框架概述：

Struts2: 基于mvc模式的应用层框架！

主要是作为控制层组件，javabean， jsp！

Hibernate: 持久层组件，简化jdbc操作！

Spring: 主要有六大模块功能

事务管理！

与其他框架整合！

Mvc开发模式：

基于mvc模式的项目分层!

# 1. Spring框架

1. 框架理解

框架，都会预先实现一些功能，给开发人员开发提供便利！

提高开发效率、提升程序可扩展性、健壮！

1. 专业术语

(1)高内聚、低耦合：

类内部的关系越紧密越好，类与类的关系越少越好！

(2)非侵入式设计：

1)侵入式设计：

引入的组件对现有的类的结构会有影响，这种就是“侵入式设计”的组件！

Struts2: 侵入式设计！

2)非侵入式设计：

引入组件对现有的类的结构没有影响！

Hibernate: 非侵入式设计！

Spring, 非侵入式设计

(3)IOC 容器:Inversion of control 控制反转

控制反转容器! -🡪 对象创建的问题！

解释：

User user = new User(); 自己控制对象的创建

现在需要对象，自己不创建，交给外部的容器创建，叫控制反转！

IOC容器= bean.xml配置 + ApplicationContext容器类

(4)DI, dependency injection 依赖注入

创建对象后，处理对象的依赖关系！

User user = new User();

user.setAddress(..) ;// 需要DI（依赖注入）

(5)Aop…

1. Spring框架

Spring 开源框架，提供的是一站式解决方案！

六大模块:

…..

1. Spring 第一个案例， 解决项目中如何创建对象

(1) 引入jar文件 (3.2版本)

commons-logging-1.1.3.jar 【单独下载】

spring-beans-3.2.5.RELEASE.jar 【spring源码, bean节点管理】

spring-context-3.2.5.RELEASE.jar 【spring上下文类】

spring-core-3.2.5.RELEASE.jar 【IOC容器】

spring-expression-3.2.5.RELEASE.jar 【spring表达式】

注意：使用的版本Spring3.2

在这个版本中，只有spring自身的核心功能，spring依赖的其他组件，需要单独下载！ 例如：日志jar文件，就需要单独下载！

(2) 新建applicationContext.xml , 源码中拷贝约束

|  |
| --- |
| <?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.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd">  </beans> |

(3) 配置

|  |
| --- |
| <!-- 创建Dao实例 -->  <bean id="userDao" class="cn.itcast.dao.UserDao"></bean>  <!-- 创建Service实例 -->  <bean id="userService" class="cn.itcast.service.UserService">  <property name="userDao" ref="userDao"></property>  </bean>    <!-- 创建Action实例 -->  <bean id="userAction" class="cn.itcast.action.UserAction">  <property name="userService" ref="userService"></property>  </bean> |

(4) UserDao.java

|  |
| --- |
| package cn.itcast.dao;  public class UserDao {  public void save() {  System.out.println("UserDao.save()");  }  } |

(5) UserService.java

|  |
| --- |
| package cn.itcast.service;  import cn.itcast.dao.UserDao;  public class UserService {  // 创建dao对象 : 单例，启动创建  private UserDao userDao;  public void setUserDao(UserDao userDao) {  this.userDao = userDao;  }  public void save() {  userDao.save();  }  } |

(6) UserAction.java

|  |
| --- |
| package cn.itcast.action;  import cn.itcast.service.UserService;  /\*\*  \* 多例Action实例：在访问的时候创建对象！  \* @heyutong  \* 2017年6月2日  \*/  public class UserAction {  // 判断一个类是单例还是多例，主要看有没有维护成员变量、且对象成员变量进行修  改！ 如果有，这个类就应该是多例！  // 创建Service:单例，启动时候创建  private UserService userService;//= new UserService();  // 提供set方法，给外部容器注入  public void setUserService(UserService userService) {  this.userService = userService;  }    public String execute() {  userService.save();  return "success";  }  } |

(4) 测试

|  |
| --- |
| package junit.test;  import org.junit.Test;  import org.springframework.context.support.ClassPathXmlApplicationContext;  import cn.itcast.action.UserAction;  public class ActionTest {  //不从IOC获取对象  //报空指针异常  /\*@Test  public void test1() {  UserAction userAction = new UserAction();  userAction.execute();  }\*/    //从IOC容器获取对象  @Test  public void test2() {  // 容器对象(加载applicationContext.xml配置文件)  //调用了setUserDao和setUserService方法  ClassPathXmlApplicationContext ac = new  ClassPathXmlApplicationContext("applicationContext.xml");  // 获取对象  UserAction userAction = (UserAction)ac.getBean("userAction");  userAction.execute();  }  } |

# 2. Spring IOC 容器

## 2.1创建bean细节：

1. bean在容器中的写法

2. bean声明周期

|  |
| --- |
| <?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.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>  <!-- 把对象加入IOC容器 -->  <!--  细节1：  id 与name：  id 不能以数字开头，不能含有特殊符号, 不能有空格、逗号等； id 不能重复！  name 可以数字开头，可以有特殊符号，如果name值重复，编译没问题但运行报错！  <bean id="user" name="user2,user3 user4" class="cn.itcast.a\_config.User"></bean>  <bean name="user5" class="cn.itcast.a\_config.User"></bean>  <bean id="user6" class="cn.itcast.a\_config.User"></bean>  <bean id="user6" class="cn.itcast.a\_config.User"></bean>  -->    <!-- 细节2： (单例/多例) -->  <!--  scope="singleton" 默认表示单例！  prototype 多例  init-method="" 在创建完对象之后执行初始化方法  destroy-method="" 在调用容器类的destroy()方法时候，对单例的对象有效！  lazy-init="true" 延迟初始化/设置只对单例有影响，对多例没有任何影响！  单例对象默认是在创建容器时创建所有单例对象，如希望在第一次访问时创建单例  对象，就设置延迟初始化  Bean生命周期：  bean在xml中配置，  1.singleton 单例  1) 创建对象  如果有配置延迟初始化，  lazy-init=true 如单例对象有配置延迟初始化， 在创建容器之后，在第一次从容器  获取对象时创建单例的对象！  如没有配置或延迟初始化为默认值, 单例对象会在创建容器时创建对象  2) 执行初始化方法 , init-method配置的方法会执行  3) 调用容器destroy() 方法时，容器在销毁单例对象实例时，会调用destroy-method  对应的方法 此时bean对象会被销毁！    2.prototype 多例  1) 每次从容器获取对象时，都会创建新的对象  2) 每次创建完对象后，就执行初始化方法  3) java回回收不用资源(jvm gc)  -->  <bean id=*"user"*  class=*"cn.itcast.a\_config.User"*  init-method=*"init"*  destroy-method=*"destroy\_"*  lazy-init=*"false"*  ~~scope=~~*~~"prototype"~~*>  </bean>  </beans> |

3. User.java

|  |
| --- |
| package cn.itcast.a\_config;  public class User {  private int id;  private String name;    public User() {  System.out.println("User.User(), 创建User对象...");  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }    //初始化时候调用  public void init() {  System.out.println("初始化...");  }    //销毁对象实例时候调用  public void destroy\_() {  System.out.println("销毁...");  }  } |

4. App.java

|  |
| --- |
| package cn.itcast.a\_config;  import org.junit.Test;  import org.springframework.context.support.ClassPathXmlApplicationContext;  public class App {  @Test  public void test() {  ClassPathXmlApplicationContext ac = new  ClassPathXmlApplicationContext("cn/itcast/a\_config/bean.xml");  System.out.println("----------");    //获取/创建对象  User user = (User) ac.getBean("user");  System.out.println(user);  User user2 = (User) ac.getBean("user");  System.out.println(user2);  // 销毁容器实例  ac.destroy();  }  } |

1. 运行测试结果:
2. 单例：未配置scope=*"prototype"*
3. new ClassPathXmlApplicationContext("cn/itcast/a\_config/bean.xml");时，打印:

User.User(), 创建User对象... //new User对象

初始化... //执行init-method="init"

2)user 和user2 对象一样，例如都是cn.itcast.a\_config.User@1147d1a

3) 销毁容器实例,即执行ac.destroy();时执行destroy-method="destroy\_",打印:

销毁...

1. 多例: 配置scope=*"prototype"*
2. User user = (User) ac.getBean("user");时,打印:

|  |
| --- |
| User.User(), 创建User对象... //创建一个User对象  初始化... //执行init-method="init"  cn.itcast.a\_config.User@111cccd |

1. User user2 = (User) ac.getBean("user");时,打印:

|  |
| --- |
| User.User(), 创建User对象... //创建另一个User对象  初始化... //执行init-method="init"  cn.itcast.a\_config.User@18441c1 |

3) 销毁容器实例,即执行ac.destroy();时不执行destroy-method="destroy\_",

## 2.2创建对象的几种方式

**2.2.1.调用无参数构造器**

**2.2.2.调用有参数构造器**

1.bean.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"  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">  <!--IOC容器， 创建对象 -->  <!—1) 调用无参数构造器 -->  <bean id="user1" class="cn.itcast.b\_create\_obj.User"></bean>    <!—2) 调用有参数构造器, 创建对象 -->  <!-- String str = new String("Jack"); -->  <bean id="str" class="java.lang.String">  <constructor-arg value="Jack"></constructor-arg>  </bean>    <!-- 创建user对象，且赋值 -->  <!--  value 当直接给属性值的时候使用value赋值  ref 当引用的是IOC容器中的对象的时候，使用ref  -->  <bean id="user" class="cn.itcast.b\_create\_obj.User">  <constructor-arg index="0" type="int" value="10000"></constructor-arg>  <constructor-arg index="1" type="String" ref="str"></constructor-arg>  </bean>  </beans> |

**2.** User.java

|  |
| --- |
| package cn.itcast.b\_create\_obj;  public class User {  private int id;  private String name;    public User() {  System.out.println("User.User(), 创建User对象...");  }  public User(int id, String name) {  super();  this.id = id;  this.name = name;  }  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  @Override  public String toString() {  return "User [id=" + id + ", name=" + name + "]";  }    // 初始化时候调用  public void init() {  System.out.println("初始化...");  }    // 销毁对象实例时候调用  public void destroy\_() {  System.out.println("销毁...");  }  } |

**3.** App.java

|  |
| --- |
| package cn.itcast.b\_create\_obj;  import org.junit.Test;  import org.springframework.context.support.ClassPathXmlApplicationContext;  public class App {  @Test  public void test() {  // 方式1：创建容器对象  /\*ClassPathXmlApplicationContext ac = new ClassPathXmlApplicationContext("cn/itcast/b\_create\_obj/bean.xml");//打印:User.User(), 创建User对象...  User user = (User) ac.getBean("user1");  System.out.println(user);\*/    // 方式2： 会从当前类所在的包下找bean.xml 【测试方便】  /\*ClassPathXmlApplicationContext ac = new  ClassPathXmlApplicationContext("bean.xml", this.getClass());  //打印:User.User(), 创建User对象...  String str = (String)ac.getBean("str");  System.out.println(str);//打印:Jack\*/    // 方式2： 会从当前类所在的包下找bean.xml 【测试方便】  ClassPathXmlApplicationContext ac = new  ClassPathXmlApplicationContext("bean.xml", this.getClass());  //打印:User.User(), 创建User对象...  User user = (User) ac.getBean("user");  System.out.println(user);//打印:User [id=10000, name=Jack]  }  } |

**2.2.3. 工厂**

1.nean2.xml

|  |
| --- |
|  |

\* 静态方法

<bean id=”’ class=”” factory-method=””/>

\* 非静态方法

<bean id=”factory” class=”..”>

<bean id=”” factory-bean=” factory” factory-method=”实例方法” />

**2.24. 反射**

## 2.3 处理对象的依赖关系

给对象属性赋值(DI, 依赖注入)，几种方式：

1) 构造函数赋值

2) set 方法注入值

\* 普通字段赋值

\* 集合属性 (list/map/property)

3) 案例

Dao/service/action实例，处理依赖关系

\* 常用的通过set方法注入

\* 内部bean

\* p 名称空间

\* 自动装配

\* 注解

注解总结：

@Component 表示一个组件(类)，把当前组件加入ioc容器

加入容器的组件的名称默认是类名第一个字母小写

@Component(“”) 指定加入ioc容器的组件类的类名

@Repository 标识是一个持久层的组件

@Service 标识是一个业务逻辑层的组件

@Controller 标识是一个控制层的组件

@Scope("prototype") 指定对象单例/多例

@Resource 1. 默认根据修饰的字段名称会取ioc容器找对象自动注入

找到后注入

2. 如果名称没有找到，再根据类型查找 找到后就立刻注入

如果改类型在ioc容器中有多个对象，报错！

3. 根据类型也没有找到对象，报错！

@Resource(name =””) 会根据指定的名称去容器找对象自动注入

配置方式与注解方式:

1. 配置， 便于维护

(配置过多，比较繁琐)

2. 注解， 开发方便

(简化配置，不利于后期维护，如果修改对象创建、关系处理，需要改代码！)

# 3. Spring 与Struts整合

Spring与struts整合，关键点:

把action的创建，交给spring的ioc容器！

所以，需要引入jar文件：

**struts2-spring-plugin-2.3.4.1.jar 【struts源码】**

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

整合完整步骤：

1. 引入jar

Struts核心jar

Spring

Core 核心 （5个）

Web 对web支持 (2个)

2. 配置

bean.xml

struts.xml

web.xml

struts2核心过滤器

启动时候，配置加载springIOC容器

3. Action 类

任务：

1. 写简单案例，练习spring 、

\* 搭建spring环境

\* 测试OK

\* dao/service/action

然后处理对象依赖关系，

\* set方法注入值, 几种方式

\* 注解

2. spring与struts整合，小案例

3. 优化前面项目，

融入spring功能！