**7. The IoC container**

**7.1 Introduction to the Spring IoC container and beans**

本章介绍了Spring框架控制反转（IoC）的实现原理。IoC也称为依赖注入（DI）。 依赖注入是描述一个Object如何定义其依赖对象(需要用到的其它Object)的过程，可以通过构造函数传递参数，给工厂方法传递参数，在构造完实例或者工厂方法返回实例后给实例设置依赖属性。然后容器在创建bean时注入这些依赖项。这个过程与bean自己通过构造函数控制实例化和依赖或一些其它机制如服务定位器模式是基本相反的，因此命名控制反转（IoC）。

org.springframework.beans和org.springframework.context包是Spring Framework的IoC容器的基。BeanFactory接口提供了一个能够管理任何类型对象的高级配置方式。ApplicationContext是BeanFactory的子接口。 它能更容易地和Spring 的AOP，message resource handling（用于国际化），event publication，以及应用层的特定上下文集成（如在Web应用程序中使用的WebApplicationContext）。

简而言之，BeanFactory提供了配置框架和基本功能，而ApplicationContext增加了更多的针对企业级的特定功能。 ApplicationContext是BeanFactory超集，在本章节中将用来专门描述Spring的IoC容器。 有关如何使用BeanFactory而非ApplicationContext的信息请参考7.16节“BeanFactory”。

在Spring里，由Spring IoC 容器管理的构成应用程序主干的Objects被称为bean。 bean是一个由Spring IoC容器实例化，装配或者管理的对象。 否则，bean只是应用程序中的众多简单对象之一。 Beans和它们之间的依赖关系反映在容器中使用的元数据配置里。

**7.2 Container overview**

org.springframework.context.ApplicationContext接口代表一个Spring IoC容器，负责Object实例化，配置和组装上述的beans。容器通过读取元数据配置获取如何实例化，配置和组合对象的信息。 配置元数据可以使用XML，Java注解或Java代码描述。 通过它可以描述组成应用程序的Objects和它们之间复杂的相互依赖关系。

Spring提供了一些现成的实现了ApplicationContext接口的容器。 在独立应用程序中比较常见的是，用ClassPathXmlApplicationContext或FileSystemXmlApplicationContext来创建一个容器实例。 除了采用传统格式XML来定义元数据配置，你也可以通过Java注解或代码作为元数据格式来配置，只需要提供少量的XML配置来声明支持这些额外的元数据格式。

在大多数应用场景中，用户不需要在代码里显式地实例化一个或多个Spring IoC容器。 例如，在Web应用场景中，在应用程序的web.xml文件中只需要引用一段八行（或更多）的XML代码通常就足够了(具体参考 “Convenient ApplicationContext instantiation for web applications” 章节).如果你正在使用Spring工具套件，Eclipse集成开发环境，只需点击几下鼠标或按些快捷键就可以很容易地创建配置文件模板。

下面是一张介绍了Spring如何工作的高级视图。 在创建和初始化ApplicationContext之后你应用程序的Objects和元数据配置会拼装在一起，然后就组成了一个可执行的系统或应用程序。



*Figure 7.1. The Spring IoC container*

**Configuration metadata**

如上图所示，Spring IoC容器需要某种形式的元数据配置;这些元数据配置是您作为应用程序开发人员告诉Spring容器如何实例化，配置和装配应用程序中的对象。

注意

基于XML的元数据不是配置元数据的唯一形式。 Spring IoC容器本身与此配置元数据的形式完全无关的，现在许多开发者选择用Java-based configuration来配置Spring 应用程序。

有关在Spring容器中使用其他形式的元数据请参考：

Annotation-based configuration: Spring 2.5引入了支持基于注解的元数据配置形式。

Java-based configuration: 从Spring 3.0开始，Spring JavaConfig项目提供的许多功能成为Spring核心框架的一部分。 因此，你可以使用Java代码而不需要XML文件给应用程序定义bean。 要使用这些新功能，请参阅@Configuration，@Bean，@Import和@DependsOn注解。

Spring配置通常由多个bean或者至少一个bean组成由容器管理。 基于XML的元数据配置显示在顶级元素<beans />中的<bean />元素里。Java配置通常使用包含在@Configuration类里的@Bean注解方法中。

这些bean的定义对应的Objects组成了你的应用程序。 通常你定义服务层对象，数据访问对象（DAO），表现层对象（如Struts Action）实例，基础架构对象（如Hibernate SessionFactories，JMS队列等）。通常情况下，不会在容器中配置细粒度的域对象，因为这通常是DAO的责任根据业务逻辑来创建和加载域对象。 但是，您可以将AspectJ集成到Spring来配置在IoC容器控制外创建的Objects。 请参考Using AspectJ to dependency-inject domain objects with Spring.

以下示例显示了基于XML的元数据配置的基本结构：



id属性是用来标识单个bean定义的字符串。 class属性定义bean的类型，使用完全限定的类名。 id属性的值可以被关联对象引用。 这段XML示例中没有显示关联对象引用的实例; 更多信息请参考Dependencies。

**Instantiating a container**

实例化一个Spring IoC容器非常简单直接。 给ApplicationContext构造函数传递的一个或多个位置路径实际上是让容器从众多的外部资源如本地文件系统，Java CLASSPATH等等中加载元数据配置



注意

在了解了Spring的IoC容器后，您可能想了解更多关于Spring资源的抽象概念，如第8章Resources中所述，它提供了一个非常方便的机制从URI语法中定义的位置读取InputStream。 尤其是如第8.7节所述，Resource路径用于构建应用程序上下文，“Application contexts and Resource paths”.

以下示例展示了服务层Objects（services.xml）的配置文件：

以下示例展示了数据访问Objects daos.xml的文件



在上述示例中，服务层由PetStoreServiceImpl和两个数据访问类型的Objects JpaAccountDao和JpaItemDao的对象（基于JPA对象/关系映射标准）。 属性名称元素是指JavaBean属性的名称，而ref元素是指另一个bean定义的名称。 id和ref之间的连接表示这两个关联对象之间的依赖关系。 有关对象的依赖关系的详细配置信息，请参考Dependencies.

**Composing XML-based configuration metadata**

在不同的XML文件中定义Bean是很有用的。 通常每个单独的XML配置文件表示应用架构中不同的逻辑层或模块。

你可以通过应用程序上下文的构造函数从这些XML片段中加载Bean的定义。如上一节所示该构造函数可以传递多个资源位置。或者也可以使用一个或多个<import />元素来从另一个文件加载bean定义文件。 例如：



在上述示例中，从三个文件（services.xml，messageSource.xml和themeSource.xml）中加载外部Bean定义。所有位置路径都是相对于导入定义的文件，所以services.xml必须与导入文件在相同的目录或类路径位置，而messageSource.xml和themeSource.xml必须在导入文件的下方的resources目录下。 显而易见的是，以斜杠起始的路径中斜杠被忽略，这些路径都是相对的，因此最好不要使用斜线开头。包括顶级元素<beans />在内的被这些导入文件的内容，必须是根据Spring Schema定义的有效XML bean。

注意

你可以用相对路径”../”来引用父目录中的文件，但不推荐这使用。这样做会造成对当前应用程序之外的文件的依赖。尤其是，这种引用不推荐用于”classpath：” 类型的URLs（例如，”classpath：../services.xml”），在运行时解析过程中选择最近的classpath root 然后才查找其父目录。 更改Classpath配置可能导致选择了另外一个错误的目录。你完全可以使用全限定资源路径而不用相对路径：例如“file:C:/config/services.xml“或”classpath：/config/services.xml“，但请注意，这样做会使你的应用程序和特定的绝对路径耦合。 通常最好在这些绝对位置中加一个间接符号，如”$ {...}”占位符，在运行时针对JVM系统属性进行解析。

import指令是由beans namespace本身提供的一个功能。 除了简单的bean定义，Spring还提供了XML namespaces 来选择其他配置功能。如“context”和“util”的namespace.

**The Groovy Bean Definition DSL**

As a further example for externalized configuration metadata, bean definitions can also be expressed in Spring’s Groovy Bean Definition DSL, as known from the Grails framework. Typically, such configuration will live in a ".groovy" file with a structure as follows:



这种配置方式和XML方式有很大的相似之处，而且也支持Spring的XML 配置的namespaces.也允许用户通过"importBeans"指令导入XML 格式的Bean 定义文件。

**Using the container**

ApplicationContext是一个高级的factory接口能够维护注册不同的bean以及他们的依赖关系，通过T getBean(String name, Class<T> requiredType) 能够获取到你定义bean的实例。

ApplicationContext可以让你能够读取bean的定义和像下面这样访问它们



使用Groovy配置，配置方法非常相似，唯一的区别是context的实现类不同，它可以处理Groovy(同时也能处理XML形式的bean 定义):



GenericApplicationContext和Reader Delegate组合非常灵活比如：用XMLBeanDefinitionReader来读取XML文件：



如果需要的话这些Reader delegates可以在同一个ApplicationContext上混合使用来匹配读取多个不同配置源上的bean定义。

然后可以用getBean方法获取定义的bean的实例。ApplicationContext接口还有一些其他方法来获取beans，但是理想情况下你的应用代码不应该使用它们。

你的应用代码应该避免调用getBean()方法从而完全不依赖Spring的API。例如，spring为集成Web框架提供了很多web框架组件的依赖注入，比如controllers JSF-managed beans，允许你通过元数据给一个bean声明依赖（如，一个autowring 注解）

**7.3 Bean overview**

Spring IoC 容器管理一个或多个bean，这些bean是根据你提供给容器的元数据配置创建的，如XML中的<bean/>格式的定义

在容器自身内部，这些bean定义是由BeanDefinition对象表示的，包含（以及其他信息）以下元数据：

* 带全限定包的类名：通常是被定义Bean的实际实现类
* Bean行为配置元素：表名Bean在容器中的行为（如作用域，生命周期，回调等等）
* 为了正常工作需要引用的其它Bean：这些引用也成为协作者或依赖
* 新建对象需要设置的其它配置参数：如管理连接池的Bean的连接数或数量的限制

这些元数据转换为组成Bean定义的一组属性

*Table 7.1. The bean definition*

|  |  |
| --- | --- |
| Property | Explained in… |
| class | the section called “Instantiating beans” |
| name | the section called “Naming beans” |
| scope | Section 7.5, “Bean scopes” |
| constructor arguments | the section called “Dependency Injection” |
| properties | the section called “Dependency Injection” |
| autowiring mode | the section called “Autowiring collaborators” |
| lazy-initialization mode | the section called “Lazy-initialized beans” |
| initialization method | the section called “Initialization callbacks” |
| destruction method | the section called “Destruction callbacks” |

ApplicationContext 的实现类除了包含如何创建特定Bean的信息外，也允许注册用户在容器外创建的类。这是通过调用ApplicationContext的getBeanFactory()方法来访问BeanFactory的，这个方法会返回BeanFactory的实现类DefaultListableBeanFactory，DefaultListableBeanFactory通过调用registerSingleton(..)和registerBeanDefinition(..)实现注册，但是通常应用程序只用那些通过Bean定义的Bean工作。

注意

Bean元数据和手动提供的单例实例应当尽早注册，以便容器能够在自动导入期间和其它检查步骤的时候正确的处理。虽然在一定程度上支持覆盖已有的元数据和已存在的单例实例，但官方没有正式支持在运行期间（同时访问factory）注册新的Bean，这可能导致并发访问的异常，或者Bean容器的状态不一致。

**Naming beans**

每个Bean都有一个或多个标识符，这些标识符在持有这个Bean的容器中必须是唯一的，一个Bean通常只有一个标识符，但是如果要求有多个，其他的可以认为是他的别名。

在XML形式的配置元数据中，你可以用id和name属性指定Bean的标识符。id属性允许你指定一个id，通常这些名字是字母（'myBean', 'fooService', 等等）但是也可以包含特殊字符。如果你想给这些Bean增加其他别名，你也可以在name属性中指定他们，并以逗号分号或者空格分隔。在Spring3.1之前的版本中，id属性被定义成一个 xsd:ID 类型限制了某些字符，从3.1版本开始id被定义成xsd:string 类型，注意容器依然强制要求Bean id的唯一性，而不是XML解释器的限制。

你不需要必须为Bean提供一个name或者id，如果没有指定name或者id，容器会为这个Bean生成一个唯一的name。但是如果你想通过使用ref元素或者Service Locator 模式查找，用name来引用这个Bean，你必须提供一个name。不提供name的动机往往和用内部类和自动导入写作者（autowiring collaborators）有关。

注意

在扫描classpath路径中的组件时Spring会给未命名的Bean生成一个name，遵循上述的规则，基本上会使用简单的类名，并把它的初始字符转为小写，但是一些特殊情况下，the first and second characters are above Case, the original packaging is saved. These are the same rules as defined java.beans.Introspector.decapitalize (Spring is used here).

**Aliasing a bean outside the bean definition**

在Bean定义里你可以通过用id属性和name组合为Bean提供不止一个name，