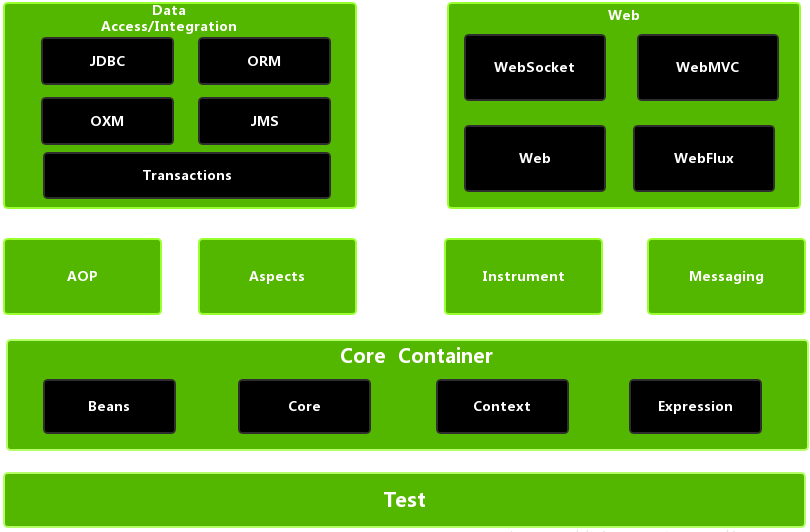
**Spring文档**

**更新于：2019.08.05 (基于spring5.x)**

# Spring架构模块

Spring总共有20个模块,1300多个不同的文件构成。而这些组件被分别整合在核心容器(core container)、AOP(Aspect Oriented Programming)和设备支持(Instrmentation)、数据访问及集成（数据访问及集成）、Web、报文发送(Messaging)、Test、6个模块集合中。



# Spring是什么可以做什么

## Spring是什么

Spring是一个轻量级的依赖注入、控制翻转和面向切面编程的框架，可以用它来快速的构建Java应用程序。

## Spring可以做什么

### 依赖注入

1. 减少粘合代码

不需要通过new直接创建对象，只需在Spring容器中查找并注入即可

1. 简化应用程序配置

比如我们需要更换一个数据库，我们只需要将配置文件里面的配置更改即可。

1. 能够在单个存储库中管理常见的依赖项

我们在创建对象时，免不了需要依赖其他的实例对象。使用依赖注入只需要 在一处地方创建它依赖项即可。

1. 改进的可测试性
2. 培养良好的应用程序设计

### **支持Java9**

### **使用Spring面向方面编程(AOP)**

### **Spring表达式语言**

### **Spring验证**

### **Spring中访问数据**

### **Spring中的对象/XML映射**

### **Spring中管理事物**

### **Spring中WebSocket支持**

### **Spring远程支持**

### **Spring邮件支持**

### **Spring作业调度支持**

### **Spring简化异常处理**

## Spring项目

1. Spring security
2. Spring Boot
3. Spring批处理和集成

## Spring模块介绍

|  |  |
| --- | --- |
| **AOP** | 包含spring的aop功能时所需的所有类。 |
| **aspectS** | AspectJ AOP库进行高级集成的所有类。 |
| **Beas** | 包含所有支持Spring对Spring bean进行操作的类。例如：处理Spring Xml 配置文件和Java注解所需要的类封装在此 |
| **Context** | 该模块包含为Spring core提供许多扩展的类。 |
| **Context-indexer** | 该模块含一个索引实现，它提供对META-INF/spring.components中定义的候选项的访问功能。 |
| **Context-support** | 该模块对spring-context模块的进一步扩展。例如:支持邮件与模板引擎集成的类、任务执行和调度库。 |
| **core** | 这是每个Spring应用程序都需要的主要模块. |
| **expression** | SpEL的所有支持类 |
| **instrument** | 该模块包含用于JVM启动的Spring工具代理。如果Spring应用程序 |
| **jdbc** | 该模板包含所有的JDBC支持类。对于需要数据库访问的所有应用程序都需要此模块。支持数据源、JDBC数据类型、JDBC模板、本地JDBC连接等的类都被打包在此模块中。 |
| **orm** | 该模块扩展了Spring的标准JDBC功能集，支持流行的ORM框架，包括hibernate、jdo、jpa |
| **oxm** | 该模块Object/xml映射(OXM)提供支持。 |
| **test** | Spring提供一组模拟类来帮助测试应用程序 |
| **tx** | 该模块提供支持spring事物的基础框架的所有类。可以从事物抽象层找到相应的类来支持JTA |
| **Web** | 此模块包含在web应用程序中使用Spring所需的核心类，包括用于自动加载ApplicationContext功能的类、文件上传支持类以及一些用于执行重复任务 |
| **Web-reactive** | 该模块包含spring web reactive模型的核心接口和类 |

# Spring中的IoC和DI

**官方文档：**

<https://docs.spring.io/spring/docs/5.1.12.RELEASE/spring-framework-reference/core.html#beans>

## 控制反转(IoC)和依赖注入(DI)是什么关系？

依赖注入是控制反转的核心，旨在提供一种更简单的方式来设置组件的依赖项，并在整个生命周期管理这些依赖项。

依赖注入是控制反转的一种特殊形式.所以说控制反转有多种类型。

## 控制反转(IOC)有哪些类型？

**1.依赖查找（包含两种形式）：**

**依赖拉取：**我们常常用的Spring中的ApplicationContext对象的getBean方法就是用的依赖拉取，是通过JNDI注册表来查找。

**依赖查找：**依赖查找是用过容器来直接查找。

**2.依赖注入（包含两种形式）：**

**构造函数注入：**

1.当要使用一个组件之前必须拥有一个依赖类的实例时

2.实现不可变对象的使用

**setter注入：** setter注入适用于一些配置参数的注入

## Spring是怎么实现IOC?

Spring中的IOC实现核心是依赖注入，不过也提供了依赖查找

Spring中的依赖注入主要是通过构造注入和setter注入以及自动装配实现。

## 使用Spring管理依赖的一个流程?

**第一步**：**注册组件(Spring有三种模式配置组件依赖：xml、自动扫描、javaConfig)**

**第二步**：**实例化容器(这个步骤在web应用里面就交给中间件来处理了)**

**第三步**：就可以使用spring管理的这些组件了

## Spring容器主要交给哪个类来管理？

Spring依赖注入容器的核心是BeanFactory接口。BeanFactory负责管理组件，包括依赖项以及他们的生命周期。

如果只需要用到Spring的DI功能，则可以通过BeanFactory接口和Spring DI容器交互。

在SpringMVC中, 通过在web.xml里面声明ContextLoaderListener类启动Spring的ApplicationContext.

**代码示例:**

DefaultListableBeanFactory beanFactory = **new** DefaultListableBeanFactory();  
XmlBeanDefinitionReader xdr = **new** XmlBeanDefinitionReader(beanFactory);  
xdr.loadBeanDefinitions(**new** ClassPathResource(**"spring.xml"**));  
  
UserService userService = (UserService) beanFactory.getBean(**"userService"**);  
userService.test();

## Spring中的ApplicationContext是什么？

**ApplicationContext是BeanFactory接口的一个扩展包含：**

**DI服务、事物、AOP服务、国际化(i18n)等功能**

## Spring中bean的配置

### Spring中bean的配置定义

|  |  |
| --- | --- |
| Class | bean的类路径 |
| Name | bean的名称，可以有多个名称，使用逗号隔开 |
| Scope | **Bean范围：**  单例(singleton)、  原型(prototype)、  单次http请求(request)、  Http生命周期(session)、  ServletContext生命周期(application)、  websocket |
| Constructor arguments | Bean的构造参数 |
| Properties | bean的属性（类里面的字段） |
| Autowiring mode | 定义自动装配的模式，byName 或者byType |
| Lazy initialization mode | Bean懒加载 |
| Initialization method | Bean的初始化方法 |
| Destruction method | Bean的销毁回调 |
|  |  |

### 构造注入和属性注入

#### **1.简单的属性注入和构造注入**

<**bean id="userDao" class="learn.spring.ioc.dao.UserDao"**>  
 <**property name="userName" value="渣渣"** />  
</**bean**>  
  
<**bean id="userService" class="learn.spring.ioc.service.UserService"**>  
 <**constructor-arg ref="userDao"**/>  
</**bean**>

#### **2.使用facotry-method配置工厂方法创建bean（工厂方法必须是静态的）**

**public class** UserService {  
  
 **private** UserDao **userDao**;  
 **private int count**;  
  
 **public** UserService(UserDao userDao) {  
 **this**.**userDao** = userDao;  
 }  
  
 **public void** test(){  
 **userDao**.find();  
 System.***out***.println(**count**);  
 }  
  
 **public int** getCount() {  
 **return count**;  
 }  
  
 **public void** setCount(**int** count) {  
 **this**.**count** = count;  
 }  
  
 **public static** UserService createInstance(UserDao userDao, **int** count){  
 System.***out***.println(**"使用工厂方法创建bean"**);  
 UserService userService = **new** UserService(userDao);  
 userService.setCount(count);  
 **return** userService;  
 }  
}

<**bean id="userDao" class="learn.spring.ioc.dao.UserDao"**>  
 <**property name="userName" value="渣渣"** />  
</**bean**>  
  
<**bean id="userService" class="learn.spring.ioc.service.UserService" factory-method="createInstance"**>  
 <**constructor-arg ref="userDao"**/>  
 <**constructor-arg value="1"**/>  
</**bean**>

#### **3.关于list、set、map、propterties的详细配置**

<**bean id="userDao" class="learn.spring.ioc.dao.UserDao"**>  
 <**property name="userName" value="渣渣"** />  
 <**property name="names"**>  
 <**list**>  
 <**value**>张三</**value**>  
 <**value**>李思</**value**>  
 <**value**>王五</**value**>  
 </**list**>  
 </**property**>  
 <**property name="sorts"**>  
 <**set**>  
 <**value**>1</**value**>  
 <**value**>2</**value**>  
 <**value**>3</**value**>  
 </**set**>  
 </**property**>  
 <**property name="types"**>  
 <**map**>  
 <**entry key="sex" value="武威"** />  
 <**entry key="place" value="水电费"** />  
 <**entry key="age" value="9999"** />  
 </**map**>  
 </**property**>  
 <**property name="properties"**>  
 <**props**>  
 <**prop key="name"**>渣渣</**prop**>  
 <**prop key="age"**>1000</**prop**>  
 </**props**>  
 </**property**>  
</**bean**>

#### **4.null设置**

<**property name="userName"**>  
 <**null**/>  
</**property**>

#### **使用p-namespace命名空间配置属性注入**

**首先需要加上命名空间定义**

**xmlns:p="http://www.springframework.org/schema/p"**

**配置**

<**bean id="jobDao" class="learn.spring.ioc.dao.JobDao"**

**p:type="1" p:name="傻仔" p:description="是个傻吊"** />

#### **使用c-namespace命名空间配置构造注入**

**首先需要加上命名空间定义**

**xmlns:c="http://www.springframework.org/schema/c"**

**配置**

<**bean id="deptDao" class="learn.spring.ioc.dao.DeptDao"**

**c:deptName="研发" c:deptMgr="渣渣辉"**/>

### 自动装配

| **模式** | **说明** |
| --- | --- |
| no | （默认）无自动装配。 |
| byName | 按属性名称自动装配 |
| byType | 按bean类型装配，如果类型有多个会装配失败 |
| constructor | 类似于byType但适用于构造函数参数。 |

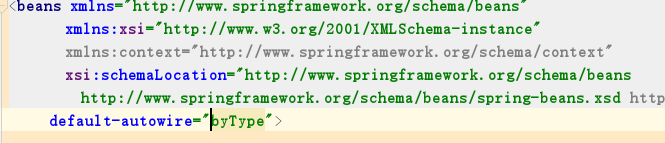
**配置：**

**public class** UserService {  
  
 **private** Dao **userDao**;  
 **private** Dao **jobDao**;  
  
 **public void** test(){  
 **userDao**.test();  
 **jobDao**.test();  
 }  
  
 **public** Dao getUserDao() {  
 **return userDao**;  
 }  
  
 **public void** setUserDao(Dao userDao) {  
 **this**.**userDao** = userDao;  
 }  
  
 **public** Dao getJobDao() {  
 **return jobDao**;  
 }  
  
 **public void** setJobDao(Dao jobDao) {  
 **this**.**jobDao** = jobDao;  
 }  
}

<**bean id="deptDao" class="learn.spring.ioc.dao.DeptDao"** />  
  
<**bean id="jobDao" class="learn.spring.ioc.dao.JobDao"** />  
  
<**bean id="userDao" class="learn.spring.ioc.dao.UserDao"**/>  
  
<**bean id="userService" class="learn.spring.ioc.service.UserService" autowire="byName"**/>

**或者**

**全局配置：**



**注意：依赖Dao必须要添加set方法，否则无法自动装配，且byName类型跟set方法名称有关跟属性名无关**

### 自动扫描

#### 1. 组件标记

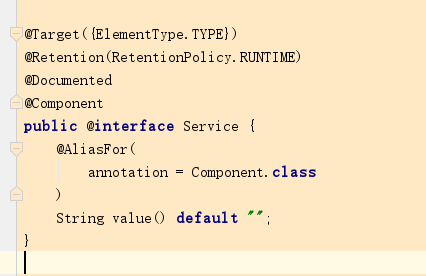
**Spring提供了一套注解用于标记组件：**

**@Service注解： 用于标记表示服务层的组件**

**@Controller注解： 用于标记表示控制器层的组件**

**@Repository注解： 用于标记表示持久层的组件**

**其实以上三个注解都是由**@Component**元注解标记**



@Component 表示组件， 其他无法表示具体含义的组件可以用@Component注解标记。

另外还可以使用@Component作为元注解创建特定含义的标记注解。

#### 2. 组件扫描

**Spring可以启动扫描，扫描被标记的组件将其注册到Spring容器中去，**

1. **通过xml配置**

*<?***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"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"**>  
  
 <**context:component-scan base-package="learn.spring.ioc"** />  
  
</**beans**>

1. **通过javaConfig配置**

@Configuration  
@ComponentScan(basePackages = **"learn.spring.ioc"**)  
**public class** ScanConfig {  
  
}

1. **过滤器配置**

**过滤器类型：**

| **过滤器类型** | **示例** | **描述** |
| --- | --- | --- |
| annotation (默认) | org.example.SomeAnnotation | 指定一个注解 |
| assignable | org.example.SomeClass | 指定接口或类 |
| aspectj | org.example..\*Service+ | aspectj表达式 |
| Regex(正则) | org\.example\.Default.\* | 正则表达式 |
| Custom(自定义) | org.example.MyTypeFilter | 自定义过滤器 |

**组件可以设置过滤器，可以根据实际需求只扫描你需要的组件**

@Configuration  
@ComponentScan(basePackages = **"learn.spring.ioc"**,  
 includeFilters = @ComponentScan.Filter(type = FilterType.***ANNOTATION***, value = Repository.**class**),  
 excludeFilters = @ComponentScan.Filter(type = FilterType.***REGEX***, pattern = **".\*Repository"**))  
**public class** ScanConfig {  
  
}

### 基于注解和javaConfig的容器配置