**Spring入门篇**

**2018年5月8日**

[第一章 Spring入门 3](#_Toc514349379)

[第二章 概述 3](#_Toc514349380)

[一 Spring入门课程简介 3](#_Toc514349381)

[二 Spring概况 3](#_Toc514349382)

[三 Spring Framework Runtime 4](#_Toc514349383)

[四 Spring的作用 4](#_Toc514349384)

[五 Spring适用范围 4](#_Toc514349385)

[六 Spring框架 5](#_Toc514349386)

[第三章 Spring IOC容器 5](#_Toc514349387)

[一 IOC及Bean容器 5](#_Toc514349388)

[二 Spring的常用注入方式 7](#_Toc514349389)

[第四章 Spring Bean装配（上） 9](#_Toc514349390)

[一 Spring Bean装配之Bean的配置项及其作用域 9](#_Toc514349391)

[二 Spring Bean装配之Bean的生命周期 10](#_Toc514349392)

[三 Spring Bean装配之Aware接口 11](#_Toc514349393)

[四 Spring Bean装配之自动装配 13](#_Toc514349394)

[五 Spring Bean装配之Resources 14](#_Toc514349395)

[第五章 Spring Bean装配（下） 15](#_Toc514349396)

[一 Spring Bean装配之Bean的定义及作用域的注解实现 15](#_Toc514349397)

[二 Spring Bean装配之Autowired注解说明—1 20](#_Toc514349398)

[三 Spring Bean装配之Autowired注解说明—2 21](#_Toc514349399)

[四 Spring Bean装配之Autowired注解说明—3 22](#_Toc514349400)

[五 Spring Bean装配之基于Java容器的注解说明—@Bean 25](#_Toc514349401)

[六 Spring Bean装配之基于Java容器的注解说明—@ImportResource和@Value 26](#_Toc514349402)

[七 Spring Bean装配之基于Java容器的注解说明—@Bean和@Scope 27](#_Toc514349403)

[八 Spring Bean装配之基于Java容器的注解说明—基于泛型的自动装配 28](#_Toc514349404)

[九 Spring Bean装配之Spring对JSR支持的说明 30](#_Toc514349405)

[第六章 Spring AOP基本概念 32](#_Toc514349406)

[一 AOP基本概念及特点 32](#_Toc514349407)

# 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

## 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）



1. Bean的初始化
   1. 基础：两个包
2. org.springframework.beans
3. org.springframework.context
4. BeanFactory提供配置结构和基本功能，加载并初始化Bean
5. ApplicationContext保存了Bean对象并在spring中被广泛应用
   1. 方式：Application
6. 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  **作用如下网址：**

**（https://www.cnblogs.com/bincoding/p/5725732.html）**

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
   1. 从Spring3.0开始，Spring JavaConfig项目提供了很多特性，包括使用java而不是XML定义的Bean（即通过java注解的方式定义Bean，java注解是从jdk5引入的），比如@Configuration，@Bean，@Improt，@DependsOn
   2. @Component是一个通用注解，可用于任何Bean
   3. @Repository，@Service，@Controller是更有针对性的注解（也可以注解Bean，但一般不用，而且都是Component的子注解，也就是基于Component定义的注解）
      1. —@ Repository通常用于注解DAO类，即持久层
      2. —@Service通常用于注解Service类，即服务层
      3. —@Controller通常用于注解Controller类，即控制层（MVC）
3. 元注解（Meta-annotations）：
   1. 许多Spring提供的注解可以作为自己的代码，即”元数据注解”，元注解是一个简单的注解，可以应用到另外一个注解
   2. 除了value()，元注解还可以有其他的属性，允许定制
   3. **Jdk5如何自定义注解？**

**https://blog.csdn.net/zymx14/article/details/55837309**

1. 类的自动检测及Bean的注册

Spring可以自动检测类并注册Bean到ApplicationContext（上下文，IOC容器）中:

例子：@Service（加到类上面）

public class SimpleMovieLister{

private MovieFinder movieFinder;

@Autowired

public SimpleMovieLister(MovieFinder movieFinder){

this.movieFinder = movieFinder;

}

}

若注解在Bean上，Spring会自动检测到，并注册到ApplicationContext中

@Repository

Public class JpaMovieFinder implements MovieFinder{

//implementation elided for clarity

}

1. 配置文件：<context：annotation-config>
   1. 通过在基于XML的Spring配置如下标签（请注意包含上下文命名空间
   2. <context：annotation-config>仅仅会查找在同一个ApplicationContext中的Bean注解
   3. <?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:annotation-config>

<!--开启注解处理器-->（完成了Bean的注册之后，去处理Bean中的基于方法或者是成员变量的注解）

</context:annotation-config>

或者

<context:annotation-config/>

</beans>

1. 类的自动检测及Bean的注册

为了能够检测这些类并注册相应的Bean，需要下面内容：

<?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="com.persia">

<!-- 开启组件扫描(component-scan) -->

<!--指定包下的所有类进行扫描-->

</context:component-scan>

或者

<context:component-scan base-package="com.persia"/>

</beans>

注：

* + - 1. <context:component-scan>包含<context:annotation-config>，通常在前者使用后，不再使用后者
      2. AutowiredAnnotationBeanPostProcessor和

CommonAnnotationBeanPostProcessor也会被包含进来

* + - 1. 当 Spring 容器启动时，AutowiredAnnotationBeanPostProcessor 将扫描 Spring 容器中所有 Bean，当发现 Bean 中拥有@Autowired 注释时就找到和其匹配（默认按类型匹配）的 Bean，并注入到对应的地方中去。

1. 使用过滤器进行自定义描述
2. 默认情况下，类被自动发现并注册bean的条件是：使用@Component，@Repository，@Service，@Controller注解或者使用@Component的自定义注解
3. 可以通过过滤器修改上面的行为，如：下面例子的XML配置忽略所有的@ Repository注解并用”Stub”代替

<bean>

<context:component-scan base-package=”org.example”>

（包含过滤器，以通配符的形式找下面的这两种类）

<context:include-filter type=”regex”

expression=”.\*Stub.\* Repository”/>

（排除过滤器，排除的类型为注解，排除所有用Repository注

解的类）

<context:enclude-filter type=”annotation”

expression=”org.springframework.stereotype. Repository”/>

</context:component-scan>

</beans>

1. 还可使用use-default-filters=”false”禁用自动发现与注册
2. 使用过滤器的类型：

Using filters to customize scanning（使用过滤器定制扫描）

|  |  |  |
| --- | --- | --- |
| Filter Type | Example Expression | Description |
| Annotation | Org.example.SomeAnnotation | An annotation to be present at the type level in target components |
| Assignable | Org.example.SomeClass | A class(or interface) that the target components are assignable to (extend/implement) |
| aspectj | Org.example..\*Service+ | An Aspectj type expression to be matched by the target components |
| Regex | Org\.example\.Default.\* | A regex expression to be matched by the target components class names |
| Custom | Org.example.MyTypeFilter | A custom implementation of the org.springframework.core.type.TypeFilter interface |

1. 定义Bean

如何使用注解定义一个Bean，并使这个Bean在系统或者应用中能够被使用

* 1. 扫描过程中组件被自动检测，那么Bean名称是由BeanNameGenerator生成的（@Conponent，@Repository，@Service，@Controller都会有一个name属性用于显式设置Bean Name）

例子：

**@Service(“MyMovieLister”)（显式指定）**

Public class SimpleMovieLister{……}

注：用service进行注解，service有一个属性是name，用service注解这

个类的时候，可以去显式的指定这个类在注册到Bean容器或者IOC容器中它所对应的ID，也就是名称，相当于XML文件中的ID，如果显式指定了，就会使用显式指定的名称

**@Repository（未显示指定）**

Public class MovieFinderImpl implements MovieFinder{……}

注：此时会根据BeanNameGenerator来自动生成该类在容器中的ID，通

常的生成规则是“以类名为基础，并把类名的首字母小写，作为Bean的ID”

* 1. 可自定义Bean命名策略，实现BeanNameGenerator接口，并一定要包含一个无参数构造器

如何使用：

<beans>

<context:component-scan base-package=”org.example”

name-generator=”org.example.MyNameGenerator”/>

</beans>

1. 作用域（Scope）
   1. 通常情况下自动查找的Spring组件，其scope是singleton,Spring2.5提供了一个标识scope的注解@scope

@Scope(“prototype”)（标识类的作用域）

@Repository

Public class MovieFinderImpl implements MovieFinder{……}

注：

若scope注解不配置作用域，其默认值依然为singleton

* 1. 也可以自定义scope策略，实现scopeMetadataResolver接口并提供一个无参构造器

<beans>

<context:component-scan base-package=”org.example”

Scope-resolver=”org.example.MyScopeResolver”/>

</beans>

比如想要让这个Bean针对每一个线程有效，此时可以使用自定义来定义一个作用域，使其只针对每一个线程有效，或其他方面也可以使用

1. 代理方式：

可以使用scoped-proxy属性指定代理，有三个值可选：no,interfaces(接口),targetclass(目标类)

<beans>

<context:component-scan base-package=”org.example”

Scoped-proxy=”interfaces”/>

</beans>

1. 使用什么标签，应该使用什么样的namespace的声明或者什么样的schemaLocation等，可以参考Spring的官方文档

## 二 Spring Bean装配之Autowired注解说明—1

1. @Required
   1. @Required注解适用于Bean属性的setter方法
   2. 这个注解仅仅表示，受影响的bean属性必须在配置时被填充，通过在bean定义或通过自动装配一个明确的属性值

示例：

public class SimpleMovieLister{

private MovieFinder movieFinder;

@Required

public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder = movieFinder;

}

}

1. @Autowired（字面意思就是：自动装配）
   1. 可以将@ Autowired注解为”传统”的setter方法

Private MovieFinder movieFinder;

@ Autowired

Public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder = movieFinder;

}

* 1. 可用于构造器或者成员变量

@Autowired

private MovieCatalog movieCatalog;

private CustomerPreferenceDao customerPreferenceDao;

@Autowired（带参构造）

public MovieRecommender(CustomerPreferenceDao customerPreferenceDao){

this.customerPreferenceDao = customerPreferenceDao

}

* 1. 默认情况下，如果因找不到合适的bean将会导致autowired失败抛出异常，可以通过下面的方式避免

Public clsaa SimpleMovieLister{

Private MovieFinder movieFinder;

@Autowired(required=false)

Public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder = movieFinder;

}

}

* 1. 每个类只能有一个构造器被标记为required=true
  2. @Autowired的必要属性，建议使用@Required注解

注：DAO：data access object

## 三 Spring Bean装配之Autowired注解说明—2

1. @Autowired
   1. 可以使用@Autowired注解那些众所周知的解析依赖性接口，比如：BeanFactory，

ApplicationContext，Environment，ResourceLoader，ApplicationEventPublisher，MessageSource

示例：

public class MovieRecommender{

@Autowired

public ApplicationContext context;

public MovieRecommender(){

}

}

注：在当前类中声明对ApplicationContext的应用，然后使用Autowired的

注解，此时就可以在当前类中得到并使用IOC容器的上下文信息

* 1. 可以通过添加注解给需要该类型的数组的字段或方法（所谓数组指的是set或者list），以提供ApplicationContext中的所有特定类型的Bean

示例：

private Set<MovieCatalog> movieCatalogs;

@Autowired

public void setMovieCatalogs(Set<MovieCatalog> movieCatalogs){

this. movieCatalogs = movieCatalogs;

}

注：当set中声明的这种类型的时候，当前的ApplicationContext中的所有

事set泛型中声明类型的这种Bean或者是他的子类，都可以被@Autowired的注解把这些Bean的实例放到当前的集合当中去（Set）

* 1. 可以用于装配Key为String的Map

示例：

private Map<String,MovieCatalog> movieCatalogs;

@Autowired

public void setMovieCatalogs(Map<String,MovieCatalog> movieCatalogs){

this. movieCatalogs = movieCatalogs;

}

注：key为所有Bean的ID，value为所有Bean的对象（而Bean的类型应

该为Map中的value的类型）

* 1. 如果希望数组有序，[可以让Bean实现org.springframework.core.Ordered接口或使用的@Order](mailto:可以让Bean实现org.springframework.core.Ordered接口或使用的@Order)注解
  2. @Autowired是由Spring BeanPostProcessor处理的，所以不能在自己的BeanPostProcessor或BeanFactoryPostProcessor类型应用这些注解，这些类型必需通过XML或者Spring的@Bean注解加载
  3. 增强For：（集合中的类型 变量：集合名称）
  4. 验证@Autowired注解list集合（set同理）（集合中是Bean的示例（对象））

@Autowired

private List<BeanInterface> list;（示例中BeanInterface有两个实现类）

public void say(){

for(BeanInterface bean : list){

System.out.println(bean.getClass().getName());

}

}

注：遍历集合之前需要加上null != list && 0 != list.size()条件

* 1. 验证@Autowired注解Map集合（集合中是Bean的ID和实例（对象））

@Autowired

private Map<String,BeanInterface> map;

public void say(){

for(Map.Entry<String,BeanInterface> entry : map.entrySet()){

System.out.println(entry.getKey()+"

"+entry.getValue().getClass().getName());

}

}

* 1. 希望数组有序，具体代码示例：

@Order(value = 1)或者@Order(1)，这个排序只针对list或set这类的集合有用，对map这类的集合无效，需要排序的Bean中加上如上注解，数字代表顺序，即可完成排序

## 四 Spring Bean装配之Autowired注解说明—3

1. @Qualifier
   1. 按类型自动装配可能多个bean实例的情况，可以使用Spring的@ Qualifier注解缩小范围或指定唯一，也可以用于指定单独的构造器参数或者方法参数

示例1：

public class MovieRecommender(){

@Autowired

@Qualifier("main")（指定该类的实现类或者子类中ID为main的类）

private MovieCatalog movieCatalog;

}

示例2：

public class MovieRecommender(){

private MovieCatalog movieCatalog;

private CustomerPerferenceDao customerPerferenceDao;

@Autowired（用在方法中指定ID名为main的MovieCatalog类，不会指定

CustomerPerferenceDao类，加在谁前面指定谁）

public void prepare(Qualifier("main")MovieCatalog movieCatalog,

CustomerPerferenceDao customerPerferenceDao){

this.movieCatalog = movieCatalog;

this.customerPerferenceDao = customerPerferenceDao;}}

* 1. 可用于注解集合类型变量（原理同上）
  2. 在xml配置文件中实现@Qualifier

<?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:annotation-config/>

<bean class="example.SimpleMovieCatalog">

<qualifier value="main"/>

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

</bean>

<bean class="example.SimpleMovieCatalog">

<qualifier value="action"/>

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

</bean>

<bean id = "movieRecommender"

class = "example.MovieRecommender">

</bean>

<beans>

* 1. 如果通过名字进行注解注入，主要使用的不是@Autowired（即使在技术上能够通过@Qualifier指定Bean的名字），替代方式是使用JSR-250@Resource注解，它是通过其独特的名称来定义来识别特定的目标（这是一个与所声明的类型是无关的匹配过程）（JSR-250是一种标准）
  2. 因语义差异，集合或Map类型的Bean无法通过@Autowired来注入，因为没有类型匹配到这样的Bean，为这些Bean使用@Resource注解，通过唯一名称引用集合或Map的bean**（可以理解为无法将一个集合或者map作为一个Bean，注入到另一个集合或者Map中，因为若集合类型为list，则无法匹配，因为list很多，应该是这样）**
  3. Qualifier和Resource的区别
     1. @Autowired注解适用于fields（字段）、constructors（构造器）、

multi-argument method（多参数方法）这些允许在参数级别使用@Qualifier注解缩小范围的情况

* + 1. @Resource适用于成员变量、只有一个参数的setter方法，所以在目标是构造器或者一个多参数方法时，最好的方式是使用qualifiers
  1. 定义自己的qualifier注解并使用

示例一：

@Target((ElementType.FIELD, ElementType.PARAMETER))

@Retention(RetentionPolicy.RUNTIME)

**@Qualifier**

public @interface Genre{

String value();

}

public class MovieRecommender{

@Autowired

@Genre("Action")

private MovieCatalog actionCatalog;

private MovieCatalog comedyCatalog;

@Autowired

public void setComedyCatalog(@Genre("Comedy")

MovieCatalog comedyCatalog){

this.comedyCatalog = comedyCatalog;

}

}

示例二：

<?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:annotation-config/>

<bean class="example.SimpleMovieCatalog">

<qualifier type="Genre" value="Action"/>

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

</bean>

<bean class="example.SimpleMovieCatalog">

<qualifier type="example.Genre" value="Comedy"/>

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

</bean>

<bean id = "movieRecommender"

class = "example.MovieRecommender">

</bean>

<beans>

## 五 Spring Bean装配之基于Java容器的注解说明—@Bean

1. @Bean标识一个用于配置和初始化一个由SpringIOC容器管理的新对象的方法（即通过@Bean可以生成一个IOC容器的Bean实例），类似于XML配置文件的<bean/>
2. 可以在Spring的@Component注解的类中使用@Bean注解任何方法（仅仅是可以），然后在方法里面去创建对象，进行返回，这就是@Bean的一种用法
3. 通常和@Bean一起使用的不是@Component，而是@Configuration（配置）

示例：

**@Configuration**

**public class AppConfig{**

**@Bean**

**public MyService myService(){**

**return new MyServiceImpl();**

**}**

**}**

注：在AppConfig上使用@Configuration这个注解，那么相当于声明Appconfig，这个类相当于一个配置文件

1. 在上面的方法注解的效果相当于在下面的在XML中的代码

**<beans>**

**<bean id = "myService Class = "com.acme.services.MyServiceImpl"/>**

**</beans>**

1. @Bean
   1. 自定义Bean name

@Configuration

public class AppConfig{

@Bean(name = “myFoo”)(指定Bean的ID，相当于XML中配置的Bean的ID)

（可以通过这种方式来自定义Bean的Name）

public Foo foo(){

return new Foo();

}

}

* 1. init-method：在初始化Foo的时候会执行Foo这个类的init方法

public class Foo{

public void init(){

//initialization logic

}

}

@Configuration

public class AppConfig{

@Bean(initMethod = "init")

public Foo foo(){

return new Foo();}

}

* 1. destroy-method：在销毁Bar的时候会执行Bar这个类的cleanup方法

public class Bar{

public void cleanup(){

//destruction logic

}

}

@Configuration

public class AppConfig{

@Bean(destroyMethod="cleanup")

public Bar bar(){

return new Bar();

}

}

**注：初始化和销毁的方法都是写在return之后new的那个类里面**

## 六 Spring Bean装配之基于Java容器的注解说明—@ImportResource和@Value

1. 本节课程将带你学习如何使用@ImportResource和@Value注解进行资源文件读取
2. XML方式加载资源文件：

<beans>

<!--enable processing of annotations such as @Autowired and @Configuration-->

<context:annotation-config/>

**<context:property-placeholder （其作用是加载资源文件）**

**location="classpath:/com/acme/jdbc.properties"/>**

<bean class="com.acme.AppConfig"/>

<bean class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="url" value="$(jdbc.url)"/>

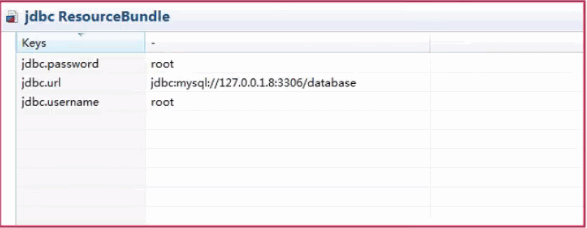
<property name="username" value="$(jdbc.username)"/>

<property name="password" value="$(jdbc.password)"/>

</bean>**（使用某一种数据源，并指定数据源的相关参数）**

</beans>

注：在使用context:property-placeholder时，它会对应一个资源文件，location代表

****资源文件的存放位置

Properties文件是以键值对的形式存在的

注：当properties文件被加载之后，可以通过$(properties中的key)这种形式来引

用properties资源文件中的内容

1. 使用注解加载资源文件：

@Configuration

@ImportResource("classpath:/com/acme/properties-config.xml")

**（这个XML文件中的内容与第二点中的XML文件中的内容类似，只不过文件**

**中只是指定了一下资源文件的路径，并不会用property给相关参数赋值，即**

**上方第二点红色字体）**

public class AppConfig{

@Value("$(jdbc.url)")

private String url;

@Value("$(jdbc.username)")

**这里如果直接使用username，会发生错误，因为username取得是登录**

**当前操作系统的用户名，而不是资源文件中key对应的value，所以使用时最好加上可以明确分辨的前缀**

private String username;

@Value("$(jdbc.password)")

private String password;

@Bean

public DataSource datasource(){

return new DriverManagerDataSource(url, username, password);

}

}

## 七 Spring Bean装配之基于Java容器的注解说明—@Bean和@Scope

1. @Bean and @Scope
   1. 默认@Bean是单例的，Bean的作用域包括：singleton、prototype、request、session、global session。

@Configuration

public class MyConfiguration{

@Bean

@Scope("prototype")（意义为每次请求都会创建一个新的对象）

public Encryptor encryptor(){

//...

}

}

**注：对于getbean获取的对象，的getClass().hashCode()的值是一样的，因为是类的hashCode值，对象的hashCode值是不一样的**

@Bean

@Scope(value = "session",

proxyMode = ScopedProxyMode.TARGET\_CLASS)(proxyMode代理方式,详见5.1)

public UserPreferences userPreferences(){

return new UserPreferences();

}

@Bean

public Service userService(){

UserService service = new SimpleUserService();

//a reference to the proxied userPreferences bean

service .setUserPreferences(userPreferences());

return service;

}

**上面代码作用是通过代理类的userPreferences找到最终使用的userService**

**proxyMode的相关作用详见:http://www.bubuko.com/infodetail-1434289.html**

## 八 Spring Bean装配之基于Java容器的注解说明—基于泛型的自动装配

stringStore和integerStore都是要实现一个接口，并且实现接口时对于接口的泛型要进行明确指定（String、Integer）

1. 示例：

@Configuration

public class MyConfiguration{

@Bean

public StringStore stringStore(){

return new StringStore();

对于上面两个类实现的接口，在编写这个接口的时候，泛型不需要明确指出，写T即可

}

@Bean

public IntegerStore integerStore(){

return new IntegerStore();

对于示例1中的store即为上面所说的接口

}

}

**基于泛型装配示例1：**

@Autowired

private Store<String> s1; //<String> qualifier, injects the stringStore bean

@Autowired

private Store<Integer> s2; //<Integer> qualifier, injects the integerStore bean

**基于泛型装配示例2：**

//Inject all Store beans as long as they have an <Integer> generic

//Store<String> beans will not appear in this list

@Autowired

private List<Store<Integer>> s;

**注：基于泛型的自动装配是在Spring4中新增的一部分内容**

1. *API*(Application Programming Interface,应用程序编程接口)
2. SPI：针对做这种服务的框架的开发或者是一些工具的基础使用的特殊类型接口

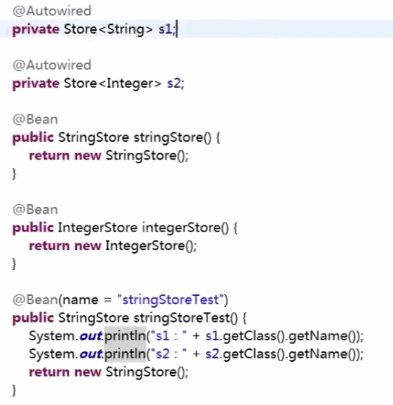
参考网址：

<https://blog.csdn.net/it_man/article/details/7578342>

<https://my.oschina.net/kipeng/blog/1789849>

1. 在执行自动装配的时候会把变量（S1）对应的泛型的类的对象赋值给变量，但若是有多余一个返回值类型与上边的类的类型一致的@Bean，则会装配失败

例如：



这样就会执行单元测试获取ID为stringStoreTest对象的时候，就会报错

把最后一个方法的StringStore类型改为Store类型即可，即不明确指定类型

1. Autowired扩展的内容（自定义qualifier注解）
   1. CustomAutowireConfigurer
      1. CustomAutowireConfigurer是BeanFactoryPostProcessor的子类，通过它可以注册自己的qualifier注解类型（即使没有使用Spring的@Qualifier注解）

<bean id="customAutowireConfigurer"

class="org.springframework.beans.factory.annotation.CustomAutowireConfigurer">

<property name="customQualifierTypes">（里面是具体的注解类型）

<set>（set里面可以放置多个）

<value>example.CustomQualifier</value>

</set>

</property>

</bean>

* + 1. 该AutowireCandidateResolver决定自动装配的候选者：（用来处理自定义的qualifier的具体类型）
       1. 每个Bean定义的autowired-candidate值
       2. 任何<bean/>中的default-autowired-candidates
       3. @Qualifier注解及使用CustomAutowireConfigurer的自定义类型

**注：三种方法任选其一即可**

## 九 Spring Bean装配之Spring对JSR支持的说明

1. @Resource
   1. Spring还支持使用JSR-250@Resource注解的成员变量或setter方法，这是一种在JavaEE5和6的通用模式（7也是支持的），Spring管理的对象也支持这种模式
   2. @Resource有一个name属性，并且默认Spring解释该值作为被注入Bean的名称

代码示例：

public class SimpleMovieLister{

private MovieFinder movieFinder;

**@Resource(name="myMovieFinder")**

public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder = movieFinder;

}

}

* 1. 如果没有显式的指定@Resource的name，默认的名称是从属性名或者是setter方法得出（看@Resource注解的位置是在成员变量上还是setter方法上）

代码示例：

public class SimpleMovieLister{

private MovieFinder movieFinder;

@Resource

public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder = movieFinder;

}

}

* 1. 注解提供的名字被解析为一个bean的名称，这是由ApplicationContext的中的CommonAnnotationBeanPostProcessor发现并处理的

1. @PostConstruct and @PreDestroy
   1. [CommonAnnotationBeanPostProcessor不仅能识别JSR-250中的生命周期注解@Resource，在Spring2.5](mailto:CommonAnnotationBeanPostProcessor不仅能识别JSR-250中的生命周期注解@Resource，在Spring2.5)中引入支持初始化回调和销毁回调，前提是CommonAnnotationBeanPostProcessor是Spring的ApplicationContext中注册的（即CommonAnnotationBeanPostProcessor类在IOC容器中注册了）

**注：相关类的注册详见如下网址：**

<https://www.cnblogs.com/iuranus/archive/2012/07/19/2599084.html>

代码示例：

public class CachingMovieLister{

@PostConstruct（初始化）

public void populateMovieCache(){

//populates the movie cache upon initialization...

}

@PreDestroy（销毁）

public void cleanMovieCache(){

//clears the movie cache upon destruction...

}

}

1. 使用JSR330标准注解
   1. 从Spring3.0开始支持JSR330标准注解（依赖注入注解），其扫描方式与Spring注解一致
   2. 使用JSR330需要依赖javax.inject包
   3. 使用Maven引入javax.inject包方式如下

代码示例：

<dependency>

<groupId>javax.inject</groupId>

<artifactId>javax.inject</artifactId>

<version>1</version>

</dependency>

1. @Inject（属于JSR330标准注解）
   1. @Inject等效于@Autowired（即都是用于自动装配的），可以使用于类、属性、方法、构造器

代码示例：

import javax.inject.Inject;

public class SimpleMovieLister{

private MovieFinder movieFinder;

@Inject

public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder = movieFinder;

}

}

1. @Named（属于JSR330标准注解）
   1. 如果想使用特定名称进行依赖注入，使用@Named

即：当同一种类型的Bean在IOC容器中有多个的时候（比如一个接口和两个实现类），如果想使用特定的Bean，那此时就可以使用@Named注解

示例代码1：（一种方式是注解在类上，下面的代码表示想使用ID为main的

MovieFinder对象）

import javax.inject.Inject;

import javax.inject.Named;

public class SimpleMovieLister{

private MovieFinder movieFinder;

@Inject（此时@Inject最好放到方法上，**放到成员变量上可能会报错**）

public void setMovieFinder

(@Named("main") MovieFinder movieFinder){

this.movieFinder = movieFinder;

}

}

* 1. @Named与@Component是等效的

示例代码2：（另一种方式是@Named是用来指定某一个名称的Bean，即这个

Bean的ID就是movieListener）

@Named("movieListener")

public class SimpleMovieLister{

private MovieFinder movieFinder;

@Inject（此时@Inject可以放成员变量上，也可以放到方法上）

public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder = movieFinder;

}

}

**注：@named这种使用方式与@Qualifier有点类似**

1. 使用配置项方式是比较老的Spring项目使用的，现在越来越多的项目都使用注解的方式了

# 第六章 Spring AOP基本概念

## 一 AOP基本概念及特点