《Spring Framework 系列》- Bean

> 版本信息：
>
> spring framework：4.3.13.RELEASE
>
> JDK: 1.8

议题概览

- BeanDefinition 容器内加载过程
  - 注解/xml配置文件加载
  - Java config加载
- Bean 如何实例化 并被 spring 容器管理
- Bean 如何完成 依赖注入

自我思考

> 一。springframework IOC 容器有多种定义bean的方式（哪几种），分别是在哪个过程中生成beanDefinition的？是怎么样统一起来的（抽象）？

spring ioc 容器 目前有3中定义bean的方式，分别是 xml配置，annotation配置和 Java config方式配置。

xml配置方式 是在xml文件的元素解析过程生成 beanDefinition的。

annotation方式则是在 xml component-scan 元素解析完，通过扫描生成 beanDefinition 的

Java config方式是在context容器加载 配置类后，通过解析类中的成员对象，满足候选对象的类会生成beanDefinition。

spring 是用高层抽象，将三种方式生成的 beanDefinition 统一装填进 BeanDefinition 中的 beanDefinitionMap 容器中，达成了 beanDefinition 的一致性。

> 二。spring bean 和 Java bean 有什么关系？区别和联系是什么？

> 三。spring Bean的实例化过程是如何进行的？它如何被spring 容器管理起来的？

> 四。spring bean 的若干种注入方式，是如何工作的？

> 五。简述 FactoryBean 和 BeanFactory 是什么？区别和联系是什么？

议题探索

> 注：本文没有特殊说明，探讨的皆为 BeanFactory 容器中的 Bean 生命周期过程。

BeanDefinition 加载

核心组件

- org.springframework.beans.factory.support.AbstractBeanDefinition

  > Bean 定义的实体抽象

  - RootBeanDefinition
  - ChildBeanDefinition

- org.springframework.beans.factory.support.BeanDefinitionReader

  > 加载外部配置文件，如 xml

  - XmlBeanDefinitionReader
- org.springframework.beans.factory.xml.BeanDefinitionDocumentReader
  - DefaultBeanDefinitionDocumentReader // 核心类

    > - 将每个 定义 bean 下的 BeanDefinition 对象注册到 spring容器中
    > - 其中，它在注册的过程中，还有解析xml配置文件的职能
    >   - 包括 定制化标签 ，如spring-context.xsd中的 component-scan
    >   - 默认标签，spring-bean.xsd中的标签

- org.springframework.beans.factory.xml.BeanDefinitionParser

  > 该接口被 DefaultBeanDefinitionDocumentReader 使用
  >
  > 用来处理 下的定制的顶级元素

- org.springframework.beans.factory.support.DefaultListableBeanFactory

  > spring 内部容器
  >
  > 在 BeanDefinition 的加载过程中，负责将BeanDefinition加载到 spring 容器中

- org.springframework.beans.factory.support.BeanDefinitionRegistry

  > 持有BeanDefinition对象
  >
  > 持有容器： Map<String, BeanDefinition> beanDefinitionMap

边缘组件

- org.springframework.beans.factory.config.BeanDefinitionHolder
  - 和 BeanNameAware 有关
  - 用来注册别名 alias

核心组件	组件作用	边缘组件	组件作用
DefaultBeanDefinitionDocumentReader	将每个 定义 bean 下的 BeanDefinition 对象注册到 spring容器中	BeanDefinitionHolder	用来注册别名 alias
BeanDefinitionReader	加载bean定义资源		
BeanDefinitionParser	用来处理解析 下的定制的顶级元素		
DefaultListableBeanFactory	spring 内部容器，加载 BeanDefinition加载到 spring 容器中		
BeanDefinitionRegistry	持有 BeanDefinition对象		
AbstractBeanDefinition	Bean 定义的实体抽象		

BeanDefinition加载过程

> 在 spring 容器启动时，spring 持久化的Bean定义会被加载成为 BeanDefinition对象，供后续Bean的实例化做准备
>
> 那么多种Bean的定义是如何转换为 BeanDefinition对象的，我们通过源码来探究此中的原理

```
spring 中保存 BeanDefinition 对象的容器：

/** Map of bean definition objects, keyed by bean name */
private final Map<String, BeanDefinition> beanDefinitionMap = new
ConcurrentHashMap<String, BeanDefinition>(256);
```

注解/xml方式配置加载

- 加载 xml 配置文件
  - org.springframework.beans.factory.xml.XmlBeanDefinitionReader#loadBeanDefinitions(Resource) // 加载xml配置
    - org.springframework.beans.factory.xml.XmlBeanDefinitionReader#registerBeanDefinitions
      - org.springframework.beans.factory.xml.**DefaultBeanDefinitionDocumentReader**#registerBeanDefinitions // 切换到 DefaultBeanDefinitionDocumentReader 实现

        - org.springframework.beans.factory.xml.DefaultBeanDefinitionDocumentReader#doRegisterBeanDefinitions // **重要方法**： 注册元素下每个bean Definition
            - org.springframework.beans.factory.xml.DefaultBeanDefinitionDocumentReader#parseBeanDefinitions

- 解析 【注解方式】 配置 BeanDefinition

  > 解析 base package

  - org.springframework.beans.factory.xml.**BeanDefinitionParserDelegate**#parseCustomElement(Element, BeanDefinition) // **重要方法**： 使用 namespace 解析 元素element
    - org.springframework.context.annotation.ComponentScanBeanDefinitionParser#parse // 此步骤，将 component-scan 解析完成，提取 base-package 路径

  > 解析 Bean Definition

  - org.springframework.context.annotation.**ClassPathBeanDefinitionScanner**#doScan // **重要方法**： 将base-package下的定义bean 扫描出来，并包装成 BeanDefinitionHolder 对象 **【思考：这一步是不是做了统一抽象，将3种配置方式的定义统一提取为 BeanDefinition】**
    - org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider#findCandidateComponents // 找到base package下所有符合的类组件
      - org.springframework.context.annotation.ClassPathBeanDefinitionScanner#registerBeanDefinition // 将 BeanDefinition注册到 指定的 BeanDefinitionRedistry 上
        - org.springframework.beans.factory.support.BeanDefinitionReaderUtils#registerBeanDefinition
          - org.springframework.beans.factory.support.**DefaultListableBeanFactory**#registerBeanDefinition // 将BeanDefinition对象添加到 spring 容器中（beanDefinitionMap）

- 解析 【xml 方式】 配置 BeanDefinition
  - org.springframework.beans.factory.xml.**DefaultBeanDefinitionDocumentReader**#parseDefaultElement
    - org.springframework.beans.factory.xml.DefaultBeanDefinitionDocumentReader#processBeanDefinition
      - org.springframework.beans.factory.support.BeanDefinitionReaderUtils#registerBeanDefinition
        - org.springframework.beans.factory.support.**DefaultListableBeanFactory**#registerBeanDefinition // 将BeanDefinition对象添加到 spring 容器中（beanDefinitionMap）

> 注：根据 注解/xml方式 的BeanDefinition的加载，我们发现，无论是何种形式的 Bean 定义，最后都会通过 DefaultBeanDefinitionDocumentReader 解析成 BeanDefinition对象，并且 加载到 spring容器中（(List<String,BeanDefinition>)BeanDefinitionMap）.

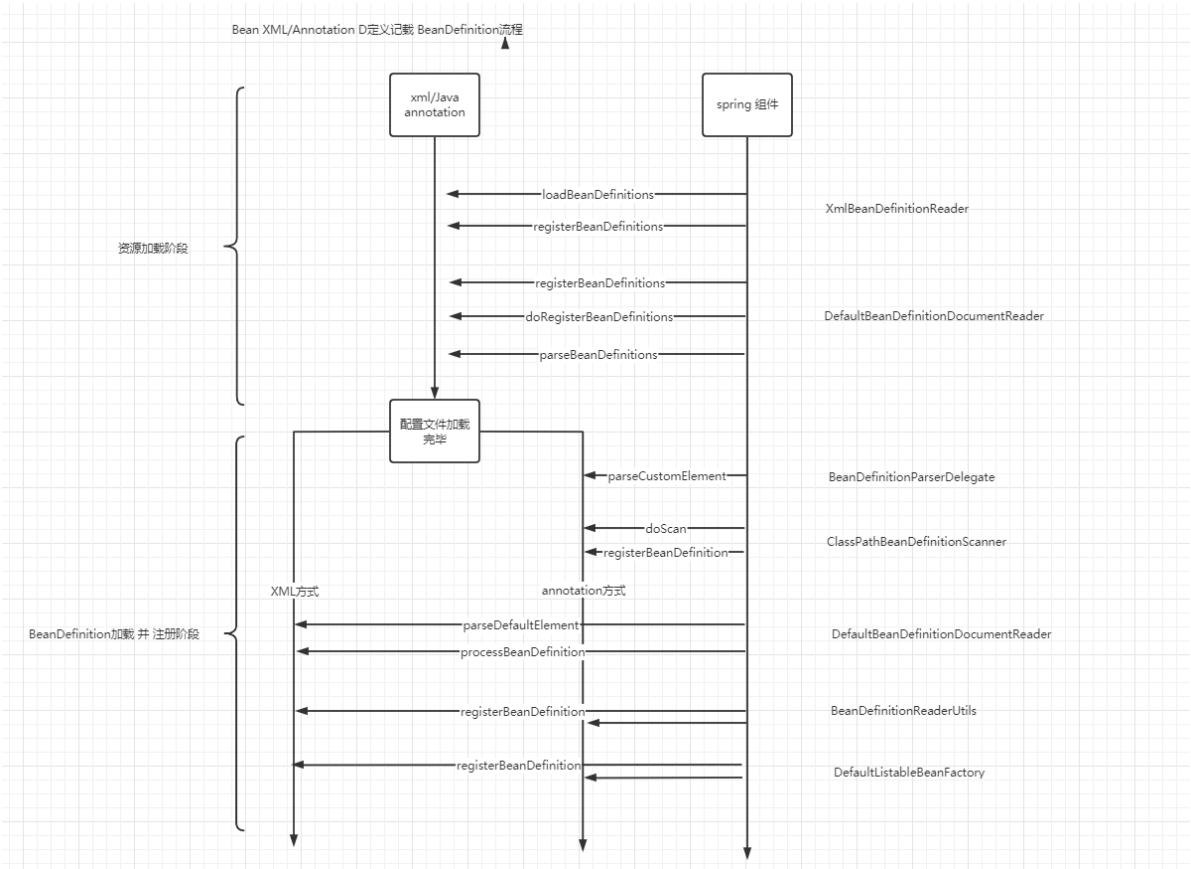
> 我们猜想 spring 仍然是否会通过 DefaultBeanDefinitionDocumentReader 来统一Javaconfig方式中BeanDefinition的注册过程，留给读者自行分析源码。

Java方式配置加载

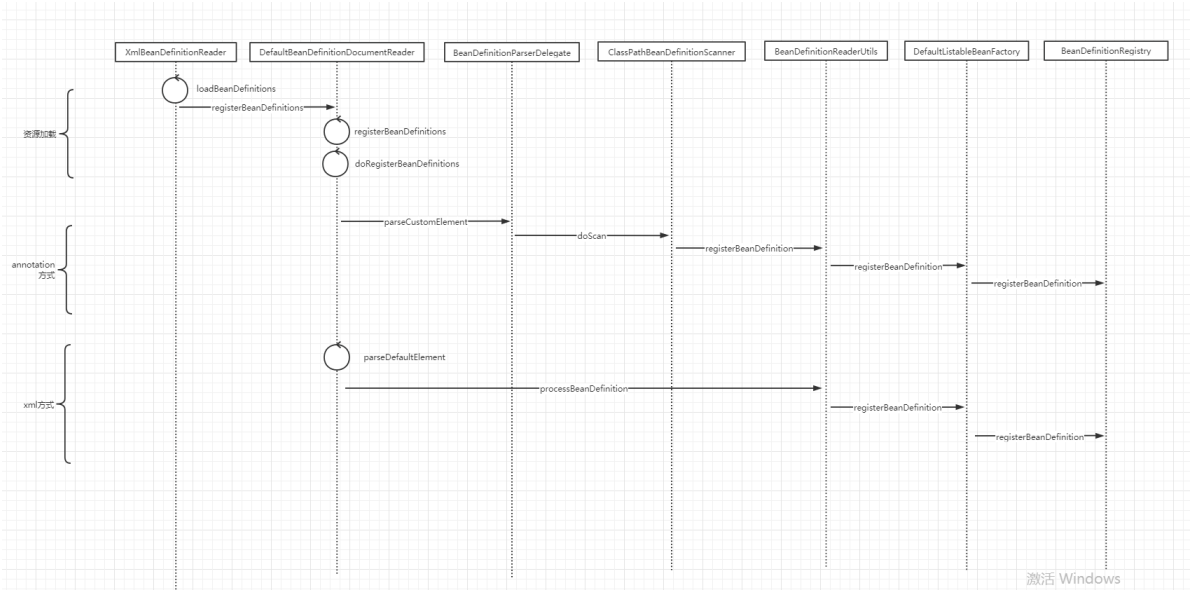
TODO

流程图 及 时序图展示 BeanDefinition 加载过程

> 流程图



> 时序图



Bean instantiate 实例化

> BeanDefinition 装配完之后，Bean的实例化就有了完整的 元数据信息，之后的工作就是根据 beanDefinition 来实例化 spring bean ，并且让spring 容器管理起来。
>
> 通过我们第一章对 BeanFactory 生命周期的探索过程，我们清楚，在BeanFactory容器中，bean 的实例化是在第一次调用该Bean时生成的，并且是在 BeanFactory 中的核心方法 org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#doCreateBean 中进行的，有的放矢，我们直接从 doCreateBean 探索 spring bean 的实例化进程。
>
> 下面我们结合源码来探究。

核心组件

- org.springframework.beans.BeanWrapper

  > Bean 的实体包装类

- org.springframework.beans.BeanUtils

  > 核心工具类：实例化bean，检查bean属性类型，复制Bean属性

边缘组件

Bean 实例化过程原理剖析

> Bean 的实例化过程非常复杂，我们通过三个核心方法 加以探究。分别是
>
> org.springframework.beans.factory.support.AbstractBeanFactory#getMergedLocalBeanDefinition -- 获取BeanDefinition对象
>
> org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#createBeanInstance -- 实例化Bean对象
>
> org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#populateBean -- 装填 Bean 对象的属性 及 依赖

>

```
//
org.springframework.beans.factory.support.AbstractBeanFactory#getMergedLocal
BeanDefinition
// 获取BeanDefinition对象
```

通过 DefaultListableBeanFactory#getBeanDefinition 方法里的这段代码，我们能发现用来生成 BeanInstance 的 BeanDefinition 是从 BeanDefinitionMap 容器中取出的。

```
BeanDefinition bd = this.beanDefinitionMap.get(beanName);
```

TIP:自此，用于生成 Bean 实例的 beanDefinition 已经获取到了

```
//
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
#createBeanInstance
// 实例化Bean对象
```

1. 特殊入口处理 ： 如果是FactoryBean方式生成的对象，则直接用工厂方法来实例化对象（重点方法，后续探讨，先不做展开）

```
        if (mbd.getFactoryMethodName() != null)  {
            return instantiateUsingFactoryMethod(beanName, mbd, args);
        }
```

2. 进入 instantiateBean(String beanName, RootBeanDefinition mbd) 方法

3. 执行 BeanUtils#instantiateClass(Constructor<T>, Object...) 方法
   (Constructor<T>) ctor.newInstance(args); 实例化完成

4. 是用BeanWrapper 包装 Bean对象

```
    BeanWrapper bw = new BeanWrapperImpl(beanInstance);
```

5. 注册属性编辑器（重点方法，后续探讨，先不做展开）

```
    protected void initBeanWrapper(BeanWrapper bw) {
        bw.setConversionService(getConversionService());
        registerCustomEditors(bw);
    }
```

TIP: 到这里，Bean的实例化对象已经通过反射的方式创建成功，其返回的对象有两种类型，一种是普通的 beanDefinition 来生成的Bean实例，另一种是通过 FactoryBean 工厂方法来生成的Bean实例

```
//
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory
#populateBean
// 装填 Bean 对象的属性 及 依赖
```

依赖注入

疑问梳理

- 在 Java config 配置模式中，具体的 Bean 对象是如何 加入到 spring 容器中的?
- 在 spring 容器中 对一个 class 定义多个Bean会怎样？spring 会如何处理?

面试题精讲

> 一。spring bean 的循环依赖是什么？怎么产生的？如何解决?

总结归纳