```
            this.beanDefinitionMap.put(beanName, beanDefinition);
            this.beanDefinitionNames.add(beanName);
            this.manualSingletonNames.remove(beanName);
        }
        this.frozenBeanDefinitionNames = null;
    }

    //检查是否已经注册过同名的 BeanDefinition
    if (oldBeanDefinition != null || containsSingleton(beanName)) {
        //重置所有已经注册过的 BeanDefinition 的缓存
        resetBeanDefinition(beanName);
    }
}
```

至此，Bean 配置信息中配置的 Bean 被解析后已经注册到 Spring IoC 容器中，被容器管理起来，真正完成了 Spring IoC 容器初始化的全部工作。现在 Spring IoC 容器中已经建立了所有 Bean 的配置信息，Bean 定义信息已经可以使用，并且可以被检索。Spring IoC 容器的作用就是对这些注册的 Bean 定义信息进行处理和维护。注册的 Bean 定义信息是 Spring IoC 容器控制反转的基础，正是有了这些信息，容器才可以进行依赖注入。

8.3 基于注解的 IoC 初始化

8.3.1 注解的前世今生

在 Spring 2.0 以后的版本中，引入了基于注解（Annotation）方式的配置，注解（Annotation）是 JDK 1.5 引入的一个新特性，用于简化 Bean 的配置，可以取代 XML 配置文件。开发人员对注解的态度也是萝卜青菜各有所爱，个人认为注解可以大大简化配置，提高开发速度，但也给后期维护增加了难度。目前来说，XML 方式相对成熟，便于统一管理。随着 Spring Boot 的兴起，基于注解的开发甚至实现了零配置。但作为个人的习惯，我还是倾向于 XML 配置文件和注解相互配合使用。Spring IoC 容器对于类级别的注解和类内部的注解处理策略如下。

（1）**类级别的注解：**如@Component、@Repository、@Controller、@Service，以及 Java EE 6 的@ManagedBean 和@Named，都是添加在类上的类级别注解，Spring IoC 容器根据注解的过滤规则扫描读取注解 Bean 定义类，并将其注册到 Spring IoC 容器中。

（2）**类内部的注解：**如@Autowire、@Value、@Resource，以及 EJB 和 WebService 相关的注解等，都是添加在类内部的字段或者方法上的类内部注解，Spring IoC 容器通过 Bean 后置注解处理器解析 Bean 内部的注解。