# maven下载

|  |
| --- |
| <https://mvnrepository.com/search?q=spring+context>  1.    2.    3. |

# 组件注册

## @Configuration & @Bean给容器中注册组件

### xml

|  |
| --- |
| beans.xml  <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">   <bean id="person" class="com.atguigu.bean.Person" scope="prototype" >  <property name="age" value="18"></property>  <property name="name" value="zhangsan"></property>  </bean> </beans>  =====================================================  Person  package com.atguigu.bean;  public class Person {   private String name;   private Integer age;   private String nickName;   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public Integer getAge() {  return age;  }   public void setAge(Integer age) {  this.age = age;  }   public String getNickName() {  return nickName;  }   public void setNickName(String nickName) {  this.nickName = nickName;  }   public Person(String name, Integer age) {  super();  this.name = name;  this.age = age;  }  public Person() {  super();  }   @Override  public String toString() {  return "Person [name=" + name + ", age=" + age + ", nickName=" + nickName + "]";  } }  =====================================================  MainTest  package com.atguigu;  import com.atguigu.bean.Person; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class MainTest {  public static void main(String[] args) {  ApplicationContext applicationContext = new ClassPathXmlApplicationContext("beans.xml");  Person bean = (Person) applicationContext.getBean("person");  System.***out***.println(bean);  } } |

### 注解方式

|  |
| --- |
| MainConfig  package com.atguigu.config;  import com.atguigu.bean.Person; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;  //配置类==配置文件，告诉Spring这是一个配置类 @Configuration public class MainConfig {  //给容器中注册一个Bean;类型为返回值的类型，id默认是用方法名作为id  @Bean("person01")  public Person person01() {  return new Person("lisi", 20);  } }  =====================================================  MainTest  package com.atguigu;  import com.atguigu.bean.Person; import com.atguigu.config.MainConfig; import org.springframework.context.ApplicationContext; import org.springframework.context.annotation.AnnotationConfigApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext;  public class MainTest {  public static void main(String[] args) {  // ApplicationContext applicationContext = new ClassPathXmlApplicationContext("beans.xml");  // Person bean = (Person) applicationContext.getBean("person");  // System.out.println(bean);   ApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig.class);  Person bean = applicationContext.getBean(Person.class);  System.***out***.println(bean);   // 获取注册组件的类名  String[] namesForType = applicationContext.getBeanNamesForType(Person.class);  for (String name : namesForType) {  System.***out***.println(name);  }  } } |

# 组件注册

## @ComponentScan-自动扫描组件&指定扫描规则

### 指定扫描规则

|  |
| --- |
| MainConfig  package com.atguigu.config;  import com.atguigu.bean.Person; import com.atguigu.service.BookService; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.ComponentScan; import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.FilterType; import org.springframework.context.annotation.ComponentScan.Filter; import org.springframework.context.annotation.ComponentScans; import org.springframework.stereotype.Controller;  //配置类==配置文件，告诉Spring这是一个配置类 @Configuration  /\*  相当于context:component-scan base-package="com.atguigu" use-default-filters="false"></context:component-scan>  @ComponentScan value:指定要扫描的包  excludeFilters = Filter[] ：指定扫描的时候按照什么规则排除那些组件  includeFilters = Filter[] ：指定扫描的时候只需要包含哪些组件  FilterType.ANNOTATION：按照注解  FilterType.ASSIGNABLE\_TYPE：按照给定的类型；  FilterType.ASPECTJ：使用ASPECTJ表达式  FilterType.REGEX：使用正则指定  FilterType.CUSTOM：使用自定义规则  useDefaultFilters = false 时includeFilters才生效  \*/ @ComponentScans(value = {  @ComponentScan(value = "com.atguigu", excludeFilters = {  @Filter(type= FilterType.***ANNOTATION***,classes={Controller.class}),  @Filter(type=FilterType.***ASSIGNABLE\_TYPE***,classes={BookService.class})  }) }) public class MainConfig {  //给容器中注册一个Bean;类型为返回值的类型，id默认是用方法名作为id  @Bean("person01")  public Person person01() {  return new Person("lisi", 20);  } }  =====================================================  IOCTest  package com.atguigu.test;  import com.atguigu.config.MainConfig; import org.junit.Test; import org.springframework.context.annotation.AnnotationConfigApplicationContext;  public class IOCTest {  @Test  public void test01() {  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig.class);  // 获取注册组件的类名  String[] definitionNames = applicationContext.getBeanDefinitionNames();  for(String name : definitionNames) {  System.***out***.println(name);  }  } }  添加@ComponentScans前    添加@ComponentScans后  @Filter(type= FilterType.***ANNOTATION***,classes={Controller.class})    @Filter(type=FilterType.***ASSIGNABLE\_TYPE***,classes={BookService.class}) |

### 自定义扫描规则

|  |
| --- |
| MyTypeFilter  package com.atguigu.config;  import org.springframework.core.io.Resource; import org.springframework.core.type.AnnotationMetadata; import org.springframework.core.type.ClassMetadata; import org.springframework.core.type.classreading.MetadataReader; import org.springframework.core.type.classreading.MetadataReaderFactory; import org.springframework.core.type.filter.TypeFilter;  import java.io.IOException;  public class MyTypeFilter implements TypeFilter {   /\*\*  \* metadataReader：读取到的当前正在扫描的类的信息  \* metadataReaderFactory:可以获取到其他任何类信息的  \*/   public boolean match(MetadataReader metadataReader, MetadataReaderFactory metadataReaderFactory) throws IOException {  //获取当前类注解的信息  AnnotationMetadata annotationMetadata = metadataReader.getAnnotationMetadata();  //获取当前正在扫描的类的类信息  ClassMetadata classMetadata = metadataReader.getClassMetadata();  //获取当前类资源（类的路径）  Resource resource = metadataReader.getResource();   String className = classMetadata.getClassName();  System.***out***.println("--->"+className);   if(className.contains("er")){  return true;  }  return false;  } }  =====================================================  MainConfig  package com.atguigu.config;  import com.atguigu.bean.Person; import com.atguigu.service.BookService; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.ComponentScan; import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.FilterType; import org.springframework.context.annotation.ComponentScan.Filter; import org.springframework.context.annotation.ComponentScans; import org.springframework.stereotype.Controller;  //配置类==配置文件，告诉Spring这是一个配置类 @Configuration @ComponentScans(value = {  @ComponentScan(value = "com.atguigu", includeFilters = {  // @Filter(type= FilterType.ANNOTATION,classes={Controller.class}),  // @Filter(type=FilterType.ASSIGNABLE\_TYPE,classes={BookService.class})  @Filter(type=FilterType.***CUSTOM***,classes={MyTypeFilter.class})  },useDefaultFilters = false) }) public class MainConfig {  //给容器中注册一个Bean;类型为返回值的类型，id默认是用方法名作为id  @Bean("person01")  public Person person01() {  return new Person("lisi", 20);  } }  =====================================================  IOCTest  package com.atguigu.test;  import com.atguigu.config.MainConfig; import org.junit.Test; import org.springframework.context.annotation.AnnotationConfigApplicationContext;  public class IOCTest {   */\*  @SuppressWarnings注解主要用在取消一些编译器产生的警告对代码左侧行列的遮挡，有时候这会挡住我们断点调试时打的断点。  https://www.cnblogs.com/perfei456/p/8962167.html  \*/* @SuppressWarnings("resource")  @Test  public void test01(){  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig.class);  // 获取注册组件的类名  String[] definitionNames = applicationContext.getBeanDefinitionNames();  for (String name : definitionNames) {  System.***out***.println(name);  }  } } |

## 自定义TypeFilter指定过滤规则

|  |
| --- |
|  |

## @Scope-设置组件作用域

|  |
| --- |
| MainConfig2  package com.atguigu.config;   import com.atguigu.bean.ColorFactoryBean; import com.atguigu.bean.Person; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Lazy; import org.springframework.context.annotation.Scope;  public class MainConfig2 {   //默认是单实例的  /\*\*  \* ConfigurableBeanFactory#SCOPE\_PROTOTYPE  \* @see ConfigurableBeanFactory#SCOPE\_SINGLETON  \* @see org.springframework.web.context.WebApplicationContext#SCOPE\_REQUEST request  \* @see org.springframework.web.context.WebApplicationContext#SCOPE\_SESSION sesssion  \* @return\  \* @Scope:调整作用域  \* prototype：多实例的：ioc容器启动并不会去调用方法创建对象放在容器中。  \* 每次获取的时候才会调用方法创建对象；  \* singleton：单实例的（默认值）：ioc容器启动会调用方法创建对象放到ioc容器中。  \* 以后每次获取就是直接从容器（map.get()）中拿，  \* request：同一次请求创建一个实例  \* session：同一个session创建一个实例  \*  \* 懒加载：  \* 单实例bean：默认在容器启动的时候创建对象；  \* 懒加载：容器启动不创建对象。第一次使用(获取)Bean创建对象，并初始化；  \*  \*/  @Scope("singleton")  // @Lazy  @Bean("person")  public Person person(){  System.***out***.println("给容器中添加Person....");  return new Person("张三", 25);  } }  =====================================================  IOCTest  @Test  public void test02(){  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class); // String[] definitionNames = applicationContext.getBeanDefinitionNames(); // for (String name : definitionNames) { // System.out.println(name); // }   System.***out***.println("ioc容器创建完成....");  Object bean = applicationContext.getBean("person");  Object bean2 = applicationContext.getBean("person");  System.***out***.println(bean == bean2);  }  @Scope("singleton")    @Scope("prototype") |

## @Lazy-bean懒加载

|  |
| --- |
| 懒加载：用到时才去实例对象， 懒加载专门对单实例  MainConfig2  package com.atguigu.config;  import com.atguigu.bean.ColorFactoryBean; import com.atguigu.bean.Person; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Lazy; import org.springframework.context.annotation.Scope;  public class MainConfig2 {   // @Scope("prototype")  @Lazy  @Bean("person")  public Person person(){  System.***out***.println("给容器中添加Person....");  return new Person("张三", 25);  } }  =====================================================  IOCTest  @Test  public void test02(){  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class); // String[] definitionNames = applicationContext.getBeanDefinitionNames(); // for (String name : definitionNames) { // System.out.println(name); // }   // System.out.println("ioc容器创建完成....");  // Object bean = applicationContext.getBean("person");  // Object bean2 = applicationContext.getBean("person");  // System.out.println(bean == bean2);  }  有注释    =====================================================  无注释  @Test  public void test02(){  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class); // String[] definitionNames = applicationContext.getBeanDefinitionNames(); // for (String name : definitionNames) { // System.out.println(name); // }   System.***out***.println("ioc容器创建完成....");  Object bean = applicationContext.getBean("person");  Object bean2 = applicationContext.getBean("person");  System.***out***.println(bean == bean2);  } |

## @Conditional-按照条件注册bean

|  |
| --- |
| /\*\*  \* 给容器中注册组件；  \* 1）、包扫描+组件标注注解（@Controller/@Service/@Repository/@Component）[自己写的类]  \* 2）、@Bean[导入的第三方包里面的组件]  \* 3）、@Import[快速给容器中导入一个组件]  \* 1）、@Import(要导入到容器中的组件)；容器中就会自动注册这个组件，id默认是全类名  \* 2）、ImportSelector:返回需要导入的组件的全类名数组；  \* 3）、ImportBeanDefinitionRegistrar:手动注册bean到容器中  \* 4）、使用Spring提供的 FactoryBean（工厂Bean）;  \* 1）、默认获取到的是工厂bean调用getObject创建的对象  \* 2）、要获取工厂Bean本身，我们需要给id前面加一个&  \* &colorFactoryBean  \*/ |

|  |
| --- |
| MainConfig2 package com.atguigu.config;  import com.atguigu.bean.ColorFactoryBean; import com.atguigu.bean.Person; import com.atguigu.condition.LinuxCondition; import com.atguigu.condition.WindowsCondition; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Conditional; import org.springframework.context.annotation.Lazy; import org.springframework.context.annotation.Scope;  //类中组件统一设置。满足当前条件，这个类中配置的所有bean注册才能生效； **// @Conditional({WindowsCondition.class})** public class MainConfig2 {   /\*\*  \* @Conditional({Condition}) ： 按照一定的条件进行判断，满足条件给容器中注册bean  \* 如果系统是windows，给容器中注册("bill")  \* 如果是linux系统，给容器中注册("linus")  \*/  @Conditional(WindowsCondition.class)  @Bean("bill")  public Person person01(){  return new Person("Bill Gates",62);  }   @Conditional(LinuxCondition.class)  @Bean("linus")  public Person person02(){  return new Person("linus", 48);  } }  ======================================  WindowsCondition  package com.atguigu.condition;  import org.springframework.context.annotation.Condition; import org.springframework.context.annotation.ConditionContext; import org.springframework.core.env.Environment; import org.springframework.core.type.AnnotatedTypeMetadata;  //判断是否windows系统 public class WindowsCondition implements Condition {   public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {  Environment environment = context.getEnvironment();  String property = environment.getProperty("os.name");  if(property.contains("Windows")){  return true;  }  return false;  }  }  ======================================  LinuxCondition  package com.atguigu.condition;  import org.springframework.beans.factory.config.ConfigurableListableBeanFactory; import org.springframework.beans.factory.support.BeanDefinitionRegistry; import org.springframework.context.annotation.Condition; import org.springframework.context.annotation.ConditionContext; import org.springframework.core.env.Environment; import org.springframework.core.type.AnnotatedTypeMetadata;  //判断是否linux系统 public class LinuxCondition implements Condition {   /\*\*  \* ConditionContext：判断条件能使用的上下文（环境）  \* AnnotatedTypeMetadata：注释信息  \*/  public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {  // TODO是否linux系统  //1、能获取到ioc使用的beanfactory  ConfigurableListableBeanFactory beanFactory = context.getBeanFactory();  //2、获取类加载器  ClassLoader classLoader = context.getClassLoader();  //3、获取当前环境信息  Environment environment = context.getEnvironment();  //4、获取到bean定义的注册类  BeanDefinitionRegistry registry = context.getRegistry();   //可以判断容器中的bean注册情况，也可以给容器中注册bean  boolean definition = registry.containsBeanDefinition("person");   String property = environment.getProperty("os.name");   if(property.contains("linux")){  return true;  }  return false;  } }  ======================================  IOCTest  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);  @Test public void test03(){  ConfigurableEnvironment environment = applicationContext.getEnvironment();  //动态获取环境变量的值；Windows 10  String property = environment.getProperty("os.name");  System.***out***.println("==>" + property);   String[] namesForType = applicationContext.getBeanNamesForType(Person.class);  for (String name : namesForType) {  System.***out***.println(name);  }   Map<String, Person> persons = applicationContext.getBeansOfType(Person.class);  System.***out***.println(persons); } |

## @Import-给容器中快速导入一个组件

|  |
| --- |
| Color  package com.atguigu.bean;  public class Color { }  ================================  Red  package com.atguigu.bean;  public class Red { }  ================================  MainConfig2  package com.atguigu.config;  import com.atguigu.bean.Color; import com.atguigu.bean.ColorFactoryBean; import com.atguigu.bean.Person; import com.atguigu.condition.LinuxCondition; import com.atguigu.condition.MyImportSelector; import com.atguigu.condition.WindowsCondition; import org.springframework.context.annotation.\*;    //@Import导入组件，id默认是组件的全类名，可以导入多个 @Import({Color.class, Red.class}) public class MainConfig2 {  }  ================================  IOCTest  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);  @Test public void testImport(){  printBeans(applicationContext); }  private void printBeans(AnnotationConfigApplicationContext applicationContext){  String[] definitionNames = applicationContext.getBeanDefinitionNames();  for (String name : definitionNames) {  System.***out***.println(name);  } } |

## @Import-使用ImportSelector

|  |
| --- |
| Blue  package com.atguigu.bean;  public class Blue { }  ====================================  Red  package com.atguigu.bean;  public class Red { }  ====================================  MyImportSelector  package com.atguigu.condition;  import org.springframework.context.annotation.ImportSelector; import org.springframework.core.type.AnnotationMetadata;  //自定义逻辑返回需要导入的组件 public class MyImportSelector implements ImportSelector {   //返回值，就是到导入到容器中的组件全类名  //AnnotationMetadata:当前标注@Import注解的类的所有注解信息  public String[] selectImports(AnnotationMetadata importingClassMetadata) {  //importingClassMetadata  //方法不要返回null值  return new String[]{"com.atguigu.bean.Blue","com.atguigu.bean.Yellow"};  } }  ====================================  package com.atguigu.config;  import com.atguigu.bean.Color; import com.atguigu.bean.ColorFactoryBean; import com.atguigu.bean.Person; import com.atguigu.bean.Red; import com.atguigu.condition.LinuxCondition; import com.atguigu.condition.MyImportSelector; import com.atguigu.condition.WindowsCondition; import org.springframework.context.annotation.\*;  @Import({Color.class, Red.class,MyImportSelector.class}) public class MainConfig2 {   }  ====================================  IOCTest  package com.atguigu.test;  import com.atguigu.bean.Blue; import com.atguigu.bean.Person; import com.atguigu.config.MainConfig; import com.atguigu.config.MainConfig2; import org.junit.Test; import org.springframework.context.annotation.AnnotationConfigApplicationContext; import org.springframework.core.env.ConfigurableEnvironment;  import java.util.Map;  public class IOCTest {  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);   @Test  public void testImport(){  printBeans(applicationContext);  Blue bean = applicationContext.getBean(Blue.class);  System.***out***.println(bean);   // //工厂Bean获取的是调用getObject创建的对象  // Object bean2 = applicationContext.getBean("colorFactoryBean");  // Object bean3 = applicationContext.getBean("colorFactoryBean");  // System.out.println("bean的类型："+bean2.getClass());  // System.out.println(bean2 == bean3);  //  // Object bean4 = applicationContext.getBean("&colorFactoryBean");  // System.out.println(bean4.getClass());  }   private void printBeans(AnnotationConfigApplicationContext applicationContext){  String[] definitionNames = applicationContext.getBeanDefinitionNames();  for (String name : definitionNames) {  System.***out***.println(name);  }  } } |

## @Import-使用ImportBeanDefinitionRegistrar

|  |
| --- |
| RainBow  package com.atguigu.bean;  public class RainBow { }  ====================================  MyImportBeanDefinitionRegistrar  package com.atguigu.condition;  import org.springframework.beans.factory.support.BeanDefinitionRegistry; import org.springframework.beans.factory.support.RootBeanDefinition; import org.springframework.context.annotation.ImportBeanDefinitionRegistrar; import org.springframework.core.type.AnnotationMetadata;  import com.atguigu.bean.RainBow;  public class MyImportBeanDefinitionRegistrar implements ImportBeanDefinitionRegistrar {   /\*\*  \* AnnotationMetadata：当前类的注解信息  \* BeanDefinitionRegistry:BeanDefinition注册类；  \* 把所有需要添加到容器中的bean；调用  \* BeanDefinitionRegistry.registerBeanDefinition手工注册进来  \*/  public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry registry) {   boolean definition = registry.containsBeanDefinition("com.atguigu.bean.Red");  boolean definition2 = registry.containsBeanDefinition("com.atguigu.bean.Blue");  if(definition && definition2){  //指定Bean定义信息；（Bean的类型，Bean。。。）  RootBeanDefinition beanDefinition = new RootBeanDefinition(RainBow.class);  //注册一个Bean，指定bean名  registry.registerBeanDefinition("rainBow", beanDefinition);  }  }  }  ====================================  package com.atguigu.config;  import com.atguigu.bean.Color; import com.atguigu.bean.ColorFactoryBean; import com.atguigu.bean.Person; import com.atguigu.bean.Red; import com.atguigu.condition.LinuxCondition; import com.atguigu.condition.MyImportBeanDefinitionRegistrar; import com.atguigu.condition.MyImportSelector; import com.atguigu.condition.WindowsCondition; import org.springframework.context.annotation.\*;   //@Import导入组件，id默认是组件的全类名 @Import({Color.class, Red.class,MyImportSelector.class, MyImportBeanDefinitionRegistrar.class}) public class MainConfig2 {  }  ====================================  IOCTest  package com.atguigu.test;  import com.atguigu.bean.Blue; import com.atguigu.bean.Person; import com.atguigu.config.MainConfig; import com.atguigu.config.MainConfig2; import org.junit.Test; import org.springframework.context.annotation.AnnotationConfigApplicationContext; import org.springframework.core.env.ConfigurableEnvironment;  import java.util.Map;  public class IOCTest {  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);   @Test  public void testImport(){  printBeans(applicationContext);  Blue bean = applicationContext.getBean(Blue.class);  System.***out***.println(bean);   // //工厂Bean获取的是调用getObject创建的对象  // Object bean2 = applicationContext.getBean("colorFactoryBean");  // Object bean3 = applicationContext.getBean("colorFactoryBean");  // System.out.println("bean的类型："+bean2.getClass());  // System.out.println(bean2 == bean3);  //  // Object bean4 = applicationContext.getBean("&colorFactoryBean");  // System.out.println(bean4.getClass());  }   private void printBeans(AnnotationConfigApplicationContext applicationContext){  String[] definitionNames = applicationContext.getBeanDefinitionNames();  for (String name : definitionNames) {  System.***out***.println(name);  }  } } |

## 使用FactoryBean注册组件

|  |
| --- |
| Color  package com.atguigu.bean;  public class Color { }  =================================  ColorFactoryBean  package com.atguigu.bean;  import org.springframework.beans.factory.FactoryBean;  //创建一个Spring定义的FactoryBean public class ColorFactoryBean implements FactoryBean<Color> {   //返回一个Color对象，这个对象会添加到容器中  public Color getObject() throws Exception {  System.***out***.println("ColorFactoryBean...getObject...");  return new Color();  }   public Class<?> getObjectType() {  return Color.class;  }   //是单例？  //true：这个bean是单实例，在容器中保存一份  //false：多实例，每次获取都会创建一个新的bean；  public boolean isSingleton() {  return false;  } }  =================================  MainConfig2  package com.atguigu.config;   import com.atguigu.bean.Color; import com.atguigu.bean.ColorFactoryBean; import com.atguigu.bean.Person; import com.atguigu.bean.Red; import com.atguigu.condition.LinuxCondition; import com.atguigu.condition.MyImportBeanDefinitionRegistrar; import com.atguigu.condition.MyImportSelector; import com.atguigu.condition.WindowsCondition; import org.springframework.context.annotation.\*;  //@Import导入组件，id默认是组件的全类名 @Import({Color.class, Red.class,MyImportSelector.class, MyImportBeanDefinitionRegistrar.class}) public class MainConfig2 {   /\*\*  \* 给容器中注册组件；  \* 1）、包扫描+组件标注注解（@Controller/@Service/@Repository/@Component）[自己写的类]  \* 2）、@Bean[导入的第三方包里面的组件]  \* 3）、@Import[快速给容器中导入一个组件]  \* 1）、@Import(要导入到容器中的组件)；容器中就会自动注册这个组件，id默认是全类名  \* 2）、ImportSelector:返回需要导入的组件的全类名数组；  \* 3）、ImportBeanDefinitionRegistrar:手动注册bean到容器中  \* 4）、使用Spring提供的 FactoryBean（工厂Bean）;  \* 1）、默认获取到的是工厂bean调用getObject创建的对象  \* 2）、要获取工厂Bean本身，我们需要给id前面加一个&  \* &colorFactoryBean  \*/  @Bean  public ColorFactoryBean colorFactoryBean(){  return new ColorFactoryBean();  } }  =====================================  IOCTest  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfig2.class);  @Test public void testImport(){  printBeans(applicationContext);  Blue bean = applicationContext.getBean(Blue.class);  System.***out***.println(bean);   //工厂Bean获取的是调用getObject创建的对象  Object bean2 = applicationContext.getBean("colorFactoryBean");  Object bean3 = applicationContext.getBean("colorFactoryBean");  System.***out***.println("bean的类型："+bean2.getClass());  System.***out***.println(bean2 == bean3);   Object bean4 = applicationContext.getBean("&colorFactoryBean");  System.***out***.println(bean4.getClass()); }  private void printBeans(AnnotationConfigApplicationContext applicationContext){  String[] definitionNames = applicationContext.getBeanDefinitionNames();  for (String name : definitionNames) {  System.***out***.println(name);  } } |

# 生命周期

## @Bean指定初始化和销毁方法

|  |
| --- |
| Car  package com.atguigu.bean;  import org.springframework.stereotype.Component;  @Component public class Car {   public Car(){  System.***out***.println("car constructor...");  }   public void init(){  System.***out***.println("car ... init...");  }   public void detory(){  System.***out***.println("car ... detory...");  } }  ==================================================  MainConfigOfLifeCycle  package com.atguigu.config;  import org.springframework.context.ApplicationListener; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.ComponentScan; import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.Scope;  import com.atguigu.bean.Car;  /\*\*  \* bean的生命周期：  \* bean创建---初始化----销毁的过程  \* 容器管理bean的生命周期；  \* 我们可以自定义初始化和销毁方法；容器在bean进行到当前生命周期的时候来调用我们自定义的初始化和销毁方法  \*  \* 构造（对象创建）  \* 单实例：在容器启动的时候创建对象  \* 多实例：在每次获取的时候创建对象  \*  \* BeanPostProcessor.postProcessBeforeInitialization  \* 初始化：  \* 对象创建完成，并赋值好，调用初始化方法。。。  \* BeanPostProcessor.postProcessAfterInitialization  \* 销毁：  \* 单实例：容器关闭的时候  \* 多实例：容器不会管理这个bean；容器不会调用销毁方法；  \*  \*  \* 遍历得到容器中所有的BeanPostProcessor；挨个执行beforeInitialization，  \* 一但返回null，跳出for循环，不会执行后面的BeanPostProcessor.postProcessorsBeforeInitialization  \*  \* BeanPostProcessor原理  \* populateBean(beanName, mbd, instanceWrapper);给bean进行属性赋值  \* initializeBean  \* {  \* applyBeanPostProcessorsBeforeInitialization(wrappedBean, beanName);  \* invokeInitMethods(beanName, wrappedBean, mbd);执行自定义初始化  \* applyBeanPostProcessorsAfterInitialization(wrappedBean, beanName);  \*}  \*  \*  \*  \* 1）、指定初始化和销毁方法；  \* 通过@Bean指定init-method和destroy-method；  \* 2）、通过让Bean实现InitializingBean（定义初始化逻辑），  \* DisposableBean（定义销毁逻辑）;  \* 3）、可以使用JSR250；  \* @PostConstruct：在bean创建完成并且属性赋值完成；来执行初始化方法  \* @PreDestroy：在容器销毁bean之前通知我们进行清理工作  \* 4）、BeanPostProcessor【interface】：bean的后置处理器；  \* 在bean初始化前后进行一些处理工作；  \* postProcessBeforeInitialization:在初始化之前工作  \* postProcessAfterInitialization:在初始化之后工作  \*  \* Spring底层对 BeanPostProcessor 的使用；  \* bean赋值，注入其他组件，@Autowired，生命周期注解功能，@Async,xxx BeanPostProcessor;  \*  \* @author lfy  \*  \*/ @ComponentScan("com.atguigu.bean") @Configuration public class MainConfigOfLifeCycle {   //@Scope("prototype")  @Bean(initMethod="init",destroyMethod="detory")  public Car car(){  return new Car();  }  }  ==================================  IOCTest\_LifeCycle  package com.atguigu.test;  import org.junit.Test; import org.springframework.context.annotation.AnnotationConfigApplicationContext;  import com.atguigu.config.MainConfigOfLifeCycle;  public class IOCTest\_LifeCycle {   @Test  public void test01(){  //1、创建ioc容器  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfLifeCycle.class);  System.***out***.println("容器创建完成...");   //applicationContext.getBean("car");  //关闭容器  // applicationContext.close();  }  }    ==================================== |

## InitializingBean和DisposableBean

|  |
| --- |
| Cat  package com.atguigu.bean;  import org.springframework.beans.factory.DisposableBean; import org.springframework.beans.factory.InitializingBean; import org.springframework.stereotype.Component;  @Component public class Cat implements InitializingBean,DisposableBean {   public Cat(){  System.***out***.println("cat constructor...");  }   public void destroy() throws Exception {  System.***out***.println("cat...destroy...");  }   public void afterPropertiesSet() throws Exception {  System.***out***.println("cat...afterPropertiesSet...");  } }  ====================================  MainConfigOfLifeCycle  @ComponentScan("com.atguigu.bean") @Configuration public class MainConfigOfLifeCycle {  //@Scope("prototype")  @Bean(initMethod="init",destroyMethod="detory")  public Car car(){  return new Car();  } }  ===================================  IOCTest\_LifeCycle  package com.atguigu.test;  import org.junit.Test; import org.springframework.context.annotation.AnnotationConfigApplicationContext;  import com.atguigu.config.MainConfigOfLifeCycle;  public class IOCTest\_LifeCycle {   @Test  public void test01(){  //1、创建ioc容器  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfLifeCycle.class);  System.***out***.println("容器创建完成...");   //applicationContext.getBean("car");  //关闭容器  applicationContext.close();  }  } |

## @PostConstruct&@PreDestroy

|  |
| --- |
| Dog  package com.atguigu.bean;  import javax.annotation.PostConstruct; import javax.annotation.PreDestroy;  import org.springframework.beans.BeansException; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.ApplicationContext; import org.springframework.context.ApplicationContextAware; import org.springframework.stereotype.Component;  @Component public class Dog implements ApplicationContextAware {   //@Autowired  private ApplicationContext applicationContext;   public Dog(){  System.***out***.println("dog constructor...");  }   //对象创建并赋值之后调用  @PostConstruct  public void init(){  System.***out***.println("Dog....@PostConstruct...");  }   //容器移除对象之前  @PreDestroy  public void detory(){  System.***out***.println("Dog....@PreDestroy...");  }   public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {  this.applicationContext = applicationContext;  } } |

## BeanPostProcessor-后置处理器

|  |
| --- |
| MyBeanPostProcessor  package com.atguigu.bean;  import org.springframework.beans.BeansException; import org.springframework.beans.factory.config.BeanPostProcessor; import org.springframework.stereotype.Component;  /\*\*  \* 后置处理器：初始化前后进行处理工作  \* 将后置处理器加入到容器中  \* @author lfy  \*/ @Component public class MyBeanPostProcessor implements BeanPostProcessor {   public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {  System.***out***.println("初始化前"+beanName+"=>"+bean);  return bean;  }   public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {  System.***out***.println("初始化后..."+beanName+"=>"+bean);  return bean;  } } |

## BeanPostProcessor原理

|  |
| --- |
|  |

## BeanPostProcessor在Spring底层的使用

|  |
| --- |
|  |

# 属性赋值

## @Value赋值、@PropertySource加载外部配置文件

|  |
| --- |
| person.properties  person.nickName=李四  ================================  Person  package com.atguigu.bean;  import org.springframework.beans.factory.annotation.Value;  public class Person {   //使用@Value赋值；  //1、基本数值  //2、可以写SpEL； #{}  //3、可以写${}；取出配置文件【properties】中的值（在运行环境变量里面的值）   @Value("张三")  private String name;  @Value("#{20-2}")  private Integer age;   @Value("${person.nickName}")  private String nickName;   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public Integer getAge() {  return age;  }   public void setAge(Integer age) {  this.age = age;  }   public String getNickName() {  return nickName;  }   public void setNickName(String nickName) {  this.nickName = nickName;  }   public Person(String name, Integer age) {  super();  this.name = name;  this.age = age;  }  public Person() {  super();  }   @Override  public String toString() {  return "Person [name=" + name + ", age=" + age + ", nickName=" + nickName + "]";  } }  =======================================  MainConfigOfPropertyValues  package com.atguigu.config;  import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.PropertySource;  import com.atguigu.bean.Person;  //使用@PropertySource读取外部配置文件中的k/v保存到运行的环境变量中;加载完外部的配置文件以后使用${}取出配置文件的值 @PropertySource(value={"classpath:/person.properties"}) @Configuration public class MainConfigOfPropertyValues {   @Bean  public Person person(){  return new Person();  } }  IOCTest\_PropertyValue  =======================================  package com.atguigu.test;  import org.junit.Test; import org.springframework.context.annotation.AnnotationConfigApplicationContext; import org.springframework.core.env.ConfigurableEnvironment;  import com.atguigu.bean.Person; import com.atguigu.config.MainConfigOfLifeCycle; import com.atguigu.config.MainConfigOfPropertyValues;  public class IOCTest\_PropertyValue {  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConfigOfPropertyValues.class);  @Test  public void test01(){  printBeans(applicationContext);  System.***out***.println("=============");   Person person = (Person) applicationContext.getBean("person");  System.***out***.println(person);    ConfigurableEnvironment environment = applicationContext.getEnvironment();  String property = environment.getProperty("person.nickName");  System.***out***.println(property);  applicationContext.close();  }   private void printBeans(AnnotationConfigApplicationContext applicationContext){  String[] definitionNames = applicationContext.getBeanDefinitionNames();  for (String name : definitionNames) {  System.***out***.println(name);  }  } } |

# 自动装配

## @Autowired&@Qualifier&@Primary

|  |
| --- |
| Boss  package com.atguigu.bean;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Component;  //默认加在ioc容器中的组件，容器启动会调用无参构造器创建对象，再进行初始化赋值等操作 @Component public class Boss {   private Car car;   //构造器要用的组件，都是从容器中获取  public Boss(Car car){  this.car = car;  System.***out***.println("Boss...有参构造器");  }   public Car getCar() {  return car;  }    //@Autowired  //标注在方法，Spring容器创建当前对象，就会调用方法，完成赋值；  //方法使用的参数，自定义类型的值从ioc容器中获取  public void setCar(Car car) {  this.car = car;  }   @Override  public String toString() {  return "Boss [car=" + car + "]";  }  }  ========================================  Red  package com.atguigu.bean;  import org.springframework.beans.BeansException; import org.springframework.beans.factory.BeanNameAware; import org.springframework.context.ApplicationContext; import org.springframework.context.ApplicationContextAware; import org.springframework.context.EmbeddedValueResolverAware; import org.springframework.stereotype.Component; import org.springframework.util.StringValueResolver;  @Component public class Red implements ApplicationContextAware,BeanNameAware,EmbeddedValueResolverAware {   private ApplicationContext applicationContext;   public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {  // *TODO Auto-generated method stub* System.***out***.println("传入的ioc："+applicationContext);  this.applicationContext = applicationContext;  }   public void setBeanName(String name) {  // *TODO Auto-generated method stub* System.***out***.println("当前bean的名字："+name);  }   public void setEmbeddedValueResolver(StringValueResolver resolver) {  // *TODO Auto-generated method stub* String resolveStringValue = resolver.resolveStringValue("你好 ${os.name} 我是 #{20\*18}");  System.***out***.println("解析的字符串："+resolveStringValue);  }  }  ========================================  MainConifgOfAutowired  package com.atguigu.config;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.beans.factory.annotation.Qualifier; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.ComponentScan; import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.Primary;  import com.atguigu.bean.Car; import com.atguigu.bean.Color; import com.atguigu.dao.BookDao;   /\*\*  \* 自动装配;  \* Spring利用依赖注入（DI），完成对IOC容器中中各个组件的依赖关系赋值；  \*  \* 1）、@Autowired：自动注入：  \* 1）、默认优先按照类型去容器中找对应的组件:applicationContext.getBean(BookDao.class);找到就赋值  \* 2）、如果找到多个相同类型的组件，再将属性的名称作为组件的id去容器中查找  \* applicationContext.getBean("bookDao")  \* 3）、@Qualifier("bookDao")：使用@Qualifier指定需要装配的组件的id，而不是使用属性名  \* 4）、自动装配默认一定要将属性赋值好，没有就会报错；  \* 可以使用@Autowired(required=false);  \* 5）、@Primary：让Spring进行自动装配的时候，默认使用首选的bean；  \* 也可以继续使用@Qualifier指定需要装配的bean的名字  \* BookService{  \* @Autowired  \* BookDao bookDao;  \* }  \*  \* 2）、Spring还支持使用@Resource(JSR250)和@Inject(JSR330)[java规范的注解]  \* @Resource:  \* 可以和@Autowired一样实现自动装配功能；默认是按照组件名称进行装配的；  \* 没有能支持@Primary功能没有支持@Autowired（reqiured=false）;  \* @Inject:  \* 需要导入javax.inject的包，和Autowired的功能一样。没有required=false的功能；  \* @Autowired:Spring定义的； @Resource、@Inject都是java规范  \*  \* AutowiredAnnotationBeanPostProcessor:解析完成自动装配功能；  \*  \* 3）、 @Autowired:构造器，参数，方法，属性；都是从容器中获取参数组件的值  \* 1）、[标注在方法位置]：@Bean+方法参数；参数从容器中获取;默认不写@Autowired效果是一样的；都能自动装配  \* 2）、[标在构造器上]：如果组件只有一个有参构造器，这个有参构造器的@Autowired可以省略，参数位置的组件还是可以自动从容器中获取  \* 3）、放在参数位置：  \*  \* 4）、自定义组件想要使用Spring容器底层的一些组件（ApplicationContext，BeanFactory，xxx）；  \* 自定义组件实现xxxAware；在创建对象的时候，会调用接口规定的方法注入相关组件；Aware；  \* 把Spring底层一些组件注入到自定义的Bean中；  \* xxxAware：功能使用xxxProcessor；  \* ApplicationContextAware==》ApplicationContextAwareProcessor；  \*  \*  \* @author lfy  \*  \*/ @Configuration @ComponentScan({"com.atguigu.service","com.atguigu.dao",  "com.atguigu.controller","com.atguigu.bean"}) public class MainConifgOfAutowired {   @Primary  @Bean("bookDao2")  public BookDao bookDao(){  BookDao bookDao = new BookDao();  bookDao.setLable("2");  return bookDao;  }   /\*\*  \* @Bean标注的方法创建对象的时候，方法参数的值从容器中获取  \* @param car  \* @return  \*/  @Bean  public Color color(Car car){  Color color = new Color();  color.setCar(car);  return color;  }  }  =======================================  BookController  package com.atguigu.controller;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Controller;  import com.atguigu.service.BookService;  @Controller public class BookController {   @Autowired  private BookService bookService; }  =======================================  BookDao  package com.atguigu.dao;  import org.springframework.stereotype.Repository;  //名字默认是类名首字母小写 @Repository public class BookDao {   private String lable = "1";   public String getLable() {  return lable;  }   public void setLable(String lable) {  this.lable = lable;  }   @Override  public String toString() {  return "BookDao [lable=" + lable + "]";  } }  =======================================  BookService  package com.atguigu.service;   import javax.annotation.Resource; import javax.inject.Inject;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.beans.factory.annotation.Qualifier; import org.springframework.stereotype.Service;  import com.atguigu.dao.BookDao;   @Service public class BookService {   //@Qualifier("bookDao")  @Autowired(required=false)  //@Resource(name="bookDao2")  // @Inject  private BookDao bookDao;   public void print(){  System.***out***.println(bookDao);  }   @Override  public String toString() {  return "BookService [bookDao=" + bookDao + "]";  }  }  =======================================  IOCTest\_Autowired  package com.atguigu.test;  import org.junit.Test; import org.springframework.context.annotation.AnnotationConfigApplicationContext;  import com.atguigu.bean.Boss; import com.atguigu.bean.Car; import com.atguigu.bean.Color; import com.atguigu.bean.Red; import com.atguigu.config.MainConifgOfAutowired; import com.atguigu.dao.BookDao; import com.atguigu.service.BookService;  public class IOCTest\_Autowired {   @Test  public void test01(){  AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext(MainConifgOfAutowired.class);   BookService bookService = applicationContext.getBean(BookService.class);  System.***out***.println(bookService);   //BookDao bean = applicationContext.getBean(BookDao.class);  //System.out.println(bean);   Boss boss = applicationContext.getBean(Boss.class);  System.***out***.println(boss);  Car car = applicationContext.getBean(Car.class);  System.***out***.println(car);   Color color = applicationContext.getBean(Color.class);  System.***out***.println(color);  System.***out***.println(applicationContext);  applicationContext.close();  }  } |

## @Resource&@Inject

|  |
| --- |
|  |

## 方法、构造器位置的自动装配

|  |
| --- |
|  |

## Aware注入Spring底层组件&原理

|  |
| --- |
|  |

## @Profile环境搭建

|  |
| --- |
| dbconfig.properties  db.user=root db.password=123456 db.driverClass=com.mysql.jdbc.Driver  ======================  MainConfigOfProfile  package com.atguigu.config;  import javax.sql.DataSource;  import org.springframework.beans.factory.annotation.Value; import org.springframework.context.EmbeddedValueResolverAware; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.Profile; import org.springframework.context.annotation.PropertySource; import org.springframework.util.StringValueResolver;  import com.atguigu.bean.Yellow; import com.mchange.v2.c3p0.ComboPooledDataSource;  /\*\*  \* Profile：  \* Spring为我们提供的可以根据当前环境，动态的激活和切换一系列组件的功能；  \*  \* 开发环境、测试环境、生产环境；  \* 数据源：(/A)(/B)(/C)；  \*  \*  \* @Profile：指定组件在哪个环境的情况下才能被注册到容器中，不指定，任何环境下都能注册这个组件  \*  \* 1）、加了环境标识的bean，只有这个环境被激活的时候才能注册到容器中。默认是default环境  \* 2）、写在配置类上，只有是指定的环境的时候，整个配置类里面的所有配置才能开始生效  \* 3）、没有标注环境标识的bean在，任何环境下都是加载的；  \*/  @PropertySource("classpath:/dbconfig.properties") @Configuration public class MainConfigOfProfile implements EmbeddedValueResolverAware{   @Value("${db.user}")  private String user;   private StringValueResolver valueResolver;   private String driverClass;   @Bean  public Yellow yellow(){  return new Yellow();  }   @Profile("test")  @Bean("testDataSource")  public DataSource dataSourceTest(@Value("${db.password}")String pwd) throws Exception{  ComboPooledDataSource dataSource = new ComboPooledDataSource();  dataSource.setUser(user);  dataSource.setPassword(pwd);  dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");  dataSource.setDriverClass(driverClass);  return dataSource;  }    @Profile("dev")  @Bean("devDataSource")  public DataSource dataSourceDev(@Value("${db.password}")String pwd) throws Exception{  ComboPooledDataSource dataSource = new ComboPooledDataSource();  dataSource.setUser(user);  dataSource.setPassword(pwd);  dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/ssm\_crud");  dataSource.setDriverClass(driverClass);  return dataSource;  }   @Profile("prod")  @Bean("prodDataSource")  public DataSource dataSourceProd(@Value("${db.password}")String pwd) throws Exception{  ComboPooledDataSource dataSource = new ComboPooledDataSource();  dataSource.setUser(user);  dataSource.setPassword(pwd);  dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/scw\_0515");   dataSource.setDriverClass(driverClass);  return dataSource;  }   public void setEmbeddedValueResolver(StringValueResolver resolver) {  // *TODO Auto-generated method stub* this.valueResolver = resolver;  driverClass = valueResolver.resolveStringValue("${db.driverClass}");  }  }  ===============================  IOCTest\_Profile  package com.atguigu.test;  import javax.sql.DataSource;  import org.junit.Test; import org.springframework.context.annotation.AnnotationConfigApplicationContext;  import com.atguigu.bean.Boss; import com.atguigu.bean.Car; import com.atguigu.bean.Color; import com.atguigu.bean.Red; import com.atguigu.bean.Yellow; import com.atguigu.config.MainConfigOfProfile; import com.atguigu.config.MainConifgOfAutowired; import com.atguigu.dao.BookDao; import com.atguigu.service.BookService;  public class IOCTest\_Profile {   //1、使用命令行动态参数: 在虚拟机参数位置加载 -Dspring.profiles.active=test  //2、代码的方式激活某种环境；  @Test  public void test01(){  AnnotationConfigApplicationContext applicationContext =  new AnnotationConfigApplicationContext();  //1、创建一个applicationContext  //2、设置需要激活的环境  applicationContext.getEnvironment().setActiveProfiles("dev");  //3、注册主配置类  applicationContext.register(MainConfigOfProfile.class);  //4、启动刷新容器  applicationContext.refresh();    String[] namesForType = applicationContext.getBeanNamesForType(DataSource.class);  for (String string : namesForType) {  System.***out***.println(string);  }   Yellow bean = applicationContext.getBean(Yellow.class);  System.***out***.println(bean);  applicationContext.close();  } } |

## @Profile根据环境注册bean

|  |
| --- |
|  |

# AOP

|  |
| --- |
|  |

# 声明式事务

|  |
| --- |
|  |