spring容器

{

spring组件:Bean工厂:

{

list

{

只储存别解析类的名字

}

map

{

GenericBeanDefinition

}

BeanFactoryPostProcessor方法\*8

}

}

创建一个新的spring工厂: scan扫描🡪parser解析🡪put map放入map🡪new实例化

AnnotationConfigApplicationContext= new AnnotationConfigApplicationContext(Appconfig.class)

{

//调用父类无参构造方法:

//this.beanFactory = new DefaultListableBeanFactory();🡪他是BeanFactory接口的实现类🡪 也就是我们常说的Bean工厂:

//另外该方法还做了两个事情,一个是实例化一个扫描器,和一个阅读器,阅读器用来阅读beanDefinition\*5

this();

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//什么作用暂时不讲,大概作用是把你传进来的Appconfig.class对象放到容器当中

register(Appconfig.class);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//一共做了三件事

//1.扫描类\*1 && 2.解析类:🡪for循环扫描所有的类,扫描原理先不讲,有兴趣可以看下invokeBeanFactoryPostProcessores()方法,初始化

上下文;

//obtainFreshBeanFactory();获取bean工厂,管理bean并没有创建bean的功能

//prepareBeanFactory(beanFactory);类加载功能,开始创建bean对象

// postProcessBeanFactory(beanFactory);后置处理,加载beanFactory无法处理的内容

//

//当它扫描到一个Test类时,它会实例化一个GenericBeanDefinition genericBeanDefinition = new GenericBeanDefinition()\*5对象;

//向genericBeanDefinition中设置类名genericBeanDefinition.setBeanClass(“Test”);

//设置作用域genericBeanDefinition.setScope(“singleton”);

//设置保存的到底是哪个类genericBeanDefinition.setBeanClass(Test.class);

//.....

//解析完成之后,他就把genericBeanDefinition\*5类放入map\*6当中;🡪map.put(“Test”, genericBeanDefinition);

//接下来,spring将实例化,调用prelnstatiateSingletons new object()方法,从map\*6中拿出beanDefinition的值,根据是否懒加载,是否是原型模式,决定是否new出对象;

//同时在list中保存被解析类的名字,也就是map的键值,为了方便获取map的key值;

//以上便是Bean工厂正常的产生bean实例过程;

//但是spring也可人为的干预spring容器的初始化过程🡪干预容器初始化的过程的方法:

//BeanFactoryPostProcessor(Config.. beanFactory)接口,通过实现这个接口,并需要加上@Component\*3标签 🡪spring会在map构建完成后,实例化之前调用接口实现类里面的方法,通过调用beanFactory的方法可以拿到map中对应名字的BeanDefinition对象🡪

具体例子BeanDefinition TestBean= beanFactory.getBeanDefinition(“Test”);

拿到名字的方法:TestBean.getBeanClassName();

修改被实例化类的方法:TestBean.setBeanClass(Test2.class);

//3.实例化Bean,调用了finishBeanFactoryInitialization(benFactory)实例化单例对象;

//例如加过@Scope(“prototype”)并不会在一开始实例化,它会在getBean时实例化🡪所以该方法又叫实例化剩下的单例对象

//finsh..方法内的beanFactory.preInstantiateSingletons()方法实例化单例模式,之前的方法都为实例化前的验证方法

//spring方法的调用链\*9直到preInstantiateSingletons()方法之前暂时当做空壳方法,preIns..方法后才正式执行实例化方法🡪

//list<String> beanNames = new ArrayList<String>(this.beanDefinitionNames); 拿到bean工厂里的list然后for遍历得到所有将要被实例化Bean的名字

// 通过名字拿到map实例,判断出不是抽象、原型、懒加载、是单例\*10的类进行实例化

//然后判断isFactoryBean(beanName)\*11 普通bean不会进入到if,之后进入到else方法内

//进入执行getBean(beanName),

//进入执行doGetBean();

//进入执行final String beanName = transformedBeanName(name);这里不做具体分析,只说大概,作用一:过滤beanName里可能会出现,spring不支持的特殊字符串,如果传来的是一个FactoryBean,名字前加会加一个&符号,这里是对&符号做一定处理,所以正常Bean实例时可以跳过不看;

//接着执行doGetBean()的Object shareInstance = getSingleton(beanName);这里是重点\*12

//

//

refresh();

}

\*1扫描的本质:

普通类的实例化: .java文件被javac编译成.class文件,虚拟机load成为字节码,new时在堆上产生一个对象

bean一定是一个对象

对象并不一定是一个bean

属于spring bean 条件:一定是一个对象,必须经历过完整的spring生命周期\*2

\*2 spring生命周期🡪bean的产生:

以一个Test类为例,这个类加了@Component\*3,当spring扫描到符合spring规则\*4的类时候,spring会将对象进行拆分,拆分成为对象,new一个BeanDefinition\*5对象🡪里面会保存类里的行为描述,比如:

( 这个类是否是单例,

是否需要懒加载,

这个类的注入模型,

这个类的名字

....

)

\*3 @Component将类实例化成为bean对象的注解

\*4 暂时理解为加了@Component标签的类

\*5 BeanDefinition叫做bean的描述对象,用来描述bean🡪是一个接口,实际是new 他的子类如GenericBeanDefinition();

\*6 spring容器当中用来存放BeanDefinition的一个Map的集合,map存在于BeanFactory(Bean工厂)里面,如果工厂是用来生产bean的实例,那么map就相当于生产bean的原料组件🡪map在未扫描之前已经有了7个自身类的初始化,不需要进行扫描直接存在map中\*7

\*7 这里之前的register()就有将Appconfig.class放入map当中的作用

\*8 实现该接口后,调用接口方法的接受参数,可以取得并设置map;

\*9 AnnotationConfigApplicationContext()🡪 refresh()🡪finishBeanFactoryInitialization()🡪preInstantiateSingletons()🡪getBean()🡪DoGetBean()

\*10 懒加载:只有在你使用时才会实例化,并且和prototype(原型模式)不同,原型模式每次getBean都会重新实例化,懒加载只会实例化一次🡪只调用一次构造方法

\*11 Bean FactoryBean BeanFactory之间的区别 没搞定,大概是FactoryBean是一个特殊bean.

\*12 阅读源码时判断代码是否需要详读,如果代码特别复杂,并且有返回值,可以选择不看