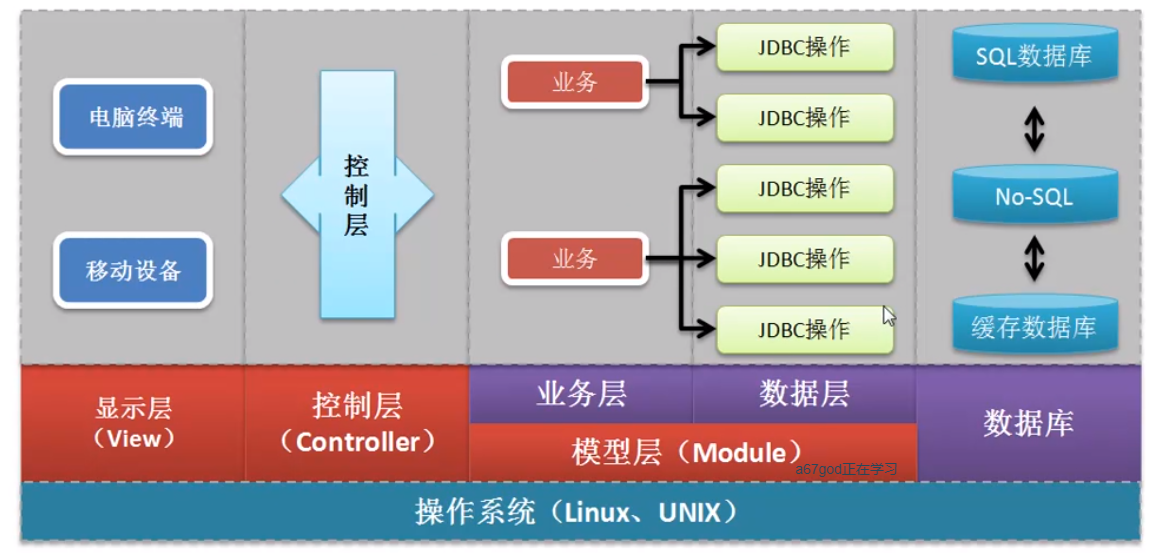
1. 传统的MVC



1. 前端页面：html、css、js等。
2. 控制层：servlet
3. 模型层：纯粹的Java程序类

使用传统的MVC，开发中最难控制的就是控制层，对于控制层的功能：

* 可以接收参数，由于整个vo结构的发展，所以控制层接收的数据往往需要转换为vo类,整个过程需要通过反射完成。

|- request.getParamaterNames()、一序列的Class、Method、Filed类的调用。

* 数据验证处理：必须要服务器端和客户端一起验证，要保证到业务层的数据安全。
* 需要自己实现资源文件的定义（读取），包括国际化的实现。
* Session处理时需要依靠cookie机制，若果客户端禁用的cookie，则控制层需要保证session的正常使用（传递一个JSESSIONID的参数，但每次传递就比较麻烦）。
* Servlet的实例化只有一次，但不同用户的处理有多次，那么久需要合理的安排vo类对象的定义。
* 很多时候的开发，我们不需要知道request和resopne.
* 文件的上传处理应该跟简单一些。

以上的问题虽然经验丰富的开发可以解决的很好，但从实质上来讲这样太耗费时间了。

1. Struts2开发框架

1、MVC和Struts的关系：

MVC就是一个设计模式，Struts是一个实现了MVC的开发框架。MVC是相当于一个标准，Struts是这个标准的具体实现。

1. 观察web.xml:

<**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**>

配置文件中使用的过滤器，之所以使用过滤器有两个原因：

* 过滤器可以像普通servlet那样处理用户请求，使用RequestDispatcher实现跳转；
* Struts 1.x用的是servlet处理请求,所以Struts2.x 为了不同filter（登录检测麻烦，只有页面登录检查可以使用过滤器，服务器端必须使用自己的拦截器）。

1. 第一个Struts2.x 程序
   1. 创建Class程序：

**public class** EchoAction **extends** ActionSupport {*//必须继承ActionSupport  
 //用户传递的参数名称* **private** String **msg**;  
  
 **public** String getMsg() {  
 *//顶替了msg=requeest.getParameter("msg")功能* **return msg**;  
 }  
  
 **public void** setMsg(String msg) {  
 *//页面读取的时候使用* **this**.**msg** = msg;  
 }  
  
 @Override  
 **public** String execute() **throws** Exception {*// 执行操作* **this**.**msg** = **"ECHO : "** + **this**.**msg** ;  
 **return "echo.page"**; *//返回路径映射的key* }  
}

* 1. 在Strus中配置路径映射：

*<?***xml version="1.0" encoding="UTF-8"***?>***<!DOCTYPE struts PUBLIC  
 "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"  
 "http://struts.apache.org/dtds/struts-2.5.dtd"*>***<**struts**>  
 *<!--所有的Struts2x 都要求设置命名空间-->* <**package name="root" namespace="/"**>  
 *<!--配置action程序的名称的及对应的路径-->* <**action name="EchoAction" class="cn.mldn.controller.EchoAction"**>  
 *<!--定义跳转路径-->* <**result name="echo.jsp"**>echo.jsp</**result**>  
 </**action**>  
 </**package**>  
</**struts**>

* 1. 创建页面：

<%@ **page language**="**java**" **import**="**java.util.\***" **pageEncoding**="**UTF-8**"%>  
<%@ **taglib prefix**="**s**" **uri**="**/struts-tags**"%>  
**<%** String path = request.getContextPath();  
 String basePath = request.getScheme() + **"://"** + request.getServerName() + **":"** + request.getServerPort()  
 + path + **"/"**;  
**%>**<**html**>  
<**head**>  
 <**title**>第一个Struts2.x 程序</**title**>  
</**head**>  
<**body**>  
 <**h1**> <**s:property value="msg"**></**s:property**></**h1**>  
 <**s:form action="EchoAction" method="POST"**>  
 <**s:textfield key="msg" label="请输入信息"**/>  
 <**s:submit value="发送"**></**s:submit**>  
 </**s:form**>  
</**body**>  
</**html**>

* 1. 异常处理：

严重: Exception starting filter struts2

java.lang.ClassNotFoundException: org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter

at org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1907)

at org.apache.catalina.loader.WebappClassLoaderBase.loadClass(WebappClassLoaderBase.java:1750)

at org.apache.catalina.core.DefaultInstanceManager.loadClass(DefaultInstanceManager.java:534)

at org.apache.catalina.core.DefaultInstanceManager.loadClassMaybePrivileged(DefaultInstanceManager.java:516)

at org.apache.catalina.core.DefaultInstanceManager.newInstance(DefaultInstanceManager.java:148)

at org.apache.catalina.core.ApplicationFilterConfig.getFilter(ApplicationFilterConfig.java:264)

at org.apache.catalina.core.ApplicationFilterConfig.<init>(ApplicationFilterConfig.java:108)

at org.apache.catalina.core.StandardContext.filterStart(StandardContext.java:4949)

at org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5651)

at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:145)

at org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase.java:1015)

at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:991)

at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:652)

at org.apache.catalina.startup.HostConfig.manageApp(HostConfig.java:1899)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:498)

at org.apache.tomcat.util.modeler.BaseModelMBean.invoke(BaseModelMBean.java:301)

at com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.invoke(DefaultMBeanServerInterceptor.java:819)

at com.sun.jmx.mbeanserver.JmxMBeanServer.invoke(JmxMBeanServer.java:801)

at org.apache.catalina.mbeans.MBeanFactory.createStandardContext(MBeanFactory.java:618)

at org.apache.catalina.mbeans.MBeanFactory.createStandardContext(MBeanFactory.java:565)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:498)

at org.apache.tomcat.util.modeler.BaseModelMBean.invoke(BaseModelMBean.java:301)

at com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.invoke(DefaultMBeanServerInterceptor.java:819)

at com.sun.jmx.mbeanserver.JmxMBeanServer.invoke(JmxMBeanServer.java:801)

at javax.management.remote.rmi.RMIConnectionImpl.doOperation(RMIConnectionImpl.java:1468)

at javax.management.remote.rmi.RMIConnectionImpl.access$300(RMIConnectionImpl.java:76)

at javax.management.remote.rmi.RMIConnectionImpl$PrivilegedOperation.run(RMIConnectionImpl.java:1309)

at javax.management.remote.rmi.RMIConnectionImpl.doPrivilegedOperation(RMIConnectionImpl.java:1401)

at javax.management.remote.rmi.RMIConnectionImpl.invoke(RMIConnectionImpl.java:829)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:498)

at sun.rmi.server.UnicastServerRef.dispatch(UnicastServerRef.java:357)

at sun.rmi.transport.Transport$1.run(Transport.java:200)

at sun.rmi.transport.Transport$1.run(Transport.java:197)

at java.security.AccessController.doPrivileged(Native Method)

at sun.rmi.transport.Transport.serviceCall(Transport.java:196)

at sun.rmi.transport.tcp.TCPTransport.handleMessages(TCPTransport.java:568)

at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run0(TCPTransport.java:826)

at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.lambda$run$0(TCPTransport.java:683)

at java.security.AccessController.doPrivileged(Native Method)

at sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run(TCPTransport.java:682)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)

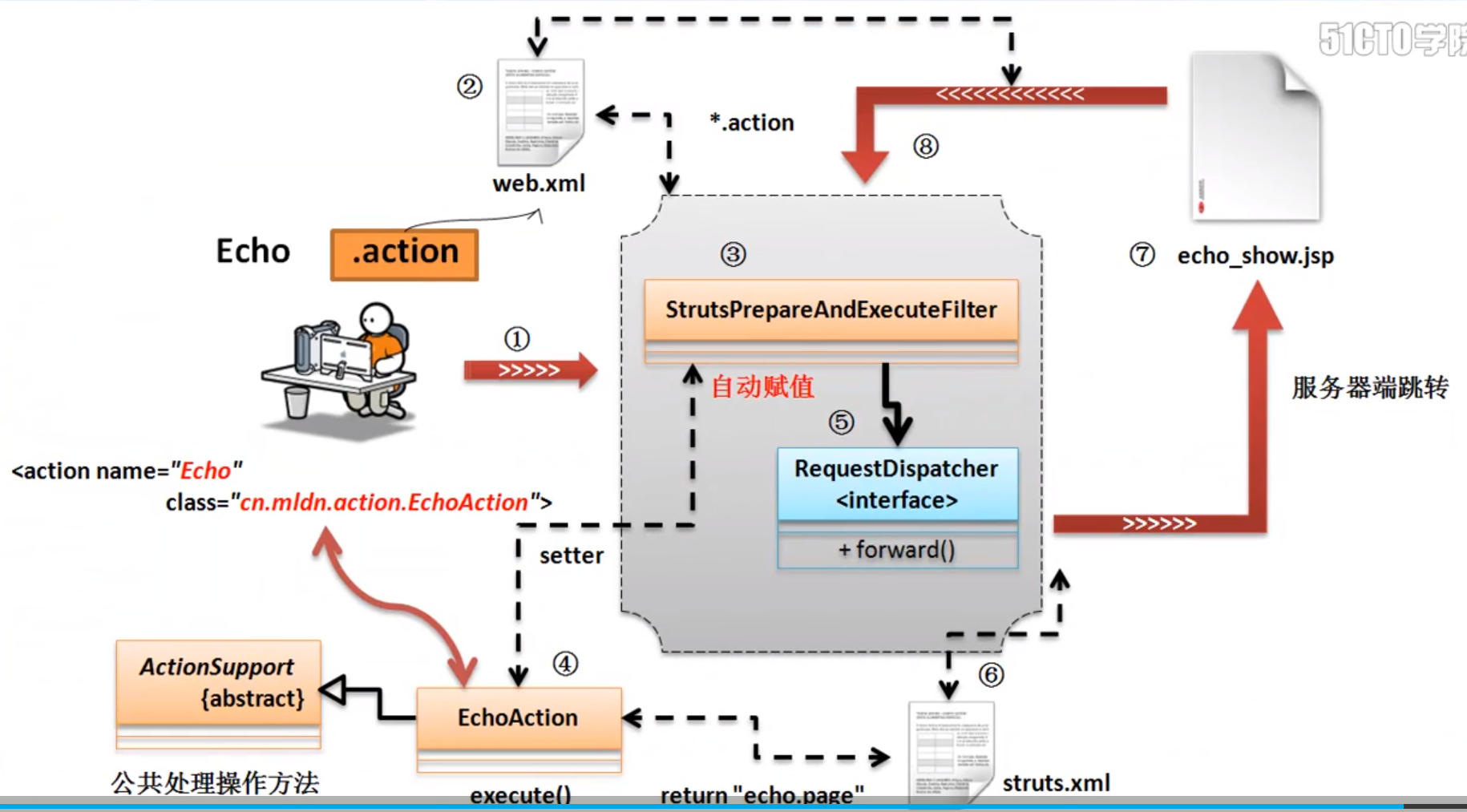
at java.lang.Thread.run(Thread.java:748)

大部分查出来的都可能是jar包问题，如果你确认jar包没问题的话，看一下你struts2 core包的版本

2.5以上的版本，请把org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter修改为org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter。

1. Strust2.x 工作原理

一旦使用了Strust2 标签，就必须由过滤器的路径跳转到jsp页面。



1. 用户首先发请求到Strust的过滤器上，输入路径“\*.action”,匹配到到web.xm配置的“<url-pattern>\*.action</url-pattern>”
2. 而后请求提交到过滤器中后，实际上并不是由过滤器来负责处理，而是过滤器会根据Strust.xml配置文件中的Action的名字，“<action name="XxxAction" class="cn.mldn.controller.XxxAction">”，根据相应的类型通过反射取得与之对应的Action处理类的名称。
3. 而后调用ActionSupport(抽象类)类中的execute()方法，但此方法是被子类所覆写，所以本质上是调用XxxAction类中的execute()方法。
4. 当XxxAction中的execute()的方法执行完毕后将返回一个字符串“return "echo.page";”
5. 根据<package>元素中namespace属性结合定位出完整路径，而后利用RequestDispatcher接口提供的forward跳转到指定的页面。
6. 跳转类型配置

默认情况下Strust2使用的是forward的服务器端跳转，，若果想要使用客户端跳转，则需要修改跳转类型，Strust2里面一共有三种类型：

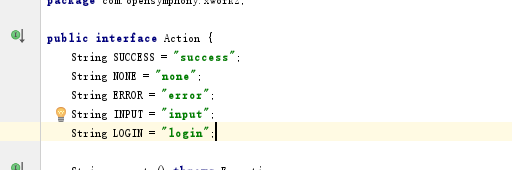
1. type="dispatcher",为默认类型表示服务器端跳转，此时可以通过标签使用XxxAction类中的属性，<result name="echo.page" type="dispatcher">/echo.jsp</result>。
2. type=”redirect”,表示客户端跳转，此时不能通过标签使用类中的属性。
3. type=”redirect-action”、”redirectAction”表示跳转到另外一个action。此时依然是服务器端跳转。
4. 过滤器配置
5. 新过滤器：
   1. org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter：在页面执行之前和页面执行之后都要执行此过滤器。
   2. org.apache.struts2.dispatcher.filter.StrutsPrepareFilter:在页面执行（页面还能生成）之前执行过滤器。
   3. org.apache.struts2.dispatcher.filter.StrutsExecuteFilter:在页面执行之后（页面已生成）执行过滤器。

以上三种过滤器均实现了javax.servlet.filter接口;

1. 老过滤器：

org.apache.struts2.dispatcher.FilterDispatcher,此过滤器是来自于webWork框架的。目前新版本的strust2已不使用。

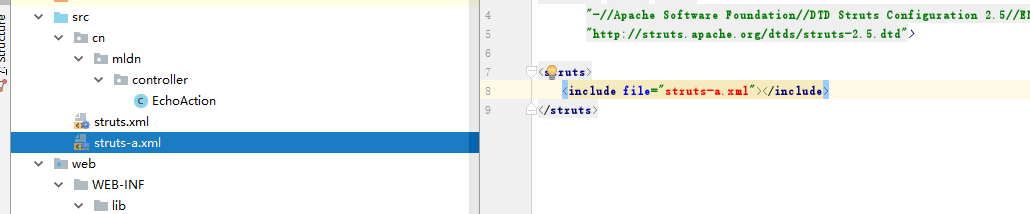
1. 内置跳转名称



1. JSP内置对象

要取得JSP内置对象，需要通过ServletActionContext，Strust2隐藏了JSP内置对象。

1. 多人开发



1. 乱码解决

在类路径下创建文件struts.properties：

struts.i18n.encoding = UTF-8

Struts.local=zh\_CN

若果没有配置，则取默认资源文件的编码：

struts2-core.jar下

