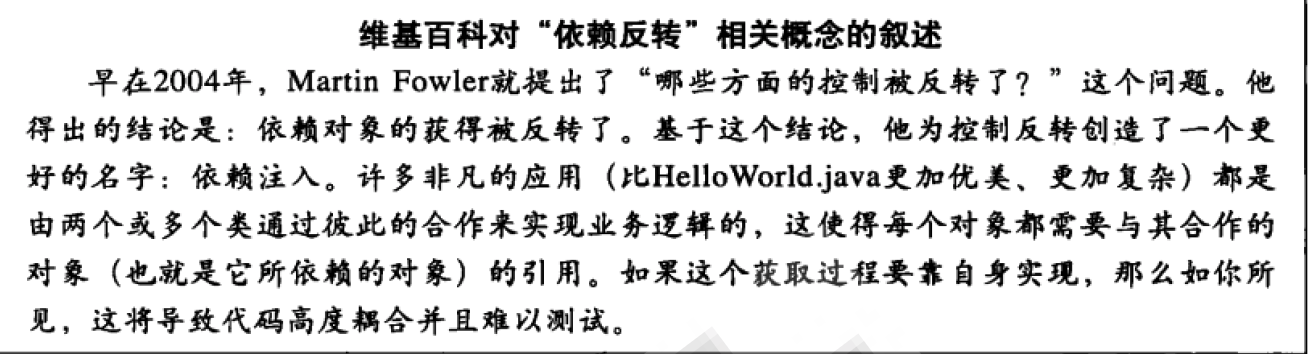
# Spring

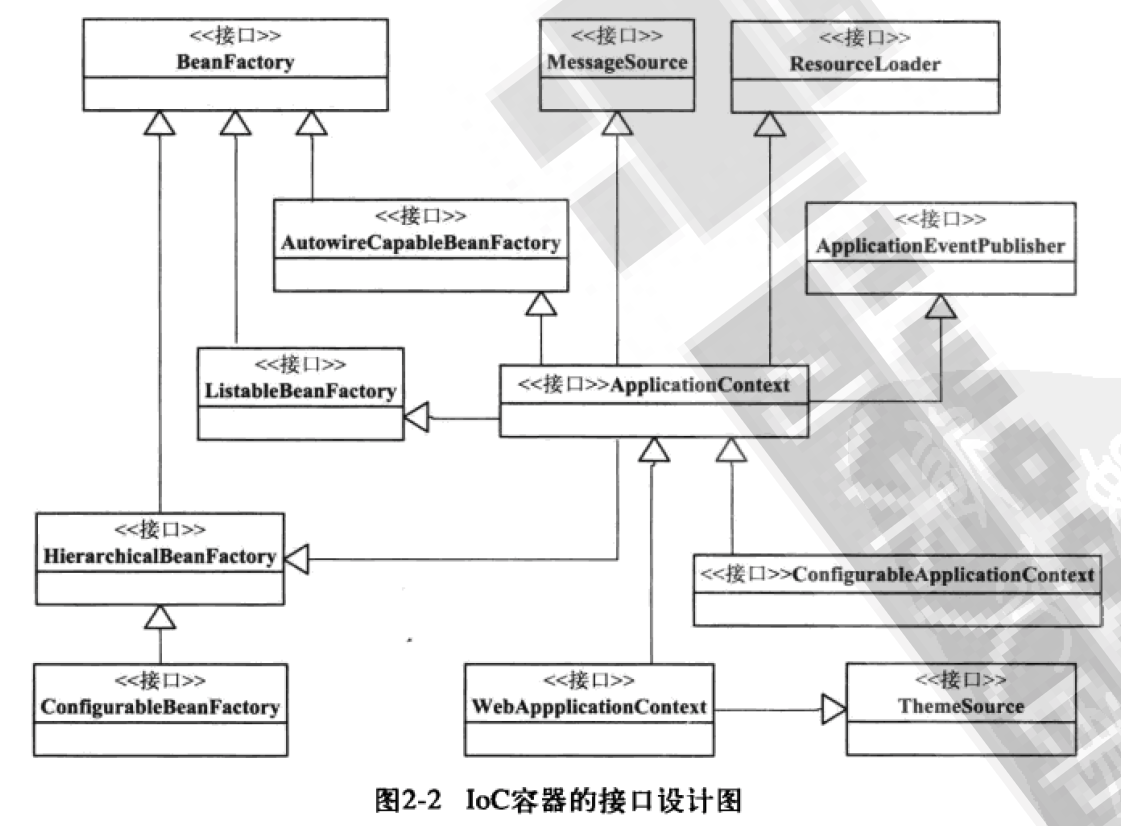
## Spring简介

IoC容器和AOP是Spring的核心。在Spring中，Java EE的服务都被抽象到IoC容器中，并通过AOP进行有效的封装，因为依赖注入的特性，这些复杂的依赖关系的管理被反转并交给容器，使复杂的依赖关系管理从应用中解放出来。在Spring中，各个模块的依赖关系通过简单的IoC配置文件进行描述，是这些外部化的信息几种并且明了。

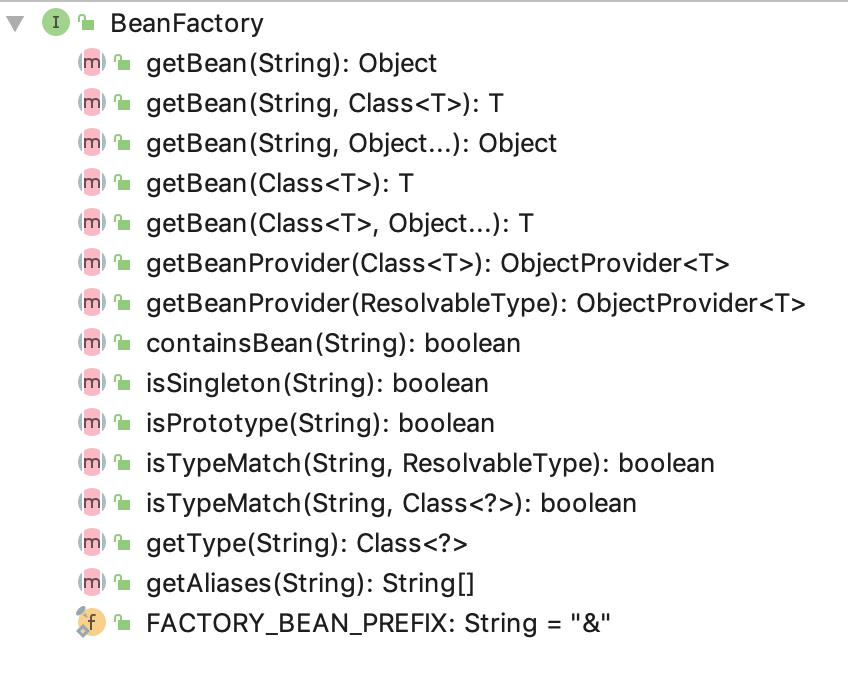
## IoC容器的实现



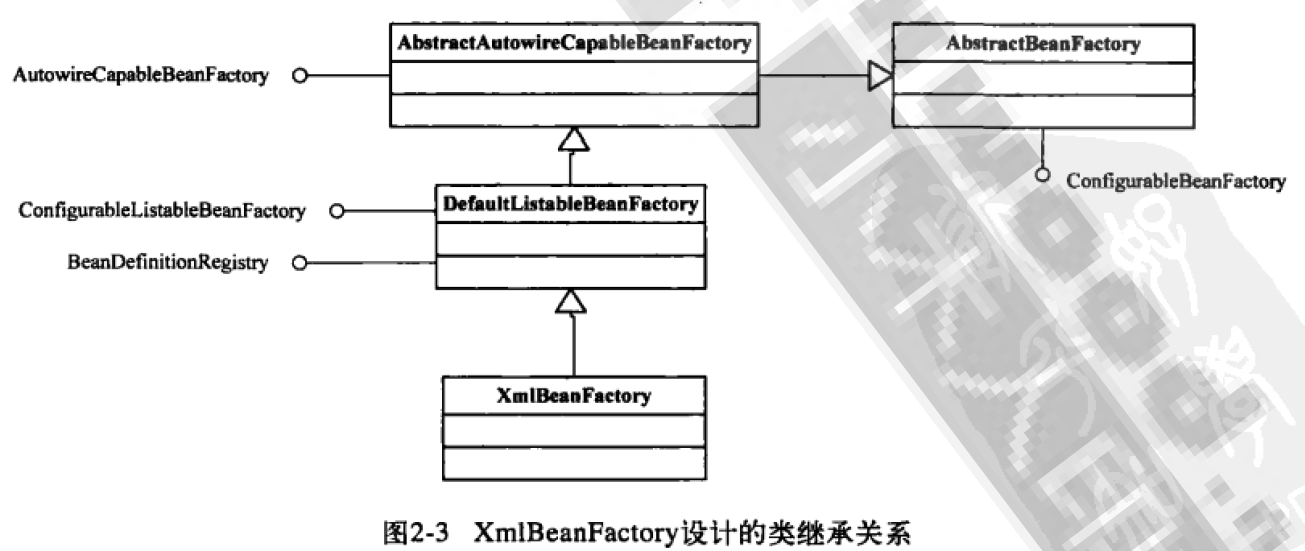
如果合作对象的引用或依赖关系的管理由具体对象来完成，会导致代码的高耦合和可测试性的降低，这对复杂的面向对象系统的设计是非常不利的。依赖控制反转的实现有很多种。在Spring中，IoC容器是实现这个模式的载体，他可以在对象生成或初始化时直接将数据注入到对象中，也可以通过将对象引用注入到对象数据域中的方式来注入对方法调用的依赖。这种依赖注入可以是递归的，对象被逐层注入。



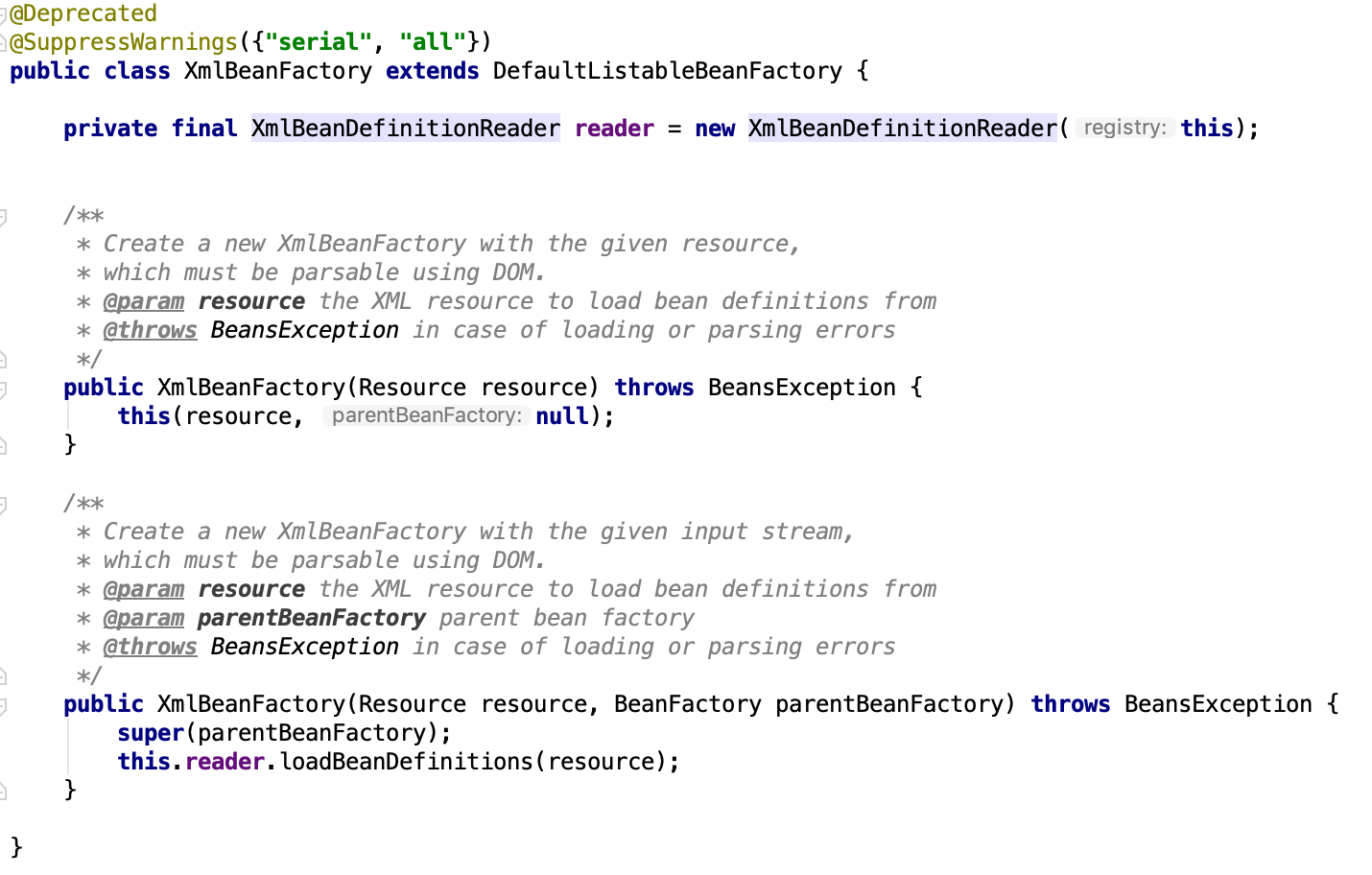
1. 从接口BeanFactory到HierarchicalBeanFactory,再到ConfigurableBeanFactory，是一条主要的BeanFactory设计路径。在这条接口设计路径中，BeanFactory接口定义了基本的IoC容器的规范。在这个接口定义中，包括了getBean这样的IoC容器的基本方法(通过这个方法可以从容器中取得Bean)。而HierarchicalBeanFactory接口在继承了BeanFactory的基本接口之后，增加了getParentBeanFactory()的接口功能，使BeanFactory具备了双亲IoC容器的管理功能。ConfigurableBeanFactory接口主要定义了一些对BeanFactory的配置功能，比如通过setParentBeanFactory()设置双亲IoC容器，通过addBeanPostProcessor()配置Bean后置处理器等等。
2. 从BeanFactory到ListableBeanFactory，再到ApplicationContext,再到我们常用的WebApplicationContext或者ConfigurableApplicationContext接口。在这个接口体系中，ListableBeanFactory和HierarchicalBeanFactory俩个接口，连接BeanFactory接口定义。在ListableBeanFactory接口中，细化了许多BeanFactory的接口功能，比如定义了getBeanDefinitionNames()接口方法。对于AplicationContext接口，他通过继承MessageSource、ResourceLoader、ApplicationEventPublisher接口，在BeanFactory简单IoC容器的基础上添加了许多对高级容器的特性的支持。
3. 这里涉及的是主要接口体系，具体的IoC容器都是在这个接口体系下实现的，比如DefaultListableBeanFactory，这个基本IoC容器的实现就是实现了ConfigurableBeanFactory，从而成为一个简单IoC容器的实现。在比如XmlBeanFactory,都是在DefaultListableBeanFactory的基础上做扩展，同样地，ApplicationContext的实现也是如此。
4. 这个 接口系统是以BeanFactory和ApplicationContext为核心的。BeanFactory是IoC容器的最基本接口。在ApplicationContext的设计中，一方面，可以看到他继承了BeanFactory接口体系中的ListableBeanFactory、AutowireCapableBeanFactory、HierarchicalBeanFactory等BeanFactory的接口，具备了BeanFactory IoC容器的基本功能。另一方面，通过继承MessageSource、ResourceLoader、ApplicationEventPublisher这些接口，BeanFactory为ApplicationContext赋予了更高级的IoC容器特性。但对于ApplicationContext而言，为了在web环境中使用它，还设计了WebApplicationContext接口，而这个接口通过继承ThemeSource接口来扩充功能。



1. getBean()不多说
2. getBeanProvider()返回指定bean的提供程序，允许对实例进行惰性按需检索，包括可用性和唯一性选项。
3. containsBean()让用户能够判断容器是否含有指定名字的Bean
4. isSingleton()查询指定名字的Bean是否是Singleton类型的Bean。对于Singleton属性，用户可以在BeanDefinition中指定。
5. isPrototype来查询指定名字的Bean是否是prototype类型的。与Singleton属性一样，这个属性也可以由用户在BeanDefinition中指定。
6. isTypeMatch()查询指定了名字的Bean的Class类型是否是特定的Class类型。这个Class类型可以由用户指定
7. getType()查询指定名字的Bean的Class类型
8. getAliases查询指定了名字的Bean的所有别名，这些别名都是用户在BeanDefinition中定义的



XmlBeanFactory源码：



XmlBeanFactory继承自DefaultListableBeanFactory，后者是经常要用到的一个IoC容器的实现，DefaultListableBeanFactory实际上包含了基本IoC容器所具有的重要功能，也是在很多地方都会用到的容器系列的一个基本产品。在Spring中，实际上是把DefaultListableBeanFatory作为一个默认的功能完整的IoC容器来使用的。

XmlBeanFactory是一个与XML相关的BeanFatory，也就是说它是一个以XML文件方式定义的BeanDefinition的IoC容器。对于XML文件定义信息的处理并不是由XmlBeanFactory直接完成的。在XmlBeanFactory中，初始化了一个XmlBeanDefinitionReader对象，对于XML形式的信息由该对象完成。

编程式使用IoC容器：

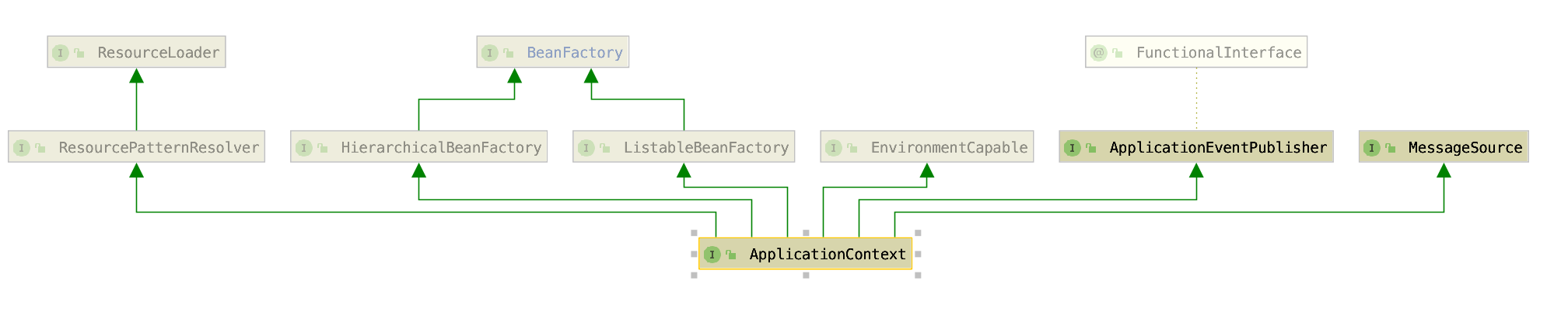
ClassPathResource res = new ClassPathResource(“beans.xml”);

DefaultListableBeanFactory factory = new DefaultListableBeanFactory();

XmlBeanDefinitionReader reader = new XmlBeanDefinitionReader(factory);

reader.loadBeanDefiniitons(res);

在Spring中，系统已经为用户提供了许多已经定义好的容器实现，而不需要开发人员事必躬亲。相遇哪些简单拓展BeanFatory的基本IoC容器，开发人员常用的ApplicationContext除了能够提供前面介绍的容器的基本功能外，还为用户提供了以下的附加服务，可以让客户更方便的使用。所以说，ApplicationContext是一个高级形态意义的IoC容器。



1. 支持不同的信息源，ApplicationContext扩展了MessageSource接口，这些信息源的扩展功能可以支持国际化的实现，为开发多语言版本的应用提供服务。
2. 访问资源。这一特性体现在对Resourceloader和Resource的支持上，这样我们可以从不同地方得到Bean定义资源。这种抽象使用户程序可以灵活的定义Bean定义信息，尤其是从不同的I/O途径得到的Bean定义信息。这在接口关系上看不出来，不过一般来说，具体ApplicationContext都是继承了DefaultResourceLoader的子类。因为DefaultResourceLoader是AbstractApplicationContext的基类。
3. 支持应用事件。继承了接口ApplicationEventPublisher，从而在上下文中引入了事件机制。这些事件和Bean的生命周期的结合为Bean的管理提供了便利。
4. 在ApplicationContext中提供的附加服务。这些服务使得基本IoC容器的功能更丰富。因为具有了这些丰富的附加功能，使得ApplicationCAontext与简单的BeanFactory相对，对它的使用是一种面向框架的使用风格，所以一般建议在开发应用时使用ApplicationContext作为IoC容器的基本形式。

在ApplicationContext容器中，我们以常用的FileSystemXmlApplicationContext的实现为例



在FileSystemXmlApplicationContext中，作为一个具体的应用上下文，只需要实现和它自身设计相关的俩个功能。

一个功能是，如果应用直接使用FileSystemXmlApplicationContext，对于实例化这个应用上下文的支持，同时启动IoC容器的refresh()过程。

另一个功能是与FileSystemXmlApplicationContext设计具体相关的功能，这部分与怎么从文件系统加载XML的Bean定义资源有关。

通过这个过程，可以为在文件系统找那个读取以XML形式存在的BeanDefinition做准备，因为不同的应用上下文实现对应着不同的读取BeanDefinition的形式，在FileSystemXmlApplicationContext中的实现代码在getResourceByPath中体现。

IoC的初始化过程

IoC容器的初始化过程是由refresh()方法来启动的，这个方法标志着容器的正式启动。具体来说，这个启动包括BeanDefinition的Resource定位，载入和注册三个基本过程。

1. Resource定位过程

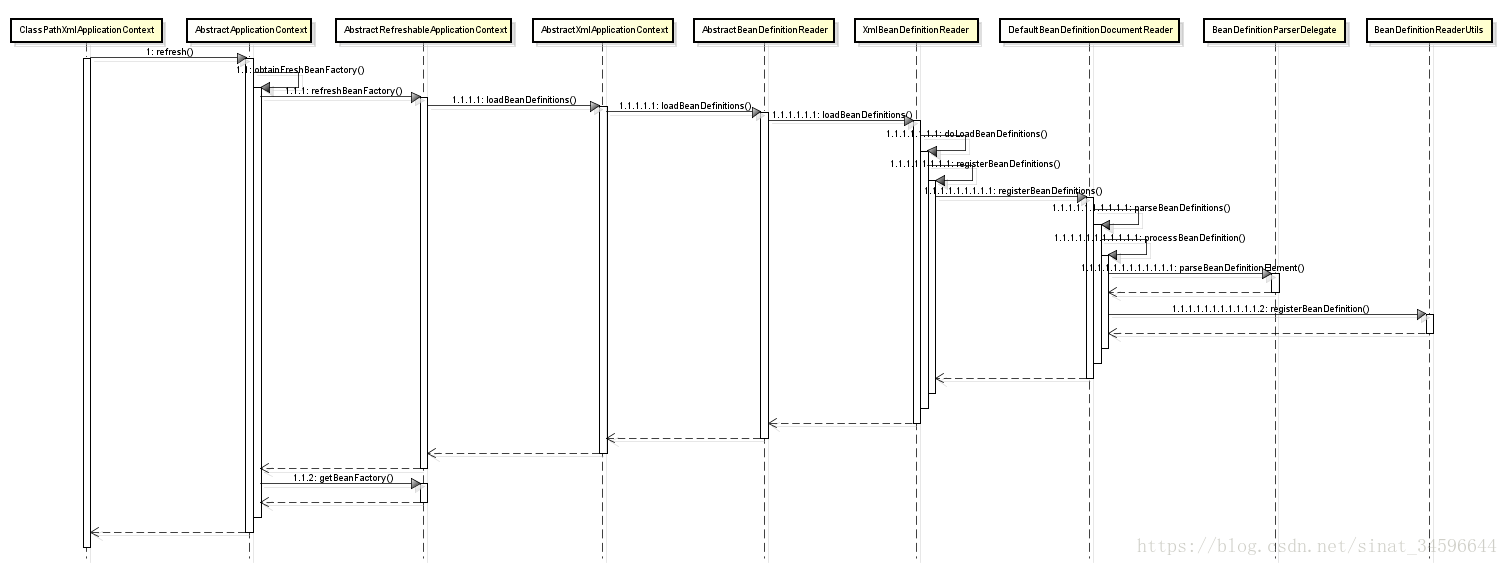
这个Resource定位指的是BeanDefinition的资源定位，它由ResourceLoader通过统一的Resource接口来完成，这个Resource对各种形式的BeanDefinition的使用都提供了统一的接口。

1. BeanDefinition的载入

这个载入过程是将用户定义好的Bean表示成IoC容器内部的数据结构，而这个容器内部的数据结构就是BeanDefinition。这个BeanDefinition就是POJO对象在IoC容器中的抽象。通过BeanDefinition定义的数据结构，使IoC容器能够方便的对POJO对象也就是Bean进行管理。

1. 向IoC容器注册这些BeanDefinition的过程

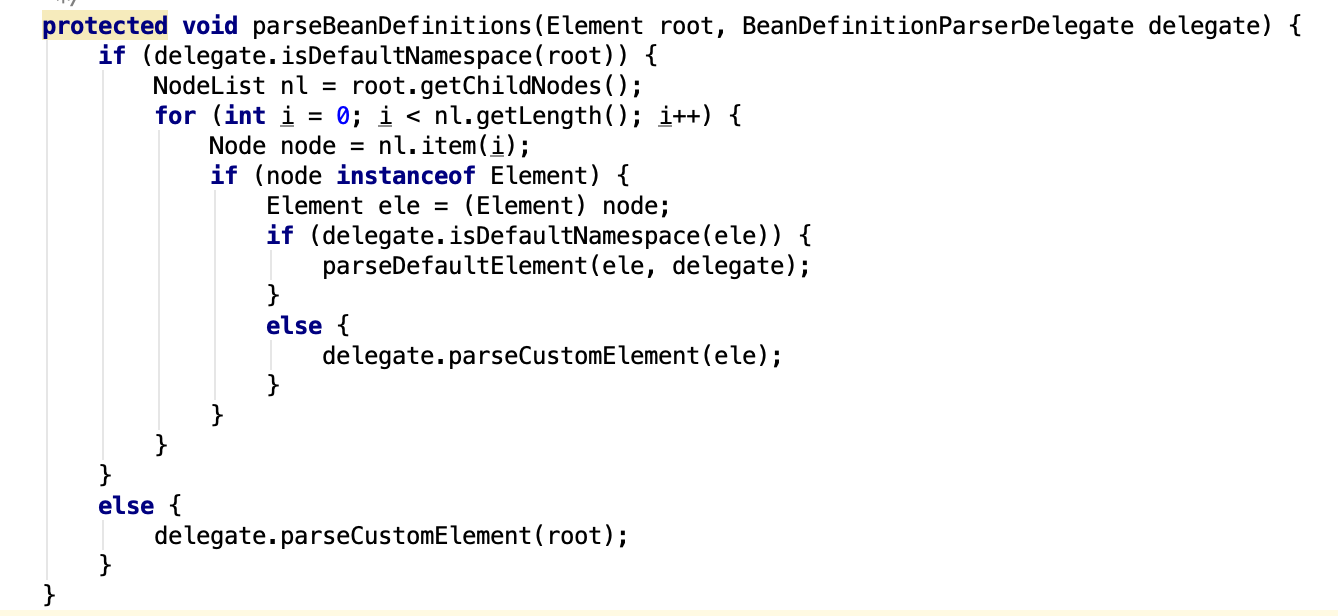
这个过程是通过调用BeanDefinitionRegistry接口的实现来完成的。这个注册过程将载入过程中解析得到的BeanDefinition向IoC容器注册。IoC容器内部将BeanDefinition注入到一个HashMap中，IoC容器就是通过这个HashMap来持有这些BeanDefinition的。



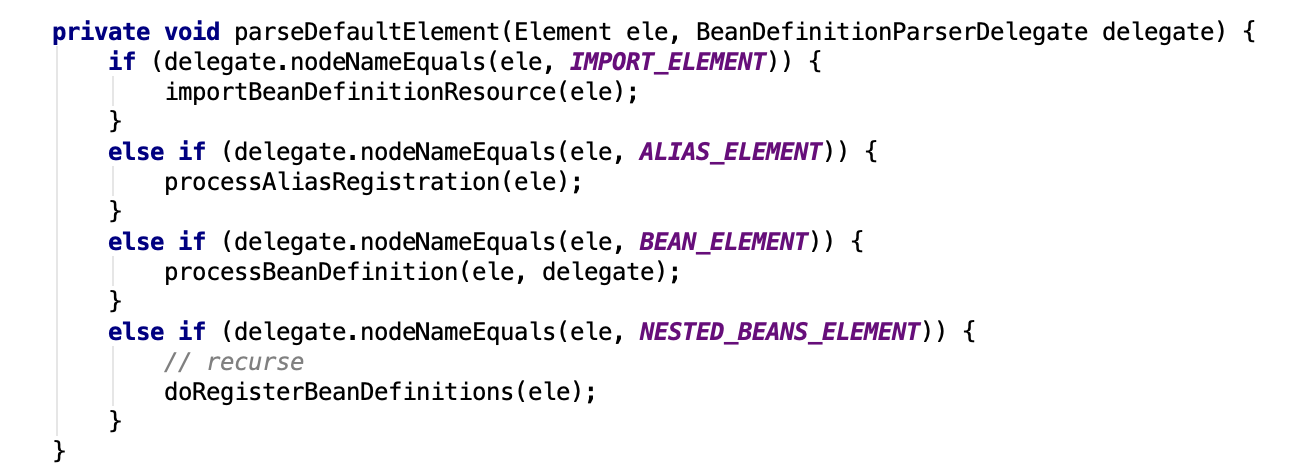
1. AbstractApplicatonContext#refresh:此方法重新定义Spring启动的”提纲”,包含Spring容器启动的一系列工作。
2. AbstractApplicationContext#obtainFreshBeanFacotry:调用refreshBeanFactory和getBeanFactory方法，重新获取一个新的BeanFactory实例。
3. AbstractRefreshableApplicationContext#refreshBeanFactory:销毁原BeanFactory,新建DefaultListableBeanFactory实例，并对其做一些初始化工作
4. AbstractXmlApplicationContext#loadBeanDefinitions:上一步创建BeanFactory实例之后，通过该BeanFactory创建XmlBeanDefinitionReader实例。也就是AppilcationContext上下文将BeanDefinition的定位加载工作交付到了XmlBeanDefinitionReader。继续跟到重载的同名方法loadBeanDefinitions(XmlBeanDefinitionReader reader)，这一步完成了将部分XML文件资源封装为Resoure对象，而用户以XML文件路径传入的BeanDefinition资源文件也会在AbstractBeanDefinitionReader#loadBeanDefinitions(String…locations)逐一封装为Resource对象，并进行进一步的处理。

以上1 – 4 完成了Resource的定位工作

1. 继续跟踪到XmlBeanDefinitionReader#loadBeanDefinitions:这一步主要是通过Resource对象获取XML文件的输入流，继续执行doLoadBeanDefinitions。因为是XML文件格式组织的BeanDefinition，因此AbstractBeanDefinitionReader需要委托到其子类。
2. XmlBeanDefinitionReader#doLoadBeanDefinitions:通过上一步获取到的输入流和Resource对象，获取Document对象。
3. XmlBeanDefinitionReader#registerBeanDefinitions:初始化DefaultBeanDefinitionDocumentReader对象，将上一步得到的Document对象交由DefaultBeanDefinitionDocumentReader对象解析，并统计解析的BeanDefinition个数。
4. DefaultDefinitionDocumentReader#registerBeanDefinitions:获取Document对象的根元素，执行doRegisterBeanDefinitions。
5. 继续跟到DefaultDefinitionDocumentReader#parseBeanDefinitions:初始化BeanDefinitionParserDelegate实例，递归解析Document（为毛我看的是迭代解析呢？？）。DefaultBeanDefinitionDocumentReader负责读入Document对象，实际将Document解析为BeanDefinition的却是BeanDefinitionParserDelegate对象。



1. 根据元素是否是默认的还是自定义的分别进行解析。如果是默认的则调用DefaultDefinitionDocumentReader#parseDefaultElement根据元素的名称进行处理。这里以这里以”bean”为例。



1. 继续跟踪到DefaultDefinitionDocumentReader#processBeanDefinition:将单个Element交由第9步得到的BeanDefinitionParserDelegate对象解析。解析过程较为复杂，有兴趣可以自行跟踪。

以上5 – 11 完成了BeanDefinition的载入工作，得到BeanDefinitionHolder对象。

1. 顺着DefaultDefinitionDocumentReader#processBeanDefinition继续往下看，跟到BeanDefinitionReader#registerBeanDefinition。传入上一步得到的BeanDefinitionHolder对象，将BeanDefinition注册到IoC容器中

以上12 完成的是向IoC容器注册这些BeanDefinition，BeanFactory是以Map的结构组织这些BeanDefinition的。可以在DefaultListableBeanFactory中看到此Map的定义。

