# 尚马教育 JAVA 课程

# Spring类配置

文档编号：C04\_1

创建日期： 2021-01-21

最后修改日期：2021-03-10

版 本 号：V3.6

电子版文件名：尚马教育-第三阶段-4\_1.spring类配置.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2021-01-21 | 冯勇涛 | 加入spring类配置容器 | V3.5 |
| 2021-03-10 | 冯勇涛 | 加入dom4j解析xml | V3.6 |

目录

[尚马教育 JAVA 课程 1](#_Toc32061)

[Spring类配置 1](#_Toc22939)

[1. 类配置取代xml配置 3](#_Toc8716)

[2. class类配置环境 3](#_Toc19210)

[2.1. 创建配置类：AppConfig .java 3](#_Toc14411)

[2.1.1. 测试代码 5](#_Toc10827)

知识点：

|  |
| --- |
| 知识点1：认识spring类配置常用注解。 |

## 类配置取代xml配置

Spring在实际开发中有两种应用方式：

基于xml的配置：一般在ssm项目中使用。

基于类的配置：一般在springBoot项目中使用。

## class类配置环境

### 创建配置类：AppConfig .java

主要注解：

@Configuration，@Bean，@Value,@ComponentScan,@Import,@EnableAspectJAutoProxy

|  |
| --- |
| */\*\*  \* 该类的职责与spring.xml的职责是一样  \* 1.声明该类是一个配置类@Configuration  \* 2.包扫描@ComponentScan  \* 3.引入外部资源文件@PropertySource  \* 4.嵌套配置类@Import  \* 5.注册bean对象进入容器@Bean  \* 6.获取容器中的property数据@Value*  *\* 7.@EnableAspectJAutoProxy 开启aop注解识别  \*/* @Configuration @ComponentScan(**"com.javasm"**) @Import(DaoConfig.**class**) @PropertySource(value=**"classpath:jdbc.properties"**,ignoreResourceNotFound = **true**)  @EnableAspectJAutoProxy **public class** ContextConfig {  @Value(**"${jdbc.url}"**)  **private** String **url**;  @Value(**"${jdbc.driver}"**)  **private** String **driver**;  @Value(**"${jdbc.username}"**)  **private** String **username**;  @Value(**"${jdbc.password}"**)  **private** String **password**;   *//先按照形参名找，再按照形参类型找bean进行注入* @Bean  **public** SysuserController createUserController(ISysuserService sysuserService){  SysuserController sysuserController = **new** SysuserController();  sysuserController.setSysuserService(sysuserService);  **return** sysuserController;  }   @Bean  **public** ISysuserService createUserService(){  **return new** SysuerServiceImpl();  }   @Bean(initMethod = **"init"**,destroyMethod = **"close"**) *// @Scope("prototype")* **public** DataSource initDruid(){  DruidDataSource ds = **new** DruidDataSource();  ds.setUrl(**url**);  ds.setDriverClassName(**driver**);  ds.setUsername(**username**);  ds.setPassword(**password**);  ds.setInitialSize(2);  **return** ds;  } |

#### 测试代码

|  |
| --- |
| @Test **public void** test1\_loadClzConfig(){  ApplicationContext ac = **new** AnnotationConfigApplicationContext(ContextConfig.**class**);  SysuserController bean = ac.getBean(SysuserController.**class**);  ((AnnotationConfigApplicationContext) ac).close(); } |

## Dom4J解析xml文件

Mybatis.spring等框架都采用xml作为配置文件,那么在框架内部必须要进行xml解析,这里学习如果解析xml.

### 常用组件

PC端解析xml,一般使用dom4j解析;

移动端解析xml,一般使用SAX解析.

### 特点

DOM4j解析XML文件时，会将XML的所有内容读取到内存中，然后使用DOM API遍历XML树、检索所需的数据。因为DOM需要将所有内容读取到内存中，所以内存的消耗比较大，一般服务器端使用;

SAX（Simple API for XML）是一个解析速度快并且占用内存少的XML解析器，非常适合用于Android等移动设备。SAX解析XML文件采用的是事件驱动，也就是说，它并不需要解析完整个文档，在按内容顺序解析文档的过程中，SAX会判断当前读到的字符是否合法XML语法中的某部分，如果符合就会触发事件。

### Dom4j代码

学习dom4j只要求熟记核心SAXReader对象.类似于Properties核心对象.

这里以解析spring风格xml文件为例:

#### Spring.xml

|  |
| --- |
| 1. **Jdbc.properties文件如下:**   **jdbc.url**=**jdbc:mysql://127.0.0.1:3306/720A?useUnicode=true&characterEncoding=utf8&useSSL=true&serverTimezone=UTC jdbc.driver**=**com.mysql.jdbc.Driver jdbc.username**=**root jdbc.password**=**root** |
| 1. **Spring.xml文件如下:**   *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:aop="http://www.springframework.org/schema/aop"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-aop.xsd"**>   <**context:component-scan base-package="com.javasm4"**></**context:component-scan**>  <**aop:aspectj-autoproxy**></**aop:aspectj-autoproxy**>   <**context:property-placeholder location="jdbc.properties"**></**context:property-placeholder**>   <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"**>  <**property name="url" value="${jdbc.url}"**></**property**>  <**property name="driverClassName" value="${jdbc.driver}"**></**property**>  <**property name="username" value="${jdbc.username}"**></**property**>  <**property name="password" value="${jdbc.password}"**></**property**>  <**property name="initialSize" value="3"**></**property**>  </**bean**> </**beans**> |

#### 解析xml,模拟ApplicationContext容器对象

解析sprng.xml文件,创建出bean对象.

|  |
| --- |
| **package** com.javasm3;  **import** org.dom4j.Document; **import** org.dom4j.DocumentException; **import** org.dom4j.Element; **import** org.dom4j.io.SAXReader; **import** java.io.IOException; **import** java.io.InputStream; **import** java.lang.reflect.Field; **import** java.lang.reflect.InvocationTargetException; **import** java.lang.reflect.Method; **import** java.util.\*;  **public class** MyApplicationContext {  **private** Map<String,String> **properties** = **new** HashMap<>();  **private** Map<String,Object> **beans** = **new** HashMap<>();  **public** MyApplicationContext(String path){  InputStream in = TestDom4j.**class**.getClassLoader().getResourceAsStream(path);  **//DOM4j入口对象** SAXReader reader = **new** SAXReader();  **//整个文档对象** Document doc=**null**;  **try** {  doc = reader.read(in);  } **catch** (DocumentException e) {  e.printStackTrace();  }   **//标签元素对象,根标签** Element root = doc.getRootElement();   **//解析子标签** List<Element> propertyElements = root.elements(**"property-placeholder"**);  List<Element> beanElements = root.elements(**"bean"**);   parsePropertyElements(propertyElements);  parseBeanElements(beanElements);  }   **private void** parsePropertyElements(List<Element> propertyElements){  Properties p = **new** Properties();  ClassLoader classLoader = TestDom4j.**class**.getClassLoader();  **for** (Element propertyElement : propertyElements) {  String location = propertyElement.attributeValue(**"location"**);**//jdbc.properties** InputStream in = classLoader.getResourceAsStream(location);  **try** {  p.load(in);  Set<Object> objects = p.keySet();  **for** (Object key : objects) {  String value = p.getProperty((String) key);  addProperty((String)key,value);  }  } **catch** (IOException e) {  e.printStackTrace();  }  }  }   **private void** parseBeanElements(List<Element> beanElements) {  **for** (Element beanElement : beanElements) {  String id = beanElement.attributeValue(**"id"**);  String aClass = beanElement.attributeValue(**"class"**);  String initMethod = beanElement.attributeValue(**"init-method"**);   **//1.初始化bean对象** Object bean = newInstance(aClass);  **//2.给对象中属性赋值(解析property标签)** parseProperty(beanElement,bean);  **//3.执行init-method方法** initBeanMethod(bean,initMethod);  **//4.注册bean** addBean(id,bean);  }  }   **private void** initBeanMethod(Object bean,String initmethodName){  **try** {  Method method = bean.getClass().getMethod(initmethodName);  method.invoke(bean);  } **catch** (NoSuchMethodException e) {  e.printStackTrace();  } **catch** (IllegalAccessException e) {  e.printStackTrace();  } **catch** (InvocationTargetException e) {  e.printStackTrace();  }  }   **/\*\*  \* 解析property标签  \* @param *beanElement* \* @param *bean* \*/  private void** parseProperty(Element beanElement,Object bean){  List<Element> property = beanElement.elements(**"property"**);   **for** (Element element : property) {  String propertyName = element.attributeValue(**"name"**);**//bean中的属性名** String value = element.attributeValue(**"value"**);**//值的表达式${}  if**(value.startsWith(**"${"**) && value.endsWith(**"}"**)){  String propertyKey = value.substring(2,value.length()-1);  **//给bean对象中的属性名,赋值** value = getProperty(propertyKey);  }  setPropertyValue2Bean(bean,propertyName,value);  }  }   **/\*\*  \* 给bean对象中某属性赋值  \* @param *bean* 对象  \* @param *name* 属性名  \* @param *value* 属性值  \*/  private void** setPropertyValue2Bean(Object bean,String name,String value){  Class<?> aClass = bean.getClass();  **//获取propertyName的属性类型** Class<?> fieldType = **null**;  **try** {  Field field = aClass.getField(name);  fieldType = field.getType();  } **catch** (NoSuchFieldException e) {  e.printStackTrace();  }   **//拼接对应set方法名** String setMethodName = **"set"**+name.substring(0,1).toUpperCase()+name.substring(1);   **//调用set方法进行赋值  try** {  Method method = aClass.getMethod(setMethodName, fieldType);  method.invoke(bean,value);  } **catch** (NoSuchMethodException e) {  e.printStackTrace();  } **catch** (IllegalAccessException e) {  e.printStackTrace();  } **catch** (InvocationTargetException e) {  e.printStackTrace();  }  }   **//初始化bean对象  private** Object newInstance( String aClass){  **try** {  Class<?> c = Class.*forName*(aClass);  **return** c.newInstance();  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  } **catch** (IllegalAccessException e) {  e.printStackTrace();  } **catch** (InstantiationException e) {  e.printStackTrace();  }  **return null**;  }   **public void** addProperty(String key,String value){  **properties**.put(key,value);  }  **public void** addBean(String key,Object bean){  **beans**.put(key,bean);  }   **public** String getProperty(String key)  {  **return properties**.get(key);  }  **public** Object getBean(String id){  **return beans**.get(id);  }  **public** <T> T getBean(String id,Class<T> clz){  **return** (T)**beans**.get(id);  }  **public** <T> T getBean(Class<T> clz){  Collection<Object> values = **beans**.values();  **for** (Object value : values) {  **if**(clz.isAssignableFrom(value.getClass())){  **return** (T)value;  }  }  **return null**;  } } |

#### 测试

|  |
| --- |
| **public static void** main(String[] args) {  MyApplicationContext ac = **new** MyApplicationContext(**"spring.xml"**);  DataSource bean = ac.getBean(DataSource.**class**);  System.***out***.println(bean); } |