07 别整虚的！揭开Spring IoC、DI的神秘面纱

更新时间：2020-08-10 14:43:39

> 没有引发任何行动的思想都不是思想，而是梦想。—— 马丁

背景

说到 Spring 框架，人们往往大谈特谈一些似乎很高深的东西，比如依赖注入DI，控制反转 IoC，面向切面编程 AOP 等等。那么 DI 和 IoC 究竟分别是什么？它们又是怎么来的？背后隐藏了什么东西呢？

“不要给我们打电话，我们会给你打电话（don't call us, we'll call you）”这是著名的好莱坞原则。在好莱坞，把简历递交给演艺公司后就只有回家等待。由演艺公司对整个娱乐项的完全控制，演员只能被动式的接受公司的差使,在需要的环节中，完成自己的演出。

IoC 是 Inversion of Control 的简称，IoC 的原理就是基于好莱坞原则，所有的组件都是被动的（Passive），所有的组件初始化和调用都由**容器**负责。

IoC（控制反转）： 全称为：Inverse of Control。从字面上理解就是控制反转了，将对在自身对象中的一个内置对象的控制反转，反转后不再由自己本身的对象进行控制这个内置对象的创建，而是由第三方系统去控制这个内置对象的创建。

DI（依赖注入）： 全称为 Dependency Injection，意思自身对象中的内置对象是通过注入的方式进行创建。

那么 **IoC** 和 **DI** 这两者又是什么关系呢？

Spring IoC 容器

IoC 就是一种软件设计思想，DI 是这种软件设计思想的一个实现。而 Spring 中的核心机制就是 DI。为什么是 DI 呢？这有一段典故：

> "*The question is, what aspect of control are [they] inverting?*" Martin Fowler posed this question about Inversion of Control (IoC) on his site in 2004. Fowler suggested renaming the principle to make it more self-explanatory and came up with *Dependency Injection*.

Bean 是 spring 的原材料，所以我们先来讲一下怎么理解 Bean。

Bean 在 Spring 中是可以复用的组件，Spring 帮忙管理这些组件，如果想要在 Spring 中使用组件时，无需实例化，可以直接使用。

一个简单的比喻：Spring 是一个外包公司即乙方，Bean 是这个公司的基层员工，甲方将项目外包给乙方，而不用关心乙方如何招聘员工，在项目紧急需要外包人员时直接使用即可。

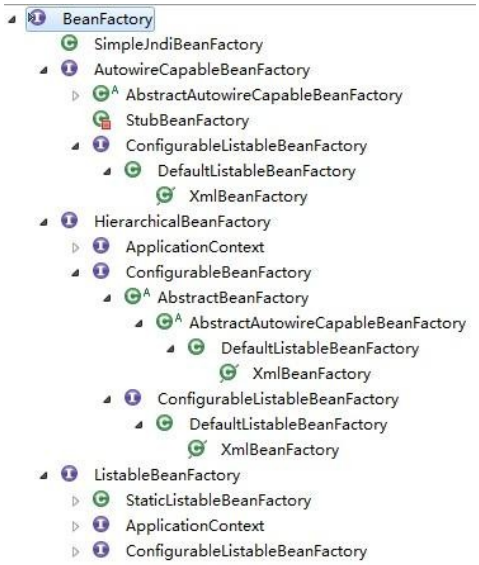
Spring 的本质就是一个 Bean 工厂（BeanFactory）或者说 Bean 容器，它按照我们的要求，生产我们需要的各种各样的 Bean，提供给我们使用。只是在生产 Bean 的过程中，需要解决 Bean 之间的依赖问题，才引入了依赖注入（DI）这种技术。也就是说依赖注入是 BeanFactory 生产 Bean 时为了解决 Bean 之间的依赖的一种技术而已。

具体的控制反转是如何操作的，Spring 启动的时候都做了什么事情，怎么将 Bean 放入到容器，在使用的时候又是如何获取 Bean 的（拿出对应的方法进行讲解）

BeanFactory 的形成

BeanFactory 是访问 Bean 容器的根接口，它是一个 Bean 容器的基本客户端视图。

先让我们看看 BeanFactory 的前生后世吧！



BeanFactory 有四个重要的子接口：

SimpleJndiBeanFactory 是 spring beanFactory 接口的基于 jndi 的简单实现。不支持枚举 Bean 定义，故不需要实现 ListableBeanFactory 接口。这个 Bean 工厂可以解析制定名称的 jndi 名称，在 J2EE 应用中，jndi 名称的命名空间为"java:/comp/env/"。这个 Bean 工厂主要和 Spring 的 CommonAnnotationBeanPostProcessor 联合使用；

ListableBeanFactory 是 BeanFactory 接口的扩展接口，它可以枚举所有的 Bean 实例，而不是客户端通过名称一个一个的查询得出所有的实例。要预加载所有的 Bean 定义的 BeanFactory 可以实现这个接口来。该接口定义了访问容器中 Bean 基本信息的若干方法，如查看 Bean 的个数、获取某一类型 Bean 的配置名、查看容器中是否包括某一 Bean 等方法；

HierarchicalBeanFactory 是一个 Beanfactory 子接口实现，可以作为层次结构的一部分。相对应的 bean Factory 方法 setParentBeanFactory 允许在一个可配置 BeanFactory 中设置它们的父 Beanfactory；

AutowireCapableBeanFactory 是 BeanFactory 接口的扩展实现，假如它们想要对已经存在的 Bean 暴露它的功能，实现它就能实现自动装配功能。定义了将容器中的 Bean 按某种规则（如按名字匹配、按类型匹配等）进行自动装配的方法；

ConfigurableBeanFactory 是一个配置接口，大部分 BeanFactory 实现了这个接口。这个接口提供了对一个 BeanFactory 进行配置的便利方法，加上 BeanFactory 接口的客户端方法。增强了 IoC 容器的可定制性，它定义了设置类装载器、属性编辑器、容器初始化后置处理器等方法；

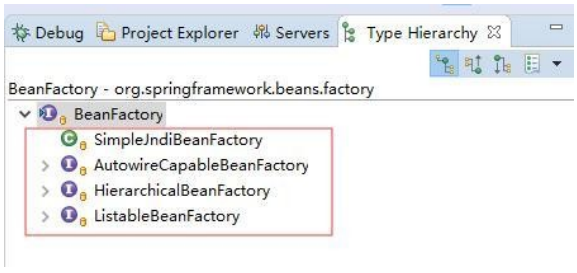
ConfigurableListableBeanFactory 它同时继承了 ListableBeanFactory，AutowireCapableBeanFactory 和 ConfigurableBeanFactory，提供了对 Bean 定义的分析和修改的便利方法，同时也提供了对单例的预实例化。

BeanFactory的形成可不是一蹴而就的，它的形成过程也经历了一番波折。

形成过程

1. 出生

BeanFactory 有四个子接口，添加了四种不同的能力：



分别是：

SimpleJndiBeanFactory：支持 jndi；

AutowireCapableBeanFactory：支持自动装配；

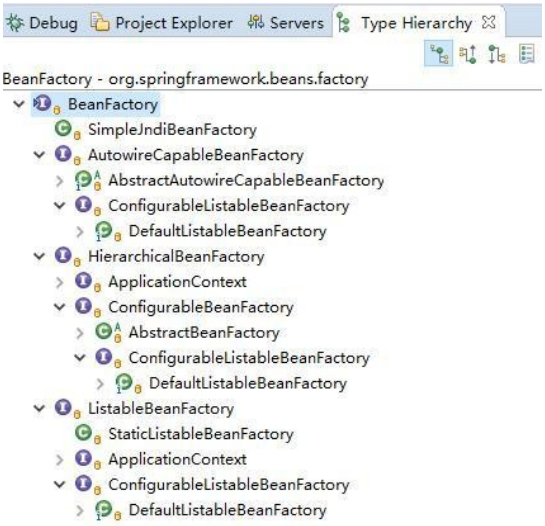
HierarchicalBeanFactory：支持层次结构，ConfigurableListableBeanFactory 实现了 HierarchicalBeanFactory，提供了可配置功能；

ListableBeanFactory：支持枚举。

2. 长大成人

Bean 经过两次进化，到 DefaultListableBeanFactory，完善了 Bean 容器的功能。

DefaultListableBeanFactory 实现 AbstractAutowireCapableBeanFactory
，ConfigurableListableBeanFactory, BeanDefinitionRegistry, Serializable 等多个接口。



此时，DefaultListableBeanFactory 已经成人，可以独立参加工作了！

javaBean：

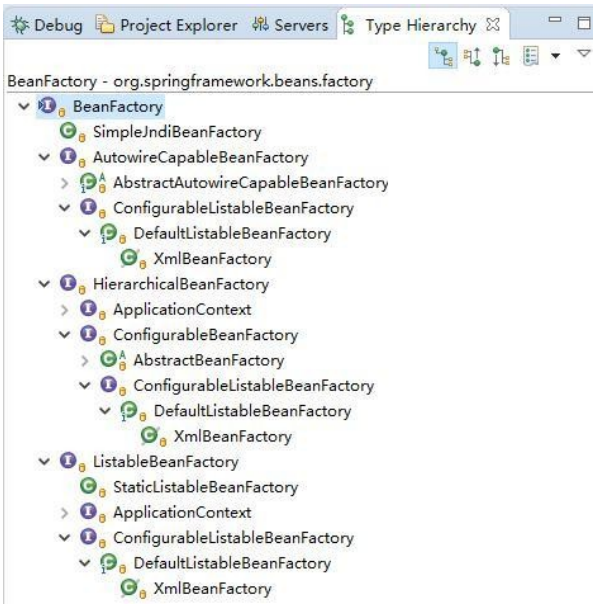
```
package org.springframework.beans;

public class HelloBean {
private String helloWorld;

/**
 * @return the helloWorld
 */
public String getHelloWorld() {
    return helloWorld;
}

/**
 * @param helloWorld the helloWorld to set
 */
public void setHelloWorld(String helloWorld) {
    this.helloWorld = helloWorld;
}
}
```

DefaultListableBeanFactory 测试类：

```java
package org.springframework.beans;

import org.springframework.beans.factory.BeanFactory;

public class BeanFactoryTest {

    public static void main(String[] args) {
        // 设置属性

        MutablePropertyValues properties = new MutablePropertyValues();

        properties.addPropertyValue("helloWorld", "Hello!david!");

        // 设置Bean定义

        RootBeanDefinition definition = new RootBeanDefinition(HelloBean.class,null, properties);

        // 注册Bean定义

        BeanDefinitionRegistry reg = new DefaultListableBeanFactory();

        reg.registerBeanDefinition("helloBean", definition);

        BeanFactory factory = (BeanFactory) reg;

        HelloBean hello = (HelloBean) factory.getBean("helloBean");

        System.out.println(hello.getHelloWorld());

    }
}
```

3. 发展

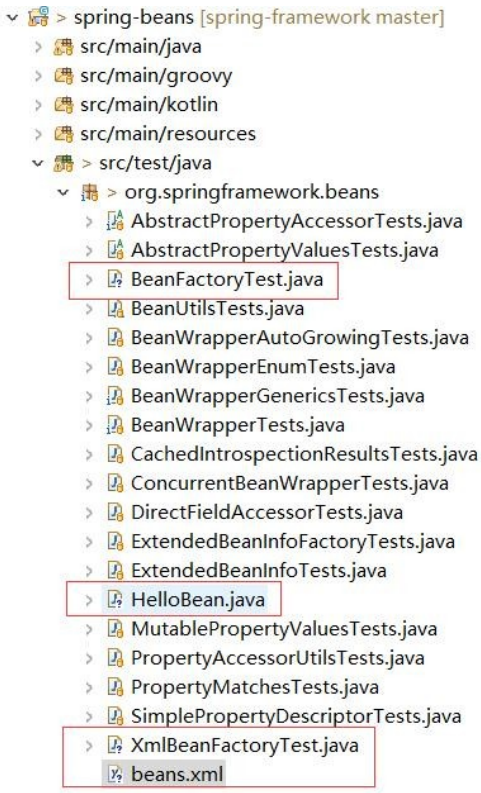
XmlBeanFactory 通过从 xml 文件中读取 Bean 的定义和依赖。



xmlBeanFactory 可以通过 xml 配置方式，减少我们的代码。

实例

文件目录



HelloBean 如上例所示。

配置文件：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd" >
<beans>
     <bean id="helloworld" class="org.springframework.beans.HelloBean">
     <property name="helloWorld" value="hello !david!"/>
     </bean>
</beans>
```

测试用例：

```java
package org.springframework.beans;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.beans.factory.xml.XmlBeanDefinitionReader;

public class XmlBeanFactoryTest {
    @SuppressWarnings("cast")
    public static void main(String[] args) {
        DefaultListableBeanFactory beanRegistry = new DefaultListableBeanFactory();
        XmlBeanDefinitionReader reader = new XmlBeanDefinitionReader(beanRegistry);
        reader.loadBeanDefinitions("classpath:org/springframework/beans/beans.xml");
        BeanFactory container = (BeanFactory)beanRegistry;
        HelloBean helloworld = (HelloBean)container.getBean("helloworld");
        System.out.println(helloworld.getHelloWorld());
    }
}
```

自己写的代码、测试用例使用代码框的形式展示

总结

我们为什么需要 Spring 框架来给我们提供这个 BeanFactory 的功能呢？原因是一般我们认为是，可以将原来硬编码的依赖，通过 Spring 这个 BeanFactory 这个工长来注入依赖，也就是说原来只有依赖方和被依赖方，现在我们引入了第三方——Spring 这个 BeanFactory，由它来解决 Bean 之间的依赖问题，达到了松耦合的效果；

这个只是原因之一，还有一个更加重要的原因：在没有 Spring 这个 BeanFactory 之前，我们都是直接通过 new 来实例化各种对象，现在各种对象 Bean 的生产都是通过 BeanFactory 来实例化的，这样的话，Spring 这个 BeanFactory 就可以在实例化 Bean 的过程中，做一些小动作——在实例化 Bean 的各个阶段进行一些额外的处理，也就是说 BeanFactory 会在 Bean 的生命周期的各个阶段中对 Bean 进行各种管理，并且 Spring 将这些阶段通过各种接口暴露给我们，让我们可以对 Bean 进行各种处理，我们只要让 Bean 实现对应的接口，那么 Spring 就会在 Bean 的生命周期调用我们实现的接口来处理该 Bean。

}

06 Spring 5.x源代码编译并导入到eclipse或者ide

08 青出于蓝而胜于蓝之揭秘Spring容器ApplicationContext