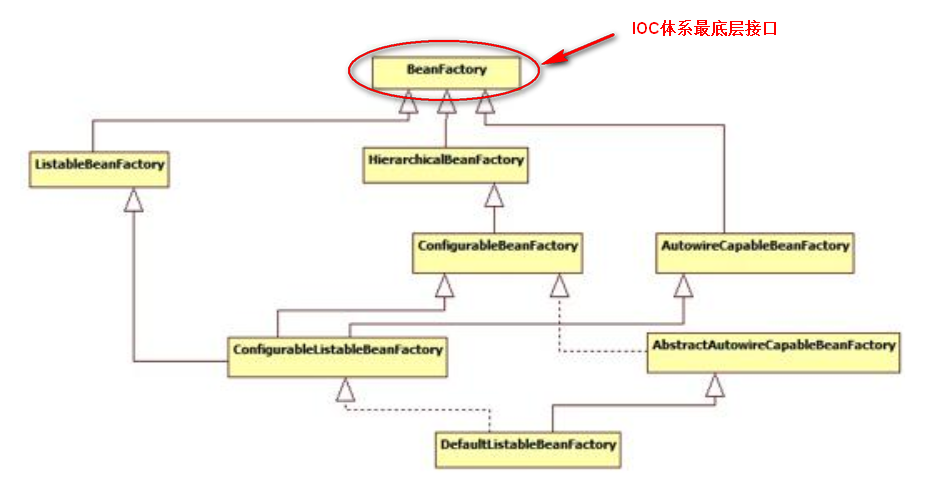
# 1. IOC体系



|  |
| --- |
| **public** **interface** BeanFactory {  //对FactoryBeand的转义定义，因为如果使用  String ***FACTORY\_BEAN\_PREFIX*** = "&";  Object getBean(String name) **throws** BeansException;  <T> T getBean(String name, Class<T> requiredType) **throws** BeansException;  <T> T getBean(Class<T> requiredType) **throws** BeansException;  Object getBean(String name, Object... args) **throws** BeansException;  <T> T getBean(Class<T> requiredType, Object... args) **throws** BeansException;  **boolean** containsBean(String name);  **boolean** isSingleton(String name) **throws** NoSuchBeanDefinitionException;  **boolean** isPrototype(String name) **throws** NoSuchBeanDefinitionException;  **boolean** isTypeMatch(String name, Class<?> targetType) **throws** NoSuchBeanDefinitionException;  Class<?> getType(String name) **throws** NoSuchBeanDefinitionException;  String[] getAliases(String name);  } |

# 2. Spring框架

## 2.1 专业术语了解

**组件/框架设计**

侵入式设计

引入了框架，对现有的类的结构有影响；即需要实现或继承某些特定类。

例如： Struts框架

非侵入式设计

引入了框架，对现有的类结构没有影响。

例如：Hibernate框架 / Spring框架

**控制反转:**

Inversion on Control , 控制反转 IOC

对象的创建交给外部容器完成，这个就做控制反转.

依赖注入， dependency injection

处理对象的依赖关系

区别：

控制反转， 解决对象创建的问题 【对象创建交给别人】

依赖注入，

在创建完对象后， 对象的关系的处理就是依赖注入 【通过set方法依赖注入】

**AOP**

面向切面编程。切面，简单来说来可以理解为一个类，由很多重复代码形成的类。

切面举例：事务、日志、权限;

## 2.2 Spring框架

### a. 概述

Spring框架，可以解决对象创建以及对象之间依赖关系的一种框架。

且可以和其他框架一起使用；Spring与Struts, Spring与hibernate

(起到整合（粘合）作用的一个框架)

Spring提供了一站式解决方案：

1） Spring Core spring的核心功能： IOC容器, 解决对象创建及依赖关系

2） Spring Web Spring对web模块的支持。

-🡪 可以与struts整合,让struts的action创建交给spring

-🡪 spring mvc模式

3） Spring DAO Spring 对jdbc操作的支持 【JdbcTemplate模板工具类】

4） Spring ORM spring对orm的支持：

🡪 既可以与hibernate整合，【session】

🡪 也可以使用spring的对hibernate操作的封装

5）Spring AOP 切面编程

6）SpringEE spring 对javaEE其他模块的支持

### b. 开发步骤

spring各个版本中：

在3.0以下的版本，源码有spring中相关的所有包【spring功能 + 依赖包】

如2.5版本；

在3.0以上的版本，源码中只有spring的核心功能包【没有依赖包】

(如果要用依赖包，需要单独下载！)

1） 源码, jar文件：spring-framework-3.2.5.RELEASE

commons-logging-1.1.3.jar 日志

spring-beans-3.2.5.RELEASE.jar bean节点

spring-context-3.2.5.RELEASE.jar spring上下文节点

spring-core-3.2.5.RELEASE.jar spring核心功能

spring-expression-3.2.5.RELEASE.jar spring表达式相关表

以上是必须引入的5个jar文件，在项目中可以用户库管理！

2） 核心配置文件: applicationContext.xml

Spring配置文件：applicationContext.xml / bean.xml

约束参考：

spring-framework-3.2.5.RELEASE\docs\spring-framework-reference\htmlsingle\index.html

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>    </beans> |

3）Api

|  |
| --- |
| **public** **class** App {  // 1. 通过工厂类得到IOC容器创建的对象  @Test  **public** **void** testIOC() **throws** Exception {  // 创建对象  // User user = new User();    // 现在，把对象的创建交给spring的IOC容器  Resource resource = **new** ClassPathResource("cn/itcast/a\_hello/applicationContext.xml");  // 创建容器对象(Bean的工厂), IOC容器 = 工厂类 + applicationContext.xml  BeanFactory factory = **new** XmlBeanFactory(resource);  // 得到容器创建的对象  User user = (User) factory.getBean("user");    System.*out*.println(user.getId());    }    //2. （方便）直接得到IOC容器对象  @Test  **public** **void** testAc() **throws** Exception {  // 得到IOC容器对象  ApplicationContext ac = **new** ClassPathXmlApplicationContext("cn/itcast/a\_hello/applicationContext.xml");  // 从容器中获取bean  User user = (User) ac.getBean("user");    System.*out*.println(user);  }  } |

## 2.3. bean对象创建的细节

|  |
| --- |
| /\*\*  \* 1) 对象创建： 单例/多例  \* scope="singleton", 默认值， 即 默认是单例 【service/dao/工具类】  \* scope="prototype", 多例； 【Action对象】  \*  \* 2) 什么时候创建?  \* scope="prototype" 在用到对象的时候，才创建对象。  \* scope="singleton" 在启动(容器初始化之前)， 就已经创建了bean，且整个应用只有一个。  \* 3)是否延迟创建  \* lazy-init="false" 默认为false, 不延迟创建，即在启动时候就创建对象  \* lazy-init="true" 延迟初始化， 在用到对象的时候才创建对象  \* （只对单例有效）  \* 4) 创建对象之后，初始化/销毁  \* init-method="init\_user" 【对应对象的init\_user方法，在对象创建爱之后执行 】  \* destroy-method="destroy\_user" 【在调用容器对象的destriy方法时候执行，(容器用实现类)】  \*/  @Test  **public** **void** testIOC() **throws** Exception {  // 得到IOC容器对象 【用实现类，因为要调用销毁的方法】  ClassPathXmlApplicationContext ac = **new** ClassPathXmlApplicationContext("cn/itcast/a\_hello/applicationContext.xml");  System.*out*.println("-----容器创建-----");    // 从容器中获取bean  User user1 = (User) ac.getBean("user");  User user2 = (User) ac.getBean("user");    System.*out*.println(user1);  System.*out*.println(user2);    // 销毁容器对象  ac.destroy();  } |

## 2.4. SpringIOC容器

### 1） 创建对象

SpringIOC容器，是spring核心内容。

作用： 创建对象 & 处理对象的依赖关系

IOC容器创建对象：

创建对象, 有几种方式：

1） 调用无参数构造器

2） 带参数构造器

3） 工厂创建对象

工厂类，静态方法创建对象

工厂类，非静态方法创建对象

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>    <!-- ###############对象创建############### -->    <!-- 1. 默认无参数构造器  <bean id="user1" class="cn.itcast.b\_create\_obj.User"></bean>  -->    <!-- 2. 带参数构造器 -->  <bean id=*"user2"* class=*"cn.itcast.b\_create\_obj.User"*>  <constructor-arg index=*"0"* type=*"int"* value=*"100"*></constructor-arg>  <constructor-arg index=*"1"* type=*"java.lang.String"* value=*"Jack"*></constructor-arg>  </bean>    <!-- 定义一个字符串，值是"Jack" ; String s = new String("jack")-->  <bean id=*"str"* class=*"java.lang.String"*>  <constructor-arg value=*"Jacks"*></constructor-arg>  </bean>  <bean id=*"user3"* class=*"cn.itcast.b\_create\_obj.User"*>  <constructor-arg index=*"0"* type=*"int"* value=*"100"*></constructor-arg>  <constructor-arg index=*"1"* type=*"java.lang.String"* ref=*"str"*></constructor-arg>  </bean>      <!-- 3. 工厂类创建对象 -->  <!-- # 3.1 工厂类，实例方法 -->  <!-- 先创建工厂 -->  <bean id=*"factory"* class=*"cn.itcast.b\_create\_obj.ObjectFactory"*></bean>  <!-- 在创建user对象，用factory方的实例方法 -->  <bean id=*"user4"* factory-bean=*"factory"* factory-method=*"getInstance"*></bean>      <!-- # 3.2 工厂类： 静态方法 -->  <!--  class 指定的就是工厂类型  factory-method 一定是工厂里面的“静态方法”  -->  <bean id=*"user"* class=*"cn.itcast.b\_create\_obj.ObjectFactory"* factory-method=*"getStaticInstance"*></bean>    </beans> |

### 2） 对象依赖关系

Spring中，如何给对象的属性赋值? 【DI, 依赖注入】

1) 通过构造函数

2) 通过set方法给属性注入值

3) p名称空间

4)自动装配(了解)

5) 注解

#### # (常用)Set方法注入值

|  |
| --- |
| <!-- dao instance -->  <bean id=*"userDao"* class=*"cn.itcast.c\_property.UserDao"*></bean>  <!-- service instance -->  <bean id=*"userService"* class=*"cn.itcast.c\_property.UserService"*>  <property name=*"userDao"* ref=*"userDao"*></property>  </bean>    <!-- action instance -->  <bean id=*"userAction"* class=*"cn.itcast.c\_property.UserAction"*>  <property name=*"userService"* ref=*"userService"*></property>  </bean> |

#### # 内部bean

|  |
| --- |
| <!-- ##############内部bean############## -->  <bean id=*"userAction"* class=*"cn.itcast.c\_property.UserAction"*>  <property name=*"userService"*>  <bean class=*"cn.itcast.c\_property.UserService"*>  <property name=*"userDao"*>  <bean class=*"cn.itcast.c\_property.UserDao"*></bean>  </property>  </bean>  </property>  </bean> |

#### # p 名称空间注入属性值 (优化)

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>    <!-- ###############对象属性赋值############### -->    <!--  给对象属性注入值：  # p 名称空间给对象的属性注入值  (spring3.0以上版本才支持)  -->  <bean id=*"userDao"* class=*"cn.itcast.c\_property.UserDao"*></bean>    <bean id=*"userService"* class=*"cn.itcast.c\_property.UserService"* p:userDao-ref=*"userDao"*></bean>    <bean id=*"userAction"* class=*"cn.itcast.c\_property.UserAction"* p:userService-ref=*"userService"*></bean>      <!-- 传统的注入：  <bean id="user" class="cn.itcast.c\_property.User" >  <property name="name" value="xxx"></property>  </bean>  -->  <!-- p名称空间优化后 -->  <bean id=*"user"* class=*"cn.itcast.c\_property.User"* p:name=*"Jack0001"*></bean>    </beans> |

#### # 自动装配(了解)

* **根据名称自动装配：autowire="byName"**

**-🡪 自动去IOC容器中找与属性名同名的引用的对象，并自动注入**

|  |
| --- |
| <!-- ###############自动装配############### -->  <bean id=*"userDao"* class=*"cn.itcast.d\_auto.UserDao"*></bean>  <bean id=*"userService"* class=*"cn.itcast.d\_auto.UserService"* autowire=*"byName"*></bean>  <!-- 根据“名称”自动装配： userAction注入的属性，会去ioc容器中自动查找与属性同名的对象 -->  <bean id=*"userAction"*  class=*"cn.itcast.d\_auto.UserAction"* autowire=*"byName"*></bean> |

**也可以定义到全局， 这样就不用每个bean节点都去写autowire=”byName”**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"* **default-autowire=*"byName"*> 根据名称自动装配（全局）**    <!-- ###############自动装配############### -->  <bean id=*"userDao"* class=*"cn.itcast.d\_auto.UserDao"*></bean>  <bean id=*"userService"* class=*"cn.itcast.d\_auto.UserService"*></bean>  <bean id=*"userAction"* class=*"cn.itcast.d\_auto.UserAction"*></bean>  </beans> |

* **根据类型自动装配：autowire="byType"**

**必须确保改类型在IOC容器中只有一个对象；否则报错。**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"* default-autowire=*"byType"*>    <!-- ###############自动装配############### -->  <bean id=*"userDao"* class=*"cn.itcast.d\_auto.UserDao"*></bean>  <bean id=*"userService"* class=*"cn.itcast.d\_auto.UserService"*></bean>    <!-- 如果根据类型自动装配： 必须确保IOC容器中只有一个该类型的对象 -->  <bean id=*"userAction"* class=*"cn.itcast.d\_auto.UserAction"*></bean>      <!-- 报错： 因为上面已经有一个该类型的对象，且使用了根据类型自动装配  <bean id="userService\_test" class="cn.itcast.d\_auto.UserService" autowire="byType"></bean>  -->  </beans> |

**总结：**

**Spring提供的自动装配主要是为了简化配置； 但是不利于后期的维护。**

**(一般不推荐使用)**

#### # 注解

注解方式可以简化spring的IOC容器的配置!

使用注解步骤：

1）先引入context名称空间

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

2）开启注解扫描

<context:component-scan base-package="cn.itcast.e\_anno2"></context:component-scan>

3）使用注解

通过注解的方式，把对象加入ioc容器。

创建对象以及处理对象依赖关系，相关的注解：

@Component 指定把一个对象加入IOC容器

@Repository 作用同@Component； 在持久层使用

@Service 作用同@Component； 在业务逻辑层使用

@Controller 作用同@Component； 在控制层使用

@Resource 属性注入

总结：

1） 使用注解，可以简化配置，且可以把对象加入IOC容器,及处理依赖关系(DI)  
 2） 注解可以和XML配置一起使用。

# 3. Spring与Struts框架整合

Spring，负责对象对象创建

Struts， 用Action处理请求

Spring与Struts框架整合，

关键点：让struts框架action对象的创建，交给spring完成！

Spring与Hibernate整合：

【SSH整合:

Spring与Struts

关键点： action交给spring创建!

Spring与Hibernate

关键点： sessionFactory对象交给spring创建！

】

**步骤：**

引入jar文件

1）引入struts .jar相关文件

2）spring-core 相关jar文件

3）spring-web 支持jar包

spring-web-3.2.5.RELEASE.jar 【Spring源码】

struts2-spring-plugin-2.3.4.1.jar 【Struts源码】

配置:

4）配置XML

struts.xml 【struts路径与action映射配置】

bean.xml 【spring ioc容器配置】

web.xml

【核心过滤器： 引入struts功能】

【初始化spring的ioc容器】

【初始化spring的ioc容器】

查看api。

4） 代码以及配置