

# 一、框架概述

## 1、框架的意义与作用：

所谓框架，就是把一些繁琐的重复性代码封装起来，使程序员在编码中把更多的经历放到业务需求的分析和理解上面。

特点：封装了很多细节，程序员在使用的时候会非常简单。

## 2、五大框架：

Struts2，Hibernate，Spring , SpringMvc, Mybatis(Ibatis)

SSH

SSM/SSI

## 3、学好框架：

由于框架中细节很多，知识点比较零散，课后总结和做好笔记就变得尤为重要。

# 二控制器：MVC中的控制器

## 1、MVC回顾

M：Model 模型，封装数据。javabean

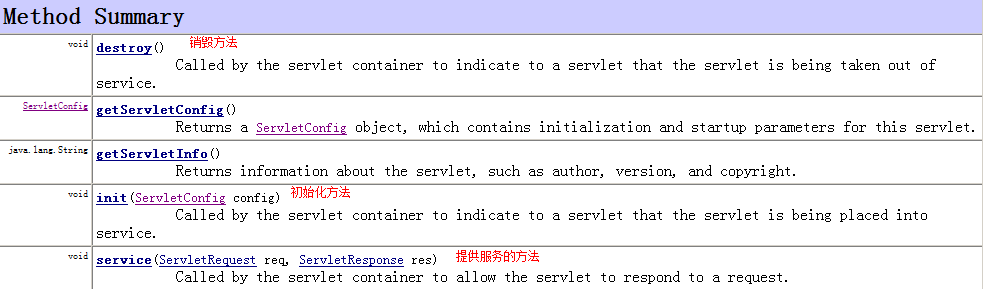
V：view 视图，展示界面。jsp

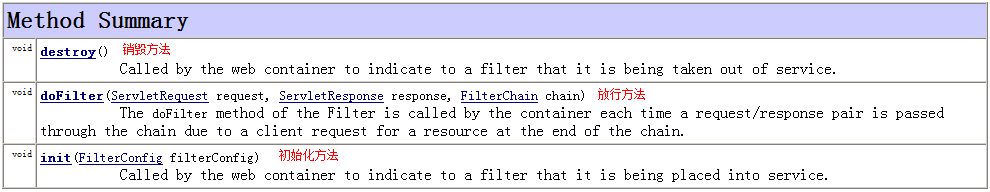
C：Controller 控制器，控制程序流程。Servlet

分层：为了简化企业级开发的 而提出来的分层，每一层有相应的功能。

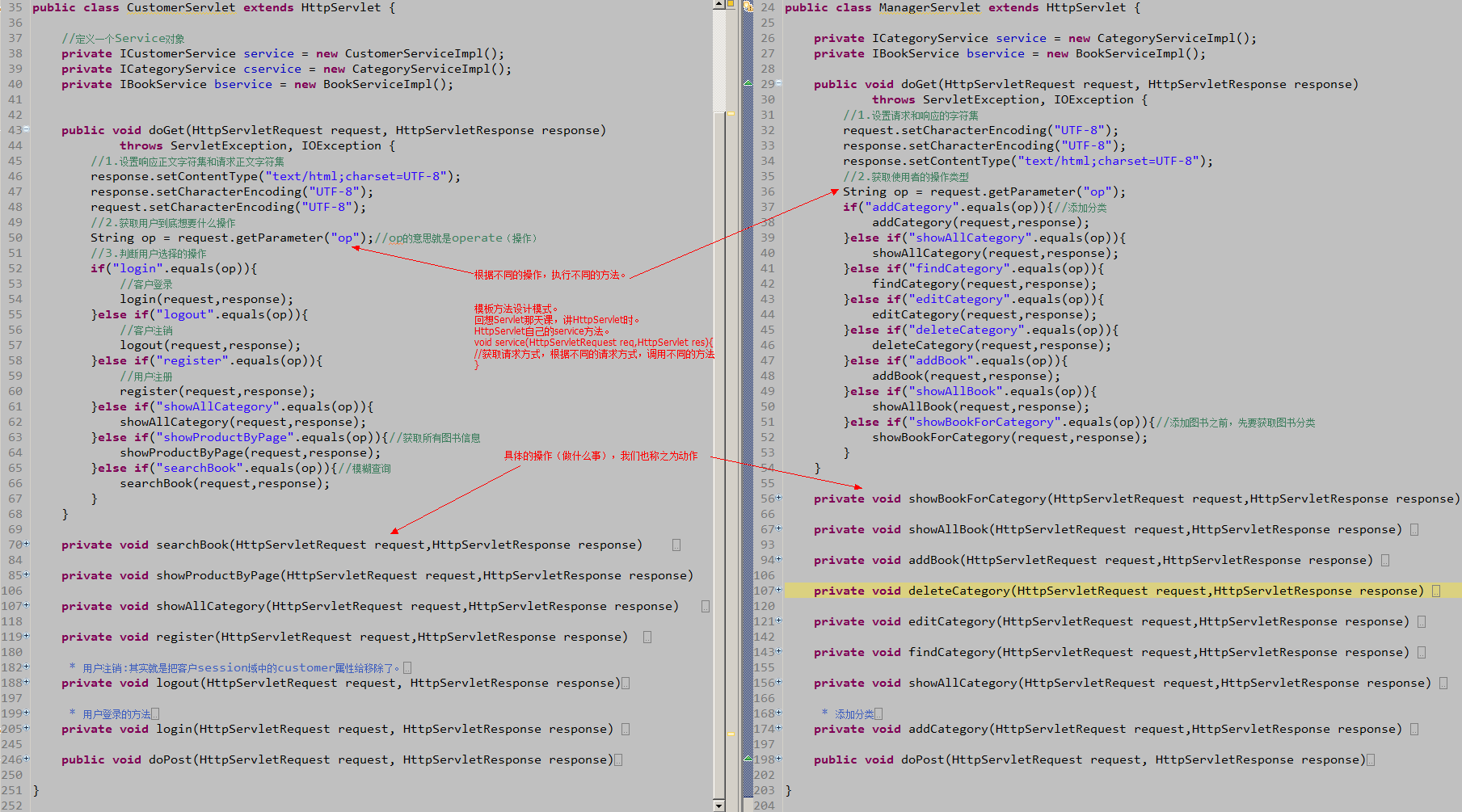
## 2、Servlet和Filter

Servlet:



Filter：

# 三、案例中的问题



**规律：**

**1、获取请求正文，用户要做什么事情**

**2、根据不同的请求，做出不同的判断**

**3、执行具体的方法代码（动作）**

**4、转向页面，展示给用户**

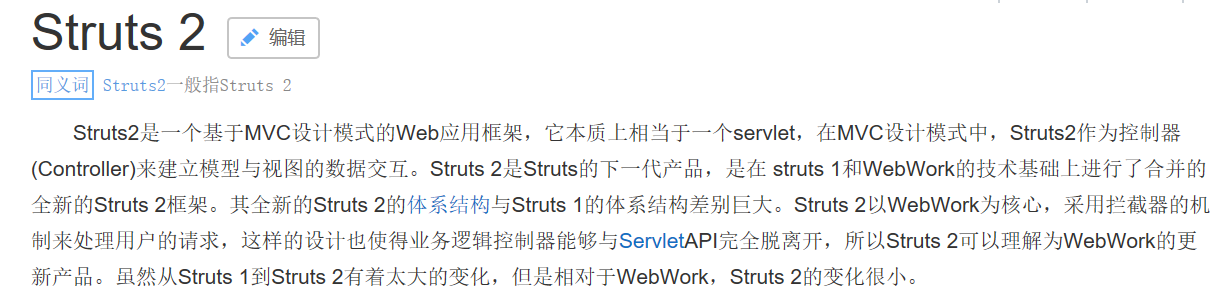
**缺陷：**

**1、重复性劳动太多，具体的执行代码也有冗余代码。**

**2、到底要做什么全是硬编码，像用户要做什么事情，对应执行什么代码，可以写在配置文件中。**

**3、具体的代码方法放到了控制器中，过于臃肿。**

# 四、Struts2简介



## 1、Struts2概述

Struts2是Apache发行的MVC开源框架。注意：充当控制器层或者说是web层的框架

## 2、Struts2的来历

Struts1：也是apache开发的一套mvc的开源框架。在2005年之前非常流行。

弊端：Struts1的核心控制器就是一个Servlet。随着使用者的增多，弊端开始出现。

Struts2：在long long ago，有一个设计超前的框架XWork，后来推出了XWork1和WebWork2。Struts2就是apache和OpenSymphony组织合并开发出来。

里面包含了WebWork2的核心及Struts的一些特性和功能。除此之外，和Struts1没有任何关系了。

# 五、搭建Struts2开发环境

## 1、下载Struts2开发包

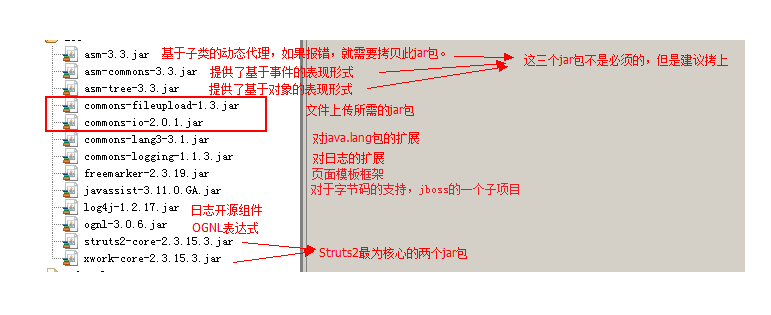
http://struts.apache.org

## 2、开发包目录结构



## 3、搭建开发环境

### 3.1、拷贝必要jar包到classpath中

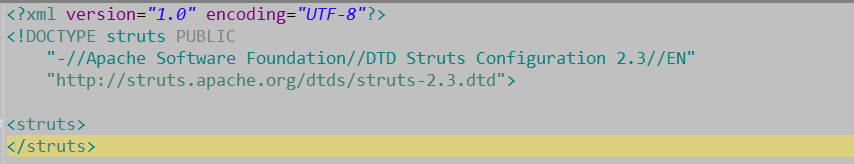


拷贝哪些jar包？

找到Struts2自带的例子中，struts-blank的war包，在它的\WEB-INF\lib目录下的jar包全拷贝。

### 3.2、建立Struts2的配置文件

at the top of classpath（在最顶层的构建路径）,建立一个默认名称为**struts.xml**的配置文件。



注意：

1.文件名大小写。

2.创建位置。

3.该名称允许修改，但是我们一般不改。

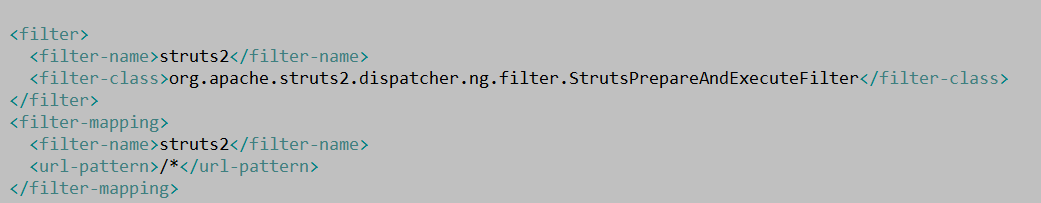
### 3.3、配置控制器

a、配置位置：在web.xml中

b、配置什么： struts2已经写好了的一个过滤器。

结论：

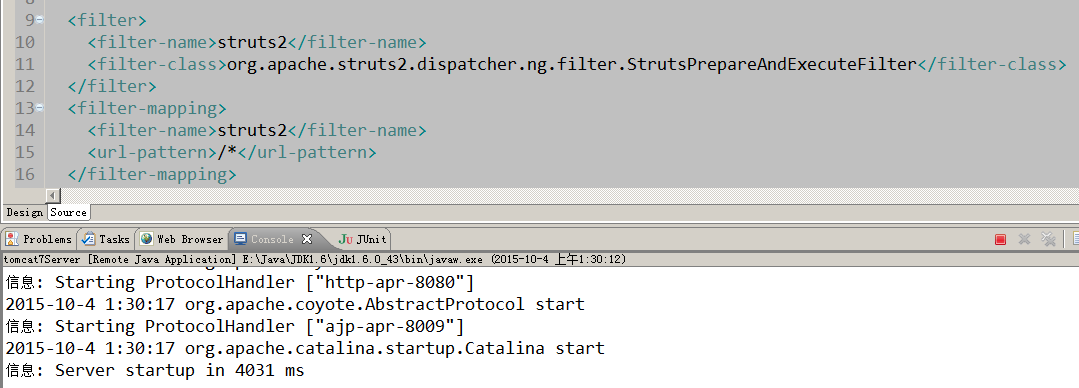
struts2比struts1优秀的一个体现就是，它用了更为强大的过滤器作为控制器了。



|  |
| --- |
| <filter>  <filter-name>struts2</filter-name>  <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>struts2</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

### 3.4、验证是否成功

部署应用，启动Tomcat，不报错表示搭建成功。



# 六、第一个Struts2案例

以下步骤是日后实际开发中经常重复的。

## 1、建立一个jsp文件



## 2、在struts.xml文件中配置

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE struts PUBLIC  "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"  "http://struts.apache.org/dtds/struts-2.3.dtd">    <struts>  <package name="default" namespace="/" extends="struts-default">  <action name="hello" class="com.qf.action.HelloAction" method="sayHello">  </action>  </package>  </struts> |

## 3、建立动作类和动作方法

Action类是动作类，是Struts2处理请求，封装数据，响应页面的控制器。需要自己编写。

|  |
| --- |
| public String sayHello(){  System.out.println("Hello Struts2!!");  return null;  } |

## 4、结果视图页面

## 5、测试：

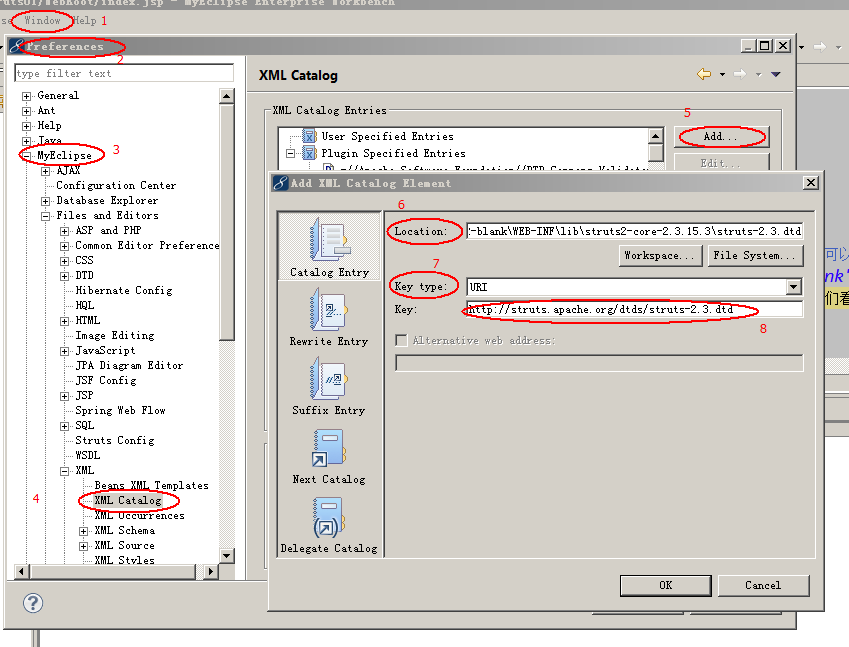
## 6、关于struts.xml没有提示的问题

分析原因：没有找到对应的dtd约束文件。

解决办法：

a.上网

b.不能上网：



## 七、Struts2的配置文件加载的顺序

### 1、加载时机：

当应用被tomcat加载的时候，struts2的配置文件就已经被加载过了。

### 2、加载顺序

顺序 配置文件名 所在位置 说明

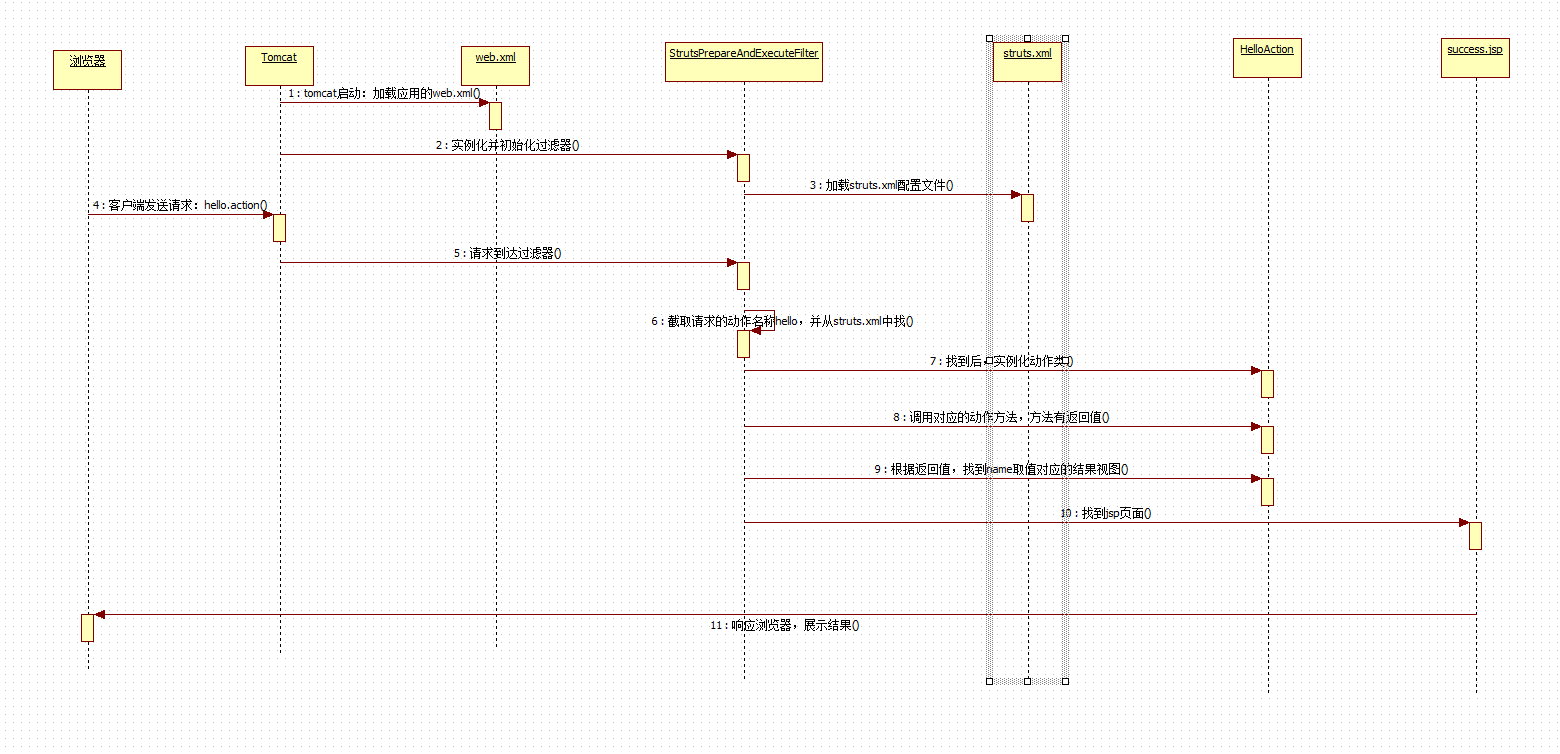
|  |
| --- |
| 1 default.properties struts2-core-2.3.15.3.jar\org\apache\struts2 不能修改  struts.i18n.encoding=UTF-8 解决乱码  struts.multipart.maxSize=2097152 上传文件的一个最大值 2M  struts.action.extension=action,, struts2的action的磨人的访问的后缀名  struts.serve.static.browserCache=true 告诉浏览器是否把一些静态的资源缓存到浏览器  struts.devMode = false 开发者 模式 会提示一些比较详细的错误信息  2 struts-default.xml struts2-core-2.3.15.3.jar 不能修改  3 strtuts-plugin.xml 在struts2提供的插件jar包中 不能修改  4 struts.xml 我们的应用中 我们修改的：推荐  5 struts.properties 我们的应用中 我们修改的  6 web.xml 我们的应用中 我们修改的，可以给过滤器配置参数  注意：配置信息会被覆盖。 |
|  |

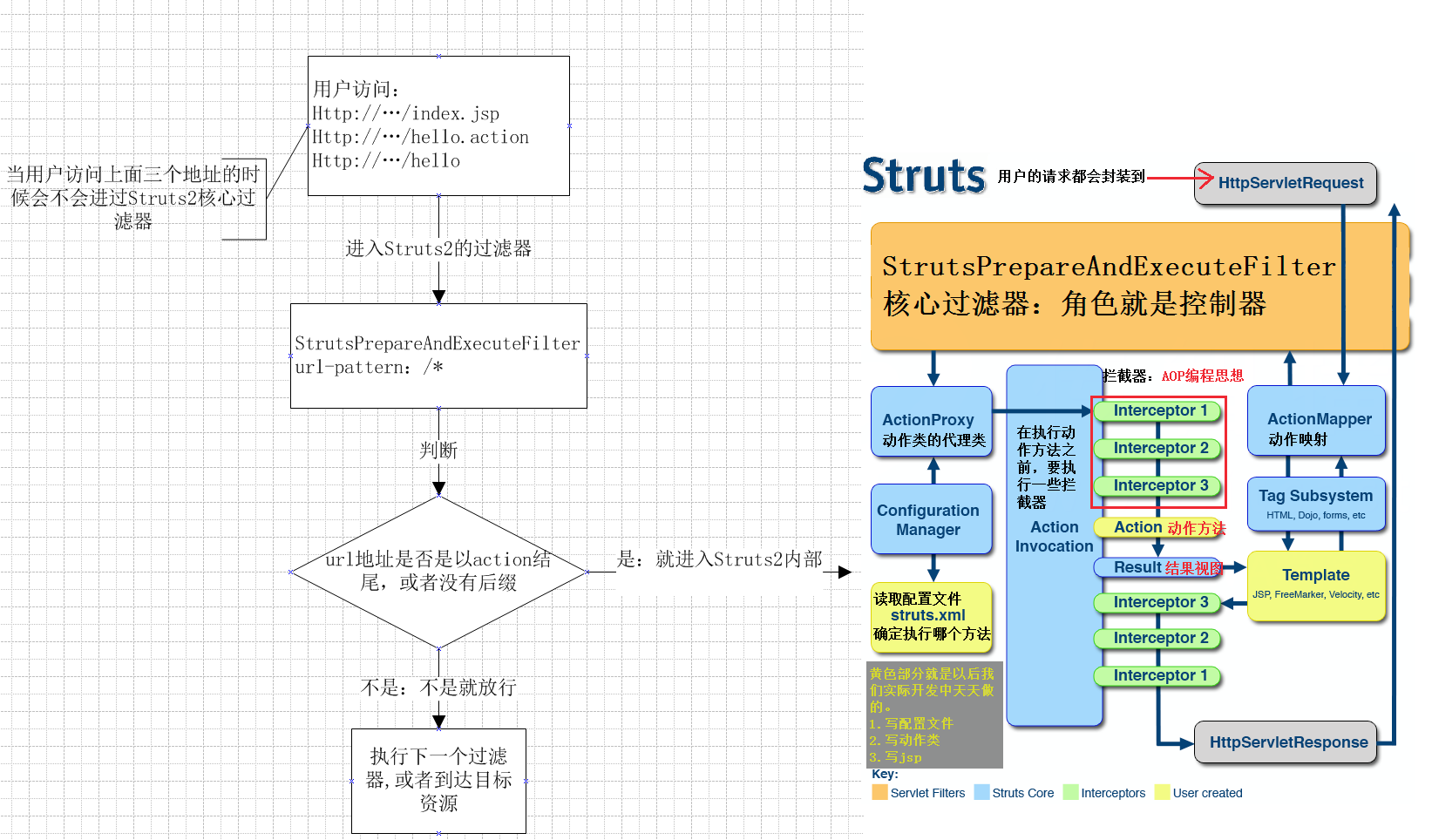
**注意：**

1、Struts2提供了两种配置的方式。一种是key=value的方式，即使用.properties文件。另一种是xml文件配置。我们推荐使用xml文件（它能描述层级关系）。

|  |
| --- |
| 重点了解的配置文件  \* default.properties -- 在org/apache/struts2/目录下，代表的是配置的是Struts2的常量的值  \* struts-default.xml -- 在Struts2的核心包下，代表的是Struts2核心功能的配置（Bean、拦截器、结果类型等）  \* struts.xml -- 重点中的重点配置，代表WEB应用的默认配置，在工作中，基本就配置它就可以了！！（可以配置常量）  \* web.xml -- 配置前端控制器  Struts2框架的核心是StrutsPrepareAndExecuteFilter过滤器，该过滤器有两个功能  Prepare -- 预处理，加载核心的配置文件  Execute -- 执行，让部分拦截器执行    StrutsPrepareAndExecuteFilter过滤器会加载哪些配置文件呢？  通过源代码可以看到具体加载的配置文件和加载配置文件的顺序  init\_DefaultProperties(); -- 加载org/apache/struts2/default.properties  init\_TraditionalXmlConfigurations(); -- 加载struts-default.xml,struts-plugin.xml,struts.xml  init\_LegacyStrutsProperties(); -- 加载自定义的struts.properties.  init\_CustomConfigurationProviders(); -- 加载用户自定义配置提供者  \* init\_FilterInitParameters() ; -- 加载web.xml |

### 3.第一个案例的执行过程



****

|  |
| --- |
| 1. 执行的流程  编写的页面，点击超链接，请求提交到服务器端。  请求会先经过Struts2的核心过滤器（StrutsPrepareAndExecuteFilter）  过滤器的功能是完成了一部分代码功能  就是一系列的拦截器执行了，进行一些处理工作。  咱们可以在struts-default.xml配置文件中看到有很多的拦截器。可以通过断点的方式来演示。  拦截器执行完后，会根据struts.xml的配置文件找到请求路径，找到具体的类，通过反射的方式让方法执行。    2. 总结  JSP页面-->StrutsPrepareAndExecuteFilter过滤器-->执行一系列拦截器（完成了部分代码）-->执行到目标Action-->返回字符串-->结果页面（result）-->页面跳转 |

## 八.入门总结之struts.xml配置文件的配置

### 1. <package>标签，如果要配置<Action>的标签，那么必须要先配置<package>标签，代表的包的概念

包含的属性

name -- 包的名称，要求是唯一的，管理action配置

extends -- 继承，可以继承其他的包，只要继承了，那么该包就包含了其他包的功能，一般都是继承struts-default

namespace -- 名称空间，一般与<action>标签中的name属性共同决定访问路径（通俗话：怎么来访问action），常见的配置如下

namespace="/" -- 根名称空间

namespace="/aaa" -- 带有名称的名称空间

abstract -- 抽象的。这个属性基本很少使用，值如果是true，

把包声明为抽象包，抽象包就是用来被继承的。只要是没有<action>元素的包，就可以声明为抽象包。那么编写的包是被继承的

### 2. <action>标签

代表配置action类，包含的属性

name -- 和<package>标签的namespace属性一起来决定访问路径的

class -- 配置Action类的全路径（默认值是ActionSupport类）

method -- Action类中执行的方法，如果不指定，默认值是execute

### 3. <result>标签

action类中方法执行，返回的结果跳转的页面

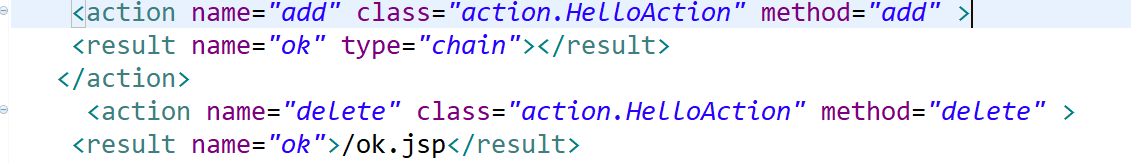
name -- 结果页面逻辑视图名称

type -- 结果类型（默认值是转发dispatcher，也可以设置其他的值）

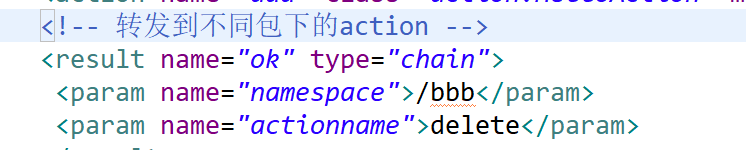
dispatcher 转发到页面 默认值

chain 转发到action

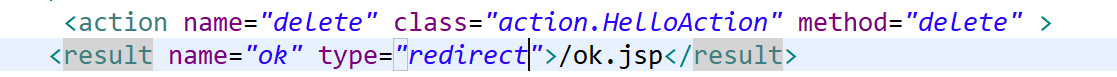
1. 转发到相同包下的action



1. 转发到不同包下的action

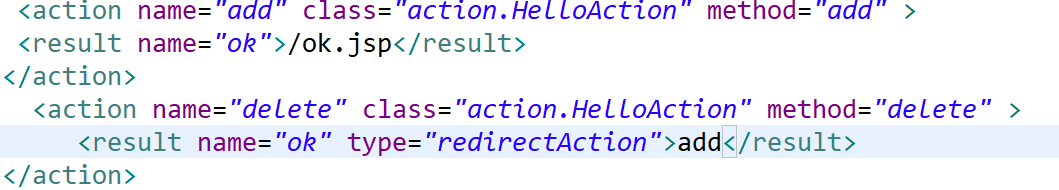


redirect: 重定向到页面

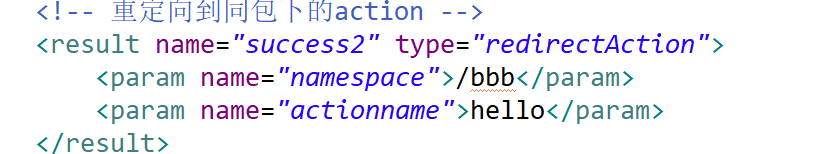


redirectAction:重定向到action

1.重定向到当前包下的action



2.重定向到其他包下的action



## 九．入门总结之Struts2配置常量

### 1. 可以在Struts2框架中的哪些配置文件中配置常量？

struts.xml（必须要掌握，开发中基本上就在该配置文件中编写常量）

<constant name="key" value="value"></constant>

web.xml

在StrutsPrepareAndExecuteFilter配置文件中配置初始化参数

注意：后加载的配置的文件的常量会覆盖之前加载的常量！！

### 2. 需要大家了解的常量

struts.i18n.encoding=UTF-8 -- 指定默认编码集,作用于HttpServletRequest的setCharacterEncoding方法

struts.action.extension=action,, -- 该属性指定需要Struts 2处理的请求后缀，该属性的默认值是action，即所有匹配\*.action的请求都由Struts2处理。如果用户需要指定多个请求后缀，则多个后缀之间以英文逗号（,）隔开

struts.serve.static.browserCache=true -- 设置浏览器是否缓存静态内容,默认值为true(生产环境下使用),开发阶段最好关闭

struts.configuration.xml.reload=false-- 当struts的配置文件修改后,系统是否自动重新加载该文件,默认值为false(生产环境下使用)

struts.devMode = false -- 开发模式下使用,这样可以打印出更详细的错误信息

## 十.入门总结之指定多个struts的配置文件（了解）

1. 在大部分应用里，随着应用规模的增加，系统中Action的数量也会大量增加，导致struts.xml配置文件变得非常臃肿。

为了避免struts.xml文件过于庞大、臃肿，提高struts.xml文件的可读性，我们可以将一个struts.xml配置文件分解成多个配置文件，然后在struts.xml文件中包含其他配置文件。

2. 可以在<package>标签中，使用<include>标签来引入其他的struts\_xx.xml的配置文件。例如：

<struts>

<include file="struts-part1.xml"/>

<include file="struts-part2.xml"/>

</struts>

3. 注意注意注意（重要的事情说三遍）：<include file="com/fm/demo2/struts-part1.xml"/>

## 十一. 技术分析之Action类的三种写法

### 1. 配置文件学习完成，下面的重点是Action类的三种写法

#### 1. Action类就是一个POJO类

什么是POJO类，POJO（Plain Ordinary Java Object）简单的Java对象.简单记：没有继承某个类，没有实现接口，就是POJO的类。

#### 2.Action类可以实现Action接口

Action接口中定义了5个常量，5个常量的值对应的是5个逻辑视图跳转页面（跳转的页面还是需要自己来配置），还定义了一个方法，execute方法。

大家需要掌握5个逻辑视图的常量

SUCCESS -- 成功.

INPUT -- 用于数据表单校验.如果校验失败,跳转INPUT视图.

LOGIN -- 登录.

ERROR -- 错误.

NONE -- 页面不转向.

#### 3.Action类可以去继承ActionSupport类（开发中这种方式使用最多）

## 十二.技术分析之Action的访问

1. 通过<action>标签中的method属性，访问到Action中的具体的方法。

传统的配置方式，配置更清晰更好理解！但是扩展需要修改配置文件等！

具体的实例如下：

页面代码

<a href="${pageContext.request.contextPath}/addBook.action">添加图书</a>

<a href="${pageContext.request.contextPath}/deleteBook.action">删除图书</a>

配置文件的代码

<package name="demo2" extends="struts-default" namespace="/">

<action name="addBook" class="com.qf.demo2.BookAction" method="add"></action>

<action name="deleteBook" class="com.qf.demo2.BookAction" method="delete"></action>

</package>

Action的代码

public String add(){

System.out.println("添加图书");

return NONE;

}

public String delete(){

System.out.println("删除图书");

return NONE;

}

2. 通配符的访问方式:(访问的路径和方法的名称必须要有某种联系.) 通配符就是 \* 代表任意的字符

使用通配符的方式可以简化配置文件的代码编写，而且扩展和维护比较容易。

具体实例如下：

页面代码

<a href="${pageContext.request.contextPath}/order\_add.action">添加订单</a>

<a href="${pageContext.request.contextPath}/order\_delete.action">删除订单</a>

配置文件代码

<action name="order\_\*" class="com.qf.demo2.OrderAction" method="{1}"></action>

Action的代码

public String add(){

System.out.println("添加订单");

return NONE;

}

public String delete(){

System.out.println("删除订单");

return NONE;

}

具体理解：在JSP页面发送请求，http://localhost/struts1/order\_add.action，配置文件中的order\_\*可以匹配该请求，\*就相当于变成了add，method属性的值使用{1}来代替，{1}就表示的是第一个\*号的位置！！所以method的值就等于了add，那么就找到Action类中的add方法，那么add方法就执行了！

3. 动态方法访问的方式（有的开发中也会使用这种方式）

如果想完成动态方法访问的方式，需要开启一个常量，struts.enable.DynamicMethodInvocation = false，把值设置成true。

注意：不同的Struts2框架的版本，该常量的值不一定是true或者false，需要自己来看一下。如果是false，需要自己开启。

在struts.xml中开启该常量。

<constant name="struts.enable.DynamicMethodInvocation" value="true"></constant>

具体代码如下

页面的代码

<a href="${pageContext.request.contextPath}/product!add.action">添加商品</a>

<a href="${pageContext.request.contextPath}/product!delete.action">删除商品</a>

配置文件代码

<action name="product" class="com.qf.demo2.ProductAction"></action>

<action name=*"admin!\*"* class=*"action.AdminAction"* method=*"{1}"*>

<result name=*"success"*>/success.jsp</result>

<result name=*"success1"* type=*"redirect"*>/success1.jsp</result>

<result name=*"success2"* type=*"redirect"*>/success2.jsp</result>

<result name=*"success3"* type=*"redirect"*>/success3.jsp</result>

</action>

Action的类的代码

public class ProductAction extends ActionSupport{

public String add(){

System.out.println("添加订单");

return NONE;

}

public String delete(){

System.out.println("删除订单");

return NONE;

}

}

## 技术分析之在Struts2框架中使用Servlet的API

### 1. 在Action类中也可以获取到Servlet一些常用的API

需求：提供JSP的表单页面的数据，在Action中使用Servlet的API接收到，然后保存到三个域对象中，最后再显示到JSP的页面上。

\* 提供JSP注册的页面，演示下面这三种方式

|  |
| --- |
| <h3>注册页面</h3>  <form action="${pageContext.request.contextPath }/xxx.action" method="post">  姓名:<input type="text" name="username" /><br/>  密码:<input type="password" name="password" /><br/>  <input type="submit" value="注册" />  </form> |

### 2. 完全解耦合的方式

如果使用该种方式，Struts2框架中提供了一个类，ActionContext类，该类中提供一些方法，通过方法获取Servlet的API

|  |
| --- |
| ActionContext是Map结构的容器,ActionContext是Action的上下文,类比ServletContext,存放着Action执行过程中的数据信息,比如Action的数据,request的数据,session的s数据,applicaton的数据等.**每次请求时会为当前线程创建一个新的ActionContext对象**.通过ThreadLocal来维护当前线程的request等数据,所以ActionContext是**线程安全**的. |

一些常用的方法如下

static ActionContext getContext() -获取ActionContext对象实例

java.util.Map<java.lang.String,java.lang.Object> getParameters() -- 获取请求参数，相当于request.getParameterMap();

java.util.Map<java.lang.String,java.lang.Object> getSession() -- 获取的代表session域的Map集合，就相当于操作session域。

java.util.Map<java.lang.String,java.lang.Object> getApplication() -- 获取代表application域的Map集合

void put(java.lang.String key, java.lang.Object value) -- 注意：向request域中存入值。

### 3. 使用原生Servlet的API的方式

Struts2框架提供了一个类，ServletActionContext，该类中提供了一些静态的方法

具体的方法如下

getPageContext()

getRequest()

getResponse()

getServletContext()

## 技术分析之结果页面的跳转

### 1. 结果页面存在两种方式

#### 全局结果页面

> 条件：如果<package>包中的一些action都返回success，并且返回的页面都是同一个JSP页面，这样就可以配置全局的结果页面。

> 全局结果页面针对的当前的包中的所有的Action，但是如果局部还有结果页面，会优先局部的。使用的标签是

<global-results>

<result>/demo3/suc.jsp</result>

</global-results>

#### 局部结果页面

<result>/demo3/suc.jsp</result>

### 2. 结果页面的类型

\* 结果页面使用<result>标签进行配置，包含两个属性

> name -- 逻辑视图的名称

> type -- 跳转的类型，值一些，需要掌握一些常用的类型。常见的结果类型去struts-default.xml中查找。

dispatcher -- 转发.type的默认值.Action--->JSP

redirect -- 重定向. Action--->JSP

chain -- 多个action之间跳转.从一个Action转发到另一个Action. Action---Action

redirectAction -- 多个action之间跳转.从一个Action重定向到另一个Action. Action---Action

stream -- 文件下载时候使用的

## 技术分析之Struts2框架的数据封装

### 1. 为什么要使用数据的封装呢？

作为MVC框架，必须要负责解析HTTP请求参数，并将其封装到Model对象中

封装数据为开发提供了很多方便

Struts2框架提供了很强大的数据封装的功能，不再需要使用Servlet的API完成手动封装了！！

### 2. Struts2中提供了两类数据封装的方式？

第一种方式：属性驱动

> 提供对应属性的set方法进行数据的封装。

表单的哪些属性需要封装数据，那么在对应的Action类中提供该属性的set方法即可。

表单中的数据提交，最终找到Action类中的setXxx的方法，最后赋值给全局变量。

注意0：Struts2的框架采用的拦截器完成数据的封装。

注意1：这种方式不是特别好:因为属性特别多,提供特别多的set方法,而且还需要手动将数据存入到对象中.

注意2：这种情况下，Action类就相当于一个JavaBean，就没有体现出MVC的思想，Action类又封装数据，又接收请求处理，耦合性较高。

> 在页面上，使用OGNL表达式进行数据封装。

\* 在页面中使用OGNL表达式进行数据的封装，就可以直接把属性封装到某一个JavaBean的对象中。

\* 在页面中定义一个JavaBean，并且提供set方法：例如：private User user:

\* 页面中的编写发生了变化，需要使用OGNL的方式，表单中的写法：<input type="text" name="user.username">

\* 注意：只提供一个set方法还不够，必须还需要提供user属性的get和set方法！！！

> 先调用get方法，判断一下是否有user对象的实例对象，如果没有，调用set方法把拦截器创建的对象注入进来，

### 第二种方式：模型驱动

> 使用模型驱动的方式，也可以把表单中的数据直接封装到一个JavaBean的对象中，并且表单的写法和之前的写法没有区别！

> 编写的页面不需要任何变化，正常编写name属性的值

> 模型驱动的编写步骤：

\* 手动实例化JavaBean，即：private User user = new User();

\* 必须实现ModelDriven<T>接口，实现getModel()的方法，在getModel()方法中返回user即可！！

## 技术分析之Struts2把数据封装到集合中

1. 封装复杂类型的参数（集合类型 Collection 、Map接口等）

2. 需求：页面中有可能想批量添加一些数据，那么现在就可以使用上述的技术了。把数据封装到集合中

3. 把数据封装到Collection中

因为Collection接口都会有下标值，所有页面的写法会有一些区别，注意：

> <input type="text" name="products[0].name" />

在Action中的写法，需要提供products的集合，并且提供get和set方法。

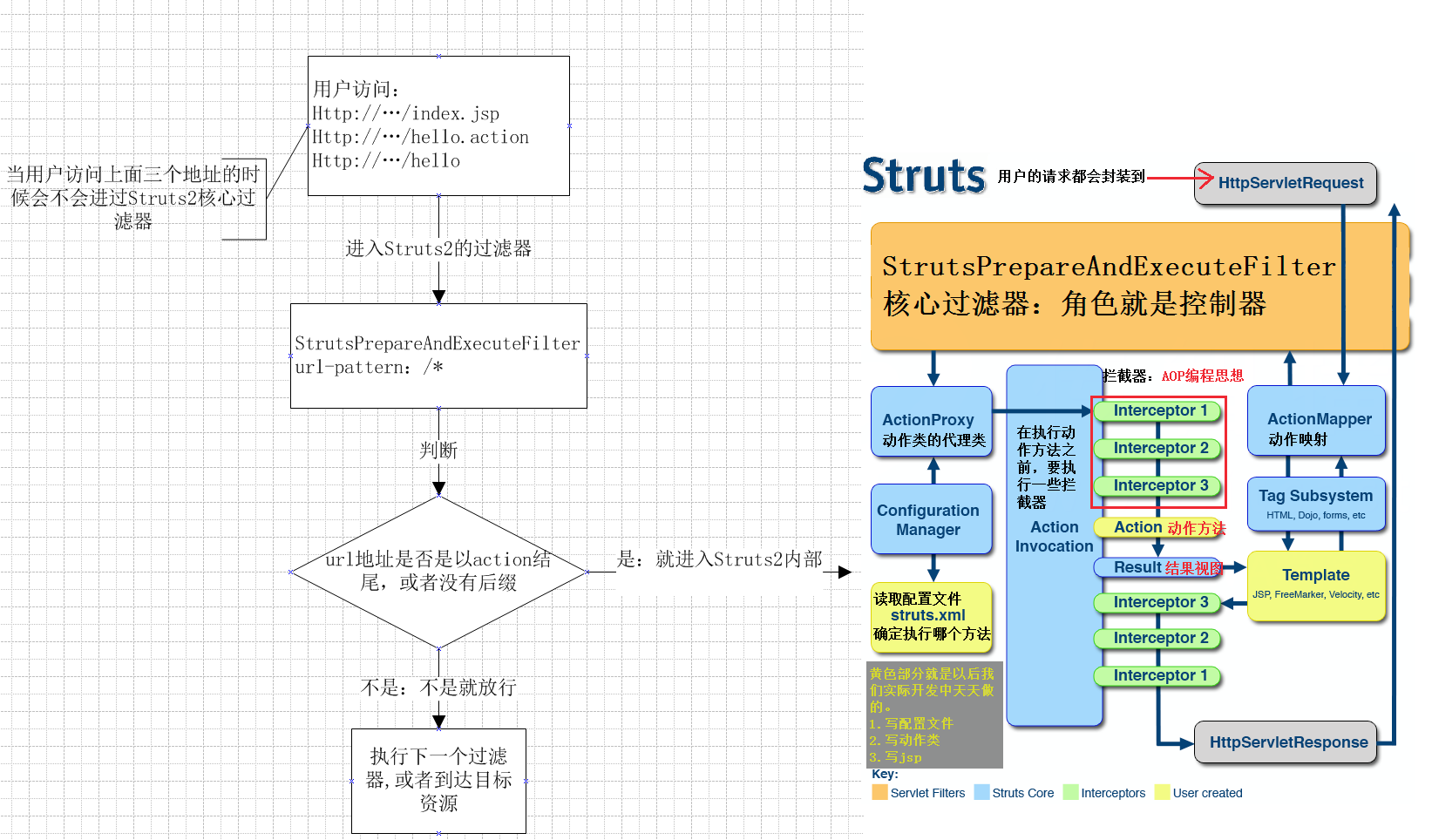
4. 把数据封装到Map中

Map集合是键值对的形式，页面的写法

> <input type="text" name="map['one'].name" />

Action中提供map集合，并且提供get和set方法

## 案例总结之Struts2的拦截器技术

****

com.opensymphony.xwork2.interceptor.Interceptor 拦截器的接口

import com.opensymphony.xwork2.interceptor.MethodFilterInterceptor，指定来不过滤哪些方法。

### 1. 拦截器的概述

拦截器就是AOP（Aspect-Oriented Programming）的一种实现。（AOP是指用于在某个方法或字段被访问之前，进行拦截然后在之前或之后加入某些操作。）

过滤器:过滤从客服端发送到服务器端请求的（可以过滤jsp,js,servlet,css,img）

拦截器:拦截对目标Action中的某些方法进行拦截

拦截器不能拦截JSP

拦截到Action中某些方法

### 2. 拦截器和过滤器的区别

1）拦截器是基于JAVA反射机制的，而过滤器是基于函数回调的

2）过滤器依赖于Servlet容器，而拦截器不依赖于Servlet容器

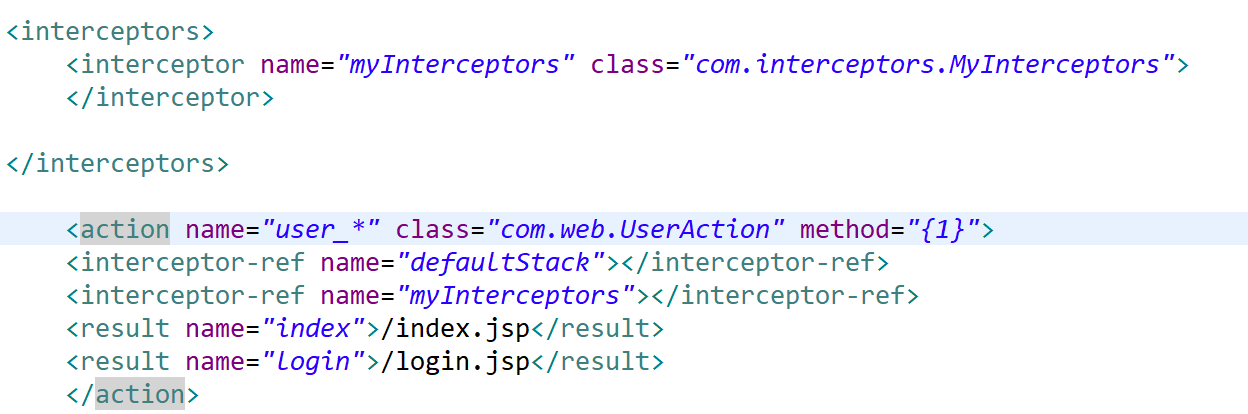
3）拦截器只能对Action请求起作用（Action中的方法），而过滤器可以对几乎所有的请求起作用（CSS JSP JS）

拦截器 采用 责任链 模式

> 在责任链模式里,很多对象由每一个对象对其下家的引用而连接起来形成一条链

> 责任链每一个节点，都可以继续调用下一个节点，也可以阻止流程继续执行

在struts2 中可以定义很多个拦截器，将多个拦截器按照特定顺序 组成拦截器栈 （顺序调用 栈中的每一个拦截器 ）



### 3. Struts2的核心是拦截器，看一下Struts2的运行流程

案例总结之自定义拦截器和配置

1. 编写拦截器，需要实现Interceptor接口，实现接口中的三个方法

protected String doIntercept(ActionInvocation invocation) throws Exception {

// 获取session对象

User user = (User) ServletActionContext.getRequest().getSession().getAttribute("existUser");

if(user == null){

// 说明，没有登录，后面就不会执行了

return "login";

}

return invocation.invoke();

}

2. 需要在struts.xml中进行拦截器的配置，配置一共有两种方式

<!-- 定义了拦截器 第一种方式

<interceptors>

<interceptor name="DemoInterceptor" class="com.qf.interceptor.DemoInterceptor"/>

</interceptors>

-->

<interceptors>

<interceptor name="DemoInterceptor" class="com.qf.interceptor.DemoInterceptor"/>

<!-- 定义拦截器栈 -->

<interceptor-stack name="myStack">

<interceptor-ref name="DemoInterceptor"/>

<interceptor-ref name="defaultStack"/>

</interceptor-stack>

</interceptors>

<action name="userAction" class="com.qf.demo3.UserAction">

<!-- 只要是引用自己的拦截器，默认栈的拦截器就不执行了，必须要手动引入默认栈

<interceptor-ref name="DemoInterceptor"/>

<interceptor-ref name="defaultStack"/>

-->

<!-- 引入拦截器栈就OK -->

<interceptor-ref name="myStack"/>

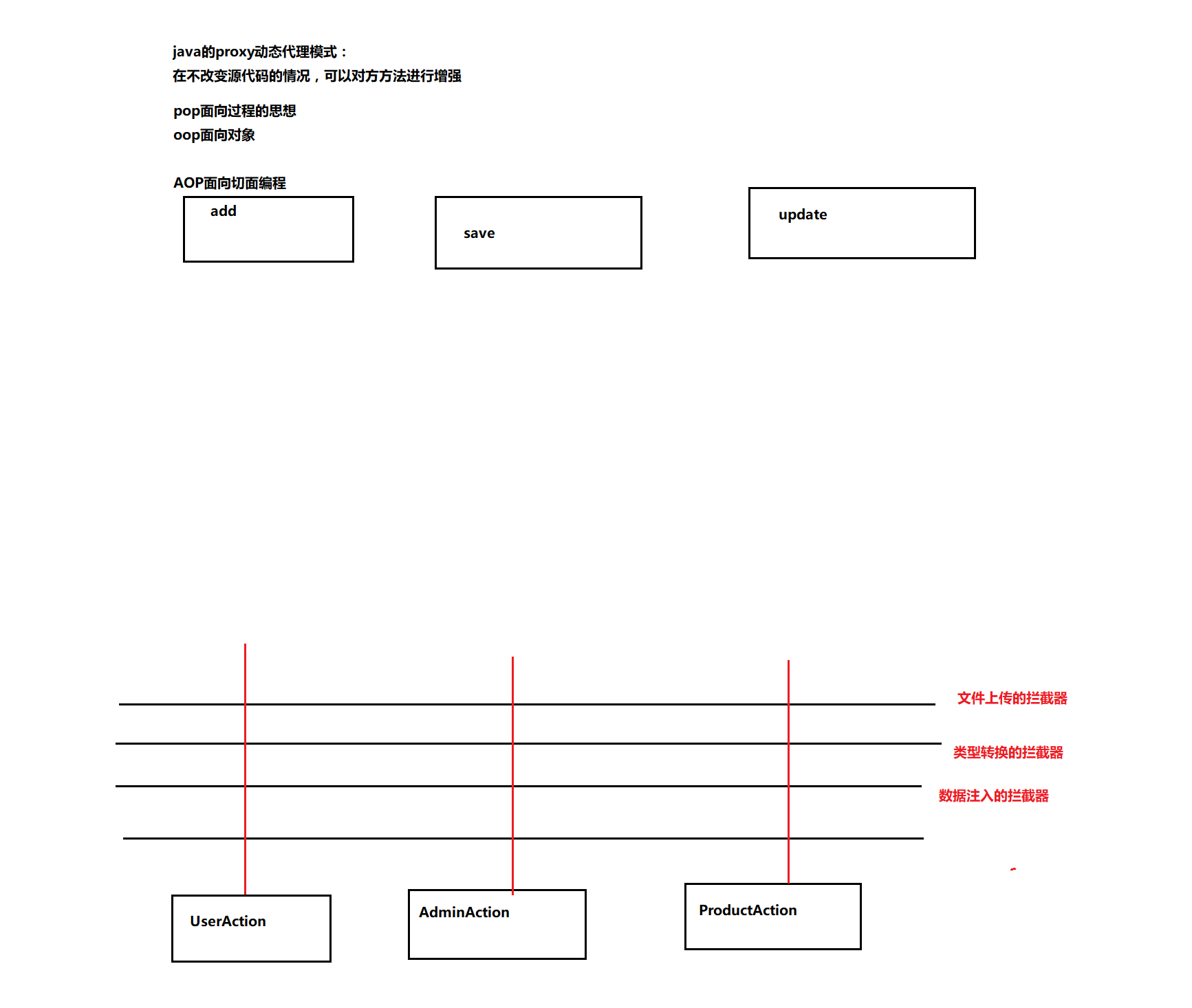
</action>

## jquery validate校验表单数据

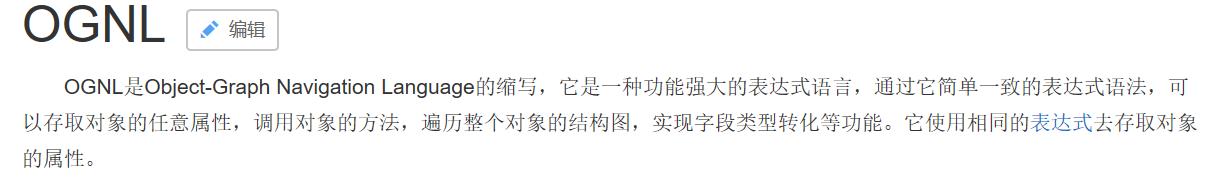
## 案例

### 注册用户

### 使用拦截器判断用户是否已经登录



## OGNL表达式概述（了解）





1. OGNL是Object Graphic Navigation Language（对象图导航语言）的缩写

所谓对象图，即以任意一个对象为根，通过OGNL可以访问与这个对象关联的其它对象

通过它简单一致的表达式语法，可以存取对象的任意属性，调用对象的方法，遍历整个对象的结构图，实现字段类型转化等功能。它使用相同的表达式去存取对象的属性

2. Struts2框架使用OGNL作为默认的表达式语言

OGNL是一种比EL强大很多倍的语言

xwork提供 OGNL表达式

ognl-3.0.5.jar

3. OGNL 提供五大类功能

支持对象方法调用

支持类静态的方法调用和值访问

访问OGNL上下文（OGNL context）和ActionContext

支持赋值操作和表达式串联

操作集合对象

4. 测试的代码

// 访问对象的方法

@Test

public void run1() throws OgnlException{

//OGNL的上下文对象

OgnlContext context = new OgnlContext();

// 获取对象的方法

Object obj = Ognl.getValue("'helloworld'.length()", context, context.getRoot());

System.out.println(obj);

}

// 获取OGNL上下文件的对象

@Test

public void run3() throws OgnlException{

OgnlContext context = new OgnlContext();

context.put("name", "haha");

// 获取对象的方法

Object obj = Ognl.getValue("#name", context, context.getRoot());

System.out.println(obj);

}

## 在Struts2框架中使用OGNL表达式

### 1. Struts2引入了OGNL表达式，主要是在JSP页面中获取值栈中的值

### 2. 具体在Struts2中怎么使用呢？如下步骤

需要先引入Struts2的标签库

<%@ taglib prefix="s" uri="/struts-tags" %>

使用Struts2提供的标签中的标签

<s:property value="OGNL表达式"/>

### 3. 在JSP页面使用OGNL表达式

访问对象方法

<s:property value="'hello'.length()"/>

## 值栈的概述

### 问题一：什么是值栈？

值栈就相当于Struts2框架的数据的中转站，向值栈存入一些数据。从值栈中获取到数据。

ValueStack 是 struts2 提供一个接口，实现类 OgnlValueStack ---- 值栈对象 （OGNL是从值栈中获取数据的 ）

Action是多例的，有一起请求，创建Action实例，创建一个ActionContext对象，代表的是Action的上下文对象，还会创建一个ValueStack对象。

每个Action实例都有一个ValueStack对象 （一个请求 对应 一个ValueStack对象 ）

在其中保存当前Action 对象和其他相关对象

Struts 框架把 ValueStack 对象保存在名为 “struts.valueStack” 的请求属性中,request中 （值栈对象 是 request一个属性）

ValueStack vs = (ValueStack)request.getAttribute("struts.valueStack");

### 问题二 ： 值栈的内部结构 ？

值栈由两部分组成

root -- Struts把动作和相关对象压入 ObjectStack 中--List

context -- Struts把各种各样的映射关系(一些 Map 类型的对象) 压入 ContextMap 中

Struts会默认把下面这些映射压入ContextMap（context）中

注意：request代表的是Map集合的key值，value的值其实也是一个Map集合。

parameters: 该 Map 中包含当前请求的请求参数 ?name=jack&password=123

request: 该 Map 中包含当前 request 对象中的所有属性

session: 该 Map 中包含当前 session 对象中的所有属性

application:该 Map 中包含当前 application 对象中的所有属性

attr: 该 Map 按如下顺序来检索某个属性: request, session, application

ValueStack中 存在root属性 (CompoundRoot) 、 context 属性 （OgnlContext ）

CompoundRoot 就是ArrayList

OgnlContext 就是 Map

context 对应Map 引入 root对象

context中还存在 request、 session、application、 attr、 parameters 对象引用

OGNL表达式访问值栈中的数据

访问root中数据时 不需要 #

访问 request、 session、application、 attr、 parameters 对象数据 必须写 #

操作值栈 默认指 操作 root 元素

### 问题三 ： 值栈对象的创建，ValueStack 和 ActionContext 是什么关系？

值栈对象是请求时创建的

ActionContext是绑定到当前的线程上，那么在每个拦截器或者Action中获取到的ActionContext是同一个。

ActionContext中存在一个Map集合，该Map集合和ValueStack的context是同一个地址。

ActionContext中可以获取到ValueStack的引用，以后再开发，使用ActionContext来获取到值栈对象

### 问题四 ： 如何获得值栈对象

获得值栈对象 有三种方法

ValueStack vs1 = (ValueStack) ServletActionContext.getRequest().getAttribute("struts.valueStack");

ValueStack vs2 = (ValueStack) ServletActionContext.getRequest().getAttribute(ServletActionContext.STRUTS\_VALUESTACK\_KEY);

ValueStack vs3 = ActionContext.getContext().getValueStack();

### 问题五： 向值栈保存数据 （主要针对root栈）

值栈里保存数据的两种方式：

1. struts2默认会吧action类的实例压到contextMap栈里， 该类的属性也都全部在里面。
2. 自己在方法里，手动获得值栈对象，自己调用下面的方法来存值

valueStack.push(Object obj);

push方法的底层调用root对象的push方法（把元素添加到0位置）

valueStack.set(String key, Object obj);

源码获取map集合（map有可能是已经存在的，有可能是新创建的），把map集合push到栈顶，再把数据存入到map集合中。

在jsp中 通过 <s:debug /> 查看值栈的内容

### 问题六： 在JSP中获取值栈的数据

总结几个小问题：

访问root中数据 不需要#

访问context其它对象数据 加 #

如果向root中存入对象的话，优先使用push方法。

如果向root中存入集合的话，优先要使用set方法。

在OgnlContext中获取数据

在Action中向域对象中存入值

request:<s:property value="#request.username"/>

session:<s:property value="#session.username"/>

application:<s:property value="#application.username"/>

attr:<s:property value="#attr.username"/>

parameters:<s:property value="#parameters.cid"/>

代码如下

<!--

// vs.push("哈哈");

// 获取到栈顶的值

<s:property value="[0].top"/>

-->

<!--

// 栈顶是map集合，通过key获取值

vs.set("msg", "小妹");

<s:property value="[0].top.msg"/>

-->

<!--

vs.push(user);

// 栈顶放user对象

<s:property value="[0].top.username"/>

<s:property value="[0].top.password"/>

// [0].top 关键字是可以省略的 findValue()

<s:property value="username"/>

-->

<!--

vs.set("user", user);

<s:property value="[0].top.user.username"/>

<s:property value="[0].top.user.password"/>

// 省略关键字

<s:property value="user.username"/>

-->

<!--

// 在ValueStack1Action提供了成员的属性

private User user = new User("小泽","456");

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

User user = new User("小苍","123");

vs.set("user", user);

// 从栈顶开始查找，找user的属性，显示名称 返回的小苍

<s:property value="user.username"/>

// [1].top获取ValueStack1Action [1].top.user返回user对象 [1].top.user.username获取对象的属性名称

<s:property value="[1].top.user.username"/>

-->

<!--

栈顶是list集合

vs.push(ulist);

<s:property value="[0].top[0].username"/>

<s:property value="[0].top[1].username"/>

-->

<!--

vs.set("ulist", ulist);

<s:property value="ulist[0].username"/>

-->

<!-- 迭代的标签

属性

\* value 要迭代的集合，需要从值栈中获取

\* var 迭代过程中，遍历的对象

\* var编写上，把迭代产生的对象默认压入到context栈中，从context栈取值，加#号

\* var不编写，默认把迭代产生的对象压入到root栈中

for(User user:ulist){}

// 编写var的属性 默认会把当前元素压到context对象

<s:iterator value="ulist" var="u">

<s:property value="#u.username"/>

<s:property value="#u.password"/>

</s:iterator>

// 没有编写var关键字 默认会吧当前对象压到root对象

<s:iterator value="ulist">

<s:property value="username"/>

<s:property value="password"/>

</s:iterator>

-->

<!-- 从context栈中获取值，加#号

HttpServletRequest request = ServletActionContext.getRequest();

request.setAttribute("msg", "美美");

request.getSession().setAttribute("msg", "小风");

<s:property value="#request.msg"/>

<s:property value="#session.msg"/>

<s:property value="#parameters.id"/>

<s:property value="#attr.msg"/>

-->

<!-- 在JSP页面上，查看值栈的内部结构 -->

<s:debug></s:debug>

EL表达式也会获取到值栈中的数据

### 问题七：为什么EL也能访问值栈中的数据？

StrutsPreparedAndExecuteFilter的doFilter代码中 request = prepare.wrapRequest(request);

对Request对象进行了包装 ，StrutsRequestWrapper

增强了request的 getAttribute

Object attribute = super.getAttribute(s);

if (attribute == null) {

attribute = stack.findValue(s);

}

访问request范围的数据时，如果数据找不到，去值栈中找

request对象 具备访问值栈数据的能力 （查找root的数据）

## 总结OGNL表达式的特殊的符号

## # 符号的用法

\* 获得contextMap中的数据

<s:property value="#request.name"/>

<s:property value="#session.name"/>

<s:property value="#application.name"/>

<s:property value="#attr.name"/>

<s:property value="#parameters.id"/>

<s:property value="#parameters.name"/>

\* 构建一个map集合

\* 例如：

<s:radio name="sex" list="{'男','女'}"></s:radio>

<s:radio name="sex" list="#{'0':'男','1':'女'}"></s:radio>

## % 符号的用法

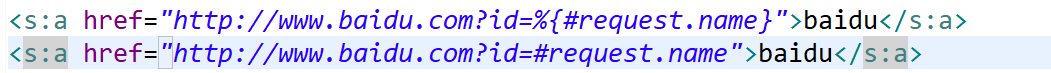
\* 强制字符串解析成OGNL表达式。

例如：在request域中存入值，然后在文本框（<s:textfield>）中取值，现在到value上。

<s:textfield value="%{#request.msg}"/>

\* { }中值用''引起来,此时不再是ognl表达式,而是普通的字符串

例如：<s:property value="%{'#request.msg'}"/>



### $ 符号的用法

\* 在配置文件中可以使用OGNL表达式，例如：文件下载的配置文件。

<action name="download1" class="cn.qf.demo2.DownloadAction">

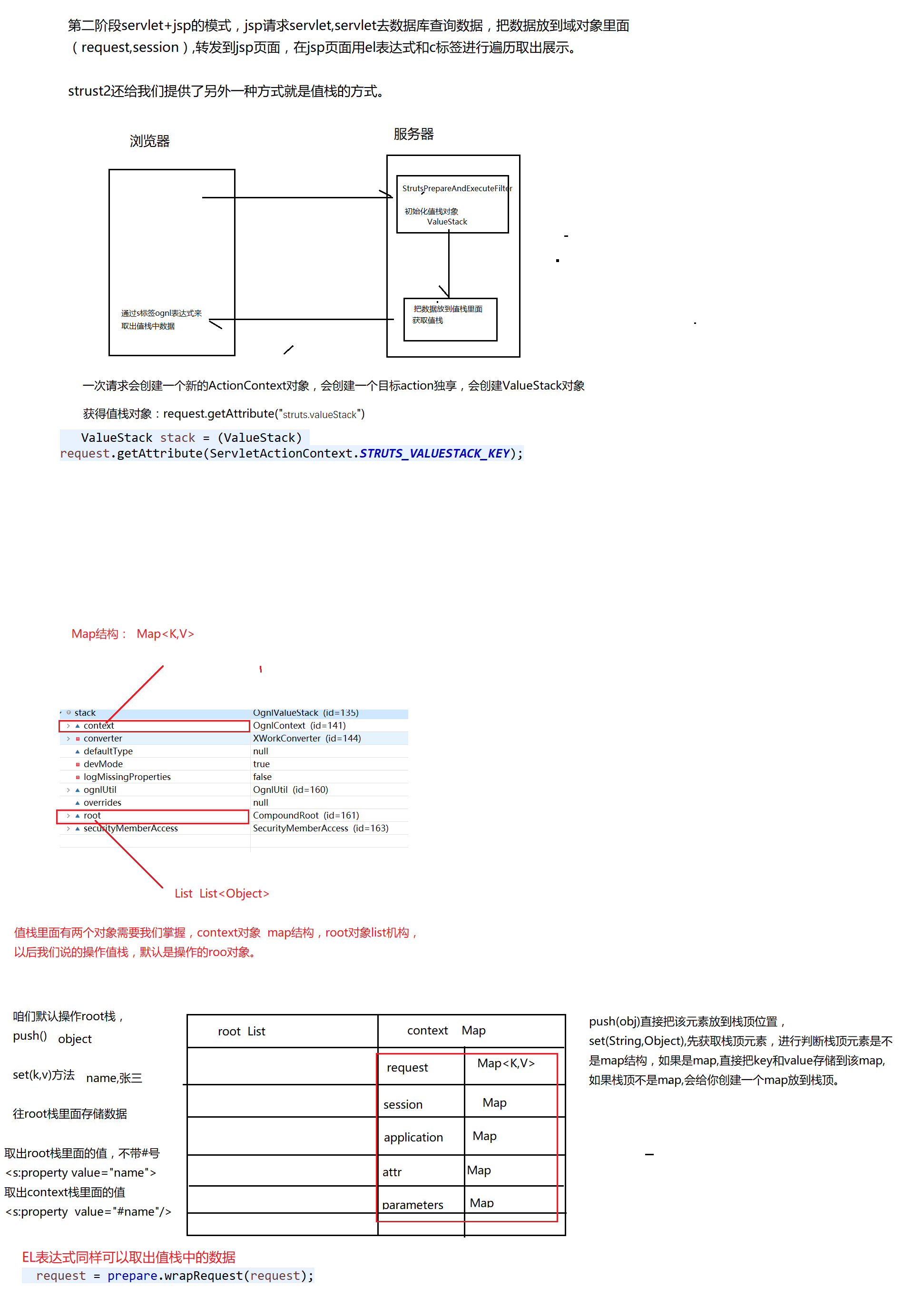
<result name="success" type="stream">

<param name="contentType">${contentType}</param>

<param name="contentDisposition">attachment;filename=${downFilename}</param>

</result>

</action>



## struts2的标签

### # 符号的用法

\* 获得contextMap中的数据

<s:property value="#request.name"/>

<s:property value="#session.name"/>

<s:property value="#application.name"/>

<s:property value="#attr.name"/>

<s:property value="#parameters.id"/>

<s:property value="#parameters.name"/>

\* 构建一个map集合

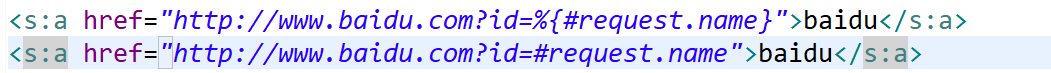
\* 例如：

<s:radio name="sex" list="{'男','女'}"></s:radio>

<s:radio name="sex" list="#{'0':'男','1':'女'}"></s:radio>

### % 符号的用法

\* 强制字符串解析成OGNL表达式。



### $ 符号的用法

\* 在配置文件中可以使用OGNL表达式，例如：文件下载的配置文件。

<action name="download1" class="cn.qf.demo2.DownloadAction">

<result name="success" type="stream">

<param name="contentType">${contentType}</param>

<param name="contentDisposition">attachment;filename=${downFilename}</param>

</result>

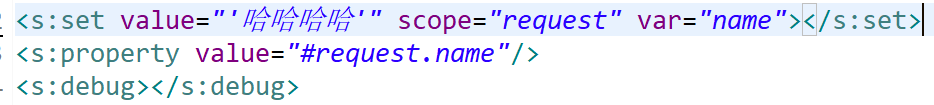
</action>

<%@ taglib uri="/struts-tags" prefix="s"%>

### 逻辑标签：

迭代： <s:iterator value="ognl表达式" var="" >

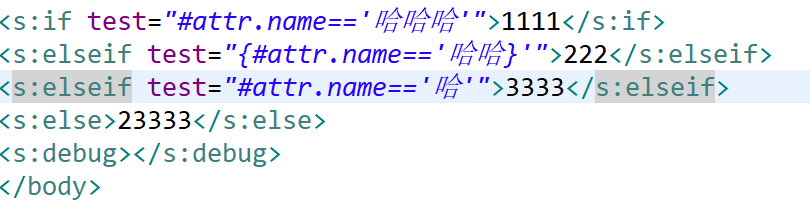
赋值： <s:set var="" value="ognl表达式"/>



获取： <s:property value="ognl表达式"/>

判断： <s:if/> <s:elseif/> <s:else/>





### UI标签：

<s:form/> <form>

<s:textfield/> <input type="text"/>

<s:password/> <input type="password"/>

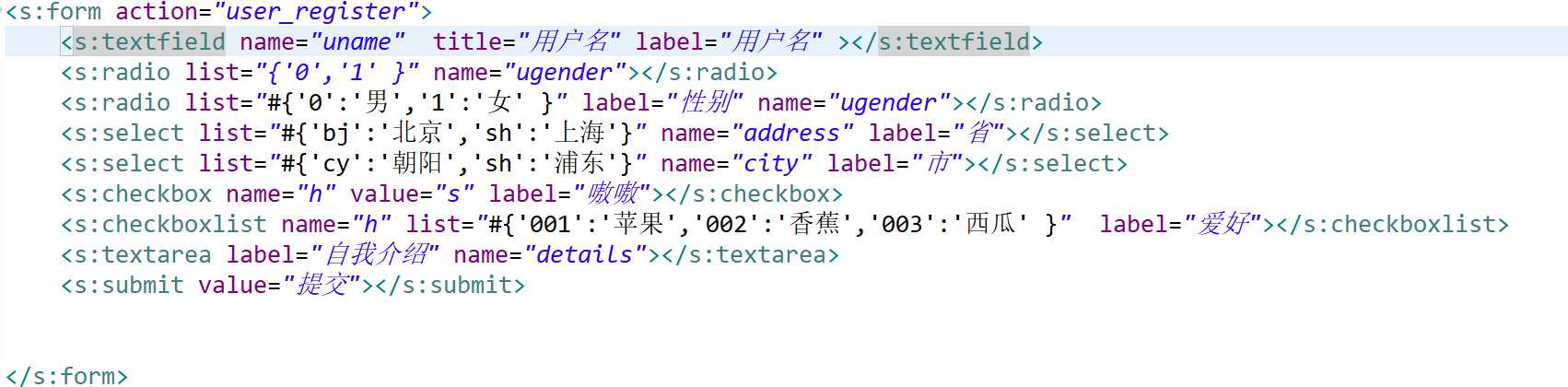
<s:checkbox/> <input type="checkbox"/>

<s:checkboxlist list="多个值"/>

多个<input type="checkbox"/>

<s:radio list="多个值"/> 多个<input type="radio"/>

<s:select list="多个值"/> <select>多个<option/></select/>



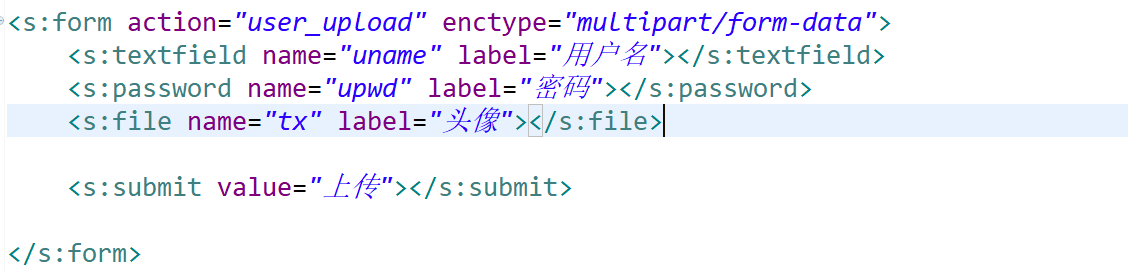
## 文件上传

表单method必须是post；

b.enctype取值必须是multipart/form-data；

c.提供文件选择域。

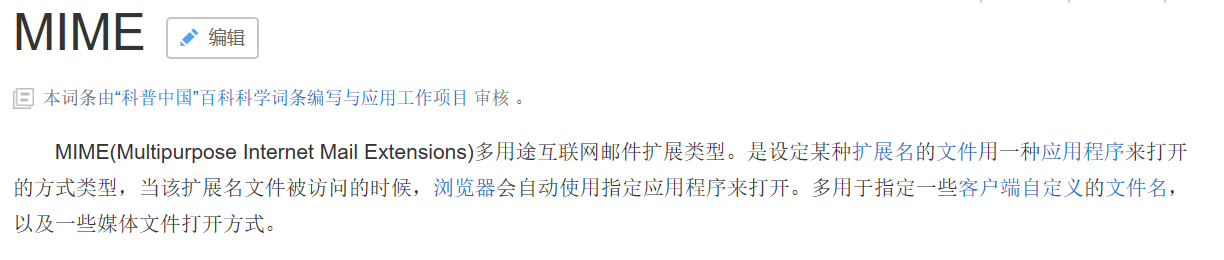
### JSP页面



### 上传Action

|  |
| --- |
| /\*\*  \* 接收上传内容  \*/  //1)接收上传文件 名字来自于表达的file的name属性名称  private File attach;  //2)接收文件类型  private String attachContentType;  //3)接收文件名称  private String attachFileName;  /\*\*  \* 一定要给setter方法.这些setter方法就是上传文件内容进行赋值  \*/  public String upload() throws Exception {  System.out.println(attach);  System.out.println(attachContentType);  System.out.println(attachFileName);  System.out.println(info);    /\*\*  \* 把文件保存到服务器硬盘  \*/  FileUtils.copyFile(attach, new File(savePath+attachFileName));    return SUCCESS;  } |

### 文件上传的类型MIME



### 文件的上传限制

|  |
| --- |
| **第一种方式：通过修改struts2的常量**  <!-- 限制上传文件的大小，。  在default.properties中配的struts.multipart.maxSize=2097152-->  <constant name="struts.multipart.maxSize" value="5242880"></constant>  <action name="upload" class="com.qf.UploadAction" method="upload">  <interceptor-ref name="defaultStack">  <!-- 限制上传文件的扩展名。如果有多个，使用逗号分隔  <param name="fileUpload.allowedExtensions">jpg,png,bmp</param>-->  <!-- 限制上传文件的MIME类型。 如果有多个，使用逗号分隔 -->  <param name="fileUpload.allowedTypes">image/jpg,image/jpeg,image/png</param>  </interceptor-ref>  <result name="input">/upload.jsp</result>  </action> |

## 多文件上传

### Jsp页面

|  |
| --- |
| <%--文件上传的必要前提：  1.请求方式必须是post  2.enctype属性的取值必须是multipart/form-data  3.提供一个文件选择域  --%>  <s:form action="upload2.action" enctype="multipart/form-data">  <s:textfield name="username" label="用户名" />  <s:file name="photo" label="照片" />  <s:file name="photo" label="照片" />  <s:submit value="上传"/>  </s:form> |

### Action代码

|  |
| --- |
| //表单上提供的字段  private String username;  private File[] photo;  //struts2在文件上传时提供的属性  private String[] photoFileName;//上传的文件名。上传字段名称+FileName 注意大小写  private String[] photoContentType;//上传文件的MIME类型。上传字段名称+ContentType 注意大小写      public String upload(){  //1.拿到ServletContext  ServletContext application = ServletActionContext.getServletContext();  //2.调用realPath方法，获取根据一个虚拟目录得到的真实目录  String filePath = application.getRealPath("/WEB-INF/files");  //3.如果这个真实的目录不存在，需要创建  File file = new File(filePath);  if(!file.exists()){  file.mkdirs();  }  //4.把photo存过去  //剪切：把临时文件剪切指定的位置，并且给他重命名。 注意：临时文件没有了  for(int i=0;i<photo.length;i++){  photo[i].renameTo(new File(file,photoFileName[i]));  }    return null;  }  public String getUsername() {  return username;  }  public void setUsername(String username) {  this.username = username;  }  public File[] getPhoto() {  return photo;  }  public void setPhoto(File[] photo) {  this.photo = photo;  }  public String[] getPhotoFileName() {  return photoFileName;  }  public void setPhotoFileName(String[] photoFileName) {  this.photoFileName = photoFileName;  }  public String[] getPhotoContentType() {  return photoContentType;  }  public void setPhotoContentType(String[] photoContentType) {  this.photoContentType = photoContentType;  } |

## 文件下载：

|  |
| --- |
| **<result name=*"success"* type=*"stream"*>**  **<param name=*"contentType"*>application/octet-stream</param>**  **<param name=*"contentDisposition"*>attachment;filename=${fileName}</param>**  **<param name=*"inputName"*>inputStream</param>**  **</result>** |

### 下载action

|  |
| --- |
| public class DownloadAction extends ActionSupport {    private InputStream inputStream;  //创建一个文件名的属性  private String filename;    public String download()throws Exception{  //1.找到文件的存储路径  String filePath = ServletActionContext.getServletContext().getRealPath("/WEB-INF/files/upload.jpg");  //2.把文件读到一个InputStream中  inputStream = new FileInputStream(filePath);    //3.返回一个成功  return SUCCESS;  //4.由一个叫做stream的结果类型为我们把剩下的事情做完。  }  public InputStream getInputStream() {  return inputStream;  }  public void setInputStream(InputStream inputStream) {  this.inputStream = inputStream;  }  public String getFilename() {  return filename;  }  public void setFilename(String filename) {  this.filename = filename;  }  } |

### 下载配置

|  |
| --- |
| <action name="download" class="com.qf.DownloadAction" method="download">  <result name="success" type="stream">  <!-- 给stream的结果类型注入参数：Content-Type -->  <param name="contentType">**application/octet-stream**</param>  <!-- 告知客户浏览器以下载的方式打开  <param name="contentDisposition">attachment;filename=photo.jpg</param> -->    <!-- 注入字节输入流 :取值要写动作类中set方法的名称，首字母改成小写-->  <param name="inputName">inputStream</param>  </result>  </action> |

# Struts2整合Json

# Echarts生成图表插件

|  |
| --- |
| jquery的插件 |
| 网络上有许多成熟的插件共使用者参考，插件就是将jquery实现的常用操作进行  封装，封装后的小框架就可以叫做插件，按照插件的语法去操作可以实现很多复杂  的功能，而这些复杂功能的背后实现我们可以不必懂。 |
|  |
| **表单校验插件** |
|  |
| 导入插件的步骤： |
| (1) 将jquery的原始文件和插件文件jquery.validate.js导入到工程中 |
| (2) 编写js代码对表单进行验证 |
|  |
| 表单验证的格式： |
| $("form表单的选择器").validate(json数据格式); //键值对 键：值（{}） |
| json数据格式： |
| {  rules:{  表单项name值:校验规则，  表单项name值:校验规则... ...  },  messages:{  表单项name值:错误提示信息，  表单项name值:错误提示信息... ...  } }   |  | | --- | | 其中：校验规则，可以是一个也可以是多个，如果是多个使用json格式 | | 常用校验规则如下： | |
|  |
|  |
| |  | | --- | | 注意：当错误提示信息不按照我们预想的位置显示时，我们可以按照如下方式进行设置  自定义错误显示标签放在我们需要显示的位置，当此表单项验证不通过时会将此  信息自动显示出来，jquery验证插件会自动帮助我们控制它的显示与隐藏  <lable for="html元素name值" class="error" style="display:none">错误信息</lable>  如果设置了错误lable标签就不必在messages中设置此表单项错误提示信息了  如果预定义的校验规则尚不能满足需求的话可以进行自定义校验规则：  自定义校验规则步骤如下：  (1) 使用$.validator.addMethod("校验规则名称",function(value,element,params)){}  (2) 在rules中通过校验规则名称使用校验规则  (3) 在messages中定义该规则对应的错误提示信息  其中： value是校验组件的value值  element是校验组件的节点对象  params是校验规则的参数 | |
|  |
|  |
|  |