

IOC配置详解

1.2. 容器概述

ApplicationContext是Spring IoC容器实现的代表，它负责实例化，配置和组装Bean。容器通过读取配置元数据获取有关实例化、配置和组装哪些对象的说明。配置元数据可以使用XML、Java注解或Java代码来呈现。它允许你处理应用程序的对象与其他对象之间的互相依赖关系。

1.2.1. 配置元数据

- 使用xml的配置

简单、直观 适合入门

- 基于注解的配置: @Component(@service @controller @repository) @Autowired

Spring 2.5 支持基于注解的元数据配置. SSM框架开发中的使用

- 基于Java的配置: @Configuration @Bean @Import

从 Spring 3.0开始, 由Spring JavaConfig项目提供的功能已经成为Spring核心框架的一部分。因此, 你可以使用Java配置来代替XML配置定义外部bean。

从spring4.0开始支持springboot1.0 之后 springboot完全采用javaConfig的方式进行开发。

1.2.2. 容器的实例化

对象在Spring容器创建完成的时候就已经创建完成, 不是需要用的时候才创建

1.2.3. 容器的使用

ApplicationContext是能够创建bean定义以及处理相互依赖关系的高级工厂接口, 使用的方法T.getBean(String name, Class<T> requiredType)获取容器实例。

```
1 // 创建spring上下文 加载所有的bean
2 ApplicationContext context = new ClassPathXmlApplicationContext("services.xml", "daos.xml");
3
4 // 获取bean
5 PetStoreService service = context.getBean("petStore", PetStoreService.class);
6
7 // 使用bean的对象
8 List<String> userList = service.getUsernameList();
```

1.3. bean的概述

1.3.1. 命名bean

```
1 <bean class="com.tuling.entity.User" id="user" name="user2 user3,user4;user5"></bean>
```

- 为外部定义的bean起别名

```
1 <alias name="user" alias="user6"></alias>
```

1.3.2. 实例化Bean

- 使用构造器实例化 默认
无法干预实例化过程
- 使用静态工厂方法实例化

```
1 <bean class="com.tuling.service.impl.UserServiceImpl" id="userService2"  
2     factory-method="createUserServiceInstance" >  
3 </bean>
```

```
1 public static UserServiceImpl createUserServiceInstance(){  
2     return new UserServiceImpl();  
3 }
```

- 使用实例工厂方法实例化

```
1 <bean class="com.tuling.service.impl.UserServiceImpl" id="userService"  
2     factory-bean="serviceFactory"  
3     factory-method="createUserService" >  
4 </bean>
```

```
1 public class createUserService{  
2  
3     public UserServiceImpl createUserFactory(){  
4         return new UserServiceImpl();  
5     }  
6 }
```

● 1.4. 依赖

○ 1.4.1. 依赖注入

- 基于setter方法的依赖注入

```
1 <!--基于setter方法的依赖注入  
2     1. 属性必须声明了set方法  
3     2. name是根据set方法的名字来的 比如方法名字是: setIdxx -> name="idxx"
```

```

4 -->
5 <bean class="cn.tulingxueyuan.beans.User" id="user2">
6     <property name="idtuling" value="1"></property>
7     <property name="username" value="zhangsan"></property>
8     <property name="realname" value="张三"></property>
9 </bean>

```

■ 基于构造函数的依赖注入

```

1 <!-- 基于构造函数的依赖注入
2     1. 将会调用自定义构造函数来实例化对象，就不会调用默认的无参构造函数
3     2. name是根据构造函数的参数名来的， 比如: User(String idxx) -> name="idxx"
4     3. name属性可以省略 但是要注意参数的位置
5     4. 如果非要把位置错开 可以使用 name 或者 index 或者 type
6     5. index 是下标 从0开始
7     6. type 在位置错开情况下只能在类型不一样的时候指定才有明显效果
8 -->
9 <bean class="cn.tulingxueyuan.beans.User" id="user3">
10     <constructor-arg name="username" value="lisi"></constructor-arg>
11     <constructor-arg name="id_xushu" value="1"></constructor-arg>
12     <constructor-arg name="realname" value="李四"></constructor-arg>
13 </bean>

```

○ 1.4.2. 依赖和配置的细节

- 直接值（基本类型，String等）
- 对其他bean的引用(装配)
- 内部bean
- 集合
- null和空的字符串值
- 使用p命名空间简化基于setter属性注入XML配置
- 使用c命名空间简化基于构造函数的XML

```

1 <!-- 复杂数据类型 -->
2 <bean class="cn.tulingxueyuan.beans.Person" id="person" p:wife-ref="wife2">
3     <property name="id" value="1"></property>
4     <property name="realName" value=""></property>
5     <!-- 设置null值 -->
6     <property name="name">
7         <null></null>

```

```

8      </property>
9      <!--当依赖其他bean: 内部bean    inner bean
10     <property name="wife">
11         <bean class="cn.tulingxueyuan.beans.Wife" >
12             <property name="age" value="18"></property>
13             <property name="name" value="迪丽热巴"></property>
14         </bean>
15     </property>-->
16     <!--当依赖其他bean: 引用外部bean
17     <property name="wife" ref="wife"></property>-->
18     <property name="birthday" value="2020/05/20"></property>
19     <property name="hobbies">
20         <list>
21             <value>唱歌</value>
22             <value>跳舞</value>
23             <!--如果List的泛型是比如: List<Wife> <bean>-->
24         </list>
25     </property>
26     <property name="course" >
27         <map>
28             <entry key="1" value="JAVA"> </entry>
29             <entry key="2" value="HTML"> </entry>
30         </map>
31     </property>
32
33 </bean>
34
35 <!--可以使用p命名空间来简化基于setter属性注入    它不支持集合-->
36 <bean class="cn.tulingxueyuan.beans.Wife" id="wife" p:age="18" p:name="迪丽热巴" >
37 </bean>
38
39 <!--可以使用c命名空间来简化基于构造函数属性注入    它不支持集合-->
40 <bean class="cn.tulingxueyuan.beans.Wife" id="wife2" c:age="20" c:name="xxx">
41     <!-- <constructor-arg name="age" value="18"></constructor-arg>-->
42 </bean>

```

○ 1.4.3. 使用 depends-on属性

```

1 <!--使用depends-on可以设置先加载的Bean    也就是控制bean的加载顺序-->
2 <bean class="cn.tulingxueyuan.beans.Person" id="person" depends-on="wife"></bean>

```

```
3 <bean class="cn.tulingxueyuan.beans.Wife" id="wife"></bean>
```

○ 1.4.4. 懒加载bean

```
1 <!--使用lazy-init设置懒加载
2 默认为false：在spring容器创建的时候加载（实例化）
3     true：在使用的时候(getBean)才会去加载（实例化）-->
4 <bean class="cn.tulingxueyuan.beans.Person" id="person2" lazy-init="true">
5     <property name="id" value="1"></property>
6     <property name="realName" value="吴彦祖"></property>
7     <property name="name" value="徐庶"></property>
8 </bean>
9
```

○ 1.4.5. 自动注入

当一个对象中需要引用另外一个对象的时候，在之前的配置中我们都是通过property标签来进行手动配置的，其实在spring中还提供了一个非常强大的功能就是自动装配，可以按照我们指定的规则进行配置，配置的方式有以下几种：

default/no：不自动装配

byName：按照名字进行装配，以属性名作为id去容器中查找组件，进行赋值，如果找不到则装配null

byType：按照类型进行装配，以属性的类型作为查找依据去容器中找到这个组件，如果有多个类型相同的bean对象，那么会报异常，如果找不到则装配null

constructor：按照构造器进行装配，先按照有参构造器参数的类型进行装配，没有就直接装配null；如果按照类型找到了多个，那么就使用参数名作为id继续匹配，找到就装配，找不到就装配null

- 通过将autowire-candidate 属性设置为false，避免对bean定义进行自动装配，[如下一节所述](#)。
- 通过将其<bean/> 元素的primary属性设置为 true，将单个bean定义指定为主要候选项。

```
1 <!-- *****基于xml自动注入 begin*****
2
3 <!-- 自动注入：
4 1. bytype 根据类型自动注入(spring会根据bean里面的所有对象属性的类型，只要它匹配到bean里面某一个
5 2. byname 会根据属性setxxx的名字来自动匹配 (spring会根据bean里面的所有对象属性的set的名字，只要
6 3. constructor 优先根据名字来找， 如果名字没有匹配到根据类型来匹配， 如果类型匹配到多个则不会自
7 注意： bytype 如果匹配到两个同样的类型会出现错误，所以一定要保证ioc容器里面只有一个对应类型的bea
8     byname 最能匹配到唯一的那个bean
```

```
9     constructor 保证构造函数不能包含多余的其他参数
```

```

10 default:不会进行自动注入
11
12 <bean class="cn.tulingxueyuan.beans.Person" id="person6" autowire="constructor" >
13     <property name="id" value="1"></property>
14     <property name="realName" value="吴彦祖"></property>
15     <property name="name" value="徐庶"></property>
16 </bean>
17
18 <bean class="cn.tulingxueyuan.beans.Wife" id="wife" p:age="18" p:name="迪丽热巴" >
19 </bean>
20 <bean class="cn.tulingxueyuan.beans.Wife" id="QBL" p:age="60" p:name="乔碧螺" >
21 </bean>
22 *****基于xml自动注入 end*****

```

• 1.5. Bean 的作用域

- 1.5.1. Singleton(单例)的作用域
- 1.5.2. Prototype(原型)的作用域

```

1 <!--作用域scope
2 singleton 默认:单例 只会在Ioc容器种创建一次
3 prototype 多例（原型bean）每次获取都会new一次新的bean-->
4
5 <bean class="cn.tulingxueyuan.beans.Person" id="person3" scope="prototype">
6     <property name="id" value="1"></property>
7     <property name="realName" value="吴彦祖"></property>
8     <property name="name" value="徐庶"></property>
9 </bean>

```

• 1.6. 自定义bean的特性

- 1.6.1. 生命周期回调
 - 初始化方法回调
 - 销毁方法回调
 - 在非Web应用中优雅地关闭Spring IoC容器

```

1 /**
2  * 生命周期回调
3  * 1. 使用接口实现的方式来实现生命周期的回调:
4  *
5  * 1.1 初始化方法: 实现接口: InitializingBean 重写afterPropertiesSet方法 初始化:

```

```

5  *      1.1  销毁的方法： 实现接口： DisposableBean  重写destroy 方法    销毁的时候自动调用方
6  *      什么时候销毁： 在spring容器关闭的时候  close()
7  *      或者 使用ConfigurableApplicationContext.registerShutdownHook方法优雅的
8  *
9  *  2. 使用指定具体方法的方式实现生命周期的回调：
10 *      在对应的bean里面创建对应的两个方法
11 *      init-method="init" destroy-method="destroy"
12 */

```

- 1.6.2. ApplicationContextAware 和 BeanNameAware
- 1.6.3. 其他的 Aware 接口
- 1.7. Bean 定义的继承

```

1  <!--bean的继承 一个bean继承另一个bean
2  可以使用parent属性指定父类bean
3  如果想让父类bean不能被实例化  abstract="true"
4
5  <bean class="cn.tulingxueyuan.beans.Person" id="person4"  abstract="true">
6      <property name="id" value="1"></property>
7      <property name="realName" value="吴彦祖"></property>
8      <property name="name" value="徐庶"></property>
9  </bean>
10
11 <bean class="cn.tulingxueyuan.beans.Person" id="person5"  parent="person4" >
12     <property name="realName" value="刘德华"></property>
13 </bean>-->

```

- ~~1.8.容器的扩展点~~ (结合源码讲解)
 - ~~1.8.1. 使用BeanPostProcessor自定义Bean~~
 - ~~例子: BeanPostProcessor风格的Hello World~~
 - ~~例子: RequiredAnnotationBeanPostProcessor~~
 - ~~1.8.2. 使用BeanFactoryPostProcessor自定义元数据配置~~
 - ~~例子: 类名替换n-PropertyPlaceholderConfigurer~~
 - ~~例子: PropertyOverrideConfigurer~~
 - 1.8.3. 使用FactoryBean自定义初始化逻辑

spring创建第三方bean对象

在Spring中，很多对象都是单实例的，在日常的开发中，我们经常需要使用某些外部的单实例对象，例如数据库连接池，下面我们来讲解下如何在spring中创建第三方bean实例。

1、导入数据库连接池的pom文件

```
1 <!-- https://mvnrepository.com/artifact/com.alibaba/druid -->
2 <dependency>
3     <groupId>com.alibaba</groupId>
4     <artifactId>druid</artifactId>
5     <version>1.1.21</version>
6 </dependency>
7 <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
8 <dependency>
9     <groupId>mysql</groupId>
10    <artifactId>mysql-connector-java</artifactId>
11    <version>5.1.47</version>
12 </dependency>
```

2、编写配置文件

ioc.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springf
5
6     <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
7         <property name="username" value="root"></property>
8         <property name="password" value="123456"></property>
9         <property name="url" value="jdbc:mysql://localhost:3306/demo"></property>
10        <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
11    </bean>
12 </beans>
```

3、编写测试文件

```
1
2 public class MyTest {
3     public static void main(String[] args) throws SQLException {
4
5         ApplicationContext context = new ClassPathXmlApplicationContext("ioc3.xml");
```



```
5      DruidDataSource dataSource = context.getBean("dataSource", DruidDataSource.class)
6      System.out.println(dataSource);
7      System.out.println(dataSource.getConnection());
8  }
9  }
```

spring引用外部配置文件

在resource中添加dbconfig.properties

```
1  username=root
2  password=123456
3  url=jdbc:mysql://localhost:3306/demo
4  driverClassName=com.mysql.jdbc.Driver
```

编写配置文件

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans
6      http://www.springframework.org/schema/beans/spring-beans.xsd
7      http://www.springframework.org/schema/context
8      http://www.springframework.org/schema/context/spring-context.xsd">
9  <!-- 加载外部配置文件
10  在加载外部依赖文件的时候需要context命名空间
11  -->
12  <context:property-placeholder location="classpath:dbconfig.properties"/>
13  <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
14      <property name="username" value="${username}"></property>
15      <property name="password" value="${password}"></property>
16      <property name="url" value="${url}"></property>
17      <property name="driverClassName" value="${driverClassName}"></property>
18  </bean>
19 </beans>
20
```

SpEL的使用

SpEL:Spring Expression Language, spring的表达式语言，支持运行时查询操作对象使用#{...}作为语法规则，所有的大括号中的字符都认为是SpEL.

```
1      <bean id="user" class="cn.tulingxueyuan.entity.User">
2          <!--支持任何运算符-->
3          <property name="id" value="#{12*2}"></property>
4          <!--可以引用其他bean的某个属性值-->
5          <property name="name" value="#{address.province}"></property>
6          <!--引用其他bean-->
7          <property name="role" value="#{address}"></property>
8          <!--调用静态方法-->
9          <property name="hobbies" value="#{T(java.util.UUID).randomUUID().toString().subst
10          <!--调用非静态方法-->
11          <property name="gender" value="#{address.getCity()}"></property>
12 </bean>
```