15 Spring IoC总结及热点面试题集萃

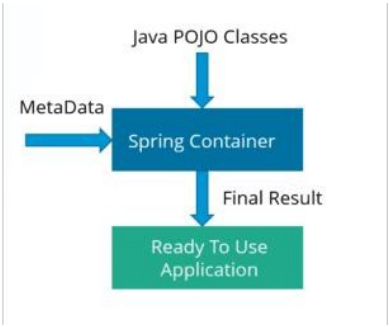
更新时间：2020-08-04 14:03:32

> 最聪明的人是最不愿浪费时间的人。——但丁

Spring 最基础的部分就是 IoC，对 IoC 的理解程度从某个方面代表着你对 Spring 的理解程度。下面是从网络上搜集的 Spring IoC 面试时碰到的热点问题及参考答案。

1. 什么是 Spring IoC 容器

Spring 框架的核心是 Spring 容器。容器创建对象，将它们装配在一起，配置它们并管理它们的完整生命周期。Spring 容器使用依赖注入来管理组成应用程序的组件。容器通过读取提供的配置元数据来接收对象进行实例化，配置和组装的指令。该元数据可以通过 XML，Java 注解或 Java 代码提供。



2. 什么是依赖注入

在依赖注入中，您不必创建对象，但必须描述如何创建它们。您不是直接在代码中将组件和服务连接在一起，而是描述配置文件中哪些组件需要哪些服务。由 IoC 容器将它们装配在一起。

3. 可以通过多少种方式完成依赖注入

通常，依赖注入可以通过 3 种方式完成，即：

- 构造函数注入
- setter 注入
- 接口注入

在 Spring Framework 中，仅使用构造函数和 setter 注入。

4. 区分构造函数注入和 setter 注入

构造函数注入	setter 注入
没有部分注入	有部分注入
不会覆盖 setter 属性	会覆盖 setter 属性
任意修改都会创建一个新实例	任意修改不会创建一个新实例
适用于设置很多属性	适用于设置少量属性

5. spring 中有哪些类型的 IoC 容器

BeanFactory：BeanFactory 就像一个包含 Bean 集合的工厂类。它会在客户端要求时实例化 bean；

ApplicationContext： ApplicationContext 接口扩展了 BeanFactory 接口。它在 BeanFactory 基础上提供了一些额外的功能。

6. 区分 BeanFactory 和 ApplicationContext

ApplicationContext 提供了解析文字消息的功能, 包括对国际化 i18n 消息的支持；

ApplicationContext 提供了更通用的加载文件资源的方式，如加载图片；

ApplicationContext 可以想注册的 Bean 发送事件消息；

BeanFactory 适合硬编码或者简单的实现，ApplicationContext 适合大工程的管理；

ApplicationContext 实现了 MessageSource 接口，整个接口用来获取本地化消息，这个接口的实现是可插拔的。

7. 列举 IoC 的一些好处

IoC 的一些好处是：

- 它将最小化应用程序中的代码量；
- 它将使您的应用程序易于测试，因为它不需要单元测试用例中的任何单例或 JNDI 找机制；
- 它以最小的影响和最少的侵入机制促进松耦合；
- 它支持即时的实例化和延迟加载服务。

8. Spring IoC 的实现机制

Spring 中的 IoC 的实现原理就是工厂模式加反射机制。

9. 什么是 spring bean

- 它们是构成用户应用程序主干的对象；
- Bean 由 Spring IoC 容器管理；
- 它们由 Spring IoC 容器实例化，配置，装配和管理；
- Bean 是基于用户提供给容器的配置元数据创建。

10. spring 提供了哪些配置方式

10.1 基于 xml 配置

Bean 所需的依赖项和服务在 XML 格式的配置文件中指定。这些配置文件通常包含许多 Bean 定义和特定于应用程序的配置选项。它们通常以 Bean 标签开头。例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

        <bean id="customerBean" class="com.davidwang456.test.Customer">
            <property name="mapA" value="#{testBean.map['MapA']}" />
            <property name="list" value="#{testBean.list[0]}" />
        </bean>

        <bean id="testBean" class="com.mkyong.core.Test" />
</beans>
```

10.2 基于注解配置

您可以通过在相关的类，方法或字段声明上使用注解，将 Bean 配置为组件类本身，而不是使用 XML 来描述 Bean 装配。默认情况下，Spring 容器中未打开注解装配。因此，您需要在使用它之前在 Spring 配置文件中启用它。例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="com.davidwang456.test" />
</beans>
```

```java
package com.davidwang456.test;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.stereotype.Component;

@Component("testBean")
public class Test {

    private Map<String, String> map;
    private List<String> list;

    public Test() {
        map = new HashMap<String, String>();
        map.put("MapA", "This is A");
        map.put("MapB", "This is B");
        map.put("MapC", "This is C");

        list = new ArrayList<String>();
        list.add("List0");
        list.add("List1");
        list.add("List2");

    }

    public Map<String, String> getMap() {
        return map;
    }

    public void setMap(Map<String, String> map) {
        this.map = map;
    }

    public List<String> getList() {
        return list;
    }

    public void setList(List<String> list) {
        this.list = list;
    }

}
```

10.3 基于 **Java API** 配置

Spring 的 Java 配置是通过使用 @Bean 和 @Configuration 来实现。

```java
package com.example.springboot;

import java.util.Arrays;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class Application {

        public static void main(String[] args) {
                SpringApplication.run(Application.class, args);
        }

        @Bean
        public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
                return args -> {

                        System.out.println("Let's inspect the beans provided by Spring
Boot:");

                        String[] beanNames = ctx.getBeanDefinitionNames();
                        Arrays.sort(beanNames);
                        for (String beanName : beanNames) {
                                System.out.println(beanName);
                        }

                };
        }

}
```

@Bean 注解扮演与 `<bean />` 元素相同的角色。

@Configuration 类允许通过简单地调用同一个类中的其他 @Bean 方法来定义 Bean 间依赖关系。

例如：

```
public class StudentConfig {

 @Bean
 public StudentBean myStudent() {
  return new StudentBean();
 }

}
```

11. Spring 支持几种 Bean scope

Spring Bean 支持 5 种 scope：

Singleton：每个 Spring IoC 容器仅有一个单实例；

Prototype：每次请求都会产生一个新的实例；

Request：每一次 HTTP 请求都会产生一个新的实例，并且该 Bean 仅在当前 HTTP 请求内有效；

Session：每一次 HTTP 请求都会产生一个新的 Bean，同时该 Bean 仅在当前 HTTP Session 内有效；

Global Session：类似于标准的 HTTP Session 作用域，不过它仅仅在基于 portlet 的 Web 应用中才有意义。Portlet 规范定义了全局 Session 的概念，它被所有构成某个 portlet web 应用的各种不同的 portlet 所共享。在 Global Session 作用域中定义的 Bean 被限定于全局 portlet Session 的生命周期范围内。如果你在 Web 中使用 Global Session 作用域来标识 Bean，那么 Web 会自动当成 Session 类型来使用。

仅当用户使用支持 Web 的 ApplicationContext 时，最后三个才可用。

12. spring bean 容器的生命周期是什么样的

Spring Bean 容器的生命周期流程如下：

Spring 容器根据配置中的 Bean 定义实例化 Bean；

Spring 使用依赖注入填充所有属性，如 Bean 中所定义的配置；

如果 Bean 实现 BeanNameAware 接口，则工厂通过传递 Bean 的 ID 来调用 setBeanName()；

如果 Bean 实现 BeanFactoryAware 接口，工厂通过传递自身的实例来调用 setBeanFactory()；

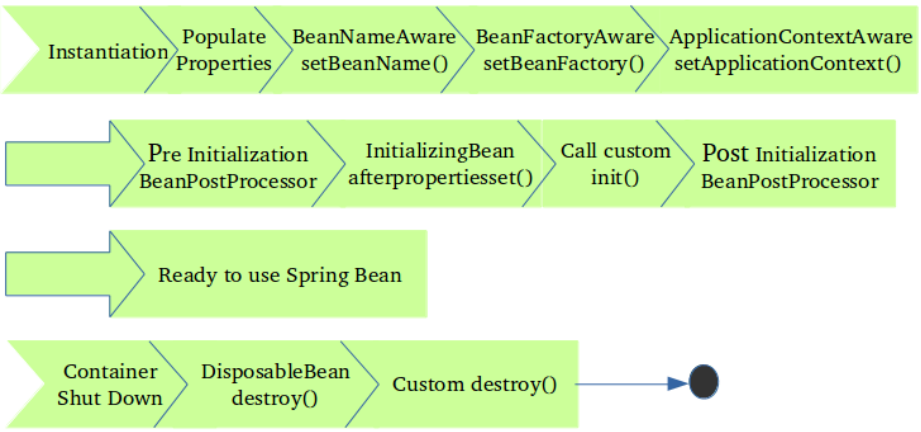
如果存在与 Bean 关联的任何 BeanPostProcessors，则调用 preProcessBeforeInitialization() 方法；

如果为 Bean 指定了 init 方法（ `<bean>` 的 init-method 属性），那么将调用它；

最后，如果存在与 bean 关联的任何 BeanPostProcessors，则将调用 postProcessAfterInitialization() 方法；

如果 Bean 实现 DisposableBean 接口，当 spring 容器关闭时，会调用 destory()；

如果为 Bean 指定了 destroy 方法（ `<bean>` 的 destroy-method 属性），那么将调用它。



13. 什么是 Spring 的内部 Bean

只有将 Bean 用作另一个 Bean 的属性时，才能将 Bean 声明为内部 Bean。为了定义 Bean，Spring 的基于 XML 的配置元数据在 <property> 或 constructor-arg> 中提供了 <bean> 元素的使用。内部 Bean 总是匿名的，它们总是作为原型。

例如，假设我们有一个 Student 类，其中引用了 Person 类。这里我们将只创建一个 Person 类实例并在 Student 中使用它。

Student.java

```
public class Student {
    private Person person;
    //Setters and Getters
}
public class Person {
    private String name;
    private String address;
    //Setters and Getters
}
```

Student.java

bean.xml

```
<bean id="StudentBean" class="com.edureka.Student">
        <property name="person">
            <!—This is inner bean —>
            <bean class="com.edureka.Person">
                <property name="name" value="Scott"></property>
                <property name="address" value="Bangalore"></property>
            </bean>
        </property>
    </bean>
```

bean.xml

14. 什么是 Spring 装配

当 Bean 在 Spring 容器中组合在一起时，它被称为装配或 Bean 装配。 Spring 容器需要知道需要什么 Bean 以及容器应该如何使用依赖注入来将 Bean 绑定在一起，同时装配 Bean。

15. 自动装配有哪些方式

Spring 容器能够自动装配 Bean。也就是说，可以通过检查 BeanFactory 的内容让 Spring 自动解析 Bean 的协作者。

自动装配的不同模式：

no： 这是默认设置，表示没有自动装配。应使用显式 Bean 引用进行装配；

byName： 它根据 bean 的名称注入对象依赖项，它匹配并装配其属性与 XML 文件中由相同名称定义的 bean；

byType： 它根据类型注入对象依赖项，如果属性的类型与 XML 文件中的一个 Bean 名称匹配，则匹配并装配属性；

构造函数： 它通过调用类的构造函数来注入依赖项，它有大量的参数；

autodetect： 首先容器尝试通过构造函数使用 autowire 装配，如果不能，则尝试通过 byType 自动装配。

16. 自动装配有什么局限

覆盖的可能性：您始终可以使用 `<constructor-arg>` 和 `<property>` 设置指定依赖项，这将覆盖自动装配；

基本元数据类型：简单属性（如原数据类型，字符串和类）无法自动装配；

令人困惑的性质：总是喜欢使用明确的装配，因为自动装配不太精确。

17. 你用过哪些重要的 Spring 注解

@Controller：用于 Spring MVC 项目中的控制器类；

@Service：用于服务类；

@RequestMapping：用于在控制器处理程序方法中配置 URI 映射；

@ResponseBody：用于发送 Object 作为响应，通常用于发送 XML 或 JSON 数据作为响应；

@PathVariable：用于将动态值从 URI 映射到处理程序方法参数；

@Autowired：用于在 Spring Bean 中自动装配依赖项；

@Qualifier：使用 @Autowired 注解，以避免在存在多个 Bean 类型实例时出现混淆；

@Scope：用于配置 Spring Bean 的范围；

@Configuration、**@ComponentScan** 和 **@Bean**：用于基于 Java 的配置；

@Aspect、**@Before**、**@After**、**@Around**、**@Pointcut**：用于切面编程（AOP）。

18. 如何在 spring 中启动注解装配

默认情况下，Spring 容器中未打开注解装配。因此，要使用基于注解装配，我们必须通过配置 <context：annotation-config /> 元素在 Spring 配置文件中启用它。

19. @Component、@Controller、@Repository、 @Service 有何区别

@Component：这将 Java 类标记为 Bean。它是任何 Spring 管理组件的通用构造型。Spring 的组件扫描机制现在可以将其拾取并将其拉入应用程序环境中；

@Controller：这将一个类标记为 Spring Web MVC 控制器。标有它的 Bean 会自动导入到 IoC 容器中；

@Service：此注解是组件注解的特化。它不会对 @Component 注解提供任何其他行为。您可以在服务层类中使用 @Service 而不是 @Component，因为它以更好的方式指定了意图；

@Repository：这个注解是具有类似用途和功能的 @Component 注解的特化。它为 DAO 提供了额外的好处。它将 DAO 导入 IoC 容器，并使未经检查的异常有资格转换为 Spring DataAccessException。

20. @Required 注解有什么用

@Required 应用于 Bean 属性 setter 方法。此注解仅指示必须在配置时使用 Bean 定义中的显式属性值或使用自动装配填充受影响的 Bean 属性。如果尚未填充受影响的 Bean 属性，则容器将抛出 BeanInitializationException。

示例：

```
public class Employee {

 private String name;

 @Required
 public void setName(String name){
  this.name=name;
 }

 public string getName(){
  return name;
 }
}
```

21. @Autowired 注解有什么用

@Autowired 可以更准确地控制应该在何处以及如何进行自动装配。此注解用于在 setter 方法，构造函数，具有任意名称或多个参数的属性或方法上自动装配 bean。默认情况下，它是类型驱动的注入。

```
public class Employee {
 private String name;
 @Autowired
 public void setName(String name) {
  this.name=name;
 }
 public string getName(){
  return name;
 }
}
```

22. @Qualifier 注解有什么用

当您创建多个相同类型的 Bean 并希望仅使用属性装配其中一个 Bean 时，您可以使用 @Qualifier 注解和 @Autowired 通过指定应该装配哪个确切的 Bean 来消除歧义。

例如，这里我们分别有两个类，Employee 和 EmpAccount。在 EmpAccount 中，使用 @Qualifier 指定了必须装配 id 为 emp1 的 Bean。

Employee.java

```
public class Employee {
    private String name;
    @Autowired
    public void setName(String name)
        this.name=name;
    }
    public string getName() {
        return name;
    }
}
```

Employee.java

EmpAccount.java

```
public class EmpAccount {
    private Employee emp;

    @Autowired
    @Qualifier(emp1)
    public void showName() {
        System.out.println("Employee name : "+emp.getName);
    }
}
```

EmpAccount.java

23. autowire 的限制性

主要有以下几点：

可能会被覆盖：当使用 <constructor-arg> 或者 <property> 设置时会覆盖 autowire；

不支持主数据结构：不能 autowire 属性注入基础数据类型、字符串,和 Class；

天然容易混淆：Autowire 没有 wire 明显, 因此可能的话还是使用 wire。

24. 在 Spring 中可以注入 null 或者空字符串吗

可以。

25. Spring 支持的事件类型有哪些

Spring 提供了以下 5 种标准的事件：

（1）上下文更新事件（**ContextRefreshedEvent**）：在调用 ConfigurableApplicationContext 接口中的 refresh()方法时被触发；

（2）上下文开始事件（**ContextStartedEvent**）：当容器调用 ConfigurableApplicationContext 的 Start()方法开始/重新开始容器时触发该事件；

（3）上下文停止事件（**ContextStoppedEvent**）：当容器调用 ConfigurableApplicationContext 的 Stop()方法停止容器时触发该事件；

（4）上下文关闭事件（**ContextClosedEvent**）：当 ApplicationContext 被关闭时触发该事件。容器被关闭时，其管理的所有单例 Bean 都被销毁；

（5）请求处理事件（**RequestHandledEvent**）：在 Web 应用中，当一个 http 请求（request）结束触发该事件，

如果一个 Bean 实现了 ApplicationListener 接口，当一个 ApplicationEvent 被发布以后，Bean 会自动被通知。

总结

很多童鞋喜欢在面试前或者考试前临时抱佛脚，面试后又抱怨内容太多记不住，其实原因在于平时积累不够。

平时多积累，用时少流泪。积累的最好办法就是通过一个个实例不断深入源码内部，深刻理解其原理，这就是本专栏的目的。

}

14 离开了Spring，IoC容器还可以怎么做？

16 Resource的前生今世