1. 常用注解

|  |  |
| --- | --- |
| 注解 | 功能 |
| [@PostConstruct](https://www.cnblogs.com/cherishforchen/p/11330832.html)  修饰一个非静态的void（）方法 | java5的时候引入的注解，指的是在项目启动的时候执行这个方法，也可以理解为在spring容器启动的时候执行，可作为一些数据的常规化加载，比如数据字典之类的。被@PostConstruct修饰的方法会在服务器加载Servlet的时候运行，并且只会被服务器执行一次。PostConstruct在构造函数之后执行 |
| @RestController | 相当于@ResponseBody ＋ @Controller合在一起的作用  <https://www.cnblogs.com/clwydjgs/p/9255046.html> |
| @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。 |

1. 常用方法：

|  |  |
| --- | --- |
|  |  |
|  |  |

1. **CompletableFuture（异步多线程操作）**

|  |
| --- |
| List<CompletableFuture> futures = Lists.*newArrayList*();  *//异步获取视频列表* CompletableFuture<NewsCardResponse> videoCardResponseCompletableFuture =  CompletableFuture.***supplyAsync***(() -> **newsBiz**.completableFutureGetVideoChannelCard(userId, paramPair, uuid, bean, request), **completableFutureExecutorService**); futures.add(videoCardResponseCompletableFuture); |

<https://www.cnblogs.com/xiangnanl/p/9939447.html>

supplyAsync是待返回结果的异步任务

1. **Single.create**
2. **Springboot URL路径匹配的问题**

|  |  |
| --- | --- |
| @RequestMapping(value = **"/channel/video"**, method = {RequestMethod.***POST***, RequestMethod.***GET***})  A | @RequestMapping(value = **"/channel/video/mini"**)  C |
| @RequestMapping(value = **"/channel"**, method = {RequestMethod.***POST***, RequestMethod.***GET***})  B | @RequestMapping(value = **"/channel/rec"**, method = {RequestMethod.***POST***, RequestMethod.***GET***})  D |

当用户请求 /channel/rec时，被D方法（不是C方法）拦截

当用户请求/channel/video/mini时，被A方法（不是B方法）拦截

原因：C方法的RequestMapping中没有指明拦截的是GET、POST还是两者都有

1. 配置文件中的值注入JavaBean

Application.properties

|  |
| --- |
| **student.stuName**=**vera student.address.province**=**北京 student.address.city**=**海淀 student.address.street**=**清河** |

Bean model

|  |
| --- |
| */\* @Component: 当前类存放读取yml配置文件的数据，要求当前类也在IOC容器中  \* @ConfigurationProperties(prefix="student"):表示和yml配置文件对应，读取其中数据  \* prefix 属性表示和yml配置文件案中以“student”开头的配置项对应\* \*/* @Component @ConfigurationProperties(prefix = **"student"**) @PropertySource(value = **"classpath:application.properties"**,encoding = **"UTF-8"**) **public class** Student {  **private** Integer **stuId**;  **private** String **stuName**;  **private** Boolean **graduated**;  **private** String[] **subject**;  **private** Date **birthday**;  **private** Map<String, String> **teachers**;  **public void** setStuName(String stuName) {  **this**.**stuName** = stuName;  }   **public** String getStuName() {  **return stuName**;  } } |
| @RunWith(SpringRunner.**class**) @SpringBootTest **public class** MySpringBootTest {  @Autowired  **private** Student **mystudent**;  Logger **logger**= LoggerFactory.*getLogger*(MySpringBootTest.**class**);   @Test  **public void** testReadYaml(){  **logger**.info(**mystudent**.getStuName());  } } |
|  |

1. 使用 value注入值

|  |
| --- |
| Application.properties  **atguigu.best.wishes**:**"你好"** |
| @RunWith(SpringRunner.**class**) @SpringBootTest **public class** MySpringBootTest {  @Value(**"${atguigu.best.wishes}"**)  **private** String **wishes**;   Logger **logger**= LoggerFactory.*getLogger*(MySpringBootTest.**class**);  *//测试配置文件中的值注入java bean* @Test  **public void** testReadYaml(){  **logger**.info(**mystudent**.getStuName()+**",province = "**+**mystudent**.getAddress().getProvince());  }   *//测试value注解* @Test  **public void** testReadSimpleValue(){  **logger**.info(**wishes**);  } } |

1. log 日志级别的问题

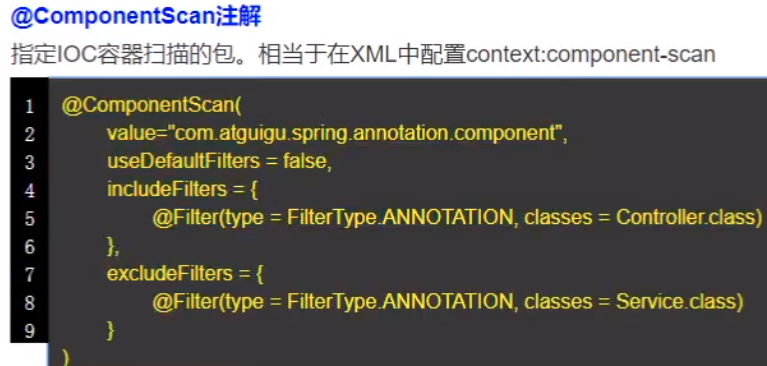
|  |
| --- |
| *# 非常不建议使用root给Springboot设置全局范围的日志级别，影响范围台大。没有特殊的需要和iashi保持默认级别INFO # logging.level.root=debug # 如果局部代码需要设置日志级别，那么使用“包名加级别”方式局部指定* |

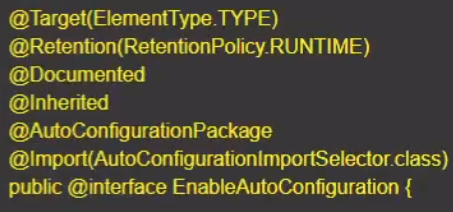
Springboot 注解：



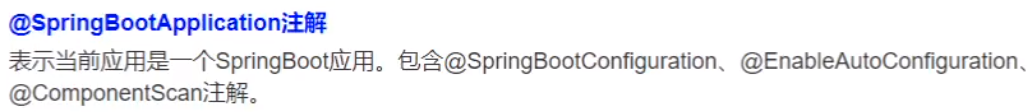












SpringBoot基本原理



