# Spring IOC容器

Spring 容器是Spring框架的核心，容器将创建对象，把他们连接在一起，配置他们，并管理他们的整个生命周期，从创建到销毁。Spring容器使用依赖注（DI）来管理组成一个应用程序的组件，这些对象被称为Spring Beans。

通过配置元数据提供的指令，容器知道对那些对象进行实例化，配置和组装。配置元数据可以通过XML，java注释或者java代码来表示。

Spring提供了两种不同类型的容器：

1、Spring BeanFactory 容器

它是最简单的容器，给DI提供了基本支持,它用org.springframework.beans.factory.BeanFactory接口来定义。BeanFactory 或者相关的接口，如 BeanFactoryAware，InitializingBean，DisposableBean，在 Spring 中仍然存在具有大量的与 Spring 整合的第三方框架的反向兼容性的目的。

2、Spring ApplicationContext 容器

该容器添加了更多的企业特定的功能，如从一个属性文件中解析文本信息的能力，发布应用程序事件给感兴趣的事件监听器的能力，由 org.springframework.context.ApplicationContext 接口定义。

ApplicationContext 容器包括 BeanFactory 容器的所有功能，所以通常建议超过 BeanFactory。BeanFactory 仍然可以用于轻量级的应用程序，如移动设备或基于 applet 的应用程序，其中它的数据量和速度是显著。

## Spring BeanFactory 容器

这是一个最简单的容器，它主要的功能是为依赖注入 （DI） 提供支持，这个容器接口在 org.springframework.beans.factory.BeanFactor 中被定义。 BeanFactory 和相关的接口，比如BeanFactoryAware、 DisposableBean、InitializingBean，仍旧保留在 Spring 中，主要目的是向后兼容已经存在的和那些 Spring 整合在一起的第三方框架。

在 Spring 中，有大量对 BeanFactory 接口的实现。其中，最常被使用的是 XmlBeanFactory 类。这个容器从一个 XML 文件中读取配置元数据，由这些元数据来生成一个被配置化的系统或者应用。

在资源宝贵的移动设备或者基于 applet 的应用当中， BeanFactory 会被优先选择。否则，一般使用的是 ApplicationContext，除非你有更好的理由选择 BeanFactory。

在主程序当中，我们需要注意以下两点：

* 第一步利用框架提供的 **XmlBeanFactory()** API 去生成工厂 bean 以及利用 **ClassPathResource()** API 去加载在路径 CLASSPATH 下可用的 bean 配置文件。**XmlBeanFactory()** API 负责创建并初始化所有的对象，即在配置文件中提到的 bean。
* 第二步利用第一步生成的 bean 工厂对象的 **getBean()** 方法得到所需要的 bean。 这个方法通过配置文件中的 bean ID 来返回一个真正的对象，该对象最后可以用于实际的对象。一旦得到这个对象，就可以利用这个对象来调用任何方法。

## Spring ApplicationContext 容器

Application Context 是 spring 中较高级的容器。和 BeanFactory 类似，它可以加载配置文件中定义的 bean，将所有的 bean 集中在一起，当有请求的时候分配 bean。 另外，它增加了企业所需要的功能，比如，从属性文件中解析文本信息和将事件传递给所指定的监听器。这个容器在 org.springframework.context.ApplicationContext interface 接口中定义。

ApplicationContext 包含 BeanFactory 所有的功能，一般情况下，相对于 BeanFactory，ApplicationContext 会更加优秀。当然，BeanFactory 仍可以在轻量级应用中使用，比如移动设备或者基于 applet 的应用程序。

最常被使用的 ApplicationContext 接口实现：

* **FileSystemXmlApplicationContext**：该容器从 XML 文件中加载已被定义的 bean。在这里，你需要提供给构造器 XML 文件的完整路径。
* **ClassPathXmlApplicationContext**：该容器从 XML 文件中加载已被定义的 bean。在这里，你不需要提供 XML 文件的完整路径，只需正确配置 CLASSPATH 环境变量即可，因为，容器会从 CLASSPATH 中搜索 bean 配置文件。
* **WebXmlApplicationContext**：该容器会在一个 web 应用程序的范围内加载在 XML 文件中已被定义的 bean。

在主程序当中，我们需要注意以下两点：

* 第一步生成工厂对象。加载完指定路径下 bean 配置文件后，利用框架提供的 FileSystemXmlApplicationContext API 去生成工厂 bean。FileSystemXmlApplicationContext 负责生成和初始化所有的对象，比如，所有在 XML bean 配置文件中的 bean。
* 第二步利用第一步生成的上下文中的 getBean() 方法得到所需要的 bean。 这个方法通过配置文件中的 bean ID 来返回一个真正的对象。一旦得到这个对象，就可以利用这个对象来调用任何方法。

## Spring Bean定义

被称作 bean 的对象是构成应用程序的支柱也是由 Spring IoC 容器管理的。bean 是一个被实例化，组装，并通过 Spring IoC 容器所管理的对象。这些 bean 是由用容器提供的配置元数据创建的，例如，已经在先前章节看到的，在 XML 的表单中的 定义。

bean 定义包含称为**配置元数据**的信息，下述容器也需要知道配置元数据：

* 如何创建一个 bean
* bean 的生命周期的详细信息
* bean 的依赖关系

上述所有的配置元数据转换成一组构成每个 bean 定义的下列属性。

|  |  |
| --- | --- |
| **属性** | **描述** |
| class | 这个属性是强制性的，并且指定用来创建 bean 的 bean 类。 |
| name | 这个属性指定唯一的 bean 标识符。在基于 XML 的配置元数据中，你可以使用 ID 和/或 name 属性来指定 bean 标识符。 |
| scope | 这个属性指定由特定的 bean 定义创建的对象的作用域，它将会在 bean 作用域的章节中进行讨论。 |
| constructor-arg | 它是用来注入依赖关系的，并会在接下来的章节中进行讨论。 |
| properties | 它是用来注入依赖关系的，并会在接下来的章节中进行讨论。 |
| autowiring mode | 它是用来注入依赖关系的，并会在接下来的章节中进行讨论。 |
| lazy-initialization mode | 延迟初始化的 bean 告诉 IoC 容器在它第一次被请求时，而不是在启动时去创建一个 bean 实例。 |
| initialization 方法 | 在 bean 的所有必需的属性被容器设置之后，调用回调方法。它将会在 bean 的生命周期章节中进行讨论。 |
| destruction 方法 | 当包含该 bean 的容器被销毁时，使用回调方法。它将会在 bean 的生命周期章节中进行讨论。 |

**Spring 配置元数据**

Spring IoC 容器完全由实际编写的配置元数据的格式解耦。有下面三个重要的方法把配置元数据提供给 Spring 容器：

* 基于 XML 的配置文件。
* 基于注解的配置
* 基于 Java 的配置

## Spring Bean的作用域

当在 Spring 中定义一个 bean 时，你必须声明该 bean 的作用域的选项。例如，为了强制 Spring 在每次需要时都产生一个新的 bean 实例，你应该声明 bean 的作用域的属性为 **prototype**。同理，如果你想让 Spring 在每次需要时都返回同一个bean实例，你应该声明 bean 的作用域的属性为 **singleton**。

Spring 框架支持以下五个作用域，如果你使用 web-aware ApplicationContext 时，其中三个是可用的。

|  |  |
| --- | --- |
| **作用域** | **描述** |
| singleton | 在spring IoC容器仅存在一个Bean实例，Bean以单例方式存在，默认值 |
| prototype | 每次从容器中调用Bean时，都返回一个新的实例，即每次调用getBean()时，相当于执行newXxxBean() |
| request | 每次HTTP请求都会创建一个新的Bean，该作用域仅适用于WebApplicationContext环境 |
| session | 同一个HTTP Session共享一个Bean，不同Session使用不同的Bean，仅适用于WebApplicationContext环境 |
| global-session | 一般用于Portlet应用环境，该运用域仅适用于WebApplicationContext环境 |

本章将讨论前两个范围，当我们将讨论有关 web-aware Spring ApplicationContext 时，其余三个将被讨论。

**singleton 作用域：**

当一个bean的作用域为Singleton，那么Spring IoC容器中只会存在一个共享的bean实例，并且所有对bean的请求，只要id与该bean定义相匹配，则只会返回bean的同一实例。

Singleton是单例类型，就是在创建起容器时就同时自动创建了一个bean的对象，不管你是否使用，他都存在了，每次获取到的对象都是同一个对象。注意，Singleton作用域是Spring中的缺省作用域。你可以在 bean 的配置文件中设置作用域的属性为 singleton

**prototype 作用域**

当一个bean的作用域为Prototype，表示一个bean定义对应多个对象实例。Prototype作用域的bean会导致在每次对该bean请求（将其注入到另一个bean中，或者以程序的方式调用容器的getBean()方法）时都会创建一个新的bean实例。Prototype是原型类型，它在我们创建容器的时候并没有实例化，而是当我们获取bean的时候才会去创建一个对象，而且我们每次获取到的对象都不是同一个对象。根据经验，对有状态的bean应该使用prototype作用域，而对无状态的bean则应该使用singleton作用域。

为了定义 prototype 作用域，你可以在 bean 的配置文件中设置作用域的属性为 prototype