Spring源码分析

# 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.

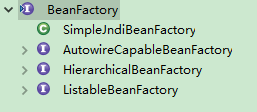
应用上下文在简单容器的基础上,增加了面向框架的特性,同时对应用环境做了许多适配.

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



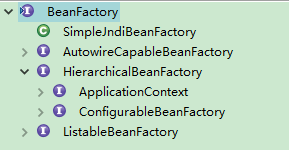
AutowireCapableBeanFactory 自动装载有能力的beanfactory

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

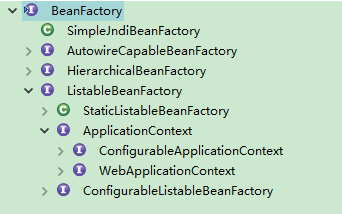
ConfigurableBeanFactory 可配置工厂

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。

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

## 高级功能



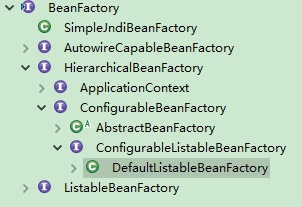
BeanFactory 🡪 ListableBeanFactory 🡪 ApplicationContext 🡪 WebApplicationContext

ListableBeanFactory 细化BeanFactory的接口功能. getBeanDefinitionNames

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.

## 具体IOC容器实现



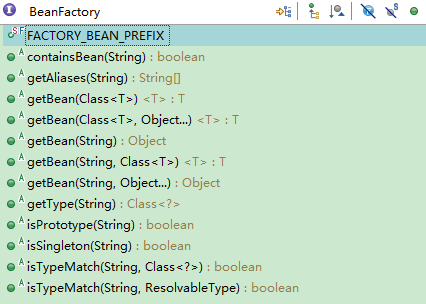
DefaultListableBeanFactory 实现了 ConfigurableBeanFactory 从而成为一个简单的IOC容器实现.

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



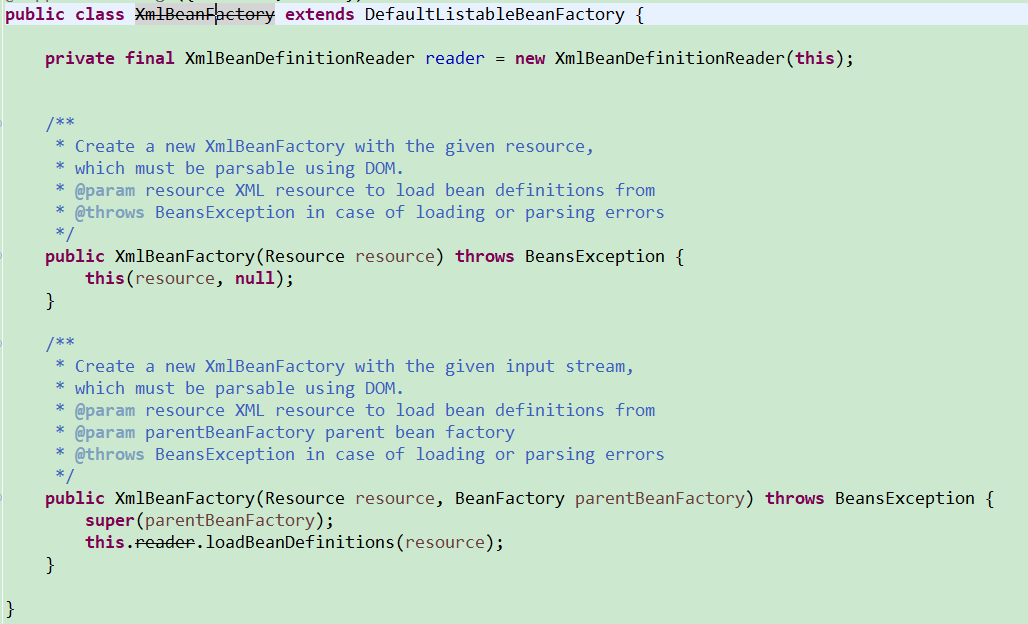
String FACTORY\_BEAN\_PREFIX = "&";

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

# BeanFactory 容器设计原理

以XmlBeanFactory 为例进行说明

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。



XmlBeanDefinitionReader XML文件处理

Resource IO流操作封装接口

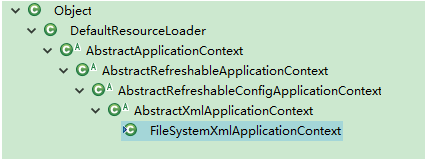
reader.loadBeanDefinitions(resource); XmlBeanDefinitionReader初始化后调用loadBeanDefinitions ，载入beandefinitions ，完成IOC容器初始化。就可以直接使用该容器。

# 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。
3. 支持应用事件。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

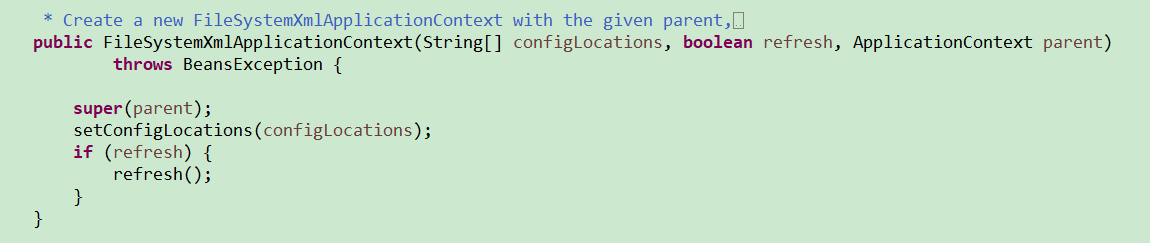


FileSystemXmlApplicationContext中ApplicationContext应用上下文的主要功能已经在FileSystemXmlApplicationContext的基类AbstractXmlApplicationContext中实现。所以作为一个具体的应用上下文，只需要实现和它自身设计相关的两个功能即可。

功能1：

容器的初始化和启动由以下方法来判定

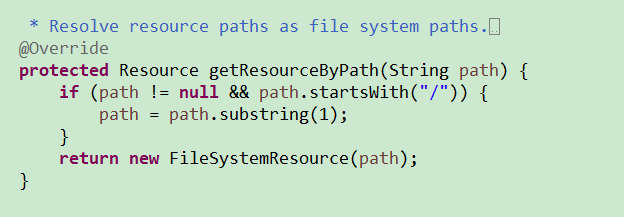
这个refresh()过程会牵涉IOC容器启动的一系列复杂操作，对于不同的容器实现，这些操作都是类似的，因此在基类中将它们封装好。所以，我们在FileSystemXmlApplicationContext的设计中看到的只是一个简单的调用。



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



# 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。

## Resource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。

## 注册Beandefinition

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

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。

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。

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

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。

# Beandefinition的载入和解析