1.前言

控制反转（`IoC`）是一种思想，而依赖注入（`Dependency Injection`）则是实现这种思想的方法

我们前面写程序的时候，通过控制反转，使得 Spring 可以创建对象，这样减低了耦合性，但是每个类或模块之间的依赖是不可能完全消失的，而这种依赖关系，我们可以完全交给 spring 来维护。

2.注入分类

bean 实例在调用无参构造器创建对象后，就要对 bean 对象的属性进行初始化。初始化是由容器自动完成的，称为注入。

根据注入方式的不同，常用的有两类：`set 注入`、`构造注入`。

2.1.set 注入(掌握)

set 注入也叫设值注入，是指通过 setter 方法传入被调用者的实例。这种注入方式简单、直观，因而在 Spring 的依赖注入中大量使用。

2.1.1. 简单类型属性的注入

```
public class School {

    private String name;
    private String address;

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "School{" +
                "name='" + name + '\'' +
                ", address='" + address + '\'' +
                '}';
    }
}
```

beans.xml（配置文件的名字可以是任意的）

```
<?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">

    <!--声明student对象
        注入：就是赋值的意思
        简单类型： spring中规定java的基本数据类型和String都是简单
类型。
        di:给属性赋值
        1. set注入（设值注入） ：spring调用类的set方法， 你可以在
set方法中完成属性赋值
         1）简单类型的set注入
            <bean id="xx" class="yyy">
               <property name="属性名字" value="此属性的值"/>
               一个property只能给一个属性赋值
               <property....>
            </bean>
    -->
    <bean id="myStudent" class="com.ba01.Student" >
        <property name="name" value="李四lisi" /><!--
setName("李四")-->
        <property name="age" value="22" /><!--setAge(21)--
>
        <property name="email" value="lisi@qq.com" /><!--
setEmail("lisi@qq.com")-->
    </bean>

</beans>
```

测试方法

```
   @Test
    public void test01(){
       System.out.println("=====test01========");
       String config="ba01/applicationContext.xml";
       ApplicationContext ac = new
ClassPathXmlApplicationContext(config);

       //从容器中获取Student对象
       Student myStudent =  (Student)
ac.getBean("myStudent");
       System.out.println("student对象="+myStudent);
   }
```

创建 `java.util.Date` 并设置初始的日期时间：

beams.xml：

```
 <bean id="mydate" class="java.util.Date">
     <property name="time" value="8364297429" /><!--
setTime(8364297429)-->
 </bean>
```

测试方法：

```
   @Test
    public void test01(){
       System.out.println("=====test01========");
       String config="ba01/beans.xml";
       ApplicationContext ac = new
ClassPathXmlApplicationContext(config);

       //从容器中获取Student对象
       Student myStudent =  (Student)
ac.getBean("myStudent");
       System.out.println("student对象="+myStudent);


       Date myDate = (Date) ac.getBean("mydate");
       System.out.println("myDate="+myDate);

   }
```

2.1.2. 引用类型属性的注入

当指定 bean 的某属性值为另一 bean 的实例时，通过 ref 指定它们间的引用关系。ref的值必须为某 bean 的 id 值。

```
public class School {

    private String name;
    private String address;

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "School{" +
                "name='" + name + '\'' +
                ", address='" + address + '\'' +
                '}';
```

```
    }
}
```

```
public class Student {

    private String name;
    private int age;

    //声明一个引用类型
    private School school;


    public Student() {
        System.out.println("spring会调用类的无参数构造方法创建
对象");
    }

    // 包名.类名.方法名称
    public void setName(String name) {
        System.out.println("setName:"+name);
        this.name = name;
    }

    public void setAge(int age) {
        System.out.println("setAge:"+age);
        this.age = age;
    }

    public void setSchool(School school) {
        System.out.println("setSchool:"+school);
        this.school = school;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", school=" + school +
                '}';
    }
}
```

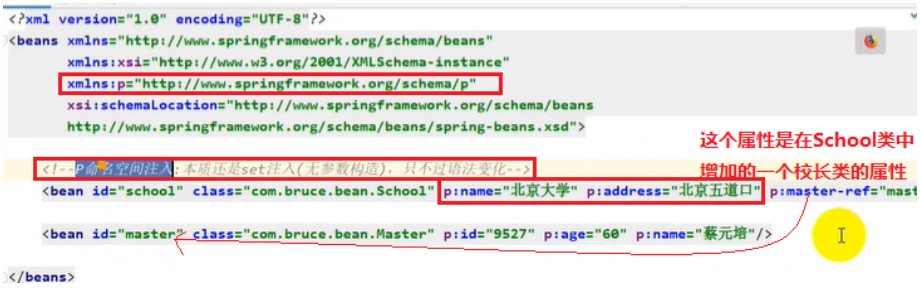
beans.xml

```
<?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">

    <!--声明student对象
        注入：就是赋值的意思
        简单类型： spring中规定java的基本数据类型和String都是简单
类型。
        di:给属性赋值
        1. set注入（设值注入） ：spring调用类的set方法， 你可以在
set方法中完成属性赋值
         1）简单类型的set注入
            <bean id="xx" class="yyy">
               <property name="属性名字" value="此属性的值"/>
               一个property只能给一个属性赋值
               <property....>
            </bean>

         2) 引用类型的set注入 ： spring调用类的set方法
           <bean id="xxx" class="yyy">
              <property name="属性名称" ref="bean的id(对象的
名称)" />
           </bean>
    -->
    <bean id="myStudent" class="com.ba02.Student" >
        <property name="name" value="李四" />
        <property name="age" value="26" />
        <!--引用类型-->
        <property name="school" ref="mySchool" /><!--
setSchool(mySchool)-->
    </bean>

    <!--声明School对象-->
    <bean id="mySchool" class="com.ba02.School">
        <property name="name" value="北京大学"/>
        <property name="address" value="北京的海淀区" />
    </bean>
</beans>
```

测试方法：

```
@Test
 public void test01(){
    System.out.println("=====test01========");
    String config="ba02/beans.xml";
    ApplicationContext ac = new
ClassPathXmlApplicationContext(config);

    //从容器中获取Student对象
    Student myStudent =  (Student)
ac.getBean("myStudent");
    System.out.println("student对象="+myStudent);
 }
```

2.1.3 setter注入变体——>P命名空间注入



2.1.4 使用注解生成gettter、setter

在pom.xml中添加依赖：

```
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.16.18</version>
            <scope>provided</scope>
        </dependency>
```

之后可以在Bean的实体类中使用@Data、@NoArgsConstructor、@AllArgsConstructor等注解代替手写构造、toString.....方法

关于这些注解的官方文档：https://projectlombok.org/features/all

官网地址：https://projectlombok.org/

lombok的Github地址：https://github.com/rzwitserloot/lombok

2.2. 构造注入(理解)

构造注入是指，在构造调用者实例的同时，完成被调用者的实例化。即，使用构造器设置依赖关系。

举例 1：

```
    /**
     * 创建有参数构造方法
     */
    public Student(String myname,int myage, School
mySchool){
        System.out.println("=====Student有参数构造方法
======");
        //属性赋值
        this.name  = myname;
        this.age  = myage;
        this.school = mySchool;

    }
```

spring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"

xsi:schemaLocation="http://www.springframework.org/schema/
beans http://www.springframework.org/schema/beans/spring-
beans.xsd">

    <!--声明student对象
        注入：就是赋值的意思
        简单类型： spring中规定java的基本数据类型和String都是简单
类型。
        di:给属性赋值
        1. set注入（设值注入） ：spring调用类的set方法， 你可以在
set方法中完成属性赋值
         1）简单类型的set注入
            <bean id="xx" class="yyy">
               <property name="属性名字" value="此属性的值"/>
               一个property只能给一个属性赋值
               <property....>
            </bean>

         2) 引用类型的set注入 ： spring调用类的set方法
           <bean id="xxx" class="yyy">
              <property name="属性名称" ref="bean的id(对象的
名称)" />
           </bean>

        2.构造注入：spring调用类有参数构造方法，在创建对象的同时，
在构造方法中给属性赋值。
          构造注入使用 <constructor-arg> 标签
          <constructor-arg> 标签：一个<constructor-arg>表示
构造方法一个参数。
          <constructor-arg> 标签属性：
```

```
            name:表示构造方法的形参名
            index:表示构造方法的参数的位置，参数从左往右位置是 0
，  1 ，2的顺序
            value：构造方法的形参类型是简单类型的，使用value
            ref：构造方法的形参类型是引用类型的，使用ref
    -->


    <!--使用name属性实现构造注入-->
    <bean id="myStudent" class="com.ba03.Student" >
        <constructor-arg name="myage" value="20" />
        <constructor-arg name="mySchool" ref="myXueXiao"
/>
        <constructor-arg name="myname" value="周良"/>
    </bean>

    <!--使用index属性-->
    <bean id="myStudent2" class="com.ba03.Student">
        <constructor-arg index="1" value="22" />
        <constructor-arg index="0" value="李四" />
        <constructor-arg index="2" ref="myXueXiao" />
    </bean>

    <!--省略index-->
    <bean id="myStudent3" class="com.ba03.Student">
        <constructor-arg  value="张强强" />
        <constructor-arg  value="22" />
        <constructor-arg  ref="myXueXiao" />
    </bean>
    <!--声明School对象-->
    <bean id="myXueXiao" class="com.ba03.School">
        <property name="name" value="清华大学"/>
        <property name="address" value="北京的海淀区" />
    </bean>

</beans>
```

标签中用于指定参数的属性有：
➢ name：指定参数名称。
➢ index：指明该参数对应着构造器的第几个参数，从 0 开始。不过，该属性不要也行，但要注意，若参数类型相同，或之间有包含关系，则需要保证赋值顺序要与构造器中的参数顺序一致。

举例 2：
使用构造注入创建一个系统类 File 对象

```
<!--创建File,使用构造注入-->
<bean id="myfile" class="java.io.File">
    <constructor-arg name="parent"
value="D:\course\JavaProjects\spring-course\ch01-hello-
spring" />
    <constructor-arg name="child" value="readme.txt" />
</bean>
```

测试类：

```
    @Test
     public void test01(){
        System.out.println("=====test01========");
        String config="ba03/applicationContext.xml";
        ApplicationContext ac = new
ClassPathXmlApplicationContext(config);

        //从容器中获取Student对象
        Student myStudent =  (Student)
ac.getBean("myStudent");
        System.out.println("student对象="+myStudent);

        File myFile = (File) ac.getBean("myfile");
        System.out.println("myFile=="+myFile.getName());

    }
```

2.3.bean标签中的autowire属性

引用类型属性自动注入

对于引用类型属性的注入，也可不在配置文件中显示的注入（也就是不使用property元素）。可以通过为`<bean/>`标签设置 autowire 属性值，为引用类型属性进行隐式自动注入；
该属性的默认值是：no，不自动注入引用类型属性；还有其他值byName、byType。

根据自动注入判断标准的不同，可以分为两种：

> 1. *byName：根据名称自动注入；以xml形式：Spring框架在底层根据xml配置文件中这个byName属性所在的bean标签对应的Bean实体类中的引用类型的属性的名字（记作F），在配置文件中自动查找并匹配id为F的bean标签。*
> 2. *byType： 根据类型自动注入；Spring框架在底层根据这个byType属性所在的bean标签对应的Bean实体类中的引用类型的属性的类型（记作T），在配置文件中自动查找并匹配class与T同源的bean标签；*
>
> *同源含义：*
>
> *1.T的数据类型和配置文件中bean元素的class的值是一样的。*
>
> *2.T的数据类型和配置文件bean元素的class的值父子类关系的。*
>
> *3.T的数据类型和配置文件bean元素的class的值接口和实现*

> *类关系的。*
> *注意：在byType中，在xml配置文件中声明bean只能有一个符合条件的，多余一个是错误的*

2.3.1.byName 方式自动注入

当配置文件中被调用者 bean 的 id 值与代码中调用者 bean 类的属性名相同时，可使用byName 方式，让容器自动将被调用者 bean 注入给调用者 bean。容器是通过调用者的 bean类的属性名与配置文件的被调用者 bean 的 id 进行比较而实现自动注入的。

举例：

```
public class School {

    private String name;
    private String address;

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "School{" +
                "name='" + name + '\'' +
                ", address='" + address + '\'' +
                '}';
    }
}
```

```
public class Student {

    private String name;
    private int age;

    //声明一个引用类型
    private School school;


    public Student() {
        //System.out.println("spring会调用类的无参数构造方法创建对象");
    }

    public void setName(String name) {
        //System.out.println("setName:"+name);
```

```
        this.name = name;
    }

    public void setAge(int age) {
        //System.out.println("setAge:"+age);
        this.age = age;
    }

    public void setSchool(School school) {
        System.out.println("setSchool:"+school);
        this.school = school;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", school=" + school +
                '}';
    }
}
```

Spring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"

xsi:schemaLocation="http://www.springframework.org/schema/
beans http://www.springframework.org/schema/beans/spring-
beans.xsd">


    <!--
            引用类型的自动注入： spring框架根据某些规则可以给引用
类型赋值。·不用你在给引用类型赋值了
        使用的规则常用的是byName, byType.
        1.byName(按名称注入) ： java类中引用类型的属性名和spring
容器中（配置文件）<bean>的id名称一样，
                                且数据类型是一致的，这样的容器中的
bean，spring能够赋值给引用类型。
```

```
            语法：
            <bean id="xx" class="yyy" autowire="byName">
                简单类型属性赋值
            </bean>

        2.byType(按类型注入) ： java类中引用类型的数据类型和
spring容器中（配置文件）<bean>的class属性
                                        是同源关系的，这样的bean能够赋值
给引用类型
            同源就是一类的意思：
             1.java类中引用类型的数据类型和bean的class的值是一样的。
             2.java类中引用类型的数据类型和bean的class的值父子类关系
的。
             3.java类中引用类型的数据类型和bean的class的值接口和实现
类关系的
            语法：
            <bean id="xx" class="yyy" autowire="byType">
                简单类型属性赋值
            </bean>
            注意：在byType中， 在xml配置文件中声明bean只能有一个符合
条件的，
                    多余一个是错误的
    -->
    <!--byName-->
    <bean id="myStudent" class="com.ba04.Student"
 autowire="byName">
        <property name="name" value="李四" />
        <property name="age" value="26" />
        <!--引用类型-->
        <!--<property name="school" ref="mySchool" />-->
    </bean>

    <!--声明School对象-->
    <bean id="school" class="com.ba04.School">
        <property name="name" value="清华大学"/>
        <property name="address" value="北京的海淀区" />
    </bean>


</beans>
```

测试：

```
    @Test
    public void test01(){
        String config="ba04/applicationContext.xml";
        ApplicationContext ac = new
ClassPathXmlApplicationContext(config);

        //从容器中获取Student对象
        Student myStudent =  (Student)
ac.getBean("myStudent");
        System.out.println("student对象="+myStudent);
    }
```

2.3.2.byType 方式自动注入

使用 byType 方式自动注入，要求：配置文件中被调用者 bean 的 class 属性指定的类，
要与代码中调用者 bean 类的某引用类型属性类型同源。即要么相同，要么有 is-a 关系（子类，或是实现类）。但这样的同源的被调用 bean 只能有一个。多于一个，容器就不知该匹配哪一个了。

举例：

```
public class School {

    private String name;
    private String address;

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "School{" +
                "name='" + name + '\'' +
                ", address='" + address + '\'' +
                '}';
    }
}
```

```
public class Student {

    private String name;
    private int age;

    //声明一个引用类型
    private School school;
```

```
    private School school2;


    public Student() {
        //System.out.println("spring会调用类的无参数构造方法创
建对象");
    }

    public void setName(String name) {
        //System.out.println("setName:"+name);
        this.name = name;
    }

    public void setAge(int age) {
        //System.out.println("setAge:"+age);
        this.age = age;
    }

    public void setSchool(School school) {
        System.out.println("setSchool:"+school);
        this.school = school;
    }

    public void setSchool2(School school2) {
        System.out.println("setSchool2222222:"+school);
        this.school2 = school2;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", school=" + school +
                ", school2=" + school2 +
                '}';
    }
}
```

```
// 子类
public class PrimarySchool extends  School {
}
```

Spring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"

xsi:schemaLocation="http://www.springframework.org/schema/
beans http://www.springframework.org/schema/beans/spring-
beans.xsd">



    <!--
            引用类型的自动注入： spring框架根据某些规则可以给引用
类型赋值。.不用你在给引用类型赋值了
        使用的规则常用的是byName, byType.
        1.byName(按名称注入) ： java类中引用类型的属性名和spring
容器中（配置文件）<bean>的id名称一样，
                                且数据类型是一致的，这样的容器中的
bean，spring能够赋值给引用类型。
          语法：
          <bean id="xx" class="yyy" autowire="byName">
             简单类型属性赋值
          </bean>

        2.byType(按类型注入) ： java类中引用类型的数据类型和
spring容器中（配置文件）<bean>的class属性
                                是同源关系的，这样的bean能够赋值
给引用类型
          同源就是一类的意思：
           1.java类中引用类型的数据类型和bean的class的值是一样的。
           2.java类中引用类型的数据类型和bean的class的值父子类关系
的。
           3.java类中引用类型的数据类型和bean的class的值接口和实现
类关系的
          语法：
          <bean id="xx" class="yyy" autowire="byType">
             简单类型属性赋值
          </bean>

          注意：在byType中， 在xml配置文件中声明bean只能有一个符合
条件的，
              多余一个是错误的
    -->
    <!--byType-->
    <bean id="myStudent" class="com.ba05.Student"
 autowire="byType">
        <property name="name" value="张飒" />
        <property name="age" value="26" />
        <!--引用类型-->
        <!--<property name="school" ref="mySchool" />-->
    </bean>
```

```
    <!--声明School对象-->
    <bean id="mySchool" class="com.ba05.School">
        <property name="name" value="人民大学"/>
        <property name="address" value="北京的海淀区" />
    </bean>

    <!--声明School的子类-->
    <!--<bean id="primarySchool"
class="com.ba05.PrimarySchool">
        <property name="name" value="北京小学" />
        <property name="address" value="北京的大兴区" />
    </bean>-->


</beans>
```

测试：

```
 @Test
  public void test01(){
     String config="ba05/applicationContext.xml";
     ApplicationContext ac = new
ClassPathXmlApplicationContext(config);

     //从容器中获取Student对象
     Student myStudent =  (Student)
ac.getBean("myStudent");
     System.out.println("student对象="+myStudent);
 }
```

2.3.3.为应用指定多个 Spring 配置文件

在实际应用里，随着应用规模的增加，系统中 Bean 数量也大量增加，导致配置文件变
得非常庞大、臃肿。为了避免这种情况的产生，提高配置文件的可读性与可维护性，可以将Spring 配置文件分解成多个配置文件。

包含关系的配置文件：
多个配置文件中有一个总文件，总配置文件将各其它子文件通过`<import/>`引入。在 Java代码中只需要使用总配置文件对容器进行初始化即可。

语法：
　　可以在resource的值中使用classpath关键字
　　关键字："classpath:" 表示我们自己编写的代码编译后在Maven中target文件夹
　　关键字："classpath*:" 表示我们自己编写的代码编译后在Maven中target文件夹、第三方jar包下的类路径
　　类路径：class文件所在的目录，这些class文件可以是我们自己写的代码编译后在Maven项目下的target文件夹
　　中的class文件，也可以是第三方jar包下的classpath路径中包含的class

文件

在spring的配置文件中要指定其他文件的位置， 需要使用classpath，告诉spring到哪去加载读取文件。

使用方式：

举例：

```
public class School {

    private String name;
    private String address;

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "School{" +
                "name='" + name + '\'' +
                ", address='" + address + '\'' +
                '}';
    }
}
```

```
public class Student {

    private String name;
    private int age;

    //声明一个引用类型
    private School school;


    public Student() {
        //System.out.println("spring会调用类的无参数构造方法创建对象");
    }

    public void setName(String name) {
        //System.out.println("setName:"+name);
        this.name = name;
    }
```

```
    public void setAge(int age) {
        //System.out.println("setAge:"+age);
        this.age = age;
    }

    public void setSchool(School school) {
        System.out.println("setSchool:"+school);
        this.school = school;
    }


    public String getName() {
        return name;
    }


    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", school=" + school +
                '}';
    }
}
```

spring-school.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">



    <!--School模块所有bean的声明， School模块的配置文件-->
    <!--声明School对象-->
    <bean id="mySchool" class="com.ba06.School">
        <property name="name" value="航空大学"/>
        <property name="address" value="北京的海淀区" />
    </bean>

</beans>
```

spring-student.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">


    <!--
      student模块所有bean的声明
    -->
    <!--byType-->
    <bean id="myStudent" class="com.ba06.Student"
 autowire="byType">
        <property name="name" value="张飒" />
        <property name="age" value="30" />
        <!--引用类型-->
        <!--<property name="school" ref="mySchool" />-->
    </bean>

</beans>
```

total.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">

    <!--
         包含关系的配置文件:
         spring-total表示主配置文件 ：  包含其他的配置文件的，主配
置文件一般是不定义对象的。
         语法：<import resource="其他配置文件的路径" />

    -->

    <!--加载的是文件列表-->
    <!--
    <import resource="classpath:ba06/spring-school.xml" />
    <import resource="classpath:ba06/spring-student.xml"
/>
    -->

    <!--
```

```
        在包含关系的配置文件中，可以通配符（*：表示任意字符）
        注意： 主的配置文件名称不能包含在通配符的范围内（不能叫做
spring-total.xml）
    -->
    <import resource="classpath:ba06/spring-*.xml" />
</beans>
```

测试:

```
@Test
 public void test01(){
   //加载的是总的文件
    String config= "ba06/total.xml";
    ApplicationContext ac = new
ClassPathXmlApplicationContext(config);

    //从容器中获取Student对象
    Student myStudent =  (Student)
ac.getBean("myStudent");
    System.out.println("student对象="+myStudent);
}
```

2.3.4.注入集合属性

为了演示这些方式，我们在成员中将常见的一些集合都写出来，然后补充其 set 方法

```
private String[] strs;
private Car[] cars; //自定义的对象数组
private List<String> list;
private Set<String> set;
private Map<String,String> map;
private Properties props;
```

在配置中也是很简单的，只需要按照下列格式写标签就可以了，可以自己测试一下

```
<!--
这是自定义Car类的bean标签
<bean id="car1" class="xxx.xxx.Car">
    <constructor-arg value="悍马"/>
</bean>
<bean id="car2" class="xxx.xxx.Car">
    <constructor-arg value="小鸟"/>
</bean>
-->
<bean id="accountService"
class="cn.ideal.service.impl.AccountServiceImpl">
    <property name="strs">
        <array>
```

```
            <value>张三</value>
            <value>李四</value>
            <value>王五</value>
        </array>
    </property>

    <property name="cars">
        <array>
            <ref bean="car1"/>
            <ref bean="car2"/>
        </array>
    </property>

    <property name="list">
        <list>
            <value>张三</value>
            <value>李四</value>
            <value>王五</value>
        </list>
    </property>

    <property name="set">
        <set>
            <value>张三</value>
            <value>李四</value>
            <value>王五</value>
        </set>
    </property>

    <property name="map">
        <map>
            <entry key="name" value="张三"></entry>
            <entry key="age" value="21"></entry>
        </map>
    </property>

    <property name="props">
        <props>
            <prop key="name">张三</prop>
            <prop key="age">21</prop>
        </props>
    </property>
</bean>
```

3.基于注解的 DI

3.0本节介绍的注解们：

1. @ComponentScan
2. @Configuration
3. @Import
4. @Bean————————————————————————
5. @Componment
6. @Controller　　定义Bean的注解，相当于
7. @Respority　　在配置文件中的bena标签
8. @Service
9. @Scope ————————— 这三个注解代替配置文件中——
10. @PostConstruct　　bean标签的属性：scope、
11. @PreDestory　　destroy-method、init-method
    ————————————————————————————
12. @Value
13. @PropertySource　　用于属性注入，在3.3节中
14. @Autowired　　介绍了注入的三种方式
15. @Qualifier
16. @Resource
    ————————————————————————————

3.1 扫描注解的三种方式

对于 DI 使用注解，将不再需要在 Spring 配置文件中声明 bean 实例。

3.1.1 方式一（配置文件）

在配置文件中使用`<context:annotation-config/>`开启注解处理器，这种方式类似于方式二，但是使用这种方式还需要在配置文件按中增加标签来配置项目中的Bean实体类，而方式二不需要，只需要在Bean实体类上加上注解就可以。

3.1.2 方式二（配置文件+注解）

Spring 中使用注解，需要在原有 Spring 运行环境基础上再做一些改变。
需要在 Spring 配置文件中配置组件扫描器，用于在指定的基本包中扫描注解。

```
<?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/conte
xt"

xsi:schemaLocation="http://www.springframework.org/schema/
beans
       http://www.springframework.org/schema/beans/spring-
beans.xsd
       http://www.springframework.org/schema/context

https://www.springframework.org/schema/context/spring-
context.xsd">
```

```
    <!--声明组件扫描器(component-scan),组件就是java对象
        base-package：指定注解在你的项目中的包名。
        component-scan工作方式： spring会扫描遍历base-package
指定的包，
            把包中和子包中的所有类，找到类中的注解，按照注解的功能创
建对象，或给属性赋值。

        加入了component-scan标签，配置文件的变化：
        1.加入一个新的约束文件spring-context.xsd
        2.给这个新的约束文件起个命名空间的名称
    -->
    <context:component-scan base-package="com.bruce.ba02"
/>

</beans>
```

指定多个包的三种方式：

- 使用多个 context:component-scan 指定不同的包路径

```
<context:component-scan base-package="com.bruce.ba02" />
<context:component-scan base-package="com.bruce.ba03" />
```

- 指定 base-package 的值使用分隔符
  分隔符可以使用逗号（，）分号（；）还可以使用空格，不建议使用空格。

逗号分隔：

```
  <context:component-scan base-
package="com.bruce.ba02,com.bruce.ba03" />
```

分号分隔：

```
<context:component-scan base-
package="com.bruce.ba02;com.bruce.ba03" />
```

- base-package 指定到父包名
  base-package 的值表是基本包，容器启动会扫描包及其子包中的注解，当然也会扫描到子包下级的子包。所以 base-package 可以指定一个父包就可以。

```
<context:component-scan base-package="com.bruce" />
```

或者最顶级的父包

```
<context:component-scan base-package="com" />
```

> *但不建议使用顶级的父包，扫描的路径比较多，导致容器启动时间变慢。指定到目标包和合适的。也就是注解所在包全路径。*

3.1.3 方式三（使用@ComponentScan注解）

可以去看链接：https://www.cnblogs.com/jpfss/p/11171655.html

配置组件扫描指令以与 @Configuration 类一起使用。提供与 Spring XML 的 context:component-scan 元素并行的支持。

可以指定 basePackageClasses() 或 basePackages()（或其别名 value()）来定义要扫描的特定包。如果未定义特定的包，则会从声明此注解的类所在的包中进行扫描。

该注解的元素：

1. value：对应的包扫描路径 可以是单个路径，也可以是扫描的路径数组
2. basePackages：和**value**一样是对应的包扫描路径 可以是单个路径，也可以是扫描的路径数组
3. basePackageClasses：指定具体的扫描类
4. nameGenerator：对应的bean名称的生成器，默认的是BeanNameGenerator
5. scopeResolver：处理检测到的bean的scope范围
6. resourcePattern：控制符合组件检测条件的类文件 默认是包扫描下的
7. useDefaultFilters：是否对带有**@Component @Repository @Service @Controller**注解的类开启检测,默认是开启的
8. includeFilters：指定某些定义Filter满足条件的组件 FilterType有5种类型如： ANNOTATION, 注解类型 默认 ASSIGNABLE_TYPE,指定固定类
   ASPECTJ， ASPECTJ类型
   REGEX,正则表达式
   CUSTOM,自定义类型
9. excludeFilters：排除某些过滤器扫描到的类
10. lazyInit：扫描到的类是都开启懒加载 ，默认是不开启的

具体使用方式：

在包com.zhang.controller下新建一个UserController带@Controller注解如下：

```
package com.zhang.controller;
import org.springframework.stereotype.Controller;
@Controller
public class UserController {
}
```

在包com.zhang.service下新建一个UserService带@Service注解如下：

```
package com.zhang.service;
import org.springframework.stereotype.Service;
@Service
public class UserService {
}
```

在包com.zhang.dao下新建一个UserDao带@Repository注解如下：

```
package com.zhang.dao;
import org.springframework.stereotype.Repository;
@Repository
public class UserDao {
}
```

新建一个配置类如下：

```
/**
 * 主配置类  包扫描com.zhang
 *
 * @author zhangqh
 * @date 2018年5月12日
 */
@ComponentScan(value="com.zhang")
@Configuration
public class MainScanConfig {
}
```

新建测试方法如下：

```
AnnotationConfigApplicationContext applicationContext2 =
new
AnnotationConfigApplicationContext(MainScanConfig.class);
        String[] definitionNames =
applicationContext2.getBeanDefinitionNames();
        for (String name : definitionNames) {
            System.out.println(name);
}
```

运行结果如下：

```
mainScanConfig
userController
userDao
userService
```

> *总结一下@ComponentScan的常用方式如下*
>
> - *自定扫描路径下边带有@Controller，@Service，@Repository，@Component注解加入spring容器*

> - *通过includeFilters加入扫描路径下没有以上注解的类加入spring容器*
> - *通过excludeFilters过滤出不用加入spring容器的类*
> - *自定义增加了@Component注解的注解方式*

3.2 定义 Bean 的注解@Component(掌握)

> *可以用此注解描述Spring中的Bean，但是它是一个泛化的概念，仅仅表示一个Bean组件，Spring创建的该注解注释的Bean实例默认是单例的。*
>
> *需要在类上使用注解@Component，该注解的value 属性用于指定该 bean 的id 值。*

```
// 注解参数中省略了value属性，该属性用于指定Bean的id
@Component("myStudent")
public class Student {
    private String name;
    private int age;
```


@Component 不指定value 属性，则bean 的id 是类名的首字母小写。

```
@Component
public class Student {
```

```
String configLocation = "applicationContext.xml";
ApplicationContext ac = new ClassPathXmlApplicationContext(configLocation);
Student student = (Student) ac.getBean("student");
```

另外，Spring 还提供了 3 个创建对象的注解：

```
➢ @Repository 用于对 DAO 实现类进行注解
➢ @Service 用于对 Service 实现类进行注解
➢ @Controller 用于对 Controller 实现类进行注解
```

这三个注解与@Component 都本质上一样（看源码），但Spring框架对这三个注解还有其他的含义：

1. @Service：创建业务层对象，业务层对象可以加入事务功能；
2. @Controller： 注解创建的对象可以作为处
   理器接收用户的请求。

@Repository，@Service，@Controller 是对@Component 注解的细化，标注不同层的对
象。即持久层对象，业务层对象，控制层对象。

3.2.1 Scope注解

该注解的默认值为singleton。

当与@Component一起用作类型级注解时，@Scope指示该Bean实例的作用范围；
当与@Bean一起用作方法级注解时，@Scope指示被注释的方法返回的Bean实例的作用范围；
在这种情况下，范围是指Bean实例的生命周期，例如单例、原型等等

3.2.2 PostConstruct、PreDestory注解

这两个注解等同关于配置文件中bean标签中的init-method属性和destory-method属性；
示例：

```
@PostConstruct
public void init(){
    System.out.println("AccountDaoImpl对象初始化.....");
}

@PreDestroy
public void destory(){
    System.out.println("AccountDaoImpl对象被销毁.....");
}
```

3.3 属性注入的三种方式：

1. 使用@Value注解（用于给简单的类型属性注入）
2. 使用@Autowired 或 @Autowired+@Qualifier注解（用于给引用类型的属性注入）
3. 使用@Resource注解（用于给引用类型的属性注入）

在后两种方式中都是对于引用类型属性的注入，也有按照名字和类型的注入方式，注解形式下的byName和byType：
在使用@Autowired、@Autowired+@Qualifier、@Resource这些注解标识的属性注入中，被注入的属性（记作M）的值可以按照M的名称（MM）获得，也可以按照M的类型（MT）进行匹配；具体使用方式看下面的示例代码。

3.3.1 简单类型属性注入@Value(掌握)

> *字段、方法|构造函数 参数级别的注释，当该注释用于参数上时指示受影响参数的默认值。*
>
> *@Value注释的实际处理是由BeanPostProcessor执行的，着反过来意味着不能在BeanPostProcessor或BeanFactoryPostProcessor类型中使用@Value*

需要在属性上使用注解@Value，该注解的 value 属性用于指定要注入的值。使用该注解完成属性注入时，类中无需 setter。当然，若属性有 setter，则也可将该注解加到 setter 上。

举例：

```
/**
```

```
 * @BelongsProject: Spring-2021
 * @BelongsPackage: com.bruce.dao.impl
 * @CreateTime: 2021-01-12 10:53
 * @Description: TODO
 */
@Component(value = "accountDao")
@Scope("singleton")  //默认值
public class AccountDaoImpl implements AccountDao {

    @Value("com.mysql.jdbc.Driver")
    private String driver;

    @Value("jdbc:mysql://localhost:3306/account2021?
characterEncoding=utf-8")
    private String url;

    @Value("root")
    private String username;

    @Value("123456")
    private String password;

    public List<Account> findAccounts() {
        System.out.println(driver);
        System.out.println(url);
        System.out.println(username);
        System.out.println(password);
        return null;
    }

}
```

@value注解可以读取外部的配置文件：

使用xml形式：

```
<context:property-placeholder
location="classpath*:db.properties" file-encoding="UTF-
8"/>
```

使用注解形式：

```
//可以使用这个注解代替上面的配置文件方式
@PropertySource(value =
"classpath:db.properties",ignoreResourceNotFound = true)
```

```
@Component(value = "accountDao")
@Scope("singleton")  //默认值
```

```
@PropertySource(value =
"classpath:db.properties",ignoreResourceNotFound = true)

public class AccountDaoImpl implements AccountDao {
    //这里使用的是SPEL表达式
    @Value("${jdbc.driver}")
    private String driver;
    //这里使用的是SPEL表达式
    @Value("${jdbc.url}")
    private String url;
    //这里使用的是SPEL表达式
    @Value("${jdbc.username}")
    private String username;
    //这里使用的是SPEL表达式
    @Value("${jdbc.password}")
    private String password;

    public List<Account> findAccounts() {
        System.out.println(driver);
        System.out.println(url);
        System.out.println(username);
        System.out.println(password);
        return null;
    }

}
```

3.3.1.1 @PropertySource

该注解提供了一种方便的声明机制，用于将PropertySource添加到Spring环境中，与@Configuration类一起使用。

有一个包含 testbean.name=myTestBean 形式的键值对的app.properties文件，下面被@Configuration注解修饰的配置类用@PropertySource将app.properties配置文件用于Environment的一组PropertySource：

```
 @Configuration
 @PropertySource("classpath:/com/myco/app.properties")
 public class AppConfig {
     @Autowired
     Environment env;

     @Bean
     public TestBean testBean() {
         TestBean testBean = new TestBean();

testBean.setName(env.getProperty("testbean.name"));
         return testBean;
     }
 }
```

注意：Environment（这个接口表示当前应用程序运行时环境的接口）类型的对象是使用@Autowired注解注入到到配置类中，然后在填充TestBean对象时使用；鉴于上述的配置，对testBean.getName()的调用将返回“myTestBean”；

该注解中的元素：

1. value（必要）
   指定要加载的书信文件的资源位置；
   该属性的值不允许使用资源位置通配符（如：*/.properties）,每个配置文件必须被精确的评估为yige.properties资源；
2. encoding（可选）
   给定资源的特定字符编码，例如“UTF-8”。
3. factory（可选）
   指定自定义 PropertySourceFactory（如果有）。 默认情况下，将使用标准资源文件的默认工厂。
4. ignoreResourceNotFound（可选）
   如果没有找到一个配置文件，是否应该被忽略；当这个属性的值是true时，表示配置文件完全是可选的，也就是找不到就忽略，这个属性的默认值为false；
   在这个注解中不能使用通配符，否则会报错。
5. name（可选）
   指明此属性源的名称。如果省略，将根据底层资源的描述生成一个名称。

3.3.1.2 @Configuration

这个注解本质和@Componment注解的作用一样（看源码），只是这个注解专门用于配置类，也就是将一个配置文件写成配置类，Spring容器中会生成这个配置类的Bean实例。

没有被@Configuration注解的类其中的@Bean方法也能被扫描到，这是Spring中的lits模式，标注@Configuration注解的类是full模式。

> *来自：《关于Spring中的@Configuration注解.md》这里只有定义的部分，详细内容看链接。*
>
> ***Full模式和Lite模式：***
>
> 1. *Lite模式——>当@Bean方法在没有使用@Configuration注释的类中声明时，它们被称为在Lite模式下处理。它包括：在@Componment中声明的@Bean，甚至只是在一个非常普通的类中声明的Bean方法，都被认为是Lits版的配置类；*
>   *与Full模式的@Configuration不同，Lite模式的@Bean方法不能声明Bean之间的依赖关系；因此，这样的@Bean方法不应该调用其他@Bean方法；每个这样的方法实际上只是一个特定Bean引用的工厂方法，没有任何特殊的运行时含义；*
>
>   > *何时为Lite模式?*
>   >
>   > *官方定义为：在没有标注`@Configuration`的类里面有`@Bean`方法就称为Lite模式的配置。透过源码再看这个定义是不完全正确的，而应该是有如下case均认为是Lite模式的配置类：*
>   >
>   > a. *类上标注有`@Component`注解*

> b. *类上标注有`@ComponentScan`注解*
> c. *类上标注有`@Import`注解*
> d. *类上标注有`@ImportResource`注解*
> e. *若类上没有任何注解，但类内存在@Bean方法*
>
> *在**Spring 5.2**之后新增了一种case也算作Lite模式：*
>
> a. *标注有`@Configuration(proxyBeanMethods = false)`，注意：此值默认是true哦，需要显示改为false才算是Lite模式*
>
> *细心的你会发现，自Spring5.2（对应Spring Boot 2.2.0）开始，内置的几乎所有的`@Configuration`配置类都被修改为了`@Configuration(proxyBeanMethods = false)`，目的何为？答：以此来降低启动时间，为Cloud Native继续做准备。*

2. *Full模式——>@Bean方法都会在标注有@Configuration的类中声明，以确保总是使用“Full模式”，交叉方法引用会被重定向到容器的生命周期管理，所以就可以更方便的管理Bean依赖。*

> *何时为Full模式？*
>
> *标注有`@Configuration`注解的类被称为full模式的配置类。自Spring5.2后这句话改为下面这样我觉得更为精确些：*
>
> - *标注有`@Configuration`或者`@Configuration(proxyBeanMethods = true)`的类被称为Full模式的配置类*
> - *（当然喽，proxyBeanMethods属性的默认值是true，所以一般需要Full模式我们只需要标个注解即可）*

3.3.1.3 @Import

指示要导入的一个或多个@Configuration；

提供与 Spring XML 中的 元素等效的功能。允许导入@Configuration 类、ImportSelector 和 ImportBeanDefinitionRegistrar 实现，以及常规组件类；

在导入的@Configuration 类中声明的@Bean 定义应该使用@Autowired 注入来访问。

如果需要导入 XML 或其他非@Configuration 的 bean 定义资源，请改用 @ImportResource 注解。

```
//Spring API中注解@Configuration中的示例代码
 @Configuration
 public class DatabaseConfig {

     @Bean
     public DataSource dataSource() {
```

```
        // instantiate, configure and return DataSource
    }
}

@Configuration
@Import(DatabaseConfig.class)
public class AppConfig {

    private final DatabaseConfig dataConfig;

    public AppConfig(DatabaseConfig dataConfig) {
        this.dataConfig = dataConfig;
    }

    @Bean
    public MyBean myBean() {
        // reference the dataSource() bean method
        return new MyBean(dataConfig.dataSource());
    }
}
```

3.3.1.4 @Bean

这个注解指示一个方法生成一个由于Spring容器管理的bean；这个注解可以在未使用@Configuration注释的类(在这种情况下，@Bean方法将以所谓的“精简”模式进行处理)中使用，例如被这个注解注释的方法可以在@Componment类中声明。

此注解的属性名称和语义与SpringXML模式中的元素的名称和语义相似。

该注解不提供范围、依赖、基础或惰性的属性，相反他应该和@Scope、@Primary注释一起使用以实现这些语义

该注解的部分元素：

1. name：虽然name属性可以使用，但确定bean名称的默认策略是使用@Bean注释的方法的名称；该元素接受一个字符串数组，允许单个bean有多个名称（即一个主bean名称加上一个或多个别名）

3.3.2 引用类型属性注入@Autowired(掌握)

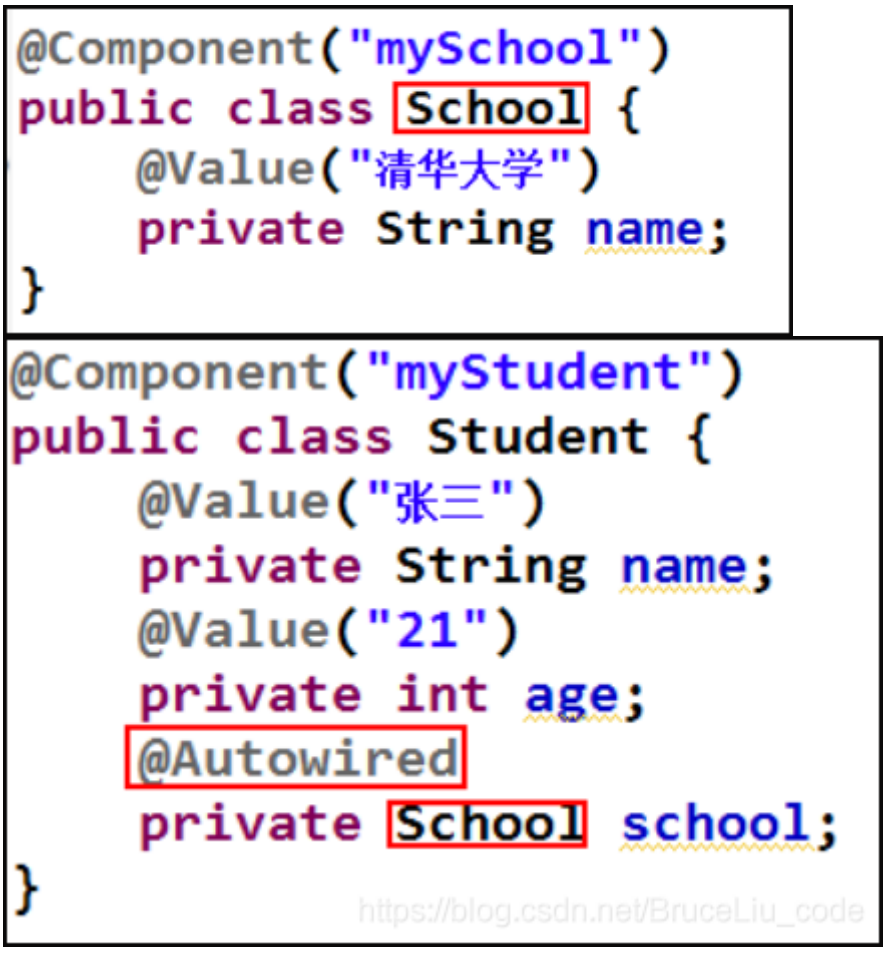
> *将构造函数、引用类型字段、引用类型字段的setter方法或配置方法标记为由Spring的依赖注入工具自动装配。*
>
> *任何给定的bean类只有一个构造函数（最多）可以带有这个注解，指示构造函数在用作Spring bean时自动配置这样的构造函数不必是公共的；*

> *实际注入是通过 BeanPostProcessor 执行的，这反过来意味着您不能使用 @Autowired 将引用注入 BeanPostProcessor 或 BeanFactoryPostProcessor 类型*

用于对类的引用属性、类的引用属性的setter方法、构造方法进行标注，配合对应的注解处理器完成Bean的自动配置工作，默认按照Bean的类型进行装配；
当使用该注解完成属性注入时，类中无需 setter。

3.3.2.1 byType 自动注入@Autowired

举例：



3.3.2.2 byNamme 自动注入@Autowired

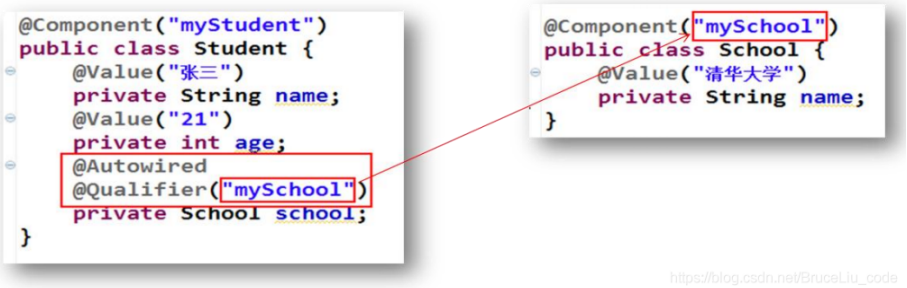
在使用byType方式进行注入时，根据待注入的属性school的类型进行匹配，如果在Spring中有多个Bean的类型和这个school属性的类型同源的话，那么就会根据school属性的名字在Spring的容器中找。

如果根据school属性的名字还找不到，就直接爆异常。

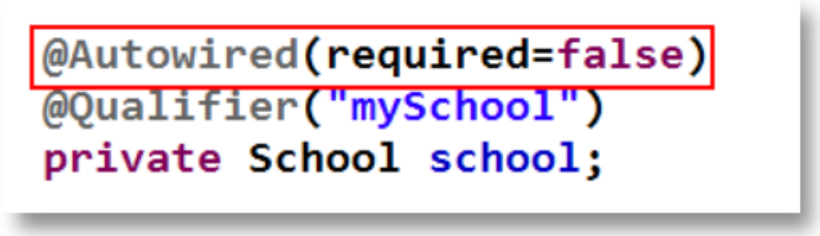
3.3.2.3 byName 自动注入@Autowired + @Qualifier

@Qualifier 注解应该与@Autowired 注解配合使用，会将默认的按照Bean类型装配修改为按照Bean的实例名称装配，Bean的实例名称由@Qualifier 注解的参数指定；
@Qualifier 的 value 属性用于指定要匹配的 Bean 的 id 值。类中无需 setter 方法，也可加到 setter 方法上。

举例：



@Autowired 还有一个属性 required，默认值为 true，表示当匹配失败后，会终止程序运行。若将其值设置为 false，当匹配失败，将被忽略，未匹配的属性值为 null；
如下示例：当注入School类型的Bean复制给给school属性时，如果没有再Spring中找到School类型的Bean，也不会报错，而是给school属性赋值未null。



3.3.3 引用类型属性注入@Resource 自动注入(掌握)

这个注解是**JDK**带的

Spring提供了对 jdk中@Resource注解的支持。@Resource 注解既可以按名称匹配Bean，也可以按类型匹配 Bean。默认是`按名称注入`。使用该注解，要求 JDK 必须是 6 及以上版本。

该注解中有两个重要的属性：name、type；

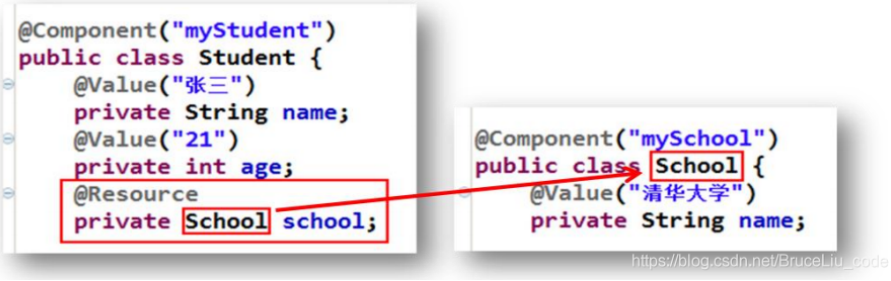
> *1. name：Spring将name属性解析为Bean实例名称*
> *2. type：Spring将type属性解析为Bean实例类型*
>
> *如果指定name属性，则只按照Bean实例名称进行装配；*
> *如果指定type属性，则只按照Bean类型进行装配；*
> *如果都不指定，则先按照Bena实例名称装配，如果不能匹配，再按照Bean类型进行装配;*
>
> *如果以上三种情况中使用指定的方式都没有装配成功，则抛出NoSunchBeanDefinitionException异常。*

@Resource 可在属性上，也可在 setter 方法上。

3.3.3.1 byType 注入引用类型属性

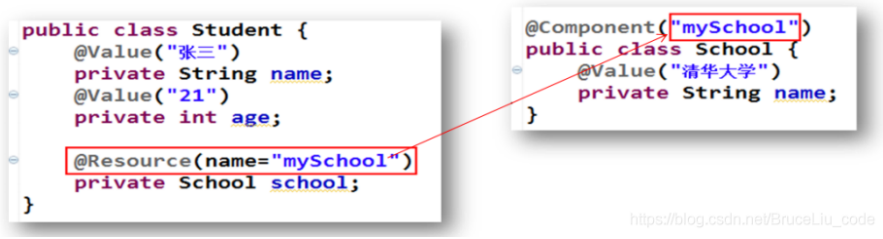
@Resource 注解若不带任何参数，采用默认按名称的方式注入，按名称不能注入 bean，则会按照类型进行 Bean 的匹配注入。

举例：



3.3.3.2 byName 注入引用类型属性

@Resource 注解指定其 name 属性，则 name 的值即为按照名称进行匹配的 Bean 的 id。



3.3.3.3 @Resource和@Autowired注解的区别

Autowired注解在使用时默认按照被注解的属性（记作K）的类型（记作KT）装配，如果按照KT没有匹配到（没有匹配到有两种情况：没有找到、找到多个）则会按照K的名字（记作KN）名字匹配；
Resource注解在使用时（假设不使用name、type属性）默认按照被注解的属性（记作L）的名字（记作LN）装配，如果按照LN没有装配到则会按照L的类型（记作LT）进行装配。

@Autowired + @Qualifier("xxx") === @Resource(name="xxx")

4.注解与XML的对比

注解优点是：

- 方便
- 直观
- 高效（代码少，没有配置文件的书写那么复杂）。

注解缺点是：

- 以硬编码的方式写入到 Java 代码中，修改是需要重新编译代码的。

XML 方式优点是：

- 配置和代码是分离的
- 在 xml 中做修改，无需编译代码，只需重启服务器即可将新的配置加载。

xml 的缺点是：

- 编写麻烦，效率低，大型项目过于复杂。

5. Spring测试模块和Junit整合测试

5.0 本节涉及到的注解们

1. RunWith
2. ContextConfiguration
   在 Spring 3.1 之前，仅支持基于路径的资源位置（通常是 XML 配置文件）。
   从 Spring 3.1 开始，该注解可以选择支持基于路径（XML配置文件）或基于类的资源（配置类）。
   从 Spring 4.0.4 开始，该注解可以选择同时支持基于路径（XML配置文件）和基于类的资源（配置类）。因此，@ContextConfiguration 可用于声明基于路径的资源位置（通过 locations() 或 value() 属性）或带注释的类（通过 classes() 属性）

步骤：

1. 使用Spring的测试框架需要加入以下依赖包：

```
<dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.9</version>
            <scope>test</scope>
</dependency>
<dependency>

 <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version> 3.2.4.RELEASE  </version>
            <scope>provided</scope>
</dependency>
```

2. 创建测试类，在测试类上加入注解@RunWith、@ContextConfiguration

```
import org.junit.runner.RunWith;
import
org.springframework.test.context.ContextConfigura
tion;
import
org.springframework.test.context.junit4.SpringJUn
it4ClassRunner;

//使用junit4进行测试，通过Junit启动Spring框架
@RunWith(SpringJUnit4ClassRunner.class)
//加载配置文件，或加载配置类，如果是配置类那么这个注解中的
元素如右：classes=配置类名.class
@ContextConfiguration(locations=
{"classpath:applicationContext.xml"})
public class BaseJunit4Test{
    //加上上面这两个注解之后就可以在这里使用注解的方式注入
了，而无需手动创建ApplicationContext

}
```