Struts

1.struts2是一个mvc容器或mvc框架.由apache组织发明的开源框架，是struts的第二代产品，是在struts1和webwork基础上整合的全新框架，拦截器为mvc注入了全新的概念。Struts2有很多新特性，例如支持主题，继承DOJO框架，内置AJAX等。它产生的目的是解决一些servlet的缺点。

2.Servlet的缺点

(1)、配置一个servlet,在web.xml中需要些很多行代码，这样的写法在团队的合作中不可取

(2)、在servlet中，doGet和doPost方法有两个参数：request,response,这两个参数具有严重的容器依赖性，所以不便于测试

(3)、在一个servlet中写很多方法，怎么样去调用。Servlet中采用传递参数的形式，分解到每个方法中。

(4)、在servlet中完成所有的逻辑，结构不好，例如用户注册的例子：

用户注册：

1、获取表单的数据

2、完成图片的上传

3、完成校验

4、完成验证码

5、保存用户

(5)、request获取页面表单的数据太麻烦了

(6)、url处理起来比较费劲

3. servlet重构

重构的目标：

1、 只写一个serlvet或者过滤器，我们这里选择过滤器。

2、 不用再写任何的servlet，这样在web.xml中写的代码就很少了。

3、 原来需要写serlvet,现在改写action。

4、 在action中把HttpServletRequest参数和HttpServletResponse参数传递过去。

5、 在过滤器中通过java的反射机制调用action。

对servlet进行重构：

(1)、在web.xml中写一个过滤器

1、利用java的反射机制调用所有的service

2、把request,response的参数传递到service层

3、response过程

(2)、在servlet中利用java的反射机制动态的调用service层的方法

(3)、对错误处理机制进行重构

重构servlet的总结

1、service的方法做到了动态的调用

2、因为所有的service方法都在一个filter中动态的调用，所以只要service的方法中出现错误，就可以错误统一的处理

3、在web.xml中声明了一个总控的过滤器，使得在web.xml中的配置大量减少了

4．第一个例子的说明

（1）在web服务器启动的时候，执行的是过滤器中的init方法。在这里回顾

一个问题：一个过滤器中的init方法在什么时候执行？总共执行几次？

（2）在启动服务器的时候，执行了过滤器中的init方法，加载了三个配置文件

struts-default.xml、struts-plugin.xml、struts.xml

（3）因为这三个文件的dtd约束是一样的，所以如果这三个文件有相同的项，后面覆盖前面的。因为先加载前面的，后加载后面的。

（4）struts.xml文件必须放在src下才能找到。

5.struts2中配置文件的相关内容

（1）struts2中的package具有继承关系，struts.xml中所有的包都继承自struts-default.xml

（2）package中的namespace命名空间的匹配策略，采用路径逐阶向前的匹配规则。

（3）action中采用通配符的配置方式进行，name中的\*和后面的{1}{2}{3}…对应。第二种方式是struts2默认识别的action！methodName的方式。

（4）result的配置，name与action类中的返回值相对应，type值指明结果集的类型，常用的结果集是dispather、redirect和redirectAction，可以在struts-default.xml文件中查找对应的匹配项。

（5）action中的class属性可以不写，默认执行ActionSupport中execute中的内容，从struts-default.xml文件中可以看出

<default-class-ref class="com.opensymphony.xwork2.ActionSupport" />

如果引入了spring框架，这里可以从spring容器中获取。

（6）拦截器的配置

1、如果有一个比较复杂的逻辑，在没有拦截器的情况下，会把这些需求全部写在action的方法中，这样会导致方法的结构不好

2、会把一些重用性的内容写在拦截器中

3、要把这些重用性比较高的代码以很好的组织形式结合起来，一个拦截器是不够的，所以需要拦截器栈

Interceptor只能配置在package中，所以不能配置全局的拦截器；

需要在<interceptors>配置<interceptor>标签指明自定义的拦截器，并申明一个自定义的拦截器栈，栈中引入默认的拦截器栈defaultStack，最后在修改默认的拦截器栈为自定义的拦截器栈。这样，如果一个package要使用自定义的拦截器栈只要通过package继承机制就可以实现了。写的拦截器类必须实现Interceptor接口

intercept(){

return invocation.invoke();//往下继续执行

}

6.action的三种写法

（1）简单的javaBean

（2）实现接口Action，需要实现其中的execute方法，在action中定义了一些返回值常量可以直接使用。public String execute() throws Exception;

（3）继承默认的实现类ActionSupport，实现了Action等接口，提供默认的一些实现。在ActionSupport类中做了很多的操作，比如：验证器、国际化等。程序员只要重写execute方法即可。

public class ActionSupport implements Action, Validateable, ValidationAware, TextProvider, LocaleProvider, Serializable {

7.ServletActionContext:是Servlet和Struts2通信的桥梁，可以获得servlet容器的上下文。

8.struts2中action的模式是多例的。请求action几次，就可以创建几次对象，所以action的创建是原型模式（多例）。action的方法如果以get开头，会带来安全性的隐患？？？？

9.模型驱动和属性驱动

（1）属性驱动

1、概念

可以利用属性驱动获取页面表单元素的内容

2、步骤

1、在action中声明属性，属性的名称和页面元素中name属性的值保持一致

2、action中的属性必须有set和get方法

3、原理

ParametersInterceptor:

1、把页面上表单的元素获取，封装成了一个TreeMap

2、遍历TreeMap

for(Entry<String,Object[]> entry:treeMap){

valueStack.setValue(entry.getKey(),entry.getValue()[0]);

}

说明：

1、在执行该拦截器的时候，action已经在栈顶了，那么action的属性在对象栈中

2、通过valueStack.setValue的方法可以改变对象栈的属性的值

（2）模型驱动

1、如果页面上元素内容太多，用属性驱动实现，action中代码就会很庞大，这个时候可以考虑用模型驱动来实现

2、步骤

1、action实现一个接口ModelDrivern

2、在action中声明一个属性，该属性会封装页面中的数据，并且用new的方法给该属性创建对象

3、填充接口中的方法getModel,返回该属性的值

public class UserAction implements ModelDriver<User>{

private User model = new User();

public User getModel(){

return this.model;

}

}

3、原理

ModelDriverInterceptor{

interceptor(){

Object action = invocation.getAction();

if (action instanceof ModelDriven) {//该类必须实现ModelDriven

ModelDriven modelDriven = (ModelDriven) action;

ValueStack stack = invocation.getStack();

Object model = modelDriven.getModel();

stack.push(model);//把模型驱动的对象放入到栈顶

}

}

在由ParameterInterceptor调用，结合属性驱动的原理。

10.valuestack

（1）struts2中存储数据的一种内存结构，用于ognl表达式把值显示出来。ValueStack分成对象栈和map栈。

（2）通过什么样的方式可以得到valueStack

1、ActionContext.getContext().getValueStack()

2、ServletActionContext.getValueStack(ServletActionContext.getRequest());

3、(ValueStack)ServletActionContext.getRequest().getAttribute("struts.valueStack")

（3）valueStack的内存结构

（4）valueStack的生命周期 一次请求

（5）对map栈的操作

1、request,response,parameters,application等都在map栈中，分别的数据结构为

{"request",RequestMap} {"session",SessionMap} {"application",ApplicationMap}

2、可以通过如下方法把数据存放在request,session,application域中

ServletActionContext.getRequest().setAttribute("msg\_request", "msg\_request");

ServletActionContext.getServletContext().setAttribute("msg\_application", "msg\_application");

ServletActionContext.getRequest().getSession().setAttribute("msg\_session", "msg\_session");

3、可以通过如下方法吧数据直接存放在map栈中

ActionContext.getContext().put("aaaa", "aaaaa");

valueStack.getContext().put("bbb", "bbb");

（6）对对象栈的操作

1、把一个数据存放在对象栈的尾部

valueStack.getRoot().add("aaaaa");

2、把一个数据存放在对象栈的栈顶

valueStack.getRoot().add(0,"adsf");

valueStack.push("rtrt");

3、把一个数据封装成map,把map放入到栈顶

valueStack.set("aaasdf", "asfd");

4、获取对象栈栈顶的元素

valueStack.peek();

valueStack.getRoot().get(0);

5、移除对象栈栈顶的元素

valueStack.getRoot().remove(0);

valueStack.pop();

11、ognl表达式

1、把一个数据放入到valueStack中，在页面上可以利用ognl表达式把该数据提取出来

2、ognl表达式总体的规则：如果一个数据存放在map栈中，则加#访问，如果一个数据在对象栈中，则不用加#访问

3、详细说明

1、s:property标签

1、如果没有value属性，则默认输出对象栈的栈顶的元素

2、<s:property value="#session.aa">输出map栈中session域中aa的值

2、s:iterator标签

1、迭代遍历集合

1、集合的类型：List,Set,Map,Object[]

2、value属性的值能直接跟ognl表达式

3、当前正在迭代的元素在栈顶存放

4、要熟练使用该标签迭代二重或者三重集合

2、属性var="a"表示当前正在迭代的元素在map栈中存放一份，key值为"a"

3、当迭代map的时候，当前正在迭代的元素是Entry

<s:iterator value="#map">

<s:property value="key"/>,<s:property value="value.pid"/>

<s:property value="key"/>,<s:property value="value.username"/>

</s:iterator>

4、属性status，代表当前迭代元素的一些性质

int getCount() 返回当前迭代的元素个数

int getIndex() 返回当前迭代元素的索引

boolean isEven() 返回当前迭代元素的索引是否是偶数

boolean isOdd() 返回当前迭代元素的索引是否是奇数

boolean isFirst() 返回当前迭代元素是否为第一个元素

boolean isLast() 返回当前迭代元素是否为最后一个元素

3、s:debug标签可以输出valueStack的内存快照

12.struts2提供了两种校验机制。

13.防止表单重复提交，原理是利用struts2内置的拦截器token实现的

<interceptor name="token" class="org.apache.struts2.interceptor.TokenInterceptor"/>

14.文件的上传

在struts2中，利用拦截器fileUpload实现了文件上传

15.类型转换

建立一个转化器类ListConverter

//把页面上的选择的内容填充到List<String>中

public Object convertFromString(Map context, String[] values, Class toClass) {

List<String> aihaos = new ArrayList<String>();

for(String s:values){

aihaos.add(s);

}

return aihaos;

}

建立一个配置文件xwork-conversion.properties，该配置文件放在src下

java.util.List=cn.itheima03.struts2.converter.action.ListConverter

key值为转化后的类型，value值为转化器

16．总结

1、struts2的过滤器解决的问题：

不用在web.xml文件中配置很多行代码

2、struts2的拦截器

ParameterInterceptor

ModelDriverInterceptor

FileUploadInterceptor

TokenInterceptor

把一些重用性的，mvc框架要用到的一些内容写到拦截器中了

3、通配符映射解决的问题：

在配置文件中通过简单的配置可以在多个action中，在action中的多个方法中实现灵活的调用

4、用模型驱动和属性驱动代替request.getParameter,这样代码更简单明了了

5、在action中没有出现与容器相关的类，所以便于测试

6、以松耦合的方式把struts2与servlet结合在一起了

7、struts2配置文件中采用了package的继承机制，可以把action、interceptor等继承过来

8、值栈和ognl表达式

1、如果把数据存放在值栈的对象栈中，页面上可以根据name的属性进行回显

2、因为在对象栈中，一个类的属性直接暴漏在对象栈中，所以根据ognl表达式可以很容易的输出其内容

3、ognl表达式在迭代的时候iterator,可以迭代Collection,Map,Object[]

课堂外的补充内容

1. struts2中的零配置

零配置并不是真的“零配置”，只是没有传统的配置文件struts.xml，但需要使用@Annotation配置。即使用注解配置。

1. 需要导入struts2-convention-plugin-2.1.8.1.jar包
2. 通过注解实现action的零配置
3. web.xml中指定Action目录
4. 访问<http://localhost:8080/itheima_struts2_login/login/login!login.action>，http://localhost:8080/itheima\_struts2\_login/config-browser/actionNames.action。
5. struts2能自动转化Date类型的数据，struts2有一个默认的日期类型转化器，但是功能有限，只能转化形如“2008-09-02”的日期，如果转化失败，还会抛出异常。可以自定义一个日期转化器。
6. （1）案例中的通配符使用<package name=”\*Book” class=”” method=”{1}”>的方式

（2）jsp页面中，<head>标签中，<struts:head theme=”ajax”>来指定主题。

<!-- <s:head theme="html"/>struts2中theme属性包括xhtml,html,simple,ajax 。默认是xhtml -->

（3）url标签：<a href=”<struts:url action=”initAddBook”/>”>该标签会在运行时将“.action”自动添加在后面。默认会增加，也可以通过配置属性includeParams=”none”禁止自动追加参数。也可以在struts.properties文件中配置所有的都禁止自动追加

struts.url.includeParams=none

（4）在进行iterator遍历的时候，使用了el表达式${book.name}。早期的struts2支持el表达式，但从struts2.0.11起，struts2标签库不再支持El表达式。

1. 数据类型转换器
2. action中获取request和response
3. 利用ServletActionContext的静态方法获取，利用ThreadLocal来维护不同线程的Servlet对象。这种方式也叫做非注射方式（非IOC方式）
4. 利用相关的Aware接口，其实是一种拦截器，注射方式（IOC方式）与request对应的接口是ServletRequestAware，实现接口之后即可通过setServletRequest()方法获取request对象。
5. 使用validator校验数据
6. 使用xml配置数据校验Validator（服务器端）
7. 配置客户端Validator（客户端）支持客户端JavaScript校验，方法很简单在<s:form>标签中加validate=”true”属性即可。
8. 使用@java注解配置Validator。使用@Validations注解在其中指明校验规则
9. OGNL表达式
10. 能够访问静态属性与静态方法。方法是在类名前、方法前加上@。[如@java.lang.Math@PI,@java.lang.String@format(‘foo%s’,’bar’)](mailto:如@java.lang.Math@PI,@java.lang.String@format('foo%25s','bar'))
11. #的作用

能够访问ognl上下文与ActionContext资源

用于过滤或筛选集合，books.{?#this.price<20}

构造map，如#{‘fool’:’bar1’,’fool2’,’bar2’}

（3）%的作用

%用于表示某字符串作为一个ognl表达式

（4）$的作用

主要用于在资源国际化文件中或者struts.xml中引用ognl表达式。${getText(filename)}

（5）div、submit、a标签实现无刷新获取数据

（6）无刷新实现登录、注销（实现的ajax效果）

（7）拦截器interceptor类似于Filter，在执行Action方法前后执行。拦截器是一种AOP面向切面思想的编程方式。它提供一种机制是开发者能把相对对立的代码抽象出来，配置到Action前后执行。

Timer计时拦截器——统计每个Action方法运行所需的时间

execAndWait执行等待拦截器——如果某个Action运行时间比较长，浏览器会等待而显示空白，这时可以通过这个拦截器显示服务器正忙的界面。这是显示繁忙页面的简单方式,是struts2内置的解决方案，更多地使用Ajax技术来实时显示繁忙页面。

项目中遇到的问题及处理

1. 当自定义java bean的构造函数时候,hibernate中使用session.get(Bean.class,1L)时候，会报无参构造函数不存在的错误。原因是，hibernate底层利用反射机制去构造bean的实例化对象，java默认会构造无参的构造函数，当自定义构造函数时候，默认的无参构造函数就不存在了，所以会报错。
2. 在写dao层的根据sql语句查询结果时候，使用的方法是session.createSqlQuery(sqlString)，该方法返回的是Object数组，在前台页面或者service层使用的时候会存在问题。Hibernate提供了转成java bean对象的方法，就是session.createSqlQuery(sqlString).addEntity(Bean.class)就将Ojbect数组中的Object对象转换成Bean的对象。

回想：Hql查询中也存在类似的转换成bean对象的方法。

1. 单表查询时候，取表的一部分字段时候select cid,cname from Classes得到的是Object数组，转换为Bean的方法是 select new …Classes(cid,cname) from Classes
2. 一对多查询时候，迫切内连接的方式from Classes c inner join fetch c.students s就把Set<Student>作为Classes的一个对象

权限部分

1. 通过监听器监听application域对象，在创建该域对象时候，把所有权限加载到map中，当访问页面时候，根据需要取出权限加以判断
2. 通过拦截器，配置全局的拦截器，当访问任何一个页面时候都要经过该拦截器，在拦截器中把所有的权限加到application域中

关于权限的问题：当权限进行维护，动态变化时候，哪种方式更好一些呢。第二种方式不需要重启，而第一种方式需要重启。

原来做过的OA项目是根据控制菜单来控制权限访问的。

实践，修改过用户权限之后，需要重新登录一下，用户的权限就显示了。原因是重新登录的时候，会把用户的权限读取到session中，在生成菜单的页面获取session中的权限，然后生成对应的菜单，就达到控制权限的目的了。