



Spring框架技术

—— 依赖注入的实现

本章内容

节	知识点	掌握程度	难易程度
解析IoC和DI	解析IoC和DI	理解	难
	控制反转/依赖注入	理解	
依赖注入的三种方式	依赖注入的三种方式	掌握	
	构造函数注入	掌握	
	属性注入	掌握	
集合类型属性的注入	实现集合类型属性的注入	掌握	
手工装配依赖对象	配置文件方法	掌握	
	Autowired注解方式	掌握	
	Resource注解方式	掌握	
自动装配对象	按类型自动装配	了解	
	按名字自动装配	了解	
自动扫描方式装配对象	自动扫描方式装配对象	掌握	

典型的企业应用不会只由单一的对象（或bean）组成。毫无疑问，即使最简单的系统也需要多个对象一起来满足最终用户的需求。接下来的内容除了阐述如何单独定义一系列bean外，还将描述如何让这些bean对象一起协同工作来实现一个完整的真实应用。

解析IoC和DI

- IoC:
 - Inversion of Control
 - Dependency Injection
- IoC的直译是控制反转。
- 在IoC模式下，控制权限从应用程序转移到了IoC容器中。组件不是由应用程序负责创建和配置，而是由IoC容器负责。
- 使用IoC的情况下，对象是被动地接收依赖类而不是主动地查找。对象不是从容器中查找他的依赖类，而是容器在实例化对象时，主动地将他所依赖的对象注入给他。
- 应用程序只需要直接使用已经创建并且配置好的组件即可，而不必自己负责创建和配置。
- 在实际的工作中人们发现使用IoC来表述这种机制，并不是很准确甚至有些晦涩，于是引发了另外一个概念：DI（依赖注入）

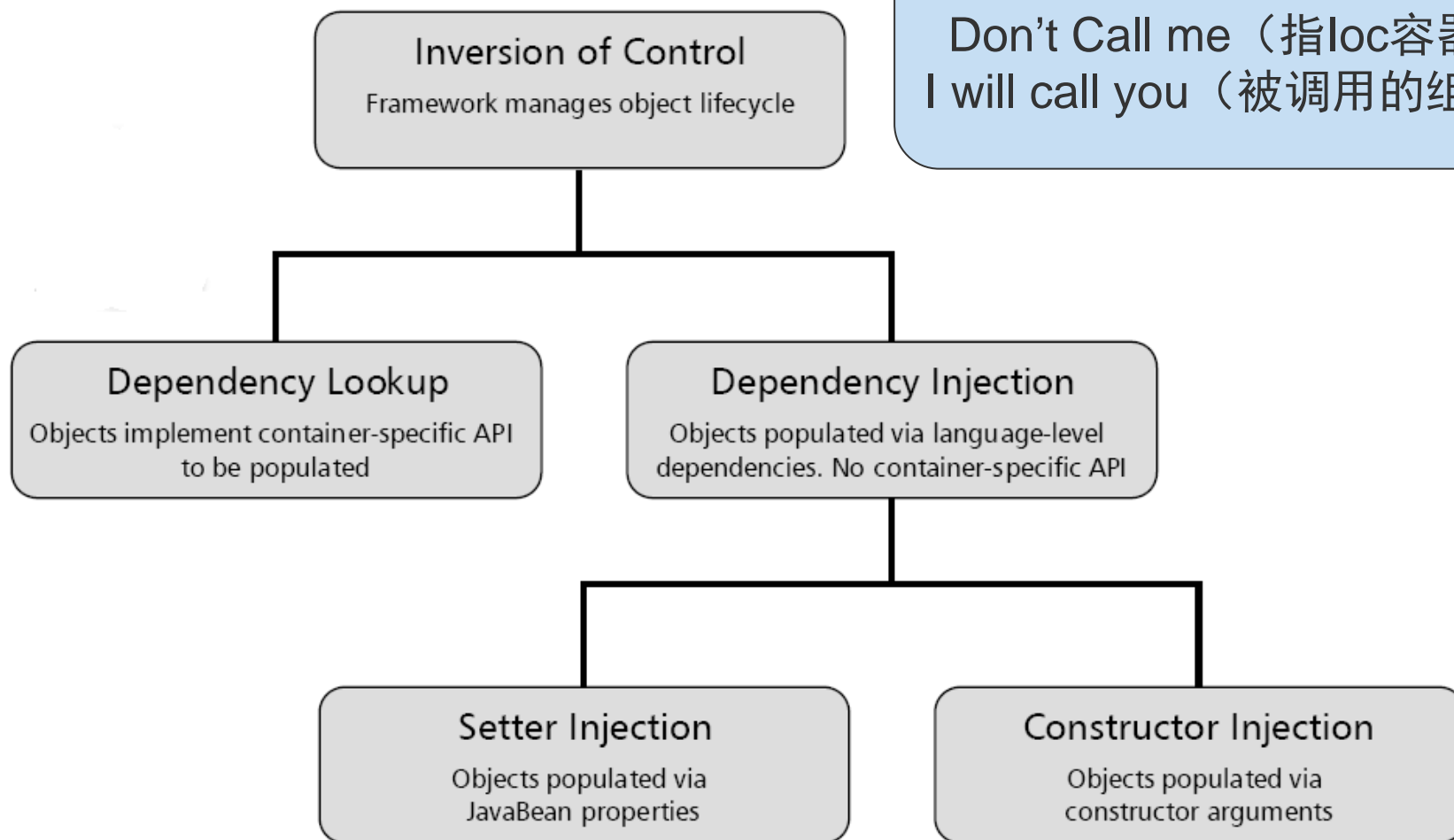
解析IoC和DI

- DI解析
 - 之所以会产生组件调用，是为了获取被调用组件的功能，调用者将自己应该做的事情，委派给了被调用的对象。也就是说，调用者要完成自身的任务，必须依赖被调用的对象。这种关系实际上就是一般依赖关系(通俗点说，就是一个组件内部包含另外一个组件的实例，把该自己干的事交给自己包含的组件去完成)。因此IoC所表述的机制，实际上就是将调用者对接口实现类的依赖关系，从程序中移除，转交第三方管理实例。并且，由第三方在运行期间将调用者依赖的具体类填充进来。也就是说组件之间的依赖关系，是在程序运行期间由第三方来管理的。这个就是依赖注入的概念(DI)，基于上述分析，DI比IoC更准确。
 - 实际上就是将调用者为完成功能所依赖的实现类，在程序运行期间，由容器自动填充给调用者，这个就是依赖注入的核心思想。在依赖注入的应用中，组件并不关心被注入的对象是谁，只关系这个对象能完成的功能，也就是这个对象是哪个接口的具体类实例。

控制反转/依赖注入

好莱坞原则

Don't Call me (指ioc容器),
I will call you (被调用的组件).



IoC的三种类型

- 构造函数注入
 - 所需信息通过构造函数参数注入
- 属性注入
 - 所需信息通过属性的set方法注入
- 接口注入
 - 实现接口，所需信息通过接口定义的方法注入。如EJB方式。
- 示例 `spring-whySpring-2`工程

Spring IOC

- Spring IOC支持构造函数注入和属性注入
 - 接口注入要求bean类实现接口，其入侵性强，与Spring设计初衷不符
- ```
public class MyServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse
response)
 throws ServletException, IOException {

 }
}
```



# Spring IOC

- IOC容器的关键点：
  - 必须将被管理的对象定义在Spring的配置文件中
  - 必须定义构造函数或setter方法，让spring将对象注入过来
- 示例 spring-whySpring-2工程
- 示例 spring-inject工程

# 构造函数注入

```
public class TextPrinter {
 private Formater formater;
 private int size;

 public TextPrinter(Formater formater, int size) {
 this.formater=formater;
 this.size=size;
 }

 public void print(String info) {
 System.out.println("Set size="+this.size);
 System.out.println(this.formater.execute(info));
 }
}
```

# 构造函数注入

```
<beans>
 <bean id="formaterBean" class="example.LowerFormater">
 <property name="title">
 <value>Hello Spring</value>
 </property>
 </bean>

 <bean id="printer" class="example.TextPrinter">
 <constructor-arg index="0">
 <ref bean="formaterBean" />
 </constructor-arg>
 <constructor-arg index="1" type="int" value="5" />
 </bean>
</beans>
```

87

# 构造函数注入

```
public static void main(String[] args) {
 XmlBeanFactory factory = new XmlBeanFactory(new ClassPathResource("mybeans.xml"))
 TextPrinter bean = (TextPrinter) factory.getBean("printer");
 bean.print("spring ioc");
}
```

输出：

Set size=5

hello spring : spring ioc

# 构造函数注入

- 构造器注入
  - 构造器参数类型匹配
  - 构造器参数索引
- 示例 `spring-inject`工程

# 属性注入

```
public class TextPrinter {
 private Formater formater;
 private int size;

 public Formater getFormater() {
 return formater;
 }

 public void setFormater(Formater formater) {
 this.formater = formater;
 }

 public int getSize() {
 return size;
 }

 public void setSize(int size) {
 this.size = size;
 }

 public void print(String info) {
 System.out.println("Set size=" + this.size);
 System.out.println(this.formater.execute(info));
 }
}
```

# 属性注入

```
<beans>
 <bean id="formaterBean" class="example.LowerFormater">
 <property name="title">
 <value>Hello Spring</value>
 </property>
 </bean>

 <bean id="printer" class="example.TextPrinter">
 <property name="formater">
 <ref bean="formaterBean"/>
 </property>
 <property name="size">
 <value type="int">5</value>
 </property>
 </bean>
</beans>
```

# 属性标签

- value: 基本类型（包装类型）或String类型
- ref: 引用类型，容器中其它的bean
- list: List或数组类型
- set: Set类型。用法与list类似。
- map: Map类型
- props: Properties类型，键值为String类型的，所以直接写值。
- 示例 spring-complex工程



# 集合属性的用法

```
<property name="listProperty">
 <list>
 <value>string
 value</value>
 <ref bean="foo"/>
 <ref bean="bar"/>
 </list>
</property>
```

```
<property name="setProperty">
 <set>
 <value>string
 value</value>
 <ref bean="foo"/>
 <ref bean="bar"/>
 </set>
</property>
```

```
<property
 name="mapProperty">
 <map>
 <entry key="key">
 <ref bean="foo"/>
 </entry>
 </map>
</property>
```

```
<property name="propsProperty">
 <props>
 <prop
 key="key1">bar1</prop>
 <prop
 key="key2">bar2</prop>
 </props>
</property>
```

# 集合属性的用法

```
<bean id="complexObject" class="example.ComplexObject">
 <property name="people">
 <props>
 <prop key="Tom">cat</prop>
 <prop key="Jerry">mouse</prop>
 </props>
 </property>

 <property name="someList">
 <list>
 <value>a list element followed by a reference</value>
 <ref bean="formaterBean" />
 </list>
 </property>

 <property name="someMap">
 <map>
 <entry key="yup an entry">
 <value>just some string</value>
 </entry>
 <entry key="yup a ref">
 <ref bean="printer" />
 </entry>
 </map>
 </property>
</bean>
```

# 依赖注入

- 使用构造器注入
- 使用属性setter方法注入
- 使用Field注入（用于注解方式）
- 注入依赖对象可以采用手工装配或自动装配，在实际应用中建议使用手工装配，因为自动装配会产生未知情况,开发人员无法预见最终的装配结果。

1.手工装配依赖对象

2.自动装配依赖对象

# 依赖注入--手工装配

1. 在xml配置文件中，通过在bean节点下配置，如

```
<bean id="orderService" class="cn.neusoft.service.OrderServiceBean">
 <constructor-arg index="0" type="java.lang.String" value="xxx"/> //构造器注入
 <property name="name" value="zhao"/> //属性setter方法注入
</bean>
```

示例 spring-inject工程

# 依赖注入--手工装配

## 2. 注解方式：示例 `spring-inject-annotation`工程

在java代码中使用@Autowired或@Resource注解方式进行装配。但我们需要在xml配置文件中配置以下信息：

```
<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-2.5.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-2.5.xsd">
 <context:annotation-config/>
</beans>
```

这个配置隐式注册了多个对注释进行解析处理的处理器：

AutowiredAnnotationBeanPostProcessor,  
CommonAnnotationBeanPostProcessor,  
PersistenceAnnotationBeanPostProcessor,  
RequiredAnnotationBeanPostProcessor

注： @Resource注解在spring安装目录的lib\j2ee\common-annotations.jar

# 依赖注入--手工装配

- 在java代码中使用 @Autowired或@Resource注解方式进行装配，这两个注解的区别是：

@Autowired 默认按类型装配，

@Resource默认按名称装配，当找不到与名称匹配的bean才会按类型装配。

@Autowired

```
private PersonDao personDao;//用于字段上
```

@Autowired

```
public void setOrderDao(OrderDao orderDao) {//用于属性的setter方法上
 this.orderDao = orderDao;
}
```

@Autowired注解是按类型装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许null值，可以设置它required属性为false。

如果我们想使用按名称装配，可以结合 @Qualifier注解一起使用。如下：

```
@Autowired @Qualifier("personDaoBean")
```

```
private PersonDao personDao;
```

# 依赖注入--手工装配

- 在java代码中使用 @Autowired或 @Resource注解方式进行装配,

@Resource注解和@Autowired一样, 也可以标注在字段或属性的setter方法上, 但它默认按名称装配。

名称可以通过@Resource的name属性指定, 如果没有指定name属性, 当注解标注在字段上, 即默认取字段的名称作为bean名称寻找依赖对象, 当注解标注在属性的setter方法上, 即默认取属性名作为bean名称寻找依赖对象。

`@Resource(name="personDaoBean")`

`private PersonDao personDao;`//用于字段上

注意: 如果没有指定name属性, 并且按照默认的名称仍然找不到依赖对象时, @Resource注解会回退到按类型装配。但一旦指定了name属性, 就只能按名称装配了。

# 依赖注入--自动装配依赖对象

- 示例 `spring-inject-autoByName`工程
- 示例 `spring-inject-autoByType`工程
- 对于自动装配，大家了解一下就可以了，实在不推荐大家使用。例子：  
`<bean id="..." class="..." autowire="byType"/>`  
autowire属性取值如下：
  - byType：按类型装配，可以根据属性的类型，在容器中寻找跟该类型匹配的bean。如果发现多个，那么将会抛出异常。如果没有找到，即属性值为null。
  - byName：按名称装配，可以根据属性的名称，在容器中寻找跟该属性名相同的bean，如果没有找到，即属性值为null。
  - constructor与byType的方式类似，不同之处在于它应用于构造器参数。如果在容器中没有找到与构造器参数类型一致的bean，那么将会抛出异常。
  - autodetect：通过bean类的自省机制（introspection）来决定是使用constructor还是byType方式进行自动装配。如果发现默认的构造器，那么将使用byType方式。



# 自动扫描

- 示例 `spring-autoscan`工程
- 通过在classpath自动扫描方式把组件纳入spring容器中管理

前面的例子我们都是使用XML的bean定义来配置组件。在一个稍大的项目中，通常会有上百个组件，如果这些组件采用xml的bean定义来配置，显然会增加配置文件的体积，查找及维护起来也不太方便。spring2.5为我们引入了组件自动扫描机制，他可以在类路径底下寻找标注了@Component、@Service、@Controller、@Repository注解的类，并把这些类纳入进spring容器中管理。它的作用和在xml文件中使用bean节点配置组件是一样的。要使用自动扫描机制，我们需要打开以下配置信息：

```
<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-2.5.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-2.5.xsd">
 <context:component-scan base-package="cn.neusoft"/>
</beans>
```

其中base-package为需要扫描的包(含子包)。

@Service用于标注业务层组件、@Controller用于标注控制层组件（如struts中的action）、@Repository用于标注数据访问组件，即DAO组件。而@Component泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注。

# 本章重点总结

- 理解IOC和DI的概念
- 掌握属性注入和构造器注入
- 掌握注解方式注入
- 掌握自动扫描方式

# Neuedu