**2.6  小结**

在本章中，我们紧密地结合 Spring 的源代码，对容器的实现原理进行了详细的分析，旨在为读者整理出一条清晰的线索。其中包括 IoC 容器和上下文的基本工作原理、容器的初始化过程、依赖注入的实现，等等。总的来说，关于容器的基本工作原理，我们可以大致地整理出以下几个方面：

BeanDefinition的定位。对IoC容器来说，它为我们管理POJO之间的依赖关系提供了帮助，但客户也需要依据 Spring 的定义规则提供 Bean 定义信息，我们可以使用各种形式的Bean定义信息，其中常用的是使用XML的文件格式。在Bean定义方面，Spring为用户提供了很大的灵活性。在初始化 IoC 容器的过程中，首先需要定位到这些有效的 Bean定义信息，这里Spring使用Resource这个接口来统一这些Bean定义信息，而这个定位由 ResourceLoader 来完成。如果使用上下文，ApplicationContext 本身就为客户提供了定位的功能。因为上下文本身就是 DefaultResourceLoader 的子类。如果使用基本的BeanFactory 作为IoC 容器，客户需要做的额外工作就是为BeanFactory 指定相应的Resource来完成Bean信息的定位。

容器的初始化。在使用上下文时，需要一个对它进行初始化的过程，完成初始化以后，这个IoC容器才是可用的。这个过程的入口是在refresh中实现的，这个refresh相当于容器的初始化函数。在初始化过程中，比较重要的部分是对 BeanDefinition 信息的载入和注册工作。相当于在 IoC 容器中需要建立一个 BeanDefinition 定义的数据映像，Spring 为了达到载入的灵活性，把载入的功能从 IoC 容器中分离出来，由BeanDefinitionReader 来完成 Bean 定义信息的读取、解析和 IoC 容器内部BeanDefinition 的 建 立 。 在 DefaultListableBeanFactory 中 ， 这 些BeanDefinition被维护在一个HashMap中，以后的IoC容器对Bean的管理和操作就是通过这些建立起来的BeanDefinition来完成的。

在容器初始化完成以后，IoC 容器的使用就准备好了，这时只是在 IoC 容器内部建立了BeanDefinition，具体的依赖关系还没有注入。在客户第一次向IoC容器请求Bean时，IoC容器对相关的 Bean 依赖关系进行注入。如果需要提前注入，客户通过 lazy-init 属性可以进行预实例化，这个预实例化是上下文初始化的一部分，起到提前完成依赖注入的控制作用。在依赖注入完成以后，IoC 容器就会保持这些具备依赖关系的 Bean 让客户来直接使用。这时可以通过getBean 来取得 Bean，这些 Bean 不是简单的 Java 对象，而是已经包含了对象之间依赖关系的Bean，尽管这些依赖注入的过程对用户来说是不可见的。

在对 IoC 容器的分析中，我们重点讲解了 BeanFactory 和 ApplicationContext 体系、ResourceLoader、refresh 初始化、容器的 loadBeanDefinition 和注册、容器的依赖注入、预实例化和FactoryBean 的工作原理等。通过对这些实现过程的深入分析，我们可以初步了解IoC容器的基本工作原理和它的基本特性的实现思路。了解了IoC容器的基本实现原理后，我们对容器的其他特性的实现原理也进行了分析。这些特性包括 init-lazy 预实例化、BeanFactory、Bean 后置处理器以及 autowiring 特性的实现。这些特性对我们更灵活和更有技巧地使用IoC容器有很大的帮助。但是，由于Spring IoC容器的内涵特性非常丰富，这里并没有对其工作原理进行面面俱到的分析。如果读者感兴趣，可以参考本章的分析方法和思路，对自己感兴趣的内容继续进行分析。