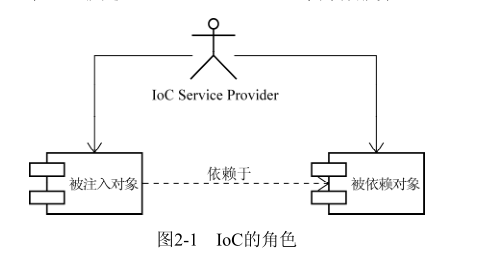
# 第二章 IOC的基本概念

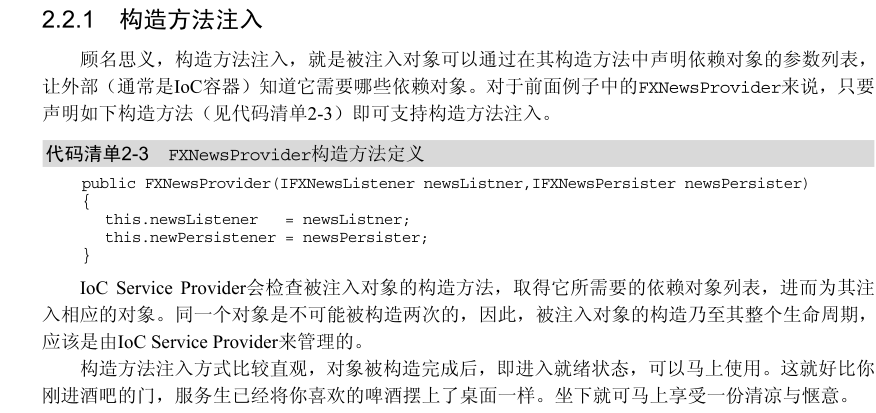
1.什么是IOC



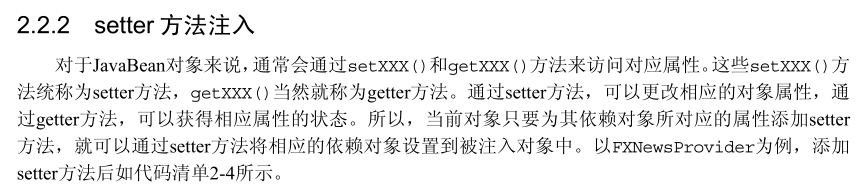
说明：通常情况下,被注入对象(A)依赖于被依赖对象(B).在IOC场景中,被注入对象和被依赖对象统一被IoC Service Provider容器统一管理。A需要B,直接向IOC容器(C)发出请求,c就会把B注入到A.在这个过程(与之前相比),获取依赖对象的方式发生反转（原来A自己创建,现在C获取直接给A）,原来的控制权在A,现在在C。

2.依赖注入的三种方式

a.构造方法注入



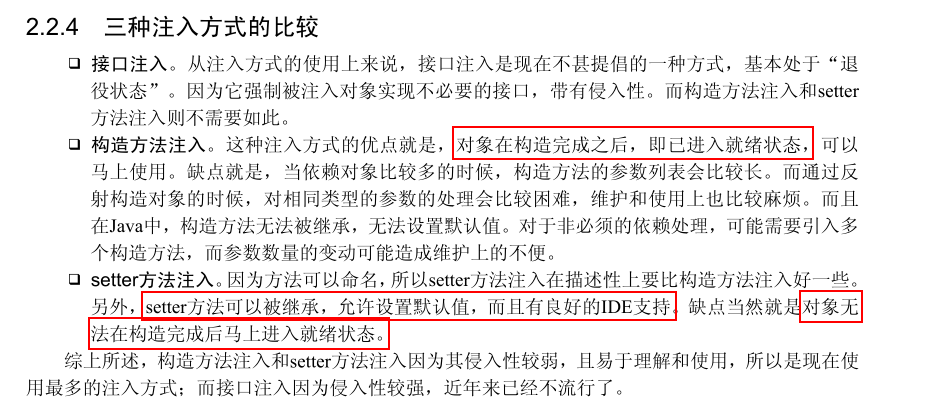
b.setter注入





c.接口注入

三种方式比较：



# 第三章 IoC Service Provider

1. IoC Service Provider 的职责

(1)业务对象的构建管理：（统一管理bean:注册+创建。）在IoC场景中，业务对象无需关心所依赖的对象如何构建如何取得，但这部分工作始终需要有人来做。所以，IoC Service Provider需要将对象的构建逻辑从客户端对象那里剥离出来，以免这部分逻辑污染业务对象的实现。

(2)业务对象间的依赖绑定：(即依赖注入,建立依赖关系)对于IoC Service Provider来说，这个职责是最艰巨也是最重要的，这是它的最终使命之所在。如果不能完成这个职责，那么，无论业务对象如何的“呼喊” ，也不会得到依赖对象的任何响应 （最常见的倒是会收到一个 NullPointerException ） 。 IoC Service Provider通过结合之前构建和管理的所有业务对象， 以及各个业务对象间可以识别的依赖关系,将这些对象所依赖的对象注入绑定，从而保证每个业务对象在使用的时候，可以处于就绪状态。

2.容器如何管理依赖关系

(1) 直接编码方式

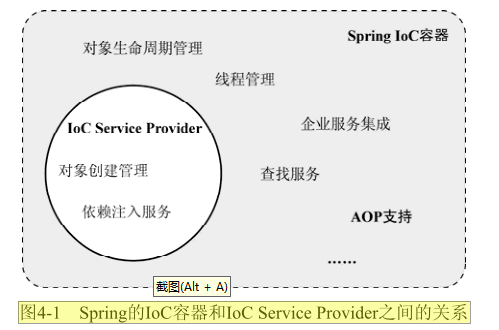
(2) 配置文件方式(applicationContext.xml)

(3) 元数据方式(注解方式)

# 第四章 Spring的IOC容器-BeanFactory

## 4.1

1. Spring的IoC容器是一个提供IoC支持的轻量级容器，除了基本的IoC支持，它作为轻量级容器还提供了IoC之外的支持。如在Spring的IoC容器之上，Spring还提供了相应的AOP框架支持、企业级服务集成等服务。



2.spring的两种容器类型

(1)BeanFactory:

o.基础类型IoC容器，提供完整的IoC服务支持.

a.默认采用延迟初始化策略（lazy\_load）;

说明：只用当客户端对象（eg.service层的impl类）需要使用某一个受管对象（eg.mapper类）的时候,才会对受管对象进行初始化以及依赖注入操作(可见，容器初始化时候，所有的bean都已经被管

理，但是此时不知道这些对象之间的依赖关系？？知道依赖关系，但是就是不注入？？）。

b. 速度and资源：由于a的原因,启动快,需要资源有限。

c.使用场景:对于资源有限,并且功能要求不是很严格的场景,BeanFactory 是比较合适的IoC容器选择.

(2)ApplicationContext:

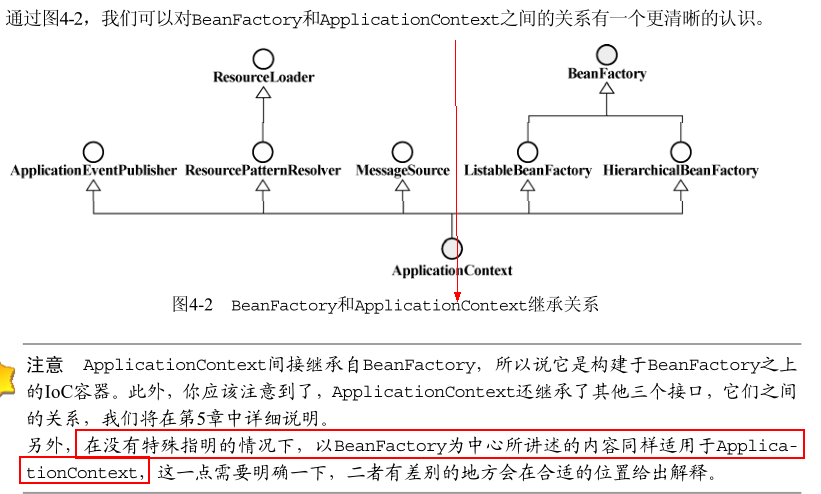
o.相对比较高级的容器,除了上面支持的之外,还提供了其他高级特性，比如事件发布、国际化信息支持等

a. 所管理的对象,在容器启动之后，默认全部初始化并绑定(是依赖注入的意思吗？？)完成。

b.速度and资源:由于a的原因,容器启动叫慢,启动所需更多系统资源。

c.使用场景:在那些系统资源充足,并且要求更多功能的场景中,ApplicationContext 类型的容器是比较合适的选择

两者关系



(3)代码演示获取对象（两种容器的不同获取方式）

**public** **static** **void** main(String[] args) {

BeanFactory container = **new** ~~XmlBeanFactory~~(**new** ClassPathResource("applicationContext.xml"));

IUserService userService = (IUserService)container.getBean("userService");

System.***out***.println(userService.readUser().getName());

ApplicationContext container1 =**new** ClassPathXmlApplicationContext("applicationContext.xml");

IUserService userService1 = (IUserService)container1.getBean("userService");

System.***out***.println(userService1.readUser().getName());

ApplicationContext container2 =**new** FileSystemXmlApplicationContext("classpath:applicationContext.xml");

IUserService userService2 = (IUserService)container2.getBean("userService");

System.***out***.println(userService2.readUser().getName());

}

## 4.2BeanFactory对象注册和依赖绑定的三种方式

1.编码方式

//容器

**DefaultListableBeanFactory** registry = new DefaultListableBeanFactory();

//创建BeanDefinition

**AbstractBeanDefinition** newsProvider =

new RootBeanDefinition(FXNewsProvider.class,true);

//注册到容器。主要利用的是:容器的注册功能，即BeanDefinitionRegistry接口的功能

registry.registerBeanDefinition("newsProvider", newsProvider);

//置顶依赖关系

// 1. 可以通过构造方法注入方式

**ConstructorArgumentValues** argValues = new ConstructorArgumentValues();

argValues.addIndexedArgumentValue(0, newsListener);

argValues.addIndexedArgumentValue(1, newsPersister);

newsProvider.setConstructorArgumentValues(argValues);

// 2. 或者通过setter方法注入方式

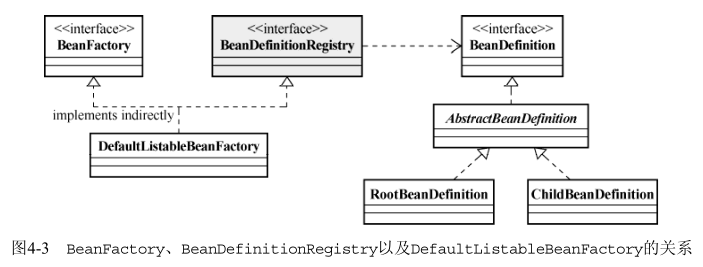
**MutablePropertyValues** propertyValues = new MutablePropertyValues();

propertyValues.addPropertyValue(new ropertyValue("newsListener",newsListener));

propertyValues.addPropertyValue(new PropertyValue("newPersistener",newsPersister));

newsProvider.setPropertyValues(propertyValues);

// 绑定完成



解释上图：

1. BeanFactory(接口)主要提供从容器获取对象的逻辑
2. BeanDefinitionRegistry(接口)主要提供想容器注册对象的逻辑
3. 每一个受管对象，在容器中都会有一个BeanDefinition实例与之相对应。

说明：该BeanDefinition 的实例负责保存对象的所有必要信息,包括其对应的对象的class类型、是否是抽象类、构造方法参数以及其他属性等.当客户端向 BeanFactory 请求相应对象的时候, BeanFactory 会通过这些信息为客户端返回一个完备可用的对象实例.

1. RootBeanDefiniton和ChildBeanDefiniton是BeanFactory的两个主要实例。

2.配置文件方式（properties,xml ,BeanDefinitionReader的实现类去加载不同配置文件）

a.applicationContext.xml配置

3.接口方式

## 4.3 BeanFactory 的XML之旅(配置文件各个节点说明解析)

### 1.<beans>

### 1.1<description></description>

说明：

### 1.2<import resource="classpath\*:applicationContext\_test.xml"/>

<!-- 引入其他配置文件,便于分门别类。一旦引入，就可以使用该配置文件定义的bean了。 其实，spring容器可以加载多个配置文件,可以不使用这个标签。 -->

<import resource=*"classpath\*:applicationContext\_test.xml"*/>

### 1.3<bean>

属性说明：

* id

@唯一标识。指定当前注册对象的 beanName 是什么.

@非必要。

有些情况id 可以省略,比如内部<bean>以及不需要根据beanName明确依赖关系的场合.

@

* name

@非必要。

@给这个bean定义别名，可以是多个。可以包含特殊字符/。多个别名以逗号或者空格或者冒号隔开

* class

@必要（极少数不必要。）

@仅在少数情况下不需要指定，如后面将提到的在使用抽象配置模板的情况下。

#### 1.3.1setter注入和构造器注入

Eg.

<bean id=*"userService"* class=*"net.shopin.xyy.service.UserServiceImpl"*></bean>

<bean id=*"constructorInjectDef"* class=*"net.shopin.xyy.ConstructorInjectDef"*>

<!-- type:指明参数的类型,默认为String类型；index:指定位于参数列表的位置，从0开始。 -->

<constructor-arg index=*"0"* type=*"Integer"* value=*"7"*></constructor-arg>

</bean>

<bean id=*"setterInjectionDef"* class=*"net.shopin.xyy.SetterInjectionDef"*>

<property name=*"name"* value=*"setter注入"*></property>

<constructor-arg index=*"0"* ref=*"userService"*></constructor-arg>

</bean>

**注意事项：**

如果只是使用 <property> 进行依赖注入的话，请确保你的对象提供了默认的构造方法,也就是一个参数都没有的那个.

#### 1.3.2<property> 和 <constructor-arg> 中可用的配置项

##### 1.3.2.1 通用元素

* <value>

说明：注入简单数据类型及其包装对象或者String类型的值（4类8种）。

<value> 是最“底层”的元素，它内部不能再嵌套使用其他元素

<constructor-arg>

<value>111111</value>

</constructor-arg>

<property name="attributeName">

<value>222222</value>

</property>

简化版：

这里的 value 是以上一层元素的属性身份出现

<constructor-arg value="111111"/>

<property name="attributeName" value="222222"/>

* <ref>

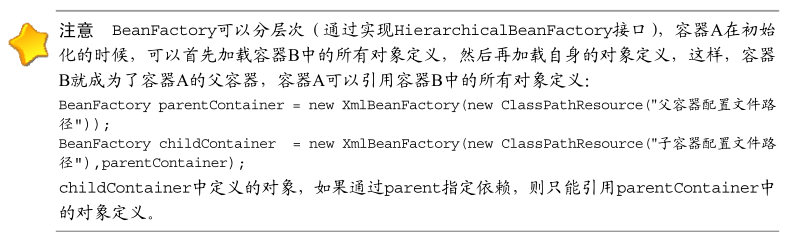
属性说明：

@parent:表明引用父容器配置文件内的对象定义。只能指定位于当前容器的父容器中定义的对象引用

@bean: 通常情况下，直接使用bean来指定对象引用就可以了

@此标签没有子元素。

@说明：此标签表示引用容器内的其他对象。（容器在实例化对象A时，发现A依赖对象B,那么容器会先去实例化对象B,B完成后，再来实例化A,so,实例化一个对象，会先去实例化它依赖的对象。显示依赖ref,或者是非显式依赖depends-on）



* <idref>

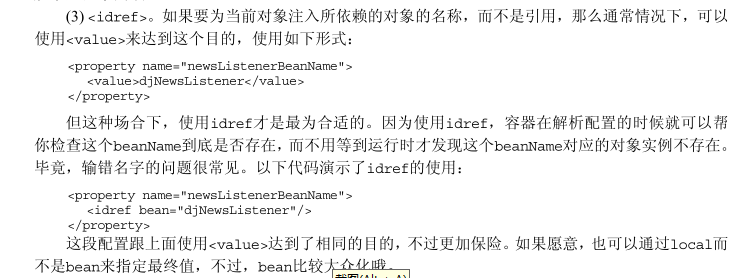
如果对象的属性值是一个依赖对象的名称，

@使用idRef,启动时就会去检查容器中是否存在这个bean

@也可以使用value,只有真正使用这个属性的时候，寻找bean,没有，再报错。

@实验案例见下面<list><set><map><props>的案例（包含在内）

的案例

。

实验案例的结果：

Spring容器中没有名称是userService2(配置文件定义的名称是userService),启动时候就报错了，起到了检查的作用。



* 内部 <bean>

说明：当一个bean不需要被其他bean所依赖，只是被某一次使用，可以直接使用内部bean

<bean id=*"innerBeanDef"* class=*"net.shopin.xyy.InnerBeanDef"*>

<constructor-arg index=*"0"*>

<bean class=*"net.shopin.xyy.service.UserServiceImpl"*></bean>

</constructor-arg>

</bean>

* <list><set><map><props>
  + <list>:对象属性type为list及其子类或者数组
  + <set>:对象属性type为set及其子类
  + <map>:对象属性type为map
  + <props>:对象属性type为Map<String,String>或者Properties

public class ListInjectDef {

private List<String> lists ;

private Set<String> sets;

private Map<String,Object> maps;

private Properties propertiess;

private String injectBeanName;

…….省略setter getter…..

toString(){……..};

}

<bean id=*"listInjectDef"* class=*"net.shopin.xyy.ListInjectDef"*>

<property name=*"injectBeanName"*>

<idref bean=*"userService2"*/>

</property>

<property name=*"lists"*>

<list>

<value>list1</value>

<value>list2</value>

<value>list3</value>

</list>

</property>

<property name=*"sets"*>

<set>

<value>set1</value>

<value>set2</value>

<value>set3</value>

</set>

</property>

<property name=*"maps"*>

<map>

<entry key=*"mapkey1"* value=*"mapvalue1"*></entry>

<entry key=*"mapkey2"* value-ref=*"userService1"* >

</entry>

<entry key=*"mapkey3"*>

<list>

<ref bean=*"userService1"*/>

<ref bean=*"userService1"*/>

<ref bean=*"userService1"*/>

</list>

</entry>

</map>

</property>

<property name=*"propertiess"*>

<props>

<prop key=*"email1"*>123@qq.com</prop>

<prop key=*"email2"*>mimi@qq.com</prop>

</props>

</property>

</bean>

**注意事项：**property的name属性,来源于setter方法，而不是我们定义的属性名称。

* <null/>

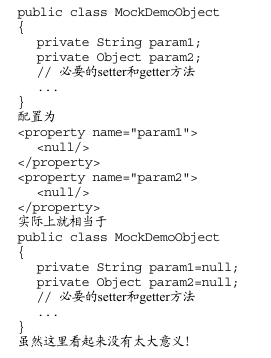
说明:当属性需要一个null空值，使用此标签。

<property name=*"injectBeanName"*>

<!-- 为String类型属性注入null值，如果使用<value></value>只用注入空字符串，不是空对象了。 -->

<null />

</property>



#### 1.3.2.2 depends-on（非显式依赖，非ref）

说明：

@显示指定所依赖的对象，要求容器初始化本bean之前，先去实例化depends-on的实例。（在没有使用ref的情况下。）

@每一个类依赖的类不一定都显示的写在类中。Eg. 系统中所有需要日志记录的类，都需要在这些类使用之前首先初始化log4j。这里就没有办法使用ref,这时就可以使用depends-on。

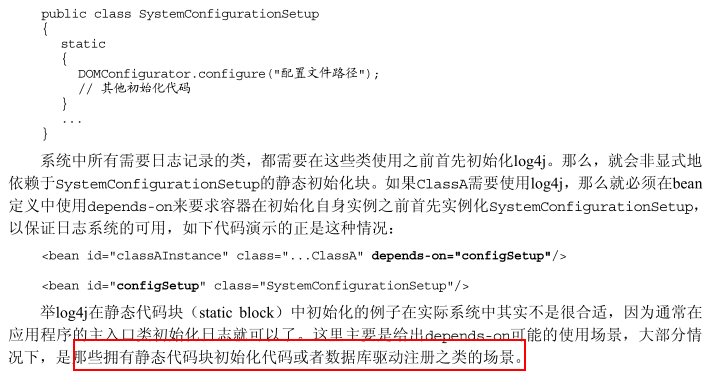
@包含多个非显式的以来，depends-on内用逗号隔开。

<bean id="classAInstance" class="...ClassA" depends-on="configSetup,configSetup2,..."/>

<bean id="configSetup" class="SystemConfigurationSetup"/>

<bean id="configSetup2" class="SystemConfigurationSetup2"/>

没看懂？？



基于 Spring 设计并实现 RESTful Web Services <http://www.linuxidc.com/Linux/2013-10/91974.htm>

Spring-3.2.4 + Quartz-2.2.0集成实例 <http://www.linuxidc.com/Linux/2013-10/91524.htm>

使用 Spring 进行单元测试 <http://www.linuxidc.com/Linux/2013-09/89913.htm>

运用Spring注解实现Netty服务器端UDP应用程序 <http://www.linuxidc.com/Linux/2013-09/89780.htm>

Spring 3.x 企业应用开发实战 PDF完整高清扫描版+源代码 <http://www.linuxidc.com/Linux/2013-10/91357.htm>

**Spring 的详细介绍**：[请点这里](http://www.linuxidc.com/Linux/2013-12/93921.htm)  
**Spring 的下载地址**：[请点这里](http://www.linuxidc.com/down.aspx?id=1143)