Rays's

Personal Study Note

of

Spring

目 录

[1. 装配Bean 1](#_Toc4964752)

[1.1. 自动化装配 1](#_Toc4964753)

[1.2. 代码配置 1](#_Toc4964754)

[1.3. XML配置 2](#_Toc4964755)

[1.4. 导入和混合 2](#_Toc4964756)

[1.4.1. 代码配置 2](#_Toc4964757)

[1.4.2. xml配置 2](#_Toc4964758)

[2. 高级装配 3](#_Toc4964759)

[2.1. 环境和Profile 3](#_Toc4964760)

[2.1.1. 配置 profile 3](#_Toc4964761)

[2.1.2. 激活 profile 3](#_Toc4964762)

# 装配Bean

## 自动化装配

自动化装配的两大要素

* 组件扫描（Component Scan）：自动发现Bean
* 自动装配（Autowiring）：自动处理Bean之间的关系

1.组件扫描

@ComponentScan或者<context:component-scan> 开启自动扫描功能。

使用 @Component 注解标记需要的Bean对应的类（或者使用JDI规范@Named）

2.自动装配

@Autowired 注解表明Bean和其他Bean的依赖关系，可以用于构造函数、set方法和直接用于属性（或者使用JDI规范@Inject）。

## 代码配置

使用第三方的jar包，无法直接使用注解来配置。JavaConfig属于配置代码，不应该夹杂业务，业务代码也不应该包含配置代码。

1.启动代码配置

为某个类添加注解@Configuration，表明某个类是配置类。

2.配置Bean

编写方法创建并返回Bean，为方法添加@Bean注解

@bean

A bean1(){ return new A(); }

3.配置注入

在配置Bean的方法中，加入对其他Bean的依赖即可有两种方式：

一种是配置方法依赖其他Bean的配置方法

B bean2() { return new B(bean1());}

一种是配置方法依赖其他的Bean的参数传入

B bean2(A a) { return new B(a); }

## XML配置

新的Spring项目应该尽量不再使用xml配置，了解xml配置应该用于维护旧项目。

1.规范

Spring的配置xml文件以<beans>元素为根，并且需要在xml顶部声明多个xml模式（XSD）文件。

2.配置<bean>

使用 <bean> 元素配置Bean

3.配置依赖

<constructor-arg> 或 -c 标签配置构造器依赖

<property> 或 -p 配置属性依赖

## 导入和混合

### 代码配置

@Import 引入其他配置类

@ImportResource 引入配置文件

### xml配置

<Import > 元素引入其他配置文件

<bean > 元素直接将配置类作为Bean配置

# 高级装配

## 环境和Profile

对于同一个bean，希望在不同的场景下使用不同的配置方案，可以使用profile功能。

常见的场景就是数据源的配置，就算在普通情况下，也会有生产和测环境的区别，这时候数据源会希望使用不同的地址、用户名等。

### 配置 profile

@Profile注解，可以用于类、配置方法表明配置的生效范围。

如 @Profile("dev") 表明在 dev profile 激活后生效。

在xml中使用 <beans> 元素的 profile 可以配置生效范围。也可以再 <beans> 中嵌套 <beans> 而不用创建多个文件来配置生效范围。

### 激活 profile

Spring 依赖两个属性来确定当前的 profile：

spring.profiles.active和spring.profiles.default

优先使用 active 若该值没有设置则使用default，若default也没有设置，则只创建不包含 profile 的bean。

这两个属性的设置方式有多种

* 作为DispatcherServlet的初始化参数；
* 作为Web应用的上下文参数；
* 作为JNDI条目；
* 作为环境变量；
* 作为JVM的系统属性；
* 在集成测试类上，使用@ActiveProfiles注解设置。

可以同时激活多个profile，一般用于多个profile彼此无关的场景可能需要。

## 条件化的bean

@Conditional 注解用于特殊场景，只有当某个条件符合时才创建某个Bean。该注解使用 Condition 接口进行条件比对。

public interface Condition {

boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata);

}

任何实现该接口的方法都可以用于条件比对，只有matches方法返回true时，才创建bean。matches方法的参数 ConditionContext 也是一个接口，大致方法如下：

* 借助getRegistry()返回的BeanDefinitionRegistry检查bean定义；
* 借助getBeanFactory()返回的ConfigurableListableBeanFactory检查bean是

否存在，甚至探查bean的属性；

* 借助getEnvironment()返回的Environment检查环境变量是否存在以及它的值是什么；
* 读取并探查getResourceLoader()返回的ResourceLoader所加载的资源；
* 借助getClassLoader()返回的ClassLoader加载并检查类是否存在。

AnnotatedTypeMetadata 也是一个接口，可以用来检查要配置的bean是否包含其他注解。在Spring 4 之后的profile 的代码也重构为基于Conditional来创建。

## 处理自动装配歧义

Spring在处理自动装配时，若没有无歧义的选项时，会抛出NoUniqueBeanDefinitionException，

@Primary 用于Bean 定义，标记该Bean是首选Bean

@Qualifier 用于注入定义，标记将注入的值首选某个Bean

## Bean的作用域

默认情况下Spring的所有的Bean均为单例模式。

Spring定义的作用域有：

单例：整个应用中，只创建一个

原型：每次注入或获取均创建新的对象。

会话：web应用中，每个会话中只有一个。

请求：web应用中，每次请求只有一个。

使用 @Scope注解，并配置ConfigurableBeanFactory中的常量，可以设置Bean的作用域。

## 运行时注入

### 注入外部的值

1. 使用 @PropertySource 声明外部属性文件，使用 Environment 类来检索属性值。

2. 属性占位符

在配置中使用 ${ ... } 表明该处是一个属性的值

然后 配置 PropertyPlaceholderConfigurer bean或PropertySourcesPlaceholderConfigurer 来处理处理占位符。

### SpringEL

略

# 面向切面

## 术语

面向切面：希望程序在某处做某事，将这种希望统一表述则形成切面。

通知Advice就是希望做的事情，Spring切面中有5种通知：

1. 前置、2. 后置、3. 返回、4. 异常、5. 环绕

从定义可以看出Spring的通知和Method是密切相关的，实际上就是在方法执行过程中寻找时机完成功能。

连接点Join Point，可以运行通知的时机

切点PointCut，需要运行通知的时机，就是某处

引入，向现有的类添加新方法或属性。

织入，实现新的切面功能，必然要对已经存在的代码进行修改，不过这种修改是模式化的。织入大致可以在以下几个阶段：

编译器：编译类的时候，加入切面的代码，这时候需要特殊的编译器来完成多出的功能。

类加载期：加载类的时候，加入切面的代码，需要特殊的加载器（ClassLoader）

运行期：应用运行的时候，加入切面的代码，一般是通过动态代理实现。Spring就是基于该方式。

## 切点选择

Spring基于动态代理实现AOP，切点都是围绕方法进行。Spring的切点指示器是AspectJ的一个子集，具体如下：

|  |  |
| --- | --- |
| 指示器 | 描述 |
| args() | 指定切点方法的参数类型 |
| @args() | 注解指定切点方法的参数类型 |
| execution() | 指定切点方法 |
| this() | 指定切点方法所在类 |
| target() |  |
| @target() |  |
| within() |  |
| @within() |  |
| @annotation |  |