书恒的Spring深入手记

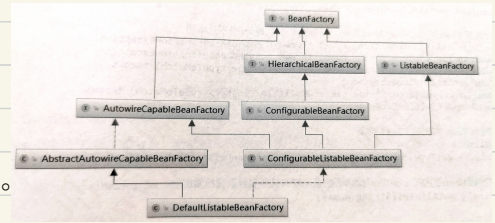
段书恒

起草与：2022年08月01日23:29:45

Ioc 容器：把代码里需要自己实现的对象创建、依赖，反转给ioc容器来实现。具体就是创建一个容器，并且需要描述创建对象和对象之间的关系，这个描述就是xml文件

DI依赖注入：对象在创建的时候不自己去找依赖类，而是容器在实例化对象的时候主动的把依赖类注入给它

1.spring核心类图

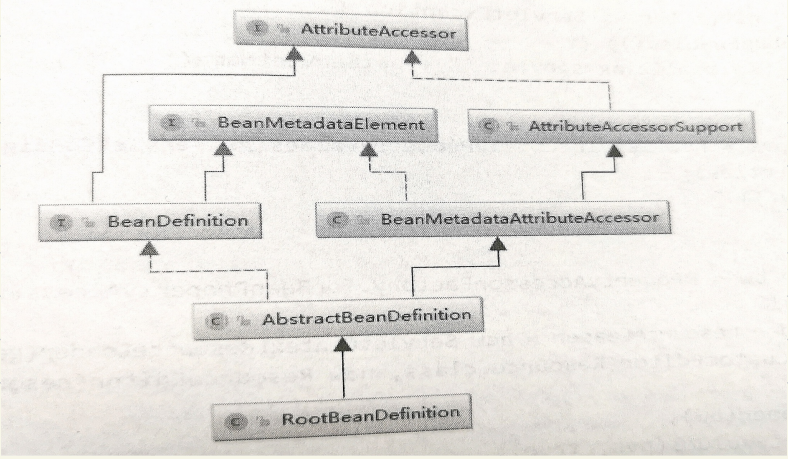


BeanFactory：
bean工厂，最典型的工厂模式。
顶层的接口类，
只定义了基本规范：
getBean：通过bean名称、Class类型、参数。共有6种组合来获取ioc容器里具体的一个Bean
isSingleton：bean是否单例
isPrototype：是否是原型模式，和单例相对
*把一个类做模板，定制化出另一个类的模式，比如克隆

isTypeMatch、geType、getAliases等等

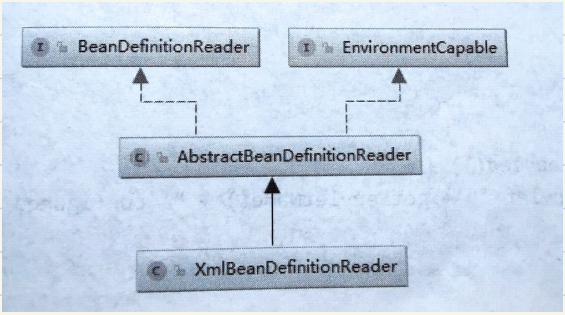
有3个重要子类：
ListableBeanFactory 这些Bean可列表化
HierarchicalBeanFactory 这些Bean有继承关系
AutowireCapableBEanFactory 这些Bean自动装配规则
共同定义了Bean的集合、关系、行为

BeanDefinition:
管理Bean对象之间相互关系,在xml的ioc初始化中用于resource定位、加载、注册



BeanDefinitionReader：

Bean的解析（主要是对spring配置文件的解析）主要通过BeanDefinitionReader完成



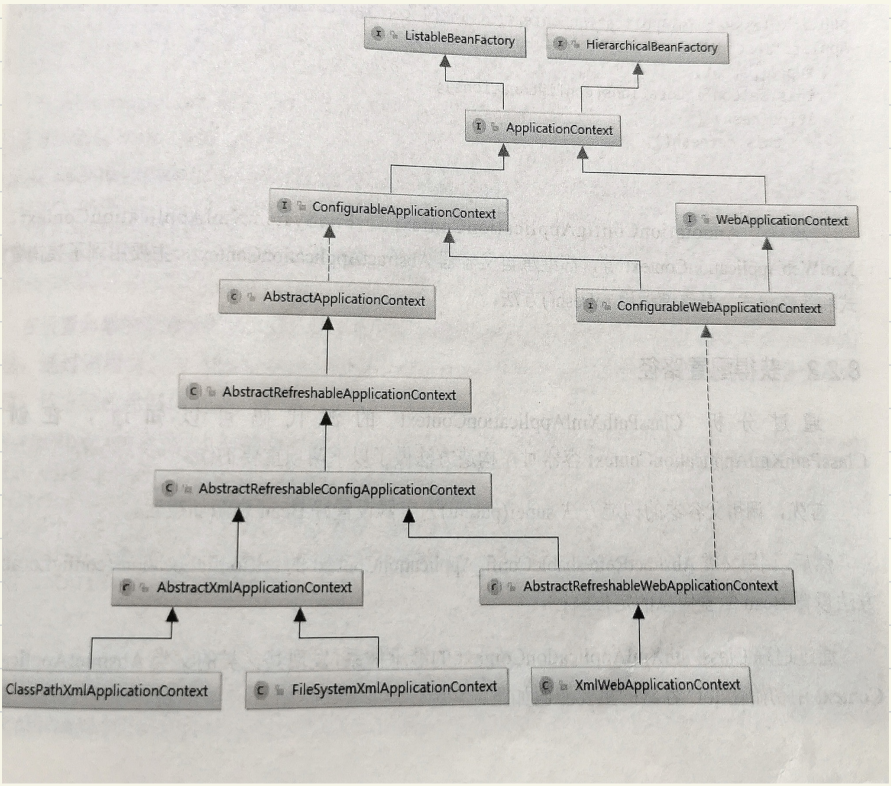
基于XML的IoC容器初始化

BeanDefinition的Resource定位、加载、注册

如：ApplicationContext

（ClassPathXmlApplicationContext）

（XmlWebApplicationContext）



1.定位入口：（加载器、定位路径）

ClassPathXmlApplicationContext，构造器做了两个工作

调用父容器super（parent）为容器设置好Bean资源加载器

调用父类AbstractRefreshConfigApplicationContext的setConfigLocations（configLocations）方法设置Bean配置信息的定位路径

关于创建spring资源加载器，实际上是在AbstractApplicationContext调用一个参数为ResourceLoader的类，在它构造方法里设置this.resouceLoader=参数，这样配置的资源加载器。

关于setConfigLocation可以用一个字符串、或者字符串数组来配置 方式的区别：

实际上肯定有很多资源文件，所以他们的区别就是一个字符串配置的方式，要多经过一次转化成字符串 数组的过程，实际上最终还是调用字符串数组配置spirng bean信息的方法

*ApplicationContext允许上下文嵌套，首先检查当前上下文，再检查父上下文，逐级向上，这样可以为不同spring应用提供共享的Bean定义环境

2.加载入口：（refresh()方法）

refresh()是个模板方法，规定了启动流程，具体子类实现。

ClassPathXmlApplicationContext通过调用父类AbstractApplicationContext的refresh()方法启动了整个IoC容器的载入。

载入实际上是ConfigurableListableBeanFactory beanFactory=obtainFreshBeanFactory()

这段代码，通过这段代码启动。剩下的都是在注册容器的信息源和生命周期。

refresh()的作用类似重启IoC容器，因为如果有容器存在，它会将已有的容器销毁、关闭。以此保证refresh()方法之后用的是新的容器，对新创建的容器进行初始化，对Bean配置资源进行载入。

3.创建容器:

obtainFreshBeanFactory调用了refreshBeanFactory()方法（实际上AbstractApplication是由子类AbstractRefreshableApplication实现的），启动载入Bean配置信息的过程。

首先会判断beanFactory是否存在，存在就会销毁Bean并且关闭beanFactory。

然后接着创建DefaultListableBeanFactory调用loadBeanDefinitions()方法装载Bean定义（实际上也是子类实现的）。

4.载入配置路径:

在AbstractRefreshableApplicationContext中只定义了loadBeanDefinitions()方法，容器真正调用的是子类AbstractXmlApplicationContext实现的方法。

创建了Bean读取器，即XmlBeanDefinitionRead

为Bean读取器设置spring资源加载器（setResourceLoader()）

为Bean读取器设置SAX xml解析器

调用loadBeanDefinitions()方法->

Resource[]获取Bean资源定位（getConfigResources()方法），如果不为空，读取器进行读取;

String[]获取ClassPathXmlApplicationContext的构造方法中设置的资源（getConfigLocations()方法），同时读取器进行读取。

其实这里就是一个策略，因为ClassPathXmlApplicationContext为例，getConfigResource()方法返回值是空，所以读取器会执行第二个分支读取。

5.分配路径处理策略:

XmlDefinitionReader的抽象父类AbstractBeanDefinitionReader调用了loadBeanDefinitions()方法

将指定位置信息的Bean配置信息解析为Spring IoC容器封装的资源

读取资源加载器（getResourceLoader()）

源码有一个分支:

如果资源加载器 属于 ResourcePatternResolver子类则:

Resource[] 接受资源加载器加载多个位置的Bean信息（getResources()，在使用资源加载器时候进行了一次强转为ResourcePatternResolver过程）

委派调用子类xmlBeanDefinitionReader的方法，实现加载功能

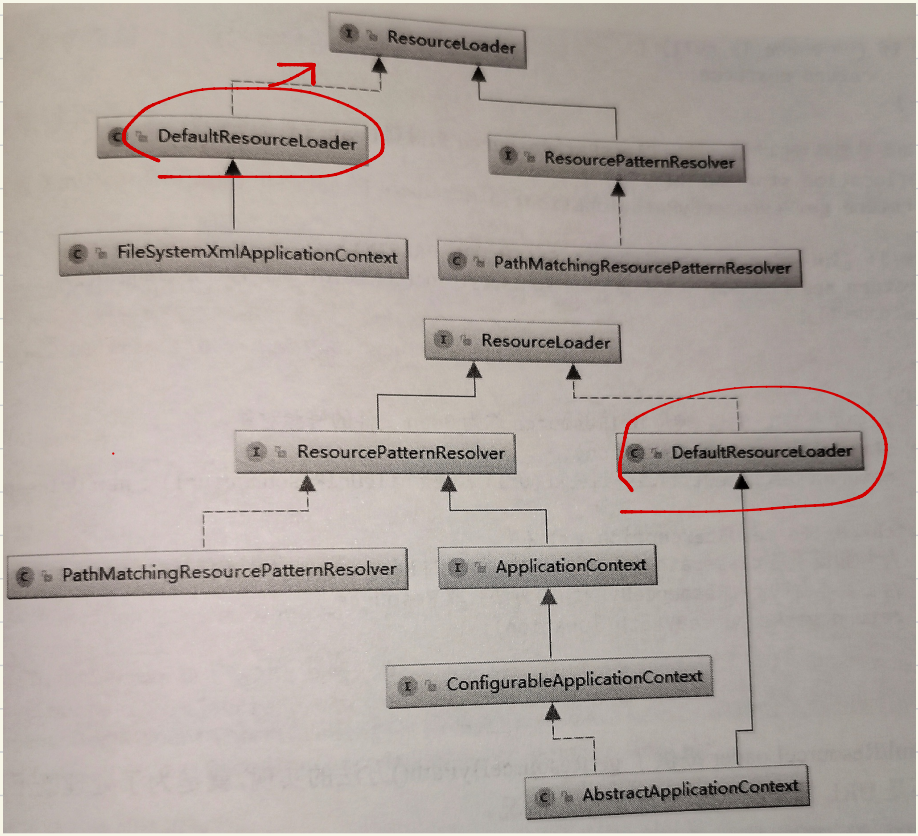
由于是多个，对resource用for增强的方式给actualResource.add上。

否则:

Resource 接受资源加载器加载单个指定位置的Bean信息（getResource()）
委派调用子类xmlBeanDefinitionReader的loadBeanDefinitions()方法，实现真的加载

这里为了应对多个资源（string ...），重载了loadBeanDefinitions方法（String ...），在for增强内依次调用了loadBeanDefinitions（String）方法，最后返回调用的结果值counter

loadBeanDefinitions()调用了getResources()方法，看下类图。



由于实际上调用的是DefaultResourceLoader的getSource()方法定位的Resource，而AbstractApplicationContext实现了它，又被ClassPathXmlApplicationContext实现，所以ClassPathXmlApplicationContext本来就是ResourceLoader的实现类

6.解析配置文件路径

XmlBeanDefinitionReaders通过调用ClassPathXmlApplicationContext的父类DefaultResourceLoader的getResource()方法获取要加载的资源：

获取Resource的具体实现方法：

getResource()

如果是类路径的方式：

则使用getResourceByPath(locations)方法（小细节：必须”/“开头（startsWith()方法））

然后使用ClassPathResource得到资源对象

如果是Url，则使用UrlResource作为资源对象

如果不是类路径和Url，使用容器本身的getResourceByPath()方法获取Resource，例如FileSystemXmlApplication重写了getResourceByPath()方法，最后使用FileSystemResource来作为资源对象（文件系统资源）

7.开始读取配置内容

转化为文档对象

XmlBeanDefinitionReader的loadBeanDefinitions()方法

先将资源文件转为InputStream流，然后包装在InputSource内，将xml对象转化为Document类（DOM）对象，种类会用到spring的Bean配置规则（regisBeanDefinitions()）

8.准备文档对象

目标：DocumentLoader将Bean配置信息转化为文档对象

过程：创建文件解析工厂，并通过工厂创建文档解析器，最后将其返回。

loadDocument()方法

在创建工厂过程中，设置解析XML的校验

这个解析过程使用JavaEE的JAXP标准处理的

意义：其实也是属于载入的过程，真正解析为Bean是在转化为文档对象之后

this.de略
目标：文档对象解析为Spring IoC管理的Bean对象

调用registerBeanDefinitions()方法，顺便返回解析的Bean的数量
这里创建了BeanDefinitionDocumentReader对象（createBeanDefinitionDocumentReader()方法），但是实际上这里使用了委派模式（我已经感觉到spring委派和工厂模式是用最多的），实际上使用实现类DefaultBeanDefinitionDocument完成的。
documentReader.registerBeanDefinitions(doc,createReaderContext(resource))
这里将创建了XmlReaderContext进行真正的解析，读取Xml中的配置并注册BeanDefinition。

10.将配置载入内存
BeanDefinitionDocumentReader对象调用registerBeanDefinitions方法委派了实现类DefaultBeanDefinitionDocumentReader对象进行解析的具体过程

获取XML描述符，并获得Document元素，然后进行解析。

在解析Bean之前进行自定义解析
preProcessXml(dom)

解析Bean定义的文档
parseBeanDefinition(dom,this.delegate)
过程：
获取文档对象根元素所有子节点
通过遍历，对Spring默认命名空间的元素节点使用默认方法解析
对用户自定义的进行另外的解析

解析：
如果是
<import>进行导入解析
<alias>别名解析
不是导入和别名，进行Bean解析

import：
从给定的导入路径加载Bean资源到IoC容器中
获取导入元素location属性
location=dom.getAttribute(常量)
如果location是空，没有资源，直接返回
用系统变量解析location属性（实际上是路径）
绝对路径：
用资源读入器加载Bean资源
相对路径：
相对路径存在：
资源读入器解析
相对路径不存在：
IoC获取Ioc容器资源读入器基本路径
并加载给定资源
解析完毕后，发送容器导入资源处理完成事件
alias：
获取location，这里获取字段name、alias的属性
这里有一个布尔判定，任何一个为空都无法注册
资源读入器注册别名：

getReaderContext().getRegistry().registerAlias(name,alias)

Bean：
BeanDefinitionHolder是对BeanDefinition的封装
调用BeanDefinitionParserDelegate类parseBeanDefinitionElement()方法进行解析（极其复杂）
完成向IoC的注册解析后，发送注册事件

在解析Bean之后进行自定义解析
postProcessXml(dom)

11.载入Bean元素
import和alias在DefaultBeanDefinitionDocumentReader已经解析完成
Bean元素由BeanDefinitionParserDelegate来解析

主要处理bean的id、name、别名属性
别名属性是一个ArrayList，用于存放所有name属性值
如果没有配置id，将别名中第一个值赋值给beanName
beanName=aliases.remove(0)

检查id、name、别名是否重复。
然后对Bean元素进行解析
    如果没有配置id、别名或者name，没有子元素
        直接使用
BeanDefinitionReaderUtils.generateBeanName()方法进行注册
    如果没有配置id、别名或者name，有子元素
        通过给别名添加类名后缀ClassName进行注册
    解析出错返回null