1. 全注解的Spring IoC
2. Spring IoC

Ioc容器是Spring的核心，可以说Spring是一种基于IOC容器编程的框架。它通过描述来生成或者获取对象，之前是通过XML来描述，这里使用全注解的方式来描述。一个系统可以生成各种对象，这些对象都需要管理，并且它们之间存在着依赖关系，通过描述就能够管理它们之间的关系。

1. IOC容器

在Spring中我们把每一个需要管理的对象成为Spring Bean，而Spring 管理这些Bean的容器，称为Spring IoC容器，IOC容器有两个功能，一个是负责发布和获取Bean，一个是负责管理Bean之间的依赖管理。

1. BeanFactory接口

Spring中所有的IoC容器都要实现接口BeanFactory，它是一个顶级容器接口。翻看源码，可以看到接口中的方法有如下的方法：

1. getBean方法

它的意义是从IOC容器中获取Bean，可以按Bean类型，名称来获取。

1. containsBean方法

判断是否包含某个Bean。

1. isSingleton方法

判断Bean是否在Spring IoC中为单例，默认情况下，Bean都是以单例的形式存在的，即调用getBean方法返回的都是同一个Bean对象。

1. isPrototype方法

判断Bean是否在Spring IoC是原型，即不是单例。如果为true，则Spring IoC容器会创建一个新的Bean返回给调用者。

1. ApplicationContext接口

为了扩展BeanFactory接口，还设计了一个更加高级的接口ApplicationContext，它是BeanFactory接口的子接口之一。在BeanFactory的基础上，扩展了消息国际化接口（MessageSource），环境可配置接口（EnvironmentCapable），应用事件发布接口（ApplicationEventPublisher）和资源模式解析接口（ResourcePatternResolver）。在开发中我们使用过的大部分Spring IoC容器都是ApplicationContext接口的实现类。

1. AnnotaionConfigApplicationContext

Spring Boot中使用全注解来开发，因此使用一个基于注解的IOC容器，它就是AnnotaionConfigApplicationContext。下面用个小Demo说明：

1. POJO

定义一个POJO，User类。

package com.springboot.chap3.pojo;  
  
public class User {  
 private String id;  
 private String userName;  
 private String note;  
  
 public User() {  
 }  
  
 public User(String id, String userName, String note) {  
 this.id = id;  
 this.userName = userName;  
 this.note = note;  
 }  
  
 public String getId() {  
 return id;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
  
 public String getUserName() {  
 return userName;  
 }  
  
 public void setUserName(String userName) {  
 this.userName = userName;  
 }  
  
 public String getNote() {  
 return note;  
 }  
  
 public void setNote(String note) {  
 this.note = note;  
 }  
}

1. 定义一个Java配置类

我们的目的是要将POJO类生成Bean丢到Spring IoC容器中，那么就要创建一个配置类，在类中创建一个Bean配置到Spring Ioc容器中。@Bean注解表示这是一个Bean，@Configuration注解表示这是一个Java配置文件，Spring容器会根据它来生成IOC容器去装配Bean。

package com.springboot.chap3.config;  
  
import com.springboot.chap3.pojo.User;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class AppConfig {  
  
 @Bean(name = "user")  
 public User initUser(){  
 User user = new User();  
 user.setId("aaa");  
 user.setUserName("username1");  
 user.setNote("note1");  
 return user;  
 }  
}

1. 测试类构建IOC容器

使用AnnotationConfigApplicationContext来新建一个IOC容器，将配置类传进去，让IOC容器的构造方法读取配置类，然后将配置里面的Bean装配到IOC容器中。接下来就可以调用IOC容器中的方法了。

package com.springboot.chap3.test;  
  
import com.springboot.chap3.config.AppConfig;  
import com.springboot.chap3.pojo.User;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
import java.util.logging.Logger;  
  
public class IocTest {  
  
 private static Logger *log* = Logger.*getLogger*(String.*valueOf*(IocTest.class));  
  
 public static void main(String[] args) {  
 ApplicationContext atx = new AnnotationConfigApplicationContext(AppConfig.class);  
 User bean = atx.getBean(User.class);  
 *log*.info(bean.getId());  
 }  
}

1. 扫描Bean
2. 概述

之前，我们通过构造一个IOC容器来获取Bean，每个Bean对象都要添加@Bean注解，当Bean对象比较多的时候非常麻烦。这里我们使用@ComponentScan和@Component注解进行扫描装配Bean。

1. @Component注解

这个注解用于哪个类就表明哪个类被扫描进入Spring IoC容器中，注解中的参数是Bean对象的名称，默认是当前类名的第一个字母小写，其他不变。注解@Value指定具体的值，使得Spring IoC给予对应的属性注入对应的值。

@Component("user")  
public class User {  
 @Value("aaa")  
 private String id;  
 @Value("username")  
 private String userName;  
 @Value("note")  
 private String note;

}

1. @ComponentScan注解

这个注解表明采用何种策略来扫描装配Bean对象。可以通过配置项basePackages自定义需要扫描的类，以及设置过滤器，过滤掉不需要的类。配置项includeFilters定义满足过滤器条件的Bean才去扫描，excludeFilters则是排除过滤器条件的Bean，都需要注解@Filter来定义，使用的一般是正则表达式。如：

@ComponentScan(“com.springboot.chapter3.\*”)或

@ComponentScan(basePackages=”com.springboot.chapter3.pojo”)

@ComponentScan(basePackageClasses={User.class})

添加过滤器的@ComponentScan，这样UserService就不会被扫描到：

@Component(basePackages=”com.springboot.chapter3.\*”, excludeFilter = [{@Filter(classes={UserService.class})}](mailto:%7b@Filter(classes=%7bUserService.class%7d)%7d))

1. 自定义第三方Bean

Java应用需要引入第三方的包，且需要将第三方的类对象注入Spring IoC中，这时还是要使用@Bean注解，在需要注入的类上添加@Bean注解，这样就能够扫描到了。

1. 依赖注入

Spring IoC还需要负责Bean之间的依赖关系，这里通过注解来配置。

1. @Autowired

当需要注入一个Bean时，使用@Autowired注解来表示注入，它注入的机制基本的一条是根据类型，但是当一个Bean有多个时，就会出现异常。这时它就会首先根据类型找到Bean，再根据属性名称和Bean的名称进行匹配，如果匹配不成功，则抛出异常。同时，这个注解默认是必须找到对应的Bean对象的，如果不确定它标注的属性一定会存在且允许这个被标注的属性为null时，我们可以设置@Autowired属性required为false，那么该注解就可以不用找到对应的Bean对象了。

1. @Pirmary

当出现注入歧义时，可以使用@Primary注解来修改注入的优先级。在优先级更高的注入类中添加该注解，那么就会首先找到该类来进行注入。

1. @Qualifier

该注解配合@Autowired来使用，能够更加精确地注入，该注解的配置项value需要一个字符串，即需要注入的Bean对象名称，这样就能消除歧义性了。如：

@Autowired

@Qualifier(“dog”)

注解同样可以在参数中使用，如下所示：

public BussinessPerson(@Autowired @Qualifier(“dog”) Animal animal){

this.animal = animal;

}

1. 生命周期

Spring IoC容器如何初始化和销毁Bean，这是Bean的生命周期。它大致分为Bean定义，Bean初始化，Bean的生存期和Bean的销毁四个部分。

1. Bean定义

Spring 通过扫描包扫描到带有@Component的类后，开始解析，并将定义的信息保存起来，然后就会把Bean定义发布到Spring Ioc容器中，此时IOC容器中只有Bean的定义，没有Bean的实例生成，更没有依赖注入。

1. Bean初始化

默认情况下，Spring在定义后，会完成Bean的实例化和依赖注入，这样从IOC容器中就可以得到一个依赖注入完成的Bean。@ComponentScan中有一个配置项lazyInit，默认是false，表示默认不进行延迟初始化，即默认Spring会完成Bean的实例化和依赖注入。

1. 自定义Bean初始化与销毁

在Spring完成依赖注入后，还提供了一系列的接口和配置来完成Bean初始化的过程。

1. 接口BeanNameAware

该接口中的setBeanName方法用于设置Bean的名称。

1. 接口BeanFactoryAware

该接口中的setBeanFactory方法用于设置BeanFactory。

1. 接口ApplicationContextAware

该接口需要一个实现了ApplicationContext接口的IOC容器，这样该接口的setApplicationContext方法才会被调用。

1. 接口BeanPostProcessor

该接口为Bean后置处理器，将对所有的Bean有效，它有两个方法：预初始化方法postProcessBeforeInitialization，后初始化方法postProcessAfterInitialization。

1. 注解@PostConstruct

该注解用于表示自定义的初始化方法。

1. 接口InitializingBean

该接口中有一个afterPropertiesSet方法，用于设置属性。

1. 注解@PreDestory

该注解用于自定义一个销毁方法。

1. 接口DIsposableBean

该接口的destory方法用于销毁Bean。

1. 使用配置文件

在Spring Boot中，我们首先要在Maven配置文件中加载依赖，如下所示，这样Spring 将创建读取配置文件的上下文：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

1. 使用默认application.properties文件

使用默认的application.properties文件时，它会自动被Spring读取到上下文中，可以直接引用，无需其他操作。一般引用的方式是Spring表达式，通过@Value注解，其中用${..}这样的占位符来读取配置在配置文件中的值。如下所示：

@Value(“${database.driverName}”)

表示 读取配置文件中的 database.driverName 字段。

也可以使用注解@ConfigurationProperties，该注解可以配置前缀属性，该前缀可以和该类中的属性名称组合起来，组成一个全限定名去配置文件中查找，这样就不需要为每一个属性添加注解@Value了。如下所示：

@ConfigurationProperties(“database”)

public class DataBaseProperties{

private String driverName = null;

}

Spring就会以 “database.driverName”的全限定名去查找配置文件。

1. 使用其他的配置文件

当需要将默认的配置文件拆分时，就需要使用注解@PropertyScan来引用需要的文件了，将它加载到Spring的上下文中。注解中的value属性为多个配置文件的名称，classpath表示在这个类的路径下去查找配置文件， ignoreResouceNotFound表示是否忽略配置文件找不到的情况。如下所示：

@SpringBootApplication  
@ComponentScan(basePackages = {"com.pojo"})  
@PropertySource(value={"classpath:jdbc.properties"}, ignoreResourceNotFound = true)  
public class spb {  
 public static void main(String[] args) {  
 SpringApplication.*run*(spb.class);  
 }  
}

（十一）条件装配Bean

1. 概述

有时候因为一些客观因素导致一些Bean无法初始化，再这样的情况下，如果IOC容器继续装配Bean就会抛出异常。这里我们可以通过判断上下文中Bean所需要的参数是否齐全来决定是否装配Bean。

1. 注解@Conditional

该注解需要配合另一个接口Condition一起使用，在需要判断的类上使用注解@Conditional()，括号内的参数为实现了接口Condition的实现类。如下所示，DatabaseConditional类实现了接口Condition。

1. 接口Condition

接口Condition要求实现mathes方法，该方法用于判断上下文中所需参数是否齐全，是否可以装配Bean。如下所示，context参数为上下文，从上下文中取出环境配置，查看环境配置中是否包含所需要的参数，返回true表示能够装配Bean：

public class DatabaseConditional implements Condition{

@Override

public Boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata){

Environment env = context.getEnvironment();

return env.containsProperty(“database.driverName”)

&& env.containsProperty(“database.url”)

&& env.containsProperty(“database.username”)

&& env.containsProperty(“database.password”)

}

}

（十二）Bean的作用域

1. 概述

在一般的容器中，Bean有单例（Singleton）和原型（Prototype）两种作用域；而在Web容器中，则有页面（Page），请求（Request），会话（Session）和应用（Application）四种作用域。

1. 注解@Scope

该注解用于定义类的作用域，单例或是原型，如下所示：

单例：

@Scope(ConfigurableBeanFactory.*SCOPE\_SINGLETON*)

原型：

@Scope(ConfigurableBeanFactory.*SCOPE\_PROTOTYPE*)

在Spring MVC环境中，还可以使用WebApplicationContext来定义其他的作用域，如下所示：

@Scope(WebApplicationContext.SCOPE\_REQUEST)

（十三）引入XML配置

在Spring Boot中也可以引入XML的配置，使用注解@ImportResource，其中的value属性为需要引入的XML配置文件的名称，如下所示：

@ImportResource(value={“classpath:spring-other.xml”})

（十四）使用Spring EL

Spring提供了表达式语言Spring EL，能够读取配置文件属性，调用方法，进行简单运算。

1. ${…}

这个符号表示占位符，它会读取上下文的属性值装配到属性中。如下所示：

@Value(“${database.driverName}”)

String driver;

1. #{…}

这个符号表示启用Spring表达式，它将具有运算的功能，T(…)表示引入类，类需要写全限定名，引入类后就可以直接调用方法了，如下所示：

@Value(“#{System}.currentTimeMillies())”)

private Long initTime = null;

1. 获取其他Bean的属性

可以直接在Spring EL中获取其他Bean的属性用于赋值，如下所示：

@Value(“#{beanName.str}”)

private String BeanProp = null;