# Spring IoC的基本概念

Ioc（控制反转）是Spring框架的基础，也是Spring框架的核心。

1. 控制反转（Inversion of Control, IoC）是Spring框架的核心，用来消减计算机程序

的耦合问题。依赖注入（Dependency Injection，DI）是IoC的另外一种说法，只是 从不同的角度描述相同的概念。

1. 当某个Java对象（调用者）需要调用另一个Java对象（被调用者），例如A对象的

方法中需要调用B对象的方法，在传统编程模式下，调用者通常会采用“new 被调 用者”的代码方式来创建被调用者对象，即在要调用的方法中声明B对象，然后通 过B对象调用其方法。但这种方式会增加调用者和被调用者之间的耦合性，不利于 后期代码的升级与维护。当Spring框架出现后，对象的实例不再由调用者通过“ne w被调用者”来创建，而是由Spring容器来创建。Spring容器会负责控制程序之间 的关系，而不是由调用者的程序代码直接控制。这样，控制权由调用者转移到Spri ng容器，控制权发生了反转，这就是Spring的控制反转。

1. 综上所述，控制反转是一种通过描述（在Spring中可以是XML或注解）并通过第三

方产生或获取特定对象的方法。在Spring中实现控制反转的是Spring IoC容器，实

现方法是依赖注入。

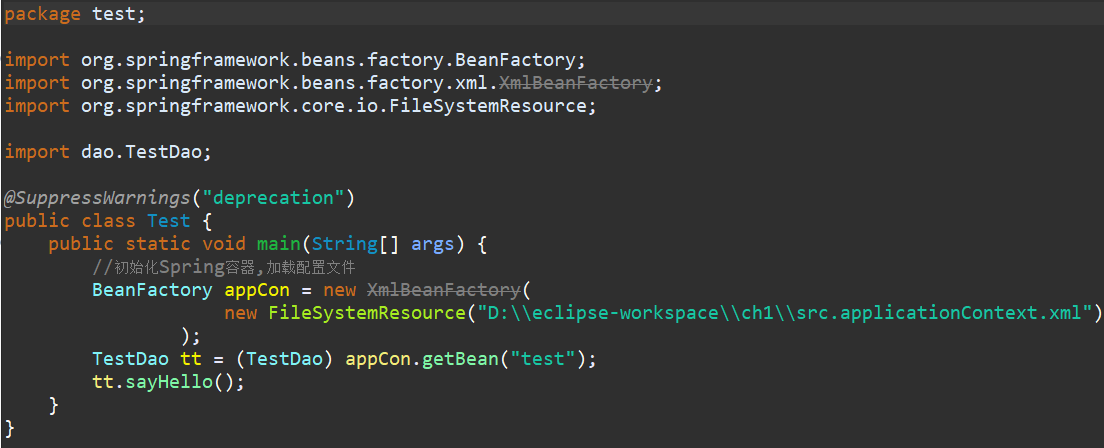
# 实现Spring IoC容器

实现控制反转的是Spring IoC容器。而Spring IoC容器的设计主要是基于BeanFactory 和ApplicationContext这两个接口，所以可以间接的理解，实现控制反转（IoC）依靠的 是BeanFactory和ApplicationContext这两个接口。

## BeanFactory

BeanFactory由org.springframework.beans.factory.BeanFactory接口定义，它提供了完整 的IoC服务支持，是一个管理Bean的工厂，主要负责初始化各种Bean。

BeanFactory 接口有多个实现类，其中比较常用的是org.springframework.beans.factory. xml.XmlBeanFactory类，该类会根据XML配置文件中的定义来装配Bean。在创建 Bea nFactory实例时需要提供XML文件的绝对路径。例如可以将上一章节的ch1应用中mai n方法的代码修改如下：



Spring并不推荐使用BeanFactory实例加载Spring配置文件，在实际开发中也并不多见， 只需了解即可。

## ApplicationContext

ApplicationContext接口是BeanFactory接口的子接口，也称为应用上下文，由 org.spri ngframework.context.ApplicationContext接口定义。ApplicationContext接口除了包含Bea nFactory的所有功能以外，还添加了国际化、资源访问、事件传播等内容的支持。创建 ApplicationContext接口的实例通常有以下3种方法：

1. **通过ClassPathXmlApplicationContext创建**

ClassPathXmlApplicationContext将从类路径目录（src根目录）中寻找指定的X ML配置文件。例如将上一章节中ch1应用中main方法的代码：



1. **通过FilsSystemXmlApplicationContext创建**

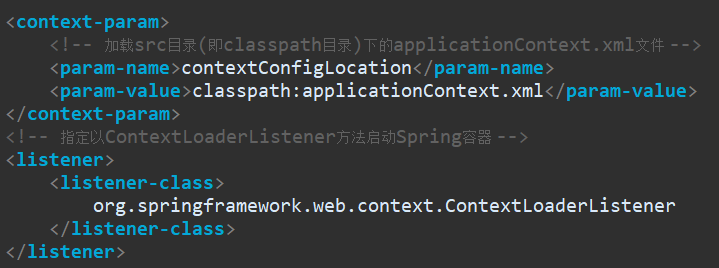
FileSystemXmlApplicationContext使用绝对路径寻找src根目录中的XML配置文件， 找到并装载完成ApplicationContext的实例化工作。例如可以将上一章节的ch1应 用中main方法的代码修改如下：



采用绝对路径的加载方式会使得程序的灵活性变差，一般不推荐使用。因此，通常在Spring的Java应用中采取通过ClassPathXmlApplicationContext类来实例化ApplicationContext容器，而在Web应用中，通过Web服务器完成ApplicationContext容器的实例化。

**3、通过Web服务器实例化ApplicationContext容器**

通过Web服务器实例化ApplicationContext容器时，一般使用基于org.springframe work.web.context.ContextLoaderListener的实现方式，需要将spring-web-5.0.2.RELE ASE.jar复制到WEB-INF/lib目录中，然后在web.xml中添加以下代码：



在这三个ApplicationContext实例化方式中：通过ClassPathXmlApplicationContext类 实 例ApplicationContext容器的方式一般在Java应用中采取；而在Web应用中， ApplicationContext容器的实例化的工作将交给Web服务器来完成。

# 依赖注入的类型

## 什么是依赖注入

1. 依赖注入（DI）是IoC的另外一种说法，只是从不同的角度描述相同的概念。
2. 从Spring容器角度来看，Spring容器负责将被依赖对象赋值给调用者的属性成员变

量，相当于为调用者的属性成员变量进行实例化，这就是Spring的依赖注入。

1. 在Spring中实现IoC容器的方法是依赖注入，依赖注入的作用是在使用Spring框架 创建对象时动态地将其所依赖的对象注入Bean组件中。
2. Spring框架的依赖注入有两种实现方式，一种是使用构造方法注入，另一种是使用

属性的setter方法注入。不管是使用哪种方式，它们实现的原理都是采用Java的反

射机制。

## 使用构造方法注入

使用构造方法注入的语法为：

**<bean id=”bean名称” class=”bean类型”>**

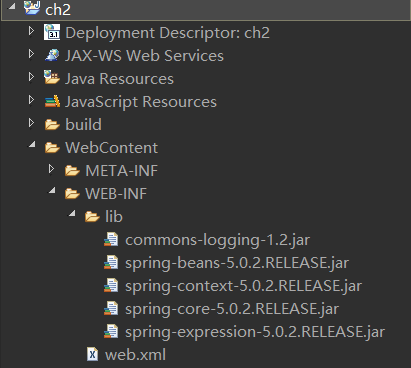
**<constructor-arg index=”参数位置” ref|value=”beanID或常量”>**

**……**

**</bean>**

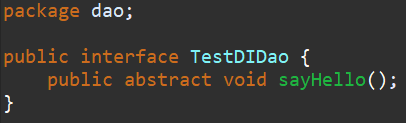
1. **在Eclipse中创建名为ch2的Web应用**

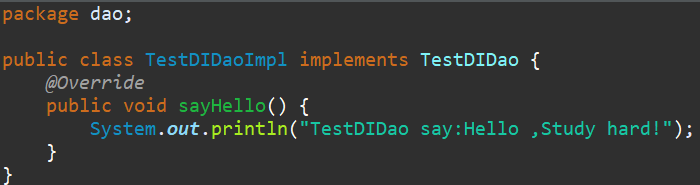
在Eclipse中创建名为ch2的Web应用，并导入相关jar包。



1. **创建dao包**

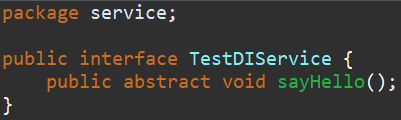
在ch2应用中创建dao包，并在该包中创建TestDIDao接口和接口的实现类TestDIDaoImpl。创建dao包的目的是在service中使用构造方法依赖注入TestDIDao接口对象。代码分别如下：

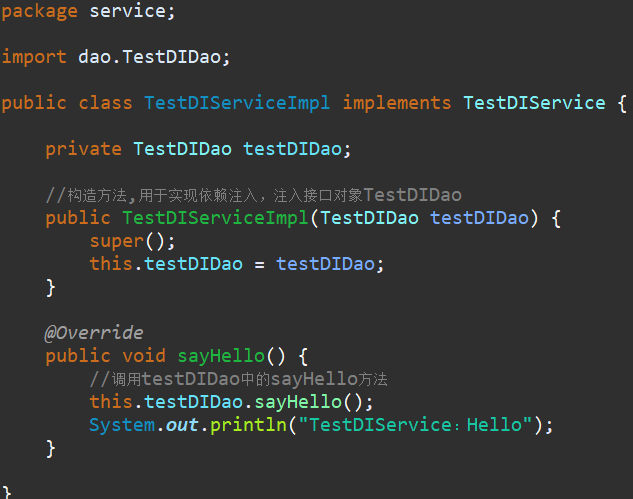




1. **创建service包**

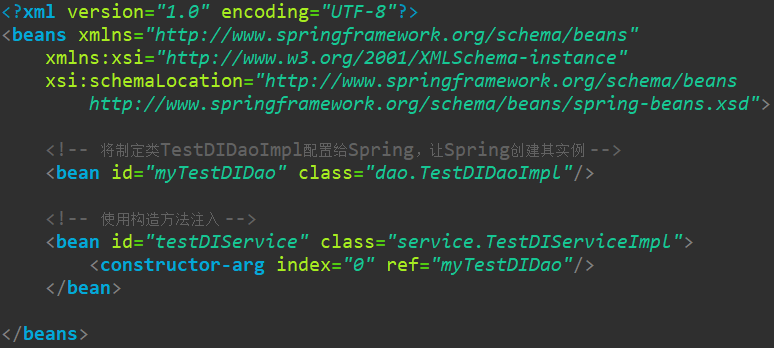
在ch2应用中创建service包，并在该包中创建TestDIService接口和接口的实现类TestDIServiceImpl。在TestDIServiceImpl中使用构造方法依赖注入TestDIDao接口对象。代码分别如下：





1. **编写Spring配置文件**

在src根目录下创建Spring配置文件applicationContext.xml。在配置文件中首先将dao.TestDIDaoImpl类托管给Spring，让Spring创建其对象，然后将service.TestDIServiceImpl类托管给Spring，让Spring创建对象，同时给构造方法传递实参，实现构造方法注入。代码如下：



在配置文件中，constructor-arg标签用于指定构造方法的参数，index用于定义参数 的位置，为0表示是构造方法的第1个参数，ref指定某个实例的引用，如果参数 是常量值，则使用value代替ref。

1. **编写测试程序**

在ch2应用中创建test包，并在该包中创建测试类TestDI。代码如下：



## 使用属性的setter方法注入

使用setter方法注入是Spring框架中最主流的注入方式，它利用Java Bean规范所定义 的setter方法来完成注入，灵活且可读性高。其语法为：

**<bean id=”bean名称” class=”bean类型”>**

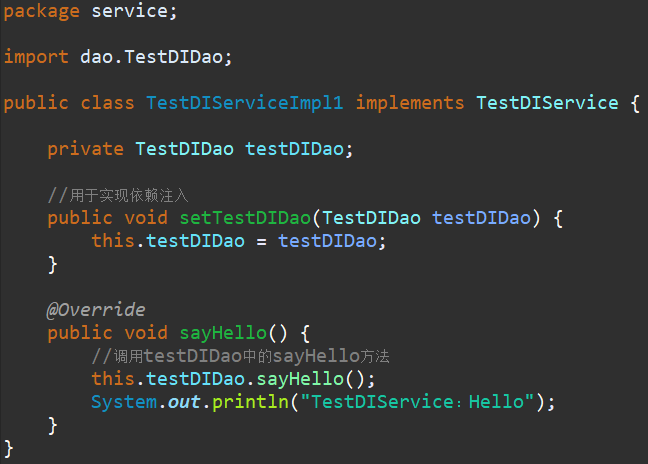
**<property name=”属性名” ref=”beanID”/>**

**……**

**</bean>**

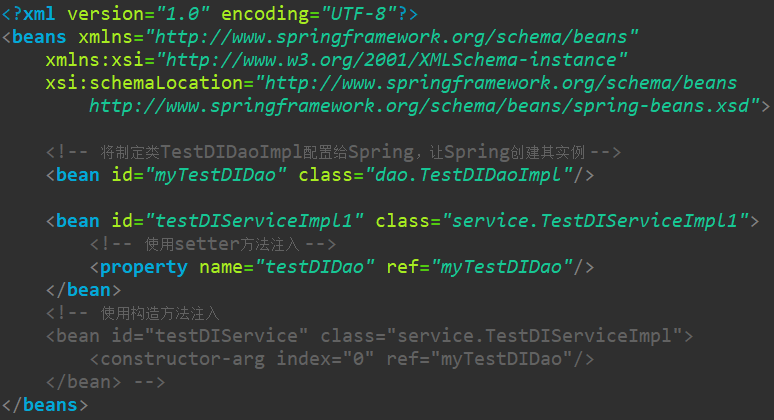
1. **在上一节的ch2应用的中，创建接口实现类TestDIServiceImpl1**

在service包中创建TestDIService接口的实现类TestDIServiceImpl1，在TestDIServiceImpl1中定义类型为TestDIDao类型的对象，并为其提供setter方法，目的是为注入做准备。代码如下：



1. **在Spring配置文件中实现setter注入**

在Spring配置文件中配置将TestDIServiceImpl1类托管给Spring，让Spring创建其对象，同时调用TestDIServiceImpl1类的setter方法完成依赖注入。代码如下：



1. **进行测试**

编写测试，代码如下：

