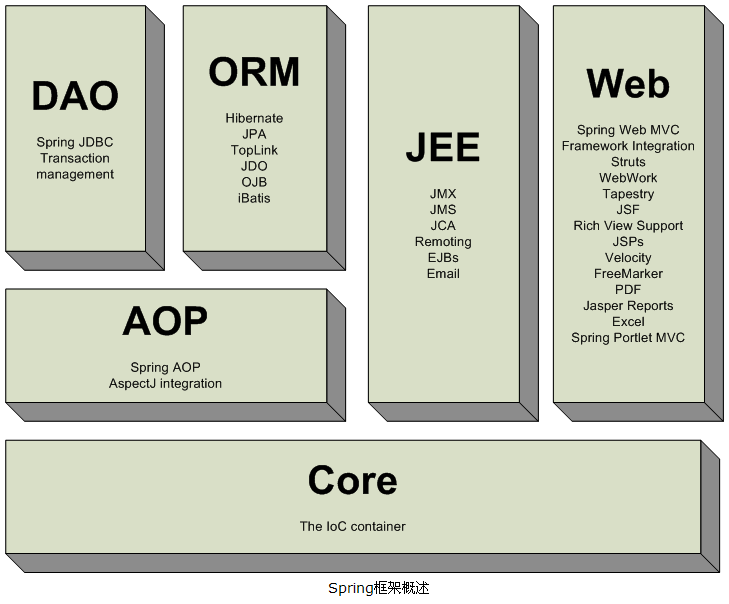
**spring框架概述**



# 配置文件详解

|  |
| --- |
| <?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:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  **xsi:schemaLocation**=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-2.0.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-2.0.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx-2.0.xsd"*>  <bean id=*"userDao4MySqlImpl"* class=*"com.bjsxt.spring.dao.UserDao4MySqlImpl"* />  <bean id=*"userDao4OracleImpl"* class=*"com.bjsxt.spring.dao.UserDao4OracleImpl"* />  <bean id=*"userManager"* class=*"com.bjsxt.spring.manager.UserManagerImpl"*>  <!-- **构造方法注入**: 在UserManagerImpl中提供构造函数，spring将实例化好的UserDao实现注入给我们  <**constructor-arg** **ref**="userDao4OracleImpl"/>-->  <!-- **set方法注入**: 当\*\*比较多时, 使用set方法比较容易理解. -->  <**property** **name**=*"bean的属性名"* **ref**=*"指定bean的一个依赖: 填写容器中bean名"* />  <!-- value属性(有ref就不能又value): value="基本数据类型" 当value为空: <value></value> 等同于 <null/>  </bean>  </beans> |

# IOC(Inversion of Control控制反转==Dependency Injection依赖注入)

## 基本概念

### Service Provider

### BeanFactory容器

### ApplicationContext容器

### 容器之扩展篇----基于注解的依赖注入

### 5种自动绑定模式

|  |
| --- |
| **no** |
| **byName** |
| **byType** |
| **constructor** |
| **autodetect** |

# AOP(动态代理)

# IOC(注入)

## bean的依赖

|  |
| --- |
| bean的depends-on属性:可以用来使用这个Bean之前, 强制执行一个或多个Bean的初始化. |

## bean的初始化：

|  |
| --- |
| 第1中：在配置文件中，制定标签<bean>的属性init-method=”init”. 在java类中增加public void init();方法 |
| 第2中：bean对应的类实现org.springframework.beans.factory.InitializingBean接口。会自动执行它的public void afterPropertiesSet()方法 |

## bean的销毁：

|  |
| --- |
| 第1种：在配置文件中，制定标签<bean>的属性destroy-method=”cleanup”. 在java类中增加cleanup()方法。 |
| 第2种：实现org.springframework.beans.factory.DisposableBean接口。会自动执行public void destroy()方法。 |

## bean 的id 和 name的区别：

|  |
| --- |
| 一个bean一般只有一个id； 如果yigebean有超过一个的id, 多个id用逗号或很好分隔，那么另外的那些id可以认为是这个bean的别名。  在Spring中可以用name或者id来制定bean的id。（建议使用id，因为id它作为XML元素的id属性来被标记，所以xml解析器能够在其他元素指向他的时候做一些额外的校验。） |

## 用ref的属性指定依赖的3种模式:

|  |
| --- |
| 第1种模式(最常用): 用ref元素的bean属性  引用bean和被引用的bean可以不在同一个XML中.  被引用bean属性可以是id属性, 也可以是name属性.  <property name=”date”>  <ref bean=”date” />  </property> |
| 第2种模式(常用): 用ref元素的local属性  引用bean和被引用bean在同一个XML中, 且 被引用的bean是只能用id来命名的.  <property name=”date”>  <ref local=”date” />  </property> |
| 第3种模式: 用ref元素的parent属性  允许引用 当前BeanFactory 或 ApplicationContext的父BeanFactory 或 ApplicationContext中的Bean  parent属性的值可以是id属性, 也可以是name属性.  <property name=”date”>  <ref parent=”date” />  </property> |

## setter方法(主) 或 构造方法(辅) 或 接口 注入的配置

|  |
| --- |
| 底层实现原理: 在new的时候就把对象传过来, 时机比较早.  <!-- 构造方法注入: 在UserManagerImpl中提供构造函数，spring将实例化好的UserDao实现注入给我们-->  <constructor-arg ref=*"userDao4OracleImpl"*/> |
| 底层实现原理: new好了再传过来. 大多数时候使用setter方法.  <!-- set方法注入: ref=注入引用 value=注入值. 当参数比较多时, 使用set方法比较容易理解. -->  <property name=*"userDao"* ref=*"userDao4MySqlImpl"* /> |
| 在接口中定义要注入的信息，并通过接口完成注入。 |
| 构造方法注入 和 setter方法注入 比较:  1. 构造方法注入可以清晰地决定注入顺序, 构造器中决定依赖关系的注入顺序, 优先依赖的优先注入. |

## ApplicationContext接口的方法

|  |  |
| --- | --- |
| FileSystemXmlApplicationContext  以指定路径的XML配置文件创建ApplicationContext | **测试类**中  **private** BeanFactory factory;  @Override  **protected** **void** setUp() **throws** Exception {  factory = **new** ClassPathXmlApplicationContext("applicationContext-\*.xml");  UserManager userManager = (UserManager)**factory**.getBean("userManager");  userManager.save("张三", "123");  } |
| Test中使用注解载入ApplicationContext | @ContextConfiguration("classpath:com/xjgz/cdc/daoservice/applicationContext.xml")  **public** **class** MapreduceInstanceTest **extends** AbstractJUnit4SpringContextTests{  @Autowired  **private** IMapReduceInstanceService mapReduceInstance;    @Test  **public** **void** testUpdateNewState(){  List<MapReduceInstance> list = mapReduceInstance.findAll();  **for**(MapReduceInstance instance : list){  **if**(StringUtils.isEmpty(instance.getYarnApplicationId())){  instance.setYarnMapreduceStatus(ApplicationState.EXCEPTION.toString());  mapReduceInstance.update(instance);  }  }  }  } |
| ClassPathXmlApplicationContext("applicationContext.xml")  以classpath下的XML配置文件创建ApplicationContext  备注: classpath方式查找, classpath包含src目录. | public class Test{  public static void main(String [] args){  ApplicationContext ctx = new FileSystemXmlApplicationContext("WebContext/WEB-INF/applicationContext.xml");  HelloWorld hello = (HelloWorld)ctx.getBean("HelloWorld");  System.out.println(hello.execute("World"));  }  } |

## 注入String/int/List/Set/Array/Map/date/Properties类型属性

|  |
| --- |
| <bean id=*"bean1"* class=*"com.bjsxt.spring.Bean1"*>  <property name=*"****str****Value"* value=*"Hello"* />  <property name=*"****int****Value"* value=*"123"* />  <property name=*"****list****Value"*>  <list>  <value>list1</value>  <value>list2</value>  </list>  </property>  <property name=*"****set****Value"*>  <set>  <value>set1</value>  <value>set2</value>  </set>  </property>  <property name=*"****arra****yValue"*>  <list>  <value>array1</value>  <value>array2</value>  </list>  </property>  <property name=*"****map****Value"*>  <map>  <entry key=*"k1"* value=*"v1"* />  <entry key=*"k2"* value=*"v2"* />  </map>  </property>  <property name=*"****Properties****Value"*>  <props>  <entry key=*"k1"* value=*"v1"* />  <entry key=*"k2"* value=*"v2"* />  </props>  </property>  <property name=*"****date****Value"*>  <value>2008/08/15</value>  </property>  </bean>  </beans> |
| 对集合的注入方式总结: 对于List/Set/Map/Properties来说都是类似的, 就是都要先把要注入的信息注入到集合中去, 然后再把集合注入到相关的bean中. |

## Spring如何管理Bean: 3种方式

|  |  |
| --- | --- |
| 第1种: 使用ApplicationContent管理Bean (最常用)  ApplicationContext factory = new FileSystemXmlApplicationContext("config.xml");  HelloWorld helloWorld = (HelloWorld) factory.getBean(“HelloWorld”);  System.out.println(helloWorld.getMsg()); |  |
| 第2种: 使用BeanWrapper管理Bean (几乎不用)  HelloWorld helloWorld = new HelloWorld();  BeanWrapper bw = new BeanWrapperlmpl(helloWorld);  bw.setPropertyValue(“msg”, “HelloWorld”);  System.out.println(bw.getPropertyValue()); | 其实没用到Spring的配置文件, 而且只能对单个Bean进行设定. |
| 第3种: 使用BeanFactory管理Bean(只有在非常关注内存的情况下采用BeanFactory)  inputStream is = new FileinputStream(“config.xml”);  XmlBeanFactory factory = new XmlBeanFactory(is);  HelloWorld helloWorld = (HelloWorld) factory.getBean(“HelloWorld”);  System.out.println(helloWorld.getMsg()); |  |

## ApplicationContext其它功能

### 国际化支持

|  |
| --- |
| **Spring配置文件:**  <bean id=*"messageSource"* class=*"org.springframework.context.support.ResourceBundleMessageSource"*>  <property name=*"basename"* value=*"messages"* />  <!-- 国际化支持文件名为: message.properties -->  </bean> |
| **message.properties**放在src下面, 也就是classpa下, 其中有条消息:  family=爸爸:{0}, 妈妈:{1} #说明: {0}和{1}用来替换外界传来的数组. |
| **java程序中使用国际化资源文件**  ApplicationContext actx = new FileSystemXmlApplicationContext("config.xml");  Object [] objs = new Object[]{“father”, “mather”}  String msg = actx.getMessage(“family”, objs, **Locale.CHINA**); |

### 资源访问

|  |
| --- |
| 存取资源  第一种: 通过Spring约定的URL虚拟路径来存取.  第二种: 通过磁盘绝对路径来存取.  第三种: 通过相对路径来存取. |

### 事件传递

|  |
| --- |
| 功过ApplicationContext的publishEvent()方法来通知ApplicationListener. |

## 抽取公共属性成抽象bean的注入

|  |
| --- |
| 如何将公共的注入定义描述出来？  \* 通过<bean>标签定义公共的属性，指定abstract=true  \* 具有相同属性的类在<bean>标签中指定其parent属性  参见：applicationContext-other.xml |
| 注入bean  <bean id=*"bean2"* class=*"com.bjsxt.spring.Bean2"*>  <property name=*"bean3"* ref=*"bean3"* />  <property name=*"bean4"* ref=*"bean4"* />  <property name=*"bean5"* ref=*"bean5"* />  </bean> |
| <!--另一种方法: bean3 和 bean4 放到 applicationContext-other.xml里面去了  <bean id="bean3" class="com.bjsxt.spring.Bean3">  <property name="id" value="1000"/>  <property name="name" value="Jack" />  <property name="password" value="123"/>  </bean>    <bean id="bean4" class="com.bjsxt.spring.Bean4">  <property name="id" value="1000"/>  <property name="name" value="Jack"/>  </bean>  -->  <bean id=*"bean5"* class=*"com.bjsxt.spring.Bean5"*>  <property name=*"age"* value=*"20"* />  </bean> |
| -------------applicationContext-other.xml 抽取公共属性成抽象bean-------------------  抽象bean不能被实例化, 因此既不能通过getBean显示地获得抽象bean,  也不能让其它bean的ref属性值指向抽象bean.  只要试图实例化抽象bean, 都将出现错误.  如果父bean没有定义class属性, 则子bean必须定义class属性.  如果父bean定义了class属性,子bean没有定义class属性, 那么子bean的class属性从父bean中继承.  如果父bean定义了class属性,子bean也定义class属性, 则子bean定义的class属性覆盖父bean的你故意的class属性.  Spring中的bean继承与java中的继承的区别:   |  |  |  | | --- | --- | --- | | Spring中的bean继承 | 实例对象与实例对象之间的关系, 主要表现为参数的延续, 不具有多态性 | 子bean和父bean可以是不同类型 | | java中的继承 | 一般到特殊的细化; 类与类之间的关系, 主要表现为方法和属性的延续 | 子类必须是一种特殊的父类 |   <bean id=*"beanAbstract"* abstract=*"true"*>  <property name=*"id"* value=*"1000"*/>  <property name=*"name"* value=*"Jack"*/><!-- 相当于覆写属性 -->  </bean>    <bean id=*"bean3"* class=*"com.bjsxt.spring.Bean3"* parent=*"beanAbstract"*>  <property name=*"name"* value=*"Tom"*/>  <property name=*"password"* value=*"123"*/>  </bean>    <bean id=*"bean4"* class=*"com.bjsxt.spring.Bean4"* parent=*"beanAbstract"*/> |

## 自定义属性编辑器

|  |
| --- |
| 什么是属性编辑器，作用？  \* 自定义属性编辑器，spring配置文件中的字符串转换成相应的对象进行注入  spring已经有内置的属性编辑器，我们可以根据需求自己定义属性编辑器    \* 如何定义属性编辑器？  \* 第一步: 继承PropertyEditorSupport类，覆写setAsText()方法，参见：UtilDatePropertyEditor.java  \* 第二步: 将属性编辑器注册到spring中，参见：applicationContext-editor.xml |
| **日期属性编辑器(系统不能自动转换, 即这里用手动转换)**  string--转换-->日期  **public** **class** UtilDatePropertyEditor **extends** PropertyEditorSupport { //继承属性编辑器  **private** String format; //set方法, 注入格式    @Override //覆写setAsText(String text)  **public** **void** setAsText(String text) **throws** IllegalArgumentException { //text: 传过来的字符串  //System.out.println("UtilDatePropertyEditor.saveAsText() -- text=" + text);    SimpleDateFormat sdf = **new** SimpleDateFormat(format);  **try** {  Date d = sdf.parse(text); //把text转换成日期类型.  **this**.setValue(d); //把转换后的值注册到spring中  } **catch** (ParseException e) {  e.printStackTrace();  }  }  **public** **void** setFormat(String format) {  **this**.format = format;  }  } |
| **配置自定义属性编辑器 applicationContext-editor.xml**  把自定义属性编辑器注入进*customEditorConfigurer里.*  <!--第1中写法: **内部bean**, 不用给id喽咯 -->  <bean id=*"customEditorConfigurer"* class=*"org.springframework.beans.factory.config.CustomEditorConfigurer"*>  <property name=*"customEditors"*> <!-- 注入到customEditors这个map类型的属性里. -->  <map>  <entry key=*"java.util.Date"*>  <bean class=*"com.bjsxt.spring.UtilDatePropertyEditor"*>  <property name=*"format"* value=*"yyyy/MM/dd"*/>  </bean>  </entry>  </map>  </property>  </bean>  </beans> |
| **配置自定义属性编辑器 applicationContext-editor.xml**  <!--第2中写法: **外部bean** -->  <bean id=*"customEditorConfigurer"* class=*"org.springframework.beans.factory.config.CustomEditorConfigurer"*>  <property name=*"customEditors"*> <!-- 注入到customEditors这个类里面 -->  <map>  <entry key=*"java.util.Date"* value-ref=*"utilDatePropertyEditor"* />  </map>  </property>  </bean>  <bean id=*"utilDatePropertyEditor"* class=*"com.bjsxt.spring.UtilDatePropertyEditor"*>  <property name=*"format"* value=*"yyyy/MM/dd"*/>  </bean> |

## Bean的作用域：scope="singleton/prototype "

|  |
| --- |
| <!--  <bean id="bean1" class="com.bjsxt.spring.Bean1" scope="singleton" />  默认scope="singleton" 每次调用getBean的时候返回相同的实例----只适合单线程(用得多)  -->    <!-- scope="prototype" 每次调用getBean的时候返回不同的实例----适合多线程 (用得少) -->  <bean id=*"bean1"* class=*"com.bjsxt.spring.Bean1"* scope=*"prototype"*/> |
| 单例依赖于多例(singleton bean依赖于non-singleton bean)时, 会产生不同步的现象. 解决该问题有如下两种思路:  方法1: 部分放弃依赖注入: singleton bean每次需要non-singleton时, 主动向容器请求新的bean实例, 保证每次产生的bean实例都是新的实例.  方法2: 利用方法注入**(常用)**. 通常使用lookup注入, // lookup注入,还需学习. |

## Bean的5种自动装配模式

### 1. 根据类型自动装配(推介)

|  |
| --- |
| 开发阶段可提高效率, 开发完成后最好写上, 以后才好维护. 约定优于配置 |
| <?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:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.0.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.0.xsd"*  default-autowire=*"byType"*> <!-- 属性和与之相对的bean名称可以不同, 只要类型对应, 就能装配上. -->  **<!-- <!-- 第一种写法: 写全所有属性(基本写法)**  <bean id="bean2" class="com.bjsxt.spring.Bean2">  <property name="bean3" ref="bean3" />  <property name="bean4" ref="bean4" />  <property name="bean5" ref="bean5" />  </bean>  -->  **<!--第二种写法: 根据bean2属性类型, 在配置文件里寻找相同class类型的bean, 自动装配上 -->**  <bean id=*"bean2"* class=*"com.bjsxt.spring.Bean2"* />  <bean id=*"beanAbstract"* abstract=*"true"*>  <property name=*"id"* value=*"1000"* />  <property name=*"name"* value=*"Jack"* />  </bean>  <bean id=*"bean3222"* class=*"com.bjsxt.spring.****Bean3****"* parent=*"beanAbstract"*>  <property name=*"name"* value=*"Tom"* />  <property name=*"password"* value=*"123"* />  </bean>  <bean id=*"bean42222"* class=*"com.bjsxt.spring.****Bean4****"* parent=*"beanAbstract"* />  <bean id=*"bean5111"* class=*"com.bjsxt.spring.****Bean5****"*>  <property name=*"age"* value=*"20"* />  </bean>  </beans> |

### 2. 根据名称自动装配(不推介)

|  |
| --- |
| 开发阶段可提高效率, 开发完成后最好写上, 以后才好维护. 约定优于配置 |
| <?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:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.0.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.0.xsd"*  default-autowire=*"byName"*> <!--添加这条属性, 只对这个文件生效 -->  <!-- <!-- 第一种写法: 写全所有属性(基本写法)  <bean id="bean2" class="com.bjsxt.spring.Bean2">  <property name="bean3" ref="bean3" />  <property name="bean4" ref="bean4" />  <property name="bean5" ref="bean5" />  </bean>  -->  <!--第二种写法: 根据bean2属性名字, 在配置文件中查找id值与属性名相同的bean, 自动装配上 -->  <bean id=*"bean2"* class=*"com.bjsxt.spring.Bean2"* />  <bean id=*"beanAbstract"* abstract=*"true"*>  <property name=*"id"* value=*"1000"*/>  <property name=*"name"* value=*"Jack"*/>  </bean>  <bean id=*"****bean3****"* class=*"com.bjsxt.spring.Bean3"* parent=*"beanAbstract"*>  <property name=*"name"* value=*"Tom"*/>  <property name=*"password"* value=*"123"*/>  </bean>  <bean id=*"****bean4****"* class=*"com.bjsxt.spring.Bean4"* parent=*"beanAbstract"*/>  <bean id=*"****bean5****"* class=*"com.bjsxt.spring.Bean5"*>  <property name=*"age"* value=*"20"* />  </bean>  </beans> |

### 3. 根据构造函数(construntor)自动装配

|  |
| --- |
|  |

### 4. 根据autodetect(通过对Bean检查类的内部来选择constructor或byType)自动装配

|  |
| --- |
|  |

### 5. 不使用自动装配: no模式

|  |
| --- |
| bean的引用必须通过ref元素定义. 这是默认配置. |

## Bean依赖检查的4种模式

|  |  |  |
| --- | --- | --- |
| 一般情况: 依赖检查和自动装配结合使用. 依赖检查会在自动装配完成后检查bean的每个属性是否都设定完成了.  当bean的属性都有默认的值, 或者 不需要对bean的属性时候都被设置到bean上检查时, 依赖检查的作用就不大了.  依赖检查能够分别对每一个bean应用 或取 消应用. 默认是不检查依赖关系.  使用bean元素的dependency-check属性来指定bean定义的依赖检查, 共4种. | | |
| 第1种: simple模式 | dependency-check=”simple” | 只会对基本类型和集合(除了合作者bean)进行依赖检查 |
| 第2种: object模式 | dependency-check=”object” | 只会对合作者bean的类型进行依赖检查 |
| 第3种: all模式 | dependency-check=”all” | 会对合作者bean 基本类型和集合进行依赖检查 |
| 第4种: none模式 | dependency-check=”none” | 不会进行依赖检查. 没有指定值的bean属性仅仅是没有设值. |

## BeanFactory的常用方法：

|  |  |
| --- | --- |
| public boolean containsBean(String name) | 判断Spring容器时候包含id为name的bean定义. |
| public object getBean(String name) | 返回id为name的bean |
| public object getBean(String name, Class requiredType) | 返回容器中id为name, 并且类型为requiredType的bean |
| public Class getType(String name) | 返回容器中id为name的bean的类型 |

# 代理(AOP:面向切面/方面编程)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 特征: 代理对象跟目标对象的接口是一致的.  代理对象控制目标对象.  传入目标对象, 生成代理对象.  Aspect默认情况下不用实现接口，但对于目标对象（UserManagerImpl.java），在默认情况下必须实现接口（因为sping在底层也是通过java的动态代理来实现） 如果没有实现接口必须引入CGLIB库。会用另一种实现方式，即他在编译时，把你需要在某些方法前需要加的所有检查等全加进去了。  /\*\*  \* Cross cutting concern: 横切性的关注点: 这里对应--安全性检查SecurityHandler  \* Aspect切面类: 对横切性关注点模块化后的这个类  \*  \* Adviso是Pointcut和Advice的配置器, 它包括Pointcut和Advice, 是将Advice注入到程序中的Pointcut位置的代码.  \*  \* Advice: 通知  \* Before Advice: 加在方法之前  \* After Advice: 加在方法之后  \*  \* Pointcut(add\*)切入点: 是Join Point的集合, 它是程序中需要注入Advice的位置的集合, 可以约定一个范围(应用在哪些类的哪些方法上), 有一些表达式  \* Join Point(addUser(...))连接点: 执行的一个点(构造方法, 实例变量, 方法), Spring只支持方法上的连接点(只能在方法前后植入).  \* target Object: 目标对象(真实的实现类)  \* weave植入: 把切面应用到目标对象方法前的这个过程.  \* Proxy Object: 代理对象  \* Introduction: 当我们这个类有方法1, 方法2的时候, 它可以动态地再加一个方法3.  \*/  Spring提供了5中Advice类型   |  |  |  | | --- | --- | --- | | Interception Around | 在JoinPoint前后都执行 | 实现Interception Around通知的类需要实现接口MethodInterceptor接口 | | Before | 在JoinPoint前执行 | 实现Before通知的类需要实现接口MethodBeforeAdvice接口 | | After Returning | JoinPoint后执行 | 实现After Returning通知的类需要实现接口AfterReturningAdvice接口 | | Throw | JoinPoint抛出异常时执行 | 实现Throw通知的类需要实现接口ThrowAdvice接口 | | Introduction | JoinPoint调用完毕后执行 | 实现Introduction通知的类需要实现接口IntroductionAdvisor和IntroductionInterceptor接口 | |

## JDK动态代理(只能对实现了接口的类进行代理)

### 手工编写切面类,

|  |
| --- |
| AOP的关键: 发现横切性关注点, 然后把它模块化. 模块化后的这个类叫切面类(Aspect)  把横切性的东西单独拿出来, 把散布在程序各个角落的关注点拿出来模块化, 单独维护这个模块就好了.  代理类在运行期生成,  需要实现InvocationHandler接口 |
| **Aspect切面类(模块化的这个类)**  **import** java.lang.reflect.InvocationHandler;  **import** java.lang.reflect.Method;  **import** java.lang.reflect.Proxy;  **public** **class** SecurityHandler **implements** InvocationHandler {  **private** Object targetObject;    /\*\*  \* 此方法: 传入的是目标对象, 返回代理对象,  \*/  **public** Object **newProxy**(Object targetObject) {  **this**.targetObject = targetObject; //注入目标对象  //返回代理对象  **return** Proxy.*newProxyInstance*(targetObject.getClass().getClassLoader(), //第一个参数: 目标对象的ClassLoader  targetObject.getClass().getInterfaces(), //第二个参数: 目标对象的Interface  **this**); //第三个参数: 传递实现InvocationHandler接口的的对象, 就是自身  }    /\*\*  \* 生成代理对象后, 调用代理对象的方法之前, 会自动调用invoke方法.  \*/  **public** Object **invoke**(Object proxy, Method method, Object[] args) **throws** Throwable {  checkSecurity(); //调用横切性的模块, 调用之前可检查此method是否需要调用横切性模块, metod可以是add, update, delete等.    //做完安全性检查后, 调用目标对象上的真实实现  Object ret = **null**; //定义目标对象实现 的返回值(不管目标对象调用的方法有无返回值, 都要这样定义)  **try** {  ret = method.invoke(**this**.targetObject, args); //调用目标对象上的方法, 有可能出现异常. 第一个参数:目标对象; 第二个参数: 就是调用目标对象 传入的参数.  }**catch**(Exception e) {  e.printStackTrace();  **throw** **new** java.lang.RuntimeException(e);  }  **return** ret;  }  //SecurityHandler  **private** **void** checkSecurity() {  System.*out*.println("----------checkSecurity()---------------");  }  } |
| **客户端**  **public** **class** Client {  **public** **static** **void** main(String[] args) {  //UserManager userManager = new UserManagerImpl();    SecurityHandler handler = **new** SecurityHandler(); //new一个安全性处理类  //方法: newProxy(new UserManagerImpl()) 传入目标对象, 返回代理对象,  //代理对象和目标对象接口是一致的, 可直接转换  UserManager userManager = (UserManager)handler.newProxy(**new** UserManagerImpl());    //使用代理对象  // userManager.deleteUser(1);  userManager.addUser("wanlinjin", "jinwanlin");  }  } |

### 采用Annotation(注解)方式

|  |
| --- |
| 动态: 运行期生成的  1、spring依赖库  \* SPRING\_HOME/dist/spring.jar  \* SPRING\_HOME/lib/jakarta-commons/commons-logging.jar  \* SPRING\_HOME/lib/log4j/log4j-1.2.14.jar  \* SPRING\_HOME/lib/aspectj/\*.jar |
| <?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:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.0.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.0.xsd"*>    **<aop:aspectj-autoproxy/> //启用对aop的支持**    <bean id=*"securityHandler"* class=*"com.bjsxt.spring.SecurityHandler"*/> //配置切面类  <bean id=*"userManager"* class=*"com.bjsxt.spring.UserManagerImpl"*/> //配置目标对象    </beans> |
| **切面类**  **import** org.aspectj.lang.annotation.Aspect;  **import** org.aspectj.lang.annotation.Before;  **import** org.aspectj.lang.annotation.Pointcut;  /\*\*  \* 定义Aspect(切面)  \*/  @Aspect  **public** **class** SecurityHandler {  /\*\*  \* 定义Pointcut(切入点),Pointcut的名称就是方法名:allAddMethod，此方法不能有返回值和参数，该方法只是一个 标识  \* Pointcut的内容是一个表达式，描述哪些对象的哪些方法（订阅Joinpoint）  \*/  @Pointcut("execution(\* add\*(..)) || execution(\* del\*(..))") //拦截哪些方法 \* add\*(..) 所有返回值的, 以add开头的方法, 参数个数任意.  **private** **void** allAddMethod() { //此方法为切入点的方法: 只是一个表标识, 不会执行的. Advice中通过方法名引用这个切人点  };  /\*\*  \* 定义Advice(标识)，标识在哪个切入点何处植入此方法  \*/  @Before("allAddMethod()")  **private** **void** checkSecurity() {  System.*out*.println("----------checkSecurity()---------------");  }  } |
| **客户端类**  **import** org.springframework.beans.factory.BeanFactory;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** Client {  **public** **static** **void** main(String[] args) {  BeanFactory factory = **new** ClassPathXmlApplicationContext("applicationContext.xml");  //Spring会自动拿动态代理对象  UserManager userManager = (UserManager)factory.getBean("userManager");  //调代理对象上的方法,  userManager.addUser("张三", "123");  userManager.deleteUser(1);  }  } |

### 采用静态配置文件方式实现AOP:

|  |
| --- |
| 静态: 实实在在能看到的  spring对AOP的只是（采用配置文件的方式）  1、spring依赖库  \* SPRING\_HOME/dist/spring.jar  \* SPRING\_HOME/lib/jakarta-commons/commons-logging.jar  \* SPRING\_HOME/lib/log4j/log4j-1.2.14.jar  \* SPRING\_HOME/lib/aspectj/\*.jar |
| 2、配置如下  <?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:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.0.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.0.xsd"*>    <bean id=*"securityHandler"* class=*"com.bjsxt.spring.SecurityHandler"*/> //切面类  <bean id=*"userManager"* class=*"com.bjsxt.spring.UserManagerImpl"*/> //目标类    <aop:config>  <aop:aspect id=*"security"* ref=*"securityHandler"*> //注册Aspect(切面类), id随意定义. <!--Aspect由Pointcut和Advice组成-->  <!--声明Pointcut(切入点), 表达式方法声明匹配被切入的类及其方法-->  <aop:pointcut id=*"allAddMethod"* expression=*"execution(\* com.bjsxt.spring.UserManagerImpl.add\*(..))"*/>  <!-- 声明Advice,即切面类要切入的方法. *checkSecurity是一个*横切性的关注点; pointcut-ref=应用到哪个切入点上? 指向上一个切入点id -->  <aop:before method=*"checkSecurity"* pointcut-ref=*"allAddMethod"*/>  </aop:aspect>  </aop:config>  </beans> |
| **切面类**  **public** **class** SecurityHandler {    **private** **void** checkSecurity() {  System.*out*.println("----------checkSecurity()---------------");  }  } |
| **客户端**  **import** org.springframework.beans.factory.BeanFactory;  **import** org.springframework.context.support.ClassPathXmlApplicationContext;  **public** **class** Client {  **public** **static** **void** main(String[] args) {  BeanFactory factory = **new** ClassPathXmlApplicationContext("applicationContext.xml");    UserManager userManager = (UserManager)factory.getBean("userManager");    userManager.addUser("张三", "123");  userManager.deleteUser(1);  }  } |

## CGLIB库实现动态代理(可对类进行代理)

|  |
| --- |
| 如何强制使用CGLIB实现AOP？  \* 添加CGLIB库，cglib-nodep-2.1.jar  \* 在spring配置文件中加入<aop:aspectj-autoproxy proxy-target-class="true"/> |
| <?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:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.0.xsd*  *http://www.springframework.org/schema/tx* [*http://www.springframework.org/schema/tx/spring-tx-2.0.xsd*](http://www.springframework.org/schema/tx/spring-tx-2.0.xsd)*"*>  <!-- 强制使用CGLIB, 如果没配置这个, 则  <aop:aspectj-autoproxy proxy-target-class="true"/>  -->  <bean id=*"securityHandler"* class=*"com.bjsxt.spring.SecurityHandler"*/>    <bean id=*"userManager"* class=*"com.bjsxt.spring.UserManagerImpl"*/>    <aop:config>  <aop:aspect id=*"security"* ref=*"securityHandler"*>  <aop:pointcut id=*"allAddMethod"* expression=*"execution(\* com.bjsxt.spring.UserManagerImpl.add\*(..))"*/>  <aop:before method=*"checkSecurity"* pointcut-ref=*"allAddMethod"*/>  </aop:aspect>  </aop:config>  </beans> |

## 从JoinPoint中可取得参数值, 方法名等等

|  |
| --- |
| spring对AOP的支持  JDK的动态代理, 只能对实现了这个接口的内生成代理类, 如果没有实现接口, 他是无法生成代理类的.  Aspect默认情况下不用实现接口，但对于目标对象（UserManagerImpl.java），在默认情况下必须实现接口, 如果没有实现接口, 则必须引入CGLIB库  我们可以通过Advice中添加一个JoinPoint参数，这个值会由spring自动传入，从JoinPoint中可以取得参数值、方法名等等  其它和上面的静态配置文件方式实现AOP一致 |
| **import** org.aspectj.lang.JoinPoint;  **public** **class** SecurityHandler {    /\*\*  \* 拿参数  \* **@param** joinPoint  \*/  **private** **void** checkSecurity(JoinPoint joinPoint) {  Object[] args = joinPoint.getArgs(); //取得参数  **for** (**int** i=0; i<args.length; i++) {  System.*out*.println(args[i]);  }    System.*out*.println(joinPoint.getSignature().getName());  System.*out*.println("----------checkSecurity()---------------");  }  } |

## spring对AOP的支持, 在JDK动态代理和CGLIB(字节码生成)之间的选择

|  |
| --- |
| 1、如果目标对象实现了接口，默认情况下会采用JDK的动态代理实现AOP 2、如果目标对象实现了接口，可以强制使用CGLIB实现AOP 3、如果目标对象没有实现了接口，必须采用CGLIB库， 4、如果有些实现有些没有实现，spring会自动在JDK动态代理和CGLIB之间转换  JDK动态代理和CGLIB字节码生成的区别？  \* (推荐) JDK动态代理只能对实现了接口的类生成代理，而不能针对类  \* (要引包, 不推荐)CGLIB是针对类实现代理，主要是对指定的类生成一个子类，覆盖其中的方法. 因为是继承，所以该类或方法最好不要声明成final |

# 专业术语

|  |  |
| --- | --- |
| IOC: (Inversion of Control) | 控制反转(交给别人了) |
| AOP(Aspect Oriented Programming) | 面向切面(方面)编程 |
| AOC容器(Spring Core) | 管理对象, 是一个大工厂, 把我们创建对象, 管理对象 |
| POJO (Plain Old Java Object) | 普通java对象 |
| ORM | 对象关系映射 |
| DI (Dependency Injection) | 依赖注入 |

|  |  |  |
| --- | --- | --- |
| 核心关注点 | 和具体应用功能相关 |  |
| 横切关注点 | 在于在哪个个系统的范围内 | 日志,权限,异常处理, 事务处理 等 |

# Spring MVC

|  |
| --- |
| SpingMVC的架构其实也很简单，它通过一个共通的入门 DispatcherServlet来接收所有的request，接下来根据request要求的页面通过handler转送给Controller处理，处理结果返回View给用户，因此对SpringMVC来说，其核心为DispatcherServlet。 |
| **web.xml**  < servlet >  < servlet-name > SpringMVCDemo </ servlet-name >  < servlet-class >org.springframework.web.servlet.DispatcherServlet</ servlet-class >  < load-on-startup > 1 </ load-on-startup >  </ servlet >  < servlet-mapping >  < servlet-name > SpringMVCDemo </ servlet-name >  < url-pattern > \*.do </ url-pattern >  </ servlet-mapping > |
| **applicationContext.xml**  < bean id ="urlMapping" class ="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping" >  < property name ="mappings" >  < props>  < prop key ="/hello.do" > springmvcController </ prop >  </ props >  </ property >  </ bean >  <!-- SpringmvcController -->  < bean id ="springmvcController" class ="org.improviser.webapp.controller.SpringmvcController" /> |
| <!-- **ViewSolver** -->  < bean id ="viewResolver" class ="org.springframework.web.servlet.view.InternalResourceViewResolver" >  < property name ="viewClass" value="org.springframework.web.servlet.view.JstlView" />  < property name ="prefix" value="/WEB-INF/pages/" />  < property name ="suffix" value=".jsp" />  </ bean > |
| **controller**  public class SpringmvcController implements Controller {  protected final Log logger = LogFactory.getLog(getClass());  public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  Map model = new HashMap();  String str = "即兴诗人";  logger.info("returning hello view with " + currTime);  model.put("coldice", str);  return new ModelAndView("hello", "message", model);  } |
| hello.**jsp页面**  <html>  <head>  <title>Spring Application</title>  </head>  <body>  Hello ${message.coldice}</br>  Spring MVC 测试成功！  </body>  </html> |

## 页面绑定对象到controller

|  |
| --- |
| JSP |
| Controller  @InitBinder  **protected** **void** initBinder(HttpServletRequest request, ServletRequestDataBinder binder) **throws** Exception {  binder.registerCustomEditor(Date.**class**, **new** CustomDateEditor(**new** SimpleDateFormat("yyyy-MM-dd"), **true**)); //绑定date  binder.registerCustomEditor(byte[].**class**, **new** ByteArrayMultipartFileEditor());//绑定二进制文件  binder.registerCustomEditor(ContentType.**class**, **new** ContentTypeEditor(contentTypeService));//绑定数据库中的对象  binder.registerCustomEditor(OrderStatus.**class**, **new** OrderStatusEditor());//绑定枚举  binder.registerCustomEditor(Set.**class**, **new** CustomCollectionEditor(Set.**class**));//绑定set  } |
|  |
| **绑定对象来源于数据库**  import java.beans.PropertyEditorSupport;  import org.apache.commons.lang.StringUtils;  import com.broadin.bookshelf.service.author.IAuthorService;  import com.broadin.bookshelf.service.contentType.IContentTypeService;  import com.broadin.bookshelf.service.publisher.IPublisherService;  /\*\*  \*  \* @author jinwanlin  \* @version 1.0 Revise History:  \*  \*/  public class ContentTypeEditor extends PropertyEditorSupport {    private IContentTypeService contentTypeService;    public ContentTypeEditor(IContentTypeService contentTypeService){  this.contentTypeService = contentTypeService;  }    @Override  public void setAsText(String text) throws IllegalArgumentException {  if (StringUtils.isNotBlank(text)) {  setValue(contentTypeService.getByPrimaryKey(Long.parseLong(text)));  }  }  } |
| **绑定枚举对象**  import java.beans.PropertyEditorSupport;  import com.broadin.bookshelf.enums.OrderStatus;  /\*\*  \* TODO class purpose description  \*  \* @author ningt  \* @version 1.0 Revise History:  \*  \*/  public class OrderStatusEditor extends PropertyEditorSupport {  @Override  public void setAsText(String text) throws IllegalArgumentException {  setValue(OrderStatus.fromCode(text));  }  } |
| **绑定Set** |
| **绑定** |

# Spring整合第三方MVC框架的通用配置

|  |
| --- |
| 在web.xml中配置ApplicationContent的自动创建有如下两种策略:  第1种: 利用ServletContentListener实现(现在都用这个)  第2种: 采用load-on-startup Servlet实现  说明: 根据Servlet2.3标准, 所有的ServletContentListener都会比Servlet优先加载(即使是load-on-startup Servlet)  考虑到Spring容器应该尽可能早地创建实例, 应该优先采用listener创建ContentListener.  不过ServletContentListener是从Servlet2.3才出现的标准, 如果使用Servlet2.2或更早的版本, 则应使用load-on-startup Servlet创建. |
| **只有一个配置文件**: 默认自动查找WEB-INF/applicationContext.xml文件  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |
| **如果有多个配置文件:** 则考虑使用<context-param>元素来确定配置文件的文件名. 多个配置文件之间以逗号隔开.  <context-param>  <param-name>contextConfigLocation</param-name> <!-- 配置了contextConfigLocation就不再去找默认的配置文件了 -->  <param-value>/WEB-INF/daoContext.xml, /WEB-INF/applicationContext.xml</param-value>  </ context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |

**MVC框架**

|  |
| --- |
| struts1, webwork, strut2, spring mvc, jsf |

**MVC框架都做什么**

|  |
| --- |
| @、以Controller为中心完成对系统流程的控制管理  @、从请求中收集数据  @、对传入参数进行验证  @、根据请求调用相应业务逻辑完成数据处理  @、将处理结果返回给视图  @、国际化支持  @、针对不同视图技术提供不同解析支持方案  @、针对JSP视图技术提供标签库  @、通过拦截器链实现面向方面编程完成系统级控制  @、对文件上传、下载等常用功能的封装 |

**Spring MVC框架的核心构件**

|  |
| --- |
| Dispatcher Servlet  Controller  Handler Mapping  ViewResolver & View  Interceptors  LocalResolver  Validate |

**Spring MVC 常用 Controller**

|  |
| --- |
| @、直接实现Controller 接口或AbstractController抽象类  @、ModelAndView对象  @、用于直接跳转页面的ParameterizableViewController, UrlFilenameViewController  @、在同一控制器类中处理多个请求的MultiActionController.具体调用的方法通过MethodNameResolver决定(InternalPathMethodResolver, ParameterMethodNameResolver, PropertiesMethodResolver)  方法签名：  public (ModelAndView | Map | String | void) actionName(HttpServletRequest request, HttpServletResponse response, [,HttpSession] [,AnyObject]);  @、能封装表单参数的CommondController、 FormController |

**Spring MVC 常用 Handler Mapping**

|  |
| --- |
| @、默认的ControllerClassNameHandlerMapping  @、常用的SimpleUrlHandlerMapping  @、BeanNameUrlHandlerMapping  @、基于注解Controller时使用的DefaultAnnotationHandlerMapping |

**基于注解的MVC**

|  |
| --- |
| @、@Controller  @、@RequestMapping  @、@RequestParam, @PathVariable, @CookieValue  @、Handler方法的可用参数与返回值 |

**Spring MVC 常用 ViewResolver & View**

|  |
| --- |
| @、 *InternalResourceViewResolver、 UrlBasedViewResolver*  @、Redirect 重定向视图 (redirect:)  @、XmlViewResovler  @、视图链  @、JstlView, ExcelView及自定义view (JsperReport, Pdf, Excel) |

**拦截器、视图国际化、验证**

|  |
| --- |
| @、 *Interceptor*  @、resourceMessage  @、JSR-303 validate bean |

# SimpleFormController的实现类里做数据校验

|  |
| --- |
| 1. 配置文件里, 加这么一句 <property name="validateOnBinding" value="true" />  2. controller里面重写这个方法:  @Override  **protected** **void** onBindAndValidate(HttpServletRequest request, Object command, BindException errors) **throws** Exception {  BusinessComplaint businessComplaint = (BusinessComplaint) command;  //对内容长度进行校验  **int** contentLength = businessComplaint.getContent().length();  **if** (contentLength <= 4) {  errors.rejectValue("content", **null**, "内容不能为空！");  }  **if** (contentLength >= 1000) {  errors.rejectValue("content", **null**, "内容不能超过1000字节！");  }  }  3. JSP页面, 加这么个显示错误的地方  <form:errors path="content"/> |

# 文件上传下载

## 上传一个文件

|  |
| --- |
| <form:form modelAttribute=*"ad"* action=*"form.do"* id=*"adForm"* enctype=*"multipart/form-data"*>  <input type=*"file"* name=*"picture"* /> 这个picture, 是ad的一个byte[]类型的属性, 专门用来上传文件, 不映射到数据库  </form:form> |
| /\*  \* @(#) AdForm.java 2011-02-15  \* Project :新疆商务领航  \* Copyright: 2010 XJGZ Inc. All rights reserved.  \*/  **package** com.ismp.web.manager.controller.ad;  **import** java.io.File;  **import** java.net.URLEncoder;  **import** java.util.Date;  **import** java.util.HashMap;  **import** java.util.Map;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.util.FileCopyUtils;  **import** org.springframework.validation.BindException;  **import** org.springframework.web.bind.ServletRequestDataBinder;  **import** org.springframework.web.bind.ServletRequestUtils;  **import** org.springframework.web.multipart.MultipartHttpServletRequest;  **import** org.springframework.web.multipart.commons.CommonsMultipartFile;  **import** org.springframework.web.multipart.support.ByteArrayMultipartFileEditor;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.SimpleFormController;  **import** com.ismp.model.tobacco.Ad;  **import** com.ismp.service.tobacco.IAdService;  /\*\*  \* 烟草新商盟: 公告管理  \*  \* **@author** jinwanlin  \* **@version** 3.0  \* Revise History:  \*  \*/  **public** **class** AdForm **extends** SimpleFormController {  **private** IAdService adService;  **private** String adDir;      **public** AdForm() {  setCommandClass(Ad.**class**);  setCommandName("ad");  setSessionForm(**true**);  }    @Override  **protected** Object formBackingObject(HttpServletRequest request) **throws** Exception {  String id = ServletRequestUtils.*getStringParameter*(request, "id", ""); //广告位编号  Ad ad = **null**;  **if** (id!=**null** && !id.equals("")) {  ad = (Ad) **this**.adService.getAdByNumber(id);  }    **if**(ad == **null**){  ad = **new** Ad();  ad.setNumber(id); //设置广告为编号  }  **return** ad;  }  @Override  **protected** **void** initBinder(HttpServletRequest request, ServletRequestDataBinder binder) **throws** Exception {  binder.registerCustomEditor(**byte**[].**class**, **new** ByteArrayMultipartFileEditor());  **super**.initBinder(request, binder);  }    @SuppressWarnings({ "unchecked" })  @Override  **protected** Map referenceData(HttpServletRequest request) **throws** Exception {  Map referenceData = **new** HashMap();  referenceData.put("adDir", adDir);  **return** referenceData;  }    @Override  **protected** **void** onBindAndValidate(HttpServletRequest request, Object command, BindException errors) **throws** Exception {  Ad ad = (Ad) command;  **if** (ad.isNew()) {  //查看 "广告位编号" 是否已存在  Ad adFromDB = adService.getAdByNumber(ad.getNumber());  **if** (adFromDB != **null**) {  errors.rejectValue("number", **null**, "广告位编号已存在, 不能重复添加！");  }  }  }    @Override  **protected** ModelAndView onSubmit(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors) **throws** Exception {  Ad ad = (Ad) command;  ad.setCreateDate(**new** Date());  MultipartHttpServletRequest multipartRequest = (MultipartHttpServletRequest) request;  CommonsMultipartFile pictureFile = (CommonsMultipartFile) multipartRequest.getFile("picture");  **byte**[] picture = ad.getPicture();    **if** (picture.length>0) {  String name = **this**.copy(request, pictureFile, adDir, picture);  ad.setName(name);  }    String message = **null**;  **if** (ad.isNew()) {  message = "添加成功！";  }**else** {  message = "修改成功！";  }    adService.saveOrUpdate(ad);  **return** **new** ModelAndView("redirect:list.do", "message", URLEncoder.*encode*(message, "UTF-8"));  }  /\*\*  \* 保存文件到硬盘上  \*  \* **@param** request  \* **@param** fileName  \* **@param** dir  \* **@return**  \* **@throws** Exception  \*/  **public** String copy(HttpServletRequest request, CommonsMultipartFile file, String dir, **byte**[] bytes) **throws** Exception {  String logoName = **null**;  // String logoPath = "";  **if** (!file.isEmpty()) {  String uploadDir = request.getSession().getServletContext().getRealPath(dir);  File dirPath = **new** File(uploadDir);  **if** (!dirPath.exists())  dirPath.mkdirs();  String sep = System.*getProperty*("file.separator");  // 获取文件扩展名:  String filename = file.getOriginalFilename();  String ext = "";  **int** pos;  **if** ((pos = filename.lastIndexOf('.')) != (-1)) {  ext = URLEncoder.*encode*(filename.substring(pos).trim(), "UTF-8");  }  logoName = generateTimeMillis() + ext;  // logoPath = dir + sep + logoName;  File uploadedFile = **new** File(uploadDir + sep + logoName);  FileCopyUtils.*copy*(bytes, uploadedFile);  }  **return** logoName;  }    **public** **synchronized** String generateTimeMillis() {  **return** String.*valueOf*(System.*currentTimeMillis*());  }    **public** **void** setAdService(IAdService adService) {  **this**.adService = adService;  }  **public** **void** setAdDir(String adDir) {  **this**.adDir = adDir;  }  } |

## 上传多文件

|  |
| --- |
| **JSP页面代码(说明: file\_1 file\_2 file\_3是随意起的名字, 跟其它无关)**  <form enctype="multipart/form-data" action="" method="post">  <input type="file" name="file\_1" />  <input type="file" name="file\_2" />  <input type="file" name="file\_3" />  </form> |
| /\*  \* @(#) CheckBillForm.java 2010-12-6  \* Project :新疆商务领航  \* Copyright: 2010 XJGZ Inc. All rights reserved.  \*/  **package** com.ismp.web.copartner.controller.topicsJoin;  **import** java.io.File;  **import** java.net.URLEncoder;  **import** java.util.Date;  **import** java.util.HashMap;  **import** java.util.Iterator;  **import** java.util.Map;  **import** java.util.Random;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.util.FileCopyUtils;  **import** org.springframework.validation.BindException;  **import** org.springframework.validation.Errors;  **import** org.springframework.web.bind.ServletRequestDataBinder;  **import** org.springframework.web.bind.ServletRequestUtils;  **import** org.springframework.web.multipart.MultipartFile;  **import** org.springframework.web.multipart.MultipartHttpServletRequest;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.SimpleFormController;  **import** com.ismp.model.copartner.CopartnerOrg;  **import** com.ismp.model.copartner.ExecplanReport;  **import** com.ismp.model.copartner.TopicExecplan;  **import** com.ismp.service.copartner.ICopartnerOrgService;  **import** com.ismp.service.copartner.IExecPlanReportService;  **import** com.ismp.service.copartner.ITopicExecPlanService;  **import** com.ismp.service.util.impl.FileHandleUtil;  **import** com.ismp.web.util.WebConfig;  /\*\*  \* 添加我的合作申请  \*  \* **@author** jinwanlin  \* **@version** 3.0  \* Revise History:  \*  \*/  **public** **class** ReportForm **extends** SimpleFormController{  **private** ICopartnerOrgService copartnerOrgService;  **private** ITopicExecPlanService topicExecPlanService;  **private** IExecPlanReportService execPlanReportService;  /\*\*附件存放路径\*\*/  **private** String execplanReportAttacheDir;      **public** ReportForm() {  setCommandClass(ExecplanReport.**class**);  setCommandName("execplanReport");  setSessionForm(**true**);  setBindOnNewForm(**false**);  }    **protected** Object formBackingObject(HttpServletRequest request) **throws** Exception {  ExecplanReport execplanReport = **null**;  **long** id = ServletRequestUtils.*getLongParameter*(request, "id", 0);  **if** (id == 0) {  execplanReport = **new** ExecplanReport();  //计划ID  **long** planId = ServletRequestUtils.*getRequiredLongParameter*(request, "planId");  TopicExecplan topicExecplan = (TopicExecplan) topicExecPlanService.getByPrimaryKey(TopicExecplan.**class**, planId);  execplanReport.setPlan(topicExecplan);  } **else** {  execplanReport = (ExecplanReport)execPlanReportService.getByPrimaryKey(ExecplanReport.**class**, id);  }  **return** execplanReport;  }  **protected** **void** initBinder(HttpServletRequest request, ServletRequestDataBinder binder) **throws** Exception {  **super**.initBinder(request, binder);  }  @Override  **protected** ModelAndView handleInvalidSubmit(HttpServletRequest request, HttpServletResponse response) **throws** Exception {  BindException errors = getErrorsForNewForm(request);  errors.reject("duplicateFormSubmission", "Duplicate form submission");  **return** showForm(request, response, errors);  }    @SuppressWarnings("unchecked")  **protected** Map referenceData(HttpServletRequest request, Object command, Errors errors) **throws** Exception {  Map referenceData = **new** HashMap();  **return** referenceData;  }  **protected** **void** onBindAndValidate(HttpServletRequest request, Object command, BindException errors) **throws** Exception {  String errorInfo = validateFile(request);  **if** (errorInfo!=**null**){  errors.rejectValue("attache", **null**, errorInfo);  }  }  **protected** ModelAndView onSubmit(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors) **throws** Exception {  ExecplanReport execplanReport = (ExecplanReport)command;    execplanReport.setCreateDate(**new** Date());  execplanReport.setSi(getCopartnerOrg(request));  String fileName = getFileSet(request);  **if**(fileName!=**null** && !"".equals(fileName)){  execplanReport.setAttache(fileName);  }  String message = **null**;  **if**(execplanReport.isNew()){  message = "添加成功！";  }**else**{  message = "修改成功！";  }  execPlanReportService.saveOrUpdate(execplanReport);  ModelAndView mav = **new** ModelAndView("redirect:listReport.do");  mav.addObject("message", URLEncoder.*encode*(message, "UTF-8"));  mav.addObject("id", execplanReport.getPlan().getId());  **return** mav;  }    /\*\*  \* 取得文件集合  \*  \* **@param** request  \* **@return**  \*/  @SuppressWarnings("unchecked")  **public** String getFileSet(HttpServletRequest request) **throws** Exception{  StringBuffer files = **new** StringBuffer();  MultipartHttpServletRequest multipartRequest = (MultipartHttpServletRequest)request;  **for** (Iterator it = multipartRequest.getFileNames(); it.hasNext();) {  String key = (String) it.next();  MultipartFile file = multipartRequest.getFile(key);  **if** (file.getOriginalFilename().length() > 0) {  String fileName = uploadFileAndCallback(file, request); //保存文件到硬盘  files.append(fileName);  files.append("|");  }  }  **if**(files.length()>0){  files.deleteCharAt(files.length()-1);  }  **return** files.toString();  }    /\*\*  \* 上传文件到硬盘  \*  \* **@param** file  \* **@return**  \* **@throws** Exception  \*/  **private** String uploadFileAndCallback(MultipartFile file,HttpServletRequest request) **throws** Exception {    //附件  String uploadDir = request.getSession().getServletContext().getRealPath(execplanReportAttacheDir);  String oldFileName = file.getOriginalFilename();  //文件夹不存在则创建文件夹  File dirPath = **new** File(uploadDir);  **if** (!dirPath.exists())  dirPath.mkdirs();  // 获取文件扩展名:  String filename = file.getOriginalFilename();  String ext = "";  **int** pos;  **if** ((pos = filename.lastIndexOf('.')) != (-1)) {  ext = URLEncoder.*encode*(filename.substring(pos).trim(), "UTF-8");  }  //按照系统时间生成唯一文件名称存放到硬盘上  String fileName = generateTimeMillis() + ext;    String fileFullPath = FileHandleUtil.*fileFullPath*(uploadDir, File.*separator*, fileName);  //组拼文件位置  File uploadedFile = **new** File(fileFullPath);  //保存  FileCopyUtils.*copy*(file.getBytes(),uploadedFile);  **return** fileName+":"+oldFileName;  }  /\*\*  \* 由系统时间产生惟一标示的字符串  \*  \* **@return** String  \*/  **public** **synchronized** String generateTimeMillis() {  **return** String.*valueOf*(System.*currentTimeMillis*() + "\_" + **new** Random().nextInt(1000000));  }    /\*\*  \* 验证文件类型  \*  \* **@param** file  \* **@return** boolean  \*/  @SuppressWarnings("unchecked")  **private** String validateFile(HttpServletRequest request) {  MultipartHttpServletRequest multipartRequest = (MultipartHttpServletRequest)request;  **for** (Iterator it = multipartRequest.getFileNames(); it.hasNext();) {  String key = (String) it.next();  MultipartFile file = multipartRequest.getFile(key);  //校验 **TODO**  **if** (file.getSize() < 0 || file.getSize() > 2097152){ //单位是k, 2097152k==2M  **return** "附件不能大于2M";  }    //文件名为空, 不作文件名校验  String filename = file.getOriginalFilename();  **if** (filename == **null** || "".equals(filename)) {  **continue**;  }    String extName = filename.substring(filename.lastIndexOf(".")).toLowerCase();  **if** (extName.equals(".exe")) {  **return** "附件不能是.exe的可执行文件";  }  **if**(extName.equals(".lnk")) {  **return** "附件不能是.lnk类型的快捷方式";  }  }  **return** **null**;  }    /\*\*  \* 获取当前合作伙伴  \*  \* **@param** request  \* **@return**  \*/  **private** CopartnerOrg getCopartnerOrg(HttpServletRequest request) {  **long** copartnerOrgId = (Long)request.getSession().getAttribute(WebConfig.*COPARTNER\_ORG\_ID*);  CopartnerOrg copartnerOrg = (CopartnerOrg)copartnerOrgService.getByPrimaryKey(CopartnerOrg.**class**, copartnerOrgId);  **return** copartnerOrg;  }  **public** **void** setCopartnerOrgService(ICopartnerOrgService copartnerOrgService) {  **this**.copartnerOrgService = copartnerOrgService;  }  **public** **void** setExecPlanReportService(  IExecPlanReportService execPlanReportService) {  **this**.execPlanReportService = execPlanReportService;  }  **public** **void** setTopicExecPlanService(ITopicExecPlanService topicExecPlanService) {  **this**.topicExecPlanService = topicExecPlanService;  }  **public** **void** setExecplanReportAttacheDir(String execplanReportAttacheDir) {  **this**.execplanReportAttacheDir = execplanReportAttacheDir;  }  } |

## 文件下载

|  |
| --- |
| /\*\*  \* 文件下载 调用此方法, 输入文件目录和文件名, 即向客户端打开文件输出流  \*  \* **@param** response  \* **@param** dir: //文件所在目录, 格式例子: c:\file\doc 注意, 目录末尾没有目录分隔符  \* **@param** fileName //文件名  \* **@throws** Exception  \*/  **public** ModelAndView download(HttpServletRequest request, HttpServletResponse response) **throws** Exception{  //格式: 文件名:别名  String fileName = ServletRequestUtils.*getRequiredStringParameter*(request, "fileName");  String[] fileNames = fileName.split("\\:");    //文件名  String fileReailName = fileNames[0];  //别名  String alias = **null**;  **if**(fileNames.length==2){  alias = fileNames[1];  }**else**{  alias = fileReailName;  }    //得到文件存放绝对目录  String uploadDir = request.getSession().getServletContext().getRealPath(execplanReportAttacheDir);  //获取文件全名  String fileFullPath = FileHandleUtil.*fileFullPath*(uploadDir, File.*separator*, fileReailName);    **try** {  FileDownload.*download*(fileFullPath, alias, response);  **return** **null**;  } **catch** (Exception e) {  **return** **new** ModelAndView("/common/fileNotFond");  }  } |
| **package** com.ismp.web.util;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.InputStream;  **import** java.io.OutputStream;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** FileDownload {  /\*\*  \* 文件下载 方法, 输入文件全名, 即向客户端打开文件输出流  \*  \* **@param** fileFullPath: 文件全路径, 例如c:\file\doc\QQ.exe  \* **@param** alias: 文件别名, 例如: 文件保存在服务器上的名字是系统生成的一个随机码, 下载后显示的是用户上传时候的名字, 如果别名为空, 则以文件实际名字为别名  \* **@param** response  \* **@throws** Exception  \*/  **public** **static** **void** download(String fileFullPath, String alias, HttpServletResponse response) **throws** Exception {  File file = **new** File(fileFullPath);  **if** (alias==**null** || "".equals(alias)) {  alias = file.getName();  }  response.setCharacterEncoding("UTF-8");  response.addHeader("Content-Disposition", "attachment;filename=" + alias);  InputStream inputStream = **new** FileInputStream(file);  OutputStream outputStream = response.getOutputStream();  **int** i = 0;  **byte** b[] = **new** **byte**[1024];  **while** ((i = inputStream.read(b)) != -1) {  outputStream.write(b, 0, i);  }  inputStream.close();  outputStream.flush();  outputStream.close();  }  } |

# 用继承MultiActionController的controller实现继承SimpleFormController的对象绑定

|  |
| --- |
| @Override  **protected** **void** initBinder(HttpServletRequest request, ServletRequestDataBinder binder) **throws** Exception {  **super**.initBinder(request, binder);  binder.registerCustomEditor(Sex.**class**, **new** SexEditor());  binder.registerCustomEditor(Contact.**class**, **new** ContactEditor());//联系方式  binder.registerCustomEditor(TelecomAdminStatus.**class**, **new** TelecomAdminStatusEditor());//管理员状态:(停用 |启用)  }  //这个方法 有时可以去掉, 没弄明白  @Override  **protected** ServletRequestDataBinder createBinder(HttpServletRequest request, Object command) **throws** Exception {  TelecomAdmin admin = (TelecomAdmin)command; //因为session里面没有绑定对象, 所以每次绑定之前都需要创建绑定目标对象  admin.setContact(**new** Contact()); //当绑定对象的成员变量是对象类型时, 需要创建成员变量  **return** **super**.createBinder(request, admin);  }    /\*\*  \* 初始化 添加 和 修改页面  \* **@param** request  \* **@param** response  \* **@return**  \*/  **public** ModelAndView form(HttpServletRequest request, HttpServletResponse response, TelecomAdmin admin) **throws** Exception {  **long** id = ServletRequestUtils.*getLongParameter*(request, "id", 0); //页面表单提交过来的被操作用户ID  **if** (id == 0) {// 添加admin  admin.setContact(**new** Contact());  }**else** {// 修改admin  admin = (TelecomAdmin)telecomAdminService.getByPrimaryKey(TelecomAdmin.**class**, id);  **if** (**null** == admin.getContact()) {  admin.setContact(**new** Contact());  }  }    ModelAndView mav = **new** ModelAndView("/admin/form");  mav.addObject("parentFunctions", getParentFunctions(request));//可分配的所有功能  mav.addObject("parentAreas", getParentAreaList(request)); //可分配的所有区域  mav.addObject("admin", admin);  mav.addObject("sex", Sex.*values*());  **return** mav;  }    /\*\*  \* 添加 和 修改 管理员  \* **@param** request  \* **@param** response  \* **@return**  \*/  **public** ModelAndView save(HttpServletRequest request, HttpServletResponse response, TelecomAdmin admin) **throws** Exception {    TelecomAdmin telecomAdmin = findTelectomAdmin(request);  Role role = telecomAdmin.getRole();//取得当前用户的role  admin.setRole(roleService.getNextLevelRoleById(role.getId()));// 设置被操作用户的 角色    // 设置被操作用户的 function  **long**[] functions = ServletRequestUtils.*getLongParameters*(request, "functionId");  List<Function> list = **new** ArrayList<Function>();  **if** (functions != **null** && functions.length > 0) {  Function function;  **for** (**int** i = 0; i < functions.length; i++) {  function = **new** Function();  function.setId(functions[i]);  list.add(function);  }  }  admin.setFunctions(list);    //设置被操作用户的管理区域(area)  String[] areas = ServletRequestUtils.*getStringParameters*(request, "areaId");  Set<Area> set = **new** HashSet<Area>();  **if** (areas != **null** && areas.length > 0) {  Area area;  **for** (**int** i = 0; i < areas.length; i++) {  area = **new** Area();  area.setId(Long.*parseLong*(areas[i]));  set.add(area);  }  }  admin.setAreas(set);//设置area    String message = **null**;//给页面的提示消息  **long** id = ServletRequestUtils.*getLongParameter*(request, "id", 0); //获取被操作用户的ID  **if** (admin.getId()==0) { //当页面传过来的用户ID为0, 则添加用户  admin.setStatus(TelecomAdminStatus.*fromCode*(normalStatusCode));//设置新添加用户的状态为:状态  telecomAdminService.create(admin);    //记录日志  telecomAdminLogService.log(telecomAdmin, LogAction.*创建*.toString(), LogObject.*管理员*.toString(), LogUtil.*toLogString*(admin));    message = "创建成功!";  }**else** {//修改用户  TelecomAdmin oldAdmin = (TelecomAdmin)telecomAdminService.getByPrimaryKey(TelecomAdmin.**class**, id); //根据ID, 获取要修改的用户  //修改用户的部分信息  oldAdmin.setLoginUserName(admin.getLoginUserName());  oldAdmin.setName(admin.getName());  oldAdmin.setEmployeeNo(admin.getEmployeeNo());  oldAdmin.setSex(admin.getSex());  oldAdmin.setContact(admin.getContact());  oldAdmin.setNote(admin.getNote());  oldAdmin.setFunctions(admin.getFunctions());//设置功能权限  oldAdmin.setAreas(admin.getAreas()); //设置管理区域  telecomAdminService.update(oldAdmin);    //记录日志  telecomAdminLogService.log(telecomAdmin, LogAction.*修改*.toString(), LogObject.*管理员*.toString(), LogUtil.*toLogString*(admin));  message = "修改成功!";  }    **return** **new** ModelAndView("forward:list.do", "message", message);  } |

# Spring配置的bean中使用properties文件里配的值

|  |  |
| --- | --- |
| **第一步:** 在classpath下新建ftp.properties文件   |  | | --- | | ip=59.108.44.42  port=21  username=mpc  password=123465 | |
| **第二步:**  **指定properties文件位置**  <bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  <!--第一种:配置多个资源文件-->  <property name="locations">  <list>  <value>classpath:ismp-webapp.properties</value><!-- web层的配置 -->  <value>classpath:ismp-service.properties</value><!-- service层的配置 -->  <value>classpath:ismp-dao.properties</value><!-- dao层的配置-->  </list>  </property>  <!--第二种:配置一个资源文件  <property name="location" value="classpath:ftp.properties"/>  -->  <!-- 配置properties的默认值, 当properties文件中没有定义, 就是用这个默认值; 当properties文件中定义了key, 就覆盖掉这里的默认值 -->  <property name=*"properties"*>  <props>  <prop key=*"orderIconDir"*>/upload/product/orderIcon</prop>  <prop key=*"logoDir"*>/upload/product/logo</prop>  <prop key=*"orderIconDir"*>/upload/product/orderedIcon</prop>  </props>  </property>  </bean>  **注入**  <bean id="originalBillJob" class="com.ismp.web.task.OriginalBillJob">  <property name="originalBillService" ref="originalBillService" />  <property name="ftpUtil" ref="ftpUtil" />  <property name="ip" value="${ip}" />  <property name="port" value="${port}" />  <property name="username" value="{username}" />  <property name="password" value="${password}" />  </bean> |

# @ResponseBody中date格式化

|  |
| --- |
| import java.io.IOException;  import java.text.SimpleDateFormat;  import java.util.Date;    import org.codehaus.jackson.JsonGenerator;  import org.codehaus.jackson.JsonProcessingException;  import org.codehaus.jackson.map.JsonSerializer;  import org.codehaus.jackson.map.SerializerProvider;    /\*\*  \* @description 自定义返回JSON 数据格中日期格式化处理  \* @author aokunsang  \* @date 2013-5-28  \*/  public class CustomDateSerializer extends JsonSerializer<Date> {    @Override  public void serialize(Date value,  JsonGenerator jsonGenerator,  SerializerProvider provider)  throws IOException, JsonProcessingException {  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  jsonGenerator.writeString(sdf.format(value));  }  } |
| @JsonSerialize(using = CustomDateSerializer.**class**)  **public** Date getCurrentTime() {  **return** currentTime;  } |

# \_\_\_\_\_\_\_\_\_\_\_\_\_未整理\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 继承MultiActionController的类, 系统启动, 把全站使用的一些基础数据数据放入application

|  |
| --- |
| /\*\*  \* 系统启动, 把全站使用的一些基础数据数据放入application 比如: 管理门户主页目录, 产品LOGO图片路径 等; (这个方法在系统启动时自动执行)  \*/  @Override  **protected** **void** initApplicationContext(ApplicationContext context) {  ServletContext application = **this**.getServletContext();  application.setAttribute("managerHome", managerHome);  application.setAttribute("adDir", adDir);  } |

# 校验, 提示

|  |
| --- |
| **JSP**  用<form:form>标签绑定对象  <form:errors path=*"content"* cssStyle=*"color:red;"* /> |
| **controller**  @Override  **protected** **void** onBindAndValidate(HttpServletRequest request, Object command, BindException errors) **throws** Exception {  Knowledge knowledge = (Knowledge)command;  //对内容长度进行校验  **int** contentLength = knowledge.getContent().length();  **if** (contentLength <= 4) {  errors.rejectValue("content", **null**, "内容不能为空！");  }  **if** (contentLength >= 1000) {  // 方法说明rejectValue ([String](eclipse-javadoc:%E2%98%82=manager-webapp/C:%5C/Documents%20and%20Settings%5C/Administrator%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring%5C/2.5.6%5C/spring-2.5.6.jar%3Corg.springframework.validation(BindException.class%E2%98%83BindException~rejectValue~Ljava.lang.String;~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) field, [String](eclipse-javadoc:%E2%98%82=manager-webapp/C:%5C/Documents%20and%20Settings%5C/Administrator%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring%5C/2.5.6%5C/spring-2.5.6.jar%3Corg.springframework.validation(BindException.class%E2%98%83BindException~rejectValue~Ljava.lang.String;~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) errorCode, [String](eclipse-javadoc:%E2%98%82=manager-webapp/C:%5C/Documents%20and%20Settings%5C/Administrator%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring%5C/2.5.6%5C/spring-2.5.6.jar%3Corg.springframework.validation(BindException.class%E2%98%83BindException~rejectValue~Ljava.lang.String;~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) defaultMessage)  errors.rejectValue("content", **null**, "内容不能超过1000字节！");  }  } |
| **jsp到controller的配置文件**  <bean id=*"SIKnowledgeForm"*  class=*"com.ismp.web.manager.controller.siKnowledge.SIKnowledgeForm"*>  <property name=*"fileDir"* value=*"${siKnowledge.templates}"* />  <property name=*"telecomAdminService"* ref=*"telecomAdminService"* />  <property name=*"knowledgeService"* ref=*"knowledgeService"* />  <property name=*"formView"* value=*"/siKnowledge/form"* /> //校验失败, 跳转到这个页面  <property name=*"validateOnBinding"* value=*"true"* /> //是否要校验  </bean> |