Spring问题集

## 1 、什么是 Spring 框架？Spring 框架有哪些主要模块？

JavaSE/EE应用一站式轻量级的开源框架。以IoC和AOP为内核。

IoC容器、数据访问和集成,、Web以及远程操作（MVC、WebService、WebSocket）、AOP（面向切面编程）、工具、消息和测试模块。

## 2 、使用 Spring 框架带来哪些好处？

## 3 、什么是控制反转(IOC/或DI) ？

在 Java 中依然注入有以下三种实现方式：

## 4 、请解释下 Spring 框架中的 IoC

Spring 中的 org.springframework.beans 包和 org.springframework.context 包

构成了 Spring 框架 IoC 容器的基础。

BeanFactory 接口提供了一个先进的配置机制，使得任何类型的对象的配置成为可能。

ApplicationContex 接口对 BeanFactory （是一个子接口）进行了扩展，在 BeanFactory

的基础上添加了其他功能，比如与 Spring 的 AOP 更容易集成，也提供了处理 message resource的机制（用于国际化）、事件传播以及应用层的特别配置，比如针对 Web 应用的

WebApplicationContext。org.springframework.beans.factory.BeanFactory 是 Spring IoC 容器的具体实现，用来包装和管理前面提到的各种 bean。BeanFactory 接口是 Spring IoC 容器的核心接口。

IOC:把对象的创建、初始化、销毁交给 spring 来管理，而不是由开发者控制，实现控制反转。

5 、BeanFactory 和 ApplicationContext 有什么区别？

BeanFactory 可以理解为含有 bean 集合的工厂类。BeanFactory 包含了种 bean 的定义，以便在接收到客户端请求时将对应的 bean 实例化。

BeanFactory 还能在实例化对象的时生成协作类之间的关系。此举将 bean 自身与 bean 客户端的配置中解放出来。BeanFactory 还包含 了 bean 生命周期的控制，调用客户端的初始化方法（initialization methods）和销毁方法（destruction methods）。

从表面上看，application context 如同 bean factory 一样具有 bean 定义、bean 关联关系的设置，根据请求分发 bean 的功能。但 applicationcontext 在此基础上还提供了其他的功能。

1. 提供了支持国际化的文本消息
2. 统一的资源文件读取方式
3. 已在监听器中注册的 bean 的事件

以下是三种较常见的 ApplicationContext 实现方式：

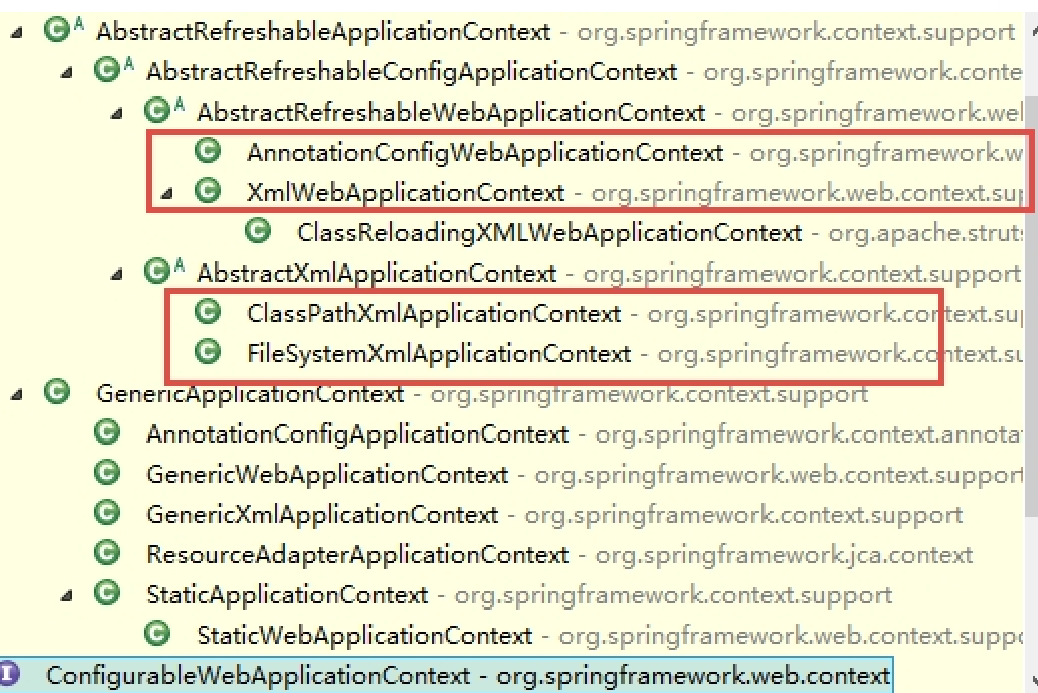
1. ClassPathXmlApplicationContext：从 classpath 的 XML 配置文件中读取上下文，并生成上下文定义。应用程序上下文从程序环境变量中

ApplicationContext context = new ClassPathXmlApplicationContext(“bean.xml”);

1. FileSystemXmlApplicationContext ：由文件系统中的 XML 配置文件读取上下文。

ApplicationContext context = new FileSystemXmlApplicationContext(“bean.xml”);

1. XmlWebApplicationContext：由 Web 应用的 XML 文件读取上下文。
2. AnnotationConfigApplicationContext(基于 Java 配置启动容器)



## 6 、Spring 有几种配置方式？

将 Spring 配置到应用开发中有以下三种方式：

1. 基于 XML 的配置

2. 基于注解的配置

3. 基于 Java 的配置

## 7 、如何用基于 XML 配置的方式配置 Spring

在 Spring 框架中，依赖和服务需要在专门的配置文件来实现，我常用的 XML 格式的配置文件。这些配置文件的格式通常用 <beans>开头，然后一系列的 bean 定义和专门的应用配置选项组成。

SpringXML 配置的主要目的时候是使所有的 Spring 组件都可以用 xml 文件的形式来进行配置。这意味着不会出现其他的 Spring 配置类型（比如声明的方式或基于 Java Class 的配置方式）

Spring 的 XML 配置方式是使用被 Spring 命名空间的所支持的一系列的 XML 标签来实现的。

Spring 有以下主要的命名空间：context、beans、jdbc、tx、aop、mvc 和 aso。

如：

<beans>

<!-- JSON Support -->

<bean name="viewResolver"

class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

<bean name="jsonTemplate"

class="org.springframework.web.servlet.view.json.MappingJackson2JsonV

iew"/>

<bean id="restTemplate"

class="org.springframework.web.client.RestTemplate"/>

</beans>

下面这个 web.xml 仅仅配置了 DispatcherServlet，这件最简单的配置便能满足应用程序配置运行时组件的需求。

<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 、如何用基于 Java 配置的方式配置 Spring

Spring 对 Java 配置的支持是由@Configuration 注解和@Bean 注解来实现的。由@Bean 注解的方法将会实例化、配置和初始化一个 新对象，这个对象将由 Spring 的 IoC 容器来管理。@Bean 声明所起到的作用与<bean/> 元素类似。被 @Configuration 所注解的类则表示这个类的主要目的是作为 bean 定义的资源。被@Configuration 声明的类可以通过在同一个类的 内部调用@bean 方法来设置嵌入 bean 的依赖关系。

最简单的@Configuration 声明类请参考下面的代码：

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

对于上面的@Beans 配置文件相同的 XML 配置文件如下：

<beans>

<bean id="myService" class="com.somnus.services.MyServiceImpl"/>

</beans>

上述配置方式的实例化方式如下：利用 AnnotationConfigApplicationContext 类进行实例化

|  |
| --- |
| ApplicationContext ctx = **new** AnnotationConfigApplicationContext(AppConfig.**class**);  MyService myService = ctx.getBean(MyService.**class**);  myService.doStuff(); |

要使用组件组建扫描，仅需用@Configuration 进行注解即可：

|  |
| --- |
| @Configuration  @ComponentScan(basePackages = "com.somnus")  public class AppConfig {  ...  } |

在上面的例子中，com.acme 包首先会被扫到，然后再容器内查找被@Component 声明的类，找

到后将这些类按照 Sring bean 定义进行注册。

如果你要在你的 web 应用开发中选用上述的配置的方式的话，需要用AnnotationConfigWebApplicationContext 类来读 取配置文件，可以用来配置 Spring 的Servlet 监听器 ContextLoaderListener 或者 Spring MVC 的 DispatcherServlet。

|  |
| --- |
| <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.AnnotationConfigWebApplicatio  nContext  </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.AnnotationConfigWebApplicatio  nContext  </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 、怎样用注解的方式配置 Spring

Spring 在 2.5 版本以后开始支持用注解的方式来配置依赖注入。可以用注解的方式来替代 XML 方

式的 bean 描述，可以将 bean 描述转移到组件类的 内部，只需要在相关类上、方法上或者字段声

明上使用注解即可。注解注入将会被容器在 XML 注入之前被处理，所以后者会覆盖掉前者对于同一

个属性的处理结 果。

注解装配在 Spring 中是默认关闭的。所以需要在 Spring 文件中配置一下才能使用基于注解的装配

模式。如果你想要在你的应用程序中使用关于注解的方法的话，请参考如下的配置。

<beans>

<context:annotation-config/>

<!-- bean definitions go here -->

</beans>

在 <context:annotation-config/>标签配置完成以后，就可以用注解的方式在 Spring 中向属

性、方法和构造方法中自动装配变量。

下面是几种比较重要的注解类型：

1. @Required：该注解应用于设值方法。
2. @Autowired：该注解应用于有值设值方法、非设值方法、构造方法和变量。
3. @Qualifier：该注解和@Autowired 注解搭配使用，用于消除特定 bean 自动装配的歧义。
4. JSR-250 Annotations：Spring 支持基于 JSR-250 注解的以下注解，@Resource、@PostConstruct 和 @PreDestroy。

## 10 、请解释 Spring Bean

Spring Bean 的生命周期简单易懂。在一个 bean 实例被初始化时，需要执行一系列的初始化操作

以达到可用的状态。同样的，当一个 bean 不在被调用时需要进行相关的析构操作，并从 bean 容

器中移除。

Spring bean factory 负责管理在 spring 容器中被创建的 bean 的生命周期。Bean 的生命周期

由两组回调（call back）方法组成。

1. 初始化之后调用的回调方法。

2. 销毁之前调用的回调方法。

Spring 框架提供了以下四种方式来管理 bean 的生命周期事件：

InitializingBean 和 DisposableBean 回调接口

 针对特殊行为的其他 Aware 接口

 Bean 配置文件中的 Custom init()方法和 destroy()方法

 @PostConstruct 和@PreDestroy 注解方式

使用 customInit() 和 customDestroy()方法管理 bean 生命周期的代码样例如下：

<beans>

<bean id="demoBean" class="com.somnus.task.DemoBean" init-

method="customInit" destroy-method="customDestroy"></bean>

</beans>

## 11 、Spring Bean 的作用域之间有什么区别？

Spring 容器中的 bean 可以分为 5 个范围。所有范围的名称都是自说明的，但是为了避免混淆，还

是让我们来解释一下：

1. singleton：这种 bean 范围是默认的，这种范围确保不管接受到多少个请求，每个容器中只有一个

bean 的实例，单例的模式由 bean factory 自身来维护。

2. prototype：原形范围与单例范围相反，为每一个 bean 请求提供一个实例。

3. request：在请求 bean 范围内会每一个来自客户端的网络请求创建一个实例，在请求完成以后，

bean 会失效并被垃圾回收器回收。

4. Session：与请求范围类似，确保每个 session 中有一个 bean 的实例，在 session 过期后，bean

会随之失效。

5. global- session：global-session 和 Portlet 应用相关。当你的应用部署在 Portlet 容器中工作

时，它包含很多 portlet。如果 你想要声明让所有的 portlet 共用全局的存储变量的话，那么这全

局变量需要存储在 global-session 中。

全局作用域与 Servlet 中的 session 作用域效果相同。

## 12 、什么是 Spring inner beans ？

在 Spring 框架中，无论何时 bean 被使用时，当仅被调用了一个属性。一个明智的做法是将这个

bean 声明为内部 bean。内部 bean 可以用 setter 注入“属性”和构造方法注入“构造参数”的方式来

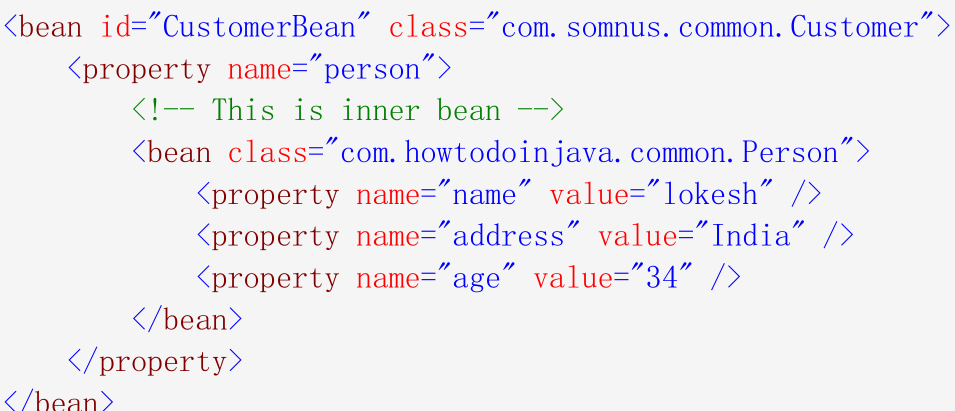
实现。

比如，在我们的应用程序中，一个 Customer 类引用了一个 Person 类，我们的要做的是创建一个

Person 的实例，然后在 Customer 内部使用。

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

内部 bean 的声明方式如下：



## 13 、Spring 框架中的单例 Beans

Spring 框架并没有对单例 bean 进行任何多线程的封装处理。关于单例 bean 的线程安全和并发问

题需要开发者自行去搞定。但实际上，大部分的 Spring bean 并没有可变的状态(比如 Serview 类

和 DAO 类)，所以在某种程度上说 Spring 的单例 bean 是线程安全的。如果你的 bean 有多种状

态的话（比如 View Model 对象），就需要自行保证线程安全。

最浅显的解决办法就是将多态 bean 的作用域由“singleton”变更为“prototype”。

## 14 、请举例说明如何在 Spring 中注入一个 Java Collection ？

Spring 提供了以下四种集合类的配置元素：

 <list> : 该标签用来装配可重复的 list 值。

 <set> : 该标签用来装配没有重复的 set 值。

 <map>: 该标签可用来注入键和值可以为任何类型的键值对。

 <props> : 该标签支持注入键和值都是字符串类型的键值对。

下面看一下具体的例子：

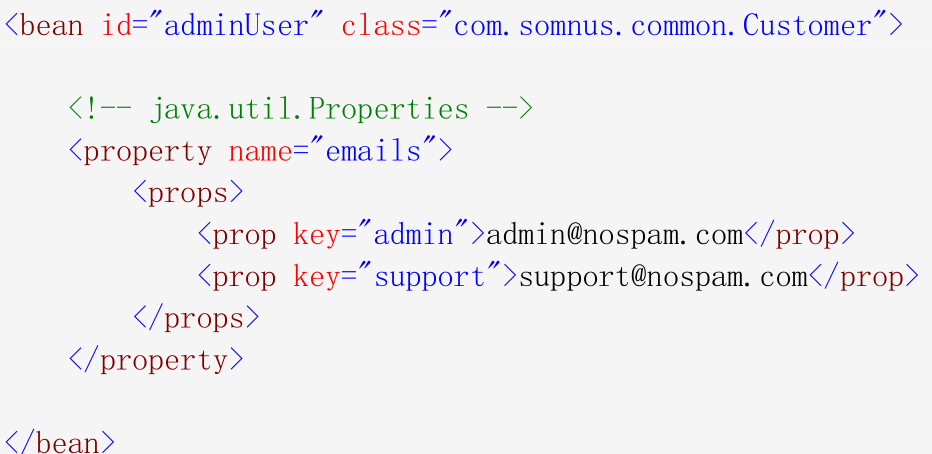






## 15 、如何向 Spring Bean 中注入一个 Java.util.Properties

第一种方法是使用如下面代码所示的<props> 标签：



也可用”util:”命名空间来从 properties 文件中创建出一个 propertiesbean，然后利用 setter 方

法注入 bean 的引用。

## 16 、请解释 Spring Bean 的自动装配？

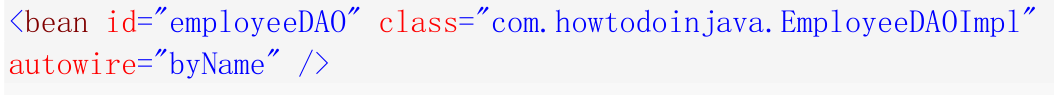
在 Spring 框架中，在配置文件中设定 bean 的依赖关系是一个很好的机制，Spring 容器还可以自

动装配合作关系 bean 之间的关联关系。这意味着 Spring 可以通过向 Bean Factory 中注入的方

式自动搞定 bean 之间的依赖关系。自动装配可以设置在每个 bean 上，也可以设定在特定的 bean

上。

下面的 XML 配置文件表明了如何根据名称将一个 bean 设置为自动装配：



除了 bean 配置文件中提供的自动装配模式，还可以使用 @Autowired 注解来自动装配指定

的 bean。在使用@Autowired 注解之前需要在按照如下的配置方式在 Spring 配

置文件进行配置才可以使用。

<context:annotation-config />

也可以通过在配置文件中配置 AutowiredAnnotationBeanPostProcessor 达到相同的效果。

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

配置好以后就可以使用 @Autowired 来标注了。

@Autowired

public EmployeeDAOImpl ( EmployeeManager manager ) {

his.manager = manager;

}

## 请解释自动装配模式的区别？

在 Spring 框架中共有 5 种自动装配，让我们逐一分析。

1. **no：**这是 Spring 框架的默认设置，在该设置下自动装配是关闭的，开发者需要自行在 bean 定义中用标签明确的设置依赖关系。

**2. byName：**该选项可以根据 bean 名称设置依赖关系。当向一个 bean 中自动装配一个属性时，容

器将根据 bean 的名称自动在在配置文件中查询一个匹配的 bean。如果找到的话，就装配这个属

性，如果没找到的话就报错。

**3. byType：**该选项可以根据 bean 类型设置依赖关系。当向一个 bean 中自动装配一个属性时，容器

将根据 bean 的类型自动在在配置文件中查询一个匹配的 bean。如果找到的话，就装配这个属性，

如果没找到的话就报错。

**4. constructor：**造器的自动装配和 byType 模式类似，但是仅仅适用于与有构造器相同参数的

bean，如果在容器中没有找到与构造器参数类型一致的 bean，那么将会抛出异常。

**5. autodetect：**该模式自动探测使用构造器自动装配或者 byType 自动装配。首先，首先会尝试找合

适的带参数的构造器，如果找到的话就是用构造器自动装配，如果在 bean 内部没有找到相应的构

造器或者是无参构造器，容器就会自动选择 byType 的自动装配方式。

## 如何开启基于注解的自动装配？

要使用 @Autowired，需要注册 AutowiredAnnotationBeanPostProcessor，可以

有以下两种方式来实现：

1. 引入配置文件中的<bean>下引入 <context:annotation-config>



1. 在 bean 配置文件中直接引入 AutowiredAnnotationBeanPostProcessor



## 19 、请举例解释@Required

在产品级别的应用中，IoC 容器可能声明了数十万了 bean，bean 与 bean 之间有着复杂的依赖关

系。设值注解方法的短板之一就是验证所有的属性是否被注解是一项十分困难的操作。可以通过在

<bean>中设置“dependency-check”来解决这个问题。

在应用程序的生命周期中，你可能不大愿意花时间在验证所有 bean 的属性是否按照上下文文件正

确配置。或者你宁可验证某个 bean 的特定属性是否被正确的设置。即使是用“dependency-

check”属性也不能很好的解决这个问题，在这种情况下，你需要使用 @Required 注解。

需要用如下的方式使用来标明 bean 的设值方法。

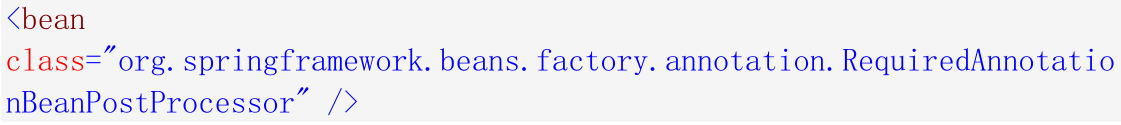
|  |
| --- |
| public class EmployeeFactoryBean extends AbstractFactoryBean<Object>{  private String designation;  public String getDesignation() {  return designation;  }  @Required  public void setDesignation(String designation) {  this.designation = designation;  }  //more code here  } |

RequiredAnnotationBeanPostProcessor 是 Spring 中的后置处理用来验证被

@Required 注解的 bean 属性是否被正确的设置了。在使用

RequiredAnnotationBeanPostProcesso 来验证 bean 属性之前，首先要在 IoC 容器中对

其进行注册：



但是如果没有属性被用 @Required 注解过的话，后置处理器会抛出一个BeanInitializationException 异常。

## 20 、请举例解释@Autowired 注解？

## 21 、请举例说明@Qualifier

## 构造方法注入和设值注入有什么区别？

请注意以下明显的区别：

1. 在设值注入方法支持大部分的依赖注入，如果我们仅需 要注入 int、string 和 long 型的变量，我

们不要用设值的方法注入。对于基本类型，如果我们没有注入的话，可以为基本类型设置默认值。

在构造方法 注入不支持大部分的依赖注入，因为在调用构造方法中必须传入正确的构造参数，否则

的话为报错。

2. 设值注入不会重写构造方法的值。如果我们对同一个变量同时使用了构造方法注入又使用了设置方法注入的话，那么构造方法将不能覆盖由设值方法注入的值。很明显，因为构造方法尽在对象被创建时调用。

3. 在使用设值注入时有可能还不能保证某种依赖是否已经被注入，也就是说这时对象的依赖关系有可能是不完整的。而在另一种情况下，构造器注入则不允许生成依赖关系不完整的对象。

4. 在设值注入时如果对象 A 和对象 B 互相依赖，在创建对象 A 时 Spring 会抛出sObjectCurrentlyInCreationException 异常，因为在 B 对象被创建之前 A 对

象是不能被创建的，反之亦然。所以 Spring 用设值注入的方法解决了循环依赖的问题，因对象的设值方法是在对象被创建之前被调用的。

## 23 、Spring框架中有哪些不同类型的事件？

Spring 的 ApplicationContext 提供了支持事件和代码中监听器的功能。

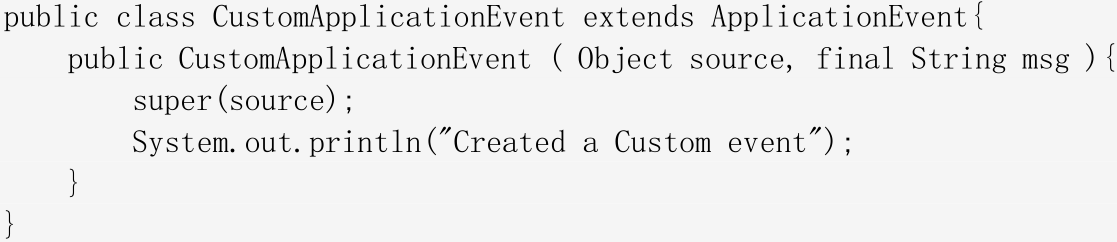
我们可以创建 bean 用来监听在 ApplicationContext 中发布的事件。 ApplicationEvent类和在 ApplicationContext 接口 中处理的事件，如果一个 bean 实现了ApplicationListener 接口，当一个 ApplicationEvent 被发布以后，bean 会自动被通知。



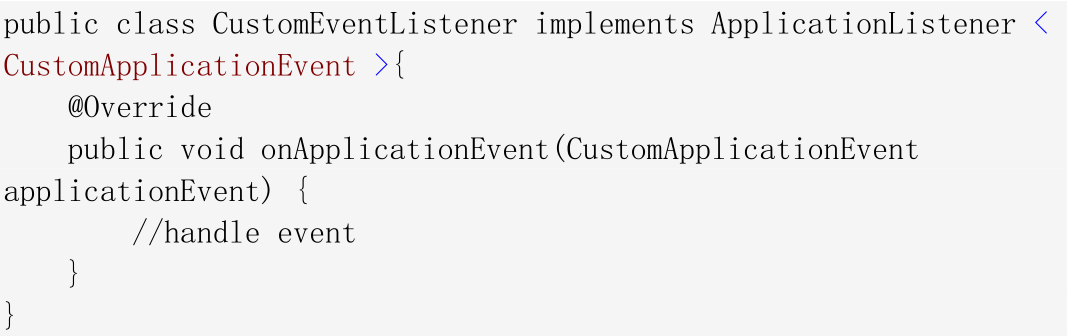
Spring 提供了以下 5 中标准的事件：

* 1. 上下文更新事件（ContextRefreshedEvent）：该事件会在 ApplicationContext 被初始化或者更新时发布。也可以在调用 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。

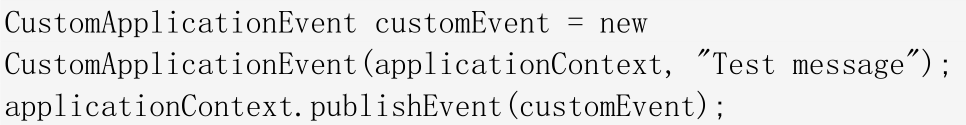
除了上面介绍的事件以外，还可以通过扩展 ApplicationEvent 类来开发自定义的事件。



为了监听这个事件，还需要创建一个监听器：



之后通过 applicationContext 接口的 publishEvent()方法来发布自定义事件。



## 24 、FileSystemResource 和 和 ClassPathResource

在 FileSystemResource 中需要给出 spring-config.xml 文件在你项目中的相对路径或者

绝对路径。在 ClassPathResource 中 spring 会在 ClassPath 中自动搜寻配置文件，所以要把

ClassPathResource 文件放在 ClassPath 下。

如果将 spring-config.xml 保存在了 src 文件夹下的话，只需给出配置文件的名称即可，因为

src 文件夹是默认。

简而言之，ClassPathResource 在环境变量中读取配置文件，FileSystemResource 在配置文件

中读取配置文件。

## 25 、Spring 框架中都用到了哪些设计模式？

Spring 框架中使用到了大量的设计模式，下面列举了比较有代表性的：

o 代理模式—在 AOP 和 remoting 中被用的比较多。

o 单例模式—在 spring 配置文件中定义的 bean 默认为单例模式。

o 模板方法—用来解决代码重复的问题。比如. RestTemplate, JmsTemplate , JpaTempl

ate。

o 前端控制器—Spring 提供了 DispatcherServlet 来对请求进行分发。

o 视图帮助(View Helper )—Spring 提供了一系列的 JSP 标签，高效宏来辅助将分散的代码

整合在视图里。

o 依赖注入—贯穿于 BeanFactory / ApplicationContext 接口的核心理念。

o 工厂模式—BeanFactory 用来创建对象的实例

## 开发中主要使用 Spring 的什么技术 ?

①. IOC 容器管理各层的组件

②. 使用 AOP 配置声明式事务

③. 整合其他框架.

## 简述 AOP 和 IOC 概念 AOP:

Aspect Oriented Program, 面向(方面)切面的编程;Filter(过滤器) 也是一种 AOP. AOP 是一种

新的方法论, 是对传统 OOP(Object-Oriented Programming, 面向对象编程) 的补充. AOP 的

主要编程对象是切面(aspect), 而切面模块化横切关注点.可以举例通过事务说明.

IOC: Invert Of Control, 控制反转. 也成为 DI(依赖注入)其思想是反转 资源获取的方向. 传统

的资源查找方式要求组件向容器发起请求查找资源.作为 回应, 容器适时的返回资源. 而应用了

IOC 之后, 则是容器主动地将资源推送 给它所管理的组件,组件所要做的仅是选择一种合适的方式

来接受资源. 这种行 为也被称为查找的被动形式

## 在 Spring 中如何配置 Bean ?

Bean 的配置方式: 通过全类名（反射）、通过工厂方法（静态工厂方法 & 实 例工厂方法）、

FactoryBean

## IOC 容器对 Bean 的生命周期

①. 通过构造器或工厂方法创建 Bean 实例

②. 为 Bean 的属性设置值和对其他 Bean 的引用

③ . 将 Bean 实 例 传 递 给 Bean 后 置 处 理 器 的 postProcessBeforeInitialization 方

法

④. 调用 Bean 的初始化方法(init-method)

⑤ . 将 Bean 实 例 传 递 给 Bean 后 置 处 理 器 的 postProcessAfterInitialization 方法

⑦. Bean 可以使用了

⑧. 当容器关闭时, 调用 Bean 的销毁方法(destroy-method)

## 31、Spring IOC 如何实现

## 说说 Spring AOP、Spring AOP 实现原理

## 动态代理（CGLib 与 JDK）、优缺点、性能对比、如何选择

## Spring 事务实现方式原理、事务的传播机制、默认的事务类别

## Spring事务失效（事务嵌套），JDK动态代理给Spring事务埋下的坑，可参考《[JDK动态代理给Spring事务埋下的坑！](http://mp.weixin.qq.com/s?__biz=MzI1NDQ3MjQxNA==&mid=2247484940&idx=1&sn=0a0a7198e96f57d610d3421b19573002&chksm=e9c5ffbddeb276ab64ff3b3efde003193902c69acda797fdc04124f6c2a786255d58817b5a5c&scene=21" \l "wechat_redirect" \t "https://mp.weixin.qq.com/_blank)》

## Spring 的单例实现原理

## Spring 框架中用到了哪些设计模式

## Spring中@Autowired和@Resource注解的区别？

## Spring声明一个 bean 如何对其进行个性化定制

## 如果一个接⼝有2个不同的实现, 那么怎么来Autowire一个指定的实现？

(可以使用Qualifier注解限定要注入的Bean，也可以使用Qualifier和Autowire注解指定要获取的bean，也可以使用Resource注解的name属性指定要获取的Bean)

## Spring框架中需要引用哪些jar包，以及这些jar包的用途

## Spring中循环注入是什么意思，可不可以解决，如何解决

## Spring 事务知道吗？有了解过吗

## Spring的声明式事务 @Transaction注解⼀般写在什么位置? 抛出了异常会⾃动回滚吗？有没有办法控制不触发回滚?

## 聊下Spring源码，知道多少，都聊一下

## 聊下Spring注解，@Autowire，@Resource，以及他们的解析过程

## Spring ioc的具体优势，和直接New一个对象有什么区别

## 说下你对 Spring 生态的了解

## 说下你对 Spring AOP 和 IOC 的理解？看过实现原理吗

## 说下 Bean 在 Spring 中的生命周期

## 讲下你知道的 Spring 注解有哪些？该什么场景使用

## 使用 Spring 或者 SpringBoot 有遇到过什么印象深刻的问题吗？当时是怎么解决的

循环依赖问题

## BeanFactory和ApplicationContext

## FactoryBean

## 58 Spring的bean的创建时机？依赖注入的时机？

<https://mp.weixin.qq.com/s/xo2V1a4HF37d6yRBmABB6Q>