Spring框架技术

版本：3.6.0-0.0.0

第2章 IoC容器与装配Bean

本章教学内容

Neuedu

节	知识点	掌握程度	难易程度	教学形式	对应微课
Ioc容器	IoC容器概述	理解	普通	线下	IoC容器概述
	IoC容器内部协作	理解	普通	线下	IoC容器内部协作
	BeanFactory与ApplicationContext接口	理解	普通	线下	BeanFactory与ApplicationContext接口
	ApplicationContext接口	掌握	普通	线下	ApplicationContext接口
依赖注入	依赖注入概述	理解	普通	线下	依赖注入概述
	注入参数详解	掌握	普通	线下	注入参数详解
	Bean的生命周期	理解	普通	线下	Bean的生命周期
	自动装配Bean	掌握	普通	线下	自动装配Bean
	Bean作用域	掌握	普通	线下	Bean作用域
	整合多个配置文件	掌握	普通	线下	整合多个配置文件
基于注解的配置	使用注解定义Bean	掌握	普通	线下	使用注解定义Bean
	扫描注解定义的Bean	掌握	普通	线下	扫描注解定义的Bean
	自动装配	掌握	普通	线下	自动装配
基于Java的配置	基于Java类的配置	掌握	普通	线下	基于Java类的配置
	Bean不同配置方式的比较	理解	普通	线下	Bean不同配置方式的比较

本章教学目标

- 理解IoC容器的概念、作用与工作机制；
- 理解BeanFactory与ApplicationContext接口的作用；
- 掌握ApplicationContext接口的使用；
- 理解依赖注入的概念；
- 掌握字面值、其它Bean、集合、Map等属性注入；
- 理解Bean作用域；
- 掌握基于注解的Bean配置；
- 掌握基于Java类的Bean配置；

CONTENTS

目录

IoC容器概述

Neuedu

- IoC(Inverse of Control,控制反转）容器是Spring框架的核心。Spring容器使用DI（依赖注入）管理构成应用的组件，它会创建相互协作的组件之间的联系，AOP、声明式事务处理等功能在此基础之上开花结果。
  - 在基于Spring的应用中，你的应用对象存在于Spring容器(container)中.Spring负责创建对象，装配它，并管理它们的整个生命周期，从生存到死亡(new 到finalize())。
  - IoC字面的意思是控制反转，包括两方面的内容，一个是控制、一个是反转，那到底是什么东西的控制呗反转了？

```
public class UserServiceImpl implements UserService{
    private UserDao userDao;
    public boolean login(String userName, String password) throws Exception{
        //在学习JSP与Servlet时候的调用方式
        //侵入性的依赖调用，使DAO与业务层直接耦合在一起
        userDao = new UserDaoImpl();
        return this.userDao.login(userName, password);
    }
}
```

  - 通过上面的示例问题，对某一接口具体实现类的选择控制权从调用类中移除，转交给Spring容器来进行控制与配置，即控制反转。
- 由于IoC描述不易理解，因此软件界的泰斗级任务MartinFowler提出了DI(Dependency Injection,依赖注入）的概念来代替IoC,即让调用类对某一接口实现类的依赖关系由第三方（容器）注入，以移除调用类对某一接口实现类的依赖，“依赖注入”比“控制反转”更直接明了，易于理解。

IoC容器概述

- 依赖方式比较
- 未使用IoC

```
class A{
    B b = new
B();
}
```

管理

```
class B{
    C c = new
C();
}
```

管理

```
classC{
    D d = new
D();
}
```

- 使用IoC

```
class A{
    B b;
    void setB(B
b)
    { this.b =
b; }
}
```

注入

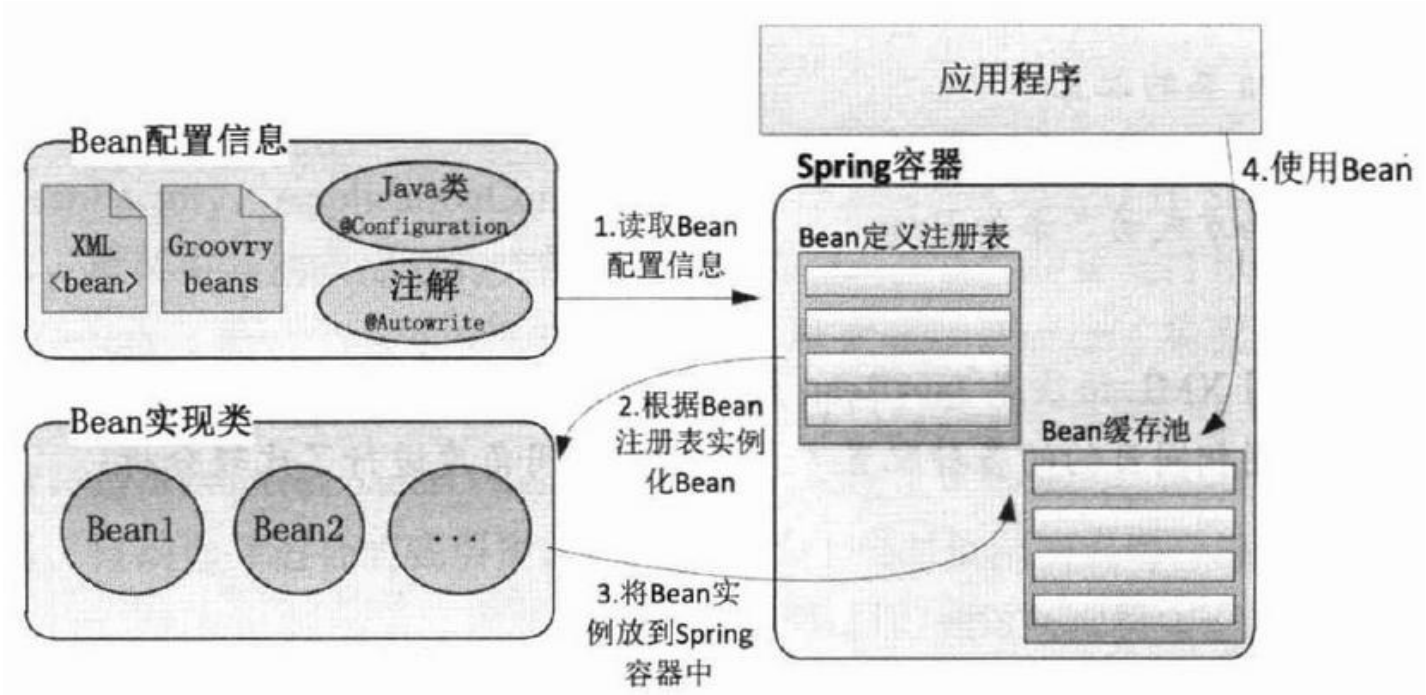
```
class B{
    C c;
    void setC(C
c)
    { this.c =
c; }
}
```

注入

```
class C{
    D d;
    void setD(D
d)
    { this.d =
d; }
}
```

IoC容器内部协作

- Spring容器、Bean配置信息、Bean实现类及应用程序四者的相互关系



BeanFactory与ApplicationContext接口

Neuedu

- Spring通过配置文件描述Bean及Bean之间的依赖关系，利用Java的反射机制实例化Bean并建立Bean之间的依赖关系。Spring的IoC容器在完成这些底层工作的基础上，还提供了Bean实例缓存、生命周期管理、Bean实例代理、事件发布、资源装载等高级服务。
- BeanFactory
  - Bean工厂（org.springframework.beans.factory.BeanFactory）是Spring框架最核心的接口，提供了IoC的配置机制，使管理不同类型的Java对象成为可能。
- ApplicationContext
  - 应用上下文（org.springframework.context.ApplicationContext）建立在BeanFactory基础之上，提供了更多面向应用的功能，比如国际化支持、框架事件体系，更易于创建实际应用。

- 我们一般称BeanFactory为IoC容器,ApplicationContext为应用上下文，BeanFactory是Spring框架的基础设施，面向Spring本身，ApplicationContext面向使用Spring框架的开发者，几乎所有的应用场合都可以直接使用ApplicationContext而非底层的BeanFactory;

ApplicationContext接口

Neuedu

- ApplicationContext是由BeanFactory派生而来，提供了更多面向实际应用的功能，在BeanFactory中很多功能需要以编程的方式实现，而在ApplicationContext中则可以通过配置的方式实现。

- ApplicationContext类体系结构，其主要实现类是以下两个
  - ClassPathXmlApplicationContext
    - 从类路径加载配置文件
  - FileSystemXmlApplicationContext
    - 从文件系统中加载配置文件

```
public class RunTest {
    //运行Main方法，进行测试
    public static void main(String[] args) throws Exception {
        //1.Java配置方式加载
            //通过ApplicationContext启动一个Spring容器，并指定Spring的配置类（AppConfig）
            //ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
        //2.Xml配置方式加载
        ApplicationContext ctx = new ClassPathXmlApplicationContext("AppConfig.xml");

        //获取容器中的Bean对象
        UserService  userService = (UserService)ctx.getBean("userService");
        //调用接口方法
        userService.login("neuedu","123");
    }
}
```

ApplicationContext接口

Neuedu

- BeanFactory
  - 采用延迟加载Bean，直到第一次使用getBean()方法获取Bean实例时，才会创建Bean。
- ApplicationContext
  - ApplicationContext在自身被实例化时一次完成所有Bean的创建。大多数时候使用ApplicationContext。
- 区别
  - 在服务启动时ApplicationContext就会校验XML文件的正确性，不会产生运行时bean装配错误。
  - BeanFactory在服务启动时，不会校验XML文件的正确性，获取bean时，如果装配错误马上就会产生异常。
- 由于我们在第一章的时候使用了Spring提供的SpringJUnit4ClassRunner测试环境，所以后续的学习中，我们将不会直接使用ApplicationContext和BeanFactory接口获取Bean,但原理大家需要明白。

CONTENTS

目录

依赖注入概述

Neuedu

- 依赖注入方式，Spring支持构造函数注入和属性注入，因为通过接口注入需要额外声明一个接口，增加了类数目，而且他的效果和属性注入无本质区别，因为不提倡采用这种注入方式。
  - 构造函数注入
  - 属性注入
  - 接口注入
- 构造函数注入（XML配置方式）
  - 保证一些必要的属性在Bean实例化时就得到设置，确保Bean实例化后就可以使用

```
<bean id="userDao" class="com.neuedu.dao.UserDaoImpl"></bean>

<bean id="userService" class="com.neuedu.service.UserServiceImpl">
    <!-- 构造方法注入 -->
    <constructor-arg name="userDao" ref="userDao"></constructor-arg>
</bean>
```

- 属性注入（XML配置方式）
  - 属性注入方式具有可选择性和灵活性高的有点，在实际应用中最常用

```
<bean id="userService" class="com.neuedu.service.UserServiceImpl">
    <!-- 构造方法注入 -->
    <!-- <constructor-arg name="userDao" ref="userDao"></constructor-arg> -->
    <!--属性注入-->
    <property name="userDao" ref="userDao"></property>
</bean>
```

- 示例代码： Spring-ch02/AppConfig.xml

注入参数详解

Neuedu

- 字面值
  - 字面值一般指可用字符串表示的值，这些值可以通过<value>元素标签进行注入。默认情况下，基本数据类型及其封装类，String等类型都可以采取字面值注入的方式。
  - 处理特殊的XML符号，<![CDATA[]]>的作用是让XML解析器将标签中的字符串当做普通的文本对待，以防特殊字符对XML格式造成破坏。

```
public class UserServiceImpl implements UserService{

    private String token;

    private UserDao userDao;
```

```
<bean id="userService" class="com.neuedu.service.UserServiceImpl">

    <property name="token">
        <value>xx00<![CDATA[&<>]]>xx11</value>
    </property>
</bean>
```

- 示例代码： Spring-ch02/AppConfig.xml、UserServiceImpl.java

注入参数详解

Neuedu

- 引用其它Bean
  - 通过<ref>元素引用其它Bean

```
public class UserServiceImpl implements UserService{

    private String token;

    private UserDao userDao;
```

```
<bean id="userDao" class="com.neuedu.dao.UserDaoImpl"></bean>
<bean id="userService" class="com.neuedu.service.UserServiceImpl">
    <!--属性注入-->
    <property name="userDao">
        <ref bean="userDao"/>
    </property>
```

  - <ref>元素可以通过以下3个属性引用容器中的其他Bean
    - bean：通过该属性可以引用统一容器或父容器的Bean，最常见的形式；
    - parent:指定引用的是父容器的Bean；

```
//父容器
ClassPathXmlApplicationContext pFactory = new ClassPathXmlApplicationContext("bean1.xml");
//子容器
ClassPathXmlApplicationContext factory = new ClassPathXmlApplicationContext(new String[] {"bean2.xml"},pFactory) ;
```

注入参数详解

Neuedu

- null值
  - 为属性设置一个null的注入值,必须使用<null />元素标签

```
<!-- null值注入 -->
<property name="token"><null></null></property>
```

- 级联属性
  - Spring支持级联属性的配置
  - 例如：为userService引入userDao中的id属性注入可以使用 userDao.id
    - <property name=“userDao.id” value=“12334”/>
- 集合类型属性
  - java.util包中的集合类型是最常用的数据结构类型，主要包括:List、Set、Map、Propeties，Spring为这些集合类型属性提供了专属的配置标签。
  - List、Map、Set

```
public class UserServiceImpl implements UserService{

    private String token;
    private UserDao userDao;
    private List<UserDao> listDao = new ArrayList<UserDao>();
    private Map<String, UserDao> mapDao = new HashMap<String, UserDao>();
```

```
<property name="listDao">
    <list>
        <ref bean="userDao"/>
        <ref bean="userDao"/>
        <ref bean="userDao"/>
    </list>
</property>
```

```
<property name="mapDao">
    <map>
        <entry>
            <key><value>u1</value></key>
            <ref bean="userDao"/>
        </entry>
    </map>
</property>
```


自动装配Bean

Neuedu

- <bean元素>提供了一个指定自动装配类型的属性:autowire=”<自动装配类型>”
- Spring提供了4种自动装配类型

自动装配类型	说　明
byName	根据名称进行自动匹配。假设 Boss 有一个名为 car 的属性，如果容器中刚好有一个名为 car 的 Bean，Spring 就会自动将其装配给 Boss 的 car 属性
byType	根据类型进行自动匹配。假设 Boss 有一个 Car 类型的属性，如果容器中刚好有一个 Car 类型的 Bean，Spring 就会自动将其装配给 Boss 的这个属性
constructor	与 ByType 类似，只不过它是针对构造函数注入而言的。如果 Boss 有一个构造函数，构造函数包含一个 Car 类型的入参，如果容器中有一个 Car 类型的 Bean，则 Spring 将自动把这个 Bean 作为 Boss 构造函数的入参；如果容器中没有找到和构造函数入参匹配类型的 Bean，则 Spring 将抛出异常
autodetect	根据 Bean 的自省机制决定采用 byType 还是 constructor 进行自动装配。如果 Bean 提供了默认的构造函数，则采用 byType；否则采用 constructor

- <beans>元素标签中的default-autowire属性可以配置全局自动匹配,default-autowire.属性的默认值为no，表示不启动自动装配。

- 在实际开发中，XML配置方式很少启用自动装配，而基于注解的配置方式默认采用byType自动装配策略。

Bean作用域

Neuedu

- 在配置文件中定义Bean时，不但可以配置Bean的属性值及相互之间的依赖关系，还可以定义Bean的作用域，作用域将对Bean的生命周期和创建方式产生影响。
- Spring支持5中作用域
  - 1、singleton：单例模式，Spring IoC容器中仅存在一个Bean实例。Singleton作用域是Spring中的缺省作用域
    - <bean id="userDao" class="com.ioc.UserDaoImpl" scope="singleton"/>
  - 2、prototype:原型模式，每次从容器获取bean时，容器都将创建一个新的Bean实例。
  - 3、request：每一次Http请求都会创建一个新的Bean，仅在当前Http Request内有效。
  - 4、session：同一个Http Session共享一个Bean,不同的Session请求则会创建新的实例，该bean实例仅在当前Session内有效。
  - 5、global Session：在一个全局的Http Session中，容器会返回该Bean的同一个实例，仅适用于Portlet应用环境。
- 一般我们对有状态的bean使用prototype作用域，而对无状态的bean使用singleton作用域。在Spring环境下对于所有的DAO类都可以采用单例模式，因为Spring利用AOP和LocalThread功能，对非线程安全的变量(或称状态)进行处理，使这些非线程安全的类变成了线程安全的类。

整合多个配置文件

Neuedu

- 对于一个大型应用来说，可能存在多个XML配置文件，在启动Spring容器时，可以通过一个String数组指定多个XML配置文件，可以通过<import>将多个配置文件引入到一个文件中，进行配置文件的集成，这样，在启动Spring容器时，仅需指定这个合并好的配置文件即可。

```
<?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.or
        http://www.springframework.org/schema/context http://www.springframework.
        ">
    <import resource="beans1.xml"/>
    <import resource="beans2.xml"/>
```

- 在实际开发中，每个模块都拥有自己独立的配置文件，应用层面提供一个整合的配置文件，通过import将各个模块整合起来，这样容器在启动时，只需要加载整合后的配置文件即可。

CONTENTS

目录

使用注解定义Bean

Neuedu

- 从Spring2.0开始就引入了注解的配置方式，2.5得到了完善，4.0进一步增强,Spring容器成功启动的三大要件分别是Bean定义信息，Bean实现类及Spring本身。 如果采用基于XML的配置，Bean定义信息和Bean实现类本身是分离的,如果采用基于注解的配置文件，Bean定义信息通过在Bean实现类上标注注解实现。
  - 下面是使用注解定义一个DAO的Bean：

```
@Component("userDao")
public class UserDaoImpl implements UserDao {
    public boolean login(String userName, String password) throws Exception{
        System.out.println(userName + " login.......");
        return true;
    }
}
```

  - @Component注解在UserDao类声明处对类进行标注，可以被Spring容器识别，Spring容器自动将POJO（简单的Java对象）转换为容器管理的Bean,等效于以下XML配置：

```
<bean id="userDao" class="com.neuedu.dao.UserDaoImpl"></bean>
```

- Spring还提供了三个功能基本和@Component等效的注解，分别用于对DAO,Service和Web层的Controller进行注解
①@Repository:用于对DAO实现类进行标注
②@Service:用于对Service类进行标注
③@Controller：用于对Controller实现类进行标注

扫描注解定义的Bean

Neuedu

- Spring提供了一个context命名空间，它提供了通过扫描类包以应用注解定义Bean的方式：

```
<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-4.3.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd
        ">
    <!-- 扫描类包，加载注解定义的Bean-->
    <context:component-scan base-package="com.neuedu"></context:component-scan>
```

  - component-scan的base-package属性指定一个需要扫描的基类包，Spring容器会扫描这个基类包里的所有类，并从类的注解信息中获取Bean的定义信息.
  - 如果仅希望扫描特定的类而非基类包下的所有类，可以使用resource-pattern属性过滤出特定的类.

```
<context:component-scan base-package="com.neuedu" resource-pattern="dao/*"></context:component-scan>
```

  - 默认情况下resource-pattern属性的值为“**/.class”，即基类包中所有的类。上面仅扫描基类包中dao子包中的类。
- 仅需过滤类包中实现了XxxService接口的类或标注了某个特定注解的类时，通过<context:component-scan>的过滤子元素可以实现。
  - <context:include-filter>表示要包含的目标类
  - <context:exclude-filter>表示要排除的目标类。
  - <context:component-scan>下可以有若干个 <context:include-filter>和<context:exclude-filter>元素

自动装配

Neuedu

- 使用@Autowired进行自动注入，Spring通过@Autowired注解实现Bean的依赖注入

```
@Service
public class UserServiceImpl implements UserService{
    @Autowired //注入dao
    private UserDao userDao;
```

  - @Autowired默认按类型(byType)匹配的方式在容器中查找匹配的Bean，当有且只有一个匹配的Bean时，Spring将其注入@Autowired标注的变量中。
  - 如果容器中没有一个和标注变量类匹配的Bean，那么Spring容器启动时将报NoSuchBeanDefinitionException异常。如果希望Spring找不到匹配的Bean完成注入也不要抛出异常，可以使用@Autowired(required=false)进行标注。
  - 默认情况下，@Autowired的required属性值为true，即必须找到匹配的Bean，否则报异常。
- 使用@Qualifier指定注入Bean 的名称
  - 如果容器中有一个以上匹配的Bean时，可以通过@Qualifier注解限定Bean的名称

```
@Service
public class UserServiceImpl implements UserService{
    @Autowired
    @Qualifier("otherUserDao") //注入名为 otherUserDao 的UserDao类型的Bean
    private UserDao userDao;
```

自动装配

Neuedu

- 对类方法进行标注
  - @Autowired可以对类成员变量及方法的入参进行标注
- 对类集合进行标注
  - 如果对类中集合类的变量或方法入参进行@Autowired标注，那么Spring会将容器中类型匹配的所有Bean都自动注入进来，默认情况下，加载顺序是不确定，可以通过@Order注解或实现Ordered接口来决定Bean加载顺序，值越小，优先被加载。
- 对延迟加载注入的支持
  - Spring4.0以后支持延迟依赖注入
  - Spring支持延迟依赖注入，在Spring容器启动的时候，对于在Bean上标注@Lazy及@Autowired注解的属性，不会立即注入属性值，而是延迟到调用此属性的时候才注入属性值。
  - 对Bean实施延迟依赖注入，注意@Lazy注解必须同时标注在属性及目标Bean上，否则无效。

```
@Lazy
@Component("otherUserDao")
public class UserDaoImpl implements UserDao {
```

```
@Lazy
@Autowired
@Qualifier("otherUserDao") //注入名为 otherUserDao 的UserDao类型的Bean
private UserDao userDao;
```

- Bean作用范围
  - 通过注解配置的Bean和通过<bean>配置的Bean一样，默认的作用范围都singleton
  - Spring为注解配置提供了一个Scope注解，可以通过它显式指定Bean的作用范围。

```
@Lazy
@Scope("prototype")
@Component("otherUserDao")
public class UserDaoImpl implements UserDao {
```


CONTENTS

目录

基于Java类的配置

Neuedu

- 使用基于Java类的配置信息启动Spring容器
  - Srping提供了一个AnnotationConfigApplicationContext类，能直接通过标注@Configuration的Java类启动Spring容器

```
public void testLogin() throws Exception {
    ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
    //获取容器中的Bean对象
    UserService  userService = ctx.getBean(UserService.class);
    //调用接口方法
    userService.login("neuedu","123");
}
```

- 通过XML配置引用@Configuration的配置
  - 标注了@Configuration和标注了@Component的类一样是一个Bean，可以被Spring的<context:component-scan>扫描到，因此，如果希望将配置类组装到XML配置文件中，通过XML配置文件启动Spring容器，仅需在XML通过<context:component-scan>扫描到相应的配置类即可。
- 通过@Configuration配置类引用ＸＭＬ配置信息
  - 在@Configuration配置类中可以通过@ImportResource引入XML配置文件，在LogonAppConfig配置类中即可直接通过@Autowired引用XML配置文件中定义的Bean。

使用Java类提供Bean定义信息

Neuedu

- JavaConfig是Spring的一个子项目，目的通过Java类的方式提供Bean的定义信息，目前成为Spring4.0的核心功能。
- 普通的POJO只要标注@Configuration注解，就可以为Spring容器提供Bean定义的信息，每个标注了@Bean的类方法都相当于提供了一个Bean的定义信息。

```
@Configuration
public class AppConfig {
    @Bean
    public UserDao userDao() {
        return new UserDaoImpl();
    }
    @Bean
    public UserService userService() {
        UserServiceImpl userService = new UserServiceImpl();
        //配置依赖关系
        userService.setUserDao(userDao());
        return userService;
    }
}
```

- 上面的代码等价于：

```
<bean id="userDao" class="com.neuedu.dao.UserDaoImpl"></bean>
<bean id="userService" class="com.neuedu.service.UserServiceImpl">
    <property name="userDao"><ref bean="userDao"/></property>
```

- 示例代码：Spring-ch2-java/AppConfig、UserServiceImplTest

Bean不同配置方式的比较

Neuedu

- Spring为实现Bean信息定义，它提供了基于XML、基于注解、基于Java类这几种配置方式，同时还允许各种配置方式复合并存，让彼此可以互通有无、取长补短，对于每种配置很难说孰优孰劣，只能说他们有不同的使用场景，下面给出一些参考意见。

- 基于XML配置：
  - Bean 实现类来源于第三方类库，如DataSource、SessionFacotry等资源Bean，因无法在类中标注注解，所以通过XML配置较好
  - 命名空间的配置，如aop、context等，智能采用基于XML的配置。
- 基于注解配置
  - Bean的实现类是当前项目开发的，可以直接在Java类中使用基于注解的配置。
- 基于Java类配置
  - 基于Java类配置的优势在于可以通过代码方式控制Bean初始化的整体逻辑。如果实例化Bean的逻辑比较复杂，则比较适合基于Java类配置的方式。

- 建议整个项目采用“基于XML + 基于注解”配置方式。

本章重点总结

- 依赖注入的概念及原理；
- 掌握字面值、其它Bean、集合、Map等属性注入；
- Bean作用域；
- 基于注解的Bean配置；
- 基于Java类的Bean配置；

课后作业

- 1、写出依赖注入的方式有哪些？
- 2、写出IoC和DI的关系是什么？
- 3、写出三个可以Bean组件上配置的注解。
- 4、Bean有哪几种作用域？

Neuedu