Spring是一个开源框架，为了解决企业应用开发的复杂性而创建的，但是现在已经不止应用于企业应用

是一个轻量级的控制反转（IOC）和面向切面（AOP）的容器框架

1. 从大小与开销两方面而言，Spring都是轻量的。
2. 通过控制反转（IOC）的技术达到松耦合的目的
3. 提供了面向切面编程的丰富支持，允许通过分离应用的业务逻辑与系统级服务进行内聚行的开发
4. 包含并管理应用对象的配置和生命周期，这个意义上是一种容器
5. 将简单的组件配置组合成为复杂的应用，这个意义上是框架

框架的特点：

1. 开源的半成品
2. 封装了特定的处理流程和控制逻辑
3. 成熟的不断改进的软件

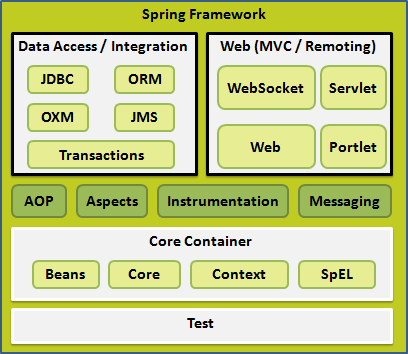
框架和类库的区别

1. 框架一般是封装了逻辑的，高内聚的，类库则是松散的工具组合
2. 框架专注于某一领域，类库则是更加通用

# 一、体系结构

## Spring模块

Spring 是模块化的，允许你挑选和选择适用于你的模块，不必要把剩余部分也引入。



## 核心容器 – Core Container

核心容器由核心，Bean，上下文和表达式语言模块组成，它们的细节如下：

1. 核心模块提供了框架的基本组成部分，包括 IOC(控制反转) 和DI(依赖注入)功能。
2. Bean 模块提供 BeanFactory，它是一个工厂模式的复杂实现。
3. 上下文模块建立在由核心和 Bean 模块提供的坚实基础上，它是访问定义和配置的任何对象的媒介。ApplicationContext 接口是上下文模块的重点。
4. 表达式语言模块在运行时提供了查询和操作一个对象图的强大的表达式语言。

## 数据访问/集成 – Data Access/Integration

数据访问/集成层包括 JDBC，ORM，OXM，JMS 和事务处理模块，它们的细节如下：

1. JDBC 模块提供了删除冗余的 JDBC 相关编码的 JDBC 抽象层。
2. ORM 模块为流行的**对象关系映射** API，包括 JPA，JDO，Hibernate 和 iBatis，提供了集成层。
3. OXM 模块提供了抽象层，它支持对 JAXB，Castor，XMLBeans，JiBX 和 XStream 的对象/XML 映射实现。
4. Java 消息服务 JMS 模块包含生产和消费的信息的功能。
5. 事务模块为实现特殊接口的类及所有的 POJO 支持编程式和声明式事务管理。

## Web层 (MVC / Remoting)

Web 层由 Web，Web-MVC，Web-Socket 和 Web-Portlet 组成，它们的细节如下：

1. **Web 模块提供了基本的面向 web 的集成功能**，例如多个文件上传的功能和使用 servlet 监听器和面向 web 应用程序的上下文来初始化 IoC 容器。
2. Web-MVC 模块包含 Spring 的模型-视图-控制器（MVC），实现了 web 应用程序。
3. Web-Socket 模块为 WebSocket-based 提供了支持，而且在 web 应用程序中提供了客户端和服务器端之间通信的两种方式。
4. Web-Portlet 模块提供了在 portlet 环境中实现 MVC，并且反映了 Web-Servlet 模块的功能。

## 其他

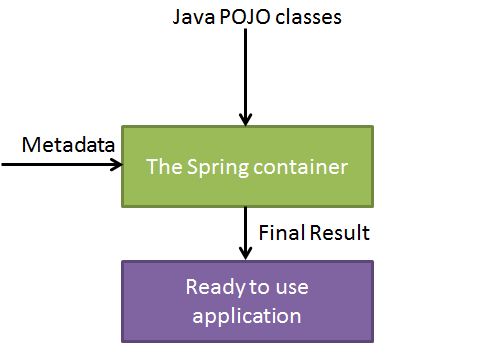
还有其他一些重要的模块，像 AOP，Aspects，Instrumentation，Web 和测试模块，它们的细节如下：

1. AOP 模块提供了面向切面的编程实现，允许你定义方法拦截器和切入点对代码进行干净地解耦，它实现了应该分离的功能。
2. Aspects 模块提供了与 AspectJ 的集成，这是一个功能强大且成熟的面向切面编程（AOP）框架。
3. Instrumentation 模块在一定的应用服务器中提供了类 instrumentation 的支持和类加载器的实现。
4. Messaging 模块为 STOMP 提供了支持作为在应用程序中 WebSocket 子协议的使用。它也支持一个注解编程模型，它是为了选路和处理来自 WebSocket 客户端的 STOMP 信息。
5. 测试模块支持对具有 JUnit 或 TestNG 框架的 Spring 组件的测试。

# 二、IOC容器

## 1. 定义：

Spring 容器是 Spring 框架的核心。容器将创建对象，把它们连接在一起，配置它们，并管理他们的整个生命周期从创建到销毁。**Spring 容器使用依赖注入（DI）来管理组成一个应用程序的组件**。这些对象被称为 Spring Beans。



1. Spring IOC 容器利用Java的 POJO 类和配置元数据来生成完全配置和可执行的系统或应用程序。

（Spring提供了两种容器） -- 接口定义的不同

1. Spring BeanFactory 容器

最简单的容器，**给 DI 提供了基本的支持**，它用

org.springframework.beans.factory.BeanFactory 接口来定义。

BeanFactory 或者相关的接口，如 BeanFactoryAware，InitializingBean，DisposableBean，在 Spring 中仍然存在具有大量的与 Spring 整合的第三方框架的反向兼容性的目的。

1. Spring ApplicationContext 容器

该容器添加了更多的**企业特定的功能**，例如从一个属性文件中解析文本信息的能力，发布应用程序事件给感兴趣的事件监听器的能力。该容器是由

org.springframework.context.ApplicationContext 接口定义。

## 2. SpingBeanFactory 容器

1. 定义： -- 资源占用比较少，移动设备一般会选择这个。

**主要的功能是为依赖注入 （DI） 提供支持**，这个容器接口在

org.springframework.beans.factory.BeanFactor 中被定义。 BeanFactory 和相关的接口，比如，BeanFactoryAware、 DisposableBean、InitializingBean，仍旧保留在 Spring 中，主要目的是向后兼容已经存在的和那些 Spring 整合在一起的第三方框架。

在 Spring 中，有大量对 BeanFactory 接口的实现。其中，最常被使用的是 XmlBeanFactory 类。这个容器从一个 XML 文件中读取配置元数据，由这些元数据来生成一个被配置化的系统或者应用。

XmlBeanFactory已经被遗弃了。

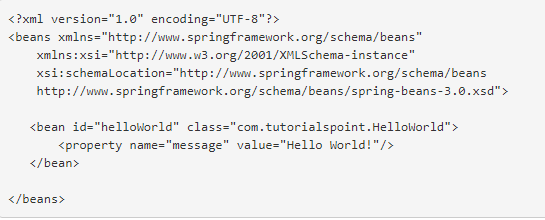
<https://blog.csdn.net/dachengzi159/article/details/81273391> 废除原因

1. 具体代码：

（主程序代码）



（Beans.xml配置）



## 3. Spring ApplicationContext 容器

1. 定义：

**Application Context 是 spring 中较高级的容器**。和 BeanFactory 类似，它可以加载配置文件中定义的 bean，将所有的 bean 集中在一起，当有请求的时候分配 bean。 另外，它增加了企业所需要的功能，比如，从属性文件从解析文本信息和将事件传递给所指定的监听器。这个容器在 org.springframework.context.ApplicationContext interface 接口中定义。

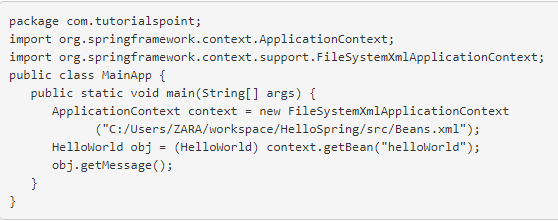
1. ApplicationContext 接口实现 – 3个

FileSystemXmlApplicationContext：该容器从 XML 文件中加载已被定义的 bean。在这里，你需要提供给构造器 XML 文件的完整路径

ClassPathXmlApplicationContext：该容器从 XML 文件中加载已被定义的 bean。在这里，你不需要提供 XML 文件的完整路径，**只需正确配置 CLASSPATH 环境变量即可**，因为，容器会从 CLASSPATH 中搜索 bean 配置文件。

WebXmlApplicationContext：该容器会在一个 web 应用程序的范围内加载在 XML 文件中已被定义的 bean。

1. 具体代码：



# 三、Bean定义

## 1. 定义

构成应用程序、有IOC容器管理。bean 是一个被实例化，组装，并通过 Spring IOC 容器所管理的对象。这些 bean 是由用容器提供的配置元数据创建的。

## 2. bean 定义包含称为配置元数据的信息

用来构成属性

1. 如何创建一个 bean
2. bean 的生命周期的详细信息
3. bean 的依赖关系

## 3. 所有的配置元数据转换成一组构成每个 bean 定义的下列属性



## 4. Spring 配置元数据

**元数据用来设置IOC容器的工作方式，Spring IOC 容器完全由实际编写的配置元数据的格式解耦。**

有下面三个重要的方法把配置元数据提供给 Spring 容器：

1. 基于 XML 的配置文件。
2. 基于注解的配置
3. 基于 Java 的配置

# 四、Bean的作用域

## 作用域属性scope

1. 当在spring声明一个bean时，必须指定作用域。
2. 例如，为了强制 Spring 在每次需要时都产生一个新的 bean 实例，你应该声明 bean 的作用域的属性为 prototype。同理，
3. 如果你想让 Spring 在每次需要时都返回同一个bean实例，你应该声明 bean 的作用域的属性为 singleton。



## singleton 作用域

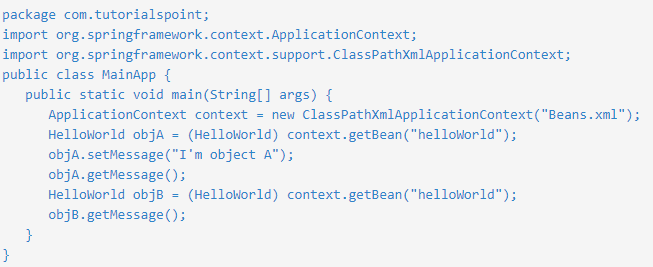
1. 一个容器一个实例

如果作用域设置为 singleton，那么 Spring IoC 容器刚好创建一个由该 bean 定义的对象的实例。**该单一实例将存储在这种单例 bean 的高速缓存中**，以及针对该 bean 的所有后续的请求和引用都返回缓存对象。

1. 默认作用域是始终是 singleton，但是当仅仅需要 bean 的一个实例时，你可以在 bean 的配置文件中设置作用域的属性为 singleton，

如：<bean id="..." class="..." scope="singleton"></bean>

1. 测试代码：



## prototype 作用域

1. 相同的类，一个容器，一次请求，一个实例
2. 如果作用域设置为 prototype，那么每次特定的 bean 发出请求时 Spring IoC 容器就创建对象的新的 Bean 实例。
3. 如：<bean id="helloWorld" class="com.tutorialspoint.HelloWorld" scope="prototype">

</bean>

# 五、Bean生命周期

## 1. 生命周期简介

1. 当一个 bean 被实例化时，可能需要执行一些初始化使它转换成可用状态。同样，当 bean不再需要，并且从容器中移除时，可能需要做一些清除工作。主要关注初始化和销毁时的回调方法。
2. **为了定义安装和拆卸一个 bean，我们只要声明带有（XML中的属性） init-method 或 destroy-method 参数的。**init-method 属性指定一个方法，在实例化 bean 时，立即调用该方法。同样，destroy-method 指定一个方法，只有从容器中移除 bean 之后，才能调用该方法。

## 2. 初始化回调

1. 基于XML配置元数据的情况下，**利用init-method属性指定void无参数方法**：

<bean id="exampleBean" class="examples.ExampleBean" init-method="init"/>

类的定义：

public class ExampleBean {

public void init() {

// do some initialization work

}

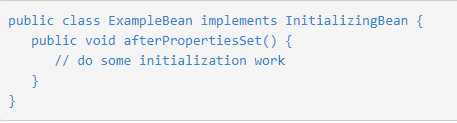
}

1. 基于接口的初始化方法 -- **实现该接口InitializingBean**

org.springframework.beans.factory.InitializingBean 接口指定一个单一的方法：

void afterPropertiesSet() throws Exception;

**具体代码**：



## 3. 销毁时回调

1. 基于XML配置元数据的情况下，**利用destroy-method 属性指定 void 无参数方法**：

<bean id="exampleBean" class="examples.ExampleBean" destroy-method="destroy"/>

类的定义：

public class ExampleBean {

public void destroy() {

// do some destruction work

}

}

方法要写在被初始化实体类中。

1. 基于接口的初始化方法 -- 实现接口DisposableBean

org.springframework.beans.factory.DisposableBean 接口指定一个单一的方法：

void destroy() throws Exception;

**具体代码：**

public class ExampleBean implements DisposableBean {

public void destroy() {

// do some destruction work

}

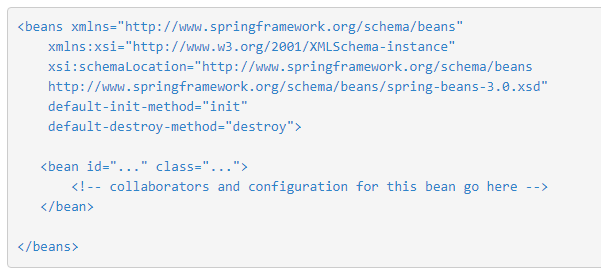
}

1. XML中destory bean的方法



## 默认的初始化和销毁方法

如果你有太多具有相同名称的初始化或者销毁方法的 Bean，那么你不需要在每一个 bean 上声明初始化方法和销毁方法。***框架使用元素中的 default-init-method 和 default-destroy-method*** 属性提供了灵活地配置这种情况



# 六、Spring – 后置处理器(不重要)

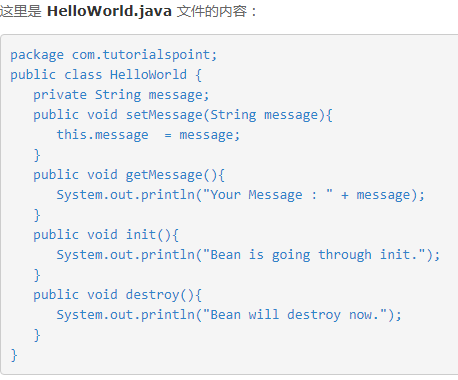
## 1. 简介

1. 接口BeanPostProcessor，外置的回调函数（初始化之前和初始化之后）
2. BeanPostProcessor 接口定义回调方法，可以实现该方法来提供自己的实例化逻辑，依赖解析逻辑等，来完成实例化，配置和初始化一个bean之后实现一些自定义逻辑回调方法。
3. 可以配置多个 BeanPostProcessor接口，通过设置 BeanPostProcessor 实现的 Ordered 接口提供的 order 属性来控制这些 BeanPostProcessor 接口的执行顺序。
4. **后置处理器对每个实例化的bean都会起作用，而且方法里必须返回bean，不能返回其他值。**

## 2. 工作原理

(1) ApplicationContext 会自动检测由 BeanPostProcessor 接口的实现定义的 bean，注册这些 bean 为后置处理器，然后通过在容器中创建 bean，在适当的时候调用它。

1. 具体代码：



实现BeanPostProcessor的类：



（Bean.xml配置文件）



# 七、Bean定义继承

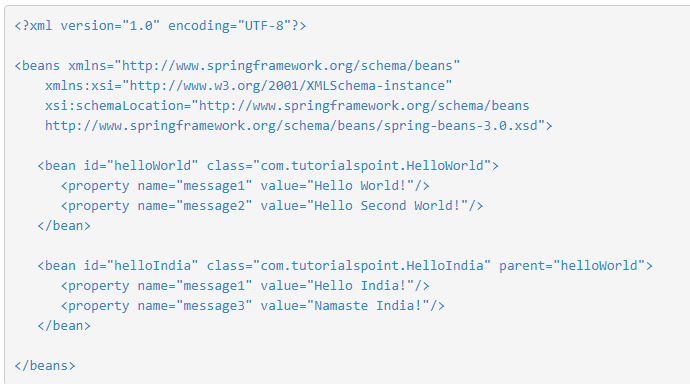
## 1. 简介

**在java中，bean定义继承不形成java中的继承，不存在实际继承**。

Spring Bean 定义的继承与 Java 类的继承无关，但是继承的概念是一样的。你可以定义一个父 bean 的定义作为模板和其他子 bean 就可以从父 bean 中继承所需的配置。

## 2. 语法格式parent属性

当你使用基于 XML 的配置元数据时，通过使用父属性，指定父 bean 作为该属性的值来表明子 bean 的定义。



## 3. Bean定义模板

abstract=true 虚拟模板，不存在实际类



# 八、依赖注入

**通过实例化一个类来实现该类对其他类的引用**

## 构造注入 – 基于构造函数

1. 当容器调用带有一组参数的类构造函数时，基于构造函数的 DI 就完成了，其中每个参数代表一个对其他类的依赖。
2. 具体实例：



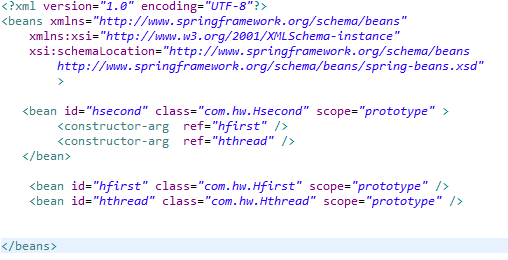




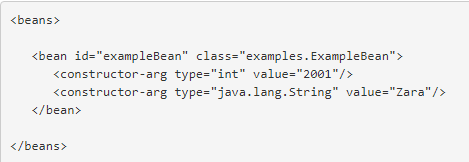


1. 当依赖注入有多个参数

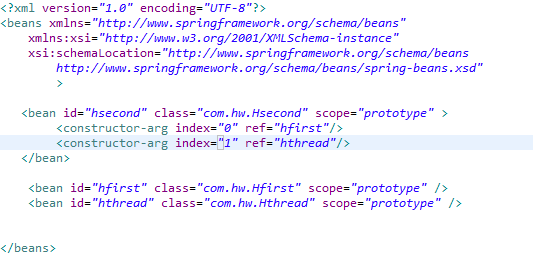
bean中排序的位置 -- 决定初始化的顺序，与构造函数中位置无关



利用type属性来匹配，简单的值，配置构造函数中基本类型的值。



利用index属性类匹配 – index序号和参数顺序要对应，index要按顺序赋值



## 设值注入 – 基于设值函数（setter）

1. 前提条件 – **优先级比较低**

当容器调用一个无参的构造函数或一个无参的静态 factory 方法来初始化你的 bean 后，通过容器在你的 bean 上调用设值函数，基于设值函数的 DI 就完成了。

1. 具体代码：

需要定义一个注入类的变量，还要设值set和get方法。

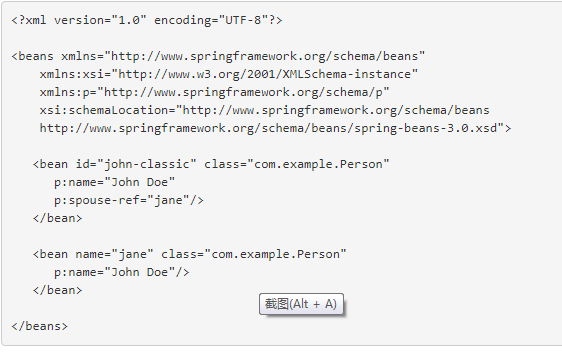
set方法必须完整 this.%%=%%



1. bean.xml 利用<property name=”” class=””>来设值



1. 使用 p-namespace 实现 XML 配置 – 直接写在bean的属性中



1. 注入顺序

构造Paper类

构造PaperExt类

调用Paper.setPaperExt()函数，将PaperExt类注入

调用show()方法

# 九、注入内部Beans

1. inner beans 是在其他 bean 的范围内定义的 bean。因此在 或 元素内 元素被称为内部bean。

特点是：内部bean仅注入一次，不能被其他bean引用。



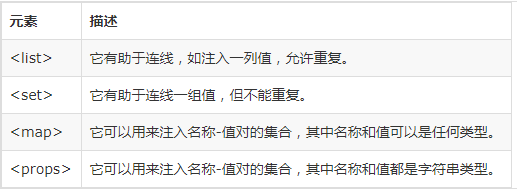
1. 具体代码： -- **和设值注入一样使用set函数**



# 十、注入集合

传递多个值，如 Java Collection 类型 List、Set、Map 和 Properties

1. Spring 提供了四种类型的集合的配置元素



Beans.xml具体配置 – 通过set方法，<property>





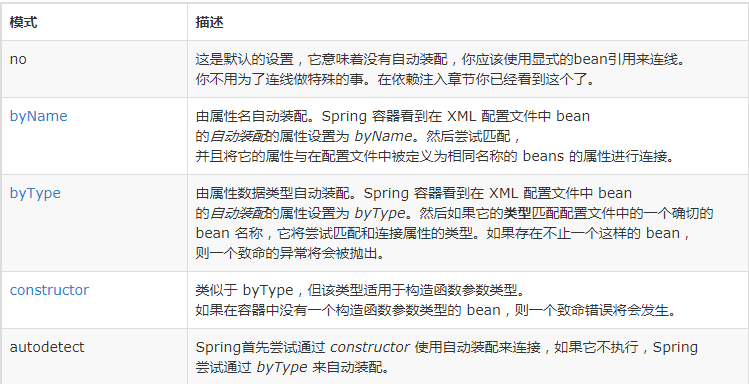
# 十一、依赖注入Bean自动装配

类属性根据bean名称自动注入，在配置文件中省略这些声明。

## 自动装配模式

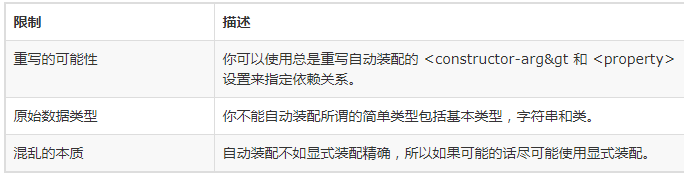
可以使用 元素的 autowire 属性为一个 bean 定义指定自动装配模式。

可以使用 byType 或者 constructor 自动装配模式来连接数组和其他类型的集合。



## 自动装配的局限性

**当自动装配始终在同一个项目中使用的效果最好**。如果通常不使用自动装配，它可能会使开发人员混淆的使用它来连接只有一个或两个 bean 定义。不过，自动装配可以显著减少需要指定的属性或构造器参数，但你应该在使用它们之前考虑到自动装配的局限性和缺点。



## 根据属性名称

1. 简介

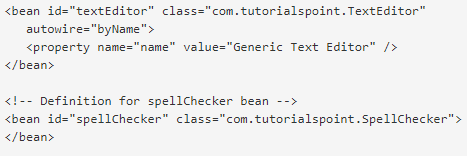
当属性为空时，**自动寻找与属性名称相匹配的bean进行注入**。byName由属性名称指定自动装配，**只能设值注入**。

Spring 容器看作 beans在 XML 配置文件中 **beans 的 auto-wire 属性设置为 byName**。然后**尝试将属性与配置文件中定义为相同名称的 beans 进行匹配和连接**。如果找到匹配项，它将注入这些 beans，否则，它将抛出异常。

1. 具体代码：在main函数中getbean类TextEditor的bean



Bean.xml配置自动匹配

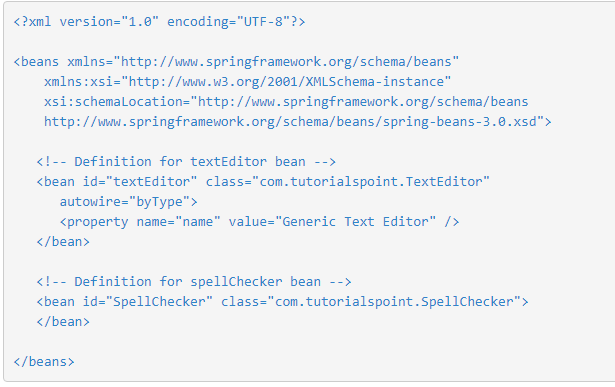


## 根据属性类型

1. 简介

**Spring 容器看作 beans，在 XML 配置文件中 beans 的 autowire 属性设置为 byType，只能用设值注入。**然后，如果它的 type 恰好与配置文件中 beans 名称中的一个相匹配，它将尝试匹配和连接它的属性。如果找到匹配项，将注入这些 beans，否则，它将抛出异常。

1. Bean.xml配置文件



## 有构造函数自动装配

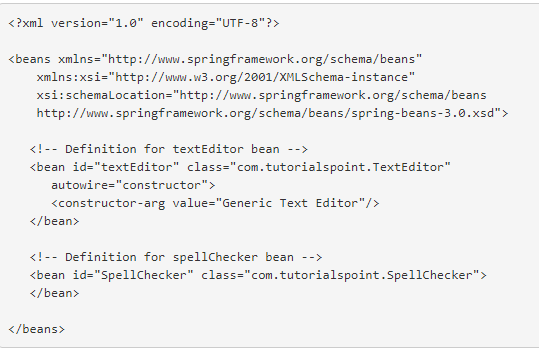
1. 简介

**仍然可以使用 <constructor-arg> 标签连接其余属性在 XML 配置文件中 beans 的 autowire 属性设置为 constructor。**然后，它尝试把它的构造函数的参数与配置文件中 beans 名称中的一个进行匹配和连线。如果找到匹配项，它会注入这些 bean，否则，它会抛出异常。

1. 具体代码：

属性autowire=”constructor”

标签<constructor-arg>不用写属性名称，bean中的排序和构造参数中一一对应。



# 十二、基于注解的依赖注入配置

## 开启注解

**使用注解配置依赖注入，取代用XML文件来配置bean之间的联系**

如果想在Spring中使用注解，要先在配置文件中启用注解。

**注解是用来省略xml文件中的注入，不用写属性**。



<?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">

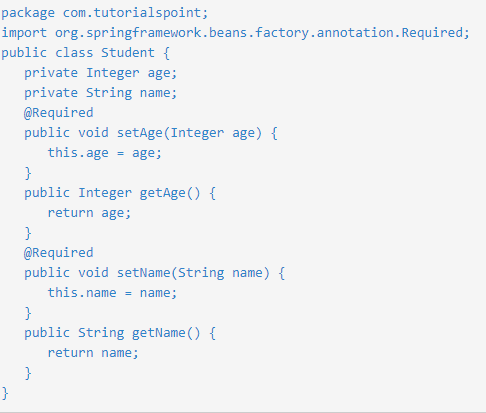
<context:annotation-config/>

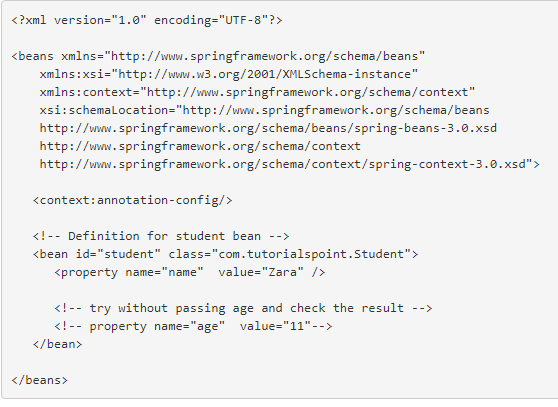
## 重要的注解



* 1. @Required 注解

**@Required 注解应用于 bean 属性的 setter 方法**，**作用是检查属性是否设值**。如果属性没有在xml中设值，容器就会抛出一个 BeanInitializationException 异常。





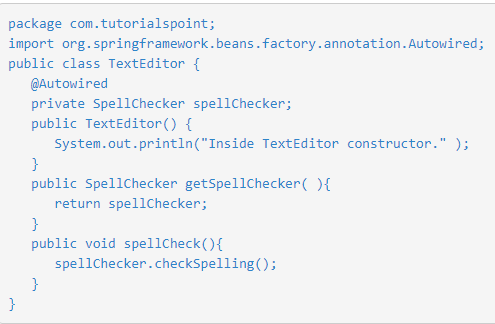
* 1. @Autowired 注解 -- 自动装配，自动匹配属性

@Autowired 注释可以在 setter 方法中被用于自动装配 bean，可以用在setter方法，非setter方法，构造函数和属性。

1. 应用在setter方法 -- 是byType方式去自动装配bean元素



1. 应用在属性中 -- byType方式



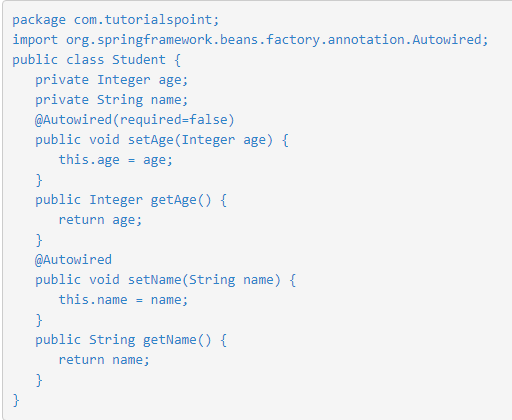
1. 应用在构造函数



1. @Autowired 的（required=false）选项

如下： age可以不赋值，不初始化，为null；

name必须赋值初始化，否则运行会报错。

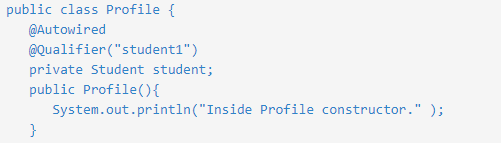


## Spring @Qualifier 注解

1. 作用：

@Autowired是利用byType进行自动装配的，当一个属性出现两个相同类型的bean时，自动装配会出现混乱。利用@Qualifier(id)进行再次筛选，指定相匹配的bean进行装配。

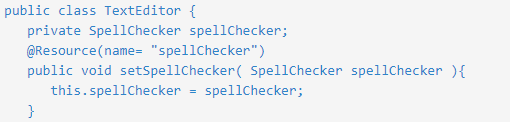
1. 具体代码：



1. Spring JSR-250 注释
2. @PostConstruct 和 @PreDestroy 注释：用来代替bean属性中的init-method和destroy-method的属性，注解放在实体类中的初始化和销毁方法上面。
3. @Resource 注释：

其他注解都是通过byType来自动装配bean。可以利用@Resource（name=””）来通过byName方式自动装配bean。

一般放在属性和set方法上面。



# 十三、基于JAVA的配置

## @Configration 和 @Bean 注解

可以把HelloWorldConfig类当做XML看待



1. @Configuration 的注解类表示这个类可以使用 Spring IoC 容器作为 bean 定义的来源。@Bean 注解告诉 Spring，一个带有 @Bean 的注解方法将返回一个对象，该对象应该被注册为在 Spring 应用程序上下文中的 bean。
2. 带有 @Bean 注解的方法名称作为 bean 的 ID，它创建并返回实际的 bean。你的配置类可以声明多个 @Bean。
3. 相当于：

<bean id=”helloWorld” class=”com.tutorialspoint.HelloWorldConfig”></bean>

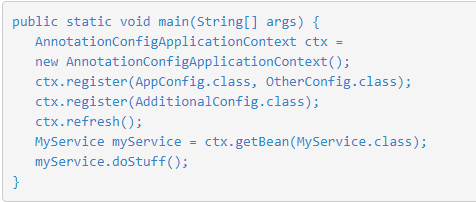
## 实际调用方式

1. ApplicationContext ctx =

new AnnotationConfigApplicationContext(HelloWorldConfig.class);



1. 这个方法可以用来加载各种配置类：



## 3. 注入Bean的依赖性

(1) FOO类中需要定义bar的变量。

(2) FOO类需要重载构造方法。



## 4. Import注解

@import 注解允许从另一个配置类中加载 @Bean 定义。相当继承

这个的意义就是，一个配置类继承上一个配置类。

## 生命周期回调

1. @Bean(initMethod=””,destroyMethod=””) – 用于定义初始化和销毁方法



1. @Bean @Scope("”) -- 用于定义范围

