# 第一章Spring IoC容器的实现

## 1 Spring IoC容器概述

#### 1.1 IOC容器和依赖反转模式

IOC控制反转：说的是创建对象实例的控制权从代码控制剥离到IOC容器控制，实际就是你在xml文件控制，侧重于原理。DI依赖注入：说的是创建对象实例时，为这个对象注入属性值或其它对象实例，侧重于实现。

Java程序中的每个业务逻辑至少需要两个或以上的对象来协作完成，这使得每个对象都需要与其合作对象（也就是它所依赖的对象）的引用。如果这个获取过程要靠自身实现，那么将导致代码高度耦合并且难以测试。而IoC的思想是：Spring容器来实现这些相互依赖对象的创建、协调工作。对象只需要关系业务逻辑本身就可以了。从这方面来说，对象如何得到它的协作对象的责任被反转了。Martin Fowler 在2004年初的一篇论文中首次提出的。他总结：控制的什么被反转了？就是：获得依赖对象的方式反转了。

所有的类都会在spring容器中登记。所有的类的创建、销毁都由 spring来控制，也就是说控制对象生存周期的不再是引用它的对象，而是spring。对于某个具体的对象而言，以前是它控制其他对象，现在是所有对象都被spring控制，所以这叫控制反转。

IoC的一个重点是在系统运行中，动态的向某个对象提供它所需要的其他对象。这一点是通过DI（Dependency Injection，依赖注入）来实现的。比如对象A需要操作数据库，以前我们总是要在A中自己编写代码来获得一个Connection对象，有了 spring我们就只需要告诉spring，A中需要一个Connection，至于这个Connection怎么构造，何时构造，A不需要知道。在系统运行时，spring会在适当的时候制造一个Connection，然后像打针一样，注射到A当中，这样就完成了对各个对象之间关系的控制。A需要依赖 Connection才能正常运行，而这个Connection是由spring注入到A中的，依赖注入的名字就这么来的。那么DI是如何实现的呢？Java 1.3之后一个重要特征是反射（reflection），它允许程序在运行的时候动态的生成对象、执行对象的方法、改变对象的属性，Spring就是通过**反射**来实现注入的。

#### 1.2 Spring IOC的应用场景

在Spring中，Spring IoC提供了一个基本的JavaBean容器，通过IoC模式管理依赖关系，并通过依赖注入和AOP切面增强了为JavaBean这样的POJO对象赋予事务管理、生命周期管理等基本功能。

具体的注入实现中，主要的注入方式有：**接口注入**、**setter注入**、**构造器注入**三种。Spring的IoC设计中，主要是setter注入和构造器注入方式；相对而言，使用Spring时常见的注入方式是：setter注入。

## 2 IOC容器

#### 2.1 Spring的IoC容器设计

在Spring IoC容器的设计中，**有两个主要的容器系列**：①实现BeanFactory接口的简单容器系列，只实现了容器的最基本功能；②ApplicationContext应用上下文，它作为容器的高级形态而存在，增加了许多面向框架的特性，对应用环境做了很多适配。

Spring通过定义BeanDefinition来管理基于Spring的应用中的各种对象以及他们之间的相互依赖关系。BeanDeifinition抽像了我们对于Bean的定义，是让容器起作用的主要数据类型。IOC容器是用来管理对象依赖关系的，对IoC容器来说，BeanDefinition就是对依赖反转模式中管理的对象依赖关系的数据抽象，也是容器实现依赖反转功能的核心数据结构，依赖反转功能都是围绕对这个BeanDefinition的处理来完成的。

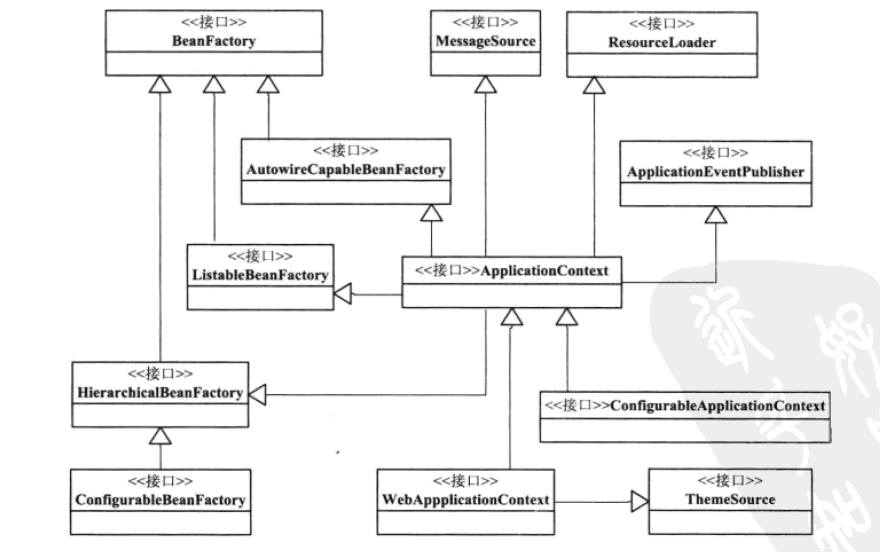


图1.1 IoC容器的接口设计图

**1.从接口BeanFactory到HiearerchialBeanFactory，再到ConfigurableBeanFactory**，是一条主要的BeanFactory设计路径。这条接口设计路径中，BeanFactory接口定义了基本的IOC容器规范。在这个接口定义中，BeanFactory包括了getBean()这样的IoC容器的基本方法（从容器中获取bean）。而HiearerchialBeanFactory接口在继承了BeanFactory的基本接口之后，增加了getParentBeanFactory()的接口功能，使得BeanFactory具备了双亲IoC容器的管理功能。ConfigurableBeanFactory主要定义了对BeanFactory的配置功能，比如setParentBeanFactory()设置双亲IOC容器，通过addBeanPostProcessor()配置Bean的后置处理器等。通过这些接口设计的叠加，定义了BeanFactory就是简单的IOC容器的基本功能。

**2.第二条设计主线是：以ApplicationContext应用上下文接口为核心的接口设计。**这里涉及的主要设计接口有，从BeanFactory到ListableBeanFactory,再到ApplicationContext,再到常用的WebApplicationContext或者ConfigurableApplicationContext接口。项目中常用的应用上下文基本都是ConfigurableApplicationContext或者WebApplicationContext的实现。ListableBeanFactory接口中，细化了许多BeanFactory的接口功能，比如定义了setBeanDefinitionNames接口方法；对于ApplicationContext接口，它通过继承了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。

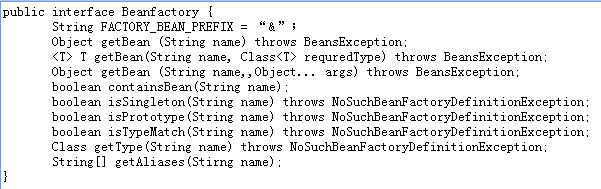
3.**这个接口系统是以BeanFactory和ApplicationContext为核心的**，而BeanFactory又是Ioc容器的最基本的接口，在ApplicationContext的设计中，一方面，可以看到他继承了HiearerchialBeanFactory等BeanFactory的接口，具备了BeanFactory Ioc容器的基本功能，另外一方面，通过继承MessageSource、ResourceLoader、ApplicationEventPublisher这些接口，BeanFactory为ApplicationContext赋予了更高级的Ioc容器特性。对于ApplicationContext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。

#### 2.2 BeanFactory

##### 2.2.1应用场景

BeanFactory接口定义了IoC容器最基本的形式，并且提供了IOC容器所应该遵守的最基本的服务契约，提供了getBean()这样最基本的功能，用来从容器中取得Bean。使用容器时，可以使用转义符“&”来得到FactoryBean本身。例如&myObject得到的是FactoryBean，而不是myObject这个对象。FactoryBean管理所有的Bean，是一个能产生或者修饰对象生成的工厂bean，它的实现与**工厂模式和修饰器模式**类似。

代码清单1.1 BeanFactory方法



**getBean(String name):** 取得IoC容器中管理的bean，bean的取得是通过指定名字来索引的。

**getBean(Class<T> requiredType):** 需要获取的bean是prototype类型的。

**containsBean():** 判断容器是否包含有指定名字的bean。

**isSingleton():** 查询指定名字的bean是否是isSingleton类型的bean。

**isPrototype():** 查询指定名字的bean是否是prototype类型的。

**isTypeMatch():** 查询指定了名字的bean的class类型是否是特定的class类型。

**getType(String name):** 查询指定名字的bean的class类型。

**getAliases():** 查询指定了名字的bean的所有别名。

##### 2.2.2设计原理

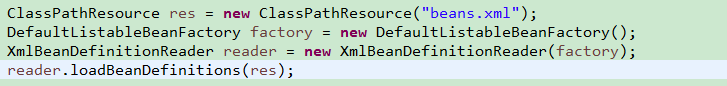
作为一个简单IoC容器系列**最底层实现**的XmlBeanFactory，它只提供个最基本的IoC容器的功能。BeanFactory实现是IoC容器的基本形式，而各种ApplicationContext的实现是IoC容器的高级表现形式。

我们可以发现，最底层的子类都是XmlBeanFactory(现在已经被废弃了)，该容器在**继承了DefaultListableBeanFactory**容器的功能（DefaultListableBeanFactory包含了基本IoC容器所具有的重要功能，在Spring中，被作为一个默认的功能完整的IoC容器来使用）的同时，增加了其他诸如XML读取的附加功能。

**实现XML读取的功能是怎样实现的呢？**对这些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，处理XML形式的信息。构造XMLBeanFactory这个IoC容器时，需要制定beanDefinition的信息来源，而这个信息来源需要封装成Spring中的Resource类来给出。Resource是Spring用来封装I/O操作的类。

##### 2.2.3编程式使用IoC容器

代码清单1.2编程式使用IoC容器



这样我们就可以通过factory对象来使用DefaultListableBeanFactory这个IoC容器了。在使用IoC容器时，需要加如下几个步骤：

①创建IoC配置文件的抽象资源，这个抽象资源包含了BeanDefinition的定义信息；

②创建一个BeanFactory，这里使用DefaultListableBeanFactory。

③创建一个载入BeanDefinition的读取器，这里使用XmlBeanDefinitionReader来载入XML文件形式的BeanDefinition，通过一个回调配置给BeanFacory。

④从定义好的资源位置读入配置信息，具体的解析过程由XmlBeanDefinitionReader来完成。完成整个载入和注册Bean定义后，需要的IOC容器就创立起来了。

#### 2.3 ApplicationContext

##### 2.3.1应用场景

ApplicationContext是一个高级形态意义的IoC容器，可以看到ApplicationContext在BeanFactory的基础上添加了很多附加功能：

①支持不同的信息源。因为扩展了MessageSource接口，可以支持国际化；

②访问资源。这一特性体现在对ResourceLoader和Resource的支持上。

③支持应用事件。继承了接口ApplicationEvenPublisher，从而引入了时间机制，这些事件和

Bean的生命周期的结合为Bean的管理提供了便利。

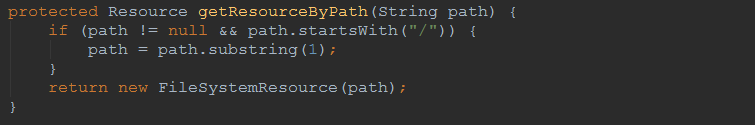
④在ApplicationContext中提供了附加服务。

##### 2.3.2 ApplicationContext的设计原理

FileSystemXMLApplicationContext中，主要功能都在基类**AbstractXMLApplicationContext** 中实现的，FileSystemXMLApplicationContext只需要实现和自身设计相关的两个功能：

1) 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【refresh设计IoC容器启动的一系列复杂操作，不同容器的实现，操作是类似的，在AbstractApplicationContext中实现的】。

2) 这部分与怎样从文件系统中加载XML的Bean定义资源有关。在文件系统中读取以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的方式。FileSystemXMLApplicationContext中通过下面的方法得到FileSystemResource的资源定位。



## 3 IoC容器的初始化过程

IoC容器的初始化是由前面介绍的refresh()方法来启动的，这个方法标志着IoC容器的正式启动。严格来说，**容器的初始化过程主要是包括三个部分：BeanDefinition的Resource定位、BeanDefinition的载入和解析以及BeanDefinition在容器中的注册。**至于容器的依赖注入，其实是发生在用户第一次向容器索要Bean时触发的，即用户调用getBean的时候将触发容器中依赖注入的过程。但是也有例外的时候，也就是我们可以在BeanDefinition中通过对lazy-init属性的设置来让容器完成对Bean的预实例化（其实也就是依赖注入的过程）。

1) Resource定位：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。

2) BeanDefinition载入：这个过程是把用户定义好的Bean表示成IoC容器内部的数据结构，

即BeanDefinition。

3) 向IoC容器注册这些BeanDefinition：这个过程通过调用BeanDefinitionRegistry接口的实现来完成的。在IoC容器内部将BeanDefinition注入到一个HashMap中去。

**注意：**IoC容器初始化过程中，不包含Bean依赖注入的实现，依赖注入一般发生在应用第

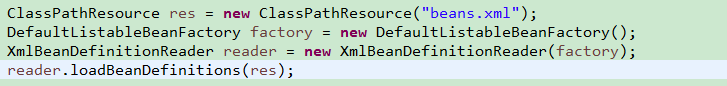
一次通过getBean向容器索取Bean的时候。

#### 3.1 BeanDefinition的Resource定位

BeanDefinition，那么它到底是什么东西呢？从字面上理解，它代表着Bean的定义。其实，它就是完整的描述了在Spring配置文件中定义的节点中所有信息，包括各种子节点。不太恰当地说，我们可以把它理解为配置文件中一个个<bean></bean>节点所包含的信息。

我们将以编程式使用DefaultListableBeanFactory来引入BeanDefinition的Resource定位。先介绍一下DefaultListableBeanFactory：经常要用到的一个IoC容器的实现，比如在设计应用上下文ApplicationContext时就会用到它。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中的一个基本产品。Spring中，实际上是把DefaultListableBeanFactory作为一个默认的功能完整的IoC容器来使用的。

##### 3.1.1 编程式使用DefaultListableBeanFactory容器



在使用IoC容器的时候，需要以下几个步骤：

①首先定义个Resource来定位容器使用的BeanDefinition，这里使用的是ClassPathResource，意味着会在类路径中去寻找以文件形式存在的BeanDefinition信息；

②创建一个BeanFactory，这里使用的是DefaultListableBeanFactory；

③创建一个载入BeanDefinition的读取器，这里使用的是XmlBeanDefinitionReader来载入XML文件形式的BeanDefinition，然后通过一个回调配置给BeanFactory；

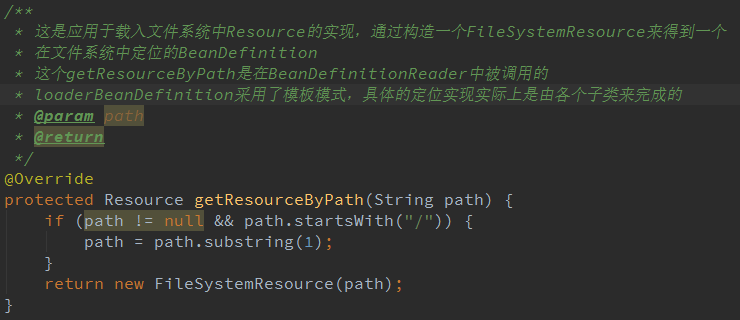
④从定义好的资源位置读取配置信息，具体的解析过程由BeanDefinitionReader来完成。

在这里，使用ApplicationContext相对于直接使用DefaultListableBeanFactory的好处。因为在ApplicationContext中，Spring已经为我们提供了一系列加载不同Resource的读取器的实现，而DefaultListableBeanFactory只是一个纯粹的IoC容器，需要为它提供特定的读取器才能完成这些功能。但是从另一方面来讲，使用DefaultListableBeanFactory这种更底层的容器能提高定制IoC容器的灵活性。

##### 3.1.2 BeanDefinition的Resource定位过程

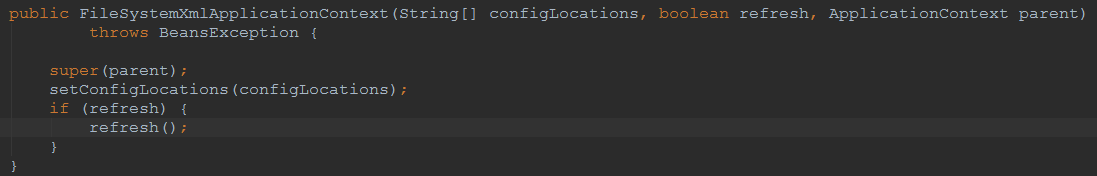
下面将以FileSystemXmlApplicationContext为例对IoC容器初始化过程进行探究。在FileSystemXmlApplicationContext的构造函数中，实现了对Configuration进行处理的能力，让所有配置在文件系统中的，以XML文件方式存在的BeanDefinition都能够得到有效的处理。比如，实现getResourceByPath方法，这个方法是一个模板方法，是为读取Resource服务的。在构造函数中通过refresh()方法来启动IoC容器的初始化。

代码清单1.3 FileSystemXmlApplicationContext中载入Resource的方法



代码清单 1.4 FileSystemXmlApplicationContext中含有refresh方法的构造函数

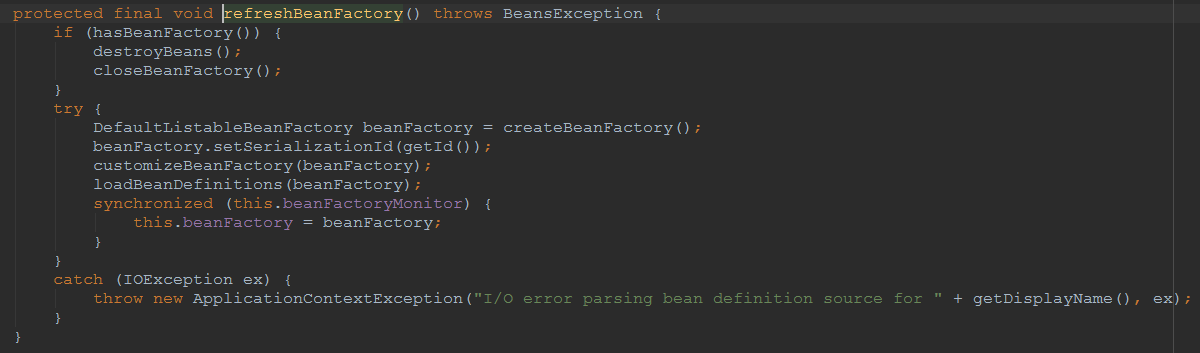
//在对象的初始化过程中，调用refresh函数载入BeanDefinition，这个refresh启动了

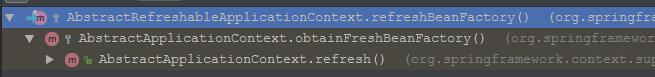


在**FileSystemXmlApplicationContext中不管调用哪个构造函数，最终都是会调用这个包含了refresh方法的构造函数**，因此得出结论：触发对BeanDefinition资源定位过程的refresh方法的调用是在FileSystemXmlApplicationContext的构造函数中启动的。

那么，FileSystemXmlApplication在什么地方定义了BeanDefinition的读入器，从而完成BeanDefinition信息的读入呢？关于这个读入器BeanDefinitionReader的配置，我们接下来到FileSystemXmlAppplicationContext的基类**AbstractRefreshableApplicationContext的refreshBeanFactory**方法去看看。

代码清单1.5 AbstractRefershableApplicationContext中的refreshBeanFactory方法





refresh的调用是在FileSystemXmlApplicationContext的构造函数中启动的，而这个refresh又是从AbstractApplicationContext继承过来的。因此，结合上面这个函数调用关系图我们知道：refreshBeanFactory被FileSystemXmlApplicationContext构造函数中的refresh调用。**在这个方法（refreshBeanFactory）中，①通过createBeanFactory构建了一个IoC容器供ApplicationContext使用**。这个IoC容器就是前面我们提到的DefaultListableBeanFactory，②同时它还**启动了loadBeanDefinition来载入BeanDefinition**，在AbstractRefreshableApplictionContext中，这里是使用BeanDefinitionReader载入Bean定义的地方，因为允许有多种载入方式(虽然常用的是XML定义的形式)，这里通过一个**抽象函数**把具体的实现委托给子类来完成。

refreshBeanFactory方法中调用的loadBeanDefinitions方法其实**最终是调用AbstractBeanDefinitionReader里面的loadBeanDefinitions方法**。

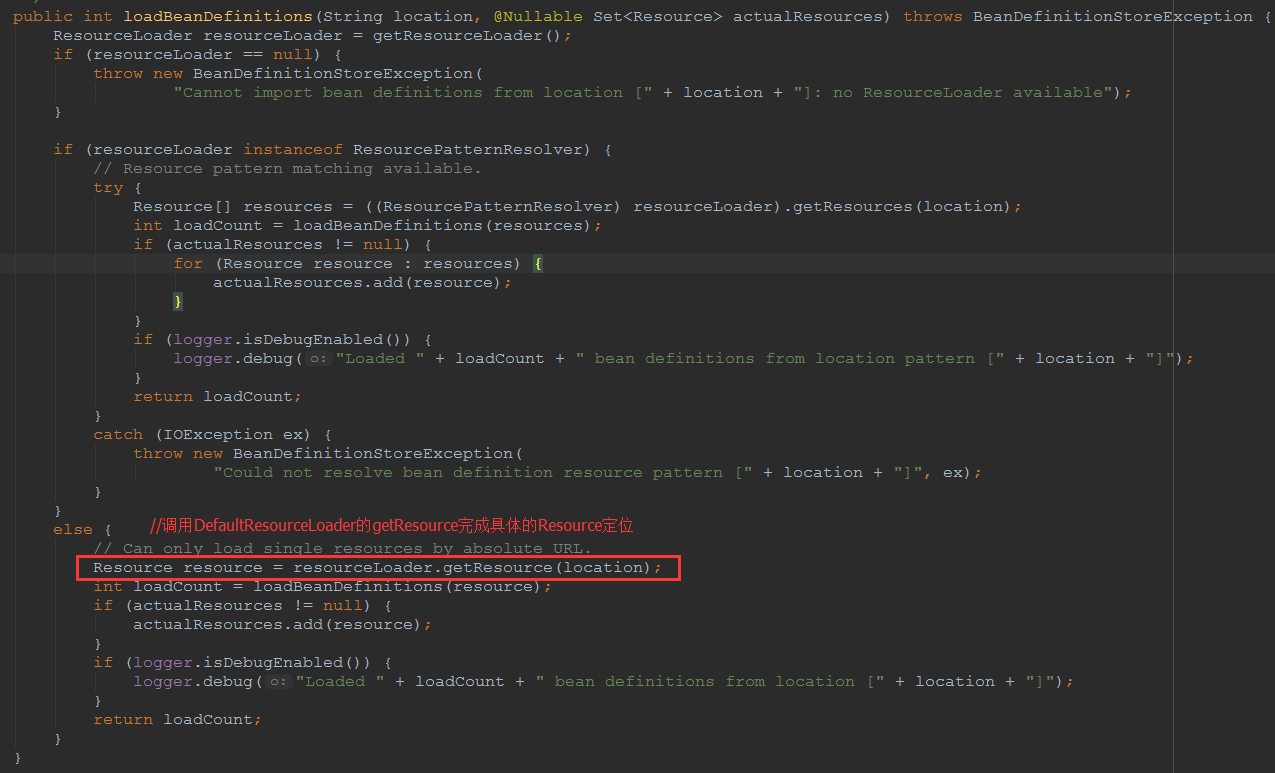


图1.1.9 AbstractBeanDefinitionReader中的loadBeanDefinitions方法

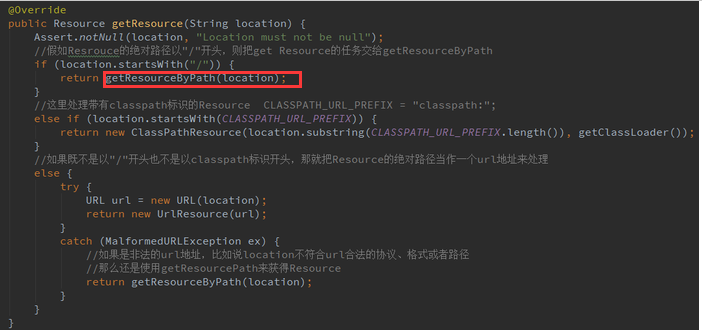
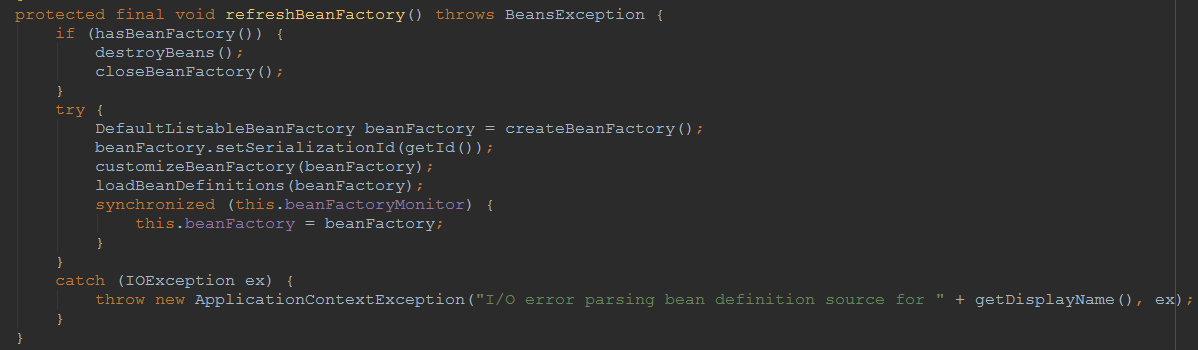
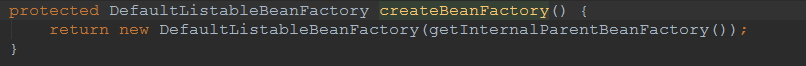


图1.10 DefaultResourceLoader中的getResource方法

图1.1.10中使用的getResourcePath方法将会被FileSystemXmlApplicationContext实现，如图图1.1.2所示。该方法返回的是一个FileSystemResource对象，通过这个对象，Spring可以进行相关的I/O操作，完成BeanDefinition的定位。如果是其他的ApplicationContext，那么会对应生成其他种类的Resource，比如ClassPathResource、ServletContextResource等。所以BeanDefinition的定位到这里就完成了。在BeanDefinition定位完成的基础上，就可以通过返回的Resource对象进行BeanDefinition的载入了。调用了AbstractApplicationContext的refresh()方法调用了obtainFreshBeanFactory()方法，obtainFreshBeanFactory()方法调用了refreshBeanFactory()方法【这是一个抽象方法，具体实现在AbstractRefreshableApplicationContext类中】，通过refreshBeanFactory构建了一个IoC容器(DefaultListableBeanFactory)。





refreshBeanFactory()方法①先判断是否已经建立了BeanFactory，有则销毁并关闭；②创建BeanFactory，这里是DefaultListableBeanFactory；③loadBeanDefinitions加载BeanDefinition的信息。

loadBeanDefinitions()最后会调用AbstractBeanDefinitionReader的loadBeanDefinitions(String location, Set<Resource> actualResources)，调用DefaultResourceLoader#getResource()