IoC 容器

# 介绍 IoC 容器和 Beans

基本概念

控制反转

Inversion of Control，缩写：IoC。

a.上层模块不应该依赖于下层模块，它们共同依赖于一个抽象

b.抽象不能依赖于具体实现，具体实现依赖于抽象

注：又称为依赖倒置原则。这是设计模式六大原则之一。

依赖注入

Dependency Injection, 缩写：DI。

依赖注入是 IoC 的最常见形式。容器全权负责的组件的装配，它会把符合依赖关系的对象通过 JavaBean属性或者构造函数传递给需要的对象。

Bean

JavaBean 是一种JAVA语言写成的可重用组件。为写成JavaBean，类必须是具体的和公共的，并且具有无参数的构造器。JavaBean 对外部通过提供getter/setter方法来访问其成员。

org.springframework.beans 和 org.springframework.context 是 IoC 容器的基础。

在 Spring 中，有两种 IoC 容器：BeanFactory 和 ApplicationContext。

BeanFactory：Spring 实例化、配置和管理对象的最基本接口。

ApplicationContext：BeanFactory 的子接口。它还扩展了其他一些接口，以支持更丰富的功能，如：国际化、访问资源、事件机制、更方便的支持 AOP、在web应用中指定应用层上下文等。

# 容器概述

实际开发中，更推荐使用 ApplicationContext 作为 IoC 容器，因为它的功能远多于 FactoryBean。

使用 IoC 容器可分为三步骤：

定义容器——需要配置一些元数据来告诉Spring，你希望容器如何工作，具体来说，就是如何去初始化、配置、管理 JavaBean 对象。

初始化容器——Spring根据配置的元数据来进行初始化工作。

使用容器

## 配置元数据

**元数据（Metadata）**：又称中介数据、中继数据，为描述数据的数据（data about data），主要是描述数据属性（property）的信息。

**配置元数据的方式：**

基于 xml 配置：Spring 的传统配置方式。在<beans></beans>标签中配置元数据内容。缺点是当 JavaBean 过多时，产生的配置文件足以让你眼花缭乱。

基于注解配置：Spring2.5 引入。可以大大简化你的配置。

基于 Java 配置：可以使用 Java 类来定义 JavaBean 。为了使用这个新特性，需要用到 @Configuration, @Bean, @Import 和 @DependsOn 注解。

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"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd"**>  
 <**bean id="..." class="..."**>  
 *<!-- collaborators and configuration for this bean go here -->* </**bean**>  
 <**bean id="..." class="..."**>  
 *<!-- collaborators and configuration for this bean go here -->* </**bean**>  
 *<!-- more bean definitions go here -->*</**beans**>

<beans>标签：Spring 配置文件的根节点。

<bean>标签：用来定义一个 JavaBean。Id属性是它的标识，在文件中必须唯一；class属性是它关联的类。

## 实例化容器

实例化容器的过程：

定位资源（XML配置文件）

读取配置信息(Resource)

转化为Spring 可识别的数据形式（BeanDefinition）

ApplicationContext context =  
 **new** ClassPathXmlApplicationContext(**new** String[] {**"services.xml"**, **"daos.xml"**});

组合xml 配置文件

配置的 Bean 功能各不相同，都放在一个xml文件中，不便管理。Java 设计模式讲究职责单一原则。配置其实也是如此，功能不同的JavaBean应该被组织在不同的xml文件中。然后使用import标签把它们统一导入。

<import resource="classpath:spring/applicationContext.xml"/>

<import resource="/WEB-INF/spring/service.xml"/>

## 使用容器

使用容器的方式就是通过getBean获取IoC容器中的JavaBean。Spring也有其他方法去获得JavaBean，但是Spring并不推荐其他方式。

*// create and configure beans*

ApplicationContext context =

**new** ClassPathXmlApplicationContext(**new** String[] {***"services.xml"***, ***"daos.xml"***});

*// retrieve configured instance*

PetStoreService service = context.getBean(***"petStore"***, PetStoreService.**class**);

*// use configured instance*

List<String> userList = service.getUsernameList();

# Bean概览

一个Spring容器管理一个或多个bean。

这些bean根据你配置的元数据（比如xml形式）来创建。

Spring IoC容器本身，并不能识别你配置的元数据。为此，要将这些配置信息转为Spring能识别的格式——BeanDefinition对象。

## 命名 Bean

**指定id和name属性不是必须的**

Spring中，并非一定要指定id和name属性。实际上，Spring会自动为其分配一个特殊名。

如果你需要引用声明的bean，这时你才需要一个标识。官方推荐驼峰命名法来命名。

**支持别名**

可能存在这样的场景，不同系统中对于同一bean的命名方式不一样。为了适配，Spring 支持为bean添加别名的功能。

**<alias name**=**"subsystemA-dataSource" alias**=**"subsystemB-dataSource"/>**

**<alias name**=**"subsystemA-dataSource" alias**=**"myApp-dataSource" />**

## 实例化Bean

**构造器方式**

**<bean id**=**"exampleBean" class**=**"examples.ExampleBean"/>**

**静态工厂方法**

**<bean id**=**"clientService"**

**class**=**"examples.ClientService"**

**factory-method**=**"createInstance"/>**

**public class** ClientService {

**private static** ClientService clientService = **new** ClientService();

**private** ClientService() {}

**public static** ClientService createInstance() {

**return** clientService;

}

}

**使用实例工厂实例一个新的bean**

**<bean id**=**"serviceLocator" class**=**"examples.DefaultServiceLocator">**

*<!-- inject any dependencies required by this locator bean -->*

**</bean>**

**<bean id**=**"clientService"**

**factory-bean**=**"serviceLocator"**

**factory-method**=**"createClientServiceInstance"/>**

**<bean id**=**"accountService"**

**factory-bean**=**"serviceLocator"**

**factory-method**=**"createAccountServiceInstance"/>**

**public class** DefaultServiceLocator {

**private static** ClientService clientService = **new** ClientServiceImpl();

**private static** AccountService accountService = **new** AccountServiceImpl();

**private** DefaultServiceLocator() {}

**public** ClientService createClientServiceInstance() {

**return** clientService;

}

**public** AccountService createAccountServiceInstance() {

**return** accountService;

}

}

# 依赖

依赖注入

依赖注入有两种主要方式：

构造器注入

Setter注入

构造器注入有可能出现循环注入的错误。如：

class A {

public A(B b){}

}

class B {

public B(A a){}

}

依赖和配置细节

使用 depends-on

Lazy-initialized Bean

自动装配

方法注入

Bean 范围

定制一个Bean的种类

Bean definition

容器扩展点

Classpath扫描和管理组件

使用 JSR330标准注解

# 基于注解的配置

Spring2.5 引入了注解。

**使用注解配置 JavaBean 是不是优于 xml？**

未必。正所谓，仁者见仁智者见智。任何事物都有其优缺点，看你如何取舍。来看看注解的优缺点：

优点：大大减少了配置，并且可以使配置更加精细——类，方法，字段都可以用注解去标记。

缺点：使用注解，不可避免产生了侵入式编程，也产生了一些问题。

你需要将注解加入你的源码并编译它；

注解往往比较分散，不易管控。

**注：spring 中，先进行注解注入，然后才是xml注入，因此如果注入的目标相同，后者会覆盖前者。**

## 使用注解

Spring 默认是不启用注解的。如果想使用注解，需要先在xml中启动注解。

启动方式：在xml中加入一个标签，很简单吧。

<context:annotation-config/>

***注：<context:annotation-config/> 只会检索定义它的上下文。什么意思呢？就是说，如果你***

***为DispatcherServlet指定了一个WebApplicationContext，那么它只在controller中查找@Autowired注解，而不会检查其它的路径。***

## 常用注解

基于 Java 的容器配置

环境抽象

注册一个 LoadTimerWeaver

ApplicationContext的补充

BeanFactory

装配 Bean

声明 Bean

注入 Bean