# ****一、注解(annotations)列表****

@SpringBootApplication：包含了@ComponentScan、@Configuration和@EnableAutoConfiguration注解。

其中@ComponentScan让spring Boot扫描到Configuration类并把它加入到程序上下文。

@Configuration

等同于spring的XML配置文件；使用Java代码可以检查类型安全。

@EnableAutoConfiguration

自动配置。

@ComponentScan

组件扫描，可自动发现和装配一些Bean。

@Component

可配合CommandLineRunner使用，在程序启动后执行一些基础任务。

@RestController

注解是@Controller和@ResponseBody的合集,表示这是个控制器bean,并且是将函数的返回值直 接填入HTTP响应体中,是REST风格的控制器。

@Autowired

自动导入。

@PathVariable获取参数。

@JsonBackReference解决嵌套外链问题。

@RepositoryRestResourcepublic配合spring-boot-starter-data-rest使用。

# @GetMapping

@GetMapping是一个组合注解, 是@RequestMapping(method = RequestMethod.GET)的缩写.

# @RestController

@RestController注解是由@ResponseBody ＋ @Controller组合在一起使用的, 注意, 如果只是使用@RestController注解中的Controller， 则Controller中的方法无法返回jsp页面，配置的视图解析器InternalResourceViewResolver不起作用，返回的内容就是Return 里的内容。  
    例如：本来应该到success.jsp页面的，则其显示success.

# ****@RequiresPermissions****

例如：

@RequiresPermissions({"file:read", "write:aFile.txt"} )  
void someMethod();

要求subject中必须同时含有file:read和write:aFile.txt的权限才能执行方法someMethod()。否则抛出异常[AuthorizationException](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/authz/AuthorizationException.html)。

SpringBoot使用@RequiresPermissions注解无效的解决办法:

参考文献

<https://blog.csdn.net/qq_36135928/article/details/79477719>

@DependsOn注解是初始化顺序, 先初始化@DependsOn的value, 然后在初始化方法.

****二、注解(annotations)详解****

@SpringBootApplication：申明让spring boot自动给程序进行必要的配置，这个配置等同于：@Configuration ，@EnableAutoConfiguration和 @ComponentScan 三个配置。

1 package com.example.myproject; 2 import org.springframework.boot.SpringApplication; 3 import org.springframework.boot.autoconfigure.SpringBootApplication; 4 5 @SpringBootApplication // same as @Configuration @EnableAutoConfiguration @ComponentScan 6 public class Application { 7 public static void main(String[] args) { 8 SpringApplication.run(Application.class, args); 9 } 10 }

[IMG_257](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

@ResponseBody：表示该方法的返回结果直接写入HTTP response body中，一般在异步获取数据时使用，用于构建RESTful的api。在使用@RequestMapping后，返回值通常解析为跳转路径，加上@esponsebody后返回结果不会被解析为跳转路径，而是直接写入HTTP response body中。比如异步获取json数据，加上@Responsebody后，会直接返回json数据。该注解一般会配合@RequestMapping一起使用。示例代码：

1 @RequestMapping(“/test”) 2 @ResponseBody 3 public String test(){ 4 return”ok”; 5 }

@Controller：用于定义控制器类，在spring项目中由控制器负责将用户发来的URL请求转发到对应的服务接口（service层），一般这个注解在类中，通常方法需要配合注解@RequestMapping。示例代码：

[IMG_258](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 @Controller 2 @RequestMapping(“/demoInfo”) 3 public class DemoController { 4 @Autowired 5 private DemoInfoService demoInfoService; 6 7 @RequestMapping("/hello") 8 public String hello(Map<String,Object> map){ 9 System.out.println("DemoController.hello()"); 10 map.put("hello","from TemplateController.helloHtml"); 11 //会使用hello.html或者hello.ftl模板进行渲染显示. 12 return"/hello"; 13 } 14 }

[IMG_259](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

@RestController：用于标注控制层组件(如struts中的action)，@ResponseBody和@Controller的合集。示例代码：

[IMG_260](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 package com.kfit.demo.web; 2 3 import org.springframework.web.bind.annotation.RequestMapping; 4 import org.springframework.web.bind.annotation.RestController; 5 6 7 @RestController 8 @RequestMapping(“/demoInfo2”) 9 publicclass DemoController2 { 10 11 @RequestMapping("/test") 12 public String test(){ 13 return "ok"; 14 } 15 }

[IMG_261](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

@RequestMapping：提供路由信息，负责URL到Controller中的具体函数的映射。

@EnableAutoConfiguration：SpringBoot自动配置（auto-configuration）：尝试根据你添加的jar依赖自动配置你的Spring应用。例如，如果你的classpath下存在HSQLDB，并且你没有手动配置任何数据库连接beans，那么我们将自动配置一个内存型（in-memory）数据库”。你可以将@EnableAutoConfiguration或者@SpringBootApplication注解添加到一个@Configuration类上来选择自动配置。如果发现应用了你不想要的特定自动配置类，你可以使用@EnableAutoConfiguration注解的排除属性来禁用它们。

@ComponentScan：其实很简单，@ComponentScan主要就是定义****扫描的路径****从中找出标识了****需要装配****的类自动装配到spring的bean容器中,你一定都有用过@Controller，@Service，@Repository注解，查看其源码你会发现，他们中有一个****共同的注解@Component****，没错@ComponentScan注解默认就会装配标识了@Controller，@Service，@Repository，@Component注解的类到spring容器中。当然，这个的前提就是你需要在所扫描包下的类上引入注解。

@Configuration：相当于传统的xml配置文件，如果有些第三方库需要用到xml文件，建议仍然通过@Configuration类作为项目的配置主类——可以使用@ImportResource注解加载xml配置文件。

@Import：用来导入其他配置类。

@ImportResource：用来加载xml配置文件。

@Autowired：自动导入依赖的bean

@Service：一般用于修饰service层的组件

@Repository：使用@Repository注解可以确保DAO或者repositories提供异常转译，这个注解修饰的DAO或者repositories类会被ComponetScan发现并配置，同时也不需要为它们提供XML配置项。

@Bean：用@Bean标注方法等价于XML中配置的bean。

@Value：注入Spring boot application.properties配置的属性的值。示例代码：

1 @Value(value = “#{message}”) 2 private String message;

@Inject：等价于默认的@Autowired，只是没有required属性；

@Component：泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注。

@Bean:相当于XML中的,放在方法的上面，而不是类，意思是产生一个bean,并交给spring管理。

@AutoWired：自动导入依赖的bean。byType方式。把配置好的Bean拿来用，完成属性、方法的组装，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。当加上（required=false）时，就算找不到bean也不报错。

@Qualifier：当有多个同一类型的Bean时，可以用@Qualifier(“name”)来指定。与@Autowired配合使用。@Qualifier限定描述符除了能根据名字进行注入，但能进行更细粒度的控制如何选择候选者，具体使用方式如下：

1 @Autowired 2 @Qualifier(value = “demoInfoService”) 3 private DemoInfoService demoInfoService;

@Resource(name=”name”,type=”type”)：没有括号内内容的话，默认byName。与@Autowired干类似的事。

****三、JPA注解****

@Entity：@Table(name=”“)：表明这是一个实体类。一般用于jpa这两个注解一般一块使用，但是如果表名和实体类名相同的话，@Table可以省略

@MappedSuperClass:用在确定是父类的entity上。父类的属性子类可以继承。

@NoRepositoryBean:一般用作父类的repository，有这个注解，spring不会去实例化该repository。

@Column：如果字段名与列名相同，则可以省略。

@Id：表示该属性为主键。

@GeneratedValue(strategy = GenerationType.SEQUENCE,generator = “repair\_seq”)：表示主键生成策略是sequence（可以为Auto、IDENTITY、native等，Auto表示可在多个数据库间切换），指定sequence的名字是repair\_seq。

@SequenceGeneretor(name = “repair\_seq”, sequenceName = “seq\_repair”, allocationSize = 1)：name为sequence的名称，以便使用，sequenceName为数据库的sequence名称，两个名称可以一致。

@Transient：表示该属性并非一个到数据库表的字段的映射,ORM框架将忽略该属性。如果一个属性并非数据库表的字段映射,就务必将其标示为@Transient,否则,ORM框架默认其注解为@Basic。@Basic(fetch=FetchType.LAZY)：标记可以指定实体属性的加载方式

@JsonIgnore：作用是json序列化时将[Java](http://lib.csdn.net/base/java)bean中的一些属性忽略掉,序列化和反序列化都受影响。

@JoinColumn（name=”loginId”）:一对一：本表中指向另一个表的外键。一对多：另一个表指向本表的外键。

@OneToOne、@OneToMany、@ManyToOne：对应[hibernate](http://lib.csdn.net/base/javaee)配置文件中的一对一，一对多，多对一。

****四、springMVC相关注解****

@RequestMapping：@RequestMapping(“/path”)表示该控制器处理所有“/path”的UR L请求。RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。  
用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。该注解有六个属性：  
params:指定request中必须包含某些参数值是，才让该方法处理。  
headers:指定request中必须包含某些指定的header值，才能让该方法处理请求。  
value:指定请求的实际地址，指定的地址可以是URI Template 模式  
method:指定请求的method类型， GET、POST、PUT、DELETE等  
consumes:指定处理请求的提交内容类型（Content-Type），如application/json,text/html;  
produces:指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回

@RequestParam：用在方法的参数前面。  
@RequestParam  
String a =request.getParameter(“a”)。

@PathVariable:路径变量。如

1 RequestMapping(“user/get/mac/{macAddress}”) 2 public String getByMacAddress(@PathVariable String macAddress){ 3 //do something; 4 }

参数与大括号里的名字一样要相同。

****五、全局异常处理****

@ControllerAdvice：包含@Component。可以被扫描到。统一处理异常。

@ExceptionHandler（Exception.class）：用在方法上面表示遇到这个异常就执行以下方法。

****六、项目中具体配置解析和使用环境****

### @MappedSuperclass：

1.@MappedSuperclass 注解使用在父类上面，是用来标识父类的

2.@MappedSuperclass 标识的类表示其不能映射到数据库表，因为其不是一个完整的实体类，但是它所拥有的属性能够映射在其子类对用的数据库表中

3.@MappedSuperclass 标识的类不能再有@Entity或@Table注解

****@Column：****

1.当实体的属性与其映射的数据库表的列不同名时需要使用@Column标注说明，该属性通常置于实体的属性声明语句之前，还可与 @Id 标注一起使用。

2.@Column 标注的常用属性是name，用于设置映射数据库表的列名。此外，该标注还包含其它多个属性，如：unique、nullable、length、precision等。具体如下：

[IMG_262](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 name属性：name属性定义了被标注字段在数据库表中所对应字段的名称

2 unique属性：unique属性表示该字段是否为唯一标识，默认为false，如果表中有一个字段需要唯一标识，则既可以使用该标记，也可以使用@Table注解中的@UniqueConstraint

3 nullable属性：nullable属性表示该字段是否可以为null值，默认为true

4 insertable属性：insertable属性表示在使用”INSERT”语句插入数据时，是否需要插入该字段的值

5 updateable属性：updateable属性表示在使用”UPDATE”语句插入数据时，是否需要更新该字段的值

6 insertable和updateable属性：一般多用于只读的属性，例如主键和外键等，这些字段通常是自动生成的

7 columnDefinition属性：columnDefinition属性表示创建表时，该字段创建的SQL语句，一般用于通过Entity生成表定义时使用，如果数据库中表已经建好，该属性没有必要使用

8 table属性：table属性定义了包含当前字段的表名

9 length属性：length属性表示字段的长度，当字段的类型为varchar时，该属性才有效，默认为255个字符

10 precision属性和scale属性：precision属性和scale属性一起表示精度，当字段类型为double时，precision表示数值的总长度，scale表示小数点所占的位数

具体如下：  
 1.double类型将在数据库中映射为double类型，precision和scale属性无效  
 2.double类型若在columnDefinition属性中指定数字类型为decimal并指定精度，则最终以columnDefinition为准  
 3.BigDecimal类型在数据库中映射为decimal类型，precision和scale属性有效  
 4.precision和scale属性只在BigDecimal类型中有效

[IMG_263](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

3.@Column 标注的columnDefinition属性: 表示该字段在数据库中的实际类型.通常 ORM 框架可以根据属性类型自动判断数据库中字段的类型,但是对于Date类型仍无法确定数据库中字段类型究竟是DATE,TIME还是TIMESTAMP.此外,String的默认映射类型为VARCHAR,如果要将 String 类型映射到特定数据库的 BLOB 或TEXT字段类型.

4.@Column标注也可置于属性的getter方法之前

****@Getter和@Setter（Lombok）****

@Setter：注解在属性上；为属性提供 setting 方法 @Getter：注解在属性上；为属性提供 getting 方法

扩展：

[IMG_264](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 @Data：注解在类上；提供类所有属性的 getting 和 setting 方法，此外还提供了equals、canEqual、hashCode、toString 方法 2 3 @Setter：注解在属性上；为属性提供 setting 方法 4 5 @Getter：注解在属性上；为属性提供 getting 方法 6 7 @Log4j2 ：注解在类上；为类提供一个 属性名为log 的 log4j 日志对象，和@Log4j注解类似 8 9 @NoArgsConstructor：注解在类上；为类提供一个无参的构造方法 10 11 @AllArgsConstructor：注解在类上；为类提供一个全参的构造方法 12 13 @EqualsAndHashCode:默认情况下，会使用所有非瞬态(non-transient)和非静态(non-static)字段来生成equals和hascode方法，也可以指定具体使用哪些属性。 14 15 @toString:生成toString方法，默认情况下，会输出类名、所有属性，属性会按照顺序输出，以逗号分割。 16 17 @NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor 18 无参构造器、部分参数构造器、全参构造器，当我们需要重载多个构造器的时候，只能自己手写了 19 20 @NonNull：注解在属性上，如果注解了，就必须不能为Null 21 22 @val:注解在属性上，如果注解了，就是设置为final类型，可查看源码的注释知道

[IMG_265](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

****@PreUpdate和@PrePersist****

@PreUpdate

1.用于为相应的生命周期事件指定回调方法。

2.该注释可以应用于实体类，映射超类或回调监听器类的方法。

3.用于setter 如果要每次更新实体时更新实体的属性，可以使用@PreUpdate注释。

4.使用该注释，您不必在每次更新用户实体时显式更新相应的属性。

5.preUpdate不允许您更改您的实体。 您只能使用传递给事件的计算的更改集来修改原始字段值。

@Prepersist

1.查看@PrePersist注释，帮助您在持久化之前自动填充实体属性。

2.可以用来在使用jpa的时记录一些业务无关的字段，比如最后更新时间等等。生命周期方法注解（delete没有生命周期事件）

3.@PrePersist save之前被调用，它可以返回一个DBObject代替一个空的 @PostPersist save到datastore之后被调用

4.@PostLoad 在Entity被映射之后被调用 @EntityListeners 指定外部生命周期事件实现类

#### 实体Bean生命周期的回调事件

[IMG_266](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

方法的标注： @PrePersist @PostPersist @PreRemove @PostRemove @PreUpdate @PostUpdate @PostLoad 。

它们标注在某个方法之前，没有任何参数。这些标注下的方法在实体的状态改变前后时进行调用，相当于拦截器;

pre 表示在状态切换前触发，post 则表示在切换后触发。

@PostLoad 事件在下列情况触发：

1. 执行 EntityManager.find()或 getreference()方法载入一个实体后；

2. 执行 JPA QL 查询过后；

3. EntityManager.refresh( )方法被调用后。

@PrePersist 和 @PostPersist事件在实体对象插入到数据库的过程中发生;

@PrePersist 事件在调用 EntityManager.persist()方法后立刻发生，级联保存也会发生此事件，此时的数据还没有真实插入进数据库。

@PostPersist 事件在数据已经插入进数据库后发生。

@PreUpdate 和 @PostUpdate 事件的触发由更新实体引起, @PreUpdate 事件在实体的状态同步到数据库之前触发，此时的数据还没有真实更新到数据库。

@PostUpdate 事件在实体的状态同步到数据库后触发，同步在事务提交时发生。

@PreRemove 和 @PostRemove 事件的触发由删除实体引起，@ PreRemove 事件在实体从数据库删除之前触发，即调用了 EntityManager.remove()方法或者级联删除

当你在执行各种持久化方法的时候，实体的状态会随之改变，状态的改变会引发不同的生命周期事件。这些事件可以使用不同的注释符来指示发生时的回调函数。

@javax.persistence.PostLoad：加载后。

@javax.persistence.PrePersist：持久化前。

@javax.persistence.PostPersist：持久化后。

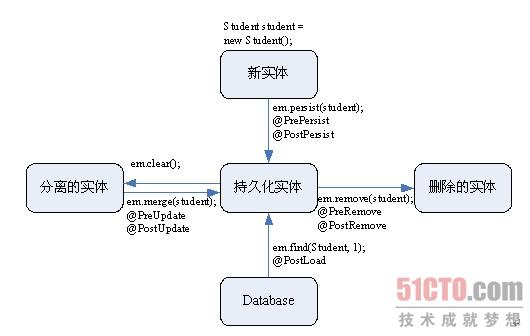
@javax.persistence.PreUpdate：更新前。

@javax.persistence.PostUpdate：更新后。

@javax.persistence.PreRemove：删除前。

@javax.persistence.PostRemove：删除后。

[IMG_267](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)



****1）数据库查询****

@PostLoad事件在下列情况下触发：

执行EntityManager.find()或getreference()方法载入一个实体后。

执行JPQL查询后。

EntityManager.refresh()方法被调用后。

****2）数据库插入****

@PrePersist和@PostPersist事件在实体对象插入到数据库的过程中发生：

@PrePersist事件在调用persist()方法后立刻发生，此时的数据还没有真正插入进数据库。

@PostPersist事件在数据已经插入进数据库后发生。

****3）数据库更新****

@PreUpdate和@PostUpdate事件的触发由更新实体引起：

@PreUpdate事件在实体的状态同步到数据库之前触发，此时的数据还没有真正更新到数据库。

@PostUpdate事件在实体的状态同步到数据库之后触发，同步在事务提交时发生。

****4）数据库删除****

@PreRemove和@PostRemove事件的触发由删除实体引起：

@PreRemove事件在实体从数据库删除之前触发，即在调用remove()方法删除时发生，此时的数据还没有真正从数据库中删除。

@PostRemove事件在实体从数据库中删除后触发。

### @NoArgsConstructor & @AllArgsConstructor（lombok）

@NoArgsConstructor，提供一个无参的构造方法。

@AllArgsConstructor，提供一个全参的构造方法。

****@Configuration & @bean****[1.@Configuration](mailto:1.@Configuration)标注在类上，相当于把该类作为spring的xml配置文件中的<beans>，作用为：配置spring容器(应用上下文)

[IMG_269](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 package com.test.spring.support.configuration; 2 3 @Configuration 4 public class TestConfiguration { 5 public TestConfiguration(){ 6 System.out.println("spring容器启动初始化。。。"); 7 } 8 }

[IMG_270](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

相当于：

[IMG_271](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 <?xml version="1.0" encoding="UTF-8"?> 2 <beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" 3 xmlns:context="http://www.springframework.org/schema/context" xmlns:jdbc="http://www.springframework.org/schema/jdbc" 4 xmlns:jee="http://www.springframework.org/schema/jee" xmlns:tx="http://www.springframework.org/schema/tx" 5 xmlns:util="http://www.springframework.org/schema/util" xmlns:task="http://www.springframework.org/schema/task" xsi:schemaLocation=" 6 http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd 7 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd 8 http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-4.0.xsd 9 http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-4.0.xsd 10 http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd 11 http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd 12 http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task-4.0.xsd" default-lazy-init="false"> 13 14 15 </beans>

[IMG_272](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

主方法进行测试：

[IMG_273](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 package com.test.spring.support.configuration; 2 3 public class TestMain { 4 public static void main(String[] args) { 5 6 //@Configuration注解的spring容器加载方式，用AnnotationConfigApplicationContext替换ClassPathXmlApplicationContext 7 ApplicationContext context = new AnnotationConfigApplicationContext(TestConfiguration.class); 8 9 //如果加载spring-context.xml文件： 10 //ApplicationContext context = new ClassPathXmlApplicationContext("spring-context.xml"); 11 } 12 }

[IMG_274](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

从运行主方法结果可以看出，spring容器已经启动了：

1 八月 11, 2016 12:04:11 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh 2 信息: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@203e25d3: startup date [Thu Aug 11 12:04:11 CST 2016]; root of context hierarchy 3 spring容器启动初始化。。。

[2.@Bean](mailto:2.@Bean)标注在方法上(返回某个实例的方法)，等价于spring的xml配置文件中的<bean>，作用为：注册bean对象

bean类：

[IMG_275](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 package com.test.spring.support.configuration; 2 3 public class TestBean { 4 5 public void sayHello(){ 6 System.out.println("TestBean sayHello..."); 7 } 8 9 public String toString(){ 10 return "username:"+this.username+",url:"+this.url+",password:"+this.password; 11 } 12 13 public void start(){ 14 System.out.println("TestBean 初始化。。。"); 15 } 16 17 public void cleanUp(){ 18 System.out.println("TestBean 销毁。。。"); 19 } 20 }

[IMG_276](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

配置类：

[IMG_277](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 package com.test.spring.support.configuration; 2 3 @Configuration 4 public class TestConfiguration { 5 public TestConfiguration(){ 6 System.out.println("spring容器启动初始化。。。"); 7 } 8 9 //@Bean注解注册bean,同时可以指定初始化和销毁方法 10 //@Bean(name="testNean",initMethod="start",destroyMethod="cleanUp") 11 @Bean 12 @Scope("prototype") 13 public TestBean testBean() { 14 return new TestBean(); 15 } 16 }

[IMG_278](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

主方法测试类：

[IMG_279](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 package com.test.spring.support.configuration; 2 3 public class TestMain { 4 public static void main(String[] args) { 5 ApplicationContext context = new AnnotationConfigApplicationContext(TestConfiguration.class); 6 //获取bean 7 TestBean tb = context.getBean("testBean"); 8 tb.sayHello(); 9 } 10 }

[IMG_280](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

****注：**  
**(1)、@Bean注解在返回实例的方法上，如果未通过@Bean指定bean的名称，则默认与标注的方法名相同；**  
**(2)、@Bean注解默认作用域为单例singleton作用域，可通过@Scope(“prototype”)设置为原型作用域；**  
**(3)、既然@Bean的作用是注册bean对象，那么完全可以使用@Component、@Controller、@Service、@Ripository等注解注册bean，当然需要配置@ComponentScan注解进行自动扫描。****

bean类：

[IMG_281](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 package com.test.spring.support.configuration; 2 3 //添加注册bean的注解 4 @Component 5 public class TestBean { 6 7 public void sayHello(){ 8 System.out.println("TestBean sayHello..."); 9 } 10 11 public String toString(){ 12 return "username:"+this.username+",url:"+this.url+",password:"+this.password; 13 } 14 }

[IMG_282](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

配置类：

[IMG_283](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1

//开启注解配置

2 @Configuration 3 //添加自动扫描注解，basePackages为TestBean包路径 4 @ComponentScan(basePackages = "com.test.spring.support.configuration") 5 public class TestConfiguration { 6 public TestConfiguration(){ 7 System.out.println("spring容器启动初始化。。。"); 8 } 9 10 //取消@Bean注解注册bean的方式 11 //@Bean 12 //@Scope("prototype") 13 //public TestBean testBean() { 14 // return new TestBean(); 15 //} 16 }

[IMG_284](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

主方法测试获取bean对象：

[IMG_285](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 public class TestMain { 2 public static void main(String[] args) { 3 ApplicationContext context = new AnnotationConfigApplicationContext(TestConfiguration.class); 4 //获取bean 5 TestBean tb = context.getBean("testBean"); 6 tb.sayHello(); 7 } 8 }

[IMG_286](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

sayHello()方法都被正常调用。

### **使用@Configuration注解来代替Spring的bean配置**

下面是一个典型的Spring配置文件（application-config.xml）：

[IMG_287](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 <beans> 2 <bean id="orderService" class="com.acme.OrderService"/> 3 <constructor-arg ref="orderRepository"/> 4 </bean> 5 <bean id="orderRepository" class="com.acme.OrderRepository"/> 6 <constructor-arg ref="dataSource"/> 7 </bean> 8 </beans>

[IMG_288](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

然后你就可以像这样来使用是bean了：

1 ApplicationContext ctx = new ClassPathXmlApplicationContext("application-config.xml"); 2 OrderService orderService = (OrderService) ctx.getBean("orderService");

现在Spring Java Configuration这个项目提供了一种通过java代码来装配bean的方案：

[IMG_289](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

1 @Configuration 2 public class ApplicationConfig { 3 4 public @Bean OrderService orderService() { 5 return new OrderService(orderRepository()); 6 } 7 8 public @Bean OrderRepository orderRepository() { 9 return new OrderRepository(dataSource()); 10 } 11 12 public @Bean DataSource dataSource() { 13 // instantiate and return an new DataSource … 14 } 15 }

[IMG_290](https://www.cnblogs.com/ldy-blogs/p/javascript:void(0);)

然后你就可以像这样来使用是bean了：

1 JavaConfigApplicationContext ctx = new JavaConfigApplicationContext(ApplicationConfig.class); 2 OrderService orderService = ctx.getBean(OrderService.class);