传统的Spring做法是使用.xml文件来对bean进行注入或者是配置aop、事物，这么做有两个缺点：  
1、如果所有的内容都配置在.xml文件中，那么.xml文件将会十分庞大；如果按需求分开.xml文件，那么.xml文件又会非常多。总之这将导致配置文件的可读性与可维护性变得很低。  
2、在开发中在.java文件和.xml文件之间不断切换，是一件麻烦的事，同时这种思维上的不连贯也会降低开发的效率。  
为了解决这两个问题，Spring引入了注解，通过"@XXX"的方式，让注解与Java Bean紧密结合，既大大减少了配置文件的体积，又增加了Java Bean的可读性与内聚性。

不使用注解：

先看一个不使用注解的Spring示例，在这个示例的基础上，改成注解版本的，这样也能看出使用与不使用注解之间的区别，先定义一个老虎：

[IMG_256](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

public class Tiger {

private String tigerName="TigerKing";

public String toString(){

return "TigerName:"+tigerName;

}

}

[IMG_257](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

再定义一个猴子：

[IMG_258](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

public class Monkey {

private String monkeyName = "MonkeyKing";

public String toString(){

return "MonkeyName:" + monkeyName;

}

}

[IMG_259](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

定义一个动物园：

[IMG_260](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

public class Zoo {

private Tiger tiger;

private Monkey monkey;

public Tiger getTiger() {

return tiger;

}

public void setTiger(Tiger tiger) {

this.tiger = tiger;

}

public Monkey getMonkey() {

return monkey;

}

public void setMonkey(Monkey monkey) {

this.monkey = monkey;

}

public String toString(){

return tiger + "\n" + monkey;

}

}

[IMG_261](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

spring的配置文件这么写：

[IMG_262](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

<?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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd

">

<bean id="zoo" class="com.spring.model.Zoo" >

<property name="tiger" ref="tiger" />

<property name="monkey" ref="monkey" />

</bean>

<bean id="tiger" class="com.spring.model.Tiger" />

<bean id="monkey" class="com.spring.model.Monkey" />

</beans>

[IMG_263](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

测试方法：

[IMG_264](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

public class TestAnnotation {

/\*\*

\* 不使用注解

\*/

@Test

public void test(){

//读取配置文件

ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext2.xml");

Zoo zoo=(Zoo) ctx.getBean("zoo");

System.out.println(zoo.toString());

}

}

[IMG_265](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

都很熟悉，权当复习一遍了。

**1、@Autowired**

@Autowired顾名思义，就是自动装配，其作用是为了消除代码Java代码里面的getter/setter与bean属性中的property。当然，getter看个人需求，如果私有属性需要对外提供的话，应当予以保留。

@Autowired默认按类型匹配的方式，在容器查找匹配的Bean，当**有且仅有一个匹配的Bean**时，Spring将其注入@Autowired标注的变量中。

因此，引入@Autowired注解，先看一下spring配置文件怎么写：

[IMG_266](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

1 <?xml version="1.0" encoding="UTF-8"?> 2 <beans 3 xmlns="http://www.springframework.org/schema/beans" 4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" 5 xmlns:p="http://www.springframework.org/schema/p" 6 xmlns:context="http://www.springframework.org/schema/context" 7 xsi:schemaLocation="http://www.springframework.org/schema/beans 8 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd 9 http://www.springframework.org/schema/context10 http://www.springframework.org/schema/context/spring-context-3.0.xsd11 ">12 13 <context:component-scan base-package="com.spring" />14 15 <bean id="zoo" class="com.spring.model.Zoo" />16 <bean id="tiger" class="com.spring.model.Tiger" />17 <bean id="monkey" class="com.spring.model.Monkey" />18 19 </beans>

[IMG_267](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

注意第13行，使用必须告诉spring一下我要使用注解了，告诉的方式有很多，<context:component-scan base-package="xxx" />是一种最简单的，spring会自动扫描xxx路径下的注解。

看到第15行，原来zoo里面应当注入两个属性tiger、monkey，现在不需要注入了。再看下，Zoo.java也很方便，把getter/setter都可以去掉：

[IMG_268](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

import org.springframework.beans.factory.annotation.Autowired;

public class Zoo {

@Autowired

private Tiger tiger;

@Autowired

private Monkey monkey;

public String toString(){

return tiger + "\n" + monkey;

}

}

[IMG_269](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

这里@Autowired注解的意思就是，当Spring发现@Autowired注解时，将自动在代码上下文中找到和其匹配（默认是类型匹配）的Bean，并自动注入到相应的地方去。

有一个细节性的问题是，假如bean里面有两个property，Zoo.java里面又去掉了属性的getter/setter并使用@Autowired注解标注这两个属性那会怎么样？答案是Spring会按照xml优先的原则去Zoo.java中寻找这两个属性的getter/setter，导致的结果就是初始化bean报错。

OK，假设此时我把.xml文件的13行、14行两行给去掉，再运行，会抛出异常：

[IMG_270](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'zoo': Injection of autowired dependencies failed; nested exception is org.springframework.beans.factory.BeanCreationException: Could not autowire field: private com.spring.model.Tiger com.spring.model.Zoo.tiger; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No matching bean of type [com.spring.model.Tiger] found for dependency: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}

at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessPropertyValues(AutowiredAnnotationBeanPostProcessor.java:285)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapableBeanFactory.java:1074)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:517)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:456)

at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:291)

at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:222)

at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:288)

at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:190)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:580)

at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:895)

at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:425)

at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:139)

at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:83)

at com.spring.test.TestAnnotation.test(TestAnnotation.java:16)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:45)

at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:15)

at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:42)

at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:20)

at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:263)

at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:68)

at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:47)

at org.junit.runners.ParentRunner$3.run(ParentRunner.java:231)

at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:60)

at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:229)

at org.junit.runners.ParentRunner.access$000(ParentRunner.java:50)

at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:222)

at org.junit.runners.ParentRunner.run(ParentRunner.java:300)

at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:45)

at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:460)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:673)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:386)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:196)

Caused by: org.springframework.beans.factory.BeanCreationException: Could not autowire field: private com.spring.model.Tiger com.spring.model.Zoo.tiger; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No matching bean of type [com.spring.model.Tiger] found for dependency: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}

at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java:502)

at org.springframework.beans.factory.annotation.InjectionMetadata.inject(InjectionMetadata.java:84)

at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessPropertyValues(AutowiredAnnotationBeanPostProcessor.java:282)

... 36 more

Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No matching bean of type [com.spring.model.Tiger] found for dependency: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}

at org.springframework.beans.factory.support.DefaultListableBeanFactory.raiseNoSuchBeanDefinitionException(DefaultListableBeanFactory.java:920)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDependency(DefaultListableBeanFactory.java:789)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDependency(DefaultListableBeanFactory.java:703)

at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java:474)

... 38 more

[IMG_271](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

因为，@Autowired注解要去寻找的是一个Bean，Tiger和Monkey的Bean定义都给去掉了，自然就不是一个Bean了，Spring容器找不到也很好理解。那么，如果属性找不到我不想让Spring容器抛出异常，而就是显示null，可以吗？可以的，其实异常信息里面也给出了提示了，就是将@Autowired注解的required属性设置为false即可：

[IMG_272](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

import org.springframework.beans.factory.annotation.Autowired;

public class Zoo {

@Autowired(required=false)

private Tiger tiger;

@Autowired(required=false)

private Monkey monkey;

public String toString(){

return tiger + "\n" + monkey;

}

}

[IMG_273](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

此时，找不到tiger、monkey两个属性，Spring容器不再抛出异常而是认为这两个属性为null。

**2、Qualifier（指定注入Bean的名称）**

如果容器中有一个以上匹配的Bean，则可以通过@Qualifier注解限定Bean的名称，看下面的例子：

定义一个Car接口：

package com.spring.service;

public interface ICar {

public String getCarName();

}

两个实现类BMWCar和BenzCar：

[IMG_274](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.service.impl;

import com.spring.service.ICar;

public class BMWCar implements ICar{

public String getCarName(){

return "BMW car";

}

}

[IMG_275](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

[IMG_276](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.service.impl;

import com.spring.service.ICar;

public class BenzCar implements ICar{

public String getCarName(){

return "Benz car";

}

}

[IMG_277](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

再写一个CarFactory，引用car（这里先不用@Qualifier注解）：

[IMG_278](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

import org.springframework.beans.factory.annotation.Autowired;

import com.spring.service.ICar;

public class CarFactory {

@Autowired

private ICar car;

public String toString(){

return car.getCarName();

}

}

[IMG_279](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

配置文件：

[IMG_280](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

<?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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd

">

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

<!-- Autowired注解配合Qualifier注解 -->

<bean id="carFactory" class="com.spring.model.CarFactory" />

<bean id="bmwCar" class="com.spring.service.impl.BMWCar" />

<bean id="benz" class="com.spring.service.impl.BenzCar" />

</beans>

[IMG_281](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

测试方法：

[IMG_282](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

/\*\*

\* Autowired注解配合Qualifier注解

\*/

@Testpublic void test1(){

//读取配置文件

ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext2.xml");

CarFactory carFactory=(CarFactory) ctx.getBean("carFactory");

System.out.println(carFactory.toString());

}

[IMG_283](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

运行一下，不用说，一定是报错的，Car接口有两个实现类，Spring并不知道应当引用哪个实现类。

[IMG_284](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'carFactory': Injection of autowired dependencies failed; nested exception is org.springframework.beans.factory.BeanCreationException:   
Could not autowire field: private com.spring.service.ICar com.spring.model.CarFactory.car; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException:  
No unique bean of type [com.spring.service.ICar] is defined: expected single matching bean but found 2: [bmwCar, benz] at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessPropertyValues(AutowiredAnnotationBeanPostProcessor.java:285)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapableBeanFactory.java:1074)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:517)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:456)

at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:291)

at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:222)

at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:288)

at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:190)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:580)

at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:895)

at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:425)

at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:139)

at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:83)

at com.spring.test.TestAnnotation.test1(TestAnnotation.java:25)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:45)

at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:15)

at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:42)

at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:20)

at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:263)

at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:68)

at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:47)

at org.junit.runners.ParentRunner$3.run(ParentRunner.java:231)

at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:60)

at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:229)

at org.junit.runners.ParentRunner.access$000(ParentRunner.java:50)

at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:222)

at org.junit.runners.ParentRunner.run(ParentRunner.java:300)

at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:45)

at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:460)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:673)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:386)

at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:196)

Caused by: org.springframework.beans.factory.BeanCreationException: Could not autowire field: private com.spring.service.ICar com.spring.model.CarFactory.car; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No unique bean of type [com.spring.service.ICar] is defined: expected single matching bean but found 2: [bmwCar, benz]

at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java:502)

at org.springframework.beans.factory.annotation.InjectionMetadata.inject(InjectionMetadata.java:84)

at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessPropertyValues(AutowiredAnnotationBeanPostProcessor.java:282)

... 36 more

Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No unique bean of type [com.spring.service.ICar] is defined: expected single matching bean but found 2: [bmwCar, benz]

at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDependency(DefaultListableBeanFactory.java:796)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDependency(DefaultListableBeanFactory.java:703)

at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java:474)

... 38 more

[IMG_285](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

出现这种情况通常有两种解决办法：  
(1)、在配置文件中删除其中一个实现类，Spring会自动去base-package下寻找Car接口的实现类，发现Car接口只有一个实现类，便会直接引用这个实现类。  
(2)、实现类就是有多个该怎么办？此时可以使用@Qualifier注解来指定Bean的名称：

[IMG_286](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

import org.springframework.beans.factory.annotation.Autowired;import org.springframework.beans.factory.annotation.Qualifier;

import com.spring.service.ICar;

public class CarFactory {

@Autowired

@Qualifier("bmwCar")

private ICar car;

public String toString(){

return car.getCarName();

}

}

[IMG_287](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

此处会注入名为"bmwCar"的Bean。

**3、Resource**

@Resource注解与@Autowired注解作用非常相似，这个就简单说了，看例子：

[IMG_288](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

import javax.annotation.Resource;

public class Zoo1 {

@Resource(name="tiger")

private Tiger tiger;

@Resource(type=Monkey.class)

private Monkey monkey;

public String toString(){

return tiger + "\n" + monkey;

}

}

[IMG_289](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

这是详细一些的用法，说一下@Resource的装配顺序：  
(1)、@Resource后面没有任何内容，默认通过name属性去匹配bean，找不到再按type去匹配  
(2)、指定了name或者type则根据指定的类型去匹配bean  
(3)、指定了name和type则根据指定的name和type去匹配bean，任何一个不匹配都将报错

然后，区分一下@Autowired和@Resource两个注解的区别：  
(1)、@Autowired默认按照byType方式进行bean匹配，@Resource默认按照byName方式进行bean匹配  
(2)、@Autowired是Spring的注解，@Resource是J2EE的注解，这个看一下导入注解的时候这两个注解的包名就一清二楚了  
Spring属于第三方的，J2EE是Java自己的东西，因此，建议使用@Resource注解，以减少代码和Spring之间的耦合。

**4、Service**

上面这个例子，还可以继续简化，因为spring的配置文件里面还有15行~17行三个bean，下一步的简化是把这三个bean也给去掉，使得spring配置文件里面只有一个自动扫描的标签，增强Java代码的内聚性并进一步减少配置文件。

要继续简化，可以使用@Service。先看一下配置文件，当然是全部删除了：

[IMG_290](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

<?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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd

">

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

</beans>

[IMG_291](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

是不是感觉很爽？起码我觉得是的。OK，下面以Zoo.java为例，其余的Monkey.java和Tiger.java都一样：

[IMG_292](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

import org.springframework.beans.factory.annotation.Autowired;import org.springframework.stereotype.Service;

@Servicepublic class Zoo {

@Autowired

private Tiger tiger;

@Autowired

private Monkey monkey;

public String toString(){

return tiger + "\n" + monkey;

}

}

[IMG_293](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

这样，Zoo.java在Spring容器中存在的形式就是"zoo"，即可以通过ApplicationContext的getBean("zoo")方法来得到Zoo.java。@Service注解，其实做了两件事情：  
(1)、声明Zoo.java是一个bean，这点很重要，因为Zoo.java是一个bean，其他的类才可以使用@Autowired将Zoo作为一个成员变量自动注入。  
(2)、Zoo.java在bean中的id是"zoo"，即类名且首字母小写。

如果，我不想用这种形式怎么办，就想让Zoo.java在Spring容器中的名字叫做"Zoo"，可以的：

[IMG_294](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

package com.spring.model;

import org.springframework.beans.factory.annotation.Autowired;import org.springframework.context.annotation.Scope;import org.springframework.stereotype.Service;

@Service("Zoo")

@Scope("prototype")public class Zoo {

@Autowired

private Tiger tiger;

@Autowired

private Monkey monkey;

public String toString(){

return tiger + "\n" + monkey;

}

}

[IMG_295](http://www.cnblogs.com/xiaoxi/p/javascript:void(0);)

这样，就可以通过ApplicationContext的getBean("Zoo")方法来得到Zoo.java了。

这里我还多加了一个@Scope注解，应该很好理解。因为Spring默认产生的bean是单例的，假如我不想使用单例怎么办，xml文件里面可以在bean里面配置scope属性。注解也是一样，配置@Scope即可，默认是"singleton"即单例，"prototype"表示原型即每次都会new一个新的出来。

————————————————————————————————————————————————————————————————————

**使用注解来构造IoC容器**  
**用注解来向Spring容器注册Bean。需要在applicationContext.xml中注册<context:component-scan base-package=”pagkage1[,pagkage2,…,pagkageN]”/>。**

如：在base-package指明一个包

<context:component-scan base-package="cn.gacl.java"/>

表明cn.gacl.java包及其子包中，如果某个类的头上带有特定的注解【@Component/@Repository/@Service/@Controller】，就会将这个对象作为Bean注册进Spring容器。也可以在<context:component-scan base-package=” ”/>中指定多个包，如：

<context:component-scan base-package="cn.gacl.dao.impl,cn.gacl.service.impl,cn.gacl.action"/>

多个包逗号隔开。

**1、@Component**  
@Component是所有受Spring 管理组件的通用形式，@Component注解可以放在类的头上，@Component不推荐使用。

**2、@Controller**  
@Controller对应表现层的Bean，也就是Action，例如：

@Controller

@Scope("prototype")public class UserAction extends BaseAction<User>{

……

}

**使用@Controller注解标识UserAction之后，就表示要把UserAction交给Spring容器管理，在Spring容器中会存在一个名字为"userAction"的action，这个名字是根据UserAction类名来取的。注意：如果**@Controller**不指定其**value【@Controller】，**则默认的bean名字为这个类的类名首字母小写，如果指定**value【@Controller(value="UserAction")】**或者**【@Controller("UserAction")】，**则使用value作为bean的名字**。

这里的UserAction还使用了@Scope注解，@Scope("prototype")表示将Action的范围声明为原型，**可以利用容器的scope="prototype"来保证每一个请求有一个单独的Action来处理，避免struts中Action的线程安全问题。spring** **默认scope** **是单例模式(scope="singleton")，这样只会创建一个Action对象，每次访问都是同一Action对象，数据不安全，struts2** **是要求每次次访问都对应不同的Action，scope="prototype"** **可以保证当有请求的时候都创建一个Action对象。**

**3、@Service**@Service对应的是业务层Bean，例如：

@Service("userService")public class UserServiceImpl implements UserService {

………

}

 @Service("userService")注解是告诉Spring，当Spring要创建UserServiceImpl的的实例时，bean的名字必须叫做"userService"，这样当Action需要使用UserServiceImpl的的实例时,就可以由Spring创建好的"userService"，然后注入给Action：在Action只需要声明一个名字叫"userService"的变量来接收由Spring注入的"userService"即可，具体代码如下：

// 注入userService

@Resource(name = "userService")private UserService userService;

注意：在Action声明的"userService"变量的类型必须是"UserServiceImpl"或者是其父类"UserService"，否则由于类型不一致而无法注入，由于Action中的声明的"userService"变量使用了@Resource注解去标注，并且指明了其name = "userService"，这就等于告诉Spring，说我Action要实例化一个"userService"，你Spring快点帮我实例化好，然后给我，当Spring看到userService变量上的@Resource的注解时，根据其指明的name属性可以知道，Action中需要用到一个UserServiceImpl的实例，此时Spring就会把自己创建好的名字叫做"userService"的UserServiceImpl的实例注入给Action中的"userService"变量，帮助Action完成userService的实例化，这样在Action中就不用通过"UserService userService = new UserServiceImpl();"这种最原始的方式去实例化userService了。如果没有Spring，那么当Action需要使用UserServiceImpl时，必须通过"UserService userService = new UserServiceImpl();"主动去创建实例对象，但使用了Spring之后，Action要使用UserServiceImpl时，就不用主动去创建UserServiceImpl的实例了，创建UserServiceImpl实例已经交给Spring来做了，Spring把创建好的UserServiceImpl实例给Action，Action拿到就可以直接用了。Action由原来的主动创建UserServiceImpl实例后就可以马上使用，变成了被动等待由Spring创建好UserServiceImpl实例之后再注入给Action，Action才能够使用。这说明Action对"UserServiceImpl"类的“控制权”已经被“反转”了，原来主动权在自己手上，自己要使用"UserServiceImpl"类的实例，自己主动去new一个出来马上就可以使用了，但现在自己不能主动去new "UserServiceImpl"类的实例，new "UserServiceImpl"类的实例的权力已经被Spring拿走了，只有Spring才能够new "UserServiceImpl"类的实例，而Action只能等Spring创建好"UserServiceImpl"类的实例后，再“恳求”Spring把创建好的"UserServiceImpl"类的实例给他，这样他才能够使用"UserServiceImpl"，这就是Spring核心思想“控制反转”，也叫“依赖注入”，“依赖注入”也很好理解，Action需要使用UserServiceImpl干活，那么就是对UserServiceImpl产生了依赖，Spring把Acion需要依赖的UserServiceImpl注入(也就是“给”)给Action，这就是所谓的“依赖注入”。对Action而言，Action依赖什么东西，就请求Spring注入给他，对Spring而言，Action需要什么，Spring就主动注入给他。

**4、@ Repository**  
@Repository对应数据访问层Bean ，例如：

@Repository(value="userDao")public class UserDaoImpl extends BaseDaoImpl<User> {

………

}

@Repository(value="userDao")注解是告诉Spring，让Spring创建一个名字叫"userDao"的UserDaoImpl实例。

当Service需要使用Spring创建的名字叫"userDao"的UserDaoImpl实例时，就可以使用@Resource(name = "userDao")注解告诉Spring，Spring把创建好的userDao注入给Service即可。

// 注入userDao，从数据库中根据用户Id取出指定用户时需要用到

@Resource(name = "userDao")private BaseDao<User> userDao;

—————————————————————————————————————————————————————————————————————

**Spring常用注解汇总**   
本文汇总了Spring的常用注解，以方便大家查询和使用，具体如下：

使用注解之前要开启自动扫描功能，其中base-package为需要扫描的包(含子包)。

<context:component-scan base-package="cn.test"/>

@Configuration把一个类作为一个IoC容器，它的某个方法头上如果注册了@Bean，就会作为这个Spring容器中的Bean。  
@Scope注解 作用域  
@Lazy(true) 表示延迟初始化  
@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。  
@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。  
@Scope用于指定scope作用域的（用在类上）  
@PostConstruct用于指定初始化方法（用在方法上）  
@PreDestory用于指定销毁方法（用在方法上）  
@DependsOn：定义Bean初始化及销毁时的顺序  
@Primary：自动装配时当出现多个Bean候选者时，被注解为@Primary的Bean将作为首选者，否则将抛出异常  
@Autowired 默认按类型装配，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") 存在多个实例配合使用  
@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。  
@PostConstruct 初始化注解  
@PreDestroy 摧毁注解 默认 单例  启动就加载  
@Async异步方法调用