Spring Bean自动注入（DI）

在Spring IoC中，容器帮助对象找相应的依赖对象并注入，而不是由对象主动创建，这样可以提升组件间重用，并为系统搭建灵活可扩展的平台。IoC和DI是Spring IoC容器同一概念的不同角度描述：

1. IoC（Inversion of Control），是Spring的核心，对于Spring框架来说由Spring来负责控制对象的生命周期和对象间的关系。Spring IoC容器根据各种形式的Bean配置信息在容器内部建立Bean定义注册表，然后根据注册表加载、实例化Bean并建立Bean之间的依赖关系，最后将其放入Bean缓存池中，以供外层应用程序的调用。
2. DI(Denpendency Injection)，IoC的另外的重点是系统运行中，动态向某个对象提供它所需要的其他对象，由DI（依赖注入)来实现。

依赖注入在Spring中有两种配置方式：XML配置和自动装配检测Bean，下面进行介绍

# XML配置Bean

配置如下：

*<bean id="student" class="com.fys.spring.basic.Student" autowire="byName">*

*<property name="name" value="tom"/>*

*</bean>*

*<bean id="teacher" class="com.fys.spring.basic.Teacher">*

*<property name="name" value="Jerry"/>*

*</bean>*

可与上文中进行对比，没有为Student对象注入Teacher属性，而是使用autowire=”byName”替代，Spring会自动根据名字将Teacher组件注入到Student对象中

*<bean id="student" class="com.fys.spring.basic.Student">*

*<property name="name" value="tom"/>*

*<property name="teacher" ref="teacher"/> //显示注入Teacher*

*</bean>*

在Spring中除了提供byName之外，还提供以下几种注入策略：

* no, 显示指明不使用Spring的自动装配功能
* byName,根据属性和组件的名称匹配关系来实现Bean的自动装配
* byType，根据属性和组件的类型匹配关系来实现Bean的自动匹配
* constructor，与byType类型类似根据类型自动匹配，但是要求待装配的bean有相应的构造函数
* autodetect，利用Spring的自省机制判断使用byType还是constructor装配

其中byType的使用示例如下：

*<bean id="student" class="com.fys.spring.basic.Student" autowire="byType">*

*<property name="name" value="tom"/>*

*</bean>*

# 基于注解的自动装配

在Java代码中，可以使用注解的方式进行装配，常用注解:

* @Resource，默认使用byName进行装配
* @Autowired，默认使用byType进行装配

使用示例如下：

|  |  |
| --- | --- |
| *@Getter*  *@Component("teacher")*  *public class Teacher {*  *@Value("Jerry-2")*  *private String name;*  *}* | *@Getter*  *@Component("student")*  *public class Student {*  *@Value("Tom")*  *private String name;*  *@Resource*  *private Teacher teacher;*  *}* |

# 注入的方式

以上是Bean自动装配的两种配置（使用方式），但是注入的位置是不同的

1. 构造器注入

|  |  |
| --- | --- |
| XML | 基于注解 |
| *autowire=”constructor”，//方式1*  *<bean id="foo" class="x.y.Foo">*  *<constructor-arg ref="bar"/>*  *<constructor-arg ref="baz"/>*  *</bean> //方式2* | *private Teacher teacher;*  *@Autowired*  *public Student(Teacher teacher) {*  *this.teacher = teacher;*  *}* |

1. Setter方法注入（属性注入）

|  |  |
| --- | --- |
| XML | 基于注解 |
| *<bean class="LogonService">*  *<property name="logDao" ref="logDao"/>*  *<property name="userDao" ref="userDao"/>*  *</bean>* | *@Autowired*  *public void setLogDao(LogDao logDao) {*  *this.logDao = logDao;*  *}* |

1. 工厂方法注入

定义工厂类来实例化Bean

|  |  |
| --- | --- |
| *@Getter*  *@Setter*  *public class Person {*  *private String name = "Anonymous";*  *}* | *public class PersonFactory implements*  *FactoryBean<Person> {*  *@Nullable*  *public Person getObject() throws Exception {*  *return new Person();*  *}*  *@Nullable*  *public Class<?> getObjectType() {*  *return Person.class;*  *}*  *public boolean isSingleton() {*  *return false;*  *}*  *}* |

定义配置类

*@Bean*

*public PersonFactory createRunnableFactory() {*

*return new PersonFactory();*

*}*

测试类如下：

*AnnotationConfigApplicationContext context = new*

*AnnotationConfigApplicationContext(BeanConfiguration.class);*

*PersonFactory factory = context.getBean(PersonFactory.class);*

*String name = factory.getObject().getName();*

工厂类负责创建一个或者多个而目标类实例，工厂类方法一般以接口或者抽象变量的形式返回目标类实例。

https://blog.csdn.net/sinat\_34596644/article/details/53080026

https://www.jianshu.com/p/f51bdde15191

http://www.cnblogs.com/xdp-gacl/p/4249939.html