# 图解 Spring 核心数据结构：BeanDefinition

使用 Spring 时，尤其是使用 XML 配置的时候，也许我们会这样的问题：

1. Bean 怎么表示？
2. Bean 的依赖怎么表示？
3. init-method 方法怎么存储？
4. Bean 的一些属性，比如 lazy-init 等，怎么表示？
5. Bean 构造函数的参数怎么存储？

# 继承体系

* AttributeAccessor: 提供对 BeanDefinition 属性操作能力。
* AttributeAccessorSupport: 使用了 Map 进行属性的存储的。
* BeanMetadataAttributeAccessor: 代表了一个 Bean 元数据的属性操作。
* BeanMetadataElement: BeanDefinition 元数据，返回该 Bean 的来源。
* BeanDefinition: 用来描述 Bean，里面存放 Bean 元数据，比如 Bean 类名、scope、属性、构造函数参数列表、依赖的 Bean、是否是单例类、是否是懒加载等一些列信息。
* AbstractBeanDefinition: 抽象类统一实现了 BeanDefinition 定义的一部分操作，可以说是定义了 BeanDefinition 很多默认的属性。
* RootBeanDefinition: 代表一个 XML，Java Config来的 BeanDefinition。
* AnnotatedBeanDefinition: 表示注解类型 BeanDefinition。有两个重要的属性：AnnotationMetadata、MethodMetadata 分别表示 BeanDefinition 的注解元信息和方法元信息。实现了此接口的 BeanDefinition 可以获取到注解元数据和方法元数据。
* ChildBeanDefinition: 可以让子 BeanDefinition 定义拥有从父母那里继承配置的能力。
* GenericBeanDefinition: 是 Spring 2.5 之后才有的，这个的想法是用来替代 RootBeanDefinition/ChildBeanDefinition，而 RootBeanDefinition/ChildBeanDefinition 可以在 Spring 预加载的时候使用。
* AnnotatedGenericBeanDefinition: 表示 @Configuration 注解注释的 BeanDefinition 类。是 AnnotatedBeanDefinition 的一个具体实现。传入指定类后，可以获取类中的注解。
* ScannedGenericBeanDefinition: 表示 @Component、@Service、@Controller 等注解注释的 Bean 类。是 AnnotatedBeanDefinition 的另一个实现，与 AnnotatedGenericBeanDefinition 不同的是，ScannedGenericBeanDefinition 是通过扫描 class，然后操作 ASM 进行解析的。

# 核心属性

下面主要介绍一下它的核心内部属性：

1. Map<String, Object> attributes = new LinkedHashMap<>()：配置的属性以及属性值。
2. Object source：存储 Bean 来源，有时是 XML Element 对象。还可以是其他对象。
3. Object beanClass：Bean 的类型定义，有时是 Class 类型的对象；有时是类的全限定名，此时就是 String 类型。
4. abstractFlag = false：默认为 false。如果为 true，则表示不打算实例化该 Bean，仅仅作为其他 Bean 的父 Bean。一般与 parent 一起使用，设置 abstract 的 Bean 定义不需要创建实例，仅仅作为 parent 来进行一些通用的配置，后面的 Bean 通过设置 parent 来获取相应的配置信息，从而达到简化配置的目的。
5. Boolean lazyInit：是否懒加载。
6. int autowireMode = AUTOWIRE\_NO：注入模式，默认为 AUTOWIRE\_NO。一共有五个可选项：
   1. AUTOWIRE\_NO — 不自动注入。
   2. AUTOWIRE\_BY\_NAME — 根据名称自动注入。
   3. AUTOWIRE\_BY\_TYPE — 根据类型自动注入。
   4. AUTOWIRE\_CONSTRUCTOR — 自动根据构造函数注入。
   5. AUTOWIRE\_AUTODETECT — 自动检测。
7. int dependencyCheck = DEPENDENCY\_CHECK\_NONE：依赖检测。一共有四个可选项：
   1. DEPENDENCY\_CHECK\_NONE — 不进行依赖检测。
   2. DEPENDENCY\_CHECK\_OBJECTS — 只检测对象引用。
   3. DEPENDENCY\_CHECK\_SIMPLE — 只检测简单对象：基础类型、Enum、CharSequence、Number、Date、Temporal、URI、URL、Locale、Class 以及这些类型的数组对象。
   4. DEPENDENCY\_CHECK\_ALL — 检测所有依赖。
8. String[] dependsOn： 这个属性在 spring-beans.xsd / xsd:attribute / depends-on 中有说明。原文是这样说的：*The names of the beans that this bean depends on being initialized. The bean factory will guarantee that these beans get initialized before this bean.* 所以，这个属性并不是一个“需要注入的依赖属性”，而是 Bean 创建的前后依赖关系。这一点可能跟大多数人的认识不一样。

Bean 属性的依赖存在 CommonAnnotationBeanPostProcessor#injectionMetadataCache 和 AutowiredAnnotationBeanPostProcessor#injectionMetadataCache 属性中。在 [**Spring Bean 生命周期概述**](https://www.diguage.com/post/spring-bean-lifecycle-overview/#post-process-merged-bean-definition) 中描述的关于属性的依赖读取和注入也是靠这个两个属性来保存依赖关系的。

在 [**深入剖析 Spring 核心数据结构：BeanFactory : Map<String, Set<String>> dependentBeanMap**](https://www.diguage.com/post/dive-into-spring-core-data-structure-bean-factory/#dependent-bean-map) 中提到的 DefaultSingletonBeanRegistry#dependentBeanMap 和 DefaultSingletonBeanRegistry#dependenciesForBeanMap 两个属性，保存了这个 dependsOn 依赖关系的正向关系和反向关系。当工厂销毁时，也会通过 dependentBeanMap 属性，先销毁依赖的 Bean，然后再销毁自身。

1. boolean autowireCandidate = true：声明是否是其他依赖的候选 Bean；只会影响根据类型注入的情况，不会影响根据名称明确指明依赖的情况。
2. boolean primary = false：是否是首选 Bean，标注了 @Primary 则为 true。当 A 类型的 Bean 需要注入 B 类型的实现类，并且 B 类型的实现类有多个，在按类型将 B 的实现类注入到 A 中时，优先注入该属性为 true 的实现类，当然如果同一个类的实现类有多个 primary 为 true，则抛出异常。
3. Map<String, AutowireCandidateQualifier> qualifiers = new LinkedHashMap<>()：
4. Supplier<?> instanceSupplier：产生对象的生产者。
5. boolean nonPublicAccessAllowed = true：是否允许访问非 public 的构造器和方法。
6. boolean lenientConstructorResolution = true：是否采用宽容模式来解析构造函数。如果是 false，则只要参数类型不匹配就抛出异常。
7. String factoryBeanName：当 Bean 的创建方式是以工厂进行创建的时候，该方法设置工厂的名称。
8. String factoryMethodName：工厂创建 Bean 时，设置创建 Bean 的方法名称。
9. ConstructorArgumentValues constructorArgumentValues：构造函数参数值。
10. MutablePropertyValues propertyValues： 获取类的属性和属性值的类 PropertyValue。
11. MethodOverrides methodOverrides = new MethodOverrides()：
12. String initMethodName：初始化方法名，对应 init-method 或者 @PostConstruct 标注的方法。
13. String destroyMethodName：销毁方法名，对应 ` 或 `@PreDestroy 标注的方法。
14. boolean enforceInitMethod = true：强制初始化方法，默认是 false。如果为 true，而且 initMethodName 为空，则报错。
15. boolean enforceDestroyMethod = true：强制销毁方法，默认是 false。如果为 true，而且 destroyMethodName 为空，则报错。
16. boolean synthetic = false：是否是合成的。
17. int role = BeanDefinition.ROLE\_APPLICATION：Bean 角色。可选项有三个：
    1. ROLE\_APPLICATION — 为应用程序定义。
    2. ROLE\_SUPPORT — 为应用程序定义的比较大的对象。
    3. ROLE\_INFRASTRUCTURE — 内部定义的基础 Bean。
18. String description：Bean 描述。
19. Resource resource：Bean 的来源 Resource 对象。
20. AnnotationMetadata metadata： 注解元信息。
21. MethodMetadata factoryMethodMetadata：方法注解元信息。

BeanDefinition 的代码在 [**spring-framework/BeanDefinition.java**](https://github.com/spring-projects/spring-framework/blob/master/spring-beans/src/main/java/org/springframework/beans/factory/config/BeanDefinition.java) 中。感兴趣，可以自己 clone 下来，把玩把玩。