链滴

Spring 源码教程笔记

作者：lingyundu

原文链接：https://ld246.com/article/1627073664208

来源网站：链滴

教程地址：

这可能是B站讲的最好的SPRING源码教程（2021年最新版）_哔哩哔哩

个人感觉该教程讲的简单易懂，仅包含前十节的笔记（其实是视频里的课件内容，略有修改 mile）。

01. 什么是 BeanDefinition

BeanDefinition 表示 Bean 的定义，Spring 根据 BeanDefinition 来创建 Bean 对象，BeanDefiniti n 有很多的属性用来描述 Bean，BeanDefinition 是 Spring 中的非常核心的概念。

BeanDefinition 中重要的属性：

- **beanClass** -- 表示一个 Bean 的类型，比如：UserService.class、OrderService.class，Spring 创建 Bean 的过程中会根据此属性来实例化得到对象。
- **scope** -- 表示一个 Bean 的作用域，比如：scope 等于 singleton，该 Bean 就是一个单例 Bean；cope 等于 prototype，该 Bean 就是一个原型 Bean。
- **isLazy** -- 表示一个 Bean 需不需要懒加载，原型 Bean 的 isLazy 属性不起作用，懒加载的单例 Bea ，会在第一次 getBean 的时候生成该 Bean，非懒加载的单例 Bean，则会在 Spring 启动过程中直 生成好。
- **dependsOn** -- 表示一个 Bean 在创建之前所依赖的其他 Bean，在一个 Bean 创建之前，它所依 的这些 Bean 得先全部创建好。
- **primary** -- 表示一个 Bean 是主 Bean，在 Spring 中一个类型可以有多个 Bean 对象，在进行依 注入时，如果根据类型找到了多个 Bean，此时会判断这些 Bean 中是否存在一个主 Bean，若存在， 直接将这个 Bean 注入给属性。
- **initMethodName** -- 表示一个 Bean 的初始化方法，一个 Bean 的生命周期过程中有一个步骤叫 始化，Spring 会在这个步骤中去调用 Bean 的初始化方法，初始化逻辑由程序员自己控制，表示程 员可以自定义逻辑对 Bean 进行加工。

我们平时开发中用到的 @Component 注解、@Bean 注解以及<bean/> 标签，这些都会解析为 Bea Definition 对象。

02. 什么是 BeanFactory

BeanFactory 是一种 “Spring 容器”，BeanFactory 翻译过来就是 Bean 工厂，顾名思义，它可以 来创建 Bean、获取 Bean，BeanFactory 是 Spring 中非常核心的组件。

BeanDefinition、BeanFactory 和 Bean对象三者之间的关系：

BeanFactory 将利用 BeanDefinition 来生成 Bean 对象，BeanDefinition 相当于 BeanFactory 的 纸（~~原材料——~~），Bean 对象就相当于 BeanFactory 所生产出来的产品。

BeanFactory 的核心子接口和实现类：

- ListableBeanFactory
- ConfigurableBeanFactory
- AutowireCapableBeanFactory
- AbstractBeanFactory

● DefaultListableBeanFactory -- 最重要，支持单例 Bean、支持 Bean 别名、支持父子 BeanFactor、支持 Bean 类型转化、支持 Bean 后置处理、支持 FactoryBean、支持自动装配，等等。

03. 什么是 Bean 生命周期

Bean 生命周期描述的是 Spring 中一个Bean 创建过程和销毁过程中所经历的步骤，其中 Bean 创建程是重点。

程序员可以利用 Bean 生命周期机制对 Bean 进行自定义加工。

Bean 声明周期的核心步骤:

1. **创建 BeanDefinition 对象** -- BeanDefinition 表示 Bean 定义，它定义了某个 Bean 的类型，Spring 就是利用 BeanDefinition 来创建 Bean 的，比如需要利用 BeanDefinition 中的 beanClass 属性确定 Bean 的类型，从而实例化出来对象。

2. **构造方法推断** -- 一个 Bean 中可以有多个构造方法，此时就需要 Spring 来判断到底使用哪个构造法，这个过程比较复杂。推断并确定一个构造方法后，就可以利用构造方法实例化得到一个对象了。

3. **实例化** -- 通过构造方法反射得到一个实例化对象，在 Spring 中，可以通过 BeanPostProcessor 制对实例化进行干预。

4. **属性填充** -- 实例化所得到的对象，是“不完整”的对象，“不完整”的意思是该对象中的某些属还没有进行属性填充，也就是 Spring 还没有自动给某些属性赋值，属性填充就是我们通常说的自动入、依赖注入。

5. **初始化** -- 在一个对象的属性填充之后，Spring 提供了初始化机制，程序员可以利用初始化机制对ean 进行自定义加工，比如可以利用 InitializingBean 接口来对 Bean 中的其他属性进行赋值，或对 Ban 中的某些属性进行校验。

6. **初始化后** -- 初始化后是 Bean 创建生命周期中最后一个步骤，我们常说的 AOP 机制就是在这个步中通过 BeanPostProcessor 机制实现的，初始化之后得到对象才是真正的 Bean 对象。

04. @QutoWaired 是如何工作的

@QutoWaired是什么?

@QutoWaired 表示某个属性是否需要进行依赖注入，可以写在属性和方法上。注解中的 required 性默认为 true，表示如果没有对象可以注入给属性则抛出异常。

```
@Service
public class OrderService {

    @Autowired
    private UserService userService;

    ......
}
```

@QutoWaired 加在某个属性上，Spring 在进行 Bean 的生命周期过程中，在属性填充这一步，会于实例化出来的对象，对该对象中加了 @QutoWaired 的属性自动给属性赋值。

Spring 会先根据属性的类型去 Spring 容器中找出该类型所有的 Bean 对象，如果找出来多个，则再据属性的名字从多个中再确定一个。如果 required 属性为 true，并且根据属性信息找不到对象，则接抛出异常。

```
@Service
public class OrderService {

    private UserService userService;

    @Autowired
    public void setUserService(UserService userService){
        this.userService = userService;
    }
}
```

当 @QutoWaired 注解写在某个方法上时，Spring 在 Bean 生命周期的属性填充阶段，会根据方法参数类型，参数名字从 Spring 容器中找到对象当做方法入参，自动放射调用该方法。

```
@Service
public class OrderService {

    private UserService userService;

    @Autowired
    public OrderService(UserService userService){
        this.userService = userService;
    }
}
```

当 @Autowired 注解加在构造方法上时，Spring 会在推断构造方法阶段，选择该构造方法来进行实化，在反射调用构造方法之前，会先根据构造方法参数类型、参数名从 Spring 容器中找到 Bean 对，当做构造方法入参。

05. @Resource 是如何工作的

@Resource是什么?

@Resource 注解和 AutoWired 注解类似，也是用来进行依赖注入的，@Resource 是 Java 层面所供的注解，Autowired 是 Spring 所提供的注解，它们依赖注入的底层实现逻辑也不同。

@Resource 注解中有一个 name 属性，针对 name 属性是否有值，@Resource 的依赖注入底层流是不同的。

- 如果 name 属性有值，那么 Spring 会直接根据所指定的 name 值去 Spring 容器找 Bean 对象，找到了则成功，否则就会报错。
- 如果 name 属性没有值，则：

1. 先判断该属性名字在 Spring 容器中是否存在 Bean 对象。
2. 若存在，则成功找到 Bean 对象进行注入。
3. 若不存在，则根据属性类型去 Spring 容器找 Bean 对象，找到一个则进行注入。

06. @Value 是如何工作的?

@Value 注解和 @Resource 注解、@Autowired 注解类似，也是用来对属性进行依赖注入的，只不 @Value 是用来从 Properties 文件中来获取值的，并且 @Value 可以解析 SpEL（Spring表达式）。

@Value("zhouyu")

直接将字符串 "zhouyu" 赋值给属性，如果属性类型不是 String，或无法进行类型转换，则报错。

@Value("${zhouyu}")

将会把 {} 中的字符串当做 key，从 Properties 文件中找出对应的 value 赋值给属性，如果没找到，会把 "{zhouyu}" 当做普通字符串注入给属性。

@Value("#{zhouyu}")

会将 #{} 中的字符串当做 Spring 表达式进行解析，Spring 会把 "zhouyu" 当做 beanName，并从 Spring 容器中找到对应的 Bean，如果找到则进行属性注入，没找到则报错。

07. 什么是 FactoryBean?

FactoryBean 是什么?

FactoryBean 是 Spring 所提供的一种较灵活的创建 Bean 的方式，可以通过实现 FactoryBean 接口的 getObject() 方法来返回一个对象，这个对象就是最终的 Bean 对象。

FactoryBean 接口中的方法

- Object getObject：返回的是 Bean 对象
- boolean isSingleton：返回的是 Bean 对象是否是单例
- Class getObjectType()：返回的是 Bean 对象的类型

示例:

```
@Component("zhouyu")
public class ZhouyuFactoryBean implements FactoryBean {
    @Override
    // Bean 对象
    public Object getObject() throws Exception {
        return new User();
    }

    @Override
    // Bean 对象的类型
    public Class<?> getObjectType() {
        return User.class;
    }

    @Override
    // 定义的 Bean 对象是单例还是原型
    public boolean isSingleton() {
        return true;
    }
}
```

上述代码，实际上对应了两个 Bean 对象：

1. beanName 为 "zhouyu"，Bean 对象为 getObject 方法所返回的 User 对象。

2. beanName 为 "@zhouyu"，Bean 对象为 ZhouyuFactoryBean 类的实例对象。

FactoryBean 对象本身也是一个 Bean，同时它相当于一个小型工厂，可以生产出另外的 Bean。

BeanFactory 是 Spring 容器，是一个大型的工厂，它可以生产出各种各样的 Bean。

FactoryBean 机制被广泛的应用在 Spring 内部和 Spring 与第三方框架或组件的整合过程中。

08. 什么是 ApplicationContext?

ApplicationContext 是什么?

ApplicationContext 是比 BeanFactory 更加强大的 Spring 容器，它既可以创建 Bean、获取 Bean 还支持国际化、事件广播、获取资源等 BeanFactory 所不具备的功能。

ApplicationContext 所继承的接口

- **EnvironmentCapable** -- ApplicationContext 继承了这个接口，表示拥有了获取环境变量的功能可以通过 ApplicationContext 获取操作系统环境变量和 JVM 环境变量。
- **ListableBeanFactory** -- ApplicationContext 继承了这个接口，就拥有了获取所有 beanNames 判断某个 beanName 是否存在 BeanDefinition 对象、统计 BeanDefinition 个数、获取某个类型对的所有 beanName 等功能。
- **HierarchicalBeanFactory** -- ApplicationContext 继承了这个接口，就拥有了获取父 BeanFactor 、判断某个 name 是否存在 Bean 对象的功能。
- **MessageSource** -- ApplicationContext 继承了这个接口，就拥有了国际化功能，比如可以直接用 MessageSource 对象获取某个国际化资源（比如不同国家语言所对应的字符）。
- **ApplicationEventPublisher** -- ApplicationContext 继承了这个接口，就拥有了事件发布功能，以发布事件，这是 ApplicationContext 相对于 BeanFactory 比较突出、常用的功能。
- **ResourcePatternResolver** -- ApplicationContext 继承了这个接口，就拥有了加载并获取资源功能，这里的资源可以是文件，图片等某个 URL 资源都可以。

09. 什么是 BeanPostProcessor?

BeanPostProcessor 是什么?

BeanPostProcessor 是 Spring 所提供的的一种扩展机制，可以利用该机制对 Bean 进行定制化加工在 Spring 底层源码实现中，也广泛的用到了该机制，BeanPostProcessor 通常也叫做 Bean 后置处器。

BeanPostProcessor 在 Spring 中是一个接口，我们定义一个后置处理器，就是提供一个类实现该接，在 Spring 中还存在一些接口继承了 BeanPostProcessor，这些子接口是在 BeanPostProcessor 基础上增加了一些其他的功能。

BeanPostProcessor 中的方法

- postProcessBeforeInitialization()：初始化前方法，表示可以利用这个方法来对 Bean 在初始化前行自定义加工。
- postProcessAfterInitialization()：初始化前方法，表示可以利用这个方法来对 Bean 在初始化后行自定义加工。

InstantiationAwareBeanPostProcessor

是 BeanPostProcessor 的一个子接口，它包含三个方法：

- postProcessBeforeInstantiation()：实例化前
- postProcessAfterInstantiation()：实例化后
- postProcessProperties()：属性注入后

10. AOP 是如何工作的?

AOP 是什么?

AOP 就是面向切面编程，是一种非常适合在无需修改业务代码的前提下，对某个或某些业务增加统的功能，比如日志记录、权限控制、事务管理等，能很好的是代码解耦，提高开发效率。

AOP 中的核心概念

- **Advice** -- 可以理解为通知、建议，在 Spring 中通过定义 Advice 来定义代理逻辑。
- **Pointcut** -- 是切点，表示 Advice 对应的代理逻辑应用在哪个类、哪个方法上。
- **Advisor** -- 等于 Advice+Pointcut，表示代理逻辑和切点的一个整体，程序员可以通过定义或者装一个 Advisor，来定义切点和代理逻辑。
- **Weaving** -- 表示织入，将 Advice 代理逻辑在源代码级别嵌入到切点的过程，就叫做织入。
- **Target** -- 表示目标对象，也就是被代理对象，在 AOP 生成的代理对象中会持有目标对象。
- **Join Point** -- 表示连接点，在 Spring AOP 中，就是方法的执行点。

AOP 的工作原理

AOP 是发生在 Bean 的生命周期过程中的：

1. Spring 生成 Bean 对象时，先实例化出来的一个对象，也就是 target 对象。
2. 再对 target 对象进行属性填充。
3. 在初始化后步骤中，会判断 target 对象有没有对应的切面。
4. 如果有切面，就表示当前 target 对象需要进行 AOP。
5. 通过 Cglib 或 JDK 动态代理机制生成一个代理对象，作为最终的 Bean 对象。
6. 代理对象中有一个 target 属性指向了 target 对象。