1.@Autowried 这个注解是对成员变量，方法以及构造函数经行注释的，它是Byname方式经行依赖注入

如果当Spring上下文中存在不止一个UserDao类型的bean时，就会抛出BeanCreationException异常；如果Spring上下文中不存在UserDao类型的bean，也会抛出BeanCreationException异常

@Autowried 注释在成员变量上spring就会将其以Byname的形式自动化注入到spring容器中，因此便省去了get/set方法和spring配置文件的配置。

view plaincopy

@Autowired

private AccountDao accountDao;

@Override

public Account read(String username, String password) {

Account account = accountDao.read(username);

if (!password.equals(account.getPassword())) {

account = null;

}

return account;

}

@Autowried 注释在成员方法中，spring就会把方法的变量将方法的参数注入进来，构造方法的注释同样。

[java] view plaincopy

@Autowired

public void setUserDao(@Qualifier("userDao") UserDao userDao) {

this.userDao = userDao;

}

2.@Qualifer 标注的对象与上一个相同，不同的是它指定了注入对象的类型，在使用@Autowried时如果没有指定注入参数的类型就会报错，因此@Qualifer是对@Autowried的有益补充。

如果没有指定userDao 的类型的，并且存在另一个名字为userDao的对象，那么是spring就不知道该注入哪一个对象，如果使用@Qualifer指定注入对象的类型的话就不会出现保存的现象。

如果不存在该对象的话

[3.@Resource](mailto:3.@Resource)

Byname 是bean中的id

Bytype 是bean中的类型

是JSR-250 的注解，spring不仅支持自定义的@Autowried注解，还支持JSR-250规范定义的注解。该注解与@Autowried的使用方法一样，但是建议使用@Resource。@Resource与@Autowried的不同就是@Autowried是通过Bytype的方式注入spring的，而@Resource是通过Byname或Bytype方式注入spring的。@Resource有两个重要的属性分别是name和type，spring将属性name解析为bean的名字，而将type属性解析成bean的类型。

如果使用name属性就是使用Byname自动注入策略，如果使用type属性就是使用Bytype自动修护乳策略，如果即不指定name也不指定type，那么spring就通过反射机制使用Byname自动注入策略。如果同时指定了name和type，则从Spring上下文中找到唯一匹配的bean进行装配，找不到则抛出异常

如果指定了name，则从上下文中查找名称（id）匹配的bean进行装配，找不到则抛出异常

如果指定了type，则从上下文中找到类型匹配的唯一bean进行装配，找不到或者找到多个，都会抛出异常

原始类型（UserDao）进行匹配，如果匹配则自动装配；

@PostConstruct 制定初始化时调用的方法

@Prodestroy 制定销毁时调用的方法

注释配置相对于 XML 配置具有很多的优势：

* 它可以充分利用 Java 的反射机制获取类结构信息，这些信息可以有效减少配置的工作。如使用 JPA 注释配置 ORM 映射时，我们就不需要指定 PO 的属性名、类型等信息，如果关系表字段和 PO 属性名、类型都一致，您甚至无需编写任务属性映射信息——因为这些信息都可以通过 Java 反射机制获取。
* 注释和 Java 代码位于一个文件中，而 XML 配置采用独立的配置文件，大多数配置信息在程序开发完成后都不会调整，如果配置信息和 Java 代码放在一起，有助于增强程序的内聚性。而采用独立的 XML 配置文件，程序员在编写一个功能时，往往需要在程序文件和配置文件中不停切换，这种思维上的不连贯会降低开发效率。

因此在很多情况下，注释配置比 XML 配置更受欢迎，注释配置有进一步流行的趋势。Spring 2.5 的一大增强就是引入了很多注释类，现在您已经可以使用注释配置完成大部分 XML 配置的功能。在这篇文章里，我们将向您讲述使用注释进行 Bean 定义和依赖注入的内容。

原来我们是怎么做的

在使用注释配置之前，先来回顾一下传统上是如何配置 Bean 并完成 Bean 之间依赖关系的建立。下面是 3 个类，它们分别是 Office、Car 和 Boss，这 3 个类需要在 Spring 容器中配置为 Bean：

Office 仅有一个属性：

**清单 1. Office.java**

|  |
| --- |
| package com.baobaotao;  public class Office {  private String officeNo =”001”;  //省略 get/setter  @Override  public String toString() {  return "officeNo:" + officeNo;  }  } |

Car 拥有两个属性：

**清单 2. Car.java**

|  |
| --- |
| package com.baobaotao;  public class Car {  private String brand;  private double price;  // 省略 get/setter  @Override  public String toString() {  return "brand:" + brand + "," + "price:" + price;  }  } |

Boss 拥有 Office 和 Car 类型的两个属性：

**清单 3. Boss.java**

|  |
| --- |
| package com.baobaotao;  public class Boss {  private Car car;  private Office office;  // 省略 get/setter  @Override  public String toString() {  return "car:" + car + "\n" + "office:" + office;  }  } |

我们在 Spring 容器中将 Office 和 Car 声明为 Bean，并注入到 Boss Bean 中：下面是使用传统 XML 完成这个工作的配置文件 beans.xml：

**清单 4. beans.xml 将以上三个类配置成 Bean**

|  |
| --- |
| <?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-2.5.xsd">  <bean id="boss" class="com.baobaotao.Boss">  <property name="car" ref="car"/>  <property name="office" ref="office" />  </bean>  <bean id="office" class="com.baobaotao.Office">  <property name="officeNo" value="002"/>  </bean>  <bean id="car" class="com.baobaotao.Car" scope="singleton">  <property name="brand" value=" 红旗 CA72"/>  <property name="price" value="2000"/>  </bean>  </beans> |

当我们运行以下代码时，控制台将正确打出 boss 的信息：

**清单 5. 测试类：AnnoIoCTest.java**

|  |
| --- |
| import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;  public class AnnoIoCTest {  public static void main(String[] args) {  String[] locations = {"beans.xml"};  ApplicationContext ctx =  new ClassPathXmlApplicationContext(locations);  Boss boss = (Boss) ctx.getBean("boss");  System.out.println(boss);  }  } |

这说明 Spring 容器已经正确完成了 Bean 创建和装配的工作。

**使用 @Autowired 注释**

Spring 2.5 引入了 @Autowired 注释，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。来看一下使用@Autowired 进行成员变量自动注入的代码：

**清单 6. 使用 @Autowired 注释的 Boss.java**

|  |
| --- |
| package com.baobaotao;  import org.springframework.beans.factory.annotation.Autowired;  public class Boss {  @Autowired  private Car car;  @Autowired  private Office office;  …  } |

Spring 通过一个 BeanPostProcessor 对 @Autowired 进行解析，所以要让@Autowired 起作用必须事先在 Spring 容器中声明AutowiredAnnotationBeanPostProcessor Bean。

**清单 7. 让 @Autowired 注释工作起来**

|  |
| --- |
| <?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-2.5.xsd">  <!-- 该 BeanPostProcessor 将自动起作用，对标注 @Autowired 的 Bean 进行自动注入 -->  <bean class="org.springframework.beans.factory.annotation.  AutowiredAnnotationBeanPostProcessor"/>  <!-- 移除 boss Bean 的属性注入配置的信息 -->  <bean id="boss" class="com.baobaotao.Boss"/>    <bean id="office" class="com.baobaotao.Office">  <property name="officeNo" value="001"/>  </bean>  <bean id="car" class="com.baobaotao.Car" scope="singleton">  <property name="brand" value=" 红旗 CA72"/>  <property name="price" value="2000"/>  </bean>  </beans> |

这样，当 Spring 容器启动时，AutowiredAnnotationBeanPostProcessor 将扫描 Spring 容器中所有 Bean，当发现 Bean 中拥有@Autowired 注释时就找到和其匹配（默认按类型匹配）的 Bean，并注入到对应的地方中去。

按照上面的配置，Spring 将直接采用 Java 反射机制对 Boss 中的 car 和 office 这两个私有成员变量进行自动注入。所以对成员变量使用@Autowired 后，您大可将它们的 setter 方法（setCar() 和 setOffice()）从 Boss 中删除。

当然，您也可以通过 @Autowired 对方法或构造函数进行标注，来看下面的代码：

**清单 8. 将 @Autowired 注释标注在 Setter 方法上**

|  |
| --- |
| package com.baobaotao;  public class Boss {  private Car car;  private Office office;  @Autowired  public void setCar(Car car) {  this.car = car;  }    @Autowired  public void setOffice(Office office) {  this.office = office;  }  …  } |

这时，@Autowired 将查找被标注的方法的入参类型的 Bean，并调用方法自动注入这些 Bean。而下面的使用方法则对构造函数进行标注：

**清单 9. 将 @Autowired 注释标注在构造函数上**

|  |
| --- |
| package com.baobaotao;  public class Boss {  private Car car;  private Office office;    @Autowired  public Boss(Car car ,Office office){  this.car = car;  this.office = office ;  }    …  } |

由于 Boss() 构造函数有两个入参，分别是 car 和 office，@Autowired 将分别寻找和它们类型匹配的 Bean，将它们作为Boss(Car car ,Office office)的入参来创建 Boss Bean。

当候选 Bean 数目不为 1 时的应对方法

在默认情况下使用 @Autowired 注释进行自动注入时，Spring 容器中匹配的候选 Bean 数目必须有且仅有一个。当找不到一个匹配的 Bean 时，Spring 容器将抛出BeanCreationException 异常，并指出必须至少拥有一个匹配的 Bean。我们可以来做一个实验：

**清单 10. 候选 Bean 数目为 0 时**

|  |
| --- |
| <?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-2.5.xsd ">    <bean class="org.springframework.beans.factory.annotation.  AutowiredAnnotationBeanPostProcessor"/>  <bean id="boss" class="com.baobaotao.Boss"/>  <!-- 将 office Bean 注释掉 -->  <!-- <bean id="office" class="com.baobaotao.Office">  <property name="officeNo" value="001"/>  </bean>-->  <bean id="car" class="com.baobaotao.Car" scope="singleton">  <property name="brand" value=" 红旗 CA72"/>  <property name="price" value="2000"/>  </bean>  </beans> |

由于 office Bean 被注释掉了，所以 Spring 容器中将没有类型为 Office 的 Bean 了，而 Boss 的office 属性标注了 @Autowired，当启动 Spring 容器时，异常就产生了。

当不能确定 Spring 容器中一定拥有某个类的 Bean 时，可以在需要自动注入该类 Bean 的地方可以使用 @Autowired(required = false)，这等于告诉 Spring：在找不到匹配 Bean 时也不报错。来看一下具体的例子：

**清单 11. 使用 @Autowired(required = false)**

|  |
| --- |
| package com.baobaotao;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Required;  public class Boss {  private Car car;  private Office office;  @Autowired  public void setCar(Car car) {  this.car = car;  }  @Autowired(required = false)  public void setOffice(Office office) {  this.office = office;  }  …  } |

当然，一般情况下，使用 @Autowired 的地方都是需要注入 Bean 的，使用了自动注入而又允许不注入的情况一般仅会在开发期或测试期碰到（如为了快速启动 Spring 容器，仅引入一些模块的 Spring 配置文件），所以@Autowired(required = false) 会很少用到。

和找不到一个类型匹配 Bean 相反的一个错误是：如果 Spring 容器中拥有多个候选 Bean，Spring 容器在启动时也会抛出 BeanCreationException 异常。来看下面的例子：

**清单 12. 在 beans.xml 中配置两个 Office 类型的 Bean**

|  |
| --- |
| …  <bean id="office" class="com.baobaotao.Office">  <property name="officeNo" value="001"/>  </bean>  <bean id="office2" class="com.baobaotao.Office">  <property name="officeNo" value="001"/>  </bean>  … |

我们在 Spring 容器中配置了两个类型为 Office 类型的 Bean，当对 Boss 的 office 成员变量进行自动注入时，Spring 容器将无法确定到底要用哪一个 Bean，因此异常发生了。

Spring 允许我们通过 @Qualifier 注释指定注入 Bean 的名称，这样歧义就消除了，可以通过下面的方法解决异常：

**清单 13. 使用 @Qualifier 注释指定注入 Bean 的名称**

|  |
| --- |
| @Autowired  public void setOffice(@Qualifier("office")Office office) {  this.office = office;  } |

@Qualifier("office") 中的 office 是 Bean 的名称，所以 @Autowired 和@Qualifier 结合使用时，自动注入的策略就从 byType 转变成 byName 了。@Autowired 可以对成员变量、方法以及构造函数进行注释，而@Qualifier 的标注对象是成员变量、方法入参、构造函数入参。正是由于注释对象的不同，所以 Spring 不将 @Autowired 和@Qualifier 统一成一个注释类。下面是对成员变量和构造函数入参进行注释的代码：

对成员变量进行注释：

**清单 14. 对成员变量使用 @Qualifier 注释**

|  |
| --- |
| public class Boss {  @Autowired  private Car car;    @Autowired  @Qualifier("office")  private Office office;  …  } |

对构造函数入参进行注释：

**清单 15. 对构造函数变量使用 @Qualifier 注释**

|  |
| --- |
| public class Boss {  private Car car;  private Office office;  @Autowired  public Boss(Car car , @Qualifier("office")Office office){  this.car = car;  this.office = office ;  }  } |

@Qualifier 只能和 @Autowired 结合使用，是对 @Autowired 有益的补充。一般来讲，@Qualifier 对方法签名中入参进行注释会降低代码的可读性，而对成员变量注释则相对好一些。

使用 JSR-250 的注释

Spring 不但支持自己定义的 @Autowired 的注释，还支持几个由 JSR-250 规范定义的注释，它们分别是 @Resource、@PostConstruct 以及 @PreDestroy。

**@Resource**

@Resource 的作用相当于 @Autowired，只不过 @Autowired 按 byType 自动注入，面@Resource 默认按 byName 自动注入罢了。@Resource 有两个属性是比较重要的，分别是 name 和 type，Spring 将@Resource 注释的 name 属性解析为 Bean 的名字，而 type 属性则解析为 Bean 的类型。所以如果使用 name 属性，则使用 byName 的自动注入策略，而使用 type 属性时则使用 byType 自动注入策略。如果既不指定 name 也不指定 type 属性，这时将通过反射机制使用 byName 自动注入策略。

Resource 注释类位于 Spring 发布包的 lib/j2ee/common-annotations.jar 类包中，因此在使用之前必须将其加入到项目的类库中。来看一个使用@Resource的例子：

**清单 16. 使用 @Resource 注释的 Boss.java**

|  |
| --- |
| package com.baobaotao;  import javax.annotation.Resource;  public class Boss {  // 自动注入类型为 Car 的 Bean  @Resource  private Car car;  // 自动注入 bean 名称为 office 的 Bean  @Resource(name = "office")  private Office office;  } |

一般情况下，我们无需使用类似于 @Resource(type=Car.class) 的注释方式，因为 Bean 的类型信息可以通过 Java 反射从代码中获取。

要让 JSR-250 的注释生效，除了在 Bean 类中标注这些注释外，还需要在 Spring 容器中注册一个负责处理这些注释的 BeanPostProcessor：

|  |
| --- |
| <bean  class="org.springframework.context.annotation.CommonAnnotationBeanPostProcessor"/> |

CommonAnnotationBeanPostProcessor 实现了 BeanPostProcessor 接口，它负责扫描使用了 JSR-250 注释的 Bean，并对它们进行相应的操作。

**@PostConstruct 和 @PreDestroy**

Spring 容器中的 Bean 是有生命周期的，Spring 允许在 Bean 在初始化完成后以及 Bean 销毁前执行特定的操作，您既可以通过实现 InitializingBean/DisposableBean 接口来定制初始化之后 / 销毁之前的操作方法，也可以通过 <bean> 元素的 init-method/destroy-method 属性指定初始化之后 / 销毁之前调用的操作方法。关于 Spring 的生命周期，笔者在《精通 Spring 2.x—企业应用开发精解》第 3 章进行了详细的描述，有兴趣的读者可以查阅。

JSR-250 为初始化之后/销毁之前方法的指定定义了两个注释类，分别是 @PostConstruct 和 @PreDestroy，这两个注释只能应用于方法上。标注了 @PostConstruct 注释的方法将在类实例化后调用，而标注了 @PreDestroy 的方法将在类销毁之前调用。

**清单 17. 使用 @PostConstruct 和 @PreDestroy 注释的 Boss.java**

|  |
| --- |
| package com.baobaotao;  import javax.annotation.Resource;  import javax.annotation.PostConstruct;  import javax.annotation.PreDestroy;  public class Boss {  @Resource  private Car car;  @Resource(name = "office")  private Office office;  @PostConstruct  public void postConstruct1(){  System.out.println("postConstruct1");  }  @PreDestroy  public void preDestroy1(){  System.out.println("preDestroy1");  }  …  } |

您只需要在方法前标注 @PostConstruct 或 @PreDestroy，这些方法就会在 Bean 初始化后或销毁之前被 Spring 容器执行了。

我们知道，不管是通过实现 InitializingBean/DisposableBean 接口，还是通过 <bean> 元素的init-method/destroy-method 属性进行配置，都只能为 Bean 指定一个初始化 / 销毁的方法。但是使用 @PostConstruct 和 @PreDestroy 注释却可以指定多个初始化 / 销毁方法，那些被标注 @PostConstruct 或@PreDestroy 注释的方法都会在初始化 / 销毁时被执行。

通过以下的测试代码，您将可以看到 Bean 的初始化 / 销毁方法是如何被执行的：

**清单 18. 测试类代码**

|  |
| --- |
| package com.baobaotao;  import org.springframework.context.support.ClassPathXmlApplicationContext;  public class AnnoIoCTest {  public static void main(String[] args) {  String[] locations = {"beans.xml"};  ClassPathXmlApplicationContext ctx =  new ClassPathXmlApplicationContext(locations);  Boss boss = (Boss) ctx.getBean("boss");  System.out.println(boss);  ctx.destroy();// 关闭 Spring 容器，以触发 Bean 销毁方法的执行  }  } |

这时，您将看到标注了 @PostConstruct 的 postConstruct1() 方法将在 Spring 容器启动时，创建Boss Bean 的时候被触发执行，而标注了 @PreDestroy注释的 preDestroy1() 方法将在 Spring 容器关闭前销毁Boss Bean 的时候被触发执行。

使用 <context:annotation-config/> 简化配置

Spring 2.1 添加了一个新的 context 的 Schema 命名空间，该命名空间对注释驱动、属性文件引入、加载期织入等功能提供了便捷的配置。我们知道注释本身是不会做任何事情的，它仅提供元数据信息。要使元数据信息真正起作用，必须让负责处理这些元数据的处理器工作起来。

而我们前面所介绍的 AutowiredAnnotationBeanPostProcessor 和 CommonAnnotationBeanPostProcessor 就是处理这些注释元数据的处理器。但是直接在 Spring 配置文件中定义这些 Bean 显得比较笨拙。Spring 为我们提供了一种方便的注册这些BeanPostProcessor 的方式，这就是 <context:annotation-config/>。请看下面的配置：

**清单 19. 调整 beans.xml 配置文件**

|  |
| --- |
| <?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-2.5.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-2.5.xsd">    <context:annotation-config/>  <bean id="boss" class="com.baobaotao.Boss"/>  <bean id="office" class="com.baobaotao.Office">  <property name="officeNo" value="001"/>  </bean>  <bean id="car" class="com.baobaotao.Car" scope="singleton">  <property name="brand" value=" 红旗 CA72"/>  <property name="price" value="2000"/>  </bean>  </beans> |

<context:annotationconfig/> 将隐式地向 Spring 容器注册AutowiredAnnotationBeanPostProcessor、CommonAnnotationBeanPostProcessor、PersistenceAnnotationBeanPostProcessor 以及equiredAnnotationBeanPostProcessor 这 4 个 BeanPostProcessor。

在配置文件中使用 context 命名空间之前，必须在 <beans> 元素中声明 context 命名空间。

**使用 @Component**

虽然我们可以通过 @Autowired 或 @Resource 在 Bean 类中使用自动注入功能，但是 Bean 还是在 XML 文件中通过 <bean> 进行定义 —— 也就是说，在 XML 配置文件中定义 Bean，通过@Autowired 或 @Resource 为 Bean 的成员变量、方法入参或构造函数入参提供自动注入的功能。能否也通过注释定义 Bean，从 XML 配置文件中完全移除 Bean 定义的配置呢？答案是肯定的，我们通过 Spring 2.5 提供的@Component 注释就可以达到这个目标了。

为什么 @Repository 只能标注在 DAO 类上呢？这是因为该注解的作用不只是将类识别为 Bean，同时它还能将所标注的类中抛出的数据访问异常封装为 Spring 的数据访问异常类型。 Spring 本身提供了一个丰富的并且是与具体的数据访问技术无关的数据访问异常结构，用于封装不同的持久层框架抛出的异常，使得异常独立于底层的框架。

Spring 2.5 在 @Repository 的基础上增加了功能类似的额外三个注解：@Component、@Service、@Constroller，它们分别用于软件系统的不同层次：

* @Component 是一个泛化的概念，仅仅表示一个组件 (Bean) ，可以作用在任何层次。
* @Service 通常作用在业务层，但是目前该功能与 @Component 相同。
* @Constroller 通常作用在控制层，但是目前该功能与 @Component 相同。

通过在类上使用 @Repository、@Component、@Service 和 @Constroller 注解，Spring 会自动创建相应的 BeanDefinition 对象，并注册到 ApplicationContext 中。这些类就成了 Spring 受管组件。这三个注解除了作用于不同软件层次的类，其使用方式与 @Repository 是完全相同的。

下面，我们完全使用注释定义 Bean 并完成 Bean 之间装配：

**清单 20. 使用 @Component 注释的 Car.java**

|  |
| --- |
| package com.baobaotao;  import org.springframework.stereotype.Component;  @Component  public class Car {  …  } |

仅需要在类定义处，使用 @Component 注释就可以将一个类定义了 Spring 容器中的 Bean。下面的代码将 Office 定义为一个 Bean：

**清单 21. 使用 @Component 注释的 Office.java**

|  |
| --- |
| package com.baobaotao;  import org.springframework.stereotype.Component;  @Component  public class Office {  private String officeNo = "001";  …  } |

这样，我们就可以在 Boss 类中通过 @Autowired 注入前面定义的 Car 和 Office Bean 了。

**清单 22. 使用 @Component 注释的 Boss.java**

|  |
| --- |
| package com.baobaotao;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Required;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.stereotype.Component;  @Component("boss")  public class Boss {  @Autowired  private Car car;  @Autowired  private Office office;  …  } |

@Component 有一个可选的入参，用于指定 Bean 的名称，在 Boss 中，我们就将 Bean 名称定义为“boss”。一般情况下，Bean 都是 singleton 的，需要注入 Bean 的地方仅需要通过 byType 策略就可以自动注入了，所以大可不必指定 Bean 的名称。

在使用 @Component 注释后，Spring 容器必须启用类扫描机制以启用注释驱动 Bean 定义和注释驱动 Bean 自动注入的策略。Spring 2.5 对 context 命名空间进行了扩展，提供了这一功能，请看下面的配置：

**清单 23. 简化版的 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"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-2.5.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-2.5.xsd">  <context:component-scan base-package="com.baobaotao"/>  </beans> |

这里，所有通过 <bean> 元素定义 Bean 的配置内容已经被移除，仅需要添加一行 <context:component-scan/> 配置就解决所有问题了——Spring XML 配置文件得到了极致的简化（当然配置元数据还是需要的，只不过以注释形式存在罢了）。<context:component-scan/> 的 base-package 属性指定了需要扫描的类包，类包及其递归子包中所有的类都会被处理。

<context:component-scan/> 还允许定义过滤器将基包下的某些类纳入或排除。Spring 支持以下 4 种类型的过滤方式，通过下表说明：

**表 1. 扫描过滤方式**

|  |  |
| --- | --- |
| **过滤器类型** | **说明** |
| 注释 | 假如 com.baobaotao.SomeAnnotation 是一个注释类，我们可以将使用该注释的类过滤出来。 |
| 类名指定 | 通过全限定类名进行过滤，如您可以指定将 com.baobaotao.Boss 纳入扫描，而将 com.baobaotao.Car 排除在外。 |
| 正则表达式 | 通过正则表达式定义过滤的类，如下所示： com\.baobaotao\.Default.\* |
| AspectJ 表达式 | 通过 AspectJ 表达式定义过滤的类，如下所示： com. baobaotao..\*Service+ |

下面是一个简单的例子：

|  |
| --- |
| <context:component-scan base-package="com.baobaotao">  <context:include-filter type="regex"  expression="com\.baobaotao\.service\..\*"/>  <context:exclude-filter type="aspectj"  expression="com.baobaotao.util..\*"/>  </context:component-scan> |

值得注意的是 <context:component-scan/> 配置项不但启用了对类包进行扫描以实施注释驱动 Bean 定义的功能，同时还启用了注释驱动自动注入的功能（即还隐式地在内部注册了AutowiredAnnotationBeanPostProcessor 和 CommonAnnotationBeanPostProcessor），因此当使用 <context:component-scan/> 后，就可以将 <context:annotation-config/> 移除了。

默认情况下通过 @Component 定义的 Bean 都是 singleton 的，如果需要使用其它作用范围的 Bean，可以通过@Scope 注释来达到目标，如以下代码所示：

**@scopee**

**清单 24. 通过 @Scope 指定 Bean 的作用范围**

|  |
| --- |
| package com.baobaotao;  import org.springframework.context.annotation.Scope;  …  @Scope("prototype")  @Component("boss")  public class Boss {  …  } |

这样，当从 Spring 容器中获取 boss Bean 时，每次返回的都是新的实例了。

采用具有特殊语义的注释

Spring 2.5 中除了提供 @Component 注释外，还定义了几个拥有特殊语义的注释，它们分别是：@Repository、@Service 和@Controller。在目前的 Spring 版本中，这 3 个注释和 @Component 是等效的，但是从注释类的命名上，很容易看出这 3 个注释分别和持久层、业务层和控制层（Web 层）相对应。虽然目前这 3 个注释和@Component 相比没有什么新意，但 Spring 将在以后的版本中为它们添加特殊的功能。所以，如果 Web 应用程序采用了经典的三层分层结构的话，最好在持久层、业务层和控制层分别采用@Repository、@Service 和 @Controller 对分层中的类进行注释，而用@Component 对那些比较中立的类进行注释。

注释配置和 XML 配置的适用场合

是否有了这些 IOC 注释，我们就可以完全摒除原来 XML 配置的方式呢？答案是否定的。有以下几点原因：

* 注释配置不一定在先天上优于 XML 配置。如果 Bean 的依赖关系是固定的，（如 Service 使用了哪几个 DAO 类），这种配置信息不会在部署时发生调整，那么注释配置优于 XML 配置；反之如果这种依赖关系会在部署时发生调整，XML 配置显然又优于注释配置，因为注释是对 Java 源代码的调整，您需要重新改写源代码并重新编译才可以实施调整。
* 如果 Bean 不是自己编写的类（如 JdbcTemplate、SessionFactoryBean 等），注释配置将无法实施，此时 XML 配置是唯一可用的方式。
* 注释配置往往是类级别的，而 XML 配置则可以表现得更加灵活。比如相比于 @Transaction 事务注释，使用 aop/tx 命名空间的事务配置更加灵活和简单。

所以在实现应用中，我们往往需要同时使用注释配置和 XML 配置，对于类级别且不会发生变动的配置可以优先考虑注释配置；而对于那些第三方类以及容易发生调整的配置则应优先考虑使用 XML 配置。Spring 会在具体实施 Bean 创建和 Bean 注入之前将这两种配置方式的元信息融合在一起