**Spring入门篇**

**2018年5月8日**

[第一章 Spring入门 3](#_Toc514054698)

[第二章 概述 3](#_Toc514054699)

[一 Spring入门课程简介 3](#_Toc514054700)

[二 Spring概况 3](#_Toc514054701)

[三 Spring Framework Runtime 4](#_Toc514054702)

[四 Spring的作用 4](#_Toc514054703)

[五 Spring适用范围 4](#_Toc514054704)

[六 Spring框架 5](#_Toc514054705)

[第三章 Spring IOC容器 5](#_Toc514054706)

[一 IOC及Bean容器 5](#_Toc514054707)

[二 Spring的常用注入方式 7](#_Toc514054708)

[第四章 Spring Bean装配（上） 9](#_Toc514054709)

[一 Spring Bean装配之Bean的配置项及其作用域 9](#_Toc514054710)

[二 Spring Bean装配之Bean的生命周期 10](#_Toc514054711)

[三 Spring Bean装配之Aware接口 11](#_Toc514054712)

[四 Spring Bean装配之自动装配 13](#_Toc514054713)

[五 Spring Bean装配之Resources 14](#_Toc514054714)

[第五章 Spring Bean装配（下） 15](#_Toc514054715)

# 第一章 Spring入门

Spring入门包含哪些内容？

1. 什么是框架
2. Spring简介
3. IOC（配置、注解）
4. Bean（配置、注解）
5. AOP（配置、注解、Aspectj、API）

# 第二章 概述

## 一 Spring入门课程简介

1. 如何学习Spring？
   1. 掌握用法
   2. 深入理解
   3. 不断实践
   4. 反复总结
   5. 再次深入理解与实践
2. 资源：

<http://spring.io/>

<http://projects.spring.io/spring-framework/>

其他……

## 二 Spring概况

1. Spring是一个开源框架，为解决企业应用程序开发的复杂性而创建
2. Spring是一个轻量级的控制反转（IOC）和面向切面（AOP）的容器框架
   1. 从大小和开销两方面而言，Spring都是轻量级的
   2. 通过控制反转（IOC）的技术达到松耦合的目的
   3. 提供了面向切面编程的丰富支持，允许通过分离应用的业务逻辑与系统服务进行内聚性的开发
   4. 包含并管理应用对象的配置和生命周期，这个意义上是一种容器
   5. 将简单的组件配置，组合成为复杂的应用，这个意义上是框架
3. Spring的好处：
   1. 在Spring上开发应用：简单、方便、快捷（以面向接口编程为理念）
   2. Spring为复杂的JavaEE开发带来了春天
4. Spring管理的对象统称为Bean，管理对象的容器统称为Bean容器

## 三 Spring Framework Runtime

Test（单元测试）

Context

上下文

SpEL

Spring表达式

Core

容器

Beans

Bean

Core Container

Messaging（消息）

Instrumentation

Aspects

AOP

Transactions

portlet

web

OXM

JMS

JDBC

ORM

servlet

websocket

Web

数据访问/集成

Data Access/Integration

## 四 Spring的作用

1. 容器
2. 提供了对多种技术的支持
   1. —JMS
   2. —MQ支持
   3. —UnitTest

……

1. AOP（事务管理、日志等）
2. 提供了众多方便应用的辅助类（JDBC Template等（模板化的应用））
3. 对主流应用框架（Hibernate、Struts等）提供了良好的支持

## 五 Spring适用范围

1. 构建企业应用（SpringMVC（MVC框架）+Spring（Framework）+Hibernate/ibatis/mybatis）
2. 单独使用Bean容器（Bean管理）
3. 单独使用AOP进行切面处理
4. 其他的Spring功能，如：对消息的支持等（包括对xml的映射）
5. 在互联网中的应用

## 六 Spring框架

1. 什么是框架？

形象比喻：别人搭好舞台，你来表演

1. 框架特点
   1. —半成品
   2. —封装了特定的处理流程和控制逻辑
   3. —成熟的、不断升级改进的软件
2. 框架与类库的区别
   1. —框架一般是封装了逻辑、高内聚的，类库则是松散的工具组合
   2. —框架专注于某一领域，类库则是更通用的
3. 为什么使用框架
   1. 软件系统日趋复杂
   2. 重用度高，开发效率和质量提高
   3. 软甲设计人员要专注于对领域的了解，是需求分析更充分
   4. 易于上手、快速解决问题

# 第三章 Spring IOC容器

## 一 IOC及Bean容器

1. 接口及面向接口编程
   1. 什么是接口？
2. 用于沟通的中介物的抽象化
3. 实体把自己提供给外界的一种抽象化说明，用以由内部操作分离出外部沟通方法，使其能被修改内部而不影响外界其他实体与其交互的方式
4. 对应Java接口既声明，声明了哪些方法是对外公开提供的
5. 在Java8中接口可以拥有方法体
   1. 什么是面向接口编程？
6. 结构设计中，分清层次及调用关系，每层只向外（上层）提供一组功能接口，各层间仅依赖接口而非实现类
7. 接口实现的变动不影响各层间的调用，这一点在公共服务中尤为重要
8. “面向接口编程”中的“接口”是用于隐藏具体实现和实现多态性的组件
9. 什么是IOC？
   1. IOC（控制反转）：控制权的转移，应用程序本身并不负责依赖对象的创建和维护，而是由外部容器负责创建和维护，依赖注入是其一种实现方式
   2. DI（依赖注入）：就是IOC容器在运行期间，动态地将某种依赖关系注入到对象之中
   3. 目的：创建对象并且组装对象之间的依赖关系
   4. 图解：

Fully Configured System（符合系统配置）

Ready For User

Configuration Metadata

配置元数据

Your Business Object（Pojos）

（业务对象）

The Spring Container

The Spring IOC Container

* 1. IOC容器形象比喻：

IOC ：找IOC容器 容器返回对象 使用对象

房屋中介：找中介 中介介绍房子 租房入住

* 1. 扩展理解（Martin Fowler）：获得依赖对象的过程被反转了，控制被反转后，获得依赖对象的过程由自身管理变为IOC容器主动注入，也可称为“依赖注入”（Dependency Injection）

1. Spring的Bean配置

<?xml version = "1.0" encoding = "UTF-8"?> //版本、编码方式

//命名空间和xml:schema的位置

<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

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id = "oneInterface" class = "com.imooc.ioc.interfaces.OneInterfaceImpl"></bean>

</beans>

注：

1. IOC机制提供对象，不必自己创建

2. 实现了面向接口编程

3. IOC隐藏了对象，无需手动管理

Bean对应的具体的类

这个Bean的唯一标识

1. 单元测试
   1. 下载junit-\*.jar并引入工程
   2. 创建UnitTestBase类，完成对spring配置文件的加载、销毁
   3. 创建具体执行单元测试的类，以及类的方法，所有的单元测试类都继承自UnitTestBase，加注解（@RunWith（BlockJUnit4ClassRunner.class）），构造无参的构造方法，通过super向UnitTestBase传入参数（super（”classpath\*:spring-ioc.xml”）），通过它的getBean方法获取想要得到的对象
   4. 执行时可指定方法执行，也可以执行整个类，此时会依次执行每个方法，单元测试方法也需要加注解（@Test）
2. Bean的初始化
   1. 基础：两个包
3. org.springframework.beans
4. org.springframework.context
5. BeanFactory提供配置结构和基本功能，加载并初始化Bean
6. ApplicationContext保存了Bean对象并在spring中被广泛应用
   1. 方式：Application
7. 本地文件：

FileSystemXmlApplicationContext context = new

FileSystemXmlApplicationContext(F:/workspace/appcontext.xml);

1. Classpath：

ClassPathXmlApplicationContext context =

new ClassPathXmlApplicationContext(“classpath:spring-context.xml”);

1. Web应用中依赖servlet或listener
   1. Servlet：

<servlet>

<servlet-name>context</servlet-name>

<servlet-class>org.springframework.web.Context.

ContextLoaderServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

* 1. Listener:

<listener>

<listener-class>org.springframework.web.Context.

ContextLoaderListener</listener-class>

</listener>

## 二 Spring的常用注入方式

1. Spring注入是指在启动Spring容器加载bean配置的时候，完成对变量的赋值行为，

常用的两种注入方式如下：

—设值注入

—构造注入

1. 理解：
   1. IOC容器加载过程：加载时会扫描磁盘文件里面Bean的配置，并对这些Bean

进行实例化和初始化。

1. 举例：A类中引用B类，声明B类对象b，注入是指在IOC容器初始化A的时候，

就把A的成员变量b进行一个赋值，这就是spring的注入。

1. 设值注入：

**<?xml version="1.0" encoding="UTD-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**

**http://www.springframework.org/schema/beans/spring-beans.xsd">**

**<bean id="injectionService" class="com.imooc.ioc.injection.service.Injecti**

**onServiceImpl">**

**//service中必须有injectionDAO的成员变量和set方法**

**<property name="injectionDAO" ref="injectionDAO"/>**

**</bean>**

**<bean id="injectionDAO" class="com.imooc.ioc.injection.dao.injectio**

**nDAOImpl">**

**</bean>**

**</beans>**

1. 构造注入：

**<?xml version="1.0" encoding="UTD-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**

**http://www.springframework.org/schema/beans/spring-beans.xsd">**

**<bean id="injectionService" class="com.imooc.ioc.injection.service.Injecti**

**onServiceImpl">**

**//实例化serviceImpl时，构造器传入参数DAOImpl完成赋值**

**public InjectionServiceImpl(InjectionDAO, injectionDAO){**

**this.injectionDAO = injectionDAO**

**}**

**<constructor-arg name="injectionDAO" ref="injectionDAO"/>**

**</bean>**

**<bean id="injectionDAO" class="com.imooc.ioc.injection.dao.injectio**

**nDAOImpl">**

**</bean>**

**</beans>**

# 第四章 Spring Bean装配（上）

## 一 Spring Bean装配之Bean的配置项及其作用域

1. Bean的配置项
   1. Id：整个IOC容器中Bean的唯一标识
   2. Class：具体要实例化的类（必需）
   3. Scope设置范围，即作用域
   4. Constructor arguments：构造器参数
   5. Properties：属性
   6. Autowiring mode：自动装配模式
   7. Lazy-initialization mode：懒加载模式
   8. Initialization/destruction method：初始化/销毁方法

注：想从Bean容器中获取一个实例有两种方法：

—根据Id获取（配置Id）

—根据Bean的类型来获取（配置class）

1. Bean的作用域

|  |  |
| --- | --- |
| Scope | Description |
| Singleton | (Default)Scopes a single bean definition to a single object instance per Spring IOC container |
| Prototype | Scopes a single bean definition to any number of object instances |
| Request | Scopes a single bean definition to the lifecycle of a single HTTP request; that is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext |
| Session | Scopes a single bean definition to the lifecycle of an HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext |
| Global session | Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext. |

* 1. Singleton：单例，指一个Bean容器中只存在一份

注：ClassPathXmlApplicationContext(SpringXmlpath.split(“[,\\s]+”));

加载xml文件，然后启动上下文（context） 即容器

* 1. Prototype：每次请求（每次使用）创建新的实例（针对同一个容器），destroy

方式不生效

注：本次请求完成后，实例会被回收，不会被继续使用

* 1. Request：每次http请求创建一个实例且仅在当前request内有效
  2. Session：每次http请求创建一个实例且仅在当前session有效
  3. Global session：基于protlet的web中有效（protlet定义了global session），如

果是在web中，同session

注：global session：应用集成级，比如portal中进入其他系统

举例：singleton与prototype模式配置

<bean id=”\*\*\*” class=”\*\*\*” scope=”singleton/prototype”>

注：Junit在执行方法前后一定会分别执行Before（加载）和After（销毁）

## 二 Spring Bean装配之Bean的生命周期

1. 定义：xml中配置的id，class等
2. 初始化：IOC容器加载配置文件，并生成Bean的实例，一共有两种方式：
   1. 实现org.springframework.beans.factory.InitializingBean接口，

覆盖afterPropertiesSet方法

举例：实现接口 只需要在xml中配置这个Bean

public class ExampleInitializingBean implements InitializingBean{

@override

public void afterPropertiesSet() throws Exception{

//do something

}

}

* 1. 配置init-method

举例：

<bean id="exampleInitBean" class="example.ExampleBean"

init-method="init"/>

public class ExampleBean{

public void init(){

//do something initialization work

}

}

1. 使用：从Bean容器中取出一个Bean的实例，并调用它的方法
2. 销毁：在Bean容器停止使用时，销毁当前Bean容器创建的所有Bean的实例，一共有两种方式：
   1. 实现org.springframework.beans.factory.DisposableBean接口，

覆盖destroy方法

举例：

public class ExampleDisposableBean implements DisposableBean{

@override

public void destroy() throws Exception{

//do something

}

}

* 1. 配置destroy-method

举例：

<bean id="exampleDestroyBean" class="example.ExampleBean"

destroy-method="destroy"/>

public class ExampleBean{

public void destroy (){

//do something initialization work（like releasing pooled connections）

}

比如释放连接池

}

1. 配置全局默认初始化、销毁方法
   1. 在xml中配置

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd"

default-init-method="init" default-destroy-method="destroy">

</beans>

注：Java中的方法同上

* 1. 实现InitializingBean，DisposableBean接口，在xml中配置这个Bean，覆盖afterPropertiesSet()和destroy()方法

public class ExampleBean implements InitializingBean,DisposableBean{

public void afterPropertiesSet() throws Exception {

//do something initialization work

}

public void destroy() throws Exception {

//do something initialization work（like releasing pooled connections） }

比如释放连接池

}

注意事项：1. 在默认、手动配置方法以及实现接口三种方法同时使用：

—首先执行接口方法；

—其次执行手动配置方法；

2. 手动配置方法以及实现接口两种方法至少有一种被使用，则默认方

法不生效，此时Java中可以不写此方法（即使已经配置默认方法）。

3. 默认初始化、销毁方法可以不写具体方法（方法体），但其余两种必

须写。

## 三 Spring Bean装配之Aware接口

1. Spring提供了一些以Aware为结尾的接口，实现了Aware接口的Bean在被初始化之后可以获取相应的资源
2. 通过Aware接口，可以对Spring相应资源进行操作（一定要慎重）
3. 为Spring进行简单的扩展提供了方便的入口
4. 常用接口举例：
   1. **ApplicationContextAware：**

向实现了该接口的bean提供上下文对象，故此，Bean必须配置到Spring的Bean配置文件中，并且由Spring的Bean容器加载，以此来实现该效果。

**官方文档解释：**

When an ApplicationContext creates a class that implements the org.springframework.context.ApplicationContextAware interface,the class is provided with a reference to that ApplicationContext.

示例：

* + - 1. ApplicationContextAware的方法重写之后，参数为ApplicationContext类型，此类型为加载了该Bean的IOC容器的上下文信息
      2. 获取该Bean的对象：applicationContext.getBean（配置文件中的ID名）
  1. **BeanNameAware：**

向实现了该接口的bean提供一个关于BeanName定义的一些内容（比如XML中配置的id），故此，Bean必须配置到Spring的Bean配置文件中，并且由Spring的Bean容器加载，以此来实现该效果。

**官方文档解释：**

When an ApplicationContext creates a class that implements the org.springframework.beans.factory.BeanNameAware interface,the class is provided with a reference to the name defined in its associated object definition.

1. 官方文档中的大部分接口

|  |  |
| --- | --- |
| Name | Injected Dependency |
| ApplicationContextAware | Declaring ApplicationContext |
| ApplicationEventPublisherAware | Event publisher of the enclosing  ApplicationContext |
| BeanClassLoaderAware | Class loader used to load the bean classes |
| BeanFactoryAware | Declaring BeanFactory |
| BeanNameAware | Name of the declaring bean |
| BootstrapContextAware | Resource adapter BootstrapContext the container runs in. Typically available only in JCA aware ApplicationContexts |
| LoadTimeWeaverAware | Defined weaver for processing class definition at load time. |
| MessageSourceAware | Configured strategy for resolving messages  （with support for parametrization and  internationalization） |
| NotificationPublisherAware | Spring JMX notification publisher |
| PortletConfigAware | Current PortletConfig the container runs in. Valid  only in a web-aware Spring ApplicationContext |
| PorletContextAware | Current PortletContext the container runs in. Valid  only in a web-aware Spring ApplicationContext |
| ResourceLoaderAware | Configured loader for low-level access to resources |
| ServletConfigAware | Current ServletConfig the container runs in. Valid only in a web-aware Spring ApplicationContext |
| ServletContextAware | Current ServletContext the container runs in. Valid  only in a web-aware Spring ApplicationContext. |

## 四 Spring Bean装配之自动装配

1. 自动装配类型：

|  |  |
| --- | --- |
| Mode | Explanation |
| NO | （Default）No autowiring. Bean references must be defined via a ref element. Changing the default setting is not recommended for larger deployments, because specifying collaborators explicitly gives greater control and clarity. To some extent, it documents the structure of a system. |
| byName | Autowiring by property name. Spring looks for a bean with the same name as the property that needs to be autowired. For example, if a bean definition is set to autowire by name, and it contains a master property(that is, it has a setMaster(…) method),Spring looks for a bean definition named master, and uses it to set the property |
| byType | Allows a property to be autowired if exactly one bean of the property type exists in the container. If more than one exists, a fatal exception is thrown, which indicates that you may not uses byType autowiring for that bean. If there are no matching beans, nothing happens; the property is not set. |
| Constructor | Analogous to byType, but applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised. |

1. 类型说明：
   1. No：不做任何操作
   2. Byname：根据属性名称自动装配，此选项将检查容器并根据名字查找与属性完全一致的bean，并将其与属性自动装配
   3. byType：如果容器中存在一个与指定属性类型相同的bean，那么将与该属性自动装配，如果存在多个该类型的bean，那么抛出异常，并指出不能使用byType方式进行自动装配，如果没有找到相匹配的bean，则什么事都不发生。
   4. constructor：与byType类似，不同之处在于它应用于构造器函数，如果容器中没有找到与构造器参数类型一致的bean，那么抛出异常
2. 配置方法：

Xml中配置default-autowire=”byName”；

有如上代码，便可以不用设值注入；

在引用的Bean中声明被引用类的set方法和属性，被引用类不需要做任何操作。

1. Constructor：创建带参构造，参数类型为要装配的java Bean，与id无关，全看类型（依然需要在xml中配置）。

## 五 Spring Bean装配之Resources

1. Resources是针对于资源文件的统一接口
2. Resources分类
   1. UrlResource：URL对应的资源，根据一个URL地址即可构建
   2. ClassPathResource：获取类路径下的资源文件
   3. FileSystemResource：获取文件系统里面的资源
   4. ServletContextResource：ServletContext封装的资源，用于访问ServletContext环境下的资源
   5. InputStreamResource：针对于输入流封装的资源
   6. ByteArrayResource：针对于字节数组封装的资源
3. ResourceLoader：对Resource进行加载的类，所有application contexts实现ResourceLoader接口，因此所有的application contexts都可以用于获取资源实例

public interface ResourceLoader{

Resource getResource(String Loaction);

}

1. ResourceLoader参数分类：

|  |  |  |
| --- | --- | --- |
| Prefix | Example | Explanation |
| Classpath: | Classpath:com/myapp/config.xml | Loaded from the classpath |
| File: | File:/data/config.xml | Loaded as a URL, from the filesystem. |
| http: | <http://myserver/logo.png> | Loaded as a URL |
| (none) | /data/config.xml | Depends on the underlying ApplicationContext. |

1. 示例：
   1. Resource template = ctx.getResource(“some/resource/path/myTemplate.txt”);(默认路径，依赖于ApplicationContext)
   2. Resource template =

ctx.getResource(“classpath:some.resource/path/myTemplate.txt”);（本节2.b）

Resource template =

* 1. ctx.getResource(“file:/some/resource/path/myTemplate.txt”); （本节2.c）

1. ApplicationContexts通过Aware接口（ApplicationContextAware）（Bean装配）获取
   1. **（Classpath）**

Resource resource = applicationContext.getResource(“classpath:config.txt”);

注：上面一行的方法有可能抛出IO异常，所以要throws IOException把异常抛出来做异常处理

Q：为什么直接使用classpath:config.txt（放在Resource目录下）就可以找到文件

A：因为Resource目录已经被配置在classpath目录下，也就是java build path的

source选项卡下

System.out.println(resource.getFilename());

System.out.println(resource.getcontextlength());

* 1. **（File）**

Resource resource = applicationContext.

getResource(“file:D:\\abc\\def\\config.txt”);

* 1. **（Url）**

Resource resource = applicationContext.

getResource(“url: <https://www.imooc.com/video/3758>”);

* 1. **（None）**

Resource resource = applicationContext.

getResource(“config.txt”);

注：默认路径情况下，依赖于ApplicationContext，而单元测试基类中的ApplicationContext的获取是依赖于classpath方式获取的xml文件，所以默认路径实际上是利用前缀为classpath的方式获取文件的

# 第五章 Spring Bean装配（下）

## 一 Spring Bean装配之Bean的定义及作用域的注解实现

1. Bean管理的注解实现及例子
2. Classpath扫描与组件管理