# 第1部分核心实现

* 第l章Spring整体架构和环境搭建
* 第2章容器的基本实现
* 第3章默认标签的解析
* 第4章自定义标签的解析
* 第5章bean的加载
* 第6章容器的功能扩展
* 第7章AOP

## 第 1 章 Spring 整体架构和环境搭建

Spring是于2003年兴起的一个轻量级Java开源框架，由RodJohnson在其著作Expert One-On-OneJ2EE Designand Development中阐述的部分理念和原型衍生而来。

Spring是为了解决企业应用开发的复杂性而创建的，它使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发，从简单性、可测试性和松搞合的角度而言，任何Java应用都可以从Spring中受益。

### 1.1Spring的整体架构

Spring框架是一个分层架构，它包含一系列的功能要素，并被分为大约20个模块，如图1-1所示。

Spring Framework Runtime

Data Access/Integration

JDBC

ORM

OXM

JMS

Transactions

Web  
(MVC/Remoting)

Servlet

Portiet

Web

AOP

Test

图 1-1 Spring 整体架构图

Aspects

Instrumentation

Core Container

Context

Expressio  
Language

Beans

Core

这些模块被总结为以下几部分。

#### 1.CoreContaine

CoreContainer（核心容器）包含有Core、Beans、Context和ExpressionLanguage模块。

Core和Beans模块是框架的基础部分，提供IoC（转控制）和依赖注入特性。这里的基础概念BeanFactory，它提供对Factory模式的经典实现来消除对程序性单例模式的需要，并真正地允许你从程序逻辑中分离出依赖关系和配置。

* Core模决主要包含Spring框架基本的核心工具类，Spring的其他组件都要用到这个包里的类，Core模块是其他组件的基本核心。当然你也可以在自己的应用系统中使用这些工具类。
* Beans模块是所有应用都妥用到的，它包含访问配直文件、创建和管理bean以及进行

Inversion of Control/Dependency Injection(IoC/DI）操作相关的所有类。

* Context模块构建于Core和Beans模块基础之上，提供了一种类似于JNDI注册器的框架式的对象访问方法。Context模块继承了Beans的特性，为Spring核心提供了大量扩展，添加了对国际化（例如资源绑定）、事件传播、资源加载和对Context的透明创建的支持。Context模块同时也支持J2EE的一些特性，例如EJB、JMX和基础的远程处理。ApplicationContext接口是Context模块的关键。
* Expression Language模块提供了强大的表达式语言，用于在运行时查询和操纵对象。它是JSP2.1规范中定义的unifed expression language的扩展。该语言支设置／获取属性的值，属性的分配，方法的调用，访问数组上下文（accessiong the context of arrays）、容器和索引器、逻辑和算术运算符、命名变量以及从Spring的IoC容器中根据名称检索对象。它也支持list投影、选择和一般的list聚合。

#### 2.DataAccess/Integration

DataAccess/Integration层包含了JDBC、ORM、OXM、JMS和Transaction模块。

* JDBC模块提供了一个JDBC抽象层，它可以消除冗长的JDBC编码和解析数据库厂商特有的错误代码。这个模块包含了Spring对JDBC数据访问进行封装的所有类
* ORM模块为流行的对象－关系映射API，如JPA、JDO、Hibernate、iBatis等，提供了一个交互层。利用ORM封装包，可以混合使用所有Spring提供的特性进行O/R映射，如前边提到的简单声明性事务管理。

Spring框架插入了若干个ORM框架，从而提供了ORM的对象关系工具，其中包括JDO、Hibernate、iBatis SQLMap。所有这些都遵从Spring的通用事务和DAO异常层次结构。

* OXM模块提供了一个对Object/XML映射实现的抽象层，Object/XML映射实现包括JAXB、Castor、XMLBeans、JiBX和XStrearn。
* JMS(Java Messaging Service）模块主要包含了一些制造和消费消息的特性。
* Transaction模块支持编程和声明性的事务管理，这些事务类必须实现特定的接口，并且对所有的POJO都适用。

#### 3.Web

Web上下文模块建立在应用程序上下文模块之上，为基于Web的应用程序提供了上下文。所以，Spring框架支持与Jakarta Struts的集成。Web模块还简化了处理大部分请求以及将请求参数绑定到域对象的工作。Web层包含了Web、Web-Servlet、Web-Struts和Web-Porlet模块，具体说明如下。

* Web模块：提供了基础的面向Web的集成特性。例如，多文件上传、使用servlet listeners初始化IoC容器以及一个面向Web的应用上下文。它还包含Spring远程支持中Web的相关部分。
* Web-Serviet模块web.servlet.jar：该模块包含Spring的model-view-controller(MVC)实现。Spring的MVC框架使得模型范围内的代码和web forms之间能够清楚地分离开来，并与Spring框架的其他特性集成在一起。
* Web-Struts模块：该模块提供了对Struts的支持，使得类在Spring应用中能够与一个典型的Struts Web层集成在一起。注意，该支持在Spring3.0中已被弃用。
* Web-Porlet模块：提供了用于Portlet环境和Web-Servlet模块的MVC的实现。

#### 4.AOP

AOP模块提供了一个符合AOP联盟标准的面向切面编程的实现，它让你可以定义例如方法拦截器和切点，从而将逻辑代码分开，降低它们之间的调合性。利用source-level的元数据功能，还可以将各种行为信息合并到你的代码中，这有点像.Net技术中的attribute概念。

通过配置管理特性，Spring AOP模块直接将面向切面的编程功能集成到了Spring框架中，所以可以很容易地使Spring框架管理的任何对象支持AOP。Spring AOP模块为基于Spring的应用程序中的对象提供了事务管理服务。通过使用Spring AOP，不用依赖EJB组件，就可以将声明性事务管理集成到应用程序中。

* Aspects模块提供了对AspectJ的集成支持。
* Instrumentation模块提供了class instrumentation支持和classloader实现，使得可以在特定的应用服务器上使用。

#### 5.Test

Test模块支持使用JUnit和TestNG对Spring组件进行测试。

### 1.2环境搭建

由于在上一版本中作者对环境搭建描述得比较粗糙，导致非常多的读者询问，作者在此表示歉意，并在这一版本中补充了非常详细的Spring环境搭建流程。如果是第一次接触Spring源码的环境搭建，确实还是比较麻烦的。作者使用的编译器为目前流行的IntelliJIDEA,I阪木为2018旗舰版。Eclipse用户还需要作己揣摩环境搭建方法，这里不再赘述。

#### 1.2.1 源码链接获取

#### 1.2.2 源码下载及 IDEA 导入

### 1.3 cglib 和 obienesis 的编译错误解决

#### 1.3.1 问题发现及原因

#### 1.3.2 问题解决

### 1.4 AspectJ 编译问题解决

#### 1.4.1 问题发现

#### 1.4.2 问题原因

#### 1.4.3 问题解决

第2章容器的基本实现  
源码分析是一件非常煎熬且极具挑战性的任务，你准备好开始战斗了吗？  
在正式开始分析Spring源问之前，我们有必要先来回顾一下Spring中最简单的用法，尽  
管我相信您已经对这个例子非常熟悉了。  
2.1容器基本用法  
bean是Spring中最核心的东西，因为Spring就像是个大水桶，而bean就像是容器中的水，  
水桶脱离了水便也没什么用处了，那么我们先看看bean的定义。

## 第2章容器的基本实现

源码分析是一件非常煎熬且极具挑战性的任务，你准备好开始战斗了吗？

在正式开始分析Spring源问之前，我们有必要先来回顾一下Spring中最简单的用法，尽管我相信您已经对这个例子非常熟悉了。

### 2.1容器基本用法

bean是Spring中最核心的东西，因为Spring就像是个大水桶，而bean就像是容器中的水，水桶脱离了水便也没什么用处了，那么我们先看看bean的定义。

|  |
| --- |
| **public class** MyTestBean {  **private** String **testStr** = **"testStr"**;   **public** String getTestStr() {  **return testStr**;  }   **public void** setTestStr(String testStr) {  **this**.**testStr** = testStr;  } } |

这么看来bean并没有任何特别之处，的确，Spring的目的就是让我们的bean能成为－个纯粹的POJO，这也是Spring所边求的。接下来看看配置文件：

|  |
| --- |
| *<?***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"**>   <**bean id="myTestBean" class="com.prosayj.springboot.spring源码深度剖析.\_02\_容器的基本实现.MyTestBean"**></**bean**> </**beans**> |

在上面的配置中我们看到了bean的声明方式，尽管Spring中bean的元素定义着N种属性来支撑我们业务的各种应用，但是我们只要声明成这样，基本上就已经可以满足我们的大多数应用了。好了，你可能觉得还有什么,但是，真没了，Spring的人门示例到这里已经结束，我们可以写测试代码测试了。

|  |
| --- |
| @SuppressWarnings(**"deprecation"**) **public class** \_01\_BeanFactoryTest {  @Test  **public void** testSimpleLoad() {  XmlBeanFactory beanFactory = **new** XmlBeanFactory(  **new** ClassPathResource(**"com/prosayj/springboot/spring源码深度剖析/\_02\_容器的基本实现/beanFactoryTest.xml"**));  MyTestBean bean = (MyTestBean) beanFactory.getBean(**"myTestBean"**);  *assertEquals*(**"testStr"**, bean.getTestStr());  } } |

相信聪明的读者会很快看到我们期望的结果：在Eclipse中显示了Grean Bar。直接使用BeanFactory作为容器对于Spring的使用来说并不多见，甚至是甚少使用，因为在企业级的应用中大多数都会使用的Application Context（后续章节我们会介绍它们之间的区别），这里只是用于测试，让读者更快更好地分析Spring的内部原理。

OK，我们又复习了一遍Spring，你是不是会很不屑呢？这样的小例子没有任何挑战性。嗯，确实，这样的使用是过于简单了，但是本书的目的并不是介绍如何使用Spring，而是帮助您更好地了解Spring的内部原理。读者可以自己先想想，上面的一句简单代码都执行了什么样的逻辑呢？这样一句简单代码其实在Spring中执行了太多太多的逻辑，即使作者用半本书的文字也只能介绍它的大致原理。那么就让我们快速地进入分析状态吧。

### 2.2功能分析

现在我们可以来好好分析一下上面测试代码的功能，来探索上面的测试代码中Spring究竟帮助我们完成了什么工作？不管之前你是否使用过Spring，当然，你应该使用过的，毕竟本书面向的是对Spring有一定使用经验的读者，你都应该能猜出来，这段测试代码完成的功能无非就是以下几点。

* 读取自己直文件beanFactoryTest.xml
* 根据beanFactoryTest.xml中的配置找到对应的类的配直，并实例化。
* 调用实例化后的实例。

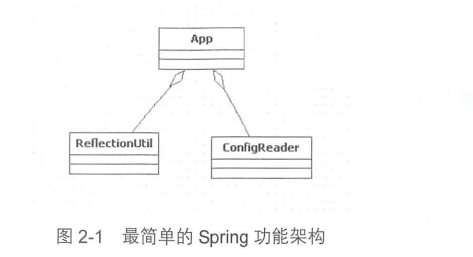
为了更清楚地描述，作者临时画了设计类图，如图2-1所示，如果想完成我们预想的功能，至少需要3个类。

* ConfigReader：用于读取及验证自己直文件。我们妥用配直文件里面的东西，当然首先要做的就是读取，然后放直在内存中。
* ReflectionUtil：用于根据配直文件中的自己直进行反射实例化。比如在上例中beanFactoryTest.xml出现的

<bean id="myTestBean" class="com.prosayj.springboot.MyTestBean"></bean>，

我们就可以根据bean.MyTestBean进行实例化。

* App:用于完成整个逻辑的串联。

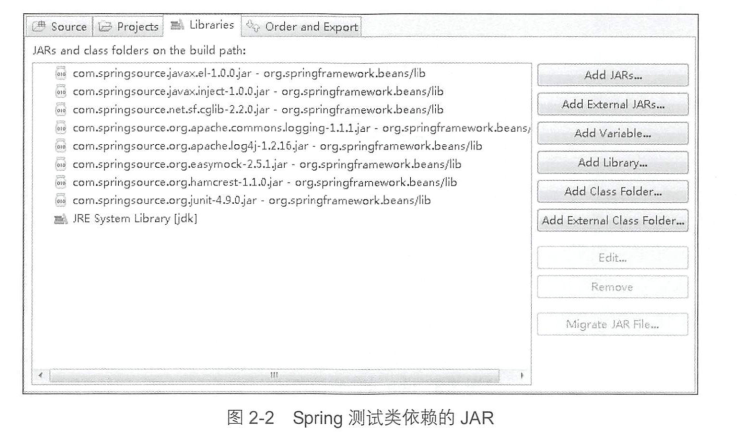


按照原始的思维方式，整个过程无非如此，但是作为一个风靡世界的优秀源代码真的就这么简单吗？

### 2.3工程搭建

不如我们首先大致看看Spring的源码。

在Spring源码中，用于实现上面功能的是org.Springframework.beans.jar，我们看源码的时候要打开这个工程，如果我们只使用上面的功能，那就没有必要引人Spring的其他更多的包，当然Core是必需的，还有些依赖的包如图二2所示。



引人依赖的JAR消除掉所有编译错误后，终于可以看源码了。或许你已经知道了答案，Spring居然用了N多代码实现了这个看似很简单的功能，那么这些代码都是做什么用的呢？

Spring在架构或者编码的时候又是如何考虑的呢？带着疑问，让我们踏上研读Spring源码的征程。

### 2.4Spring的结构组成

我们首先尝试梳理Spring的框架结构，从全局的角度了解Spring的结构组成。

#### 2.4.1 beans包的层级结构

作者认为阅读源码的最好方法是通过示例跟着操作一遍，虽然有时候或者说大多数时候会被复杂的代码绕来绕去，绕到最后已经不知道自己身在何处了，但是，如果配以UML还是可以搞定的。作者就是按照自己的思路进行分析，并配合必要的UML，希望读者同样可以跟得上思路。

我们先看看整个beans工程的源码结构，如图2-3所示。

* beans包中的各个源码包的功能如下。
* src/main/java用于展现Spring的主要逻辑。
* src/main/resources用于存放系统的配置文件。
* src/test/iava用于对主要逻辑进行单元测试。
* src/test/resources用于存放测试用的配直文件。



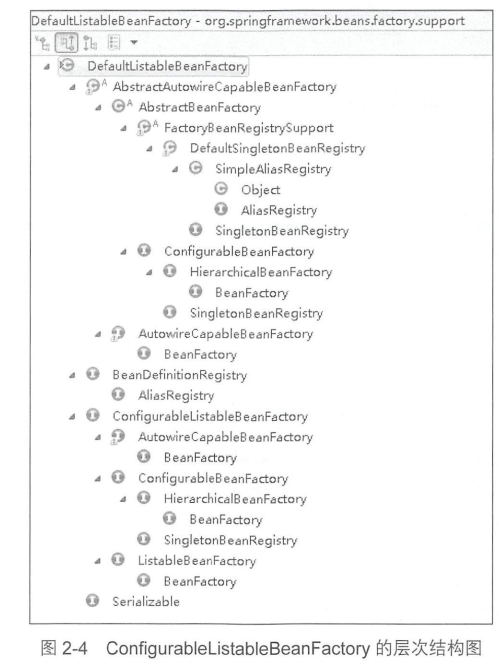
#### 2.4.2 核心类介绍

通过beans工程的结构介绍，我们现在对beans的工程结构有了初步的认识，但是在正式开始源码分析之前，有必要了解Spring中核心的两个类。

##### DefaultlistableBeanFactory

XmlBeanFactory继承向DefaultListableBeanFacto1y，而DefaultListableBeanFactmy是整个bean加载的核心部分，是Spring注册及加载bean的默认实现，而对于XmlBeanFactory与DefaultListableBeanFactory不同的地方其实是在XmlBeanFactory中使用了自定义的XML读取器XmlBeanDefinitionReader，实现了个性化的BeanDefinitionReader读取，DefaultListableBeanFacto1y继承了AbstractAutowireCapableBeanFactory并实现了ConfigurableListableBeanFactory可以及BeanDefinitionRegistry接口。

图2-4是ConfigurableListableBeanFactory的层次结构图，图2-5是相关类图。



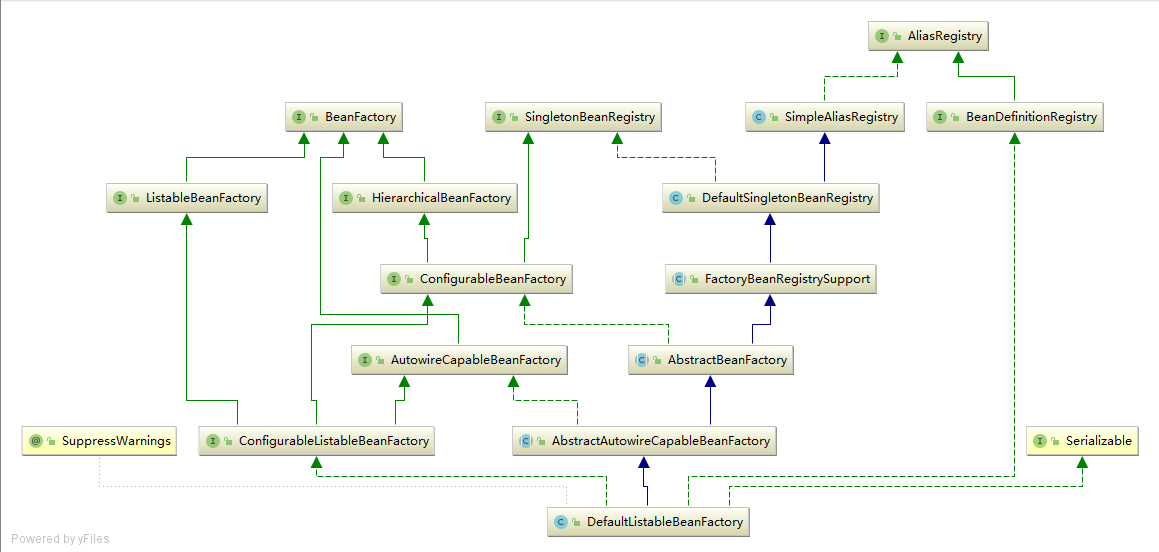
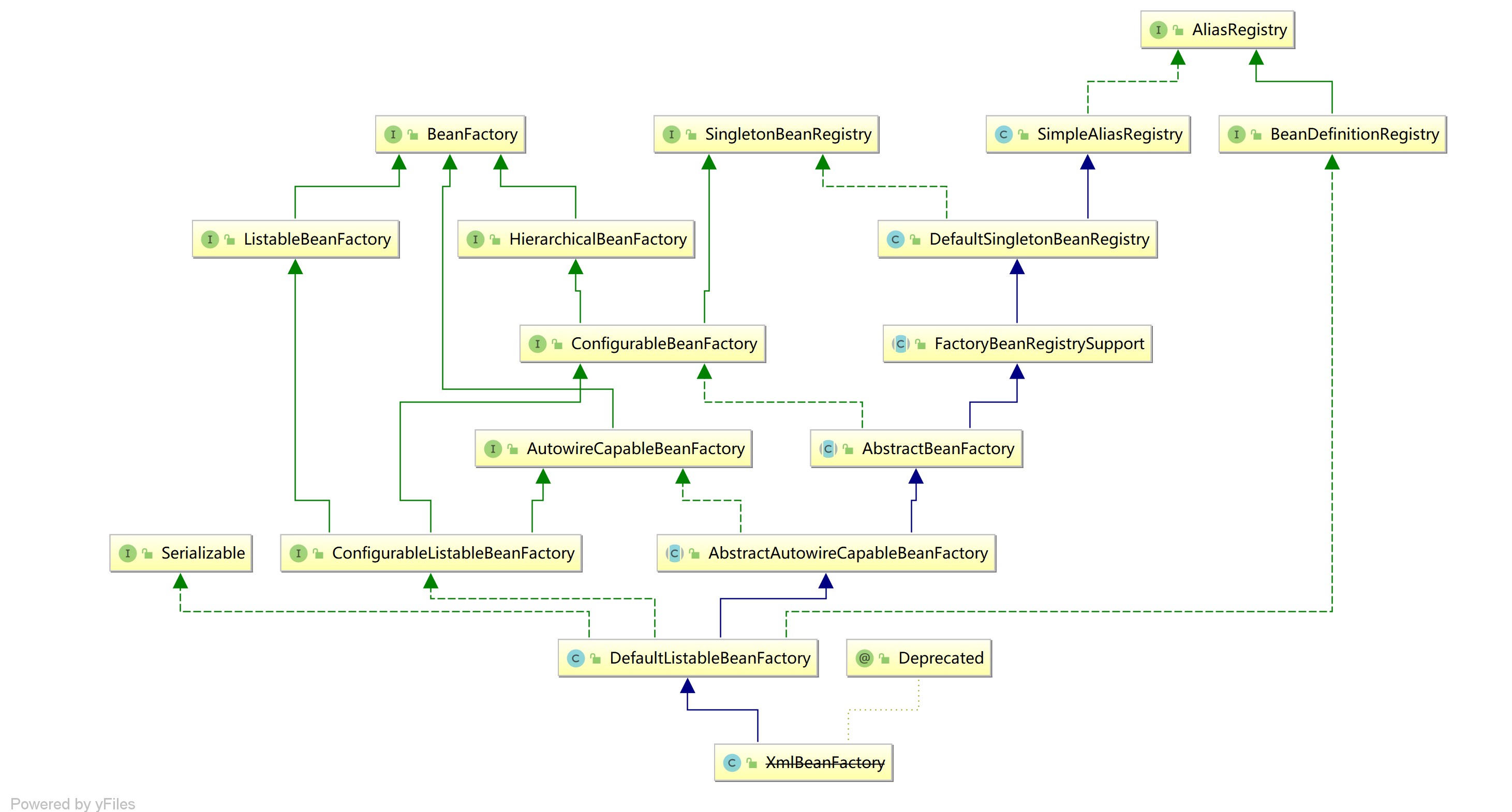


图2-5 容器加载相关类图

从上面的类图以及层次结构阁中，我们可以很清晰地从全局角度了解DefaultListableBeanFactory的脉络。如果读者没有了解过Spring源码可能对上面的类图不是很理解，不过没关系，通过后续的学习，你会逐渐了解每个类的作用。那么，让我们先简单地了解图2-5中各个类的作用。

* AliasRegistrγ：定义对alias的简单增删改等操作。
* SimpleAliasRegistry：主要使用map作为alias的缓存，并对接口AliasRegistry进行实现。
* SingletonBeanRegistry：定义对单例的注册及获取。
* BeanFactory：定义获取bean及bean的各种属性。
* DefauItSingletonBeanRegistry：对接口SingletonBeanRegistry各函数的实现。
* HierarchicalBeanFactory：继承BeanFactory，也就是在BeanFactory定义的功能的基础上增加了对parentFactory的支持。
* BeanDefinitionRegistry：定义对BeanDefinition的各种增删改操作。
* FactoryBeanRegistrySupport：在DefaultSingletonBeanRegistry基础上增加了对FactoryBean的特殊处理功能。
* ConfigurableBeanFactory：提供配直Factory的各种方法。
* ListableBeanFactory：根据各种条件获取bean的配直清单。
* AbstractBeanFactory：综合FactoryBeanRegistrySupport和ConfigurableBeanFactory的功能。
* AutowireCapableBeanFactory：提供创建bean、自动注入、初始化以及应用bean的后处理器。
* AbstractAutowireCapableBeanFactory：综合AbstractBeanFacto1y并对接口AutowireCapableBeanFactory进行实现。
* ConfigurableListableBeanFactory:Beanfactory配直清单，指定忽略类型及接口等。
* DefaultListableBeanFactory：综合上面所有功能，主要是对bean注册后的处理。



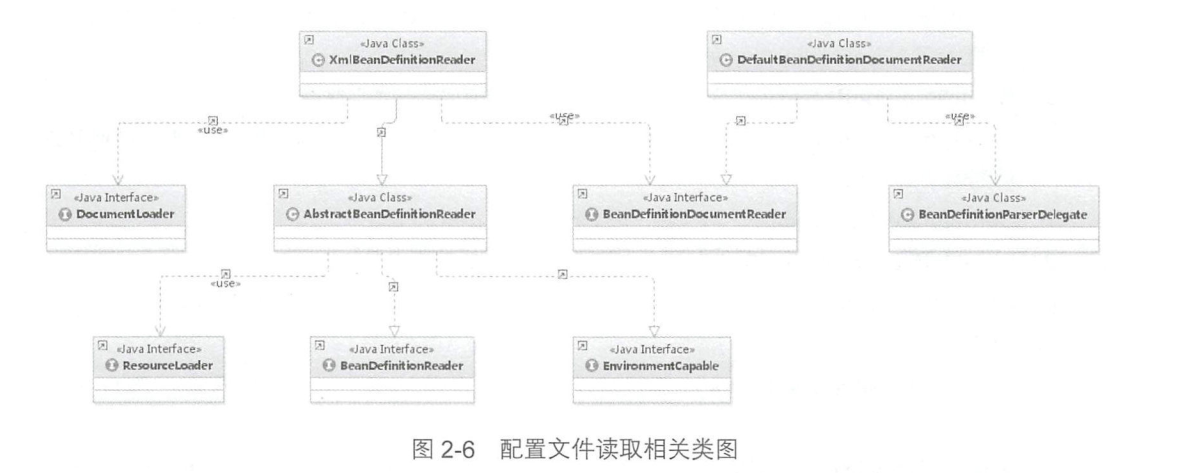
XmlBeanFactory对DefaultListableBeanFactorγ类进行了扩展，主要用于从XML文档中读取BeanDefinition，对于注册及获取bean都是使用从父类DefaultListableBeanFactory继承的方法去实现，而唯独与父类不同的个性化实现就是增加了XmlBeanDefinitionReader类型的reader属性。在XmlBeanFactory中主要使用reader属性对资源文件进行读取和注册。

##### XmlBeanDefinitionReader

XML配置文件的读取是Spring巾重要的功能，因为Spring的大部分功能都是以配置作为切入，点的，那么我们可以从XmlBeanDefinitionReader中梳理一下资源文件读取、解析及注册的大致脉络，首先我们看看各个类的功能。

* ResourceLoader：定义资源加载器，主妥应用于根据给定的资源文件地址返回对应的Resource。
* BeanDefinitionReader：主要定义资源文件读取并转换为BeanDefinition的各个功能。
* EnvironmentCapable：定义获取Environment方法。
* DocumentLoader：定义从资源、文件加载到转换为Document的功能。
* AbstractBeanDefinitionReader：对EnvironmentCapable、BeanDefinitionReader类定义的功能进行实现
* BeanDefinitionDocumentReader：定义读取Docuemnt并注册BeanDefinition功能。
* BeanDefinitionParserDelegate：定义解析Element的各种方法。

经过以上分析，我们可以梳理出整个XML配置文件读取的大致流程，如图2-6所示，在XmlBeanDefinitionReader中主要包含以下几步的处理。



* 通过继承向AbstractBeanDefinitionReader中的方法，来使用ResourLoader将资源文件路径转换为对应的Resource文件。
* 通过DocumentLoaderX才Resource文件进行转换，将Resource文件转换为Document文件。
* 通过实现接口BeanDefinitionDocumentReader的DefaultBeanDefinitionDocumentReader类对Document进行解析，并使用BeanDefinitionParserDelegate对Element进行解析。

## 2.5 容器的基础XmlBeanFactory

好了，到这里我们已经对Spring的容器功能有了大致的了解，尽管你可能还很迷糊，但是不要紧，接下来我们会详细探索每个步骤的实现。再次重申一下代码，我们接下来要深入分析以下功能的代码实现：

BeanFactory bf = new XmlBeanFactory(new ClassPathResource(”beanFactoryTest.xml”)),

通过XmlBeanFactory初始化时序图（如图2-7所示）我们来看一看上面代码的执行逻辑。

时序图从BeanFactoryTest测试类开始，通过时序图我们可以一目了然地看到整个逻辑处理顺序。在测试的BeanFactoryTest中首先调用ClassPathResource的构造函数来构造Resource资源文件的实例对象，这样后续的资源处理就可以用Resource提供的各种服务来操作了，当我们有了Resource后就可以进行XmlBeanFactory的初始化了。那么Resource资源是如何封装的呢？

