDefinitionReader的loadBeanDefinitions(Resource resource)方法实现：

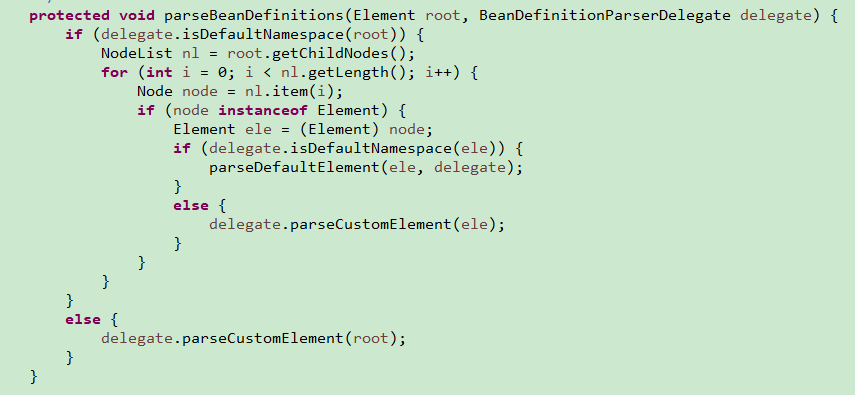






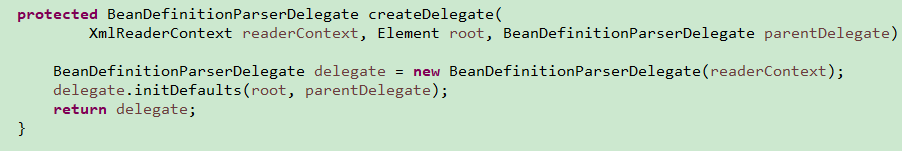
DefaultBeanDefinitionDocumentReader





BeanDefinitionParserDelegate的创建和初始化

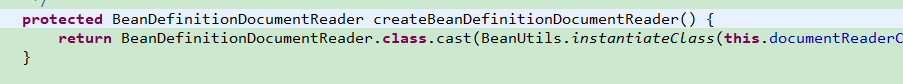
按照Spring的Bean定义规则来对这个XML的文档树进行解析，这个解析是交给BeanDefinitionParserDelegate来完成的



Spring的BeanDefinition是怎么按照Spring的Bean语义要求进行解析并转化成容器内部数据结构的，这个过程是在XmlBeanDefinitionReader的registerBeanDefinitions（）方法中完成的。

BeanDefintion的载入分成两个部分，

1. 首先通过调用XML的解析器得到document对象，但这些document对象并没有按照Spring的Bean规则进行解析。在完成通用的XML解析以后，才是按照spring的Bean规则进行解析的地方，这个按照Spring的Bean规则进行解析的过程是在documentReader中实现的。这里使用的documentReader是默认设置好的DefaultBeanDefinitionDocumentReader.这个DefaultBeanDefinitionDocumentReader的创建是通过



This.documentReaderClass=DefaultBeanDefinitionDocumentReader.class;

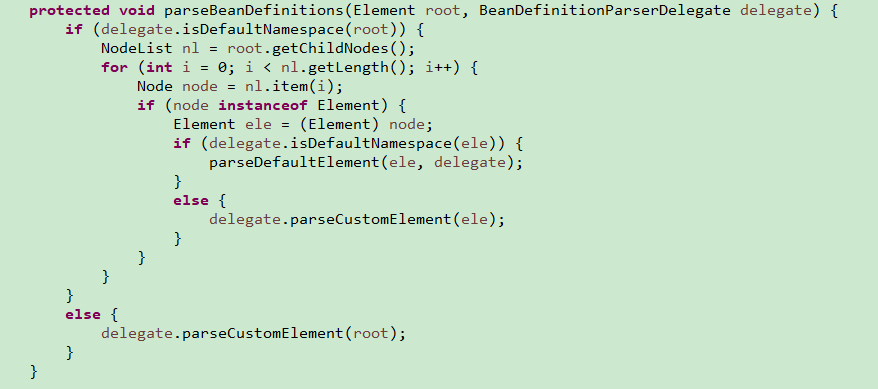
C:\Users\roseonly\AppData\Local\Temp\1562578803(1).png

来完成的。

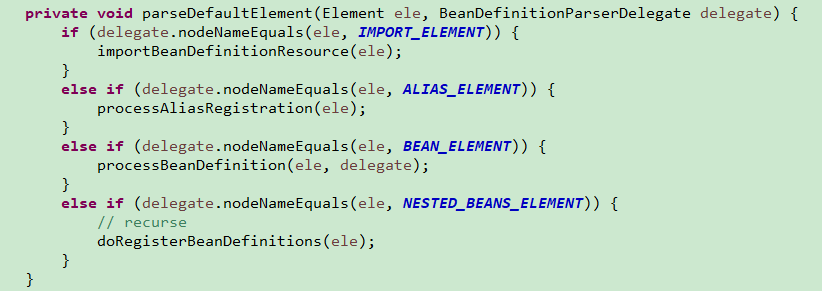
1. 完成BeanDefinition的处理，处理的结果由BeanDefinitionHolder对象来持有。

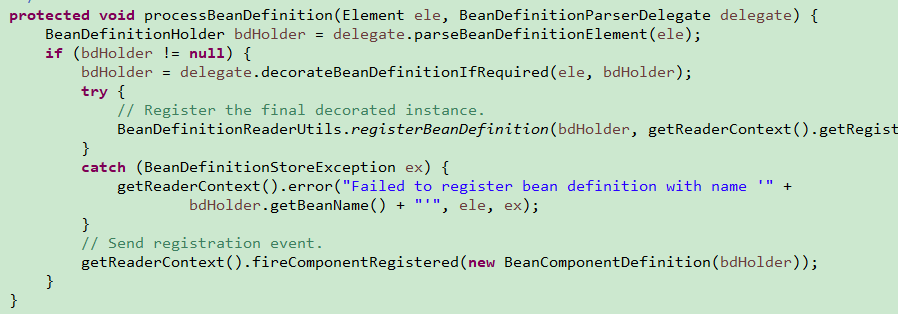
BeanDefinitionHolder除了持有BeanDefinition对象外，还持有其他与BeanDefinition的使用相关的信息，比如Bean的名字，别名集合。BeanDefinitionHolder的生成是通过对Document文档树的内容进行解析来完成的。 可以看到这个解析过程是由BeanDefinitionParserDelegate的processBeanDefinition方法中实现的，同时 这个解析是与Spring对BeanDefinition的配置规则紧密相关的。

DefaultBeanDefinitionDocumentReader的parseBeanDefinitions方法

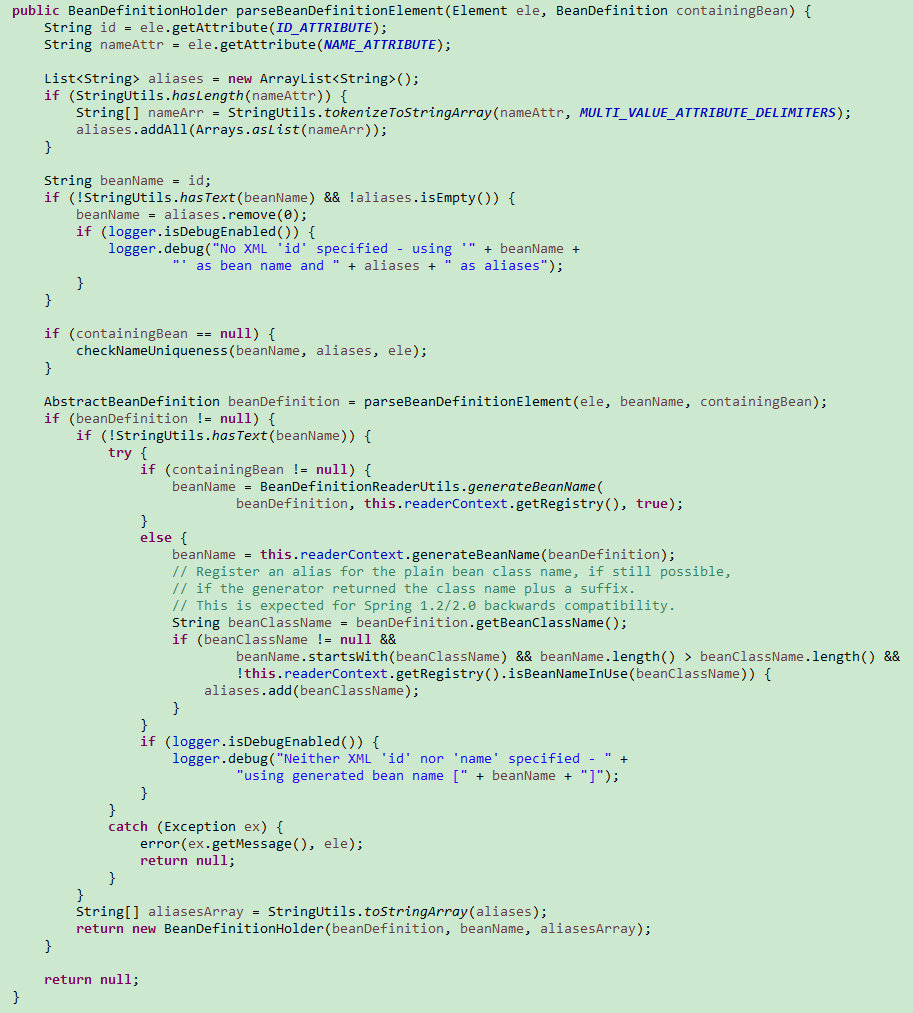


parseDefaultElement(ele, delegate);



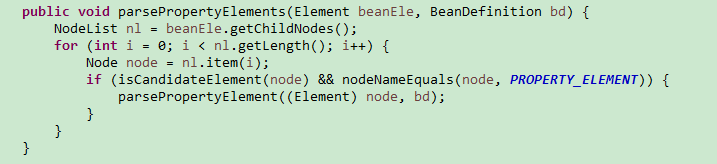


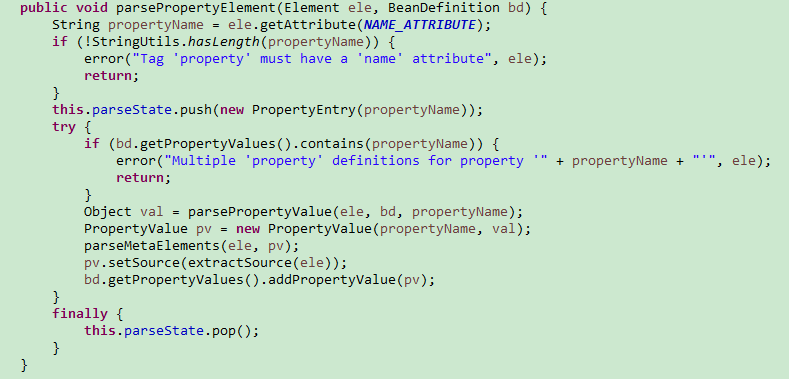
Bean属性配置，通过一个较为复杂的解析过程来完成，这个过程是由parseBeanDefinitionELement来完成的。解析完成以后，会把解析结果放到Beandefinition对象中并设置到BeandefinitionHolder中去。

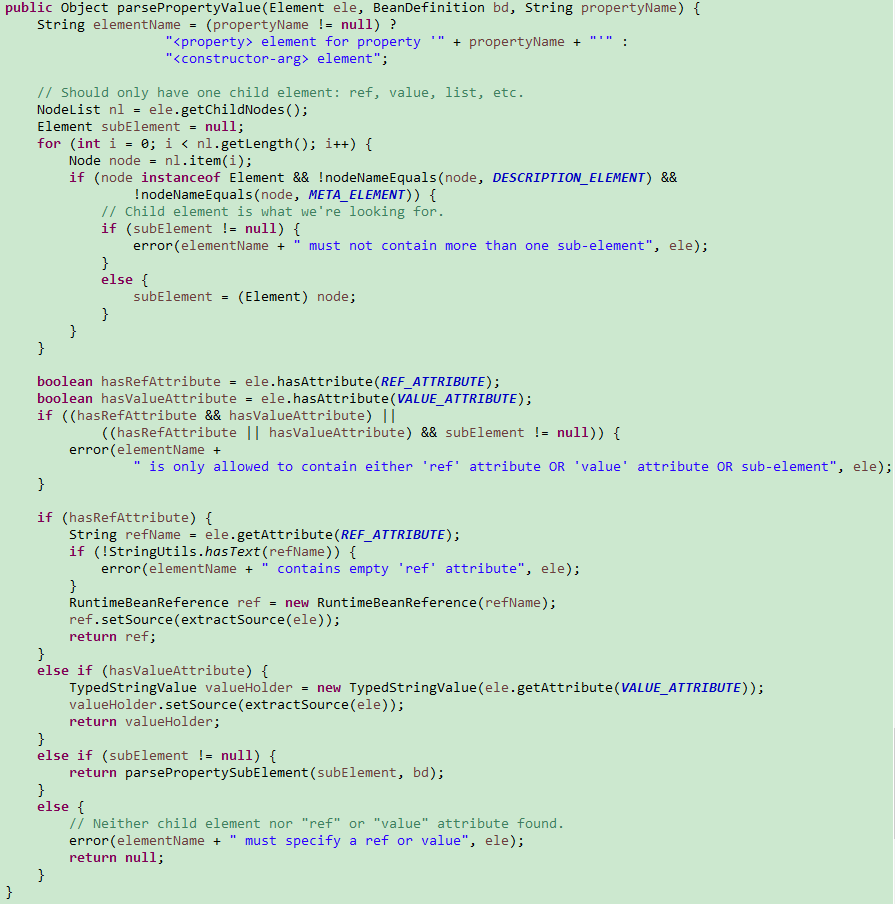




举例说明对property的解析，来完成对整个Beandefinition载入过程的分析，对属性值的处理会被封装到PropertyValue对象并设置到BeanDefinition对象中去。





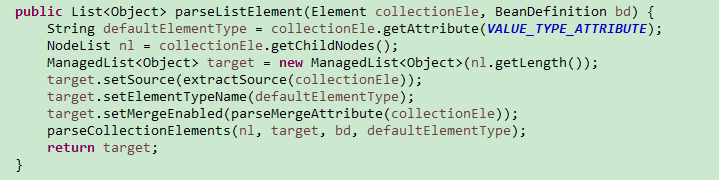


parseArrayElement,parseListElement,parseSetElement,parseMapElement,parsePropELement对应着不同类型的数据解析，同时这些具体的解析方法在BeanDefinitionParserDelegate类中也都能找到。

对ProPerty元素的解析代码实现如下



List的属性配置是怎么被解析的，依然是在BeanDefinitionParserDelegate中，返回的是一个List对象。这个list是Spring定义的ManagedList,作为封装List这类配置定义的数据封装。



经过这样逐层解析，我们在XML文件中定义的BeanDefinition就被整个载入到了IOC容器中，并在容器中建立了数据映射。在IOC容器中建立了对应的数据结构，或者说可以看成是POJO对象在IOC容器中的抽象，这些数据结构可以以AbstractBeandefinition为入口，让IOC容器执行索引，查询和操作。经过以上的载入过程，IOC容器大致完成了管理Beand对象的数据准备工作，

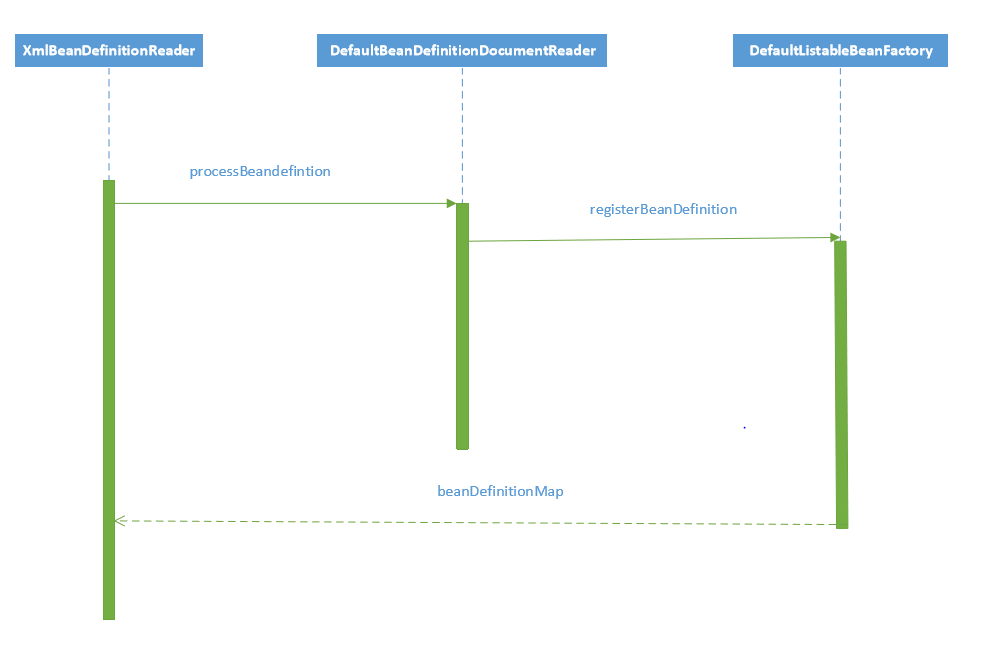
现在IOC容器Beandefinition中存在的还只是一些静态的配置信息。严格地说，这时候的容器还没有完全起作用，要完全发挥容器的作用，还需完成数据向容器的注册。

# Beandefinition的注册

在DefaultListableBeanFactory中，是通过一个HashMap来持有载入的Beandefinition的，这个HashMap的定义在DefaultListableBeanFactory中可以看到



将解析得到的BeanDefinition向IOC容器中的BeanDefinitionMap注册的过程是在载入BeanDefinition完成后进行的。



把解析得到的BeanDefinition设置到hashMap中去。需要注意的是，如果遇到同名的BeanDefinition，进行处理的时候需要依据allowBeanDefintionOverriding的配置来完成。



完成了BeanDefinition的注册，就完成了IOC容器的初始化过程，此时，在使用的IOCrongqi defaultListableBeanFactory中已经建立整个Bean的配置信息，而且这些BeanDefintion已经可以被容器使用了，它们都在beandefinitionMap里检索和使用，。容器的作用就是对这个信息进行处理和维护。

# IOC容器的依赖注入

从DefaultListableBeanFactory的基类AbstractBeanFactory来看依赖注入的实现