单元 4.3.3-注解方式属性注入

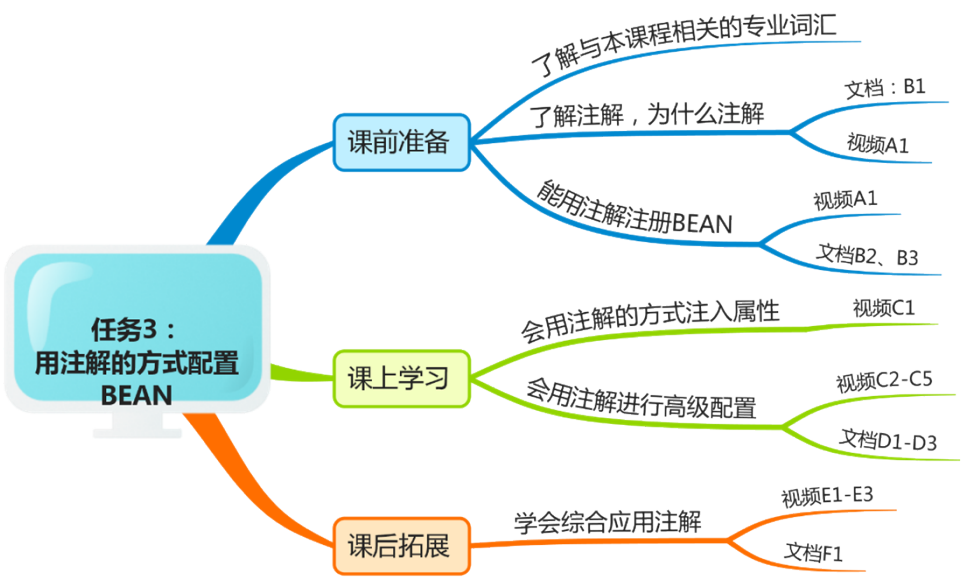
课程导入

同学们， 在上个单元中我们学习了如何用 XML 的方式进行依赖注入的配置，但是在实际生产中我们会更多的用注解+XML 的方式来注入。虽然 XML 看起来比较落后，注解用的范围更广，但是我们还是要用 XML 去深入理解，掌握 XML 之后再去学习注解。

在本单元中，我们将学习 spring 框架的常用注解，操作非常简单，理解非常困难是本阶段学习的特定，请同学们注重理解，不要只局限在应用上。

知识目标	能力目标	素质目标
1. 掌握 Spring Bean 的配置、实例化、作用域、生命周期以及装配方式等内容。 2.了解 Spring Bean 的生命周期， 掌握 Spring Bean 的配置、实例化、作用域以及装配方式。 3.掌握 spring 的常用注解	1.能用注解的方式注册 bean，进行属性注入和配置 bean 与 bean 的关系。 2.能够灵活应用常用注解。	1.培养学生的团队意识和团队协作精神，锻炼学生的沟通交流能力; 2.通过项目教学，让学生真切的体验项目分析、设计、管理及实施的全过程;
学习任务	**重点难点**	**突破方法**
用注解的方式来完成 bean 的注册，配置 bean 的属性和 bean 与 bean 之间的关系	1.常用 spring 注解的用法 2.bean 与 bean 之间的关联关系	采用翻转课堂、项目导入的教学模式，进行分组讨论、演示动画原理。

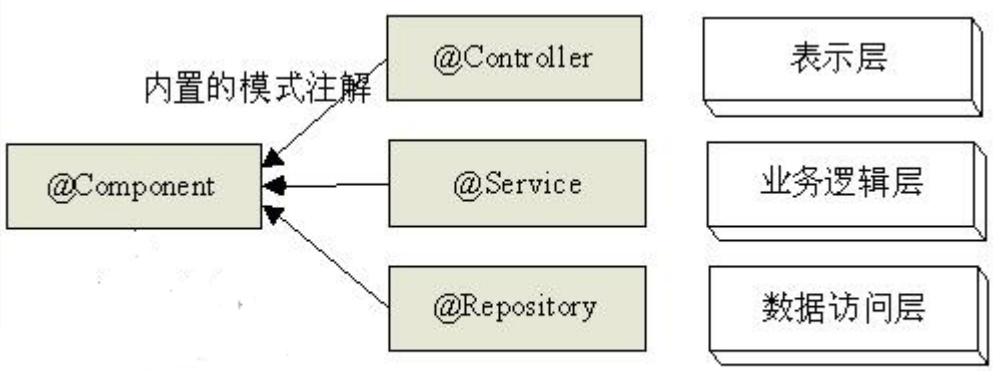
学习导航：



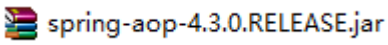
任务 1

spring 注解环境配置

Spring 自带的@Component 注解及扩展@Repository、@Service、@Controller，如图



在使用注解方式配置 bean 时，需要引进一个包：



使用方法：

1、为需要使用注解方式的类添加注解标记

```
@Component("标识符")
POJO 类
```

在类上使用@Component 注解，表示该类定义为 Spring 管理 Bean，使用默认 value（可选）属性表示 Bean 标识符。如果不指定标识符，默认为首字母小写类名。例如类 UserController 的标识符为 userController

2、在 xml 中配置自动扫描策略

```
1 <context:component-scan
2         base-package=""
3         resource-pattern="**/*.class"
4         name-
generator="org.springframework.context.annotation.AnnotationBeanNameGenerator"
5         use-default-filters="true"
6         annotation-config="true">
7                 <context:include-filter type="aspectj" expression=""/>
8                 <context:exclude-filter type="regex" expression=""/>
9 </context:component-scan>
```

- **base-package：**表示扫描注解类的开始位置，即将在指定的包中扫描，其他包中的注解类将不被扫描，默认将扫描所有类路径；
- **resource-pattern：**表示扫描注解类的后缀匹配模式，即“base-package+resource-pattern”将组成匹配模式用于匹配类路径中的组件，默认后缀为“**/*.class”，即指定包下的所有以.class 结尾的类文件；
- **name-generator：**默认情况下的 Bean 标识符生成策略，默认是 AnnotationBeanNameGenerator，其将生成以小写开头的类名（不包括包名）；可以自定义自己的标识符生成策略；
- **use-default-filters：**默认为 true 表示过滤@Component、@ManagedBean、@Named 注解的类，如果改为 false 默认将不过滤这些默认的注解来定义 Bean，即这些注解类不能被过滤到，即不能通过这些注解进行 Bean 定义；
- **annotation-config：**表示是否自动支持注解实现 Bean 依赖注入，默认支持，如果设置为 false，将关闭支持注解的依赖注入，需要通过<context:annotation-config/>开启。
- **<context:include-filter>：**表示过滤到的类将被注册为 Spring 管理 Bean。需要配合 use-default-filters 使用
- **<context:exclude-filter>：**表示过滤到的类将不被注册为 Spring 管理 Bean，它比<context:include-filter>具有更高优先级；
- **type：**表示过滤器类型，目前支持注解类型、类类型、正则表达式、aspectj 表达式过滤器，当然也可以自定义自己的过滤器，实现 org.springframework.core.type.filter.TypeFilter 即可；
- **expression：**表示过滤器表达式。

任务 2

用注解的方式配置 bean 的步骤

```
1 package com.proc.bean;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class TestObject {
7
8 }
```

```
1 package com.proc.bean.Controller;
2
```

```
3 import org.springframework.stereotype.Controller;
4
5 @Controller
6 public class UserController {
7
8 }
```

```
1 package com.proc.bean.repository;
2
3 public interface UserRepository {
4
5     void save();
6 }
```

```
 1 package com.proc.bean.repository;
 2
 3 import org.springframework.stereotype.Repository;
 4
 5 @Repository("userRepository")
 6 public class UserRepositoryImps implements UserRepository{
 7
 8     @Override
 9     public void save() {
10         System.out.println("UserRepository save");
11     }
12 }
```

```
1 package com.proc.bean.service;
2
3 import org.springframework.stereotype.Service;
4
5 @Service
6 public class UserService {
7
8 }
```

1、在 xml 中配置，通过 base-package 指定扫描指定包及其子包下所有类

```
1 <context:component-scan base-package="com.proc.bean"></context:component-scan>
```

测试输出

```
ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");

TestObject testObject=(TestObject) ctx.getBean("testObject");
System.out.println(testObject);

UserController userController=(UserController) ctx.getBean("userController");
System.out.println(userController);

UserService userService=(UserService) ctx.getBean("userService");
System.out.println(userService);

UserRepository userRepository=(UserRepository)ctx.getBean("userRepository");
System.out.println(userRepository);
```

输出结果：

com.proc.bean.TestObject@50d156
com.proc.bean.Controller.UserController@61c7e3
com.proc.bean.service.UserService@11d0846
com.proc.bean.repository.UserRepositoryImps@1e4c80f

2、指定 resource-pattern:资源匹配，只扫描 controller 包下面的所有类

```
<context:component-scan base-package="com.proc.bean" resource-pattern="controller/*.class">
</context:component-scan>
```

这里是能够正确获取到 com.proc.bean.Controller.UserController@61c7e3

3、 排除使用指定注解标签的类

```
<context:component-scan base-package="com.proc.bean">
    <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Repository"/>
</context:component-scan>
```

type：选择类型 annotation：注解标签、assignable：类名方式

这里能够正确获取到

com.proc.bean.TestObject@191f517
com.proc.bean.controller.UserController@5965f2
com.proc.bean.service.UserService@9bd883

4、排除指定标识符的类

```
1 <context:component-scan base-package="com.proc.bean">
2     <context:exclude-filter type="assignable"
expression="com.proc.bean.controller.UserController"/>
3 </context:component-scan>
```

这里排除了 com.proc.bean.controller.UserController 类型的，所以只能够正确得到

com.proc.bean.TestObject@134517
com.proc.bean.service.UserService@50d156
com.proc.bean.repository.UserRepositoryImps@61c7e3

5、包含指定注解标记的类

```
1 <context:component-scan base-package="com.proc.bean" use-default-
filters="false">
2     <context:include-filter type="annotation"
expression="org.springframework.stereotype.Repository"/>
3 </context:component-scan>
```

这里只能够正确得到

com.proc.bean.repository.UserRepositoryImps@1200884

在使用 include-filter 是需要配合 use-default-filters="false"，让自动扫描注解不使用默认的 filter，而是使用我们指定的 filter，否则将无效

6、包含指定类型的类

```
1 <context:component-scan base-package="com.proc.bean" use-default-
filters="false">
2     <context:include-filter type="assignable"
expression="com.proc.bean.TestObject"/>
3 </context:component-scan>
```

这里只能够正确得到

com.proc.bean.TestObject@17f23d9

任务 3

掌握 Spring @Required 注解的使用

@Required 注释应用于 bean 属性的 setter 方法，它表明受影响的 bean 属性在配置时必须放在 XML 配置文件中，否则容器就会抛出一个 BeanInitializationException 异常。下面显示的是一个使用 @Required 注释的示例。

示例：

让我们使 Eclipse IDE 处于工作状态，请按照下列步骤创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 *SpringExample* 的项目，并且在所创建项目的 **src** 文件夹下创建一个名为 *com.tutorialspoint* 的包。
2	使用 *Add External JARs* 选项添加所需的 Spring 库文件，就如在 *Spring Hello World Example* 章节中解释的那样。
3	在 *com.tutorialspoint* 包下创建 Java 类 *Student* 和 *MainApp*。
4	在 **src** 文件夹下创建 Beans 配置文件 *Beans.xml*。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并且按如下解释的那样运行应用程序。

下面是 **Student.java** 文件的内容：

```
package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Required;
public class Student {
   private Integer age;
   private String name;
   @Required
   public void setAge(Integer age) {
      this.age = age;
   }
   public Integer getAge() {
      return age;
   }
   @Required
   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }
}
```

下面是 **MainApp.java** 文件的内容：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");
      Student student = (Student) context.getBean("student");
      System.out.println("Name : " + student.getName() );
      System.out.println("Age : " + student.getAge() );
   }
```

```
}
```

下面是配置文件 Beans.xml: 文件的内容：

```
<?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-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <!-- Definition for student bean -->
   <bean id="student" class="com.tutorialspoint.Student">
      <property name="name"  value="Zara" />

      <!-- try without passing age and check the result -->
      <!-- property name="age"  value="11"-->
   </bean>

</beans>
```

一旦你已经完成的创建了源文件和 bean 配置文件，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将引起 *BeanInitializationException* 异常，并且会输出一下错误信息和其他日志消息：

```
Property 'age' is required for bean 'student'
```

下一步，在你按照如下所示从 “age” 属性中删除了注释，你可以尝试运行上面的示例：

```
<?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-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <!-- Definition for student bean -->
   <bean id="student" class="com.tutorialspoint.Student">
```

```
    <property name="name"  value="Zara" />
    <property name="age"  value="11"/>
  </bean>

</beans>
```

现在上面的示例将产生如下结果：

```
Name : Zara
Age : 11
```

任务 4

学会使用 Spring @Autowired 注解

@Autowired 注释对在哪里和如何完成自动连接提供了更多的细微的控制。

@Autowired 注释可以在 setter 方法中被用于自动连接 bean，就像 @Autowired 注释，容器，一个属性或者任意命名的可能带有多个参数的方法。

Setter 方法中的 @Autowired

你可以在 XML 文件中的 setter 方法中使用 **@Autowired** 注释来除去 元素。当 Spring 遇到一个在 setter 方法中使用的 @Autowired 注释，它会在方法中视图执行 **byType** 自动连接。

示例

让我们使 Eclipse IDE 处于工作状态，然后按照如下步骤创建一个 Spring 应用程序：

步骤	描述
1	创建一个名为 SpringExample 的项目，并且在所创建项目的 src 文件夹下创建一个名为 com.tutorialspoint 的包。
2	使用 Add External JARs 选项添加所需的 Spring 库文件，就如在 Spring Hello World Example 章节中解释的那样。
3	在 com.tutorialspoint 包下创建 Java 类 TextEditor, SpellChecker 和 MainApp。
4	在 src 文件夹下创建 Beans 配置文件 Beans.xml。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容，并且按如下解释的那样运行应用程序。

这里是 **TextEditor.java** 文件的内容：

```
package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Autowired;
public class TextEditor {
   private SpellChecker spellChecker;
   @Autowired
   public void setSpellChecker( SpellChecker spellChecker ){
```

```
      this.spellChecker = spellChecker;
   }
   public SpellChecker getSpellChecker( ) {
      return spellChecker;
   }
   public void spellCheck() {
      spellChecker.checkSpelling();
   }
}
```

下面是另一个依赖的类文件 SpellChecker.java 的内容：

```
package com.tutorialspoint;
public class SpellChecker {
   public SpellChecker(){
      System.out.println("Inside SpellChecker constructor." );
   }
   public void checkSpelling(){
      System.out.println("Inside checkSpelling." );
   }
}
```

下面是 MainApp.java 文件的内容：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");
      TextEditor te = (TextEditor) context.getBean("textEditor");
      te.spellCheck();
   }
}
```

下面是配置文件 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-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">
```

```
   <context:annotation-config/>

   <!-- Definition for textEditor bean without constructor-arg  -->
   <bean id="textEditor" class="com.tutorialspoint.TextEditor">
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
   </bean>

</beans>
```

一旦你已经完成的创建了源文件和 bean 配置文件，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```
Inside SpellChecker constructor.
Inside checkSpelling.
```

属性中的 @Autowired

你可以在属性中使用 **@Autowired** 注释来除去 setter 方法。当时使用 为自动连接属性传递的时候，Spring 会将这些传递过来的值或者引用自动分配给那些属性。所以利用在属性中 @Autowired 的用法，你的 **TextEditor.java** 文件将变成如下所示：

```
package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Autowired;
public class TextEditor {
   @Autowired
   private SpellChecker spellChecker;
   public TextEditor() {
      System.out.println("Inside TextEditor constructor." );
   }
   public SpellChecker getSpellChecker( ){
      return spellChecker;
   }
   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

下面是配置文件 **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-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <!-- Definition for textEditor bean -->
   <bean id="textEditor" class="com.tutorialspoint.TextEditor">
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
   </bean>

</beans>
```

一旦你在源文件和 bean 配置文件中完成了上面两处改变，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```
Inside TextEditor constructor.
Inside SpellChecker constructor.
Inside checkSpelling.
```

构造函数中的 @Autowired

你也可以在构造函数中使用 @Autowired。一个构造函数 @Autowired 说明当创建 bean 时，即使在 XML 文件中没有使用 元素配置 bean ，构造函数也会被自动连接。让我们检查一下下面的示例。

这里是 **TextEditor.java** 文件的内容：

```
package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Autowired;
public class TextEditor {
   private SpellChecker spellChecker;
   @Autowired
   public TextEditor(SpellChecker spellChecker){
      System.out.println("Inside TextEditor constructor." );
      this.spellChecker = spellChecker;
   }
   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

下面是配置文件 **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-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:annotation-config/>

   <!-- Definition for textEditor bean without constructor-arg  -->
   <bean id="textEditor" class="com.tutorialspoint.TextEditor">
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
   </bean>

</beans>
```

一旦你在源文件和 bean 配置文件中完成了上面两处改变，让我们运行一下应用程序。如果你的应用程序一切都正常的话，这将会输出以下消息：

```
Inside SpellChecker constructor.Inside TextEditor constructor.
Inside checkSpelling.
```

@Autowired 的（required=false）选项

默认情况下，@Autowired 注释意味着依赖是必须的，它类似于 @Required 注释，然而，你可以使用 @Autowired 的 **（required=false）** 选项关闭默认行为。
即使你不为 age 属性传递任何参数，下面的示例也会成功运行，但是对于 name 属性则需要一个参数。你可以自己尝试一下这个示例，因为除了只有 **Student.java** 文件被修改以外，它和 @Required 注释示例是相似的。
Student.java 文件内容：

```
package com.tutorialspoint;
import org.springframework.beans.factory.annotation.Autowired;
public class Student {
   private Integer age;
   private String name;
   @Autowired(required=false)
   public void setAge(Integer age) {
      this.age = age;
   }
   public Integer getAge() {
```

```
        return age;
    }
    @Autowired
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```

其中，setAge()方法使用 @Autowired 的 （required=false） 选项关闭默认行为。

配置文件 Beans.xml 如下

```
<?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-3.0.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-3.0.xsd">

       <context:annotation-config/>

    <!-- Definition for textEditor bean without constructor-arg  -->
    <bean id="student" class="hello.Student" >
        <property name="name" value="番茄"></property>
    </bean>
</beans>
```

这里，，没有 XML 文件中设置属性 age 的值，仍能正确运行该程序，结果如下：

```
name: 番茄

age: 0
```