

---

## 知识点列表

编号	名称	描述	级别
1	Spring 的作用及其好处	了解 Spring 框架的作用	**
2	Spring 容器实例化	掌握 2 中实例化的方式	**
3	Spring 容器对 Bean 组件的管理	通过案例掌握管理 Bean 组件的方式,理解常用属性。	**
4	DI 依赖注入	理解依赖注入的概念,掌握两种注入方式,了解集合的注入。	**
5	组件自动扫描功能	注解的使用较流行,理解并掌握常用的几种。	**

注:    \*\*"理解级别    \*\*\*"掌握级别    \*\*\*\*"应用级别

---

# 目录

1. Spring 的作用及优势 *	3
【案例 1】Spring HelloWorld **	3
2. Spring 使用基础 **	12
2.1. Spring 容器实例化 **	12
【案例 2】Spring 容器实例化 **	12
2.2. Spring 容器对 Bean 组件的管理 **	13
【案例 3】Spring 容器对 Bean 的管理 **	14
2.3. DI 依赖注入 **	24
【案例 4】DI 依赖注入 **	24
3. 注解方式配置 **	35
3.1. 组件自动扫描功能 **	35
【案例 5】注解方式配置 **	36

---

## 1. Spring 的作用及优势 \*

Spring 用于整合，好处是解耦。

解耦，可以降低组件与组件之间的关联，改善程序结构，便于系统的维护和扩展。

我们在使用 Spring 框架时，主要是使用 Spring 容器的两个特性：IoC 和 AOP。

IoC 全称 Inverse of Control ( 反向控制或控制反转 )。

在类和类之间存在控制权，控制权指的是对象的创建和使用，

比如有类 A 和类 B，我们之前的做法是在 A 中调用 B，那么控制权就在 A 中，这样做的耦合度较高，如果修改了 B，A 也要做相应修改。

引入 Spring 框架后，控制权由 spring 容器来负责。当 A 想使用 B 时，需要由 Spring 容器通过配置文件进行注入。这种思想就是 IoC ( 为了更好的理解，我们可以这样认为，对象创建和使用的控制权转移到了 Spring 容器，由 Spring 容器来控制 )。

AOP 为 Aspect Oriented Programming 的缩写，意为：面向切面编程 ( 也叫面向方面 )，可以通过预编译方式和运行期动态代理实现在不修改源代码的前提下给程序动态统一添加功能的一种技术。

Struts2 中的拦截器，就是使用 AOP 的思想。使用 AOP 思想编写程序，会是程序更加灵活。

一般而言，使用 Spring 框架的主要作用：

我们会使用 IoC 整合组件 ( 各种 Bean )，使用 AOP 来管理事务。

和 Hibernate 相同，Spring 的使用也没有限制，到底是用于 Web 工程还是普通 Java 程序。

### 【案例 1】Spring HelloWorld \*\*

#### 1) 新建工程 Spring1

在之前我们的程序开发中，在还没有学习“解耦”思想之前，调用一个组件的过程，我们都这样写：

#### 2) 调用组件 ( 最初的方式 )

##### a. HelloBean

```
HelloBean.java X UseBean.java Test.java
1 package tarena.demol;
2
3 public class HelloBean {
4
5     public void sayHello(){
6         System.out.println("你好，宝!");
7     }
8 }
9
```

#### b. UseBean

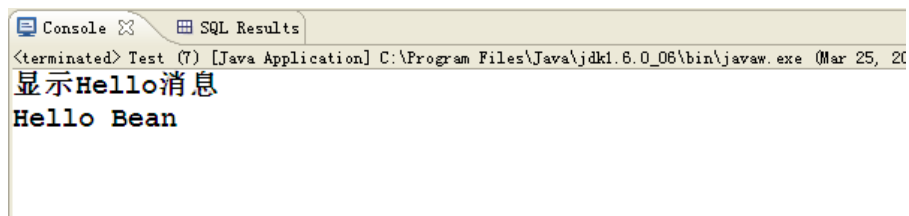
```
HelloBean.java UseBean.java X Test.java
1 package tarena.demol;
2 public class UseBean {
3
4     public void show() {
5         System.out.println("显示Hello消息");
6         HelloBean hello = new HelloBean();
7         hello.sayHello();
8     }
9 }
10
```

#### c. Test

```
HelloBean.java UseBean.java Test.java X
1 package tarena.demol;
2
3 public class Test {
4     public static void main(String[] args) {
5         UseBean bean = new UseBean();
6         bean.show();
7     }
8 }
9
```

#### d. 运行 Test

实现了调用组建的功能



The screenshot shows a Java IDE's console window. At the top, there are tabs for 'Console' and 'SQL Results'. The console output shows the command prompt path 'C:\Program Files\Java\jdk1.6.0\_06\bin\javaw.exe' followed by the text '显示Hello消息' (Display Hello message) and 'Hello Bean'.

```
<terminated> Test (7) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 2008)
显示Hello消息
Hello Bean
```

但是当用户需求发生改变，比如要求输出英文。

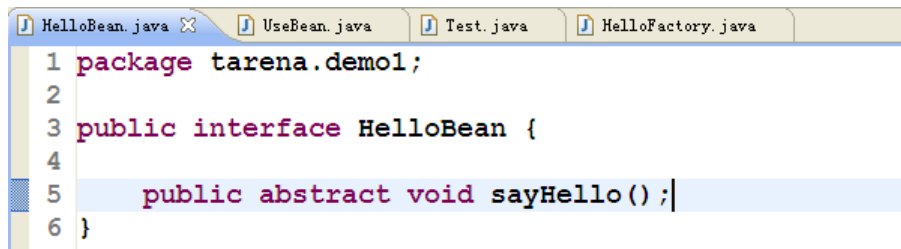
我们怎么改？修改全部源代码吗？不好。

有些问题在开发之初就要预见到，我们需要更良好的程序结构和思想。

### 3) 调用组件（更好的方式，工厂模式）

#### a. 新建 HelloBean

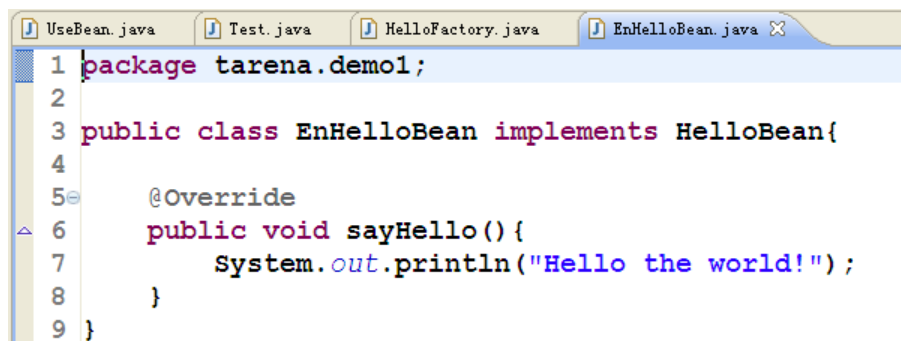
父类接口



The screenshot shows a Java IDE with several tabs: 'HelloBean.java', 'UseBean.java', 'Test.java', and 'HelloFactory.java'. The 'HelloBean.java' file is selected and shows the following code:

```
1 package tarena.demol;
2
3 public interface HelloBean {
4
5     public abstract void sayHello();
6 }
```

#### b. 新建 EnHelloBean



The screenshot shows a Java IDE with tabs: 'UseBean.java', 'Test.java', 'HelloFactory.java', and 'EnHelloBean.java'. The 'EnHelloBean.java' file is selected and shows the following code:

```
1 package tarena.demol;
2
3 public class EnHelloBean implements HelloBean{
4
5     @Override
6     public void sayHello(){
7         System.out.println("Hello the world!");
8     }
9 }
```

#### c. 新建 ZhHelloBean

```
UseBean.java | Test.java | HelloFactory.java | ZhHelloBean.java X
1 package tarena.demo1;
2
3 public class ZhHelloBean implements HelloBean {
4
5     @Override
6     public void sayHello(){
7         System.out.println("世界你好!");
8     }
9 }
10
```

#### d. 新建 HelloFatory

```
UseBean.java | Test.java | HelloFactory.java X
1 package tarena.demo1;
2
3 public class HelloFactory {
4
5     public static HelloBean getHelloBean(){
6         return new EnHelloBean();
7         //return new ZhHelloBean();
8     }
9 }
10
```

#### e. 修改 UseBean

```
UseBean.java X | Test.java
1 package tarena.demo1;
2 public class UseBean {
3
4     public void show(){
5         System.out.println("显示Hello消息");
6         HelloBean hello =
7             HelloFactory.getHelloBean();
8         hello.sayHello();
9     }
10 }
11
```

#### f. 运行 Test 进行测试 (略)

以后只需要在 HelloFactory 中修改需要创建的组件 (HelloBean 子类对象) 即可。

如此这般, 就降低了 UseBean 和 EnHelloBean (或 ZhHelloBean) 之间的耦合度。

---

而当我们使用 [Spring 框架](#)后，不需要自己写工厂，也可以实现解耦合功能。  
工厂功能由 Spring 来实现。

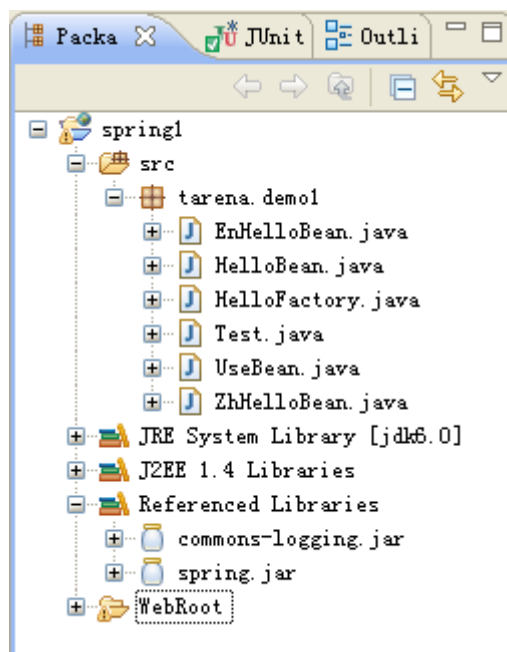
Spring 的框架及 API 帮助文档，请自行下载或到项目经理处拷贝。

文档使用 Spring 版本为 [spring-framework-2.5.6.SEC01](#)

#### 4) 调用组件 (方式 3, 使用 Spring 框架)

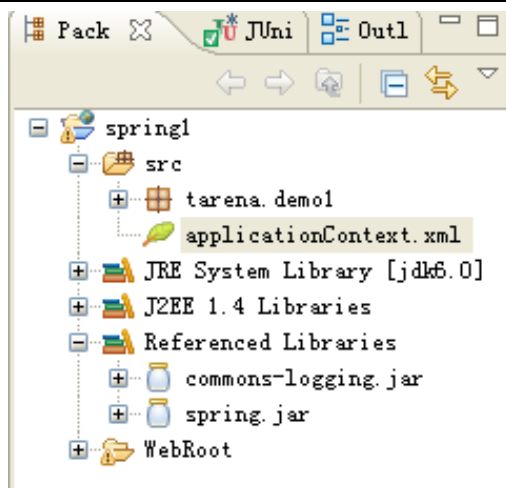
##### a. 拷贝核心 Jar 包到项目下 (共 2 个)

spring.jar 是核心的 Jar 包；该 Jar 包还需要记录日志的 commons-logging.jar 包  
请下载 [spring\\_core.zip](#)



##### b. 新建 Spring 配置文件 applicationContext.xml

固定的写法



### c. applicationContext.xml

将各种组件 ( Bean ) 纳入到 Spring 的管理中

```
<?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:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       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
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

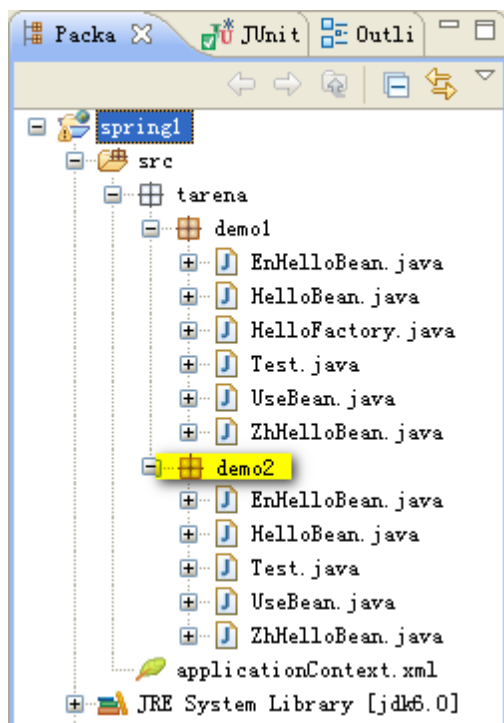
    <!--创建Bean , 有指定的id , 对应的class-->
    <bean id="usebean" class="tarena.demo2.UseBean"></bean>

    <bean id="enhellobean" class="tarena.demo2.EnHelloBean"></bean>
    <bean id="zhhellobean" class="tarena.demo2.ZhHelloBean"></bean>

</beans>
```

### d. 拷贝文件到 tarena.demo2 目录下





#### e. 修改 UseBean

将需要调用的组件 HelloBean 声明为自己的属性。

**注意：**此处不需要 new 实例化对象

```
package tarena.demo2;

public class UseBean {
    private HelloBean hello;

    public void show(){
        System.out.println("显示Hello消息");
        hello.sayHello();
    }

    public void setHello(HelloBean hello) {this.hello = hello;}
}
```

实例化对象在配置文件中进行配置，由 spring 框架完成

---

#### f. 修改 applicationContext.xml

在配置文件中，通过<property>标签的配置，将 enhellobean 通过属性注入的方式注入到 usebean 中

```
<?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:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       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
                           http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

    <bean id="usebean" class="tarena.demo2.UseBean">
        <property name="hello" ref="enhellobean"></property>
    </bean>
    <bean id="enhellobean" class="tarena.demo2.EnHelloBean"></bean>
    <bean id="zhhellobean" class="tarena.demo2.ZhHelloBean"></bean>

</beans>
```

#### g. 修改测试文件 Test

使用 Spring 框架

```
package tarena.demo2;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import tarena.demo2.UseBean;

public class Test {

    /**
     * @param args
     */
    public static void main(String[] args) {
```

```

        ApplicationContext ac =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        UseBean bean = (UseBean)ac.getBean("usebean");
        bean.show();
    }
}

```

applicationContext.xml 可以放到任意位置，比如这样写

```

public class Test {

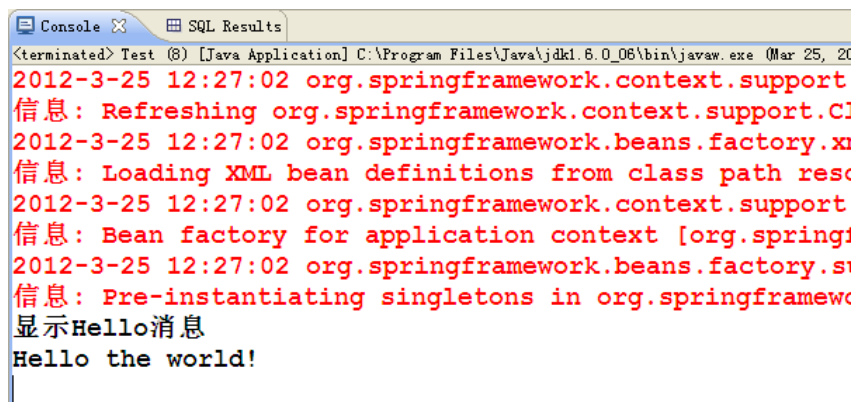
    /**
     * @param args
     */
    public static void main(String[] args) {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext(
                "tarena/demo2/applicationContext.xml");
        UseBean bean = (UseBean) ac.getBean("usebean");
        bean.show();
    }

}

```

#### h. 运行 Test

控制台打印



```

<terminated> Test (8) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 20
2012-3-25 12:27:02 org.springframework.context.support
信息: Refreshing org.springframework.context.support.Cl
2012-3-25 12:27:02 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-25 12:27:02 org.springframework.context.support
信息: Bean factory for application context [org.springf
2012-3-25 12:27:02 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
显示Hello消息
Hello the world!

```

采用 Spring 配置的方式，将来只需要修改配置文件即可，如此就实现了“解耦”。

(案例结束)

---

## 2. Spring 使用基础 \*\*

### 2.1. Spring 容器实例化 \*\*

- 1) BeanFactory  
XMLBeanFactory-->Resource--ClassPathResource\FileSystemResource
- 2) ApplicationContext(推荐)
  - ✓ ClassPathXmlApplicationContext
  - ✓ FileSystemXmlApplicationContext

### 【案例 2】Spring 容器实例化 \*\*

#### 1) 使用工程 spring1

在 Spring 当中有两个重要的对象。

容器在实例化时，我们可以使用 2 种方式加载 Spring 配置文件，创建 Spring 容器：

其中 ApplicationContext 是 BeanFactory 的子类，

在我们实例化对象时，如下两种方式的效果是等价的。

在一些文档中推荐使用 ApplicationContext，功能更强大。

推荐使用方式 1

```
13e public static void main(String[] args) {
14     //方式1
15     //ApplicationContext ac =
16     //    new ClassPathXmlApplicationContext(
17     //        "tarena/demo2/applicationContext.xml");
18
19     //方式2
20     Resource resource =
21         new ClassPathResource(CONFIGS);
22     BeanFactory ac =
23         new XmlBeanFactory(resource);
24
25     UseBean bean = (UseBean) ac.getBean("usebean");
26
27     bean.show();
28 }
29 }
```

ApplicationContext 还有一些子类，可实现不同功能：

假设我们的配置文件直接存放在硬盘某个位置,我们可以使用 FileSystemXmlApplicationContext 来找到配置文件

```
11 public class Test {
12
13     private static final String CONFIGS =
14         "tarena/demo3/applicationContext.xml";
15
16     public static void main(String[] args) {
17         //方式1
18         //ApplicationContext ac =
19         //     new FileSystemXmlApplicationContext(
20         //         "D:/applicationContext.xml");
21
22         //方式2
23         Resource resource =
24             new FileSystemResource(
25                 "D:/applicationContext.xml");
26         BeanFactory ac = new XmlBeanFactory(resource);
27
28         UseBean bean = (UseBean) ac.getBean("usebean");
29
30         bean.show();
31     }
32 }
```

(案例结束)

## 2.2. Spring 容器对 Bean 组件的管理 \*\*

### 1) Bean 对象创建的时机

默认是随着容器创建,可以使用 lazy-init=true (在调用 getBean 创建) 延迟创建  
也可以用 <beans default-lazy-init="true"/> 批量延迟创建

### 2) Bean 对象的创建模式

- ✓ 默认是单例,可以使用 scope 属性改变。
  - singleton:单例,每次调用 getBean 返回同一个
  - prototype:原型,每次调用 getBean 返回一个新的
  - request:仅限于 Web 环境
  - session:仅限于 Web 环境
  - global session:仅限于 Web 环境

### 3) Bean 对象初始化和销毁

- ✓ init-method 属性用于指定初始化方法
- ✓ destroy-method 属性用于指定销毁方法,仅适用于 singleton 模式

---

### 【案例 3】Spring 容器对 Bean 的管理 \*\*

- 1) 使用 spring1 工程
- 2) 新建 tarena.demo3

提问：Bean 对象是什么时候创建的？

让我们测试一下

#### 3) 新建组件 tarena.demo3.MyBean

为 MyBean 添加构造器

```
package tarena.demo3;

public class MyBean {
    public MyBean(){
        System.out.println("创建MyBean对象");
    }
}
```

#### 4) applicationContext.xml 中进行配置

将 MyBean 纳入 spring 的管理

```
<?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:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    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
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.5.xsd" >

    <bean id="mybean" class="tarena.demo3.MyBean"></bean>
</beans>
```

#### 5) 创建 Spring 容器实例

此时，我们只通过加载配置文件创建 Spring 容器实例

```
package tarena.demo3;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

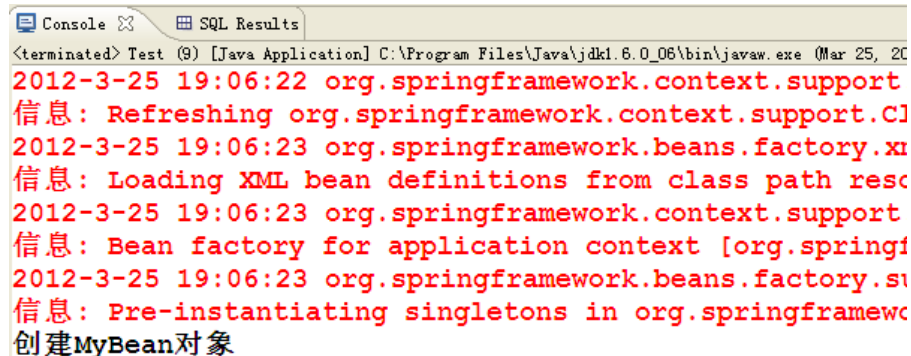
public class Test {
    private static final String[] CONFIGS =
        {"tarena/demo3/applicationContext.xml"};

    /**
     * @param args
     */
    public static void main(String[] args) {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext(CONFIGS);
    }
}
```

## 6) 运行 Test

MyBean 组件是什么时间创建的？

控制台打印结果显示，默认情况下 MyBean 在 Spring 容器被创建时就会被创建。



```
<terminated> Test (9) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 20
2012-3-25 19:06:22 org.springframework.context.support.
信息: Refreshing org.springframework.context.support.Cl
2012-3-25 19:06:23 org.springframework.beans.factory.xi
信息: Loading XML bean definitions from class path resc
2012-3-25 19:06:23 org.springframework.context.support.
信息: Bean factory for application context [org.springfi
2012-3-25 19:06:23 org.springframework.beans.factory.si
信息: Pre-instantiating singletons in org.springframework
创建MyBean对象
```

如果我们想改变对象创建的时机，该怎么做？

通过修改配置文件属性参数，可以改变 Spring 容器创建对象的时机。

## 7) 修改 applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```

```

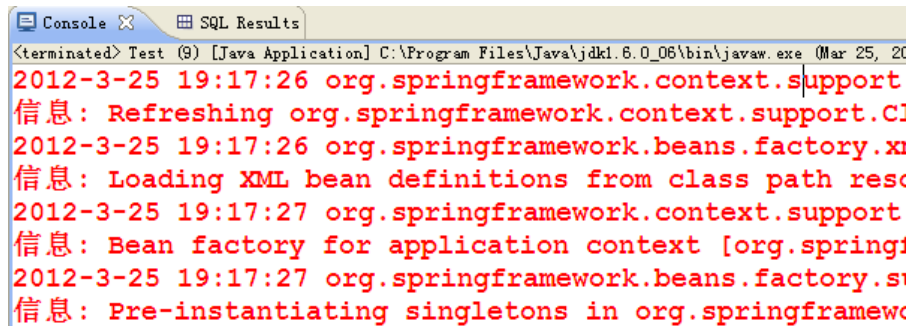
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<bean id="mybean"
    lazy-init="true"
    class="tarena.demo3.MyBean"></bean>
</beans>

```

## 8) 运行 Test

控制台什么都没有打印，对象实例化被延迟了。



```

<terminated> Test (9) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 20
2012-3-25 19:17:26 org.springframework.context.support
信息: Refreshing org.springframework.context.support.Cl
2012-3-25 19:17:26 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-25 19:17:27 org.springframework.context.support
信息: Bean factory for application context [org.springf
2012-3-25 19:17:27 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework

```

设置了<bean lazy-init="false">之后，MyBean 是什么时候被创建的？

当调用（使用）MyBean 时，MyBean 对象被创建，如下所示

## 9) 修改 Test

代码片段

```

private static final String[] CONFIGS =
    {"tarena/demo3/applicationContext.xml"};

/**

```



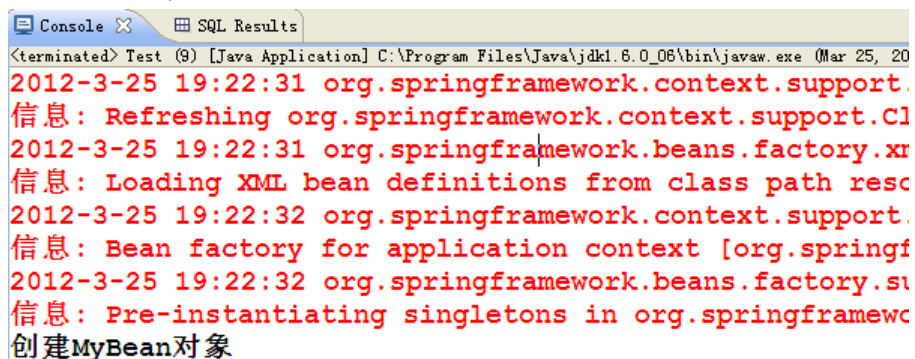
```

* @param args
*/
public static void main(String[] args) {
    ApplicationContext ac =
        new ClassPathXmlApplicationContext(CONFIGS);
    MyBean bean = (MyBean)ac.getBean("mybean");
}

```

## 10) 运行 Test

当调用/使用 MyBean 对象时，才被创建



```

<terminated> Test (9) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 20
2012-3-25 19:22:31 org.springframework.context.support.Cl
信息: Refreshing org.springframework.context.support.Cl
2012-3-25 19:22:31 org.springframework.beans.factory.xr
信息: Loading XML bean definitions from class path resc
2012-3-25 19:22:32 org.springframework.context.support.
信息: Bean factory for application context [org.springf
2012-3-25 19:22:32 org.springframework.beans.factory.si
信息: Pre-instantiating singletons in org.springframework
创建MyBean对象

```

我们还可以这样配置，让所有的 bean 都延迟创建。

## 11) 修改 applicationContext.xml

这样可以批量指定延迟加载 bean

```

<?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:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    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
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd"
    default-lazy-init="false">

    <bean id="mybean"

```

```
        class="tarena.demo3.MyBean"></bean>
</beans>
```

再提一个问题：

bean 对象的**创建模式**是什么，是单例模式创建 Bean 对象还是每次创建都是新的？

让我们进行测试

## 12) 修改 Test

创建 2 个 MyBean 对象，通过对比，

如果为 true，则表明是单例模式；如果为 false，则表明每次都创建新的 MyBean 对象

```
package tarena.demo3;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {
    private static final String[] CONFIGS =
        {"tarena/demo3/applicationContext.xml"};

    /**
     * @param args
     */
    public static void main(String[] args) {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext(CONFIGS);
        MyBean bean1 = (MyBean)ac.getBean("mybean");
        MyBean bean2 = (MyBean)ac.getBean("mybean");
        System.out.println(bean1==bean2);
    }
}
```

## 13) 运行 Test

通过运行结果，我们可以看得出，Spring 容器是通过**单例模式**创建 Bean 对象的，也就是说，**默认**情况下，通过调用 ac.getBean("mybean")方法获得的对象都是同一个 mybean 对象

```
Console SQL Results
<terminated> Test (9) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 2012-3-25 19:38:00 org.springframework.context.support
信息: Refreshing org.springframework.context.support.Cl
2012-3-25 19:38:00 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-25 19:38:01 org.springframework.context.support
信息: Bean factory for application context [org.springf
2012-3-25 19:38:01 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
创建MyBean对象
true
```

使用单例模式有风险，风险在于多线程并发访问时会有一些状况。

那么如何取消容器默认单例模式创建对象？

#### 14) 修改 applicationContext.xml

设置创建 bean 的模式为原型模式 ( prototype ) 即可以

代码片段

```
<bean id="mybean"
    lazy-init="true"
    scope="prototype"
    class="tarena.demo3.MyBean"></bean>
```

#### 15) 运行 Test

注意：调用了 2 次 MyBean 的构造方法，说明创建了 2 个对象

```
Console SQL Results
<terminated> Test (9) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 2012-3-25 19:56:20 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C
2012-3-25 19:56:20 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-25 19:56:20 org.springframework.context.support
信息: Bean factory for application context [org.springf
2012-3-25 19:56:20 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
创建MyBean对象
创建MyBean对象
false
```

---

### scope 属性的取值

在 web ( 仅限于 web 项目 ) 环境中 , 还可以设置所创建的 bean 对象的生命周期和 request、session

- ✓ request                      表示 bean 对象生命周期和 request 生命周期相同
- ✓ session                      同 session
- ✓ global session              相当于 application
- ✓ single
- ✓ prototype

### Bean 对象的初始化和销毁

init-method 属性用于指定初始化方法

destroy-method 属性用于指定销毁方法 , 仅适用于 singleton 模式

## 16) 修改 MyBean

加入方法 myinit()

方法 mydestory()

```
package tarena.demo3;

public class MyBean {

    public MyBean(){
        System.out.println("创建MyBean对象");
    }

    public void myinit(){
        System.out.println("执行MyBean对象的初始化！");
    }

    public void mydestory(){
        System.out.println("执行MyBean对象的销毁！释放资源！");
    }

}
```

## 17) 修改 applicationContext.xml

希望在 bean 对象创建后自动调用 myinit()方法

代码片段

```
<bean id="mybean"
      lazy-init="true"
```

```
scope="prototype"
init-method="myinit"
class="tarena.demo3.MyBean"></bean>
```

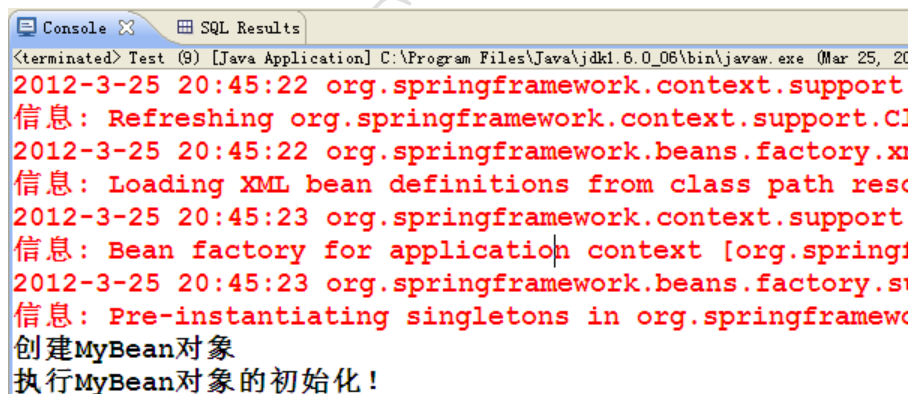
## 18) 修改 Test

```
public class Test {
    private static final String[] CONFIGS =
        {"tarena/demo3/applicationContext.xml"};

    /**
     * @param args
     */
    public static void main(String[] args) {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext(CONFIGS);
        MyBean bean = (MyBean)ac.getBean("mybean");
    }
}
```

## 19) 执行 Test

自定义的初始化的方法在对象被创建后调用



```
<terminated> Test (9) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 2012)
2012-3-25 20:45:22 org.springframework.context.support.Cl
信息: Refreshing org.springframework.context.support.Cl
2012-3-25 20:45:22 org.springframework.beans.factory.xi
信息: Loading XML bean definitions from class path res
2012-3-25 20:45:23 org.springframework.context.support
信息: Bean factory for application context [org.springf
2012-3-25 20:45:23 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
创建MyBean对象
执行MyBean对象的初始化!
```

## 20) 修改 applicationContext.xml

希望在 bean 对象销毁前自动调用 mydestory()方法

代码片段

```
<bean id="mybean"
    lazy-init="true"
    scope="prototype"
```

```
init-method="myinit"  
destroy-method="mydestroy"  
class="tarena.demo3.MyBean">  
</bean>
```

如果想看演示结果，我们需要知道 bean 对象在什么时候被销毁的。

那么 bean 对象在什么时候被销毁呢？

bean 对象在 spring 容器关闭的时候被销毁。

为了看到结果，我们需要做一些修改

## 21) 修改 Test

```
public class Test {  
    private static final String[] CONFIGS =  
        {"tarena/demo3/applicationContext.xml"};  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        AbstractApplicationContext ac =  
            new ClassPathXmlApplicationContext(CONFIGS);  
        MyBean bean = (MyBean)ac.getBean("mybean");  
        ac.close();  
    }  
}
```

## 22) 运行 Test

```
Console X SQL Results
<terminated> Test (9) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 2012)
2012-3-25 20:55:28 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C:
2012-3-25 20:55:28 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res:
2012-3-25 20:55:28 org.springframework.context.support
信息: Bean factory for application context [org.spring:
2012-3-25 20:55:28 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
创建MyBean对象
执行MyBean对象的初始化!
2012-3-25 20:55:28 org.springframework.context.support
信息: Closing org.springframework.context.support.Clas:
2012-3-25 20:55:28 org.springframework.beans.factory.s
信息: Destroying singletons in org.springframework.bea
```

我们发现没有打印预期的执行了 destroy()方法的结果。

原因是因为在 applicationContext.xml 文件中设置的 destroy-method=""属性仅仅对单例模式起作用，在 prototype 模式下没有意义。

### 23) 修改 applicationContext.xml

```
<bean id="mybean"
    lazy-init="true"
    scope="singleton"
    init-method="myinit"
    destroy-method="mydestroy"
    class="tarena.demo3.MyBean">
</bean>
```

### 24) 运行 Test

```
Console SQL Results
<terminated> Test (9) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 25, 2012-3-25 20:58:48 org.springframework.context.support
信息: Refreshing org.springframework.context.support.Cl
2012-3-25 20:58:48 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2012-3-25 20:58:48 org.springframework.context.support
信息: Bean factory for application context [org.spring
2012-3-25 20:58:48 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframework
创建MyBean对象
执行MyBean对象的初始化！
2012-3-25 20:58:48 org.springframework.context.support
信息: Closing org.springframework.context.support.Clas
2012-3-25 20:58:48 org.springframework.beans.factory.s
信息: Destroying singletons in org.springframework.bea
执行MyBean对象的销毁！释放资源！
```

(案例结束)

## 2.3. DI 依赖注入 \*\*

DI ( 依赖注入 ) 是 IoC 实现的重要技术，有如下 2 中方式：

- 1) setter 方式注入
- 2) 构造方式注入

注入类型有如下几种：简单值、集合、bean 对象

Ioc 和 DI 的关系？

我们认为 Spring 是具有 IoC 特性的框架。

实现 IoC 是由 Spring 容器来完成的，Spring 容器通过依赖注入 DI 建立起对象（组件、Bean）之间的关系。

我们可以这样理解：DI 是 IoC 实现的一种手段，Ioc 通过 DI 来实现。

### 【案例 4】DI 依赖注入 \*\*

像这样的 bean 组件是怎么注入的？



```

9      <bean id="usebean" class="tarena.demo2.UseBean">
10          <property name="hello" ref="enhellobean">
11          </property>
12      </bean>
13      <bean id="enhellobean" class="tarena.demo2.EnHelloB
14      <bean id="zhhellobean" class="tarena.demo2.ZhHelloB
15

```

引入 Spring 中一个重要概念 DI ( 依赖注入 )

DI ( 依赖注入 ) 有 2 种方式 :

- 1) setter 方式注入  
我们在之前的案例中使用的方式 ( 推荐使用 )
- 2) 构造方式注入

演示如下

- 1) 使用 spring1 工程
- 2) 新建 tarena.demo4

我们想在 bean 对象 A 中使用 bean 对象 B  
如果这样写，好吗？

```

3 public class A {
4     private B b;
5
6     public void print(){
7         System.out.println("----打印输出----");
8     }
9 }
10

```

不好。这样写和不使用 Spring 框架没区别。类 A 和类 B 之间的耦合度太高。  
所以，我们要这样写

方式 1 : set 方法注入

- 3) 新建 IB

```
applicationContext.xml  A.java  B.java  Test.java  IB.java X
1 package tarena.demo4;
2
3 public interface IB {
4
5     public abstract void show();
6
7 }
```

#### 4) 新建 B

```
applicationContext.xml  A.java  B.java X  Test.java  IB.java
1 package tarena.demo4;
2
3 public class B implements IB{
4
5     @Override
6     public void show(){
7         System.out.println("Hello!");
8     }
9 }
10
```

#### 5) 新建 A

```
applicationContext.xml  A.java X  B.java  Test.java
1 package tarena.demo4;
2
3 public class A {
4     private IB b;
5
6     public void print(){
7         System.out.println("----打印输出----");
8         b.show();
9     }
10
11     public void setB(IB b) {this.b = b;}
12
13 }
```

#### 6) 修改 applicationContext.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schem
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema
4       xmlns:context="http://www.springframework.o
5       xmlns:tx="http://www.springframework.org/s
6       xsi:schemaLocation="http://www.springframe
7           http://www.springframework.org/sch
8           http://www.springframework.org/sch
9       default-lazy-init="true">
10
11     <bean id="a" class="tarena.demo4.A">
12       <property name="b" ref="b"></property>
13     </bean>
14
15     <bean id="b" class="tarena.demo4.B"></bean>
16
17 </beans>
18

```

## 7) 新建 Test

```

1 package tarena.demo4;
2
3 import org.springframework.context.ApplicationConte
4 import org.springframework.context.support.ClassPat
5
6 public class Test {
7     private static final String[] CONFIGS =
8         {"tarena/demo4/applicationContext.x
9
10    public static void main(String[] args) {
11        ApplicationContext ac =
12            new ClassPathXmlApplicationContext(CONF
13
14        A a = (A) ac.getBean("a");
15        a.print();
16    }
17 }
18

```

如上是 set 方式注入，接下里演示构造方式注入

## 8) 修改 A

添加构造器，不需要属性的 set 方法

```

1 package tarena.demo4;
2
3 public class A {
4     private IB b;
5
6     public A(IB b){this.b = b;}
7
8     public void print(){
9         System.out.println("----打印输出----");
10        b.show();
11    }
12
13    //public void setB(IB b) {this.b = b;}
14 }
15

```

#### 9) 修改 applicationContext.xml

```

8         http://www.springframework.org/sche
9         default-lazy-init="true">
10
11        <bean id="a" class="tarena.demo4.A">
12            <!-- property name="b" ref="b"></property -->
13            <constructor-arg index="0" ref="b">
14                </constructor-arg>
15        </bean>
16
17        <bean id="b" class="tarena.demo4.B"></bean>
18
19 </beans>
20

```

#### 10) 运行 Test

```

Console  SQL Results
<terminated> Test (10) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe 0
2012-3-25 21:43:32 org.springframework.context.support
信息: Refreshing org.springframework.context.support
2012-3-25 21:43:32 org.springframework.beans.factory
信息: Loading XML bean definitions from class path
2012-3-25 21:43:32 org.springframework.context.support
信息: Bean factory for application context [org.sp
2012-3-25 21:43:32 org.springframework.beans.factory
信息: Pre-instantiating singletons in org.springfr
----打印输出----
Hello!

```

如果想注入多个值怎么做？

使用构造方式注入多个值

### 11) 修改 A

```
Test.java  applicationContext.xml  A.java X
1 package tarena.demo4;
2
3 public class A {
4     private IB b;
5     private String name;
6
7     public A(IB b , String name) {
8         this.b = b;
9         this.name = name;
10    }
11
12    public void print() {
13        System.out.println("----打印输出----");
14        b.show();
15        System.out.println("姓名:" + name);
16    }
17 }
18
```

### 12) 修改 applicationContext.xml

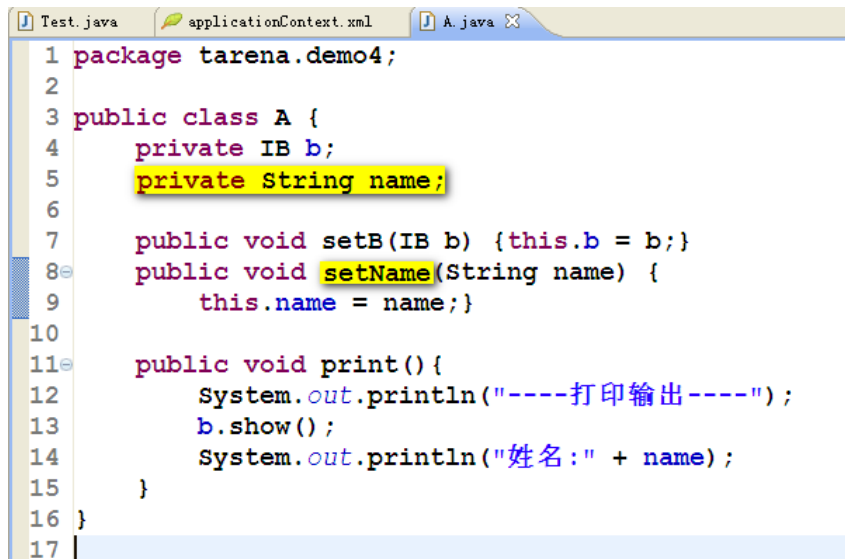
```
9         default-lazy-init="true">
10
11     <bean id="a" class="tarena.demo4.A">
12         <!-- property name="b" ref="b"></property --
13         <constructor-arg index="0" ref="b">
14         </constructor-arg>
15         <constructor-arg index="1" value="张三">
16         </constructor-arg>
17     </bean>
18
19     <bean id="b" class="tarena.demo4.B"></bean>
20
21 </beans>
22
```

### 13) 运行 Test

```
2012-3-26 10:06:05 org.springframework.beans.factory.xml
信息: Loading XML bean definitions from class path resource
2012-3-26 10:06:05 org.springframework.context.support
信息: Bean factory for application context [org.springframework
2012-3-26 10:06:05 org.springframework.beans.factory.support
信息: Pre-instantiating singletons in org.springframework
----打印输出----
Hello!
姓名张三
```

Set 方式注入多个值

#### 14) 修改 A



```
Test.java  applicationContext.xml  A.java X
1 package tarena.demo4;
2
3 public class A {
4     private IB b;
5     private String name;
6
7     public void setB(IB b) {this.b = b;}
8     public void setName(String name) {
9         this.name = name;}
10
11    public void print(){
12        System.out.println("----打印输出----");
13        b.show();
14        System.out.println("姓名:" + name);
15    }
16 }
17 |
```

#### 15) 修改 applicationContext.xml

```

9         default-lazy-init="true">
10
11     <bean id="a" class="tarena.demo4.A">
12         <property name="b" ref="b"></property>
13         <!-- 方式1 -->
14         <!-- property name="name" value="张四">
15     </property -->
16
17         <!-- 方式2(等价) -->
18     <property name="name">
19         <value>张思</value>
20     </property>
21 </bean>
22
23 <bean id="b" class="tarena.demo4.B"></bean>
24

```

那么到底该如何选择使用 set 方式注入还是构造方式注入？

如果需要注入的值非常多，那么使用构造方式就不太合适，在开发过程中，set 方式使用的也较多些。

如果 bean 属性中有集合，那么如何配置使用？

## 16) 新建 CollecitonBean

```

package tarena.demo4;

import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

public class CollectionBean {

    private List<String> city;
    private Set<String> name;
    private Map<String,Object> books;
    private Properties params;

    public void setParams(Properties params) {this.params = params;}
    public void setBooks(Map<String, Object> books) {this.books = books;}
    public void setName(Set<String> name) {this.name = name;}
    public void setCity(List<String> city) {this.city = city;}
}

```

```

public void show(){
    System.out.println("##List城市信息##");
    for(String s:city){
        System.out.println(s);
    }

    System.out.println("##Set朋友信息##");
    for(String s:name){
        System.out.println(s);
    }

    System.out.println("##Map图书信息##");
    Set<String> keys = books.keySet();
    for(String key:keys){
        System.out.println(key+ " : "+books.get(key));
    }

    System.out.println("##Properties数据库连接参数信息##");
    Set<Object> ids = params.keySet();
    for(Object id:ids){
        System.out.println(
            id+ " : "+params.getProperty(id.toString()));
    }
}

```

## 17) applicationContext.xml

```

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

```



<http://www.springframework.org/schema/tx/spring-tx-2.5.xsd>

default-lazy-init="true">

<bean id="collectionbean" class="tarena.demo4.CollectionBean">

<!-- List -->

<property name="city">

<list>

<value>北京</value>

<value>上海</value>

<value>深圳</value>

</list>

</property>

<!-- Set -->

<property name="name">

<set>

<value>tom</value>

<value>jack</value>

<value>rose</value>

</set>

</property>

<!-- Map -->

<property name="books">

<map>

<entry key="ISBN001" value="Struts框架开发"></entry>

<entry key="ISBN002" value="Hibernate框架开发"></entry>

<entry key="ISBN003" value="Spring框架开发"></entry>

</map>

</property>

<!-- Properties -->

<property name="params">

<props>

<prop key="username">root</prop>

<prop key="password">root</prop>

<prop key="driverClass">com.mysql.jdbc.Driver</prop>

<prop key="url">jdbc:mysql://localhost:3306/test</prop>

```
        </props>
    </property>
</bean>

</beans>
```

## 18) Test

```
package tarena.demo4;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {
    private static final String[] CONFIGS =
        {"tarena/demo4/applicationContext.xml"};

    public static void main(String[] args) {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext(CONFIGS);

        CollectionBean bean =
            (CollectionBean)ac.getBean("collectionbean");
        bean.show();
    }
}
```

## 19) 运行 Test

```
Console X SQL Results
<terminated> Test (10) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 26, 2012)
信息: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext
2012-3-26 10:28:34 org.springframework.beans.factory.xml.XmlBeanDefinitionReader
信息: Loading XML bean definitions from class path resource [applicationContext.xml]
2012-3-26 10:28:34 org.springframework.context.support.AbstractApplicationContext
信息: Bean factory for application context [org.springframework.beans.factory.xml.XmlBeanDefinitionReader]
2012-3-26 10:28:34 org.springframework.beans.factory.support.DefaultListableBeanFactory
信息: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory
List城市信息
北京
上海
深圳
Set朋友信息
tom
jack
rose
Map图书信息
ISBN001 : Struts框架开发
ISBN002 : Hibernate框架开发
ISBN003 : Spring框架开发
Properties数据库连接参数信息
driverClass : com.mysql.jdbc.Driver
url : jdbc:mysql://localhost:3306/test
password : root
username : root
```

(案例结束)

### 3. 注解方式配置 \*\*

常用配置方式有 [XML 文档配置](#)，还有一种是通过[注解方式配置](#)。

采用注解方式的[目的](#)就是为了简化 XML 配置文件。

注解方式（也叫注释）是 JDK5 版本提供的，之前的版本不支持。

Spring2.5 版本后支持注解方式，之前的版本不支持。

#### 3.1. 组件自动扫描功能 \*\*

首先需要在 applicationContext.xml 中添加<context:component-scan/>

1) 扫描 Bean 组件的注解，替代 xml 中的<bean>元素的定义。

- ✓ @Service 用于 Service 业务组件
- ✓ @Control 用于 Action 控制组件
- ✓ @Respository 用于 DAO 数据访问组件

- 
- ✓ @Component 用于其他组件
  - ✓ Bean 组件扫描到容器后，  
默认名字为类名(首字母小写)如果需要自定义名称可以使用@Service("id 名")

## 2) 依赖注入的注解标记

- ✓ @Resource 按名称@Resource(name="id 名")
- ✓ @AutoWired 按名称
- ✓ @Autowired
- ✓ @Qualifier("id 名")

## 3) 其他注解

- ✓ @Scope 等价于<bean scope="">
- ✓ @PostConstruct 等价于<bean init-method="">
- ✓ @PreDestroy 等价于<bean destroy-method="">

## 【案例 5】注解方式配置 \*\*

我们不采用 xml 的方式注入属性了，采用注解的方式注入

### 1) 使用工程 spring1

### 2) 采用注解的方式

#### a. 修改 applicationContext.xml

Xml 文档中不再配置 bean 了，我们引入新的标签

<context:component-scan> 标签的作用是进行组件自动扫描

注意，使用此标签的前提是必须具有 xmlns:context 命名空间

注意，和之前的 applicationContext.xml 做对比，有些命名空间现在用不到就可以删除。

```
<?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: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="tarena.demo5">
    </context:component-scan>

</beans>
```

---

## b. HelloBean

```
package tarena.demo5;

public interface HelloBean {

    public abstract void sayHello();

}
```

## c. 修改 ZhHelloBean

注意：容器如何找到 ZhHelloBean 的？

如果只写@Service，默认情况下相当于在 applicationContext.xml 中这样配置

```
<bean id="zhHelloBean" class="tarena.demo5.ZhHelloBean"></bean>
```

默认情况下，容器将类名 ZhHelloBean 首字母小写，作为<bean>的 id

```
package tarena.demo5;

import org.springframework.stereotype.Service;

@Service
public class ZhHelloBean implements HelloBean {

    public void sayHello(){
        System.out.println("世界你好！");
    }

}
```

## d. 修改 Test

```
package tarena.demo5;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {

    private static final String CONFIG =
        "tarena/demo5/applicationContext.xml";

    public static void main(String[] args) {
        ApplicationContext ac =
```

```
new ClassPathXmlApplicationContext(CONFIG;
```

```
    ZhHelloBean hello = (ZhHelloBean)ac.getBean("zhHelloBean");  
    hello.sayHello();  
}  
}
```

#### e. 运行 Test

```
<terminated> Test (11) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 26, 2012 14:12:06)  
2012-3-26 14:12:06 org.springframework.context.support.  
信息: Refreshing org.springframework.context.support.C.  
2012-3-26 14:12:06 org.springframework.beans.factory.xml.  
信息: Loading XML bean definitions from class path res  
2012-3-26 14:12:06 org.springframework.context.support.  
信息: Bean factory for application context [org.spring:  
2012-3-26 14:12:07 org.springframework.beans.factory.s  
信息: Pre-instantiating singletons in org.springframework  
世界你好!
```

### 3) 采用 xml 配置的方式 ( 之前的方式 )

#### a. 修改 applicationContext.xml

使用 xml 配置

这里<bean>的 id 可以自定义为任意的字符，比如此处都为小写

```
<?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: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">  
    <bean id="zhhellobean" class="tarena.demo5.ZhHelloBean"></bean>  
</beans>
```

#### b. 修改 ZhHelloBean

木有注解

```
package tarena.demo5;
```

```
public class ZhHelloBean implements HelloBean {

    public void sayHello(){
        System.out.println("世界你好！");
    }

}
```

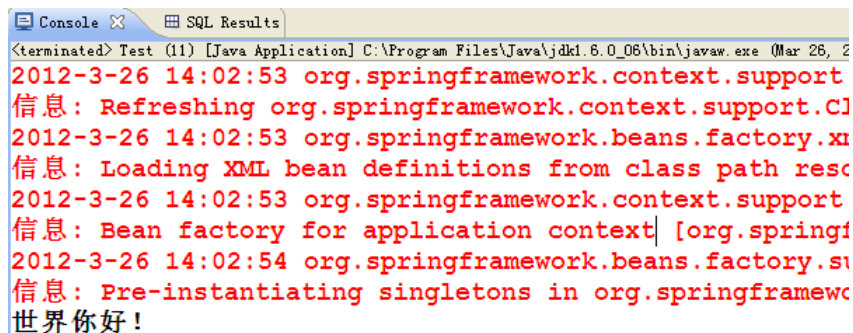
### c. Test

```
public class Test {
    private static final String CONFIG =
        "tarena/demo5/applicationContext.xml";

    public static void main(String[] args) {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext(CONFIG);

        ZhHelloBean hello =
            (ZhHelloBean)ac.getBean("zhhellobean");
        hello.sayHello();
    }
}
```

### d. 运行 Test



```
<terminated> Test (11) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 26, 2012 14:02:53)
2012-3-26 14:02:53 org.springframework.context.support
信息: Refreshing org.springframework.context.support.Cl
2012-3-26 14:02:53 org.springframework.beans.factory.xml
信息: Loading XML bean definitions from class path res
2012-3-26 14:02:53 org.springframework.context.support
信息: Bean factory for application context [org.springf
2012-3-26 14:02:54 org.springframework.beans.factory.st
信息: Pre-instantiating singletons in org.springframework
世界你好！
```

通过对比，我们发现注解的方式更简便些。

当然，我们可以自定义注解需要使用 bean 的名字

### 4) 修改 ZhHelloBean

```
applicationContext.xml | HelloBean.java | ZhHelloBean.java | Test.java
1 package tarena.demo5;
2
3 import org.springframework.stereotype.Service;
4
5 @Service("zhhellohellobean")
6 public class ZhHelloBean implements HelloBean {
7
8     public void sayHello(){
9         System.out.println("世界你好！");
10    }
11 }
12
```

## 5) 修改 Test

```
applicationContext.xml | Test.java
package tarena.demo5;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {
    private static final String CONFIG =
        "tarena/demo5/applicationContext.xml";

    public static void main(String[] args) {
        ApplicationContext ac =
            new ClassPathXmlApplicationContext(CONFIG);

        ZhHelloBean hello =
            (ZhHelloBean) ac.getBean("zhhellohellobean");
        hello.sayHello();
    }
}
```

如果我们使用 UseBean 调用 HelloBean，使用注解方式该怎么做？

使用配置文件的方式，我们这样做

```
10e <bean id="usebean" class="tarena.demo2.UseBean">
11e     <property name="hello" ref="enhellobean">
12     </property>
13 </bean>
14 <bean id="enhellobean" class="tarena.demo2.EnHel
15 <bean id="zhhellobean" class="tarena.demo2.ZhHel
```

使用注解方式，我们这样做



## 6) UseBean

在属性上加入@Resource 注解，相当于

```
<bean>
```

```
    <property></property>
```

```
</bean>
```

```
package tarena.demo5;

import javax.annotation.Resource;    //该注解是由JDK提供，而非Spring提供的
import org.springframework.stereotype.Service;

@Service("usebean")
public class UseBean {

    @Resource
    private HelloBean hello;

    public void show(){
        System.out.println("显示Hello消息");
        hello.sayHello();
    }

    public void setHello(HelloBean hello) {
        this.hello = hello;
    }

}
```

## 7) ZhHelloBean

```
package tarena.demo5;

import org.springframework.stereotype.Service;

@Service("zhhellobean")
public class ZhHelloBean implements HelloBean {

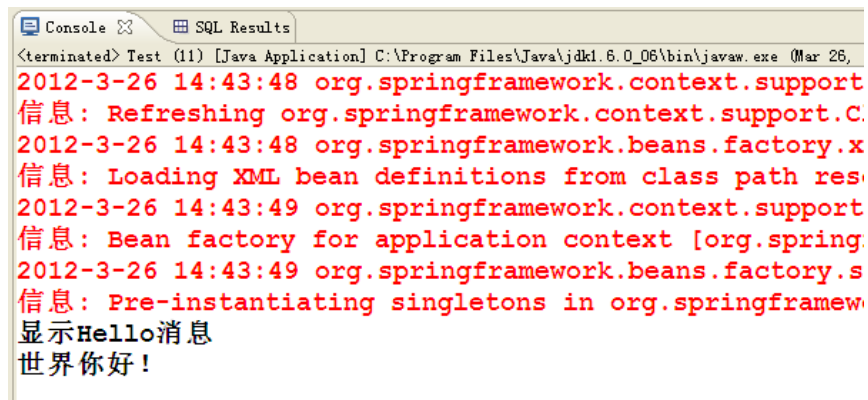
    public void sayHello(){
        System.out.println("世界你好！");
    }
}
```

```
}  
}
```

## 8) Test

```
package tarena.demo5;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Test {  
    private static final String CONFIG =  
        "tarena/demo5/applicationContext.xml";  
  
    public static void main(String[] args) {  
        ApplicationContext ac =  
            new ClassPathXmlApplicationContext(CONFIG);  
  
        UseBean bean = (UseBean)ac.getBean("usebean");  
        bean.show();  
    }  
}
```

## 9) 运行 Test



```
<terminated> Test (11) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Mar 26,  
2012-3-26 14:43:48 org.springframework.context.support.C  
信息: Refreshing org.springframework.context.support.C  
2012-3-26 14:43:48 org.springframework.beans.factory.x  
信息: Loading XML bean definitions from class path res  
2012-3-26 14:43:49 org.springframework.context.support  
信息: Bean factory for application context [org.spring  
2012-3-26 14:43:49 org.springframework.beans.factory.s  
信息: Pre-instantiating singletons in org.springframe  
显示Hello消息  
世界你好!
```

## 总结：

@Service("zhhellobean")相当于xml文件中配置<bean>

<http://www.springframework.org/schema/tx> http

```
<bean id="usebean" class="tarena.demo2.UseBean"/>
<bean id="enhellobean" class="tarena.demo2.EnHelloBean"/>
<bean id="zhhellobean" class="tarena.demo2.ZhHelloBean"/>

</beans>
```

@Resource相当于xml文件中配置<property>

```
<bean id="usebean" class="tarena.demo2.UseBean">
  <property name="hello" ref="enhellobean"/>
</bean>
<bean id="enhellobean" class="tarena.demo2.EnHell
```

### 其他的注解的使用

首先需要在 applicationContext.xml 中添加<context:component-scan/>

1. 扫描 Bean 组件的注解，替代 xml 中的<bean>元素的定义。

- ✓ @Service            用于 Service 业务组件
- ✓ @Control            用于 Action 控制组件
- ✓ @Respository        用于 DAO 数据访问组件
- ✓ @Component        用于其他组件

Bean 组件扫描到容器后，

默认名字为类名(首字母小写)如果需要自定义名称可以使用@Service("id 名")

2. 依赖注入的注解标记

- ✓ @Resource            JDK 提供的  
                          先按类型，后按名称来自动装配
- ✓ @AutoWired            Spring 提供的  
                          先按名称，后按类型来自动装配
- ✓ @Qualifier("id 名")

3. 其他注解

- ✓ @Scope 等价于<bean scope="">
- ✓ @PostConstruct 等价于<bean init-method="">
- ✓ @PreDestroy 等价于<bean destroy-method="">

和@Resource的功能相同，@Autowired也是用于自动装配的。

## 10) 修改 UseBean

我们使用@Autowired 结果是一样的。

**注意：**不论使用@Resource 还是@Autowired，我们不用再写 set 方法的。

```
8 @Service("usebean")
9 public class UseBean {
10
11     @Autowired
12     private HelloBean hello;
13
14     public void show() {
15         System.out.println("显示Hello消息");
16         hello.sayHello();
17     }
18
19     // public void setHello(HelloBean hello) {
20     //     this.hello = hello;
21     // }
22 }
```

我们这样写的时候，表示只有注解名字相同时，才自动装配。

## 11) 修改 UseBean

```
6 @Service("usebean")
7 public class UseBean {
8
9     @Resource(name="zhhellobean")
10     private HelloBean hello;
11
12     public void show() {
13         System.out.println("显示Hello消息");
14         hello.sayHello();
15     }
16
17     // public void setHello(HelloBean hello) {
18     //     this.hello = hello;
19     // }
20 }
```

在之前，我们都只装配 zhhellobean，现在我们想将 enhellobean 也装配进来

## 12) ZhHelloBean

```

HelloBean.java  ZhHelloBean.java X  UseBean.java  Test.java  EnHelloBes
1 package tarena.demo5;
2
3 import org.springframework.stereotype.Service;
4
5 @Service("zhhellobean")
6 public class ZhHelloBean implements HelloBean {
7
8     public void sayHello(){
9         System.out.println("世界你好!");
10    }
11 }
12

```

### 13) 修改 EnHelloBean

```

HelloBean.java  UseBean.java  Test.java  EnHelloBean.java X
1 package tarena.demo5;
2
3 import org.springframework.stereotype.Service;
4
5 @Service("enhellobean")
6 public class EnHelloBean implements HelloBean{
7     public void sayHello(){
8         System.out.println("Hello the world!");
9     }
10 }
11

```

此时，如果我们这样写，就会出异常

### 14) UseBean

```

HelloBean.java  UseBean.java X  Test.java  EnHelloBean.java
2
3 import javax.annotation.Resource;
4 import org.springframework.stereotype.Service;
5
6 @Service("usebean")
7 public class UseBean {
8
9     @Resource
10    private HelloBean hello;
11
12    public void show(){
13        System.out.println("显示Hello消息");
14        hello.sayHello();
15    }
16

```

## 15) 运行 Test

出现异常

```
Exception: No unique bean of type [tarena.demo5.HelloBean] i
anFactory.resolveDependency(DefaultListableBeanFactory.
PostProcessor.autowireResource(CommonAnnotationBeanPost
PostProcessor.getResource(CommonAnnotationBeanPostProce
PostProcessor$ResourceElement.getResourceToInject(Commo
```

异常显示如下，表示有两个符合条件的 bean，spring 不知该选哪个了

Caused by: [org.springframework.beans.factory.NoSuchBeanDefinitionException](#):  
No unique bean of type [tarena.demo5.HelloBean] is defined: expected single  
matching bean but found 2: [enhellobean, zhhellobean]

这种情况下，只能通过@Resource(name="")来指定。

```
6 @Service("usebean")
7 public class UseBean {
8
9     @Resource(name="enhellobean")
10     private HelloBean hello;
11
12     public void show() {
13         System.out.println("显示Hello消息");
14         hello.sayHello();
15     }
16 }
```

当然，一般情况下，还是推荐直接使用简单的@Resource

@Resource按照名字注入的方法较简单，@Autowired按照名字注入需要再加一个注解

## 16) 修改 UseBean

```
9 @Service("usebean")
10 public class UseBean {
11
12     @Autowired
13     @Qualifier("zhhellobean")
14     private HelloBean hello;
15
16     public void show() {
17         System.out.println("显示Hello消息");
18         hello.sayHello();
19     }
20
21 }
```

这些也有配套的注解

```
9         default-lazy-init="false">
10
11     <bean id="mybean"
12         lazy-init="true"
13         scope="singleton"
14         init-method="myinit"
15         destroy-method="mydestroy"
16         class="tarena.demo3.MyBean">
17     </bean>
18 </beans>
```

@scope等价于scope属性

@PostConstruct等价于init-method

@PreDestroy等价于destroy-method

如下所示

#### 17) 修改 UseBean

```
9
10 @Service("usebean")
11 @Scope("prototype")
12 public class UseBean {
13
14     @Autowired
15     @Qualifier("zhhellobean")
16     private HelloBean hello;
17
18     public void show() {
19         System.out.println("显示Hello消息");
20         hello.sayHello();
21     }
22
23     @PostConstruct
24     public void myinit() {
25         System.out.println("init...");
26     }
27
28     @PreDestroy
29     public void mydestroy() {
30         System.out.println("destory..");
31     }
32 }
```

注解形式和 xml 形式各有优劣，注解方式现在比较流行。

注解方式的优点是使用方便，缺点是和 Java 代码掺和在一起，不好修改。

Xml 方式的优点是修改方便，但是缺点是配置工作量较大。

**( 案例结束 )**

---