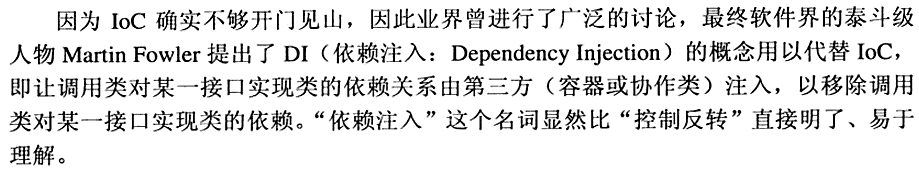
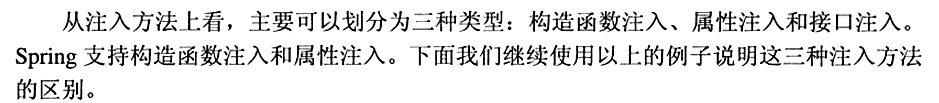
# Spring IOC

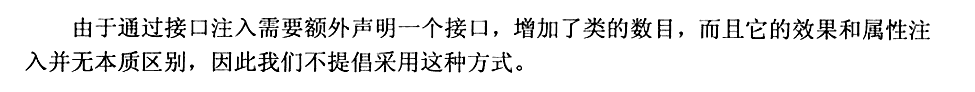
1. IOC容器概述
2. IOC概念

1.1、IOC和DI

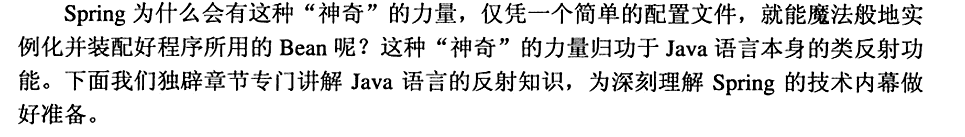


1.2、三种DI的方式



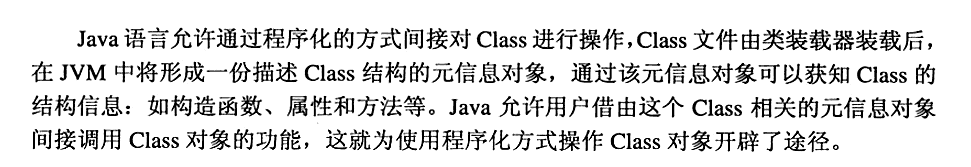


1.3、基于反射



1. Java反射

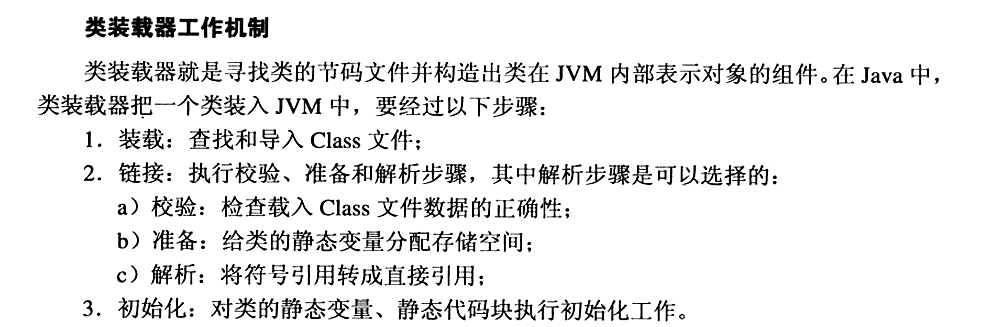
2.1、反射

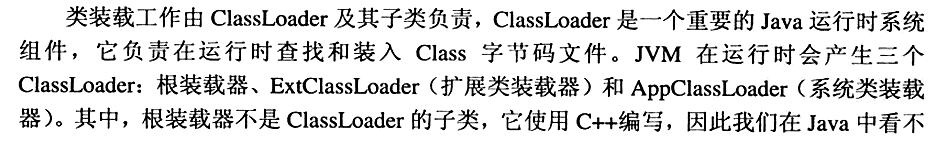


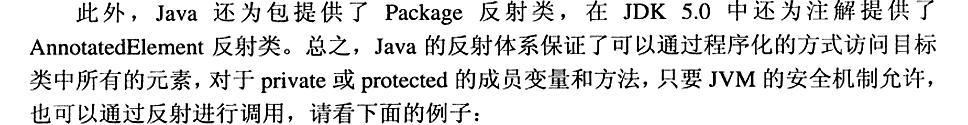
Eg：com.springioc.reflect.ReflectTest

2.2、类装载

器

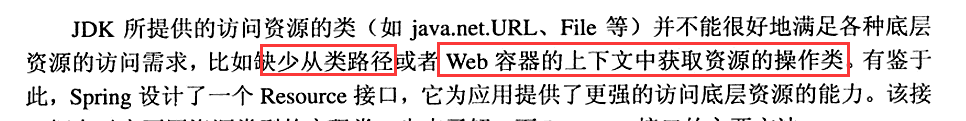


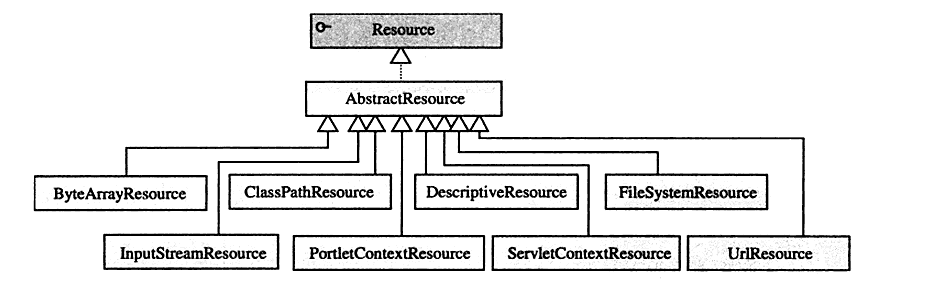


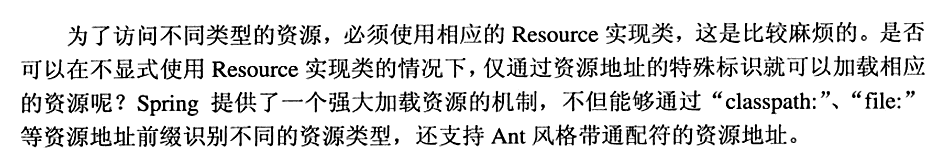


Eg:com.springioc.reflect.PrivateCarReflect

1. 关于资源访问：Resource

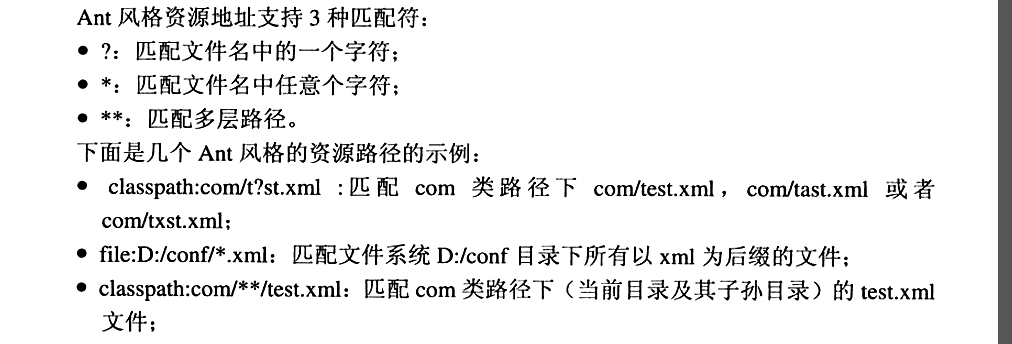


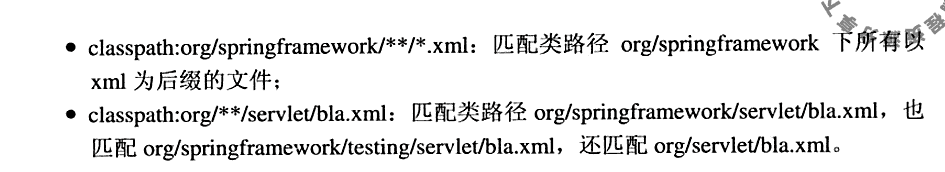


（1）关于通配访问



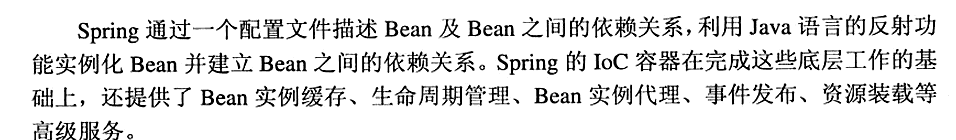
例如：

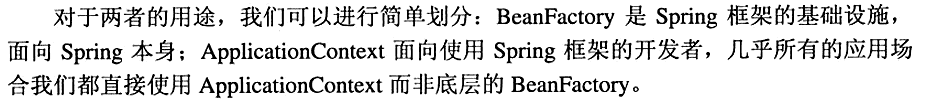




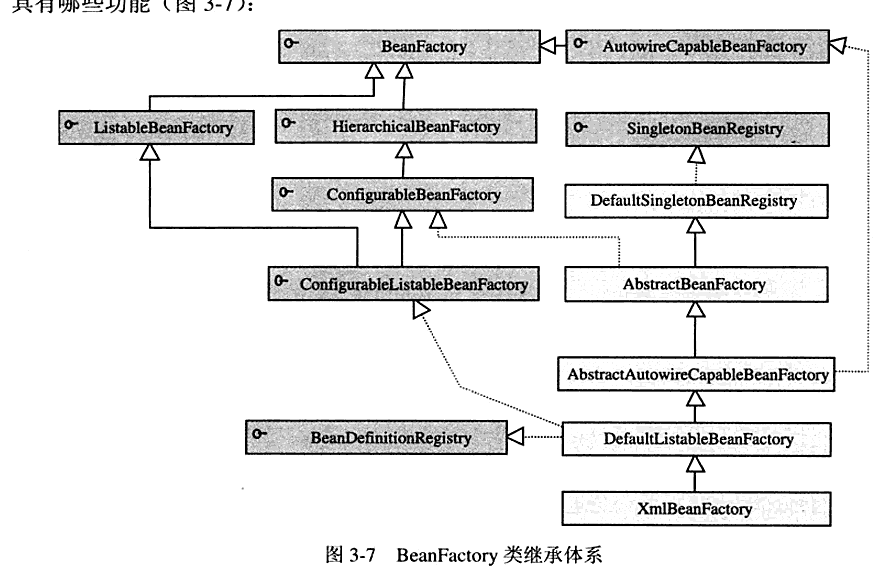
**Eg：org.springframework.core.io.Resource.PatternResolverTest.java**

1. 基础接口：BeanFactory、ApplicationContext、WebApplicationContext

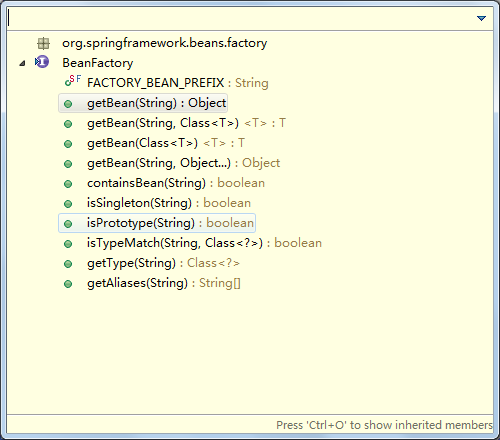




3.1、BeanFactory



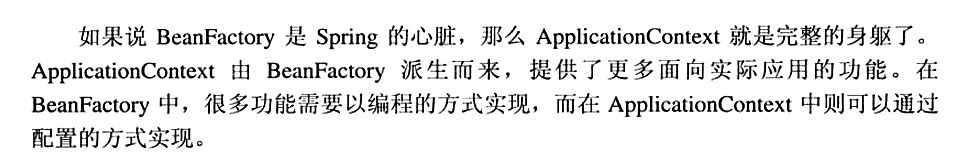
BeanFactory最主要的方法就是从容器中返回特定名称的Bean。



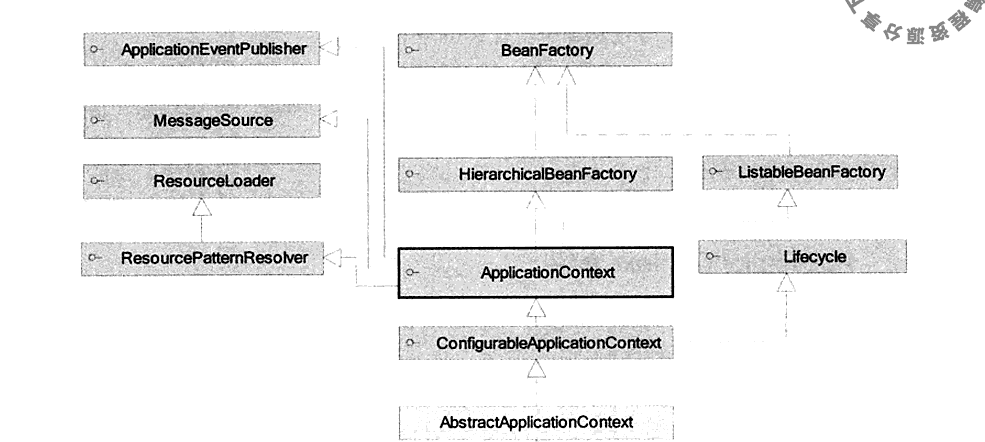
**Eg：从BeanFactory中获取Bean，**

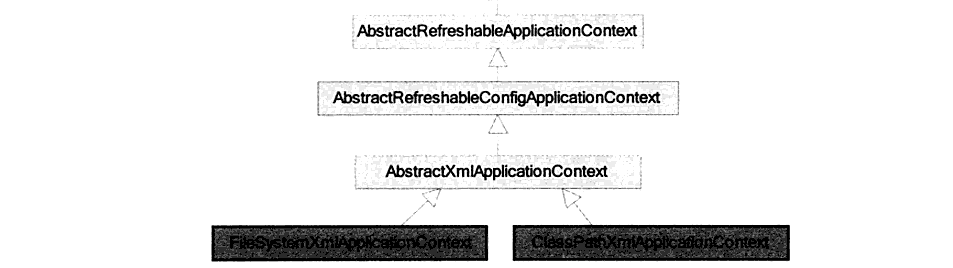
**com.springioc.beanfactory.BeanFactoryTest**

3.2、ApplicationContext

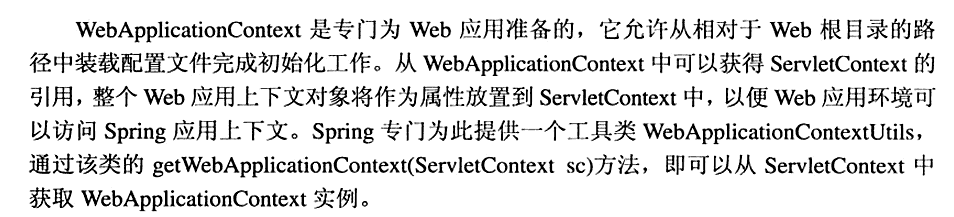


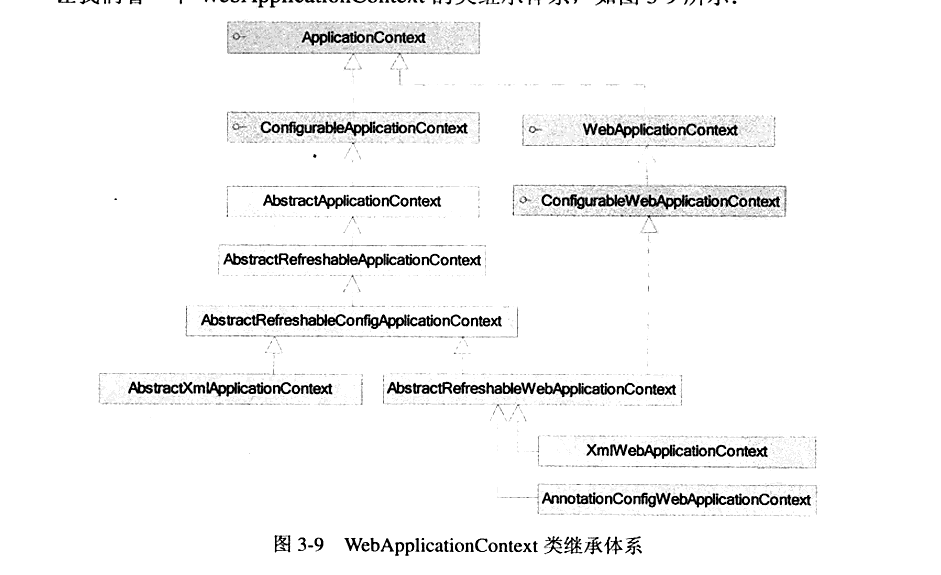
Applicaion 的类图

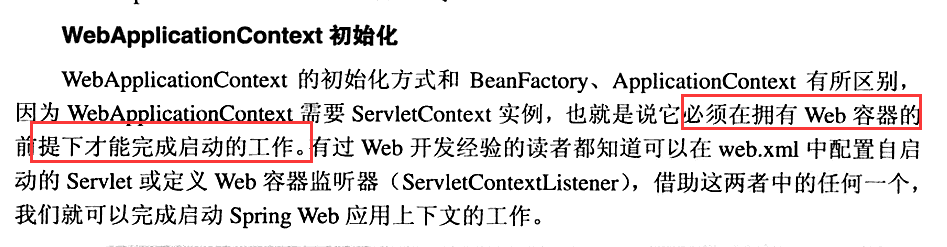




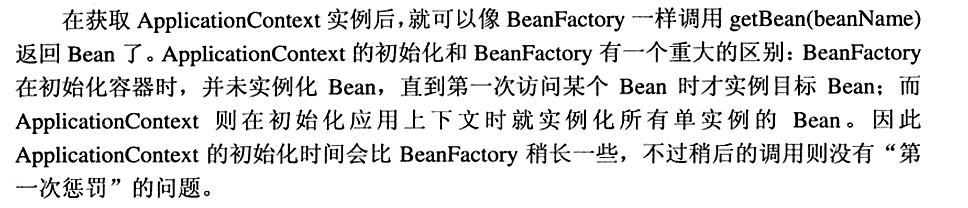
3.3、WebApplicationContext



9



备注：beanFactory和ApplicationContext的区别



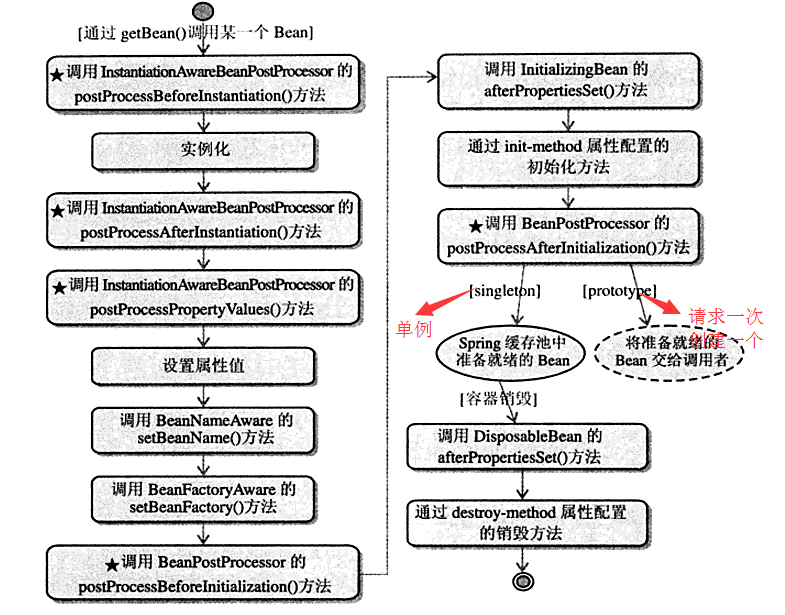
**Eg：从ApplicationContext中获取Bean**

**com.springioc.context.AnnotationApplicationContext**

1. Bean的生命周期

类似Servlet，Bean是有一个完整的生命周期，而且相对比较复杂：

4.1、BeanFactory中的Bean生命周期

Bean的生命周期，大致分为3大类：

第一类：Bean自身的方法，有init-method和destoryp-method所指定的方法。

第二类：Bean级生命周期接口的方法： BeanFactoryAware, BeanNameAware, InitializingBean,DisposableBean

第三类：容器级生命周期接口的方法：上图带有星号的都是容器级接口，一般称之为**后处理器**（InstantiationAwareBeanPostProcessor和BeanPostProcessor二个接口定义）。一般不需要Bean实现，独立于Bean。通过合理编程。可以对感兴趣的bean进行处理。

说明：Bean级别和容器级接口是共性和特性辩证统一思想的提现，前置解决Bean个性的问题，后者解决Bean共性化的问题。

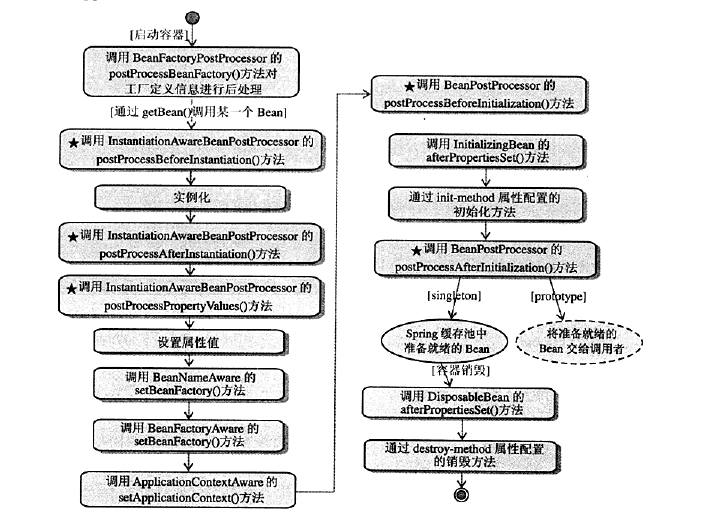
Eg：BeanFactory中Bean 的生命周期，以及后处理器的注入，**com.springioc.beanfactory.BeanLifeCycle**

说明：

除非编写一个基于Spring之上的插件或者是子项目，否则完全可以抛开4个Bean级别的接口类，因为它让我们的代码和Spring紧耦合。

但是容器级接口，可以完全像一个插件注册到Spring容器之中，为容器提供了额外的功能。一般而言也不需要，但是在Spring插件项目或是子项目中可以完成很多激动人心的功能。

4.2、ApplicationContext中的Bean生命周期



ApplicationContext会利用Java反射自动识别出配置的后处理器，并自动注册到上下文中。但是BeanFactory需要编程实现，所以我们普遍使用ApplicationContext而很少使用BeanFactory。

**Eg：使用XmlApplicationContext加载上下文：**

**com.springioc.context.XmlApplicationContextTest**

**使用FileSystemXmlApplicationContext加载上下文：**

**com.springioc.context.FileSystemXmlApplicationContextTest**

1. 总结

**Bean的生命周期，可以分成，Bean创建前后、Bean设置属性值前后、Bean初始化前后、Bean丢入容器或返回调用者、Bean销毁前后。总共12大类步骤！！！**

**通过三个级别，依次实现，分别是容器级别、Bean级别以及Bean 本身！！！**

**前二个级别不用开发人员过多关注，很少用到，理解原理！！！Bean本身需要掌握！！！**

**IoC控制反转，通过接口实现类的选择控制权，同时选择控制权从调用移动到第三方的容器手中。**

1. **BeanFactory、ApplicationContext、WebApplicationContext是Spirng三个最核心的接口。**
2. **BeanFactory是顶层接口，ApplicationContext继承BeanFactory；WebApplicationContext继承ApplicationContext；**
3. **注意：BeanFactory在调用Bean（单例）的时候，才会创建Bean；但是ApplicationContext在加载容器的时候，就会自动注入单例Bean对象！！！**

**并且这些接口的实现，离不开Resource。**

**Spring中Bean的生命周期，三个级别，本身、Bean级别、以及容器级别（后处理器）。**

**几个关键的Spring包：**

**Beans：BeanFactory**

**Context：ApplicationContext**

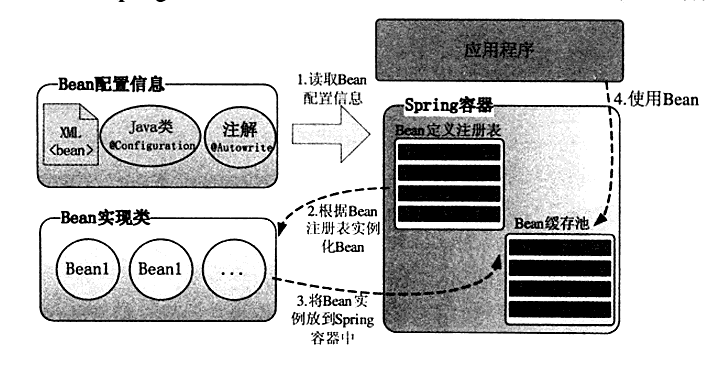
**Core：Resource**

**Expression**

**Aop**

**下一章节，讲详细介绍在Ioc容器中装配Bean。装配Bean推荐使用配置方式。或是注解方式。**

1. 在IOC中装配Bean
2. Spring配置概述



表象上不同，但是本质上一样，XML提供的功能最强大。Spring的配置是向前兼容，但是我们建议使用最新的配置。

1. Bean基本配置

使用Id属性作为Bean的名称，Id在IoC容器中必须唯一，命名要符合Java命名规范。

1. 依赖注入

3.1、属性注入

JavaBean属性的命名规范。

|  |
| --- |
| <!--注入方式一： 基于getter\setter方法的配置 ；属性必须有对应的getter和setter方法-->  <bean id=*"car"* class=*"com.springdi.ditype.Car"*>  <property name=*"brand"* value=*"红旗&amp;CA72"*/>  <property name=*"maxSpeed"* value=*"200"*/>  <property name=*"price"* value=*"20000.00"*/>  </bean> |
| <!-- 属性注入 二-->  <bean id=*"car1"* class=*"com.springdi.ditype.Car"* p:brand=*"红旗&amp;CA72"*  p:maxSpeed=*"200"* p:price=*"20000.00"* /> |

Eg：com.springdi.ditype.DitypeTests

3.2、构造函数注入

|  |
| --- |
| <!--构造函数注入一 -->  <bean id=*"car2"* class=*"com.springdi.ditype.Car"*>  <constructor-arg type=*"java.lang.String"*>  <value>红旗CA72</value>  </constructor-arg>  <constructor-arg type=*"double"*>  <value>20000</value>  </constructor-arg>  </bean> |
| <!--构造函数注入二 -->  <bean id=*"car3"* class=*"com.springdi.ditype.Car"*>  <constructor-arg index=*"0"*>  <value>红旗CA72</value>  </constructor-arg>  <constructor-arg index=*"1"*>  <value>20000</value>  </constructor-arg>  <constructor-arg index=*"2"*>  <value>200</value>  </constructor-arg>  </bean> |
| <!--构造函数注入三 -->  <bean id=*"car4"* class=*"com.springdi.ditype.Car"*>  <constructor-arg index=*"0"* type=*"java.lang.String"*>  <value>红旗CA72</value>  </constructor-arg>  <constructor-arg index=*"1"* type=*"double"*>  <value>20000</value>  </constructor-arg>  <constructor-arg index=*"2"* type=*"int"*>  <value>200</value>  </constructor-arg>  </bean> |
| <!--构造函数注入四：自动识别入参类型 -->  <bean id=*"boss1"* class=*"com.springdi.ditype.Boss"*>  <constructor-arg>  <value>John</value>  </constructor-arg>  <constructor-arg>  <ref bean=*"car"* />  </constructor-arg>  <constructor-arg>  <ref bean=*"office"* />  </constructor-arg>  </bean> |

Eg：com.springdi.ditype.DitypeTests

3.3、工厂方法注入

|  |
| --- |
| <!-- 工厂方法 -->  <bean id=*"carFactory"* class=*"com.springdi.ditype.CarFactory"* />  <!-- 通过工厂类注入一 -->  <bean id=*"car5"* factory-bean=*"carFactory"* factory-method=*"createHongQiCar"* /> |
| <!-- 通过工厂类注入二：静态工厂方法 -->  <bean id=*"car6"* class=*"com.springdi.ditype.CarFactory"* factory-method=*"createCar"* /> |

Eg：com.springdi.ditype.DitypeTests

3.4、如何选择合适的注入方式

Spring没有强制的推荐使用那种注入，但是对于新应用，**不推荐使用工厂方法注入**，因为他会增加额外的代码，Spring已经帮我们做了。

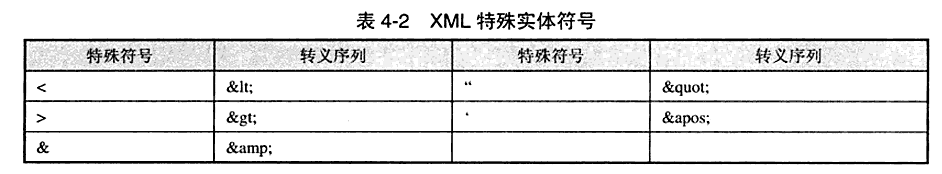
1. 注入参数详解

Spring还可以注入Map、集合到Bean中。

（1）加载字面值，以及特殊字符注入

|  |
| --- |
| <!-- 加载字面值 -->  <bean id=*"car"* class=*"com.springdi.attr.Car"* lazy-init=*"default"*>  <property name=*"brand"*>  <value>  <!-- 加载特殊字符 -->  <![CDATA[红旗&<>CA72]]>  </value>  </property>  <property name=*"maxSpeed"*>  <value>200</value>  </property>  <property name=*"price"* value=*"2000.00"* />  </bean> |

或者使用转义符



（2）内部bean

|  |
| --- |
| <!-- 内部Bean-->  <bean id=*"boss"* class=*"com.springdi.attr.Boss"*>  <property name=*"car"*>  <!-- 内部Bean不能被别人调用-->  <bean class=*"com.springdi.attr.Car"*>  <property name=*"maxSpeed"* value=*"200"* />  <property name=*"price"* value=*"2000.00"* />  </bean>  </property>  </bean> |

1. 集合类型属性：主要包括List、Set、Map等

|  |
| --- |
| <bean id=*"boss1"* class=*"com.springdi.attr.Boss"*>  <property name=*"car"* ref=*"car"* />  <!-- 集合注入 -->  <property name=*"favoritesList"*>  <list>  <value>看报</value>  <value>赛车</value>  <value>高尔夫</value>  </list>  </property>  <property name=*"favoritesSet"*>  <set>  <value>看报</value>  <value>赛车</value>  <value>高尔夫</value>  </set>  </property>  <property name=*"jobs"*>  <map>  <entry >  <key>  <value>AM</value>  </key>  <value>会见客户</value>  </entry>  <entry>  <key>  <value>PM</value>  </key>  <value>公司内部会议</value>  </entry>  </map>  </property>  <property name=*"mails"*>  <props>  <prop key=*"jobMail"*>john-office@baobaotao.com</prop>  <prop key=*"lifeMail"*>john-life@baobaotao.com</prop>  </props>  </property>  </bean> |

1. 方法注入--【用的场景不多】

Bean的作用范围，默认是单例，scope=*prototype*表示每次创建新的对象。

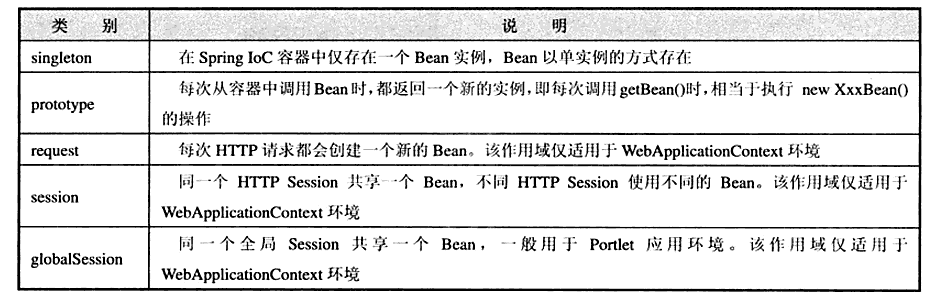
|  |
| --- |
| <!-- prototype -->  <bean id=*"car"* class=*"com.springdi.injectfun.Car"* p:brand=*"红旗CA72"*  p:price=*"2000"* scope=*"prototype"* />  <!-- lookup方法注入 -->  <bean id=*"magicBoss"* class=*"com.springdi.injectfun.MagicBoss"*>  <lookup-method name=*"getCar"* bean=*"car"* />  </bean> |
| <!-- 方法替换 -->  <bean id=*"boss2"* class=*"com.springdi.injectfun.Boss2"* />  <bean id=*"boss1"* class=*"com.springdi.injectfun.Boss1"*>  <replaced-method name=*"getCar"* replacer=*"boss2"* />  </bean> |

1. bean之间的关系

类似OOP，子类通过继承父类的方法，实现代码的精简。Bean配置之间也是可以。

|  |
| --- |
| <!-- 父子<bean>,子类继承于父类的bean配置，简化配置量 -->  <bean id=*"abstractCar"* class=*"com.springdi.tagdepend.Car"* p:brand=*"红旗CA72"*  p:price=*"2000.00"* p:color=*"黑色"* abstract=*"true"* />  <bean id=*"car1"* parent=*"abstractCar"*>  <property name=*"color"* value=*"红色"* />  </bean>  <bean id=*"car2"* parent=*"abstractCar"*>  <property name=*"color"* value=*"白色"* />  </bean> |

1. bean的作用域



无状态或者状态不可变的类适合使用单例模式(对象是线程安全的)。例如java.sql.connection。SqlSessionFactory.

ApplicationContext容器在启动的时候，会自动实例化所有单例的bean,并缓存到容器中。牺牲了启动的时间，但是加快了运行的时间。

在默认情况下，Spring容器在启动的时候，不实例化prototype的Bean。

1. 自定义配置：factoryBean

通过实现factoryBean接口，可以实现自定义的配置方式配置Bean。

|  |
| --- |
| <bean id=*"car1"* class=*"com.springdi.fb.CarFactoryBean"* p:carInfo=*"红旗CA72,200,20000.00"* /> |

1. 基于注解的配置

9.1、使用注解定义Bean

从2.0开始支持注解，2.5得到完善；

|  |
| --- |
| /\*\*  \* 使用Component注解在UserDao类声明，它可以被Spring容器识别，Spring将其作为Bean管理  \*  \* 等价于在xml配置：<bean id="userDao" class="com.springdi.anno.UserDao"/>  \*  \* 另外，除了Component，Spring另外提供了三个注解，功能和Component一样，分别用于：  \*  \* Repository：持久层，Dao  \* Service：服务层，Service  \* Controller：控制层，Controller  \*  \* **@author** Ryan  \*  \*/  @Component("userDao")  **public** **class** UserDao {  ... ...  } |
| 等价于  <bean id=”userDao” class=”.../userDao”/> |

9.2、自动装配Bean

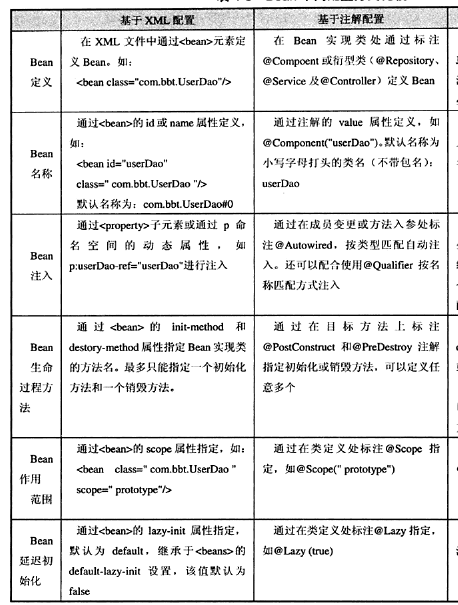
|  |
| --- |
| /\*\*  \* 使用Service注解，自动装配Bean  \* Bean的名称就是logonService，默认不写也是它！  \*  \* **@author** Ryan  \*  \*/  @Service("logonService")  **public** **class** LogonService {  /\*\*  \* 通过Autowired自动注入Ｂｅａｎ  \* 使用logDao Bean装配对象  \*/  @Autowired  @Qualifier("logDao")  **private** LogDao logDao;  ... |
| Autowired:自动装配Bean  Qualifier：指定装配时的Bean名称 |

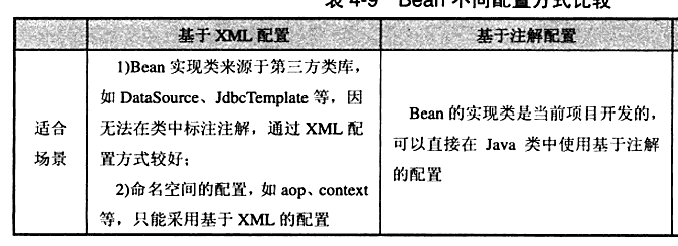
9.3、Bean的作用范围和生命周期

|  |
| --- |
| 1、调用默认构造函数，创建对象  2、调用setter方法，注入属性  3、init-method  4、destory-method |

1. 不同配置之间的对比

基于Xml配置、基于注解配置、基于Java代码配置--不常用



使用原则：一般都是Xml配置+注解

1. Spring容器高级主题

Spring容器像一台精妙的机器，通过配置文件向机器传递各种信息，机器按照设定的模式工作；

如果BeanFactory发动机、ApplicationContext则是整个汽车；

1. 使用外部化的属性文件

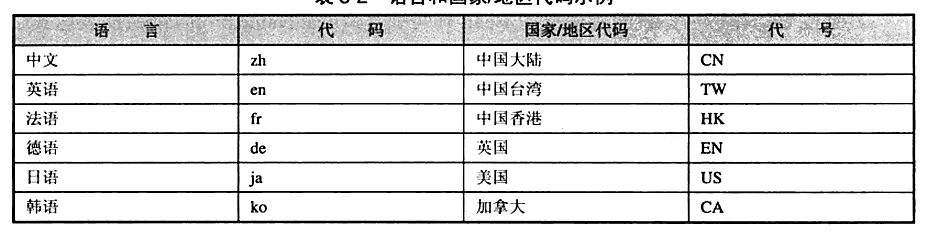
例如Spring可以在配置文件中，配置数据库连接字符串，包括用户名、密码等。但是更好的方式，是利用占位符，加载外部化的属性文件，好处：减少维护的工作量以及部署更加简单，运维人员只需要关注属性文件即可。不需要知会庞大的Spring配置文件。

Eg：spring-ioc/com.springioc.placeholder

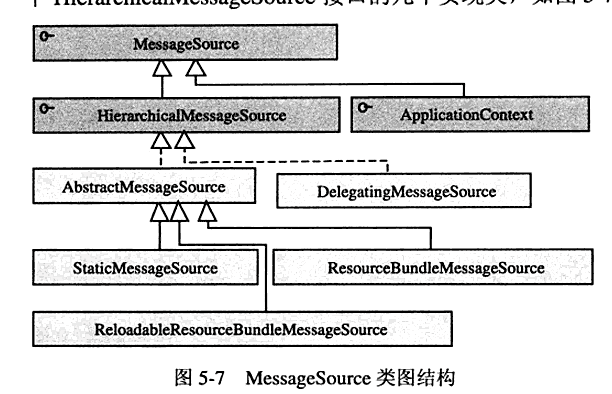
1. 国际化信息

为每一种语言，提供一套相应的资源文件，并以规划化命名方式保存到特定的目录中，由系统自动根据客户端语言选择合适的资源文件。

ISO中规定，每个国家或地区，由二个大写字母表示。



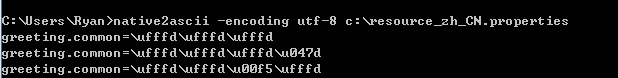
Spring通过MessageSource接口访问国际化信息



Eg：spring-ioc/com.springioc.i18n

补充：使用Jdk/bin下提供的工具，可以把本地文字转AscII。

命令：



源文件内容：

|  |
| --- |
| greeting.common=你好  greeting.common=你好呀  greeting.common=你好的 |