走向自动装配

Spring 模式注解装配

模式注解（Stereotype Annotations）

> A **stereotype annotation** is an annotation that is used to declare the role that a component plays within the application. For example, the `@Repository` annotation in the Spring Framework is a marker for any class that fulfills the role or *stereotype* of a repository (also known as Data Access Object or DAO).
>
> `@Component` is a generic stereotype for any Spring-managed component. Any component annotated with `@Component` is a candidate for component scanning. Similarly, any component annotated with an annotation that is itself meta-annotated with `@Component` is also a candidate for component scanning. For example, `@Service` is meta-annotated with `@Component`.

模式注解是一种用于声明在应用中扮演“组件”角色的注解。如 Spring Framework 中的 `@Repository` 标注在任何类上 ，用于扮演仓储角色的模式注解。

`@Component` 作为一种由 Spring 容器托管的通用模式组件，任何被 `@Component` 标准的组件均为组件扫描的候选对象。类似地，凡是被 `@Component` 元标注（**meta-annotated**）的注解，如 `@Service` ，当任何组件标注它时，也被视作组件扫描的候选对象

模式注解举例

Spring Framework 注解	场景说明	起始版本
`@Repository`	数据仓储模式注解	2.0
`@Component`	通用组件模式注解	2.5
`@Service`	服务模式注解	2.5
`@Controller`	Web 控制器模式注解	2.5
`@Configuration`	配置类模式注解	3.0

装配方式

`<context:component-scan>` 方式

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context.xsd">

    <!-- 激活注解驱动特性 -->
    <context:annotation-config />

    <!-- 找寻被 @Component 或者其派生 Annotation 标记的类（Class），将它们注册为 Spring Bean -->
    <context:component-scan base-package="com.imooc.dive.in.spring.boot" />

</beans>
```

`@ComponentScan` 方式

```
@ComponentScan(basePackages = "com.imooc.dive.in.spring.boot")
public class SpringConfiguration {
    ...
}
```

自定义模式注解

`@Component` “派生性”

```
/**
 * 一级 {@link Repository @Repository}
 *
 * @author <a href="mailto:mercyblitz@gmail.com">Mercy</a>
 * @since 1.0.0
 */
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Repository
public @interface FirstLevelRepository {

    String value() default "";

}
```

- `@Component`
  - `@Repository`
    - `FirstLevelRepository`

`@Component` “层次性”

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@FirstLevelRepository
public @interface SecondLevelRepository {

    String value() default "";

}
```

- `@Component`
  - `@Repository`
    - `FirstLevelRepository`
      - `SecondLevelRepository`

Spring @Enable 模块装配

Spring Framework 3.1 开始支持”@Enable 模块驱动“。所谓“模块”是指具备相同领域的功能组件集合， 组合所形成一个独立的单元。比如 Web MVC 模块、AspectJ代理模块、Caching（缓存）模块、JMX（Java 管 理扩展）模块、Async（异步处理）模块等。

`@Enable` 注解模块举例

框架实现	`@Enable` 注解模块	激活模块
Spring Framework	`@EnableWebMvc`	Web MVC 模块
	`@EnableTransactionManagement`	事务管理模块
	`@EnableCaching`	Caching 模块
	`@EnableMBeanExport`	JMX 模块
	`@EnableAsync`	异步处理模块
	`EnableWebFlux`	Web Flux 模块
	`@EnableAspectJAutoProxy`	AspectJ 代理模块
Spring Boot	`@EnableAutoConfiguration`	自动装配模块
	`@EnableManagementContext`	Actuator 管理模块
	`@EnableConfigurationProperties`	配置属性绑定模块
	`@EnableOAuth2Sso`	OAuth2 单点登录模块
Spring Cloud	`@EnableEurekaServer`	Eureka服务器模块
	`@EnableConfigServer`	配置服务器模块
	`@EnableFeignClients`	Feign客户端模块
	`@EnableZuulProxy`	服务网关 Zuul 模块
	`@EnableCircuitBreaker`	服务熔断模块

实现方式

注解驱动方式

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Documented
@Import(DelegatingWebMvcConfiguration.class)
public @interface EnableWebMvc {
}
```

```
@Configuration
public class DelegatingWebMvcConfiguration extends
WebMvcConfigurationSupport {
    ...
}
```

接口编程方式

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Import(CachingConfigurationSelector.class)
public @interface EnableCaching {
...
}
```

```
public class CachingConfigurationSelector extends AdviceModeImportSelector<EnableCaching> {
    /**
    * {@inheritDoc}
    * @return {@link ProxyCachingConfiguration} or {@code
    AspectJCacheConfiguration} for
    * {@code PROXY} and {@code ASPECTJ} values of {@link
    EnableCaching#mode()}, respectively
    */
    public String[] selectImports(AdviceMode adviceMode) {
        switch (adviceMode) {
            case PROXY:
                return new String[] {
AutoProxyRegistrar.class.getName(),ProxyCachingConfiguration.class.getName() };
        case ASPECTJ:
            return new String[] {
                AnnotationConfigUtils.CACHE_ASPECT_CONFIGURATION_CLASS_NAME };
        default:
            return null;
    }
}
```

自定义 `@Enable` 模块

基于注解驱动实现 - `@EnableHelloWorld`

TODO

基于接口驱动实现 - `@EnableServer`

`HelloWorldImportSelector` -> `HelloWorldConfiguration` -> HelloWorld

Spring 条件装配

从 Spring Framework 3.1 开始，允许在 Bean 装配时增加前置条件判断

条件注解举例

Spring 注解	场景说明	起始版本
`@Profile`	配置化条件装配	3.1
`@Conditional`	编程条件装配	4.0

实现方式

配置方式 - `@Profile`

编程方式 - `@Conditional`

```
@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Conditional(OnClassCondition.class)
public @interface ConditionalOnClass {

   /**
    * The classes that must be present. Since this annotation is parsed by loading class
    * bytecode, it is safe to specify classes here that may ultimately not be on the
    * classpath, only if this annotation is directly on the affected component and
    * <b>not</b> if this annotation is used as a composed, meta-annotation. In order to
    * use this annotation as a meta-annotation, only use the {@link #name} attribute.
    * @return the classes that must be present
    */
   Class<?>[] value() default {};

   /**
    * The classes names that must be present.
```

```
     * @return the class names that must be present.
     */
    String[] name() default {};

}
```

自定义条件装配

基于配置方式实现 - `@Profile`

计算服务，多整数求和 sum

@Profile("Java7") : for 循环

@Profile("Java8") : Lambda

基于编程方式实现 - `@ConditionalOnSystemProperty`

Spring Boot 自动装配

在 Spring Boot 场景下，基于约定大于配置的原则，实现 Spring 组件自动装配的目的。其中使用了

底层装配技术

- Spring 模式注解装配
- Spring `@Enable` 模块装配
- Spring 条件装配装配
- Spring 工厂加载机制
    - 实现类：`SpringFactoriesLoader`
    - 配置资源：`META-INF/spring.factories`

自动装配举例

参考 `META-INF/spring.factories`

实现方法

1. 激活自动装配 - `@EnableAutoConfiguration`
2. 实现自动装配 - `XXXAutoConfiguration`
3. 配置自动装配实现 - `META-INF/spring.factories`

自定义自动装配

`HelloWorldAutoConfiguration`

- 条件判断：`user.name == "Mercy"`
- 模式注解：`@Configuration`
- `@Enable` 模块：`@EnableHelloWorld` -> `HelloWorldImportSelector` -> `HelloWorldConfiguration` - > `helloWorld`