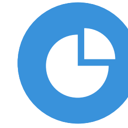
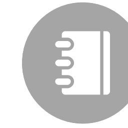
第2章 Spring中的Bean的管理

湖南科技大学
计算机科学与工程学院

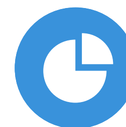
学习目标/Target





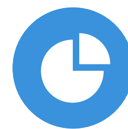
- 了解Spring IoC容器的原理
- 掌握Bean标签及其属性的使用
- 熟悉Bean的实例化

学习目标/Target



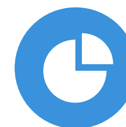
- 掌握Bean的作用域
- 掌握Bean的装配方式
- 熟悉Bean的生命周期

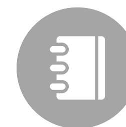
章节概述/ Summary



第1章详细讲解了控制反转和依赖注入，它们实现了组件的实例化不再由应用程序完成，转而交由Spring容器完成，从而将组件之间的依赖关系进行了解耦。控制反转和依赖注入都是通过Bean实现的，Bean是注册到Spring容器中的Java类，任何一个Java类都可以是一个Bean。Bean由Spring进行管理，本章将针对Bean的管理进行详细讲解。

目录/Contents



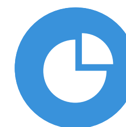


01 Spring IoC容器

02 Bean的配置

03 Bean的实例化

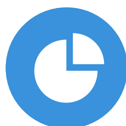
目录/Contents



04 Bean的作用域

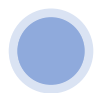
05 Bean的装配方式

06 Bean的生命周期



2.1 Spring IoC容器

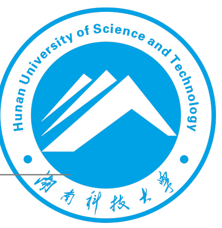
2.1.1 BeanFactory接口



先定一个小目标！

了解Spring中的BeanFactory接口，能够说出BeanFactory接口的常用方法有哪些

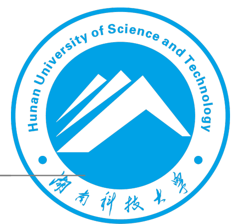
2.1.1 BeanFactory接口



BeanFactory接口的常用方法

方法名称	描述
getBean（String name）	根据参数名称获取Bean
getBean（String name,Class<T> type）	根据参数名称、参数类型获取Bean
<T>T getBean（Class<T> requiredType）	根据参数类型获取Bean
Object getBean（String name,Object... args）	根据参数名称获取Bean
isTypeMatch（String name,Resolvable Typetype）	判断是否有与参数名称、参数类型匹配的Bean
Class <?>getType（String name）	根据参数名称获取类型
String[] getAliases（String name）	根据实例的名字获取实例的别名数组
boolean containsBean（String name）	根据Bean的名称判断Spring容器是否含有指定的Bean

2.1.1 BeanFactory接口

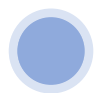


BeanFactory接口实例的语法格式

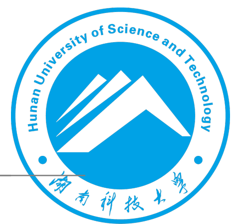
Spring提供了几个BeanFactory接口的实现类，其中最常用的是XmlBeanFactory，它可以读取XML文件并根据XML文件中的配置信息生成BeanFactory接口的实例，BeanFactory接口的实例用于管理Bean。XmlBeanFactory类读取XML文件生成BeanFactory接口实例的具体语法格式如下。

```
BeanFactory beanFactory=new XmlBeanFactory
        (new FileSystemResource(”D:/bean.xml”));
```

2.1.2 ApplicationContext接口



了 解 ApplicationContext 接 口 ， 能 够 说 出 ApplicationContext接口的常用方法有哪些

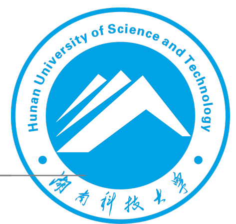


ApplicationContext接口的特点

ApplicationContext接口建立在BeanFactory接口的基础之上，它丰富了BeanFactory接口的特性，例如，添加了对国际化、资源访问、事件传播等方面的支持。

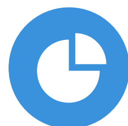
ApplicationContext接口可以为单例的Bean实行预初始化，并根据<property>元素执行setter方法，单例的Bean可以直接使用，提升了程序获取Bean实例的性能。

2.1.2 ApplicationContext接口



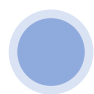
ApplicationContext接口的常用实现类

类名称	描述
ClassPathXmlApplicationContext	从类路径加载配置文件,实例化ApplicationContext接口
FileSystemXmlApplicationContext	从文件系统加载配置文件,实例化ApplicationContext接口
AnnotationConfigApplicationContext	从注解中加载配置文件,实例化ApplicationContext接口
WebApplicationContext	在Web应用中使用,从相对于Web根目录的路径中加载配置文件,实例化ApplicationContext接口
ConfigurableWebApplicationContext	扩展了WebApplicationContext类,它可以通过读取XML配置文件的方式实例化WebApplicationContext类



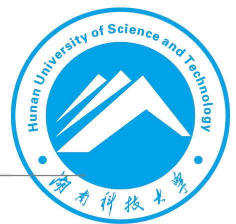
2.2 Bean的配置

2.2 Bean的配置



掌握Bean的配置，能够运用<bean>元素的属性以及常用的子元素配置Bean

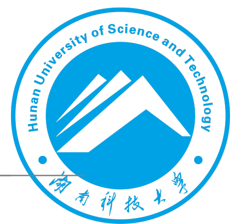
2.2 Bean的配置



Spring容器所支持的配置文件格式

Spring容器支持XML和Properties两种格式的配置文件，在实际开发中，最常用的是XML格式的配置文件。XML是标准的数据传输和存储格式，方便查看和操作数据。在Spring中，XML配置文件的根元素是<beans>，<beans>元素包含<bean>子元素，每个<bean>子元素可以定义一个Bean，通过<bean>元素将Bean注册到Spring容器中。

2.2 Bean的配置



<bean>元素的常用属性

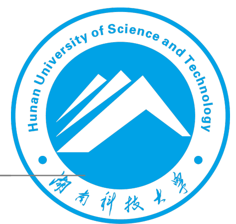
属性	描述
id	id属性是<bean>元素的唯一标识符,Spring容器对Bean的配置和管理通过id属性完成,装配Bean时也需要根据id值获取对象。
name	name属性可以为Bean指定多个名称,每个名称之间用逗号或分号隔开。
class	class属性可以指定Bean的具体实现类,其属性值为对象所属类的全路径。
scope	scope属性用于设定Bean实例的作用范围,其属性值有：singleton（单例）、prototype（原型）、request、session和global session。

2.2 Bean的配置

<bean>元素的常用子元素

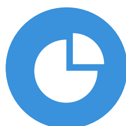
元素	描述
<constructor-arg>	使用<constructor-arg>元素可以为Bean的属性指定值。
<property>	<property>元素的作用是调用Bean实例中的setter方法完成属性赋值,从而完成依赖注入。
ref	ref是<property>、<constructor-arg>等元素的属性，可用于指定Bean工厂中某个Bean实例的引用；也可用于指定Bean工厂中某个Bean实例的引用。
value	value是<property>、<constructor-arg>等元素的属性，用于直接指定一个常量值；也可以用于直接指定一个常量值。
<list>	<list>元素是<property>等元素的子元素，用于指定Bean的属性类型为List或数组。
<set>	<set>元素是<property>等元素的子元素，用于指定Bean的属性类型为set。
<map>	<map>元素是<property>等元素的子元素，用于指定Bean的属性类型为Map。
<entry>	entry>元素是<map>元素的子元素，用于设定一个键值对。<entry>元素的key属性指定字符串类型的键。

2.2 Bean的配置



普通的Bean通常只需定义id（或者name）和class两个属性

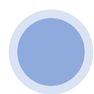
```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
   <!--使用id属性定义bean1，对应的实现类为com.itheima.Bean1-->
   <bean id="bean1" class="com.hnust.Bean1">
   </bean>
   <!--使用name属性定义bean2，对应的实现类为com.itheima.Bean2-->
   <bean name="bean2" class="com.hnust.Bean2"/>
</beans>
```



2.3 Bean的实例化

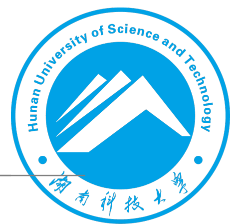
2.3.1 构造方法实例化

先定一个小目标！



熟悉Spring构造方法实例化，能够通过构造方法实例化Bean

2.3.1 构造方法实例化



下面通过一个案例演示Spring容器如何通过构造方法实例化Bean。

STEP 01

在IDEA中创建一个名为chapter07的Maven项目，然后在项目的pom.xml文件中配置需使用到的Spring四个基础包和Spring的依赖包。

```
<!– 这里只展示了一个依赖包-->
<!-- spring-expression的依赖包 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-expression</artifactId>
        <version>5.2.8.RELEASE</version>
    </dependency>
```

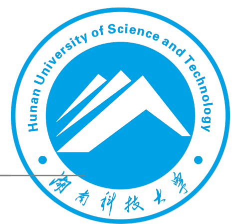
2.3.1 构造方法实例化

STEP 02

创建一个名称为com.hnust的包，在该包中创建Bean1类。

```
package com.hnust;
public class Bean1 {
    public Bean1(){
        System.out.println("这是Bean1");
    }
}
```

2.3.1 构造方法实例化



STEP 03

新建applicationBean1.xml作为Bean1类的配置文件，在该配置文件中定义一个id为bean1的Bean，并通过class属性指定其对应的实现类为Bean1。

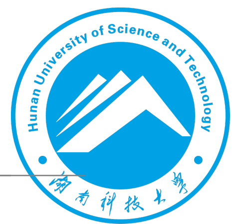
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="bean1" class="com.hnust.Bean1"></bean>
</beans>
```

2.3.1 构造方法实例化

STEP 04

创建测试类Bean1Test，在main()方法中通过加载applicationBean1.xml配置文件初始化Spring容器，再通过Spring容器生成Bean1类的实例bean1，用来测试构造方法是否能实例化Bean1。

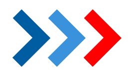
```
public class Bean1Test {
    public static void main(String[] args){
        // 加载applicationBean1.xml配置
        ApplicationContext applicationContext=new
                                     ClassPathXmlApplicationContext("applicationBean1.xml");
        // 通过容器获取配置中bean1的实例
        Bean1 bean=(Bean1) applicationContext.getBean("bean1");
        System.out.print(bean);
       }
}
```



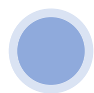
STEP 05

在IDEA中启动Bean1Test类，控制台会输出结果。

```
Run:  Bean1Test
"C:\Program Files\Java\jdk1.8.0_201\bin\java.exe" ...
这是Bean1
com.itheima.Bean1@53b32d7
Process finished with exit code 0
```

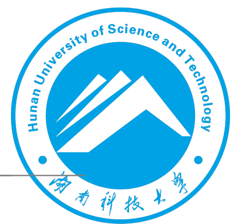


2.3.2 静态工厂实例化



熟悉Spring静态工厂实例化，能够通过静态工厂实例化Bean

2.3.2 静态工厂实例化



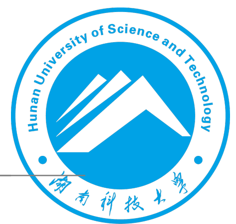
下面通过一个案例演示如何使用静态工厂方式实例化Bean。

STEP 01

创建Bean2类，该类与Bean1类一样，只定义一个构造方法，不需额外添加任何方法。

```
package com.itheima;
public class Bean2 {
    public Bean2(){
        System.out.println("这是Bean2");
    }
}
```

2.3.2 静态工厂实例化

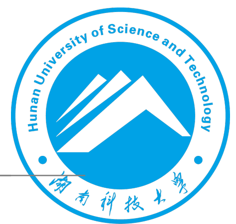


STEP 02

创建一个MyBean2Factory类，在该类中定义一个静态方法createBean()，用于创建Bean的实例。createBean()方法返回Bean2实例。

```
package com.hnust;
public class MyBean2Factory {
        //使用MyBean2Factory类的工厂创建Bean2实例
        public static Bean2 createBean(){
        return new Bean2();
        }
}
```

2.3.2 静态工厂实例化



STEP 03

新建applicationBean2.xml文件，作为MyBean2Factory类的配置文件。

```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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.3.xsd">
<bean id="bean2"
      class="com.hnust.MyBean2Factory"
      factory-method="createBean"/>
</beans>
```

2.3.2 静态工厂实例化

STEP 04

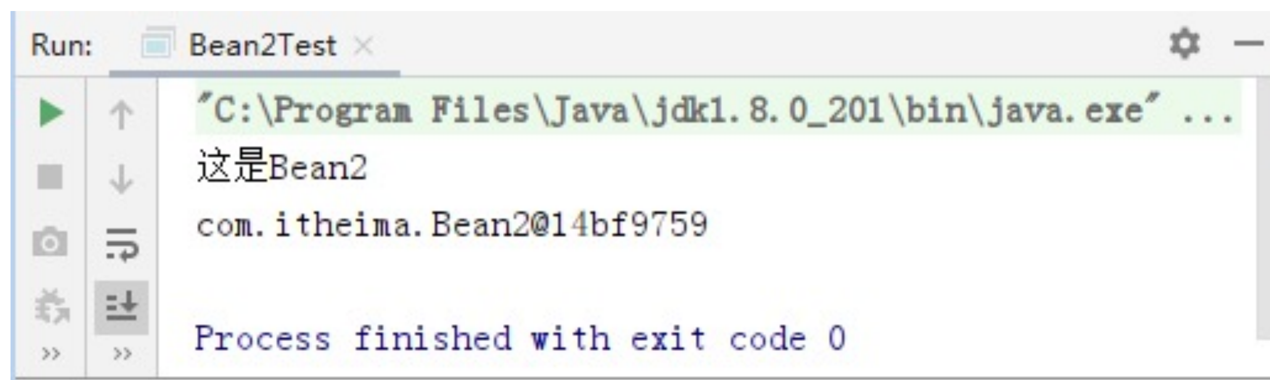
创建一个测试类Bean2Test，用于测试使用静态工厂方式是否能实例化Bean。

```
public class Bean2Test {
    public static void main(String[] args) {
        // ApplicationContext在加载配置文件时，对Bean进行实例化
        ApplicationContext applicationContext =
                new ClassPathXmlApplicationContext("applicationBean2.xml");
        System.out.println(applicationContext.getBean("bean2"));
    }
}
```

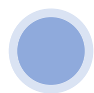
2.3.2 静态工厂实例化

STEP 05

在IDEA中启动Bean2Test类，控制台会输出结果。



2.3.3 实例工厂实例化



熟悉Spring实例工厂实例化，能够通过实例工厂实例化Bean

2.3.3 实例工厂实例化

下面通过一个案例演示如何使用实例工厂方式实例化Bean。

STEP 01

创建Bean3类，该类与Bean1一样，只需编写一个构造方法。

```
package com.itheima;
public class Bean3 {
    public Bean3(){
        System.out.println("这是Bean3");
    }
}
```

2.3.3 实例工厂实例化

STEP 02

创建一个MyBean3Factory类，在该类中定义无参构造方法，并定义createBean()方法用于创建Bean3对象。

```
package com.itheima;
public class MyBean3Factory {
    public MyBean3Factory() {
        System.out.println("bean3工厂实例化中");
    }
    public Bean3 createBean() { //创建Bean3实例的方法
        return new Bean3();
    }
}
```

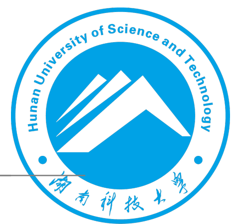
2.3.3 实例工厂实例化

STEP 03

新建applicationBean3.xml文件，作为MyBean3Factory类的配置文件。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.3.xsd">
    <!-- 配置工厂 -->
    <bean id="myBean3Factory”          class="com.hnust.MyBean3Factory" />
    <!-- 使用factory-bean属性指向配置的实例工厂-->
    <bean id="bean3" factory-bean="myBean3Factory"
          factory-method="createBean" />
</beans>
```

2.3.3 实例工厂实例化



STEP 04

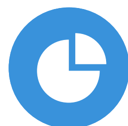
创建一个测试类BeanTest3，用于测试使用实例化工厂方式是否能实例化Bean。

```
public class Bean3Test {
    public static void main(String[] args) {
        // ApplicationContext在加载配置文件时，对Bean进行实例化
        ApplicationContext applicationContext =
                new ClassPathXmlApplicationContext("applicationBean3.xml");
        System.out.println(applicationContext.getBean("bean3"));
    }
}
```

2.3.3 实例工厂实例化

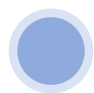
STEP 05

在IDEA中启动Bean3Test类，控制台会输出结果。



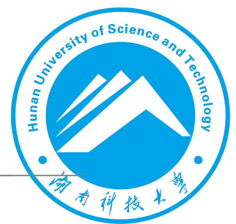
2.4 Bean的作用域

2.4.1 singleton作用域



掌握singleton作用域，能够在Spring中对Bean设置singleton作用域，并说出singleton作用域的作用范围

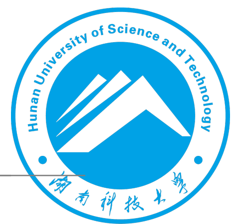
2.4.1 singleton作用域



Spring支持的5种作用域

作用域名城	描述
singleton	单例模式。在单例模式下，Spring 容器中只会存在一个共享的Bean实例，所有对Bean的请求，只要请求的id（或name）与Bean的定义相匹配，会返回Bean的同一个实例。
prototype	原型模式，每次从容器中请求Bean时，都会产生一个新的实例。
request	每一个HTTP请求都会有自己的Bean实例，该作用域只能在基于Web的Spring ApplicationContext中使用。
session	每一个HTTPsession请求都会有自己的Bean实例，该作用域只能在基于Web的Spring ApplicationContext中使用。
global session	限定一个Bean的作用域为Web应用（HTTPsession）的生命周期，只有在Web应用中使用Spring时，该作用域才有效。

2.4.1 singleton作用域



下面将通过案例的方式演示Spring容器中singleton作用域的使用。

STEP 01

将id为bean1的作用域设置为singleton。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="bean1" class="com.hnust.Bean1" scope="singleton"></bean>
</beans>
```

2.4.1 singleton作用域

STEP 02

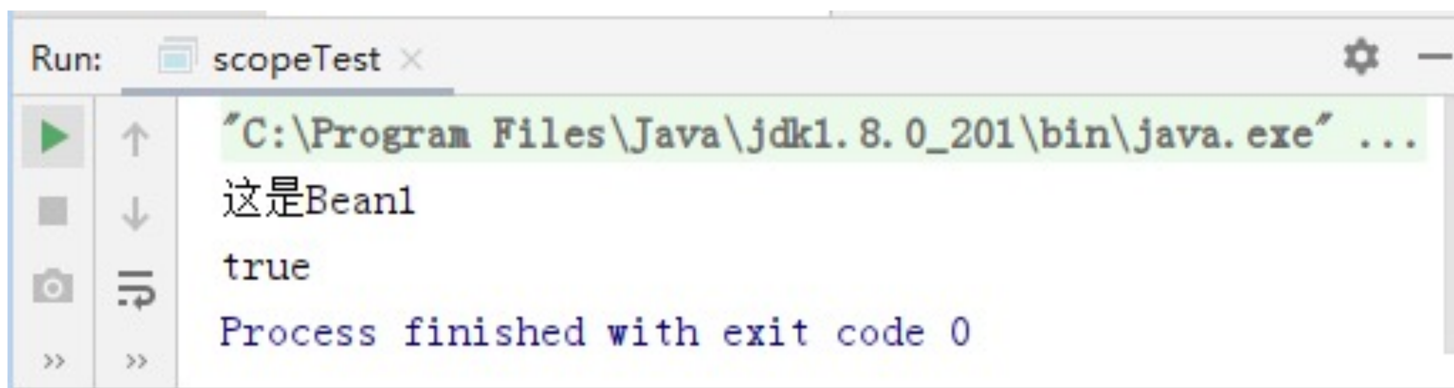
创建测试类scopeTest，在main()方法中通过加载applicationBean1.xml配置文件初始化Spring容器，通过Spring容器获取Bean1类的两个实例，判断两个实例是否为同一个。

```
public class scopeTest{
    public static void main(String[] args){
        ApplicationContext applicationContext=new
              ClassPathXmlApplicationContext("applicationBean1.xml");
        Bean1 bean1_1=(Bean1) applicationContext.getBean("bean1");
        Bean1 bean1_2=(Bean1) applicationContext.getBean("bean1");
        System.out.print(bean1_1==bean1_2);     }
}
```

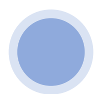
2.4.1 singleton作用域

STEP 03

在IDEA中启动scopeTest类，控制台会输出运行结果。

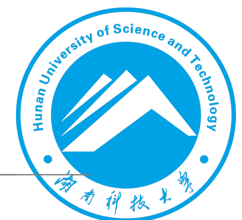


2.4.2 prototype作用域



掌握prototype作用域，能够在Spring中对Bean设置prototype作用域，并说出prototype作用域的作用范围

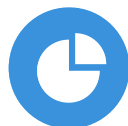
2.4.2 prototype作用域



prototype作用域的使用

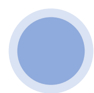
在7.4.1节的基础上修改配置文件，将id为bean1的作用域设置为prototype。

```
<bean id="bean1" class="com.hnust.Bean1" scope="prototype"></bean>
```

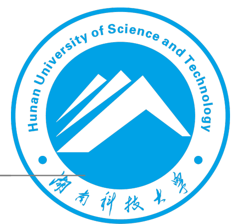


2.5 Bean的装配方式

2.5.1 基于XML的装配

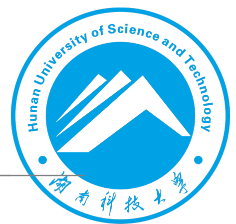


掌握Bean基于XML的装配，能够使用XML装配方式对Bean进行装配



两种基于XML的装配方式

在基于XML的装配就是读取XML配置文件中的信息完成依赖注入，Spring容器提供了两种基于XML的装配方式，属性setter方法注入和构造方法注入。下面分别对这两种装配方式进行介绍。



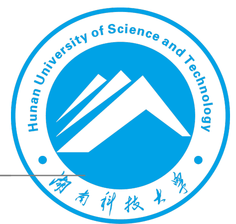
a.属性setter方法注入

属性setter方法注入要求一个Bean必须满足以下两点要求。

（1）Bean类必须提供一个默认的无参构造方法。

（2）Bean类必须为需要注入的属性提供对应的setter方法。

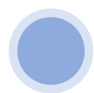
2.5.1 基于XML的装配



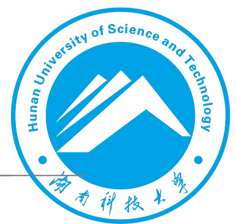
b.构造方法注入

使用构造方法注入时，在配置文件里，需要使用<bean>元素的子元素<constructor-arg>来定义构造方法的参数，例如，可以使用其value属性（或子元素）来设置该参数的值。

2.5.2 基于注解的装配



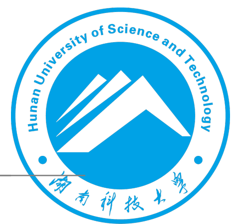
掌握Bean**基于注解的装配**，能够使用注解装配方式装配Bean



XML与注解装配的比较

在Spring中，使用XML配置文件可以实现Bean的装配工作，但在实际开发中如果Bean的数量较多，会导致XML配置文件过于臃肿，给后期维护和升级带来一定的困难。为解决此问题，Spring提供了注解，通过注解也可以实现Bean的装配。

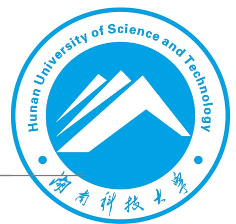
2.5.2 基于注解的装配



Spring的常用注解

注解	描述
@Component	指定一个普通的Bean，可以作用在任何层次。
@Controller	指定一个控制器组件Bean，用于将控制层的类标识为Spring中的Bean，功能上等同于@Component。
@Service	指定一个业务逻辑组件Bean，用于将业务逻辑层的类标识为Spring中的Bean，功能上等同于@Component。
@Repository	指定一个数据访问组件Bean，用于将数据访问层的类标识为Spring 中的Bean，功能上等同于@Component。
@Scope	指定Bean实例的作用域。
@Value	指定Bean实例的注入值。

2.5.2 基于注解的装配



Spring的常用注解

注解	描述
@Autowired	指定要自动装配的对象。
@Resource	指定要注入的对象。
@Qualifier	指定要自动装配的对象名称，通常与@Autowired联合使用。
@PostConstruct	指定Bean实例完成初始化后调用的方法。
@PreDestroy	指定Bean实例销毁前调用的方法。

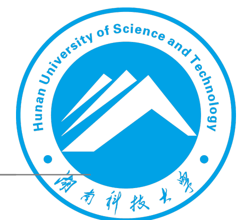
下面通过案例演示如何使用注解来装配Bean，具体实现步骤如下。

STEP 01

导入依赖：在项目chapter07的pom.xml文件中导入spring-aop-5.2.8.RELEASE.jar依赖包，导入代码如下所示。

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>5.2.8.RELEASE</version>
</dependency>
```

2.5.2 基于注解的装配



STEP 02

XML配置文件：创建applicationContext.xml，在该文件中引入Context约束并启动Bean的自动扫描功能。

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

STEP 03

定义实体类：新建entity包，在entity包下创建User实体类。

```
@Component("user")
@Scope("singleton")
public class User {
    @Value("1")
    private int id;
    @Value("张三")
    private String name;
    @Value("123")
    private String password;
    // 省略getter/setter方法和toString()方法
}
```

2.5.2 基于注解的装配

STEP 04

定义dao层：创建UserDao接口作为数据访问层接口，并在UserDao接口中声明save()方法，用于查询User实体的对象信息。

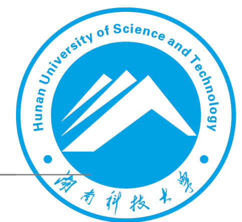
```
package com.hnust.dao;
public interface UserDao {
    public void save();
}
```

STEP 05

实现dao层： 创建UserDaoImpl作为UserDao的实现类，并在UserDaoImpl类中实现UserDao接口中的save()方法。

```
@Repository("userDao")
public class UserDaoImpl implements UserDao {
    public void save(){
        ApplicationContext applicationContext=new
            ClassPathXmlApplicationContext("applicationContext.xml");
        User user=(User) applicationContext.getBean("user");
        System.out.println(user);
        System.out.println("执行UserDaoImpl.save()");
    }
}
```

2.5.2 基于注解的装配



STEP 06

定义service层：创建UserService接口作为业务逻辑层接口，并在UserService接口中定义save()方法。

```
package com.hnust.service;
public interface UserService {
    public void save();
}
```

STEP 07

实现service层：创建UserServiceImpl作为UserService的实现类，并在UserServiceImpl类中实现UserService接口中的save()方法。

```
@Service("userService")
public class UserServiceImpl implements UserService {
    //使用@Resource注解注入UserDao
    @Resource(name="userDao")
    private UserDao userDao;
    public void save(){
        this.userDao.save();
        System.out.println("执行UserServiceImpl.save()");
    }
}
```

2.5.2 基于注解的装配

STEP 08

定义controller层： 创建UserController类作为控制层。

```
@Controller
public class UserController {
    //使用@Resource注解注入UserService
    @Resource(name="userService")
    private UserService userService;
    public void save(){
        this.userService.save();
        System.out.println("执行UserController.save()");
    }
}
```

2.5.2 基于注解的装配

STEP 09

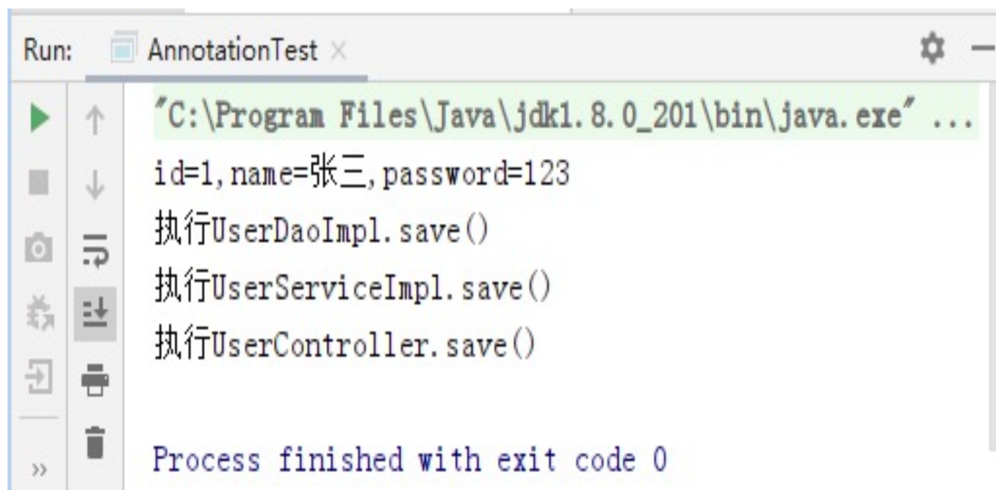
定义测试类：创建测试类AnnotationTest，在该类中编写测试代码，通过Spring容器加载配置文件并获取UserController实例，然后调用实例中的save()方法。

```
public class AnnotationTest {
    public static void main(String[] args){
        ApplicationContext applicationContext=new
            ClassPathXmlApplicationContext("applicationContext.xml");
        UserController usercontroller=(UserController)
            applicationContext.getBean("userController");
        usercontroller.save();
    }
}
```

2.5.2 基于注解的装配

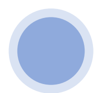
STEP 10

查看运行结果：在IDEA中启动AnnotationTest类，控制台会输出结果。



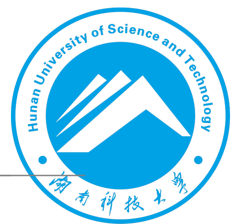
2.5.3 自动装配

先定一个小目标！



掌握Bean的**自动装配**，能够使用自动装配方式装配Bean

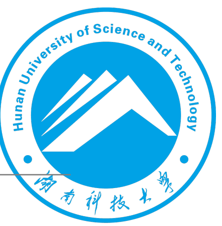
2.5.3 自动装配



如何实现自动装配

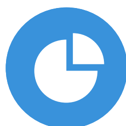
Spring的<bean>元素中包含一个autowire属性，可以通过设置autowire属性的值实现Bean的自动装配。

2.5.3 自动装配



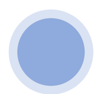
autowire属性的值

属性值	描述
default（默认值）	由<bean>的上级元素<beans>的default-autowire属性值确定。
byName	根据<bean>元素id属性的值自动装配。
byType	根据<bean>元素的数据类型（Type）自动装配，如果一个Bean的数据类型，兼容另一个Bean中的数据类型，则自动装配。
constructor	根据构造函数参数的数据类型，进行byType模式的自动装配。
no	默认值，不使用自动装配，Bean依赖必须通过<ref>元素或ref属性定义。



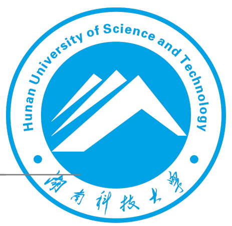
2.6 Bean的生命周期

2.6 Bean的生命周期



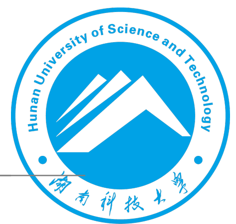
熟悉Bean的生命周期，能够说出Bean的生命周期包含的节点

2.6 Bean的生命周期



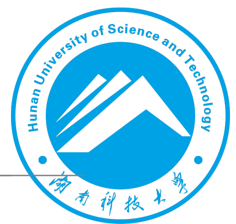
Bean在不同作用域内的生命周期

Bean的生命周期是指Bean实例被创建、初始化和销毁的过程。在Bean的两种作用域singleton和prototype中，Spring容器对Bean的生命周期的管理是不同的。在singleton作用域中，Spring容器可以管理Bean的生命周期，控制着Bean的创建、初始化和销毁。在prototype作用域中， Spring容器只负责创建Bean实例，不会管理其生命周期。



Bean生命周期的两个时间节点

在Bean的生命周期中，有两个时间节点尤为重要，这两个时间节点分别是Bean实例初始化后和Bean实例销毁前，在这两个时间节点通常需要完成一些指定操作。因此，常常需要对这两个节点进行监控。



监控时间节点的方式

监控两个节点的方式有两种，一种是使用XML配置文件，一种是使用注解。

Spring容器提供了@PostConstruct用于监控Bean对象初始化节点，
提供了@PreDestroy用于监控Bean对象销毁节点。下面通过案例演示这两个注解的使用。

2.6 Bean的生命周期

STEP 01

创建Student类，在类中定义id和name字段，并使用@PostConstruct指定初始化方法，使用@PreDestroy指定Bean销毁前的方法。

```
@Component("student")
public class Student {
    @Value("1")
    private String id;
    @Value("张三")
    private String name;        // 省略getter/setter方法，以及toString()方法
    @PostConstruct
    public void init(){System.out.println("Bean的初始化完成，调用init()方法");  }
    @PreDestroy
    public void destroy(){System.out.println("Bean销毁前调用destroy()方法");            }}
```

2.6 Bean的生命周期

STEP 02

创建applicationStudent.xml，在该文件中引入Context约束并启动Bean的自动扫描功能。

```
<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.xsd
      http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- 使用context命名空间,在配置文件中开启相应的注解处理器 -->
    <context:component-scan base-package="com.hnust" />
</beans>
```

2.6 Bean的生命周期

STEP 03

创建测试类StudentTest，在该类中编写测试代码，通过Spring容器加载配置文件并获取Student实例。

```
public class StudentTest {
    public static void main(String[] args){
    ApplicationContext applicationContext=new
            ClassPathXmlApplicationContext("applicationStudent.xml");
    Student student=(Student)applicationContext.getBean("student");
    System.out.println(student);
    //销毁Spring容器中的所有Bean
    AbstractApplicationContext ac=(AbstractApplicationContext)
            applicationContext;
    ac.registerShutdownHook();
}}
```

2.6 Bean的生命周期

STEP 04

在IDEA中启动StudentTest类，控制台会输出结果。

本章小结

本章主要讲解了Spring对Bean的管理。首先介绍了Spring IoC容器，包括BeanFactory接口和ApplicationContext接口；其次讲解了Bean的两种配置方式，包括属性setter方法注入和构造方法注入；接着讲解了Bean 的3种实例化方法，包括构造方法实例化、静态工厂实例化和实例工厂实例化；然后讲解了Bean的作用域，包括singleton作用域和prototype作用域；接着讲解了Bean的3种装配方式，包括基于XML的装配、基于注解的装配和自动装配，最后讲解了Bean的生命周期。通过本章的学习，读者可以对Spring中Bean的管理有基本的了解，为以后框架开发奠定基础。