# Core:

IoC容器，Events(事件)，Resources(资源)，i18n(国际化)，Validation(验证)，Data Binding(数据绑定)，Type Conversion(类型转换)，SpEL(Spring表达式语言,是比JSP的EL更强大的一种表达式语言)，AOP(面向切面编程)。

这部分参考文档涵盖了Spring Framework绝对不可或缺的所有技术。

其中最重要的是Spring Framework的控制反转（IoC）容器。对Spring框架的IoC容器进行彻底的处理之后，紧接着是对Spring的面向切面编程（AOP）技术的全面覆盖。Spring Framework有自己的AOP框架，它在概念上易于理解，并且成功地解决了Java企业编程中AOP要求的最佳点的80%。

还提供了Spring与AspectJ（目前最丰富的-在功能方面-当然也是Java企业领域中最成熟的AOP实现）集成的覆盖范围。

## 1. IoC容器

本章介绍Spring的控制反转（IoC）容器。

### 1.1 Spring IoC容器和Beans简介

本章介绍了控制反转（IoC）原理的Spring Framework实现。IoC也称为依赖注入（DI）。这是一个过程，通过这个过程，对象仅仅通过构造函数参数，工厂方法的参数或对象通过构造方法实例话后设置的属性或从工厂方法返回后在来定义它们的依赖关系（即:它们使用的其他对象）。然后容器在创建bean时注入这些依赖项。此过程基本上是bean本身通过使用类的直接构造或诸如服务定位器模式的机制来控制其依赖关系的实例化或位置的逆过程（因此名称，控制反转）。

在org.springframework.beans和org.springframework.context包是Spring框架的IoC容器的基础。该 [BeanFactory](https://docs.spring.io/spring-framework/docs/5.1.9.RELEASE/javadoc-api/org/springframework/beans/factory/BeanFactory.html) 接口提供了一种能够管理任何类型对象的高级配置机制。 [ApplicationContext](https://docs.spring.io/spring-framework/docs/5.1.9.RELEASE/javadoc-api/org/springframework/context/ApplicationContext.html) 是BeanFactory的一个子接口。它补充了：

* 更容易与Spring的AOP功能集成
* 消息资源处理（用于国际化）
* 事件发布
* 应用程序层特定的上下文，例如在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，ApplicationContext添加了更多特定于企业的功能。ApplicationContext是BeanFactory的完整的超集，BeanFactory在本章中仅用于Spring的IoC容器的描述。有关使用BeanFactory而不是ApplicationContext,可以看 [BeanFactory](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html" \l "beans-beanfactory)更多信息。

在Spring中，构成应用程序主干并由Spring IoC容器管理的对象称为bean。bean是一个由Spring IoC容器实例化，组装和管理的对象。另一方面，bean只是应用程序中许多对象之一。Bean及其依赖之间的关系反映在容器使用的配置元数据中。

### 1.2 Container概览

org.springframework.context.ApplicationContext接口代表Spring IoC容器，负责实例化，配置和组装bean。容器通过读取配置元数据获取有关要实例化，配置和组装的对象的指令。以XML，Java注解或Java代码配置元数据。它允许您表达组成应用程序的对象以及这些对象之间丰富的相互依赖性。

Spring提供了ApplicationContext接口几种实现。在独立应用程序中，通常会创建一个[ClassPathXmlApplicationContext](https://docs.spring.io/spring-framework/docs/5.1.9.RELEASE/javadoc-api/org/springframework/context/support/ClassPathXmlApplicationContext.html)

或 [FileSystemXmlApplicationContext](https://docs.spring.io/spring-framework/docs/5.1.9.RELEASE/javadoc-api/org/springframework/context/support/FileSystemXmlApplicationContext.html)的实例。

虽然XML是一直是定义配置元数据的传统格式，但您可以通过提供少量XML配置来声明容器使用Java注释或代码作为元数据格式，以声明方式启用对这些其他元数据格式的支持。

在大多数应用程序方案中，不需要显式用户代码来实例化Spring IoC容器的一个或多个实例。例如，在Web应用程序场景中，在应用程序 的web.xml文件中的简单八行（左右）样板Web描述符XML通常就足够了（请参阅[Web应用程序的便捷ApplicationContext实例](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#context-create)）。如果您使用 [Spring Tool Suite](https://spring.io/tools/sts)（基于Eclipse的开发环境），只需点击几下鼠标或按键即可轻松创建此样板配置。

下图显示了Spring如何工作的高级视图。您的应用程序类与配置元数据相结合，以便在ApplicationContext创建和初始化之后，您拥有完全配置且可执行的系统或应用

程序。

*图1. Spring IoC容器*

#### 1.2.1 配置元数据

如上图所示，Spring IoC容器使用一种配置元数据。此配置元数据表示您作为应用程序开发人员告诉Spring容器在应用程序中如何实例化，配置和组装对象。

传统上，配置元数据以简单直观的XML格式提供，本章的大部分内容用于传达Spring IoC容器的关键概念和功能。

基于XML的元数据不是唯一允许的配置元数据形式。Spring IoC容器本身完全与实际编写此配置元数据的格式分离。目前，许多开发人员为其Spring应用程序选择 [基于Java的配置](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-java)。

有关在Spring容器中使用其他形式的元数据的信息，请参阅：

* [基于注解的配置](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-annotation-config)：Spring 2.5引入了对基于注解的配置元数据的支持。
* [基于Java的配置](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-java)：从Spring 3.0开始，项目提供的许多功能Spring JavaConfig成为核心Spring Framework的一部分。因此，您可以使用Java而不是XML文件在应用程序类外部定义bean。要使用这些新功能，请参阅 [@Configuration](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Configuration.html)， [@Bean](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Bean.html)， [@Import](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Import.html)，和[@DependsOn](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/DependsOn.html)注释。

Spring配置包含容器必须管理的至少一个且通常不止一个bean定义。基于XML的配置元数据将这些bean配置为顶级元素<beans/>内的<bean/>元素。Java配置通常在带有@Configuration类中使用@Bean注释方法。

这些bean定义对应于构成应用程序的实际对象。通常，您定义服务层对象，数据访问对象（DAO），表示对象（如Struts Action实例），基础结构对象（如Hibernate SessionFactories，JMS Queues等）。通常，不会在容器中配置细粒度域对象，因为DAO和业务逻辑通常负责创建和加载域对象。但是，您可以使用Spring与AspectJ的集成来配置在IoC容器控制之外创建的对象。请参阅[使用AspectJ使用Spring依赖注入域对象](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#aop-atconfigurable)。

以下示例显示了基于XML的配置元数据的基本结构：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  https://www.springframework.org/schema/beans/spring-beans.xsd">  <bean id="..." class="...">  *<!-- collaborators and configuration for this bean go here -->*  </bean>  <bean id="..." class="...">  *<!-- collaborators and configuration for this bean go here -->*  </bean>  *<!-- more bean definitions go here -->*  </beans> |
| 该id属性是一个标识单个bean定义的字符串。  该class属性定义bean的类型并使用完全限定的类名。 |

id属性的值指的是协作对象。在此示例中未显示用于引用协作对象的XML。有关更多信息，请参阅 [依赖项](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-dependencies)。

#### 1.2.2 实例化容器

提供给ApplicationContext构造函数的位置路径或路径是资源字符串，它允许容器从各种外部资源（如本地文件系统，Java类路径-CLASSPATH等）加载配置元数据。

|  |
| --- |
| ApplicationContext context = **new** ClassPathXmlApplicationContext("services.xml", "daos.xml"); |

在了解了Spring的IoC容器之后，您可能想要了解有关Spring Resource抽象的更多信息 （如[参考资料中所述](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html" \l "resources)），它提供了一种从URI语法中定义的位置读取InputStream的便捷机制。特别是， Resource路径用于构建应用程序上下文，如[应用程序上下文和资源路径中所述](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#resources-app-ctx)。

以下示例显示了服务层对象(services.xml)配置文件：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  https://www.springframework.org/schema/beans/spring-beans.xsd">  *<!-- services -->*  <bean id="petStore" class="org.springframework.samples.jpetstore.services.PetStoreServiceImpl">  <property name="accountDao" ref="accountDao"/>  <property name="itemDao" ref="itemDao"/>  *<!-- additional collaborators and configuration for this bean go here -->*  </bean>  *<!-- more bean definitions for services go here -->*  </beans> |

以下示例显示了数据访问对象daos.xml文件：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  https://www.springframework.org/schema/beans/spring-beans.xsd">  <bean id="accountDao"  class="org.springframework.samples.jpetstore.dao.jpa.JpaAccountDao">  *<!-- additional collaborators and configuration for this bean go here -->*  </bean>  <bean id="itemDao" class="org.springframework.samples.jpetstore.dao.jpa.JpaItemDao">  *<!-- additional collaborators and configuration for this bean go here -->*  </bean>  *<!-- more bean definitions for data access objects go here -->*  </beans> |

在前面的示例中，服务层由PetStoreServiceImpl类和JpaAccountDao和JpaItemDao类型的两个数据访问对象（基于JPA对象关系映射标准）组成。该property name元素是指JavaBean属性的名称，以及ref元素指的是另一个bean定义的名称。元素id和ref元素之间的这种联系表达了协作对象之间的依赖关系。有关配置对象的依赖关系的详细信息，请参阅 [依赖关系](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-dependencies)。

##### 编写基于XML的配置元数据

让bean定义跨越多个XML文件会很有用。通常，每个单独的XML配置文件都代表架构中的逻辑层或模块。

您可以使用应用程序上下文构造函数从所有这些XML片段加载bean定义。此构造函数采用多个Resource位置，如上[一节中](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-instantiation)所示 。或者，使用一个或多个<import/>元素来从另一个或多个文件加载bean定义。以下示例显示了如何执行此操作：

|  |
| --- |
| <beans>  <import resource="services.xml"/>  <import resource="resources/messageSource.xml"/>  <import resource="/resources/themeSource.xml"/>  <bean id="bean1" class="..."/>  <bean id="bean2" class="..."/>  </beans> |

在前面的例子中，外部bean定义是从三个文件加载： services.xml，messageSource.xml，和themeSource.xml。所有位置路径都与执行导入的定义文件相关，因此services.xml必须与执行导入的文件位于相同的目录或类路径位置，messageSource.xml和themeSource.xml必须位于导入文件位置下方的resources位置下。如您所见，前面的斜杠被忽略了。但是，鉴于这些路径是相对的，所以最好不要使用斜杠。正在导入的文件的内容（包括顶级元素<beans/>）必须是根据Spring Schema有效的XML bean定义。

可以（但不建议）使用相对“../”路径引用父目录中的文件。这样做会对当前应用程序之外的文件创建依赖关系。特别是，不建议对classpath:URL（例如，classpath:../services.xml）使用此引用，其中运行时解析过程选择“最近的”类路径根，然后查看其父目录。类路径配置更改可能导致选择不同的，不正确的目录。

您始终可以使用完全限定的资源位置而不是相对路径：例如，file:C:/config/services.xml或classpath:/config/services.xml。但是，请注意您将应用程序的配置与特定的绝对位置耦合。通常最好为这些绝对位置保持间接 - 例如，通过在运行时针对JVM系统属性解析的“$ {...}”占位符。

命名空间本身提供了导入指令功能。Spring提供的一系列XML命名空间中提供了除普通bean定义之外的其他配置功能 - 例如，context和util名称空间。

##### Groovy Bean定义DSL

作为外化配置元数据的另一个示例，bean定义也可以在Spring的Groovy Bean定义DSL中表示，如Grails框架中所知。通常，此类配置位于“.groovy”文件中，其结构如下例所示：

|  |
| --- |
| beans {  dataSource(BasicDataSource) {  driverClassName = "org.hsqldb.jdbcDriver"  url = "jdbc:hsqldb:mem:grailsDB"  username = "sa"  password = ""  settings = [mynew:"setting"]  }  sessionFactory(SessionFactory) {  dataSource = dataSource  }  myService(MyService) {  nestedBean = { AnotherBean bean ->  dataSource = dataSource  }  }  } |

此配置样式在很大程度上等同于XML bean定义，甚至支持Spring的XML配置命名空间。它还允许通过importBeans指令导入XML bean定义文件。

#### 1.2.3 使用容器

ApplicationContext是高级工厂的接口，能够维护不同bean及其依赖项的注册表。通过使用T getBean(String name, Class<T> requiredType)方法，您可以得到Bean的实例。

ApplicationContext允许您读取bean定义并访问它们，如下例所示：

|  |
| --- |
| *// create and configure beans*  ApplicationContext context = **new** ClassPathXmlApplicationContext("services.xml", "daos.xml");  *// retrieve configured instance*  PetStoreService service = context.getBean("petStore", PetStoreService.class);  *// use configured instance*  List<String> userList = service.getUsernameList(); |

使用Groovy配置，引导看起来非常相似。它有一个不同的上下文实现类，它支持Groovy

（但也理解XML bean定义）。以下示例显示了Groovy配置：

|  |
| --- |
| ApplicationContext context = **new** GenericGroovyApplicationContext("services.groovy", "daos.groovy"); |

最灵活的变体是GenericApplicationContext与reader委托相结合 - 例如，使用XmlBeanDefinitionReader来处理XML文件，如以下示例所示：

|  |
| --- |
| GenericApplicationContext context = **new** GenericApplicationContext();  **new** XmlBeanDefinitionReader(context).loadBeanDefinitions("services.xml", "daos.xml");  context.refresh(); |

您还可以对Groovy文件使用GroovyBeanDefinitionReader，如以下示例所示：

|  |
| --- |
| GenericApplicationContext context = **new** GenericApplicationContext();  **new** GroovyBeanDefinitionReader(context).loadBeanDefinitions("services.groovy", "daos.groovy");  context.refresh(); |

您可以在相同的ApplicationContext上混合和匹配这样的reader委托，从不同的配置源读取bean定义。

然后，您可以使用getBean来检索Bean的实例。ApplicationContext 接口还有一些其他方法可用于获得bean，但理想情况下，应用程序代码不应使用它们。实际上，您的应用程序代码完全不应该调用该 getBean()方法，因此根本不依赖于Spring API。例如，Spring与Web框架的集成为各种Web框架组件（如控制器和JSF管理的bean）提供依赖注入，允许您通过元数据（例如自动装配注解）声明对特定bean的依赖性。

### 1.3 Bean概述

Spring IoC容器管理一个或多个bean。这些bean是使用您提供给容器的配置元数据创建的（例如，以XML <bean/>定义的形式 ）。

在容器本身内，这些bean定义表示为BeanDefinition 对象，其中包含（以及其他信息）以下元数据：

* 包限定的类名：通常是正在定义的bean的实际实现类。
* Bean行为配置元素，说明bean在容器中的行为方式（范围，生命周期回调等）。
* 引用bean执行其工作所需的其他bean。这些引用也称为协作者或依赖项。
* 要在新创建的对象中设置的其他配置设置 - 例如，池的大小限制或在管理连接池的Bean中使用的连接数。

此元数据转换为构成每个bean定义的一组属性。下表描述了这些属性：

*表1. bean定义*

| **属性** | **解释** |
| --- | --- |
| 类 | [实例化Bean](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-class) |
| 名称 | [命名Bean](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-beanname) |
| 范围 | [Bean范围](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-scopes) |
| 构造函数参数 | [依赖注入](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-collaborators) |
| 属性 | [依赖注入](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-collaborators) |
| 自动装配模式 | [自动化协作者](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-autowire) |
| 延迟初始化模式 | [懒加载的Bean](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-lazy-init) |
| 初始化方法 | [初始化回调](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-lifecycle-initializingbean) |
| 破坏方法 | [毁灭回调](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-lifecycle-disposablebean) |

除了包含有关如何创建特定bean的信息的bean定义之外，这些ApplicationContext实现还允许注册在容器外部（由用户）创建的现有对象。这是通过getBeanFactory()方法访问ApplicationContext的BeanFactory来完成的，该方法返回BeanFactory的 DefaultListableBeanFactory实现。DefaultListableBeanFactory 通过registerSingleton(..)和 registerBeanDefinition(..)方法支持此注册。但是，典型的应用程序仅使用通过常规bean定义元数据定义的bean。

Bean元数据和手动提供的单例实例需要尽早注册，以便容器在自动装配和其他步骤中正确地推理它们。虽然在某种程度上支持覆盖现有的元数据和现有的单例实例，但是在运行时注册新bean(与对工厂的实时访问同时进行)并不是官方支持的，并且可能会导致并发访问异常、bean容器中的状态不一致，或者两者兼有。

#### 1.3.1 命名Bean

每个bean都有一个或多个标识符。这些标识符在管理bean的容器中必须是唯一的。bean通常只有一个标识符。但是，如果它需要多个，则额外的可以被视为别名。

在基于XML的配置元数据中，您可以使用id属性，name属性或两者来指定bean标识符。该id属性允许您指定一个id。通常，这些名称是字母数字（'myBean'，'someService'等），但它们也可以包含特殊字符。如果要为bean引入其他别名，还可以在name 属性中指定它们，用逗号”,”，分号”;”,或空格” ”分隔。作为历史记录，在Spring 3.1之前的版本中，该id属性被定义为一种xsd:ID类型，它约束了可能的字符。从3.1开始，它被定义为一种xsd:string类型。请注意，容器仍然强制实施bean id的唯一性，但不再是XML解析器执行。

您不需要为bean提供 name或id。如果您不显式提供提供 name或id，则容器会为该bean生成唯一的名称。但是，如果要按名称引用该bean,通过使用ref元素或[Service Locator](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-servicelocator)样式查找，您必须提供一个名称。而不提供名称的动机是因为与使用[内部bean](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-inner-beans)和[自动装配协作者有关](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html" \l "beans-factory-autowire)。

|  |
| --- |
| Bean命名约定  惯例是在命名bean时使用标准Java约定作为实例字段名称。也就是说，bean名称以小写字母开头，并从那里开始驼峰。这样的名字的例子包括accountManager， accountService，userDao，loginController，等等。  命名bean应该始终使您的配置更易于阅读和理解。此外，如果您使用Spring AOP，那么当您将建议应用于与名称相关的一组bean时，它会有很大帮助。 |

通过类路径中的组件扫描，Spring按照前面描述的规则为未命名的组件生成bean名称：实质上，采用简单的类名并将其初始字符转换为小写。但是，在（不常见的）特殊情况下，当有多个字符且第一个和第二个字符都是大写字母时，则保留原来的大小写。这些规则与java.beans.Introspector.decapitalize（Spring在此处使用）定义的规则相同。

##### 在Bean定义之外定义Bean的别名

在bean定义本身中，您可以为bean提供多个名称，方法是使用id属性指定的最多一个名称和属性name中的任意数量的其他名称。这些名称可以是同一个bean的等效别名，对某些情况很有用，例如让应用程序中的每个组件通过使用特定于该组件本身的bean名称来引用公共依赖项。

但是，指定实际定义bean的所有别名并不总是足够的。有时需要为其他地方定义的bean引入别名。在大型系统中通常就是这种情况，其中配置在每个子系统之间分配，每个子系统具有其自己的一组对象定义。在基于XML的配置元数据中，您可以使用该<alias/>元素来完成此任务。以下示例显示了如何执行此操作：

|  |
| --- |
| <alias name="fromName" alias="toName"/> |

在这种情况下，fromName在使用此别名定义之后，命名的bean（在同一容器中）也可以称为toName。

例如，子系统A的配置元数据可以通过名称subsystemA-dataSource引用DataSource 。子系统B的配置元数据可以通过subsystemB-dataSource名称引用DataSource 。在编写使用这两个子系统的主应用程序时，主应用程序通过名称myApp-dataSource引用DataSource 。要使所有三个名称引用同一对象，可以将以下别名定义添加到配置元数据中：

|  |
| --- |
| <alias name="myApp-dataSource" alias="subsystemA-dataSource"/>  <alias name="myApp-dataSource" alias="subsystemB-dataSource"/> |

现在，每个组件和主应用程序都可以通过一个唯一的名称引用dataSource，并保证不与任何其他定义冲突（有效地创建命名空间），但它们引用相同的bean。

|  |
| --- |
| Java的配置  如果使用Javaconfiguration，则@Bean可以使用注释来提供别名。有关详细信息，请参阅[使用@Bean注释](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html" \l "beans-java-bean-annotation)。 |

#### 1.3.2 实例化Bean

bean定义本质上是用于创建一个或多个对象的配方。容器在被询问时查看命名bean的配方，并使用由该bean定义封装的配置元数据来创建（或获取）实际对象。

如果使用基于XML的配置元数据，则指定要在<bean/>元素的class属性中实例化的对象的类型（或类）。此 class属性（在内部，是BeanDefinition 实例上的一个类属性）通常是必需的。（有关例外，请参阅 [使用实例工厂方法](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-class-instance-factory-method)和[Bean定义继承](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-child-bean-definitions)[进行实例化](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-class-instance-factory-method)。）您可以通过以下Class两种方式之一使用该属性：

* 通常，在容器本身通过反射调用其构造函数直接创建bean的情况下直接创建bean的情况下,指定要构造的bean类，这有点类似于使用new运算符的Java代码。
* 要指定包含创建对象时而调用的静态工厂方法的实际类，在不太常见的情况下，容器在类上调用静态工厂方法来创建bean。从调用静态工厂方法返回的对象类型可以完全是同一个类,也可以完全是另一个类。

|  |
| --- |
| *内部类名*  如果要为static嵌套类配置bean定义，则必须使用嵌套类的二进制名称。  例如，如果您SomeThing在com.example包中调用了一个类，并且此类 SomeThing具有一个被调用的静态嵌套类OtherThing，则class bean定义中的属性值将为com.example.SomeThing$OtherThing。  请注意，名称中使用$字符将嵌套类名与外部类名分开。 |

##### 使用构造函数实例化

当您通过构造方法创建bean时，所有普通类都可以使用并与Spring兼容。也就是说，正在开发的类不需要实现任何特定接口或以特定方式编码。简单地指定bean类就足够了。但是，根据您为该特定bean使用的IoC类型，您可能需要一个默认（空）构造函数。

Spring IoC容器几乎可以管理您希望它管理的任何类。它不仅限于管理真正的JavaBeans。大多数Spring用户更喜欢实际的JavaBeans，只有一个默认（无参数）构造函数，并且在容器中的属性之后构建了适当的setter和getter。您还可以在容器中拥有更多非bean样式类。例如，如果您需要使用完全不符合JavaBean规范的旧连接池，那么Spring也可以对其进行管理。

使用基于XML的配置元数据，您可以按如下方式指定bean类：

|  |
| --- |
| <bean id="exampleBean" class="examples.ExampleBean"/>  <bean name="anotherExample" class="examples.ExampleBeanTwo"/> |

有关为构造函数提供参数的机制（如果需要）以及在构造对象后设置对象实例属性的详细信息，请参阅 [注入依赖项](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-collaborators)。

##### 使用静态方法工厂实例化

定义使用静态工厂方法创建的bean时，请使用该class 属性指定包含静态方法工厂的类，并使用factory-method名称的属性指定工厂方法本身的名称。您应该能够调用此方法（使用可选参数，如稍后所述）并返回一个活动对象，随后将其视为通过构造函数创建的对象。这种bean定义的一个用途是在遗留代码中调用静态工厂。

以下bean定义指定通过调用工厂方法来创建bean。该定义未指定返回对象的类型（类），仅指定包含工厂方法的类。在此示例中，该createInstance() 方法必须是静态方法。以下示例显示如何指定工厂方法：

|  |
| --- |
| <bean id="clientService"  class="examples.ClientService"  factory-method="createInstance"/> |

以下示例显示了一个可以使用前面的bean定义的类：

|  |
| --- |
| **public** **class** **ClientService** {  **private** **static** ClientService clientService = **new** ClientService();  **private** ClientService() {}  **public** **static** ClientService createInstance() {  **return** clientService;  }  } |

有关在从工厂返回对象后为工厂方法提供（可选）参数和设置对象实例属性的机制的详细信息，请参阅[依赖关系和详细配置](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-properties-detailed)。

##### 使用实例工厂方法实例化

与通过[静态工厂方法](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-class-static-factory-method)实例化类似，使用实例[工厂方法进行](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-class-static-factory-method)实例化会从容器调用现有bean的非静态方法来创建新bean。要使用此机制，请将该class属性保留为空，并在factory-bean属性中指定当前（或父或祖先）容器中bean的名称，该容器包含要调用以创建对象的实例方法。使用factory-method属性设置工厂方法本身的名称。以下示例显示如何配置此类bean：

|  |
| --- |
| *<!-- the factory bean, which contains a method called createInstance() -->*  <bean id="serviceLocator" class="examples.DefaultServiceLocator">  *<!-- inject any dependencies required by this locator bean -->*  </bean>  *<!-- the bean to be created via the factory bean -->*  <bean id="clientService"  factory-bean="serviceLocator"  factory-method="createClientServiceInstance"/> |

以下示例显示了相应的Java类：

|  |
| --- |
| **public** **class** **DefaultServiceLocator** {  **private** **static** ClientService clientService = **new** ClientServiceImpl();  **public** ClientService createClientServiceInstance() {  **return** clientService;  }  } |

一个工厂类也可以包含多个工厂方法，如以下示例所示：

|  |
| --- |
| <bean id="serviceLocator" class="examples.DefaultServiceLocator">  *<!-- inject any dependencies required by this locator bean -->*  </bean>  <bean id="clientService"  factory-bean="serviceLocator"  factory-method="createClientServiceInstance"/>  <bean id="accountService"  factory-bean="serviceLocator"  factory-method="createAccountServiceInstance"/> |

以下示例显示了相应的Java类：

|  |
| --- |
| **public** **class** **DefaultServiceLocator** {  **private** **static** ClientService clientService = **new** ClientServiceImpl();  **private** **static** AccountService accountService = **new** AccountServiceImpl();  **public** ClientService createClientServiceInstance() {  **return** clientService;  }  **public** AccountService createAccountServiceInstance() {  **return** accountService;  }  } |

这种方法表明工厂bean本身可以通过依赖注入（DI）进行管理和配置。请参阅[详细信息中的依赖关系和配置](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-properties-detailed)。

在Spring文档中，“factory bean”是指在Spring容器中配置并通过[实例](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-class-instance-factory-method)或 [静态](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html#beans-factory-class-static-factory-method)工厂方法创建对象的bean 。相比之下， FactoryBean（注意大写）指的是特定于Spring的 [FactoryBean](https://docs.spring.io/spring/docs/5.1.x/spring-framework-reference/core.html" \l "beans-factory-extension-factorybean)。

### 1.4 依赖