Spring Boot 常用注解汇总

目录

[一、启动注解 @SpringBootApplication 2](#_Toc85907753)

[@SpringBootConfiguration 2](#_Toc85907754)

[@EnableAutoConfiguration 2](#_Toc85907755)

[二、Controller 相关注解 2](#_Toc85907756)

[@RestController 复合注解 2](#_Toc85907757)

[@RequestBody 3](#_Toc85907758)

[三、取请求参数值 4](#_Toc85907759)

[四、注入bean相关 5](#_Toc85907760)

[@Repository 5](#_Toc85907761)

[@Service 5](#_Toc85907762)

[@Scope作用域注解 6](#_Toc85907763)

[@Entity实体类注解 6](#_Toc85907764)

[@Bean产生一个bean的方法 7](#_Toc85907765)

[@Autowired 自动导入 7](#_Toc85907766)

[@Component 7](#_Toc85907767)

[五、导入配置文件 7](#_Toc85907768)

[@PropertySource注解 7](#_Toc85907769)

[@ImportResource导入xml配置文件 7](#_Toc85907770)

[@Value 8](#_Toc85907771)

[@Import 导入额外的配置信息 8](#_Toc85907772)

[六、事务注解 @Transactional 8](#_Toc85907773)

[七、全局异常处理 8](#_Toc85907774)

[@ControllerAdvice 统一处理异常 8](#_Toc85907775)

<https://www.cnblogs.com/tqlin/p/11687811.html>

# 一、启动注解 @SpringBootApplication

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Inherited

@SpringBootConfiguration

@EnableAutoConfiguration

@ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM， classes = TypeExcludeFilter.class)，

@Filter(type = FilterType.CUSTOM， classes = AutoConfigurationExcludeFilter.class) })

public @interface SpringBootApplication {

// ... 此处省略源码

}

查看源码可发现，**@SpringBootApplication是一个复合注解，包含了@SpringBootConfiguration，@EnableAutoConfiguration，@ComponentScan这三个注解。**

## @SpringBootConfiguration

@SpringBootConfiguration 注解，继承@Configuration注解，主要用于加载配置文件

@SpringBootConfiguration继承自@Configuration，二者功能也一致，标注当前类是配置类， 并会将当前类内声明的一个或多个以@Bean注解标记的方法的实例纳入到spring容器中，并且实例名就是方法名。

## @EnableAutoConfiguration

@EnableAutoConfiguration 注解，开启自动配置功能

@EnableAutoConfiguration可以帮助SpringBoot应用将所有符合条件的@Configuration配置都加载到当前SpringBoot创建并使用的IoC容器。借助于Spring框架原有的一个工具类：SpringFactoriesLoader的支持，@EnableAutoConfiguration可以智能的自动配置功效才得以大功告成

@ComponentScan 注解，主要用于组件扫描和自动装配

@ComponentScan的功能其实就是自动扫描并加载符合条件的组件或bean定义，最终将这些bean定义加载到容器中。我们可以通过basePackages等属性指定@ComponentScan自动扫描的范围，如果不指定，则默认Spring框架实现从声明@ComponentScan所在类的package进行扫描，默认情况下是不指定的，所以SpringBoot的启动类最好放在root package下。

# 二、Controller 相关注解

@Controller控制器，处理http请求。

## @RestController 复合注解

查看@RestController源码：

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Controller

@ResponseBody

public @interface RestController {

/\*\*

\* The value may indicate a suggestion for a logical component name，

\* to be turned into a Spring bean in case of an autodetected component.

\* @return the suggested component name， if any (or empty String otherwise)

\* @since 4.0.1

\*/

@AliasFor(annotation = Controller.class)

String value() default "";

}

从源码我们知道，**@RestController注解相当于@ResponseBody+@Controller合在一起的作用，RestController使用的效果是将方法返回的对象直接在浏览器上展示成json格式**.

## @RequestBody

通过HttpMessageConverter读取Request Body并反序列化为Object（泛指）对象

@RequestMapping

@RequestMapping 是 Spring Web 应用程序中最常被用到的注解之一。这个注解会将 HTTP 请求映射到 MVC 和 REST 控制器的处理方法上

@GetMapping用于将HTTP get请求映射到特定处理程序的方法注解

注解简写：@RequestMapping(value = "/say"，method = RequestMethod.GET)等价于：@GetMapping(value = "/say")

GetMapping源码：

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@RequestMapping(method = RequestMethod.GET)

public @interface GetMapping {

//...

}

是@RequestMapping(method = RequestMethod.GET)的缩写

@PostMapping用于将HTTP post请求映射到特定处理程序的方法注解

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@RequestMapping(method = RequestMethod.POST)

public @interface PostMapping {

//...

}

是@RequestMapping(method = RequestMethod.POST)的缩写

# 三、取请求参数值

@PathVariable：获取url中的数据。

@Controller

@RequestMapping("/User")

public class HelloWorldController {

@RequestMapping("/getUser/{uid}")

public String getUser(@PathVariable("uid") Integer id, Model model) {

System.out.println("id:"+id);

return "user";

}

}

请求示例：http://localhost:8080/User/getUser/123

@RequestParam：获取请求参数的值。

@Controller

@RequestMapping("/User")

public class HelloWorldController {

@RequestMapping("/getUser")

public String getUser(@RequestParam("uid")Integer id, Model model) {

System.out.println("id:"+id);

return "user";

}

}

请求示例：http://localhost:8080/User/getUser?uid=123

@RequestHeader 把Request请求header部分的值绑定到方法的参数上

@CookieValue 把Request header中关于cookie的值绑定到方法的参数上

# 四、注入bean相关

## @Repository

**DAO层注解，DAO层中接口继承JpaRepository<T，ID extends Serializable>，需要在build.gradle中引入相关jpa的一个jar自动加载。**

Repository注解源码：

@Target({ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Component

public @interface Repository {

/\*\*

\* The value may indicate a suggestion for a logical component name，

\* to be turned into a Spring bean in case of an autodetected component.

\* @return the suggested component name， if any (or empty String otherwise)

\*/

@AliasFor(annotation = Component.class)

String value() default "";

}

## @Service

@Target({ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Component

public @interface Service {

/\*\*

\* The value may indicate a suggestion for a logical component name，

\* to be turned into a Spring bean in case of an autodetected component.

\* @return the suggested component name， if any (or empty String otherwise)

\*/

@AliasFor(annotation = Component.class)

String value() default "";

}

**@Service是@Component注解的一个特例，作用在类上**

**@Service注解作用域默认为单例。使用注解配置和类路径扫描时，被@Service注解标注的类会被Spring扫描并注册为Bean。**

**@Service用于标注服务层组件，表示定义一个bean。**

@Service使用时没有传参数，Bean名称默认为当前类的类名，首字母小写。

@Service(“serviceBeanId”)或@Service(value=”serviceBeanId”)使用时传参数，使用value作为Bean名字。

## @Scope作用域注解

@Scope作用在类上和方法上，用来配置 spring bean 的作用域，它标识 bean 的作用域

@Scope源码：

@Target({ElementType.TYPE， ElementType.METHOD})

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface Scope {

/\*\*

\* Alias for {@link #scopeName}.

\* @see #scopeName

\*/

@AliasFor("scopeName")

String value() default "";

@AliasFor("value")

String scopeName() default "";

ScopedProxyMode proxyMode() default ScopedProxyMode.DEFAULT;

}

属性介绍

value

singleton 表示该bean是单例的。(默认)

prototype 表示该bean是多例的，即每次使用该bean时都会新建一个对象。

request 在一次http请求中，一个bean对应一个实例。

session 在一个httpSession中，一个bean对应一个实例。

proxyMode

DEFAULT 不使用代理。(默认)

NO 不使用代理，等价于DEFAULT。

INTERFACES 使用基于接口的代理(jdk dynamic proxy)。

TARGET\_CLASS 使用基于类的代理(cglib)。

## @Entity实体类注解

@Table(name ="数据库表名")，这个注解也注释在实体类上，对应数据库中相应的表。

@Id、@Column注解用于标注实体类中的字段，pk字段标注为@Id，其余@Column。

## @Bean产生一个bean的方法

@Bean明确地指示了一种方法，产生一个bean的方法，并且交给Spring容器管理。支持别名@Bean("xx-name")。

## @Autowired 自动导入

@Autowired注解作用在构造函数、方法、方法参数、类字段以及注解上

@Autowired注解可以实现Bean的自动注入

## @Component

把普通pojo实例化到spring容器中，相当于配置文件中的。

虽然有了@Autowired，但是我们还是要写一堆bean的配置文件，相当麻烦，而@Component就是告诉spring，我是pojo类，把我注册到容器中吧，spring会自动提取相关信息。那么我们就不用写麻烦的xml配置文件了

# 五、导入配置文件

## @PropertySource注解

引入单个properties文件：

@PropertySource(value = {"classpath : xxxx/xxx.properties"})

引入多个properties文件：

@PropertySource(value = {"classpath : xxxx/xxx.properties"，"classpath : xxxx.properties"})

## @ImportResource导入xml配置文件

可以额外分为两种模式 相对路径classpath，绝对路径（真实路径）file

注意：单文件可以不写value或locations，value和locations都可用相对路径（classpath）

引入单个xml配置文件：@ImportSource("classpath : xxx/xxxx.xml")

引入多个xml配置文件：@ImportSource(locations={"classpath : xxxx.xml" ， "classpath : yyyy.xml"})

绝对路径（file）

引入单个xml配置文件：@ImportSource(locations= {"file : d:/hellxz/dubbo.xml"})

引入多个xml配置文件：@ImportSource(locations= {"file : d:/hellxz/application.xml" ， "file : d:/hellxz/dubbo.xml"})

## @Value

取值：使用@Value注解取配置文件中的值

@Value("${properties中的键}")

private String xxx;

## @Import 导入额外的配置信息

功能类似XML配置的，用来导入配置类，可以导入带有@Configuration注解的配置类或实现了ImportSelector/ImportBeanDefinitionRegistrar。

使用示例

@SpringBootApplication

@Import({SmsConfig.class})

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class， args);

}

}

# 六、事务注解 @Transactional

在Spring中，事务有两种实现方式，分别是**编程式事务管理**和**声明式事务管理**两种方式。

编程式事务管理： 编程式事务管理使用TransactionTemplate或者直接使用底层的PlatformTransactionManager。对于编程式事务管理，spring推荐使用TransactionTemplate。

声明式事务管理： 建立在AOP之上的。其本质是对方法前后进行拦截，然后在目标方法开始之前创建或者加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务，通过@Transactional就可以进行事务操作，更快捷而且简单。推荐使用

# 七、全局异常处理

## @ControllerAdvice 统一处理异常

@ControllerAdvice 注解定义全局异常处理类

@ControllerAdvice

public class GlobalExceptionHandler {

}

@ExceptionHandler 注解声明异常处理方法

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(Exception.class)

@ResponseBody

String handleException(){

return "Exception Deal!";

}

}