47 Spring总结及常见面试问题与答案集锦

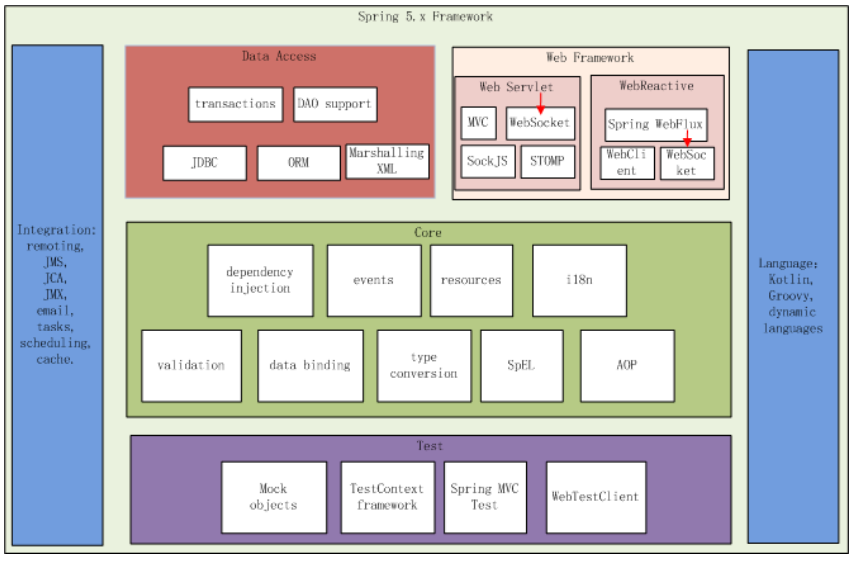
更新时间：2020-09-09 10:08:50

> 构成我们学习最大障碍的是已知的东西，而不是未知的东西。 —— 贝尔纳

Spring 总结

Spring 5架构

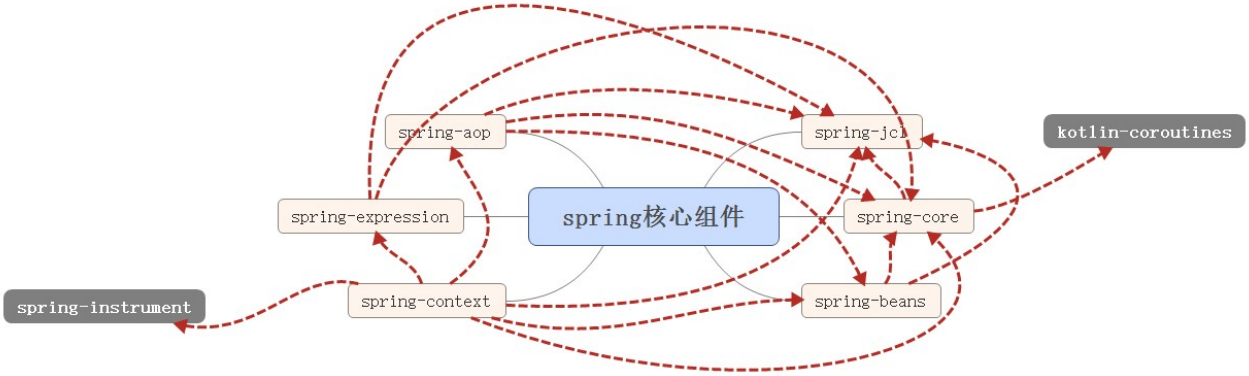
提到Spring，大家的第一想法是IoC和AoP，确实，IoC和AoP很重要，但Spring是一个历经了近二十载风雨的框架，它不仅仅只有IoC和AoP，还包含了诸多内容：



可以看到 Spring 核心组件 core 位于所有组件的核心，它包含了 DI，事件，资源，国际化，验证，数据绑定，类型转换，SpEL，AOP 等主要功能，基于核心应用组件之上的组件有：数据访问（data access）、Web 框架、Spring 还支持动态脚本语言如 Groovy，也提供了 Kotlin 的支持、及其它各种框架及组件如调度，邮件，缓存，远程调用等的支持集成。最后，Spring 还提供了方便的测试支持 spring-test。

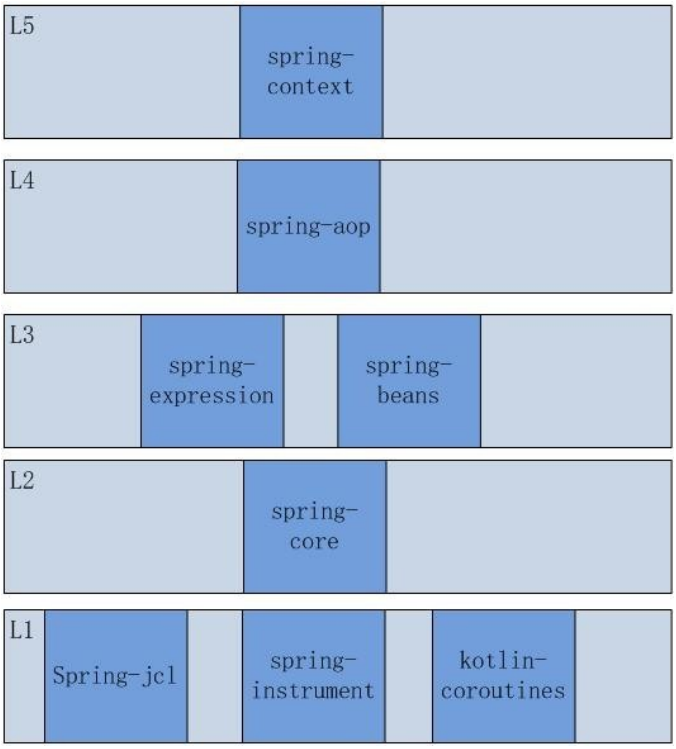
Spring-core核心基础组件

Spring5 基础组件包含六个，相应的关系如下图：



从箭头的流向可以归纳出从底到上的分层结构：

- spring-jcl、spring-instrument、kotlin-coroutines 是底层基础，不依赖 Spring；
- 其它组件，可以标记为 L1；
- spring-core 依赖于 spring-jcl、kotlin-coroutines 而又不依赖其它 Spring 组件，可以标记为 L2；
- spring-expression、spring-beans 依赖于 spring-jcl 和 spring-core 又不依赖其它 Spring 组件，可以标记为 L3；
- spring-aop 依赖于 spring-jcl、spring-core、spring-beans 而又不依赖其它 Spring 组件，可以标记为 L4；
- spring-context 依赖于 spring-jcl、spring-core、spring-expression、spring-beans、spring-aop，可标记为L5。

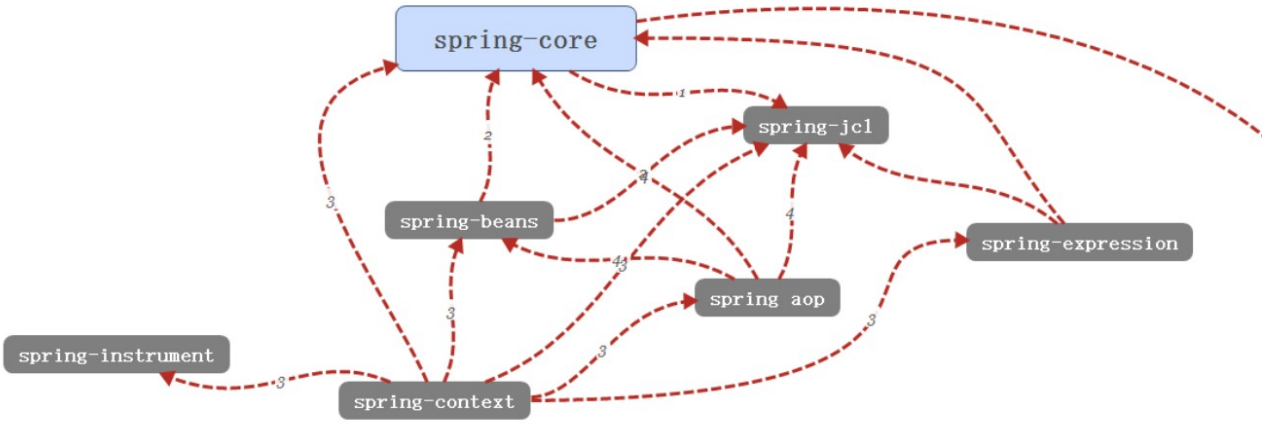


- **Spring-jcl**： JCL 全称：Jakarta Commons Logging，Spring-jcl 采用了设计模式中的“适配器模式”，它对外提供统一的接口，然后在适配类中将对日志的操作委托给具体的日志框架；
- **Spring-core**： Core 模块主要的功能是实现了反向控制 IOC（Inversion of Control）与依赖注入 DI（Dependency Injection）、Bean 配置以及加载。Core 模块中有 Beans、BeanFactory、BeanDefinitions、ApplicationContext 等几个重要概念；
- **Spring-expression**： Spring 表达式语言，解析 Spring 表达式语言；
- **Spring-beans**： 负责 Bean 工厂中 Bean 的装配，所谓 Bean 工厂即是创建对象的工厂，Bean 的装配也就是对象的创建工作。**重点：BeanFactory**；
- **Spring-aop**： Spring 提供了面向切面功能的模块；
- **Spring-context**： Spring 的 IOC 容器，因大量调用 Spring Core 中的函数，整合了 Spring 的大部分功能。Bean 创建好对象后，由 Context 负责建立 Bean 与 Bean 之间的关系并维护。所以也可以把 Context 看成是 Bean 关系的集合，**重点：ApplicationContext**。

使用的其它组件：

- **Spring-instrument**： 相当于一个检测器，提供对 JVM 以及对 Tomcat 的检测；
- **kotlin-coroutines**： 引入了协程。

为了方便查看，我们去掉了 Spring 核心组件的描述，仅仅描述核心组件之间的关系，如下图所示：



Spring热点面试题目集萃

我收集了一些 Spring 面试的问题，这些问题可能会在下一次技术面试中遇到。对于其他 Spring 模块，我将单独分享面试问题和答案。

如果你能将在以前面试中碰到的，且你认为这些应该是一个有 Spring 经验的人可能被问到的问题发给我，我将不胜感激！

我将把它们添加到这个列表中。这将对其他学习者也会有很大的帮助。

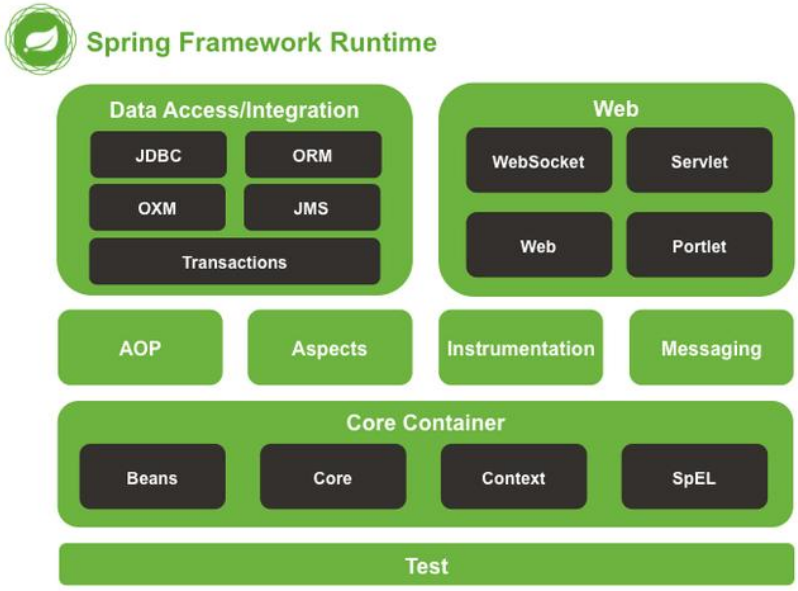
1. 什么是 Spring 框架?它的主要模块有哪些?
2. 使用 Spring 框架的好处是什么?
3. 什么是控制反转（IoC）和依赖注入?
4. 在 Spring 框架中的 IoC 是怎么样子的 ?
5. BeanFactory 和 ApplicationContext 之间的区别?
6. 将 Spring 配置到应用程序中的方式有哪些?
7. 基于XML 的 Spring 配置是什么样子的?
8. 基于 Spring Java 的配置是什么样子的?
9. 基于 Spring 注解的配置是什么样子的?
10. 请解释一下 Spring Bean 的生命周期?
11. Spring Bean 作用域的有哪些?
12. Spring 的内部 Bean 是什么?
13. 在 Spring 框架中，单例 Bean 线程安全吗?
14. 如何在 Spring 中注入 Java 集合?请给个例子好吗?
15. 如何将一个 java.util.* 属性注入到 Spring Bean?
16. 解释一下 Spring Bean 的自动注入式什么样的?
17. 请解释一下不同的 Bean 自动注入模式?
18. 怎么打开基于注释的自动注入的?
19. 能否用例子解释一下 @Required 注解吗?
20. 能否用例子解释一下 @Autowired 注解吗?
21. 能否用例子讲解一下 @Qualifier 注解吗?
22. 构造方法注入和 setter 注入之间的区别吗?
23. Spring 框架的事件类型有哪些?

24. FileSystemResource 和 ClassPathResource 之间的区别吗?
25. 列举一些 Spring 框架中使用的设计模式?

1. Spring 框架是什么?它的主要模块有哪些?

Spring 框架是一个 Java 平台，提供全面的基础设施支持开发 Java 应用程序。Spring 处理基础设施部分，这样你就可以专注于应用程序部分。Spring 框架内，把一级对象通过设计模式封装起来，您可以放心的集成到您自己的应用程序而不用关注他们如何在后台工作。

目前，Spring 框架由功能组织成大约 20 个模块。这些模块分为核心容器、数据访问/集成、Web、AOP（面向切面的编程）、Instrument（支持和类加载器的实现来在特定的应用服务器上使用）、消息和测试，如下列图所示。



2. 使用 Spring 框架的好处是什么?

下面是一些使用 Spring 框架的好处的列表：

- 通过依赖注入（DI）方式，在构造方法或者 Java Bean 属性上，依赖关系是明确的和明显的；
- IoC 容器往往是轻量级的，特别是与 EJB 容器相比。这是有利于在有限的内存和 CPU 资源的计算机上开发和部署应用程序；
- Spring 不重新发明轮子，相反，它利用一些现有的技术如几个 ORM 框架，日志框架，JEE，quartz 和 JDK 计时器，其他视图技术等；
- Spring 是模块化的，尽管包和类很重要，你只关心你需要的模块，忽略其它模块；
- 在 Spring 测试应用程序很简单，因为依赖环境的代码被移入到框架本身。此外，通过使用 JavaBean-style pojo 方式，使用依赖注入注入测试数据变得更容易；
- Spring 的 Web 框架是一个设计良好的 Web MVC 框架，它可以很好的替代其它 Web 框架如 Struts ；
- Spring 提供了一致的事务管理界面，可以管理小到一个本地事务（例如，使用一个数据库）和大到全局事务（例如，使用 JTA）。

3. 什么是控制反转(IoC)和依赖项注入?

依赖注入和控制反转是对同一件事情的不同描述，从某个方面讲，就是它们描述的角度不同。

依赖注入是从应用程序的角度描述，可以把依赖注入描述完整点：应用程序依赖容器创建并注入它所需要的外部资源；

而控制反转是从容器的角度描述，描述完整点：容器控制应用程序，由容器反向的向应用程序注入应用程序所需要的外部资源。

在 Java 中，依赖注入可能发生 3 种注入方式：

1. 构造方法注入；
2. setter 方法注入；
3. 接口注入。

4. Spring 框架的 IoC 是怎么样的?

org.springframework.beans 和 org.springframework.context 包是 Spring 框架 IoC 容器的基础。

BeanFactory 接口提供了一个高级的配置机制来管理任意属性的对象。

ApplicationContext 接口基于 BeanFactory 构建（是一个子接口）并添加其他功能，如：Spring 的 AOP 功能、信息资源处理（用于国际化）、事件传播和应用程序层的特定上下文如在 Web 应用程序中使用 WebApplicationContext。

org.springframework.beans.factory.BeanFactory 是 Spring IoC 容器真实展现，负责管理上述 Bean。BeanFactory 接口是 Spring IoC 容器接口的核心。

5. BeanFactory 和 ApplicationContext 之间的区别?

一个 BeanFactory 就像包含 bean 集合的工厂类。BeanFactory 在内部持有多个 Bean 的定义，当客户端请求 bean 时，将 bean 进行实例化。

初始化时 BeanFactory 能够保持对象的依赖关系。这减轻了负担从 bean 本身和 bean 客户机的配置。BeanFactory 在一个 bean 的生命周期也能起作用，它可以调用 bean 的自定义初始化和销毁方法。

表面上看，applicationContext 和 BeanFactory 是一样。同样加载 bean 定义，将 bean 连接在一起，分发 bean。但 applicationContext 还提供：

1. 一种解析消息的手段，包括对国际化的支持；
2. 一个更通用的加载文件资源的方法；
3. bean 事件注册为监听器。

三个常用的 ApplicationContext 实现是：

- **ClassPathXmlApplicationContext**：它从 classpath 路径下的一个 XML 文件加载 Context ，将 Context 作为 classpath 下的资源。加载应用程序 classpath 下的 Context 使用的代码如下：

```
ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");
```

- **FileSystemXmlApplicationContext**：它从文件系统的一个 XML 文件加载上下文定义的。从文件系统加载应用程序上下文通过如下代码实现：

```
ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");
```

- **XmlWebApplicationContext**：它从一个 Web 应用程序中包含的 XML 文件加载 Context。

6. 将 Spring 配置应用程序的方式有哪些呢?

配置 Spring 到您的应用程序有 3 种方式：

1. 基于 XML 的配置；
2. 基于注解的配置；
3. 基于 Java 的配置。

7. 基于 XML 的 Spring 配置是什么样子的?

在 Spring 框架中，bean 所需的依赖和服务定义在配置文件中，配置文件通常是 XML 格式。通常这些配置文件都以`<beans>`标签开始，含有大量的 bean 定义和特定于应用程序的配置选项。Spring XML 配置的主要目标是让所有 Spring 组件通过使用 XML 配置文件。

这意味着不会出现任何其他类型的 Spring 配置（如通过 Java 类注释或配置）。Spring XML 配置中使用 Spring 命名空间提供的 XML 标记中使用的配置；Spring 命名空间主要有：context、bean、jdbc、tx、aop、mvc 等。

```
<beans>
   <!-- JSON Support -->
   <bean name="viewResolver" class="org.springframework.web.servlet.view.BeanNameViewResolver"/>
   <bean name="jsonTemplate" class="org.springframework.web.servlet.view.json.MappingJackson2JsonView"/>
   <bean id="restTemplate" class="org.springframework.web.client.RestTemplate"/> </beans>
```

最简单的让您的应用程序加载配置文件和配置运行时组件方式是在 web.xml 文件中配置 DispatcherServlet，如下所示：

```
<web-app>
   <display-name>Archetype Created Web Application</display-name>
   <servlet>
      <servlet-name>spring</servlet-name>
      <servlet-class> org.springframework.web.servlet.DispatcherServlet </servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>spring</servlet-name>
      <url-pattern>/</url-pattern>
   </servlet-mapping>
</web-app>
```

8. 基于 Java 的 Spring 配置是什么样子的?

在支持 Spring 的新 Java 配置组件中，@Configuration 注解的类和 @Bean 注解的方法是核心组件。

@Bean 注解用于通过方法来实例化，配置和初始化一个新的由 Spring IoC 容器管理的对象。@Bean 注解和`<bean />` 元素扮演相同的角色。

在一个类上使用 @Configuration 注解，其主要用途是作为 bean 定义的来源。此外，在同一个类中@Configuration 类允许 inter-bean 定义通过简单地调用实现依赖关系。最简单的 @Configuration 注解类如下：

```
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

上面注解类等价于基于 XML 配置文件如下：

```
<beans>
    <bean id="myService" class="com.howtodoinjava.services.MyServiceImpl"/>
</beans>
```

为了使这样的配置能生效，需要使用 AnnotationConfigApplicationContext 的帮助。

```
public static void main(String[] args) {
    ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
    MyService myService = ctx.getBean(MyService.class); myService.doStuff();

}
```

为使组件扫描生效，只需要 @Configuration 类注解如下：

```
@Configuration
@ComponentScan(basePackages = "com.howtodoinjava")
public class AppConfig { ... }
```

在上面的示例中 com.howtodoinjava 包将被扫描，寻找任何带注解 @Component 的类，这些类将在容器内登记为 Spring Bean。

如果你使用以上的方式配置一个 Web 应用程序，那么需要 AnnotationConfigWebApplicationContext 类来使之生效。AnnotationConfigWebApplicationContext 的使用可以通过配置 Spring ContextLoaderListener 的 servlet listener，Spring MVC DispatcherServlet 等。

```
<web-app>
  <!-- Configure ContextLoaderListener to use AnnotationConfigWebApplicationContext instead of the default XmlWebApplicationContext -->
  <context-param>
    <param-name>contextClass</param-name>
    <param-value>      org.springframework.web.context.support.AnnotationConfigWebApplicationContext
    </param-value>
 </context-param>
  <!-- Configuration locations must consist of one or more comma- or space-delimited fully-qualified @Configuration classes. Fully-qualified packages may also be specified for component-scanning -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>com.howtodoinjava.AppConfig</param-value>
 </context-param>
  <!-- Bootstrap the root application context as usual using ContextLoaderListener --> <listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
  <!-- Declare a Spring MVC DispatcherServlet as usual -->
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class> <!-- Configure DispatcherServlet to use AnnotationConfigWebApplicationContext instead of the default XmlWebApplicationContext -->
    <init-param>
      <param-name>contextClass</param-name>
      <param-value> org.springframework.web.context.support.AnnotationConfigWebApplicationContext </param-value>
    </init-param> <!-- Again, config locations must consist of one or more comma- or space-delimited and fully-qualified @Configuration classes -->
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>com.howtodoinjava.web.MvcConfig</param-value>
    </init-param>
  </servlet>
  <!-- map all requests for /app/* to the dispatcher servlet -->
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/app/*</url-pattern>
  </servlet-mapping>
</web-app>
```

9. 基于 Spring 注解的配置是什么样子的?

从 Spring 2.5 就可以使用注解来配置依赖注入。而不是使用 XML 来描述一个 bean 的注入，你可以通过使用注解相关的类，方法或字段声明将 bean 配置的移到注解类本身。

注释注入执行 XML 注入之前，因此后者配置将会覆盖前者属性连接通过这两种方法。

默认情况下，Spring 容器没有打开自动注解功能。所以在使用具有 Spring 注解之前，我们需要在我们的 Spring 配置文件启用它。如果你想在 Spring 应用程序中使用的自动注解，考虑配置文件上加上下面的配置。

```
<beans>
 <context:annotation-config/> <!-- bean definitions go here -->
</beans>
```

一旦配置了 `<context:annotation-config/>`，表明在 Spring 中您可以开始使用属性，方法和构造函数的自动注入。

一些重要的注解：

1. **@Required**：@Required 注解适用于 bean 属性 setter 方法；
2. **@Autowired**：@Autowired 注解可以适用于 bean 属性 setter 方法、non-setter 方法、构造函数和属性；
3. **@Qualifier**：@Qualifier 注解加上 @Autowired 可以用来消除多个 bean 混乱来保证唯一的 bean 注入；
4. **jsr - 250 注释**：Spring 支持基于 jsr - 250 的注解如 @Resource、@PostConstruct 和 @PreDestroy。

10.请解释一下 Spring Bean 的生命周期?

一个 Spring Bean 的生命周期很容易理解。当一个 bean 实例化时可能需要执行一些初始化动作进入使 bean 达到一个可用的状态。同样，当不再需要 bean 时，将 bean 从容器中移除，可能需要销毁。

Spring BeanFactory 通过 Spring 容器负责管理 bean 的生命周期。bean 的生命周期包括可以大体分类为两类的回调方法

1. 初始化后的回调方法；
2. 销毁前的回调方法。

Spring 框架提供了以下 4 种方法控制 bean 的生命周期事件：

- InitializingBean 和 DisposableBean 回调接口；
- 其他知道接口为特定的行为；
- 定制的 init() 和 destroy() 方法在 bean 配置文件；
- @PostConstruct 和 @PreDestroy 注解。

例如： customInit() 和 customDestroy() 方法生命周期方法的例子。

```
<beans>
   <bean id="demoBean" class="com.howtodoinjava.task.DemoBean" init-method="customInit" destroy-method="customDestroy"></bean>
</beans>
```

11. Spring Bean 的作用域 scope 有哪些?

Spring 容器中的 bean 有 5 种 scope，分别是：

1. **单例 singleton**：默认情况下都是单例的，它要求在每个 Spring 容器内不论你请求多少次这个实例，都只有一个实例。单例特性是由 BeanFactory 本身维护的；
2. **原型 prototype**：这个 bean 的实例和单例相反，一个新的请求产生一个新的 bean 实例；
3. **请求 request**：在一个请求内，将会为每个 Web 请求的客户端创建一个新的 bean 实例。一旦请求完成后，bean 将失效，然后被垃圾收集器回收掉；
4. **会话 session**：就像请求范围，这样可以确保每个用户会话 bean 的一个实例。当用户结束其会话，bean 失效；
5. **全局会话 global-session**：应用到 Portlet 应用程序。基于 Servlet 的应用程序和会话相同。

12. Spring 的内部 Bean 是什么?

在 Spring 框架中，当一个 bean 只用于一个特定属性，建议将它声明为一个内在的 bean。内部 bean 同时支持 setter 注入属性和构造函数注入“constructor-arg”。

例如，假设一个 Customer 类的引用 Person 类。在我们的应用程序中，我们将只创建一个 Person 类的实例，并在 Customer 使用它。

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

现在内部 bean 声明是这样的：

```
<bean id="CustomerBean" class="com.howtodoinjava.common.Customer">
   <property name="person">
      <!-- This is inner bean -->
      <bean class="com.howtodoinjava.common.Person">
         <property name="name" value="adminis"></property>
         <property name="address" value="India"></property>
         <property name="age" value="34"></property>
      </bean>
   </property>
</bean>
```

13. 在 Spring 框架中，单例 Bean 是线程安全的吗?

Spring 框架不对单例的 bean 做任何多线程的处理。单例的 bean 的并发问题和线程安全是开发人员的责任。

而实际上，大多数 Spring Bean 没有可变状态（例如服务和 DAO 的类），这样的话本身是线程安全的。但如果您的 bean 有可变状态（例如视图模型对象），这就需要你来确保线程安全。

这个问题最简单和明显的解决方案是改变 bean Scope，可变的 bean 从“单例”到“原型”。

14. 如何在 Spring 里注入 Java 集合?请给个例子好吗?

Spring 提供了四种类型的配置元素集合，如下:

`<list>`：帮助注入一组值，允许重复；

`<set>`：帮助注入一组值，不允许重复；

`<map>`：帮助注入一个 K-V 的集合，名称和值可以是任何类型的；

`<props>`：帮助注入一个名称-值对集合，名称和值都是字符串。

让我们看看每种类型的例子。

```
<beans>
   <!-- Definition for javaCollection -->
   <bean id="javaCollection" class="com.howtodoinjava.JavaCollection">
      <!-- java.util.List -->
      <property name="customList">
         <list>
            <value>INDIA</value>
            <value>Pakistan</value>
            <value>USA</value> <value>UK</value>
         </list>
      </property>
      <!-- java.util.Set -->
      <property name="customSet">
         <set>
            <value>INDIA</value>
            <value>Pakistan</value>
            <value>USA</value>
            <value>UK</value>
         </set>
      </property>
      <!-- java.util.Map -->
      <property name="customMap">
         <map>
            <entry key="1" value="INDIA"/>
            <entry key="2" value="Pakistan"/>
            <entry key="3" value="USA"/>
            <entry key="4" value="UK"/>
         </map>
      </property>
      <!-- java.util.Properties -->
      <property name="customProperies">
         <props>
            <prop key="admin">admin@nospam.com</prop>
            <prop key="support">support@nospam.com</prop>
         </props>
      </property>
   </bean>
</beans>
```

15. 如何将一个 java.util 属性注入到 Spring Bean？

第一个方法是使用 `<props>` 标记如下。

```
<bean id="adminUser" class="com.howtodoinjava.common.Customer">
   <!-- java.util.Properties -->
   <property name="emails">
      <props>
         <prop key="admin">admin@nospam.com</prop>
         <prop key="support">support@nospam.com</prop>
      </props>
   </property>
</bean>
```

也可以使用“util:“名称空间创建 bean 的属性文件，并使用 bean 的 setter 方法注入。

```
<util:properties id="emails" location="classpath:com/foo/emails.properties" />
```

16. 解释一下 Spring Bean 的自动注入是怎么样的?

在 Spring 框架中，在配置文件中设置 bean 的依赖是一个很好的办法，但 Spring 容器也能够自动注入不同 bean 之间的关系。这意味着，通过检查 BeanFactory 的内容它可以为您的 bean 自动注入其他 bean。

可以为每个 bean 指定是否自动注入，因此可以支持一些 Bean 支持自动注入，而一些 bean 不会自动注入。

下面从 XML 配置文件摘录了自动根据名称注入的 bean。

```
<bean id="employeeDAO" class="com.howtodoinjava.EmployeeDAOImpl" autowire="byName" />
```

除了提供的自动装配模式 bean 配置文件，也可以在 bean 类中指定自动装配使用 @Autowired 注释。

> **Tips**：在 bean 类使用 @Autowired 注解，您必须在 Spring 应用程序中先启用下面的注解。

```
<context:annotation-config />
```

也可以通过在配置文件使用 AutowiredAnnotationBeanPostProcessorbean 来完成。

```
<bean class ="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>
```

现在，当注释配置已经启用，您可以自由使用 @Autowired 来自动注入 bean 依赖关系，以你喜欢的方式。

```
@Autowired
public EmployeeDAOImpl ( EmployeeManager manager ) {
   this.manager = manager;
}
```

17. 请解释一下不同的 bean 自动注入模式?

在 Spring 有五个自动注入模式。让我们逐个讨论。

1. **no**：默认情况下，Spring 框架的自动注入选项，即默认情况不开启自动注入。这意味着你必须使用标签在 bean 定义中显式地设置依赖项；
2. **byName**：这个选项是基于 bean 的名称的依赖项注入。当自动装配在 bean 属性，用属性名搜索匹配的 bean 定义配置文件。如果找到这样的 bean，注入属性。如果没有找到这样的 bean，就会产生一个错误；
3. **byType**：该选项允许基于 bean 的类型的依赖项注入。当在 bean 属性需要自动注入时，使用属性类的类型来搜索匹配的 bean 定义配置文件。如果找到这样的 bean，注入属性。如果没有找到这样的 bean，就会产生一个错误；
4. **constructor**：构造方法类似于 byType 自动注入，但适用于构造方法的参数。在自动注入 bean 时，它在所有构造函数参数类型中寻找匹配的构造函数的类类型参数，然后进行自动注入。请注意，如果在容器中没有一个 bean 构造函数参数类型满足，会抛出一个致命错误；
5. **autodetect**：自动侦测使用两种模式即构造函数或 byType 模式的自动注入。首先它将试图寻找有效的构造方法参数，如果发现构造方法模式则选择。如果没有构造方法中定义 bean，或者明确的默认无参构造方法，则选择 byType 模式自动注入。

18. 怎么打开基于注释的自动注入的?

要启用 @Autowired，你必须注册 AutowiredAnnotationBeanPostProcessor，你可以用两种方式。

在 bean 配置文件使用 `<context:annotation-config >` 。

```
<beans> <context:annotation-config /> </beans>
```

直接将 AutowiredAnnotationBeanPostProcessor 放到 bean 配置文件。

```
<beans>
   <bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>
</beans>
```

19. 能否用例子解释一下 @Required 注解吗？

在大规模的应用程序中，IoC 容器中可能会有成百上千的 bean 声明，以及它们之间的依赖关系通常是非常复杂的。

setter 注入的缺点之一是，很难给你检查出所需的所有属性是否已经注入。

为了克服这个问题，您可以设置 bean 的“dependency-check”属性，可以设置四个属性的其中之一即 none，simple， objects or all (没有一个是默认选项)。

在现实生活中应用程序中，您将不会感兴趣检查所有上下文中的 bean 属性配置文件。而你想要检查一些特定的 bean 是否已设置特定的属性。在这种情况下，Spring 的依赖项检查功能将不再适用。

为了解决这个问题，您可以使用 @Required 注解。在 bean 属性使用 @Required 注解的 setter 方法类文件如下：

```
public class EmployeeFactoryBean extends AbstractFactoryBean<Object> {
   private String designation;
   public String getDesignation() {
      return designation;
   }
   @Required
   public void setDesignation(String designation) {
      this.designation = designation;
   }
   //more code here
}
```

RequiredAnnotationBeanPostProcessor 是一个 Spring Bean 后置处理程序，检查 @Required 注解的所有的 bean 属性是否已设置。使用这个 bean 属性检查后置处理程序，您必须注册在 Spring IoC 容器中。

```
<bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor" />
```

如果 @Required 注解的任何属性没有设置，这个 bean 的处理器会抛出一个 BeanInitializationException 异常。

20.能否用例子解释一下 @Autowired 注解吗？

@Autowired 注解提供了更细粒度的控制，以及应该如何完成自动注入。@Autowired 注解和 @Required 注解一样，可用于 bean 的自动注入，它可以作用于构造方法，属性或具有任意名称和/或多个参数的方法。

例如，您可以使用 @Autowired 注解的 setter 方法来代替在 XML 配置文件中的 <property> 元素。当 Spring 找到一个 @Autowired 注解的方法，它尝试使用 byType 自动注入的方法。

您可以将 @Autowired 应用到构造方法。一个构造方法使用 @Autowired 注解表明，即使在 XML 文件没有配置 bean 的 <constructor-arg> 元素，当创建 bean 时，构造方法也会自动注入。

```
public class TextEditor {
   private SpellChecker spellChecker;
   @Autowired
   public TextEditor(SpellChecker spellChecker){
      System.out.println("Inside TextEditor constructor." );
      this.spellChecker = spellChecker;
   }
   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

没有构造方法参数的配置。

```
<beans> <context:annotation-config/>
   <!-- Definition for textEditor bean without constructor-arg -->
   <bean id="textEditor" class="com.howtodoinjava.TextEditor">
   </bean>
   <!-- Definition for spellChecker bean -->
   <bean id="spellChecker" class="com.howtodoinjava.SpellChecker">
   </bean>
</beans>
```

21. 能否用例子讲解一下 @Qualifier 注解吗？

@Qualifier 限定哪个 bean 应该被自动注入。当 Spring 无法判断出哪个 bean 应该被注入时，@Qualifier 注解有助于消除歧义 bean 的自动注入。

参见下面的例子：

```
public class Customer { @Autowired private Person person; }
```

我们有两个 bean 定义为 Person 类的实例。

```
<bean id="customer" class="com.howtodoinjava.common.Customer" />
<bean id="personA" class="com.howtodoinjava.common.Person" >
   <property name="name" value="lokesh" />
</bean>
<bean id="personB" class="com.howtodoinjava.common.Person" >
   <property name="name" value="alex" />
</bean>
```

Spring 知道哪个 bean 应该自动注入?不。当您运行上面的例子时，抛出如下异常：

```
Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No unique bean of type [com.howtodoinjava.common.Person] is define
d: expected single matching bean but found 2: [personA, personB]
```

要解决以上问题，你需要使用 @Quanlifier 注解告诉 Spring 哪个 bean 应该被 autowired 的。

```
public class Customer {
   @Autowired
   @Qualifier("personA")
   private Person person;
}
```

22. 构造方法注入和 setter 注入之间的区别吗?

有以下几点明显的差异：

1. 在 Setter 注入，可以将依赖项部分注入，构造方法注入不能部分注入，因为调用构造方法如果传入不匹配的参数就会报错；
2. 如果我们为同一属性提供 Setter 和构造方法注入，Setter 注入将覆盖构造方法注入。但是构造方法注入不能覆盖 setter 注入值。显然，构造方法注入被称为创建实例的第一选项；
3. 使用 setter 注入你不能保证所有的依赖都被注入，这意味着你可以有一个对象依赖没有被注入。在另一方面构造方法注入直到你所有的依赖都注入后才开始创建实例；
4. 在构造函数注入，如果 A 和 B 对象相互依赖：A 依赖于 B，B 也依赖于 A，此时在创建对象的 A 或者 B 时，Spring 抛出 ObjectCurrentlyInCreationException。所以 Spring 可以通过 setter 注入，从而解决循环依赖的问题。

23. spring 框架的事件类型有哪些?

Spring 的 ApplicationContext 具有代码层上支持事件和监听器的功能。我们可以创建 bean 监听通过 ApplicationContext 发布的事件。ApplicationContext 里的事件处理通过提供 ApplicationEvent 类和 ApplicationListener 接口来完成。所以如果一个 bean 实现了 ApplicationListener 接口，当一个 ApplicationEvent 发布到 ApplicationContext 时，该 bean 将接到通知。

```
public class AllApplicationEventListener implements ApplicationListener < ApplicationEvent > {
   @Override
   public void onApplicationEvent(ApplicationEvent applicationEvent) {
      //process event
   }
}
```

Spring 提供了以下 5 个标准事件:

1. **ContextRefreshedEvent**：当 ApplicationContext 初始化或刷新时发布这个事件。这个事件也可以通过 ConfigurableApplicationContext 接口的 refresh() 方法来触发；
2. **ContextStartedEvent**：当 ApplicationContext 被 ConfigurableApplicationContext 接口的 start() 方法启动时发布这个事件。你可以在收到这一事件后查询你的数据库或重启/启动任何停止的应用程序；
3. **ContextStoppedEvent**：当 ApplicationContext 被 ConfigurableApplicationContext 接口的 stop() 方法关闭时发布这个事件。你可以在收到这一事件后做一些清理工作；
4. **ContextClosedEvent**：当 ApplicationContext 时被 ConfigurableApplicationContext 接口的 close() 方法关闭时发布这个事件。一个终结的上下文达到生命周期结束，它不能刷新或重启；
5. **RequestHandledEvent**：这是一个网络自身的事件，告诉所有 bean：HTTP 请求服务已经处理完成。

除了上面的事件，您可以通过扩展 ApplicationEvent 类创建自定义事件。如：

```
public class CustomApplicationEvent extends ApplicationEvent {
   public CustomApplicationEvent ( Object source，  final String msg ) {
      super(source);
      System.out.println("Created a Custom event");
   }
}
```

监听这个事件，创建一个监听器是这样的：

```
public class CustomEventListener implements ApplicationListener < CustomApplicationEvent > {
    @Override
    public void onApplicationEvent(CustomApplicationEvent applicationEvent) {
        //handle event
    }
}
```

发布这个事件：

```
CustomApplicationEvent customEvent = new CustomApplicationEvent( applicationContext, "Test message" );
applicationContext.publishEvent ( customEvent );
```

24. FileSystemResource 和 ClassPathResource 之间的区别吗？

在 FileSystemResource 中你需要给出 spring-config.xml（Spring 配置）文件相对于您的项目的相对路径或文件的绝对位置。

在 ClassPathResource 中 Sping 查找文件使用 ClassPath，因此 spring-config.xml 应该包含在类路径下。

一句话，ClassPathResource 在类路径下搜索和 FileSystemResource 在文件系统下搜索。

25. 列举一下 Spring 框架使用的一些设计模式？

有很多不同的设计模式，但有一些明显的:

- **代理**：在 AOP 大量使用，还有远程模块；
- **单例**：Spring 配置文件中定义的 bean 默认是单例；
- **模板方法**：广泛使用处理重复逻辑的代码。例如 RestTemplate， JmsTemplate， JpaTemplate；
- **前端控制器**：Spring 提供了 DispatcherServlet，确保传入请求被分派到你的控制器；
- **视图助手**：Spring 有许多定制 JSP 标记，和 velocity 宏，协助在视图层分离展示代码；
- **依赖注入**：BeanFactory / ApplicationContext 的核心概念；
- **工厂模式**：BeanFactory 创建一个对象的实例。

原文：不可链接

}

46 Spring AOP 总结及面试热点题目集萃

48 Spring展望及后续(未完)