链滴

SpringBoot 中的注解学习

作者：flowaters

原文链接：https://ld246.com/article/1531907304802

来源网站：链滴

背景

SpringBoot依赖注解来简化了参数配置，通过使用默认值实现了工程可以开箱即用。

本文来从注解的角度，学习一下SpringBoot。

罗列的内容较大，使用时只需要有个大概印象即可。

SpringBoot中基本注解

新建一个空应用时，遇到的注解有：

- 1 SpringBootApplication
- 1.1 SpringBootConfiguration
- 1.1.1 Configuration
- 1.1.1.1 Component
- 1.2 EnableAutoConfiguration
- 1.2.1 AutoConfigurationPackage
- 1.2.2 Import
- 1.3 ComponentScan
- 1.4 AliasFor
- 2 RunWith: 测试相关
- 3 SpringBootTest: 测试相关
- 3.1 BootstrapWith

SpringBootApplication

SpringBootApplication是一个组合注解，包含下列3个注解：

- SpringBootConfiguration
- EnableAutoConfiguration
- ComponentScan

同时声明了4个接口

- exclude(): 排除auto-configuration类
- excludeName(): 排除auto-configuration类名
- scanBasePackages(): 自动扫描注解类的基类
- scanBasePackageClasses(): 自动扫描注解类的类列表

用来灵活的配置不进行auto-configuration的类，和扫描注解的类。

SpringBootConfiguration

表明SpringBoot的配置可以从Configuration中自动得到。

被标注的类等于在spring的XML配置文件中(applicationContext.xml)，装配所有bean事务，提供了个spring的上下文环境

应用应该只包含一个SpringBootConfiguration，基本所有的SpringBoot应用都会继承这个类。

Configuration

Configuration表明类中定义了一些Bean方法，可以通过Spring来管理生成Bean的定义和运行时的求。

```
@Configuration
public class AppConfig {
    @Bean
    public MyBean myBean() {
        // instantiate, configure and return bean ...
    }
}
```

Bean的发现

Spring怎么发现Bean呢?

AnnotationConfigApplicationContext

第一种方法是通过AnnotationConfigApplicationContext，示例如下:

```
AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
ctx.register(AppConfig.class);
ctx.refresh();
MyBean myBean = ctx.getBean(MyBean.class);
// use myBean ...
```

spring beans xml

第二种方式是通过spring的<beans>xml，如下:

```
<beans>
  <context:annotation-config/>
  <bean class="com.acme.AppConfig"/>
</beans>
```

SomeBean

第三种方式是通过组件扫描，如下:

```
@Configuration
public class AppConfig {
    private final SomeBean someBean;
```

```
    public AppConfig(SomeBean someBean) {
        this.someBean = someBean;
    }

    // @Bean definition using "SomeBean"
}
```

也可以通过ComponentScan注解来扫描

```
@Configuration
@ComponentScan("com.acme.app.services")
public class AppConfig {
    // various @Bean definitions ...
}
```

Bean的调用

怎么样在外面使用呢?

通过Environment API

方法一：通过Environment API，如下：

```
@Configuration
public class AppConfig {

    @Autowired Environment env;

    @Bean
    public MyBean myBean() {
        MyBean myBean = new MyBean();
        myBean.setName(env.getProperty("bean.name"));
        return myBean;
    }
}
```

也可以同时通过PropertySource来指定Configuration中的配置，示例如下：

```
@Configuration
@PropertySource("classpath:/com/acme/app.properties")
public class AppConfig {

    @Inject Environment env;

    @Bean
    public MyBean myBean() {
        return new MyBean(env.getProperty("bean.name"));
    }
}
```

通过Value注释

方法二：通过@Value注解，示例如下：

```
@Configuration
@PropertySource("classpath:/com/acme/app.properties")
public class AppConfig {

    @Value("${bean.name}") String beanName;

    @Bean
    public MyBean myBean() {
        return new MyBean(beanName);
    }
}
```

在启用了Spring的PropertySourcesPlaceholderConfigurer后，这样使用就很方便了。

编写Configuration类

怎么样编写一个Configuration类呢?

import注解

方法一：通过import注解

```
@Configuration
public class DatabaseConfig {

    @Bean
    public DataSource dataSource() {
        // instantiate, configure and return DataSource
    }
}

@Configuration
@Import(DatabaseConfig.class)
public class AppConfig {

    private final DatabaseConfig dataConfig;

    public AppConfig(DatabaseConfig dataConfig) {
        this.dataConfig = dataConfig;
    }

    @Bean
    public MyBean myBean() {
        // reference the dataSource() bean method
        return new MyBean(dataConfig.dataSource());
    }
}
```

这样，通过注册AppConfig，可以同时使用AppConfig和引入的DatabaseConfig。

```
new AnnotationConfigApplicationContext(AppConfig.class);
```

通过Profile注解

方法二，通过@Profile注解

通过@Profile来表明只有profile(s)活跃时，才来处理。

```
@Profile("development")
 @Configuration
 public class EmbeddedDatabaseConfig {

    @Bean
    public DataSource dataSource() {
        // instantiate, configure and return embedded DataSource
    }
}

 @Profile("production")
 @Configuration
 public class ProductionDatabaseConfig {

    @Bean
    public DataSource dataSource() {
        // instantiate, configure and return production DataSource
    }
}
```

也可以通过@Bean注释，来实现profile的条件执行。

```
@Configuration
 public class ProfileDatabaseConfig {

    @Bean("dataSource")
    @Profile("development")
    public DataSource embeddedDatabase() { ... }

    @Bean("dataSource")
    @Profile("production")
    public DataSource productionDatabase() { ... }
}
```

通过Spring xml和ImportResource注解

方法三，通过Spring xml和ImportResource注解，如：

```
@Configuration
 @ImportResource("classpath:/com/acme/database-config.xml")
 public class AppConfig {

    @Inject DataSource dataSource; // from XML

    @Bean
    public MyBean myBean() {
        // inject the XML-defined dataSource bean
        return new MyBean(this.dataSource);
    }
}
```

通过嵌套的Configuration类

方法四，通过嵌套的Configuration类，如下：

```
@Configuration
 public class AppConfig {

    @Inject DataSource dataSource;

    @Bean
    public MyBean myBean() {
        return new MyBean(dataSource);
    }

    @Configuration
    static class DatabaseConfig {
        @Bean
        DataSource dataSource() {
            return new EmbeddedDatabaseBuilder().build();
        }
    }
 }
```

嵌套的情况，只需要注册AppConfig即可。

lazy启动

默认情况下，@Bean方法会在容器启动时初始化。如果不想这样做的话，可以通过增加@Lazy注解来实现lazy初始化。

在测试中使用

示例

```
@RunWith(SpringJUnit4ClassRunner.class)
 @ContextConfiguration(classes={AppConfig.class, DatabaseConfig.class})
 public class MyTests {

    @Autowired MyBean myBean;

    @Autowired DataSource dataSource;

    @Test
    public void test() {
        // assertions against myBean ...
    }
 }
```

启用内置Spring特征

通过@Enable注解来启动Spring特征，比如：

- @EnableAsync
- @EnableScheduling
- @EnableTransactionManagement
- @EnableAspectJAutoProxy
- @EnableWebMvc

Component

表明这个类，在从包路径搜索类的注解时，可以被搜索到。

EnableAutoConfiguration

允许SpringBoot根据应用中声明的依赖，对Spring框架上下文的进行自动配置。

使用SpringBootApplication时，会自动启用EnableAutoConfiguration。不过在类中多写一个EnablAutoConfiguration也不会有副作用。

其中提供了exclude()和excludeName()方法，

AutoConfigurationPackage

表明本注解类会使用AutoConfigurationPackages进行注册。

Import

表明需要引入的Configuration配置类

ComponentScan

表明Configuration类中使用中组件扫描指令。

组件扫描，可自动发现和装配Bean，默认扫描SpringApplication的run方法里的Booter.class所在的路径下文件，所以最好将该启动类放到根包路径下

详情略

AliasFor

AliasFor设置注解的别名。

使用场景

字段别名

示例如下，@ContextConfiguration中，locations和value互为别名。

public @interface ContextConfiguration {

```
    @AliasFor("locations")
    String[] value() default {};

    @AliasFor("value")
    String[] locations() default {};

    // ...
}
```

字段属性别名

示例如下，

```
 @ContextConfiguration
 public @interface XmlTestConfig {

    @AliasFor(annotation = ContextConfiguration.class, attribute = "locations")
    String[] xmlFiles();
}
```

在@XmlTestConfig中，xmlFiles是@ContextConfiguration中locations的别名，即xmlFiles参数覆了@ContextConfiguration中的locations属性。

字段属性字段别名

就是上面两种情况的综合体

```
@ContextConfiguration
 public @interface MyTestConfig {

    @AliasFor(annotation = ContextConfiguration.class, attribute = "locations")
    String[] value() default {};

    @AliasFor(annotation = ContextConfiguration.class, attribute = "locations")
    String[] groovyScripts() default {};

    @AliasFor(annotation = ContextConfiguration.class, attribute = "locations")
    String[] xmlFiles() default {};
}
```

其中value()，groovyScripts()和xmlFiles()互为别名，同是也都是ContextConfiguration中locations别名。

注解中隐式别名

```
@MyTestConfig
 public @interface GroovyOrXmlTestConfig {

    @AliasFor(annotation = MyTestConfig.class, attribute = "groovyScripts")
    String[] groovy() default {};

    @AliasFor(annotation = ContextConfiguration.class, attribute = "locations")
```

```
    String[] xml() default {};
}
```

在@GroovyOrXmlTestConfig中，groovy覆盖了@MyTestConfig中的groovyScripts属性，xml覆了@ContextConfiguration中的locations属性，其中groovy和xml又互为别名，均覆盖了@Context onfiguration中的locations属性。

参考

- springboot 注解整理
- springBoot注解大全
- SpringBoot中常见注解含义总结
- Spring Boot 核心注解与配置文件