# Spring体系结构

1. 底层是核心容器

\*Beans

\*Core

\*Context

\*SpringEI表达式

1. 中间层技术

\*AOP

\*Aspects

1. 应用层技术

\*数据访问与数据集成

\*Web集成

4、基于Test测试

## IoC

IoC(Inversion Of Control)控制反转，Spring反向控制应用程序所需要使用的外部资源。

Spring控制的资源全部放置在Spring容器中，该容器称为IoC容器；

传统模式中，对象的主控权在类手中，IoC模式下，对象的主控权在spring手中。

## 入门案例

模拟三层架构中表现层调用业务层功能

-表现层：UserApp模拟UserServlet(使用main方法模拟)

-业务层：UserService

IoC入门案例制作步骤

1. 导入spring坐标（5.1.9.release）
2. 编写业务层与表现层（模拟）接口与实现类
3. 建立spring配置文件
4. 配置所需资源（Service）为spring控制的资源
5. 表现层（App）通过spring获取资源（Service实例）

## IoC配置（XML格式）

### bean

名称：bean

类型：标签

归属：beans标签

作用：定义spring中的资源，受此标签定义的资源将受到spring控制

格式：

<beans>

<bean />

<beans>

基本上属性

<bean id=”beanId” name=”beanName1,beanName2” class=”ClassName”></bean>

\*id：bean的名称，通过id值获取bean

\*class：bean的类型

\*name：bean的名称，可以通过name值获取bean，用于多人配合时给bean起别名；

bean可以定义多个名称，使用name属性完成，中间使用，分割。

\*scope：用于控制bean创建后的对象是否是单例的

名称：scope

类型：属性

归属：bean标签

作用：定义bean的作用范围

格式：

<bean scope=”singleton”></bean>

取值：

1. singleton：设定创建出的对象保存在spring容器中，是一个单例的对象；对象的创建时机为加载容器时
2. prototype：设定创建出的对象保存在spring容器中，是一个非单例的对象；对象的创建时机为获取对象即getbean()时才创建的对象；
3. request、session、application、websocket：设定创建出的对象放置在web容器对应的位置；

\*bean生命周期

名称：init-method、destroy-method

类型：属性

归属：bean标签

作用：定义bean对象在初始化或销毁时完成的工作

格式：

<bean init-method=”init” destroy-method=”destroy”></bean>

取值：bean对应的类中对应的具体方法名

注意事项：

- 当scope=“singleton”时，spring容器中有且仅有一个对象，init方法在创建容器时仅执行一次；

- 当scope=“prototype”时，spring容器要创建同一类型的多个对象，init方法在每个对象创建时均执行一次；

- 当scope=“singleton”时，关闭容器会导致bean实例的销毁，调用destroy方法一次；

- 当scope=“prototype”时，对象的销毁由垃圾回收机制gc()控制，destroy方法将不会被执行；

\*bean对象创建方式

名称：factory-bean，factory-method

类型：属性

归属：bean标签

作用：定义bean对象创建方式，使用静态工厂的形式创建bean，兼容早期一六系统的升级工作

1、格式：

<bean class=”FactoryClassName” factory-method=”factoryMethodName”></bean>

取值：工厂bean中用于获取对象的静态方法名

注意事项：

-class属性必须配置成静态工厂的类名；

2、格式：

<bean factory-bean=”factoryBeanId” factory-method=”factoryMethodName”></bean>

取值：工厂bean中用于获取对象的实例方法名

注意事项：

-使用实例工厂创建bean首先需要将实例工厂配置bean，交由spring管理；

-factory-bean是实例工厂的beanId

<!—静态工厂创建bean-->

<bean class="FactoryClassName" factory-method="factoryMethodName"></bean>

<!—实例工厂对应的bean-->

<bean id=”factoryBean” class=”com.yuxun.service.UserServiceFactory2”/>

<!—实例工厂创建bean，依赖工厂对象对应的bean-->

<bean factory-bean="factoryBean" factory-method="factoryMethodName"/>

### DI---IoC与DI是同一件7事站在不同角度看待问题

DI（Dependency Injection）依赖注入，应用程序运行依赖的资源由spring为其提供，资源进入应用程序的方式称为注入；

#### set注入(主流)

名称：property

类型：标签

归属：bean标签

作用：使用set方法的形式为bean提供资源

格式：

<bean>

<property />

<bean/>

基本属性

<property name=”propertyName” value=”propertyValue” ref=”beanId”/>

-name：对应bean中的属性名，要求该属性必须提供可访问的set方法(严格规范为此名称是set方法对应名称)；

-value：设定非引用类型属性对应的值，不能与ref同时使用；

-ref：设定引用类型属性对应bean的id，不能与value同时使用；

注意：一个bean可以有多个property标签

#### 构造器注入

名称：constructor-arg

类型：标签

归属：bean标签

作用：使用构造方法的形式为bean提供资源，兼容早期遗留系统的升级工作；

格式：

<bean>

<constructor-arg />

</bean>

基本属性：

<constructor-arg name=”argsName” value=”argsValue” />

-name：对应bean中的构造方法所携带的参数名

-value：设定非引用类型构造方法参数对应的值，不能与ref同时使用；

注意：一个bean可以有多个constructor-arg标签

其他属性：

<constructor-arg index=”arg-index” type=”arg-type” ref=”beanId” />

-ref：设定引用类型构造方法参数对应bean的id，不能与value同时使用；

-type：设定构造方法参数的类型，用于按类型匹配参数或进行类型校验；

-index：设定构造方法参数的位置，用于按位置匹配参数，参数index值从0开始计数；

#### 集合类型数据注入

名称：array、list、set、map、props；

类型：标签；

归属：property标签或constructor-arg标签；

作用：注入集合数据类型属性；

格式：

<property>

<list></list>

</property>

例如：集合类型数据注入—list

<!—List集合类型注入数据-->

<property name=”myList”>

<list>

<value>yuxun</value>

<value>6666</value>

<ref bean=”userService”/>

<bean class=”com.yuxun.service.ApplyService”/>

</list>

</property>

<!—array集合类型注入数据--><!—array标签使用与List是互通的，可以互用-->

<property name=”myArray”>

<array>

<value>yuxun</value>

<value>6666</value>

<ref bean=”userService”/>

<bean class=”com.yuxun.service.ApplyService”/>

</array>

</property>

<!—properties类型注入数据-->

<property name=”myProps”>

<props>

<prop key=”username”>root</prop>

<prop key=”password”>root</prop>

</props>

</property>

<!—set集合类型注入数据-->

<property name=”myset”>

<set>

<value>yuxun</value>

<value>6666</value>

<ref bean=”userService”/>

<bean class=”com.yuxun.service.ApplyService”/>

</set>

</property>

<!—map集合类型注入数据-->

<property name=”myMap”>

<map>

<entry key=”name” value-ref=”yuxun”/>

<entry key=”fame” value-ref=”666”>

<entry key=”userService”>

<ref bean=”userService”></ref>

</entry>

<entry key=”applyService”>

<bean class=”applyService”/>

</entry>

</map>

</property>

#### 使用P命名空间简化配置

名称：p：propertyName， p：propertyName-ref；

类型：属性；

归属：bean标签；

作用：为bean注入属性值；

格式：

<bean p:propertyName=”propertyValue” p:propertyName-ref=”beanId”/>

注意：使用p命令空间需要先开启spring对p命令空间的支持，在beans标签中添加对应空间支持；

#### SpEL

Spring提供了对EL表达式的支持，统一属性注入格式

- 类型：属性值

- 归属：value属性值

- 作用：为bean注入属性值

- 格式：

<property value="EL"></bean>

- 注意：所有属性值不区分是否引用类型，统一使用value赋值

- 所有格式统一使用 value=“\*\*\*\*\*\*\*\*”

- 常量 #{10} #{3.14} #{2e5} #{‘itcast’}

- 引用bean #{beanId}

- 引用bean属性 #{beanId.propertyName}

- 引用bean方法 beanId.methodName().method2()

- 引用静态方法 T(java.lang.Math).PI

- 运算符支持 #{3 lt 4 == 4 ge 3}

- 正则表达式支持 #{user.name matches‘[a-z]{6,}’}

- 集合支持 #{likes[3]}

- 案例：

<bean id="userService" class="com.itheima.service.impl.UserServiceImpl">

<property name="userDao" value="#{userDao}"/>

<property name="bookDao" value="#{bookDao}"/>

<property name="num" value="#{666666666}"/>

<property name="version" value="#{'itcast'}"/>

</bean>

### properties文件

Spring提供了读取外部properties文件的机制，使用读取到的数据为bean的属性赋值；

操作步骤：

1.准备外部properties文件

2.开启context命名空间支持

xmlns:context="http://www.springframework.org/schema/context"

​ 3.加载指定的properties文件

<context:property-placeholder

location="classpath:filename(配置文件名).properties">

4.使用加载的数据

<property name="propertyName" value="${propertiesName}"/>

注意：

-如果需要加载所有的properties文件，可以使用\*.properties表示加载所有的properties文件

-读取数据使用${propertiesName}（这里面的名称使用properties文件内的数据名称）格式进行，其中propertiesName指properties文件中的属性名

开启命名空间除了操作步骤中第二步外，还需要在xsi:schemaLocation中增加配置

<https://www.springframework.org/schema/context>(空间名称-一共有三处，标红)

<https://www.springframework.org/schema/context/spring-context.xsd>

可以使用原本用来开beans空间的配置语句，但是这三处要记得都改了；

配置文件中目前有两种读数据的符号：

${ }：这是读properties属性的；

#{ }：这是spring的EL表达式；

### 团队开发

名称：import；

类型：标签；

归属：beans标签；

作用：在当前配置文件中导入其他配置文件中的项；

格式：

<beans>

<import />

<beans>

基本属性：

<import resource=”config.xml”>

resource：加载的配置文件名

Spring容器加载多个配置文件

new ClassPathXmlApplicationContext(“config1.xml”,”config2.xml”)；

Spring容器中的bean定义冲突(如果在同一个配置文件中出现同名bean，执行会出错)问题：

- 同id的bean，后定义的覆盖先定义的；

- 导入配置文件可以理解为将导入的配置文件复制粘贴到对应位置；

- 导入配置文件的顺序与位置不同可能会导致最终程序运行结果不同；

### ApplicationContext

1.ApplicationContext是一个接口，提供了访问spring容器的API

2.ClassPathXmlApplicationContext是一个类，实现了上述功能

3.ApplicationContext的顶层接口是BeanFactory

4.BeanFactory定义了bean相关的最基本操作

5.ApplicationContext在BeanFactory基础上追加了若干新功能

\*\*对比BeanFactory\*\*

1.BeanFactory创建的bean采用延迟加载形式，使用才创建

2.ApplicationContext创建的bean默认采用立即加载形式

\*\*FileSystemXmlApplicationContext\*\*

可以加载文件系统中任意位置的配置文件，而ClassPathXmlApplicationContext只能加载类路径下的配置文件

\*\*BeanFactory\*\*

Resource res = new ClassPathResource("applicationContext.xml");

BeanFactory bf = new XmlBeanFactory(res);

UserService userService = (UserService)bf.getBean("userService");

### 第三方资源配置

#### 阿里数据方案Druid

Xml配置文件内容：

<bean id="dataSource"

class="com.alibaba.druid.pool.DruidDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"></property>

<property name="url" value="jdbc:mysql://localhost:3306/spring\_ioc"></property>

<property name="username" value="root"></property>

<property name="password" value="root"></property>

</bean>

#### 阿里数据方案Druid