Struts2 培训教材

腾科 Java 教学部

腾科Java教学部

目录

腾科Java教学部

腾科Java教学部

腾科Java教学部

1. 关于 Struts

1.1. Struts 简介

Struts 通过采用 Java Servlet/JSP 技术，实现了基于 Java EE Web 应用的 MVC 设计模式的应用框架，是 MVC 经典设计模式中的一个经典产品。

Struts 最早是作为 Apache Jakarta 项目的组成部分，项目的创立者希望通过对该项目的研究，改进和提高 Java Server Pages 、Servlet、标签库以及面向对象的技术水准。

1.2. Struts 1

Struts 1 框架以 ActionServlet 作为核心控制器，整个应用由客户端请求驱动。当客户端向 Web 应用发送请求时，请求将被 Struts 1 的核心控制器 ActionServlet 拦截，ActionServlet 根据请求决定是否需要调用业务逻辑控制器处理用户请求（实际上，业务逻辑控制器还是控制器，它只是负责调用模型来处理用户请求），当用户请求处理完成后，其处理结果通过 JSP 呈现给用户。

1.2.1. Struts 1 的技术特点

对于整个 Struts 1 框架而言，控制器就是它的核心。Struts 1 的控制器由两个部分组成：核心控制器和业务逻辑控制器。其中核心控制器就是 ActionServlet，由 Struts 1 框架提供；业务逻辑控制就是用户自定义的 Action，由应用开发者提供。

对于大部分用户请求而言，都需要得到服务器的处理。当用户发送一个需要得到服务器处理的请求时，该请求被 ActionServlet 拦截到，ActionServlet 将该请求转发给对应的业务逻辑控制器，业务逻辑控制器调用模型来处理用户请求；如果用户请求只是希望得到某个 URL 资源，则由 ActionServlet 将被请求的资源转发给用户。

Struts 1 的程序运行流程如图 1.7 所示。

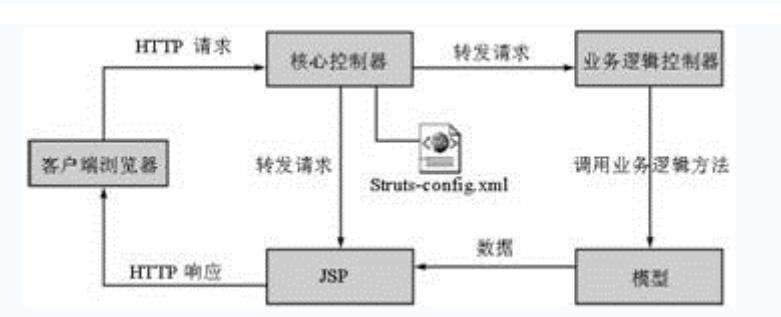


图 1.1 Struts 1 的程序运行流程

下面就 Struts 1 程序流程具体分析 MVC 中的三个角色。

（1）Model 部分

Struts 1 的 Model 部分主要由底层的业务逻辑组件充当，这些业务逻辑组件封装了底层数据库访问、业务逻辑方法实现。实际上，对于一个成熟的企业应用而言，Model 部分也不是一个简单的 JavaBean 所能完成的，它可能是一个或多个 EJB 组件，可能是一个 WebService 服务。总之，Model 部分封装了整个应用的所有业务逻辑，但整个部分并不是由 Struts 1 提供的，Struts 1 也没有为实现 Model 组件提供任何支持。

（2）View 部分

Struts 1 的 View 部分采用 JSP 实现。Struts 1 提供了丰富的标签库，通过这些标签库可以最大限度地减少脚本的使用。这些自定义的标签库可以输出控制器的处理结果。

（3）Controller 部分

Struts 1 的 Controller 由两个部分组成。

— 系统核心控制器：由 Struts 1 框架提供，就是系统中的 ActionServlet。

— 业务逻辑控制器：由 Struts 1 框架提供，就是用户自己实现的 Action 实例。

核心控制器（ActionServlet）由 Struts 1 框架提供，继承 HttpServlet 类，因此可以配置成一个标准的 Servlet，该控制器负责拦截所有 HTTP 请求，然后根据用户请求决定是否需要调用业务逻辑控制器，如果需要调用业务逻辑控制器，则将请求转发给 Action 处理，否则直接转向请求的 JSP 页面。

业务逻辑控制器负责处理用户请求，但业务逻辑控制器本身并不具有处理能

力，而是调用 Model 来完成处理。

1.2.2. Struts 1 的缺陷

对于 Struts 1 框架而言，因为它与 JSP/Servlet 耦合非常紧密，因而导致了许多不可避免的缺陷，随着 Web 应用的逐渐扩大，这些缺陷逐渐变成制约 Struts 1 发展的重要因素——这也是 Struts 2 出现的原因。下面具体分析 Struts 1 中存在的种种缺陷。

（1）支持的表现层技术单一

Struts 1 只支持 JSP 作为表现层技术，不提供与其他表现层技术，例如 Velocity、FreeMarker 等技术的整合。这一点严重制约了 Struts 1 框架的使用，对于目前的很多 Java EE 应用而言，并不一定使用 JSP 作为表现层技术。

虽然 Struts 1 处理完用户请求后，并没有直接转到特定的视图资源，而是返回一个 ActionForward 对象（可以理解 ActionForward 是一个逻辑视图名），在 struts-config.xml 文件中定义了逻辑视图名和视图资源之间的对应关系，当 ActionServlet 得到处理器返回的 ActionForword 对象后，可以根据逻辑视图名和视图资源之间的对应关系，将视图资源呈现给用户。

> 提示 Struts 1 已经通过配置文件管理逻辑视图名和实际视图之间的对应关系，只是没有做到让逻辑视图名可以支持更多的视图技术。

虽然 Struts 1 有非常优秀的设计，但由于历史原因，它没有提供与更多视图技术的整合，这严重限制了 Struts 1 的使用。

（2）与 Servlet API 严重耦合，难于测试

因为 Struts 1 框架是在 Model 2 的基础上发展起来的，因此它完全是基于 Servlet API 的，所以在 Struts 1 的业务逻辑控制器内，充满了大量的 Servlet API。

看下面的 Action 代码片段：

```
// 业务逻辑控制器必须继承Struts 1提供的Action类
public class LoginAction extends Action {
    // 处理用户请求的execute方法
    public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response)throws
    AuctionException{
    //获取封装用户请求参数的ActionForm对象
    //将其强制类型转换为登录用的ActionForm
    LoginForm loginForm = (LoginForm)form;
```

```
    //当用户名为scott,密码为tiger时返回成功
    if ("scott".equals(loginForm.getUsername()
    && "tiger".equals(loginForm.getPassword())){
    //处理成功，返回一个ActionForward对象
    return mapping.findForward("success");
    }else{
    //处理失败，返回一个ActionForward对象
    return mapping.findForward("success");
    }
    }
}
```

当我们需要测试上面 Action 类的 execute 方法时，该方法有 4 个参数：ActionMapping、ActionForm、HttpServletRequest 和 HttpServletResponse，初始化这 4 个参数比较困难，尤其是 HttpServletRequest 和 HttpServletResponse 两个参数，通常由 Web 容器负责实例化。

因为 HttpServletRequest 和 HttpServletResponse 两个参数是 Servlet API，严重依赖于 Web 服务器。因此，一旦脱离了 Web 服务器，Action 的测试非常困难。

（3）代码严重依赖于 Struts 1 API，属于侵入式设计

正如从上面代码片段中所看到的，Struts 1 的 Action 类必须继承 Struts 1 的 Action 基类，实现处理方法时，又包含了大量 Struts 1 API：如 ActionMapping、ActionForm 和 ActionForward 类。这种侵入式设计的最大弱点在于，一旦系统需要重构时，这些 Action 类将完全没有利用价值，成为一堆废品。

可见，Struts 1 的 Action 类这种侵入式设计导致了较低的代码复用。

1.3. WebWork 简介

WebWork 虽然没有 Struts 1 那样赫赫有名，但也是出身名门，WebWork 来自另外一个优秀的开源组织：opensymphony，这个优秀的开源组织同样开发了大量优秀的开源项目，如 Qutarz、OSWorkFlow 等。实际上，WebWork 的创始人则是另一个 Java 领域的名人：Rickard Oberg（他就是 JBoss 和 XDoclet 的作者）。

相对于 Struts 1 存在的那些先天性不足而言，WebWork 则更加优秀，它采用了一种更加松耦合的设计，让系统的 Action 不再与 Servlet API 耦合。使单元测试更加方便，允许系统从 B/S 结构向 C/S 结构转换。

相对于 Struts 1 仅支持 JSP 表现层技术的缺陷而言，WebWork 支持更多的表

现层技术，如 Velocity、FreeMarker 和 XSLT 等。

WebWork 可以脱离 Web 应用使用，这一点似乎并没有太多优势，因为，一个应用通常开始已经确定在怎样的环境下使用。WebWork 有自己的控制反转（Inversion of Control）容器，通过控制反转，可以让测试变得更简单，测试中设置实现服务接口的 Mock 对象完成测试，而不需要设置服务注册。

WebWork 2 使用 OGNL 这个强大的表达式语言，可以访问值栈。OGNL 对集合和索引属性的支持非常强大。

WebWork 建立在 XWork 之上，使用 ServletDispatcher 作为该框架的核心控制器，处理 HTTP 的响应和请求。

从处理流程上来看，WebWork 与 Struts 1 非常类似，它们的核心都由控制器组成，其中控制器都由两个部分组成：

- 核心控制器 ServletDispatcher，该控制器框架提供。
- 业务逻辑控制器 Action，该控制器由程序员提供。

相对 Struts 1 的 Action 与 Servlet API 紧紧耦合的弱点来说，WebWork 的 Action 则完全与 Servlet API 分离，因而该 Action 更容易测试。

WebWork 的 Action 可以与 Servlet API 分离，得益于它灵巧的设计，它使用一个拦截器链，负责将用户请求数据转发到 Action，并负责将 Action 的处理结果转换成对用户的响应。

当用户向 Web 应用发送请求时，该请求经过 ActionContextCleanUp、SiteMesh 等过滤器过滤，由 WebWork 的核心控制器拦截，如果用户请求需要 WebWork 的业务逻辑控制器处理，该控制器则调用 Action 映射器，该映射器将用户请求转发到对应的业务逻辑控制器。值得注意的是，此时的业务逻辑控制器并不是开发者实现的控制器，而是 WebWork 创建的控制器代理。

创建控制器代理时，WebWork 需要得到开发者定义的 xwork.xml 配置文件，控制器代理以用户实现的控制器作为目标，以拦截器链中的拦截器作为处理（Advice）。

> 提示 WebWork 中创建控制器代理的方式，就是一种 AOP（面向切面编程）编程方式，只是这种 AOP 中的拦截器由系统提供，因此无需用户参与。如果读者需要获取更多关于 AOP 编程的知识，请参阅 AOP 相关资料。

开发者自己实现的业务逻辑控制器只是 WebWork 业务控制器的目标——这

就是为什么开发者自己实现的 Action 可以与 Servlet API 分离的原因。当开发者自己的 Action 处理完 HTTP 请求后，该结果只是一个普通字符串，该字符串将对应到指定的视图资源。

指定的试图资源经过拦截器链的处理后，生成对客户端的响应输出。

上面整个过程的数据流图如图 1.2 所示。

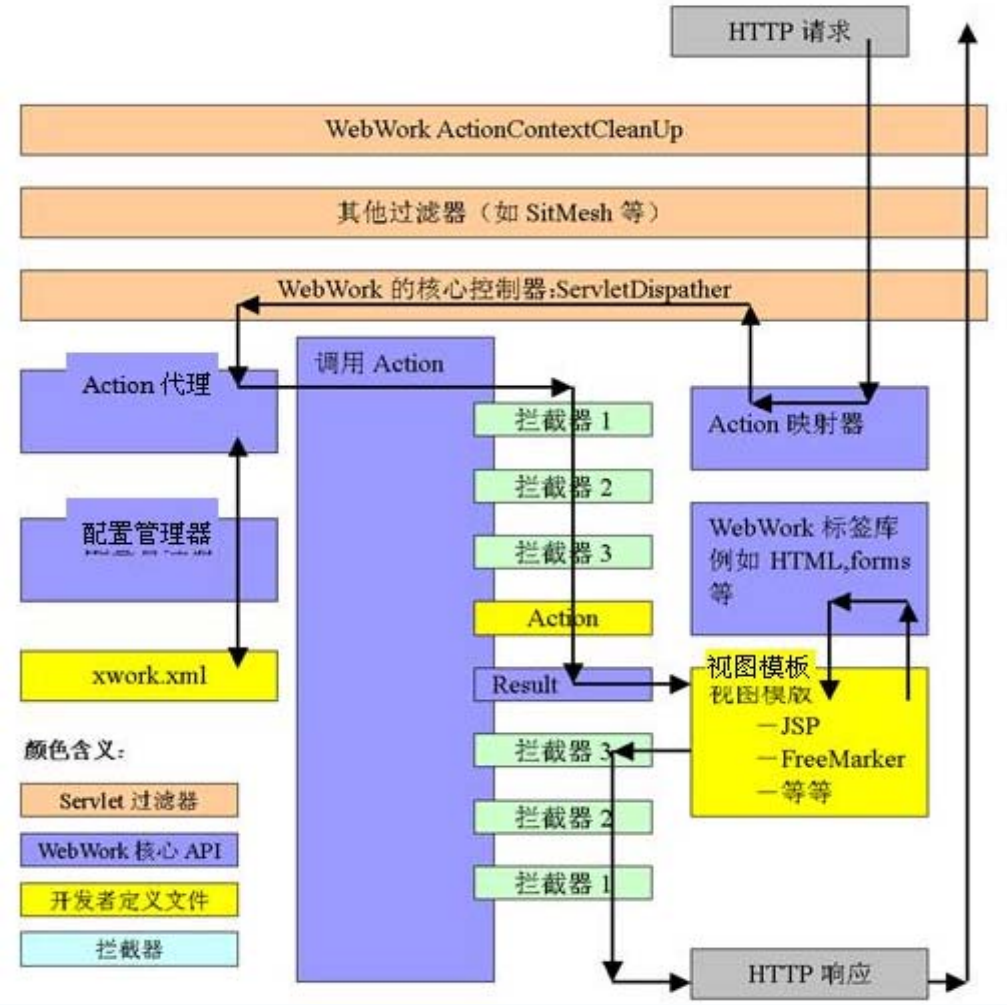


图 1.2 WebWork 的数据流图

与前面的 Struts 1 框架对比，不难发现 WebWork 在很多地方确实更优秀。相对 Struts 1 的种种缺点而言，WebWork 存在如下优点：

（1）Action 无需与 Servlet API 耦合，更容易测试

相对于 Struts 1 框架中的 Action 出现了大量 Servlet API 而言，WebWork 的 Action 更像一个普通 Java 对象，该控制器代码中没有耦合任何 Servlet API。看下面的 WebWork 的 Action 示例：

```
public class LoginAction implements Action {
```

```
    // 该字符串常量将作为Action的返回值
    private final static String LOGINFAIL = "loginfail";
    // 该Action封装的两个请求参数
    private String password;
    private String username;
    // password请求参数对应的getter方法
    public String getPassword() {
        return password;
    }
    // password请求参数对应的setter方法
    public void setPassword(String password) {
        this.password = password;
    }
    // username请求参数对应的getter方法
    public String getUsername() {
        return username;
    }
    // username请求参数对应的setter方法
    public void setUsername(String username) {
        this.username = username;
    }
    // 处理用户请求的execute方法
    public String execute() throws Exception {
        if ("yeeku".equalsIgnoreCase(getUsername())
                && "password".equals(getPassword())) {
            ActionContext ctx = ActionContext.getContext();
            // 将当前登录的用户名保存到Session
            Map session = ctx.getSession();
            session.put("username", getUsername());
            return SUCCESS;
        } else {
            return LOGINFAIL;
        }
    }
}
```

在上面的 Action 代码中，我们看不到任何的 Servlet API，当系统需要处理两个请求参数：username 和 password 时，Action 并未通过 HttpServletRequest 对象来获得请求参数，而是直接调用访问该 Action 的 username 和 password 成员属性——这两个属性由 Action 拦截器负责初始化，以用户请求参数为其赋值。

即使 Action 中需要访问 HTTP Session 对象，依然没有在代码中直接出现 HttpSession API，而是以一个 Map 对象代表了 HTTP Session 对象。

当我们将 WebWork 的 Action 和 Struts 1 的 Action 进行对比时，不难发现 Struts 1 的 Action 确实太臃肿了，确实不如 WebWork 的 Action 那么优雅。

如果需要测试上面的 Action 代码，测试用例的书写将非常容易，因为 execute 方法中没有包含任何 Servlet API，甚至没有 WebWork 的 API。

（2）Action 无需与 WebWork 耦合，代码重用率高

在上面的 Action 代码中，不难发现 WebWork 中的 Action 其实就是一个 POJO，该 Action 仅仅实现了 WebWork 的 Action 接口，包含了一个 execute 方法。

Struts 1 中的 Action 类需要继承 Struts 1 的 Action 类。我们知道，实现一个接口和继承一个类是完全不同的概念：实现一个接口对类的污染要小得多，该类也可以实现其他任意接口，还可以继承一个父类；但一旦已经继承一个父类，则意味着该类不能再继承其他父类。

除此之外，Struts 1 中 Action 也包含了一个 execute 方法，但该方法需要 4 个参数，类型分别是 ActionMapping、ActionForm、HttpServletRequest 和 HttpServletResponse，一个包含了这 4 个参数的方法，除了在 Struts 1 框架下有用外，笔者难以想象出该代码还有任何复用价值。但 WebWork 的 execute 方法则完全不同，该方法中没有出现任何 Servlet API，也没有出现任何 WebWork API，这个方法在任何环境下都有重用的价值。

得益于 WebWork 灵巧的设计，WebWork 中的 Action 无需与任何 Servlet API、WebWork API 耦合，从而具有更好的代码重用率。

（3）支持更多的表现层技术，有更好的适应性

正如从图 1.8 所见到的，WebWork 对多种表现层技术：JSP、Velocity 和 FreeMarker 等都有很好的支持，从而给开发更多的选择，提供了更好的适应性。

1.4. Struts 2

Struts 2 的体系与 Struts 1 体系的差别非常大，因为 Struts 2 使用了 WebWork 的设计核心，而不是使用 Struts 1 的设计核心。Struts 2 大量使用拦截器来处理用户请求，从而允许用户的业务逻辑控制器与 Servlet API 分离。

从数据流图上来看，Struts 2 与 WebWork 相差不大，Struts 2 同样使用拦截器作为处理（Advice），以用户的业务逻辑控制器为目标，创建一个控制器代理。

控制器代理负责处理用户请求，处理用户请求时回调业务控制器的 execute

方法，该方法的返回值将决定了 Struts 2 将怎样的视图资源呈现给用户。

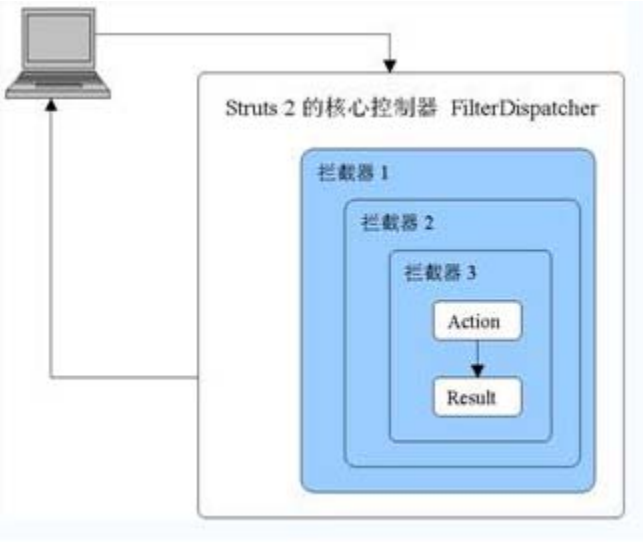


图 1.3 Struts 2 的体系概图

(1) Struts 2 框架的大致处理流程如下：

- 浏览器发送请求，例如请求/mypage.action、/reports/myreport.pdf 等。
- 核心控制器 FilterDispatcher 根据请求决定调用合适的 Action。
- WebWork 的拦截器链自动对请求应用通用功能，例如 workflow、validation 或文件上传等功能。
- 回调 Action 的 execute 方法，该 execute 方法先获取用户请求参数，然后执行某种数据库操作，既可以是将数据保存到数据库，也可以从数据库中检索信息。实际上，因为 Action 只是一个控制器，它会调用业务逻辑组件来处理用户的请求。
- Action 的 execute 方法处理结果信息将被输出到浏览器中，可以是 HTML 页面、图像，也可以是 PDF 文档或者其他文档。此时支持的视图技术非常多，既支持 JSP，也支持 Velocity、FreeMarker 等模板技术。

腾讯Java教学部

2. Struts2-快速入门

2.1. Struts2-环境搭建

建立 Web 项目

建立 Struts2 的配置文件(struts.xml)；

在 WEB-INF 下的 classes 目录下（当前 classpath 中）添加一个 struts.xml 配置文件。

注意：Struts2 的默认配置文件名为：struts.xml，该文件应该放在 Web 应用的类加载路径下，通常就是放在 WEB-INF/classes 路径下

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!--
     struts.devMode ： 是否设置为开发模式 true:是开发模式，否则不是
     注：在开发模式下，修改Struts的配置文件后不需要重新启动Tomcat服务器即生效。否则修改Struts配置文件后需要重新启动Tomcat服务器才生效。
     -->
    <constant name="struts.devMode" value="true" />
    <!-- 中文处理 -->
    <constant name="struts.i18n.encoding" value="GBK" />
    <package name="user" namespace="/user" extends="struts-default">
        <global-results>
            <result name="Welcome">Welcome.jsp</result>
        </global-results>
        <action name="user"  class="controllers_models.User"/>
    </package>
</struts>
```

复制 Struts2 相应的 jar 包及第三方包到项目的 WebRoot/WEB-INF/lib 目录下

- 注意：关于包处理：

将 Struts2 的包（xwork-2.0.4.jar、commons-logging-1.0.4.jar、freemarker-2.3.8.jar、ognl-2.6.11.jar、struts2-core-2.0.11.jar）添加道 lib 目录下。

不要盲目添加其他包，否则 Tomcat 可能启不起来。

因为：“If any Struts 2 Plugins are included, then other JARs may be needed too. For example, the optional Spring Plugin requires the Spring JARs to be present.

也就是说，如果添加了 Spring 插件，就要把 Spring 的相关包都添加进来。

另外，关于 JAR 包的版本号不必纠结于哪一个版本，一般下载最新版的即可，前面的命

名几乎一样。

在 web.xml 中添加 Struts2 的过滤器：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<filter>
<!-- 定义核心Filter的名字 -->
    <filter-name>Struts2</filter-name>
<!-- 定义核心Filter的实现类 -->
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</fi
lter-class>
</filter>
<!-- FilterDispatcher用来初始化Struts 2并且处理所有的Web请求 -->
    <filter-mapping>
        <filter-name>Struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
  <welcome-file-list>
    <welcome-file>user/index.jsp</welcome-file>
  </welcome-file-list>
    </web-app>
```

2.2. 输入页面

*在 user 目录下*建立 login.jsp:

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head></head>
  <body>
    <s:form action="user!login">
    <s:textfield name="userName" label="用户名"/>
    <s:password name="userPassword" label="密码"/>
    <s:submit value="增加用户"/>
    </s:form>
  </body>
</html>
```

2.3. 输出页面

*在 user 目录下*建立 Welcome.jsp:

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head></head>
  <body>
    <s:property value="result"/>
  </body>
</html>
```

2.4. 模型&控制器类

```
package controllers_models;
public class User {
    private String userName;
    private String userPassword;
    // 保存返回结果
    private Object result;
    public String login(){
        System.out.println( this.userName + " - " + this.userPassword );
        this.result = "Hello " + this.userName;
        return "Welcome";
    }
    public String getUserName() {
        return userName;
    }
    public String getUserPassword() {
        return userPassword;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public void setUserPassword(String userPassword) {
        this.userPassword = userPassword;
    }
    public Object getResult() {
        return result;
    }
    public void setResult(Object result) {
        this.result = result;
    }
}
```

2.5. 关于配置的说明

(1) Namespace

Namespace决定了action的访问路径,

如果Namespace定义为""或者不写（默认为""(空)），表示可以接受任何路径的action。比如下面这些URL都可以访问.

```
"http://localhost:port/项目名/index.action",
"http://localhost:port/项目名/aaa/index.action",
"http://localhost:port/项目名/aaa/bbb/ccc/index.action"
```

如果Namespace定义为"/",那么当你在访问hello.action的时候,只能访问"http://localhost:port/项目名/action名称" (Struts2中.action可以省略掉).

Namespace还可以/xxx,或者/xxx/yyy，对应的action访问路径为/index.action、/xxx/index.action、或者/xxx/yyy/index.action。

注意：请求过来先去相应的Namespace里找找看有没有匹配的action,如果没有再到Namespace为空的package里找找看有没有,如果还没有就报错了。

(2) struts-default

struts-default.xml 中配置了struts2中核心的bean和拦截器，包括工厂bean,类型检测bean,文件上传bean,标签库bean,试图bean,类型转换bean,静态注入的bean，Struts2默认包，结果类型的种类,另外还有各种拦截器：

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

(3) <package>标签

<package>是用来解决重名的问题，例如当系统的前台和后台都有一个action名叫hello，这时就需要用package来区分。

前台：<package name="front">；后台：<package name="back">

struts2中的package与java的package是相同的作用的。

(4) 注意：

控制器的配置一定要放在某个 package 的 namespace 下，并且输入页面也要放在和 namespace 一样的目录下，这样，页面和控制器的 url 相对路径就在同一路径下了，才能相互访问。

2.6. struts2 配置文件 struts.xml 和 struts.properties 详解

2.6.1. struts.xml 详解

- action 的详细讲解

package 的可以使用的属性：

属性	是否必须	说明

name	是	package 的表示，为了让其他的 package 引用
extends	否	从哪个 package 继承行为
namespace	否	参考 namespace 配置说明
abstract	否	定义这个 package 为抽象的，这个 package 中不需要定义 action

由于 struts.xml 文件是自上而下解析的，所以被继承的 package 要放在继承 package 的前边。namespace 将 action 分成逻辑上的不同模块，每一个模块有自己独立的前缀。使用 namespace 可以有效的避免 action 重名的冲突，可以根据 namespace 的不同向服务器提交不同的 package 的 action 的请求。

“/”表示根 namespace，所有直接在应用程序上下文环境下的请求（Context）都在这个 package 中查找。

“”表示默认 namespace，当所有的 namespace 中都找不到的时候就在这个 namespace 中寻找。

例如，有如下配置：

```
<package name="default">
    <action name="foo" class="mypackage.simpleAction>
            <result name="success" type="dispatcher">
                greeting.jsp
            </result>
    </action>
    <action name="bar" class="mypackage.simpleAction">
        <result name="success" type="dispatcher">bar1.jsp</result>
    </action>
</package>
<package name="mypackage1" namespace="/">
    <action name="moo" class="mypackage.simpleAction">
        <result name="success" type="dispatcher">moo.jsp</result>
    </action>
</package>
<package name="mypackage2" namespace="/barspace">
    <action name="bar" class="mypackage.simpleAction">
        <result name="success" type="dispatcher">bar2.jsp</result>
    </action>
</package>
```

1) 如果请求为/barspace/bar.action

查找 namespace：/barspace，如果找到 bar 则执行对应的 action，否则将会查找默认的 namespace，在上面的例子中，在 barspace 中存在名字为 bar 的 action，所以这个 action 将会被执行，如果返回结果为 success，则画面将定为到 bar2.jsp

2) 如果请求为/moo.action

根 namespace（'/'）被查找，如果 moo action 存在则执行，否则查询默认的 namespace，上面的例子中，根 namespace 中存在 moo action，所以该 action 被调用，返回 success 的情况下画面将定位到 moo.jsp

2.6.2. struts.properties 配置

- struts.configuration

该属性指定加载 Struts 2 配置文件的配置文件管理器。该属性的默认值是 org.apache.Struts2.config.DefaultConfiguration，这是 Struts 2 默认的配置文件管理器。如果需要实现自己的配置管理器，开发者则可以实现一个实现 Configuration 接口的类，该类可以自己加载 Struts 2 配置文件。

- struts.locale

  指定 Web 应用的默认 Locale。

- struts.i18n.encoding

  指定 Web 应用的默认编码集。该属性对于处理中文请求参数非常有用，对于获取中文请求参数值，应该将该属性值设置为 GBK 或者 GB2312。

提示：当设置该参数为 GBK 时，相当调用 HttpServletRequest 的 setCharacterEncoding 方法。

- struts.objectFactory

  指定 Struts 2 默认的 ObjectFactory Bean，该属性默认值是 spring。

- struts.objectFactory.spring.autoWrite

  指定 Spring 框架的自动装配模式，该属性的默认值是 name，即默认根据 Bean 的 name 属性自动装配。

- struts.objectFactory.spring.useClassCache

  该属性指定整合 Spring 框架时，是否缓存 Bean 实例，该属性只允许使用 true 和 false 两个属性值，它的默认值是 true。通常不建议修改该属性值。

- struts.objectTypeDeterminer

  该属性指定 Struts 2 的类型检测机制，通常支持 tiger 和 notiger 两个属性值。

- struts.multipart.parser

  该属性指定处理 multipart/form-data 的 MIME 类型（文件上传）请求的框架，该属性支持 cos、pell 和 jakarta 等属性值，即分别对应使用 cos 的文件上传框架、pell 上传及 common-fileupload 文件上传框架。该属性的默认值为 jakarta。

注意：如果需要使用 cos 或者 pell 的文件上传方式，则应该将对应的 JAR 文件复制到 Web 应用中。例如，使用 cos 上传方式，则需要自己下载 cos 框架的 JAR 文件，并将该文件放在 WEB-INF/lib 路径下。

- struts.multipart.saveDir

  该属性指定上传文件的临时保存路径，该属性的默认值是 javax.servlet.context.tempdir。

- struts.multipart.maxSize

该属性指定 Struts 2 文件上传中整个请求内容允许的最大字节数。

- struts.custom.properties

  该属性指定 Struts 2 应用加载用户自定义的属性文件，该自定义属性文件指定的属性不会覆盖 struts.properties 文件中指定的属性。如果需要加载多个自定义属性文件，多个自定义属性文件的文件名以英文逗号（,）隔开。

- struts.mapper.class

  指定将 HTTP 请求映射到指定 Action 的映射器，Struts 2 提供了默认的映射器：org.apache.struts2.dispatcher.mapper.DefaultActionMapper。默认映射器根据请求的前缀与 Action 的 name 属性完成映射。

- struts.action.extension

  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 Struts 2 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

该属性设置是否通过 JAR 文件提供静态内容服务，该属性只支持 true 和 false 属性值，该属性的默认属性值是 true。

- struts.serve.static.browserCache

该属性设置浏览器是否缓存静态内容。当应用处于开发阶段时，我们希望每次请求都获得服务器的最新响应，则可设置该属性为 false。

- struts.enable.DynamicMethodInvocation

该属性设置 Struts 2 是否支持动态方法调用，该属性的默认值是 true。如果需要关闭动态方法调用，则可设置该属性为 false。

- struts.enable.SlashesInActionNames

该属性设置 Struts 2 是否允许在 Action 名中使用斜线，该属性的默认值是 false。如果开发者希望允许在 Action 名中使用斜线，则可设置该属性为 true。

- struts.tag.altSyntax

该属性指定是否允许在 Struts 2 标签中使用表达式语法，因为通常都需要在标签中使用表达式语法，故此属性应该设置为 true，该属性的默认值是 true。

- struts.devMode

该属性设置 Struts 2 应用是否使用开发模式。如果设置该属性为 true，则可以在应用出错时显示更多、更友好的出错提示。该属性只接受 true 和 flase 两个值，该属性的默认值是 false。通常，应用在开发阶段，将该属性设置为 true，当进入产品发布阶段后，则该属性设置为 false。

- struts.i18n.reload

该属性设置是否每次 HTTP 请求到达时，系统都重新加载资源文件。该属性默认值是 false。在开发阶段将该属性设置为 true 会更有利于开发，但在产品发布阶段应将该属性设置为 false。

提示：开发阶段将该属性设置了 true，将可以在每次请求时都重新加载国际化资源文件，从而可以让开发者看到实时开发效果；产品发布阶段应该将该属性设置为 false，是为了提供响应性能，每次请求都需要重新加载资源文件会大大降低应用的性能。

- struts.ui.theme

该属性指定视图标签默认的视图主题，该属性的默认值是 xhtml。

- struts.ui.templateDir

该属性指定视图主题所需要模板文件的位置，该属性的默认值是 template，即默认加载 template 路径下的模板文件。

- struts.ui.templateSuffix

该属性指定模板文件的后缀，该属性的默认属性值是 ftl。该属性还允许使用 ftl、vm 或 jsp，分别对应 FreeMarker、Velocity 和 JSP 模板。

- struts.configuration.xml.reload

该属性设置当 struts.xml 文件改变后，系统是否自动重新加载该文件。该属性的默认值是 false。

- struts.velocity.configfile

该属性指定 Velocity 框架所需的 velocity.properties 文件的位置。该属性的默认值为 velocity.properties。

- struts.velocity.contexts

该属性指定 Velocity 框架的 Context 位置，如果该框架有多个 Context，则多个 Context 之间以英文逗号（,）隔开。

- struts.velocity.toolboxlocation

该属性指定 Velocity 框架的 toolbox 的位置。

- struts.url.http.port

该属性指定 Web 应用所在的监听端口。该属性通常没有太大的用户，只是当 Struts 2 需要生成 URL 时（例如 Url 标签），该属性才提供 Web 应用的默认端口。

- struts.url.https.port

该属性类似于 struts.url.http.port 属性的作用，区别是该属性指定的是 Web 应用的加密服务端口。

- struts.url.includeParams

该属性指定 Struts 2 生成 URL 时是否包含请求参数。该属性接受 none、get 和 all 三个属性值，分别对应于不包含、仅包含 GET 类型请求参数和包含全部请求参数。

- struts.custom.i18n.resources

该属性指定 Struts 2 应用所需要的国际化资源文件，如果有多份国际化资源文件，则多个资源文件的文件名以英文逗号（,）隔开。

- struts.dispatcher.parametersWorkaround

对于某些 Java EE 服务器，不支持 HttpServlet Request 调用 getParameterMap() 方法，此时可以设置该属性值为 true 来解决该问题。该属性的默认值是 false。对于 WebLogic、Orion 和 OC4J 服务器，通常应该设置该属性为 true。

- struts.freemarker.manager.classname

该属性指定 Struts2 使用的 FreeMarker 管理器该属性的默认值是 org.apache.struts2.views.freemarker.FreemarkerManager，这是 Struts 2 内建的 FreeMarker 管理器。

- struts.freemarker.wrapper.altMap

该属性只支持 true 和 false 两个属性值，默认值是 true。通常无需修改该属性值。

- struts.xslt.nocache

该属性指定 XSLT Result 是否使用样式表缓存。当应用处于开发阶段时，该属性通常被设置为 true；当应用处于产品使用阶段时，该属性通常被设置为 false。

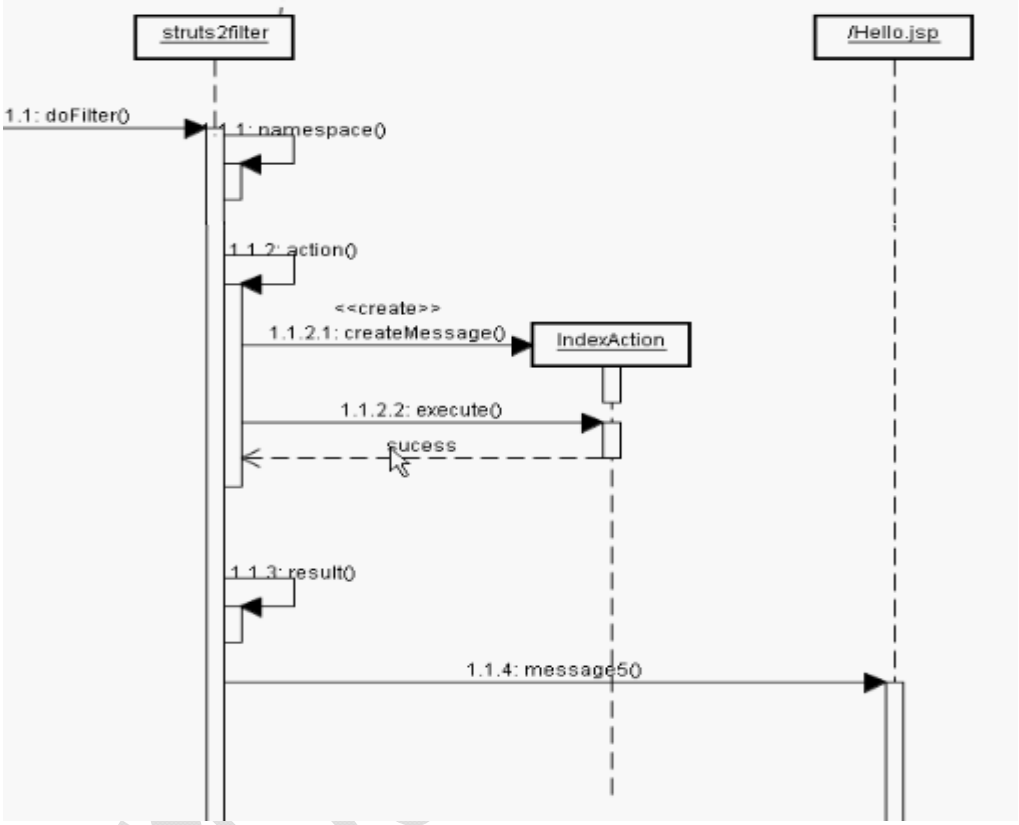
- struts.configuration.files

该属性指定 Struts 2 框架默认加载的配置文件，如果需要指定默认加载多个配置文件，则多个配置文件的文件名之间以英文逗号（,）隔开。该属性的默认值为 struts-default.xml,struts-plugin.xml,struts.xml，看到该属性值，读者应该明白为什么 Struts 2 框架默认加载 struts.xml 文件了

3. Struts2-Action 专题

3.1. Action 的实现

- 具体视图的返回可以由用户自己定义的 Action 来决定；
- 具体的手段是根据返回的字符串找到对应的配置项，来决定视图的内容，有三种手段：




```
<constant name="struts.devMode" value="true" />
    <package name="front" extends="struts-default" namespace="/">
        <action name="index"
class="com.bjsxt.struts2.front.action.IndexAction1">
            <result name="success">/ActionIntroduction.jsp</result>
        </action>
</package>
```

注：<action>标签中的 class 属性是表示 action 的对应的类(这个类是一个普通 Java 类)，当访问这个 action 时会创建这个类成为一个对象，然后执行这个对象中的 execute 方法()(execute 方法返回类型为 String)。

(1) 第一种:Action 普通 Java 类

```
public class IndexAction1 {
    public String execute() {
        return "success";
    }
    }
```

当 <action> 标签中 class 属性省略时，则默认执行 com.opensymphony.xwork2.ActionSupport 类中的 execute 方法,而这个方法返回一个字符串常量 SUCCESS(常量值为：“success”)。

(2) 第二种:Action

实现 com.opensymphony.xwork2.Action 接口，这个接口中定义了一些常量和一个 execute 方法，我们重写 execute()方法就可以了。

```
import com.opensymphony.xwork2.Action;
public class IndexAction2 implements Action {
    @Override
    public String execute() {
        //return "success";
        return this.SUCCESS; //SUCCESS常量值为："success"
    }
}
```

(3) 第三种:Action

继承 com.opensymphony.xwork2.ActionSupport 类，而这个类又实现了 com.opensymphony.xwork2.Action 接口，我们重写 execute()方法就可以了。

```
import com.opensymphony.xwork2.ActionSupport;
public class IndexAction3 extends ActionSupport {
    @Override
    public String execute() {
        //return "success";
        return this.SUCCESS;//SUCCESS常量值为："success"
    }
}
```

注：第三种 Action 是我们需要使用的方式，因为这个类不担实现了 com.opensymphony.xwork2.Action 接口，更重要的是它已经帮我封装了许多其它有用的方法。

(4) 路径问题的说明

Struts2 中的路径问题是根据 action 的路径而不是 jsp 路径来确定，所以尽量不要使用相对路径。虽然可以用 redirect 方式解决，但 redirect 方式并非必要。

解决办法非常简单，统一使用绝对路径。（在 jsp 中用 request.getContextRoot 方式来拿到 webapp 的路径）或者使用 myeclipse 经常用的，指定 basePath

腾科Java教学部

还有另一种方式，就是在<head>标签中，指定<base>标签值，这样就使用统一绝对路径。

```
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"
+request.getServerName()+":"+request.getServerPort()+path+"/";
%>

<head>
    <base href="<%=basePath%>" />
    ……
```

注：<base>标签：当前页面中所有连接都会在前面加上 base 地址。

3.2. Action 的动态调用方法

Action 执行的时候并不一定要执行 execute 方法，我们可以指定 Action 执行哪个方法：

3.2.1. 通过 methed 属性指定执行方法（方法一）

可以在配置文件中配置 Action 的时候用 method=来指定执行哪个方法

```
<action name="userAdd"
class="com.bjsxt.struts2.user.action.UserAction" method="add">
          <result>/user_add_success.jsp</result>
</action>
```

这样，只要在 action 的对象中有一个 add 的方法，并且返回类型为 String 就可以了。如果没有 method 属性，则默认执行 execute()方法。

```
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport {
    public String add() {
        return SUCCESS;
    }
}
```

3.2.2. 动态方法调用 DMI(推荐方法二)

可以在 url 地址中动态指定 action 执行那个方法。Url 地址如下：

```
http://localhost:8080/Struts2_0500_ActionMethod/user/user!add
```

方法：action + ! + 方法名

注：只要 Action 对象中有这个方法，并且返回类型为 String 就可以调用。这样

Struts.xml 配置文件中不需要配置 methed 属性。代码如下：

```
<action name="user"
class="com.bjsxt.struts2.user.action.UserAction">
<result>/user_add_success.jsp</result>
    </action>
```

Action 类：

```
public class UserAction extends ActionSupport {
    public String add() {
        return SUCCESS;
    }
    }
```

总结：推荐使用第二种动态方法调用 DMI，因为第一种需要大量的 Action 配置，后者可以在 url 中动态指定执行 action 中哪个方法。

3.3. Action 通配符(wildcard)的配置

使用通配符，将配置量降到最低，不过，一定要遵守"约定优于配置"的原则

通配符	含义
星号(*)	表示所有
{数字}	表示第几个通配符
例如：Student*	那么{1}代表第一个星号(*)
例如：*_*	那么{1}代表第一个星号(*)，{2}代表第二个星号(*)

实例

```
<package name="actions" extends="struts-default" namespace="/actions">
        <action name="Student*"
class="com.bjsxt.struts2.action.StudentAction" method="{1}">
            <result>/Student{1}_success.jsp</result>
        </action>
        <action name="*_*" class="com.bjsxt.struts2.action.{1}Action"
method="{2}">
            <result>/{1}_{2}_success.jsp</result>
            <!-- {0}_success.jsp -->
        </action>
    </package>
```

- **解释：**

第一个 Action 的名称为 name=“Student*” method=“{1}”,表示所有 Action 以 Student 开始的都会执行这个 Action,并且执行 Student 后字符为方法名的方法，例如：访问的 Action 为 Studentadd，会执行这个 Action(Student*)，并且执行 add 的方法.因为{1}在这里代表 add,

并且返回/Studentadd_success.jsp 页面。

第二个 Action 的名称 name="*_*" method="{2}" class="…action.{1}Action" 表示所有 Action 中包含下划线("_")都会执行这个 Action,例如：Teacher_add,那么会执行这个 Action, 并且 Action 对应的类为 TeacherAction,且执行 Action 中的 add 方法，返回结果页面为 /Teacher_add_success.jsp，因为在这里的{1}表示 Teacher,{2}表示 add

- **匹配顺序**

当匹配的 Action 有两个以上时，则会按匹配精确度高的那个 Action，当有个相同的匹配精确度时，则按先后顺序进行。

3.4. Action 的属性接收参数

3.4.1. 在 Action 添加成员属性接受参数

例如请求的 URL 地址：

```
http://localhost:8080/Struts2_0700_ActionAttrParamInput/user/user!add?name=a&age=8
```

其中传递了两个参数：name 和 age,其值分别为：a、8，此 Action 执行的是 add()方法。那我们只要在 user 这个 Action 对象中添加这两个成员属性并生成 set、get 方法。

```
public class UserAction extends ActionSupport {
    private String name;
    private int age;
    public String add() {
        System.out.println("name=" + name);
        System.out.println("age=" + age);
        return SUCCESS;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

3.4.2. 域模型(Domain Model)

就是利用对象域来进行传递和接受参数，例如请求的 URL 地址：

```
http://localhost:8080/Struts2_0800_DomainModelParamInput/user/user!add?user.name=a&user.age=8
```

其中，访问的是 namespace=“/user” action 的 name=“user” Action 所执行的方法 method=“add”利用对象域 user 来传递参数,为对象的属性赋值(user.name=a user.age=8)；

注：需要一个对象 user 并且这个对象需要有两个成员属性，且具有 get、set 方法。然后在 Action 中添加一个 User 对象的成员属性。并且有 get、set 方法，就可以了。

```
//User对象
public class User {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

//Action类

```
public class UserAction extends ActionSupport {
    private User user;
    //private UserDTO userDTO;
    public String add() {
        System.out.println("name=" + user.getName());
        System.out.println("age=" + user.getAge());
        return SUCCESS;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
```

传科Java教学部

```
        this.user = user;
    }
}
```

3.4.3. ModelDriven 接收参数

使 Action 实现 com.opensymphony.xwork2.ModelDriven<User>(在实现接口时需要使用泛型，否则使用时需要转型)中利用其 getModel()方法返回对象模型，从而获得传入的参数。例如 URL 如下：

```
http://localhost:8080/Struts2_0900_ModelDrivenParamInput/user/user!add?name=a&age=8
```

其：访问的是 namespace="/user"，action 的 name="user"，Action 所执行的方法 method="add"，其传入了两个参数：name=a，age=8。

参数被传入至 Action 后，会被 ModelDriven 对象根据参数名自动赋值给 User 对象相应的属性而生成 User 对象，并且由 getModel()返回。那么我们在 Action 中就可以利用这个对象了。

注意：传入的参数名需要与对象模型中的成员属性一致。对象模型 User:

```
public class User {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

Action 对象

```
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;
public class UserAction extends ActionSupport implements
ModelDriven<User>{
    private User user = new User();
    public String add() {
```

```
        System.out.println("name=" + user.getName());
        System.out.println("age=" + user.getAge());
        return SUCCESS;
    }
    @Override
    public User getModel() {
        return user;
    }
}
```

3.4.4. 获取参数

- 得到获取 Servlet 数据的入口

  ActionContext ctx = ActionContext.*getContext*();

- 得到 Session

  Map session = ctx.getSession();

- 得到 Application

  Map application = ctx.getApplication();

- 从 Action 跳转到 Action

  <result name="delete" type="redirect-action">user!delete.action</result>

3.5. Action 属性接收参数中文问题

如果表单提交数据中有中文时，尽量使用 post 方式。需要在 Struts.xml 配置文件中加入一个常量配置，如下：

```
<struts>
    <constant name="struts.devMode" value="true" />
    <constant name="struts.i18n.encoding" value="GBK" /><!--
internationalization -->
    <package name="user" extends="struts-default" namespace="/user">
        <action name="userAdd"
class="com.bjsxt.struts2.user.action.UserAction" method="add">
            <result>/user_add_success.jsp</result>
        </action>
    </package>
    </struts>
```

但是，在 Struts2 2.7 之前，这个配置无效，需要其它方法设置。如下：

手动在 web.xml 中在 Struts 过滤器之前配置一个过滤器用于解决中文的问题。

3.6. 访问 Web 元素

- 取得 Map 类型：request,session,application；

  依赖于容器 ； IOC (只用这种)

- 真实类型：HttpServletRequest, HttpSession, ServletContext 的引用:

  依赖于容器 ； IOC

3.6.1. 方法一：ActionContext 方式

一般在 Action 类的构造方法、或 execute()方法中获取。

```
public class LoginAction1 extends ActionSupport {
    private Map request;
    private Map session;
    private Map application;
    public LoginAction1() {
        request = (Map)ActionContext.getContext().get("request");
        session = ActionContext.getContext().getSession();
        application = ActionContext.getContext().getApplication();
    }
    public String execute() {
        request.put("r1", "r1");
        session.put("s1", "s1");
        application.put("a1", "a1");
        return SUCCESS;
    }
    }
```

然后在 Jsp 页面中获取相关 web 元素。

```
<body>
    User Login Success!
    <br />
    <s:property value="#request.r1"/> | <%=request.getAttribute("r1") %>
<br />
    <s:property value="#session.s1"/> | <%=session.getAttribute("s1") %>
<br />
    <s:property value="#application.a1"/> |
<%=application.getAttribute("a1") %> <br />
    <s:property value="#attr.a1"/><br />
    <s:property value="#attr.s1"/><br />
    <s:property value="#attr.r1"/><br />
    <s:debug></s:debug>
    <br />
    </body>
```

腾讯Java教学部

注：因为 request、session、application 对象 Struts2 将在放入到 Action Context 中，因此需要使用#key 来访问对象们。后面的是 java 脚本代码的访问方式。

3.6.2. 方式二：Ioc(控制反转)—推荐使用

让 Action 类实现 RequestAware、SessionAware、ApplicationAware 接口，然后重写他们的 set 方法(setRequest、setSession、setApplication)，通过依赖注入、控制反转(原来自己控制，现在由别人来控制值。)

```
import org.apache.struts2.interceptor.ApplicationAware;
import org.apache.struts2.interceptor.RequestAware;
import org.apache.struts2.interceptor.SessionAware;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction2 extends ActionSupport implements
RequestAware,SessionAware, ApplicationAware {
    private Map<String, Object> request;
    private Map<String, Object> session;
    private Map<String, Object> application;
    //DI dependency injection依赖注入
    //IoC inverse of control控制反转
    public String execute() {
        request.put("r1", "r1");
        session.put("s1", "s1");
        application.put("a1", "a1");
        return SUCCESS;
    }
    @Override
    public void setRequest(Map<String, Object> request) {
        this.request = request;
    }
    @Override
    public void setSession(Map<String, Object> session) {
        this.session = session;
    }
    @Override
    public void setApplication(Map<String, Object> application) {
        this.application = application;
    }
}
```

在视图(JSP)页面中获取相关对象，同方式一。

传智Java教学部

3.6.3. 方式三：获取原类型

获取是的 HttpServletRequest/HttpSession/ServletContext

```
public class LoginAction3 extends ActionSupport {

    private HttpServletRequest request;
    private HttpSession session;
    private ServletContext application;
    public LoginAction3() {
        request = ServletActionContext.getRequest();
        session = request.getSession();
        application = session.getServletContext();
    }
    public String execute() {
        request.setAttribute("r1", "r1");
        session.setAttribute("s1", "s1");
        application.setAttribute("a1", "a1");
        return SUCCESS;
    }
}
```

3.6.4. 方式四：获取原类型-控制反转

首先需要 Action 实现 org.apache.struts2.interceptor.ServletRequestAware 接口，然后重写 setServletRequest()方法，获取 HttpServletRequest 对象，再通过 HttpServletRequest 对象获取 HttpSession 和 ServletContext 对象。

```
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import org.apache.struts2.interceptor.ServletRequestAware;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction4 extends ActionSupport implements
ServletRequestAware {
    private HttpServletRequest request;
    private HttpSession session;
    private ServletContext application;
    public String execute() {
        request.setAttribute("r1", "r1");
        session.setAttribute("s1", "s1");
        application.setAttribute("a1", "a1");
        return SUCCESS;
    }
```

传智Java教学部

```
    @Override
    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
        this.session = request.getSession();
        this.application = session.getServletContext();
    }
}
```

3.7. Struts2 配置文件模块化包含(include)

<include>标签

当 Struts 配置文件比较多，需要模块化划分或分开成为多个配置文件时，则需要使用<include>标签把其它的配置文件引入到 Struts.xml 配置文件中就可以了。例如：有一个 login.xml 配置文件如下：

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="login" extends="struts-default" namespace="/login">
        <action name="login*"
class="com.bjsxt.struts2.user.action.LoginAction{1}">
            <result>/user_login_success.jsp</result>
        </action>
    </package>
</struts>
```

则需要在 struts.xml 文件中使用<include>标签将其引入就可以了。Struts.xml 如下

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.devMode" value="true" />
    <include file="login.xml" />
</struts>
```

3.8. 默认的 Action

当用户访问的 namespace 下一个不存在的 Action，则会将使用默认的 Action。使用<default-action-ref name="name">标签 其中 name 属性指向下面已经定义的 Action 名称了。

```
<struts>
    <constant name="struts.devMode" value="true" />

    <package name="default" namespace="/" extends="struts-default">
    <default-action-ref name="index"></default-action-ref>
    <action name="index">
        <result>/default.jsp</result>
    </action>
    </package>
</struts>
```

当前访问 namespace="/"下不存在的 Action 时，则返回自动转到访问 /default.jsp 页面。

4. struts2 标签

4.1. 数据标签

4.1.1. property 标签

1） 功能说明

获取对象的属性值，目标对象默认位于 ValueStack 栈顶

2） 标签属性

名称	必选	类型	说明
value	no	Object	对象的属性名称 默认直接输出ValueStack栈顶对象
default	no	String	默认值，如果value为null则输出此值
escape	no	Boolean	是否进行html转义，默认true
escapeJavaScript	no	Boolean	是否进行javascript转义，默认false

eg: <s:property value="user.userName"/>

4.1.2. set 标签

1） 功能说明

设定指定 scope 范围内对象的属性值。指出:这里的 scope 只能修改如下表格中的属性。

2） 标签属性

名称	必选	类型	说明
var	yes	String	被设置的对象属性名称
value	no	Object	值表达式
scope	no	String	范围，默认action(ActionContext) request/session/application

4.1.3. push 标签

1） 功能说明

将对象压入 ValueStack 栈顶，标签结束则出栈

2） 标签属性

名称	必选	类型	说明
value	yes	Object	被入栈对象的表达式

4.1.4. bean 标签

1） 功能说明

创建一个对象，并压入 ValueStack 栈顶

2） 标签属性

名称	必选	类型	说明
name	yes	String	实例的完全限定类名
var	no	String	在标签结束后能够继续引用实例的变量名

4.2. Form UI 标签

4.2.1. Form UI 简单表单

Struts2 Form UI Tag	html tag	说明
textfield	input type="text"	文本输入框
password	input type="password"	密码输入框
textarea	textarea	多行文本输入框
hidden	input type="hidden"	隐藏域
file	input type="file"	文件输入框
submit	input type="submit"	提交按钮
reset	input type="reset"	重置按钮
checkbox	input type="checkbox"	复选框

4.2.2. select 标签

1） 功能说明：

生成 select 标签（含 option）

2） 标签属性：

名称	类型	说明
list	集合	指定option列表项
listKey	String	如果list元素是复合对象，指定option value对应属性
listValue	Boolean	如果list元素是复合对象，指定option text对应属性
emptyOption	Boolean	是否添加空option
multiple	Boolean	是否允许多选

eg:

<s:select list="products" listKey="id" listValue="name" label="产品" />

4.2.3. checkboxlist 标签

1） 功能说明：
生成一组 checkbox
2） 标签属性：

名称	类型	说明
list	集合	指定option列表项
listKey	String	如果list元素是复合对象，指定option value对应属性
listValue	Boolean	如果list元素是复合对象，指定option text对应属性

eg: <s:checkboxlist name="favors" list="products" listKey="id" listValue="name"/>

4.2.4. Radio 标签

1） 功能说明：
生成一组 radio
2） 标签属性：

名称	类型	说明
list	集合	指定option列表项
listKey	String	如果list元素是复合对象，指定option value对应属性
listValue	Boolean	如果list元素是复合对象，指定option text对应属性

eg: **<s:radio name="favorite" list="products" listKey="id" listValue="name"/>**

4.3. 控制标签

4.3.1. if-elseif-else 标签

1） 功能说明：
替代 Java 语法中的 if/else
2） 标签属性：

名称	必选	类型	说明
test	yes	Boolean	表达式，决定是否显示 if/elseif 标签体的内容

```
欢迎${user.name}，
<s:if test="user.role == 'admin'">系统管理员</s:if>
<s:elseif test="user.role == 'member'">会员</s:elseif>
<s:else>游客</s:else>
```

4.3.2. iterator 标签

1） 功能说明：
替代 Java 语言中 foreach 的功能
2） 标签属性：

名称	必选	类型	说明
value	true	Object	被迭代的对象
status	no	String	迭代状态，IteratorStatus对象

```
<s:iterator value="products">
<s:property value="name"/>,<s:property value="price"/>
<br />
</s:iterator>
```

5. struts2 ognl 表达式

OGNL：OGNL(Object Graph Navigating Language)即对象导航图语言，是表达式语言的一种。通常用来绑定 Java 对象的属性，用一个表达式就可以读取和设置对应 Java 对象的属性，所以可以用来沟通 Web 组件和业务逻辑的模型部分。它是 Jakarta“属性存取语言”项目和 JSTL 的表达式语言的良好替代，常用的标号有#,%,$

1. #的作用：
1） 访问 OGNL 上下文和 Action 上下文，#相当于 ActionContext.getContext()；
下表有几个 ActionContext 中有用的属性：

- parameters:包含当前 HTTP 请求参数的 Map #parameters.id[0]作用相当于 request.getParameter("id")
- request:包含当前 HttpServletRequest 的属性（attribute)的 Map #request.userName 相当于 request.getAttribute("userName")
- session:包含当前 HttpSession 的属性（attribute）的 Map #session.userName 相当于 session.getAttribute("userName")
- application:包含当前应用的 ServletContext 的属性（attribute）的 Map #application.userName 相当于 application.getAttribute("userName")
- attr:用于按 request > session > application 顺序访问其属性(attribute) #attr.userName 相当于按顺序在以上三个范围（scope）内读取 userName 属性，直到找到为止

2） 用于过滤和投影（projecting)集合，如
#request.books.{?#this.price<100}；

- ?：显示所有满足条件的对象信息
- ^：显示满足条件的第一个对象信息
- $：显示满足条件的最后一个对象信息

```
<s:iterator value="#request.users.{?#this.role=='admin'}">
        <tr>
            <td><s:property value="name"/></td>
            <td><s:property value="password"/></td>
            <td><s:property value="role"/></td>
        </tr>
</s:iterator>
```

3） 构造 Map，如#{'foo1':'bar1', 'foo2':'bar2'}。

```
<s:set name="foobar" value="#{'foo1':'bar1', 'foo2':'bar2'}" />
<p>The value of key "foo1" is <s:property value="#foobar['foo1']" /></p>
```

2. %的作用：

1） 在标志的属性为字符串类型时，计算 OGNL 表达式的值。

2） 国际化时，读取资源文件的数据

3. $的作用：

1） 用于在国际化资源文件中，引用 OGNL 表达式

2） 在 Struts 2 配置文件中，引用 OGNL 表达式，如：

```
<action name="ognlAction" class="day3.com.action.OGNLAction">
        <param name="myurl">/index.jsp</param>
        <result>${myurl}</result>
</action>
```

3） 在验证框架引用变量：

```
<field name ="user.age">
    <field-validator type ="int">
        <param name="min">1</param>
        <param name="max">100</param>
          <message> This age's value must between ${min} and ${max}</message>
    </field-validator>
</field>
```

6. struts2 国际化

1. 国际化概念

国际化（internationalization，又称 i18n，因为 i 和 n 之间有 18 个字母）是指应用程序支持多种语言和格式化习惯集合。它是为了满足应用程序在世界上不同地区的用户群的习惯而提出的，这包括两个方面的主要内容，即语言和格式化。

2. 国际化实现步骤

1） 在 resources 包下建立 Lomen_en.properties（英文皮肤）

Lomen_zh_CN.properties（中文皮肤）文件：

- Lomen_en.properties

```
login.userName = user name
```

```
login.userPassword = password
```

- Lomen_zh_CN.properties

```
login.userName = \u7528\u6237\u540D(Unicode编码—用户名)

login.userPassword = \u5BC6\u7801(Unicode 编码—密码)
```

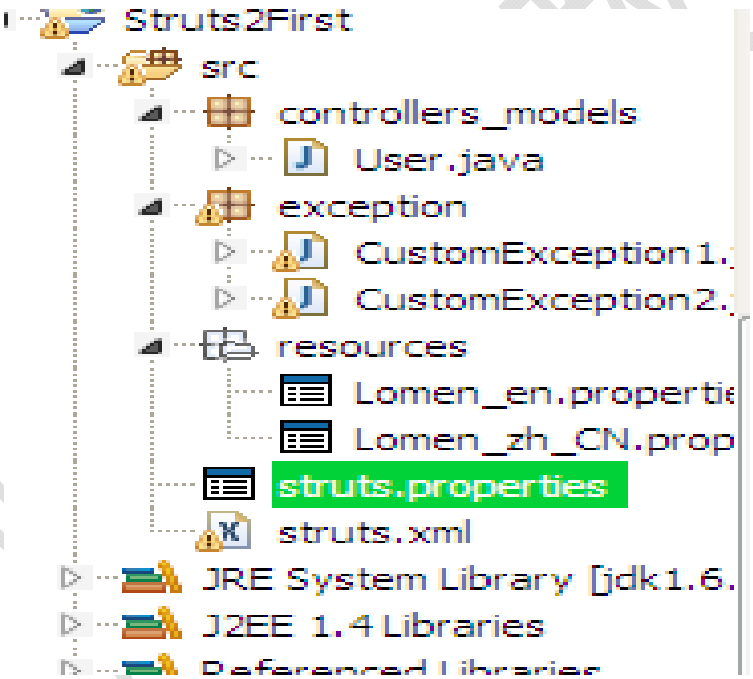
注意：中文要经过编码，配置好 JDK 的情况下，在 cmd 命令行中输入（以下只是一个实例，其中第一个参数是原文件后面一个参数是生成后的目标文件，使用时用后面这个文件就可）

```
F:\>native2ascii zh_CN.properties EricResource_zh_CN.properties
```

2） 添加国际化资源库

在 WEB-INF 下的 classes 下建立 struts.properties 文件：

struts.custom.i18n.resources=resources.Lomen



3） 页面显示国际化信息

```
<s:form action="user!login">
<s:text name="login.userName"/>:<s:textfield
name="userName"/>
<s:text name="login.userPassword"/>:<s:password
name="userPassword"/>

<s:submit value="增加用户"/>
```

```
</s:form>
   或：
<s:textfield name="userName" key="login.userName"/>
```

3. 资源文件查找顺序

struts 2.0的国际化更灵活是因为它可以能根据不同需要配置和获取资源（properties）文件。在struts 2.0中有下面几种方法：

1) 使用全局的资源文件，方法如上例所示。这适用于遍布于整个应用程序的国际化字符串，它们在不同的包（package）中被引用，如一些比较共用的出错提示；
2) 使用包范围内的资源文件。做法是在包的根目录下新建名的package.properties和package_xx_XX.properties文件。这就适用于在包中不同类访问的资源；
3) 使用Action范围的资源文件。做法为Action的包下新建文件名（除文件扩展名外）与Action类名同样的资源文件。它只能在该Action中访问。如此一来，我们就可以在不同的Action里使用相同的properties名表示不同的值。例如，在ActonOne中title为“动作一”，而同样用title在ActionTwo表示“动作二”，节省一些命名工夫；
4) 使用<s:i18n>标志访问特定路径的properties文件。使用方法请参考我早前的文章《常用的Struts 2.0的标志（Tag）介绍》。在您使用这一方法时，请注意<s:i18n>标志的范围。在<s:i18n name="xxxxx">到</s:i18n>之间，所有的国际化字符串都会在名为xxxxx资源文件查找，如果找不到，Struts 2.0就会输出默认值（国际化字符串的名字）。
5) 上面我列举了四种配置和访问资源的方法，它们的范围分别是从大到小，而Struts 2.0在查找国际化字符串所遵循的是特定的顺序，如下图所示：

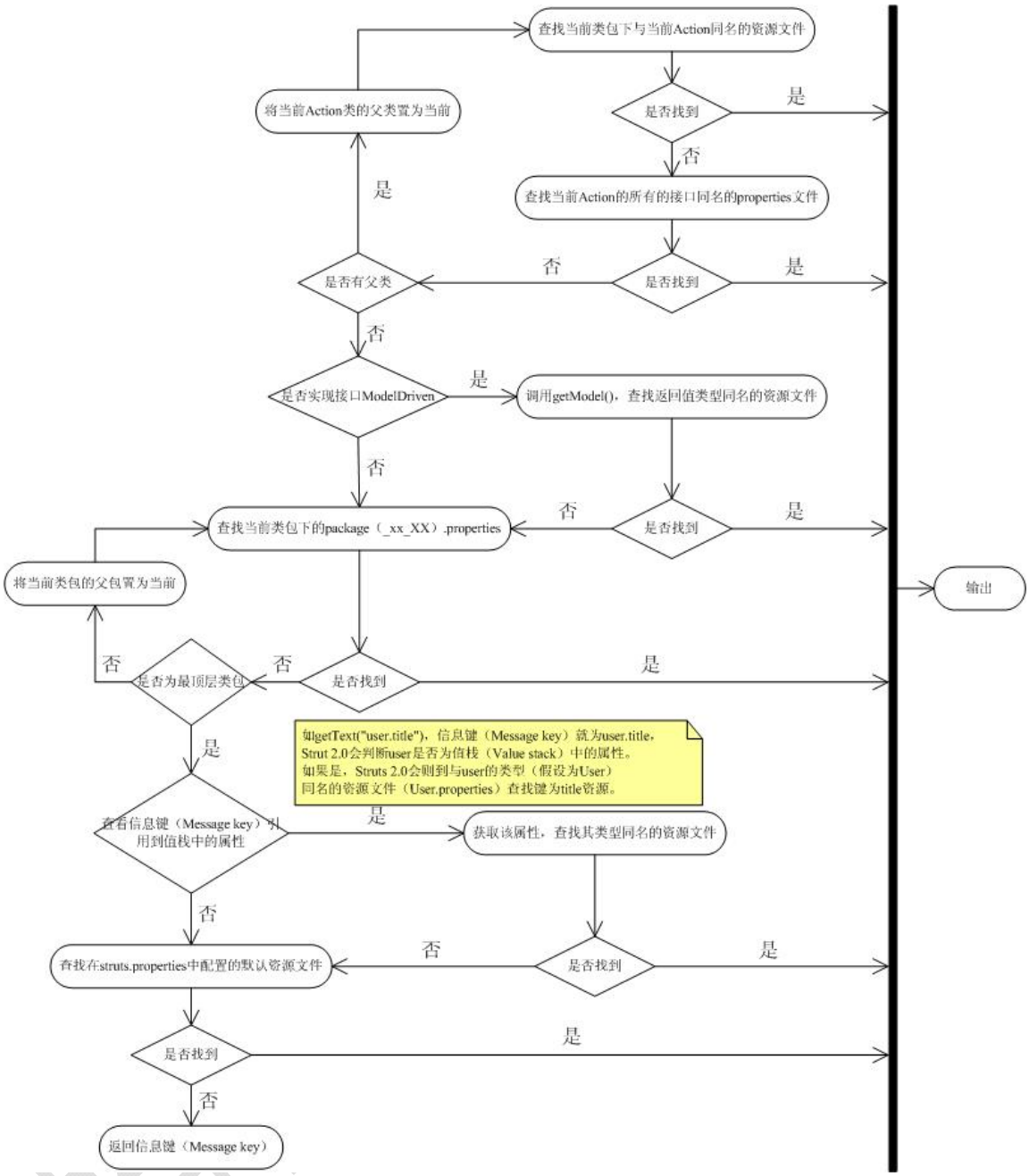


图3 资源文件查找顺序图

假设我们在某个ChildAction中调用了getText("user.title")，Struts 2.0的将会执行以下的操作：

1) 查找ChildAction_xx_XX.properties文件或ChildAction.properties；
2) 查找ChildAction实现的接口，查找与接口同名的资源文件MyInterface.properties；
3) 查找ChildAction的父类ParentAction的properties文件，文件名为ParentAction.properties；
4) 判断当前ChildAction是否实现接口ModelDriven。如果是，调用getModel()获得对象，查找与其同名的资源文件；
5) 查找当前包下的package.properties文件；

6) 查找当前包的父包，直到最顶层包；
7) 在值栈（Value Stack）中，查找名为user的属性，转到user类型同名的资源文件，查找键为title的资源；
8) 查找在struts.properties配置的默认的资源文件，参考例1；
9) 输出user.title。
10) 参数化国际化字符串
11) 许多情况下，我们都需要在动行时（runtime）为国际化字符插入一些参数，例如在输入验证提示信息的时候。在Struts 2.0中，我们通过以下两种方法做到这点：
12) 在资源文件的国际化字符串中使用OGNL，格式为${表达式}，例如：
validation.require=${getText(fileName)} is required
13) 使用java.text.MessageFormat中的字符串格式，格式为{ 参数序号（从0开始），格式类形（number | date | time | choice），格式样式}，例如：
validation.between=Date must between {0, date, short} and {1, date, short}
14) 在显示这些国际化字符时，同样有两种方法设置参数的值：
A. 使用标志的value0、value1...valueN的属性，如：

```
<s:text name="validation.required" value0="User Name"/>
```

B. 使用param子元素，这些param将按先后顺序，代入到国际化字符串的参数中，例如：

```
<s:text name="validation.required">
<s:param value="User Name"/>
</s:text>
```

4. 让用户选择语言

```
public class I18NAction    {
    private String lang;
    public String getLang() {
        return lang;
    }
    public void setLang(String lang) {
        this.lang = lang;
    }
    public String execute(){
        if(lang.equals("zh")){
            ActionContext.getContext().setLocale(Locale.CHINA);
        }
        if(lang.equals("en")){
            ActionContext.getContext().setLocale(Locale.ENGLISH);
        }

        return "i18N";
    }
}
```

腾讯java教学部

7. struts2 拦截器

1. 拦截器概念：功能是类似 web 中的 Filter

在 struts-default.xml 中已经配置了以上的拦截器。如果您想要使用上述拦截器，只需要在应用程序 struts.xml 文件中通过“<include file="struts-default.xml" />”将 struts-default.xml 文件包含进来，并继承其中的 struts-default 包（package），最后在定义 Action 时，使用“<interceptor-ref name="xx" />”引用拦截器或拦截器栈（interceptor stack）。一旦您继承了 struts-default 包（package），所有 Action 都会调用拦截器栈 ——defaultStack。当然，在 Action 配置中加入“<interceptor-ref name="xx" />”可以覆盖 defaultStack。

2. 已有的拦截器

```
<interceptor name="alias"
class="com.opensymphony.xwork2.interceptor.AliasInterceptor"/>
<interceptor name="autowiring"
    class="com.opensymphony.xwork2.spring.interceptor.ActionAutowiringIntercept
or"/>
<interceptor                                                    name="chain"
class="com.opensymphony.xwork2.interceptor.ChainingInterceptor"/>
<interceptor name="conversionError"
    class="org.apache.struts2.interceptor.StrutsConversionErrorInterceptor"/>
<interceptor                                                  name="cookie"
class="org.apache.struts2.interceptor.CookieInterceptor"/>
<interceptor name="clearSession"
    class="org.apache.struts2.interceptor.ClearSessionInterceptor" />
<interceptor name="createSession"
    class="org.apache.struts2.interceptor.CreateSessionInterceptor" />
<interceptor name="debugging"
    class="org.apache.struts2.interceptor.debugging.DebuggingInterceptor" />
<interceptor name="externalRef"
    class="com.opensymphony.xwork2.interceptor.ExternalReferencesInterceptor"/>
<interceptor name="execAndWait"
    class="org.apache.struts2.interceptor.ExecuteAndWaitInterceptor"/>
<interceptor name="exception"
    class="com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor"/>
<interceptor                                                  name="fileUpload"
class="org.apache.struts2.interceptor.FileUploadInterceptor"/>
<interceptor                                                     name="i18n"
class="com.opensymphony.xwork2.interceptor.I18nInterceptor"/>
<interceptor                                                   name="logger"
class="com.opensymphony.xwork2.interceptor.LoggingInterceptor"/>
<interceptor name="modelDriven"
    class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>
<interceptor name="scopedModelDriven"
    class="com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor"/
```

```
>
<interceptor name="params"
    class="com.opensymphony.xwork2.interceptor.ParametersInterceptor"/>
<interceptor name="actionMappingParams"
    class="org.apache.struts2.interceptor.ActionMappingParametersInteceptor"/>
<interceptor name="prepare"
    class="com.opensymphony.xwork2.interceptor.PrepareInterceptor"/>
<interceptor name="staticParams"
    class="com.opensymphony.xwork2.interceptor.StaticParametersInterceptor"/>
<interceptor name="scope" class="org.apache.struts2.interceptor.ScopeInterceptor"/>
<interceptor name="servletConfig"
    class="org.apache.struts2.interceptor.ServletConfigInterceptor"/>
<interceptor name="sessionAutowiring"
    class="org.apache.struts2.spring.interceptor.SessionContextAutowiringIntercepto
r"/>
<interceptor                                                    name="timer"
class="com.opensymphony.xwork2.interceptor.TimerInterceptor"/>
<interceptor name="token" class="org.apache.struts2.interceptor.TokenInterceptor"/>
<interceptor name="tokenSession"
    class="org.apache.struts2.interceptor.TokenSessionStoreInterceptor"/>
<interceptor name="validation"
    class="org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor"
/>
<interceptor name="workflow"
    class="com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor"/>
<interceptor                                                      name="store"
class="org.apache.struts2.interceptor.MessageStoreInterceptor" />
<interceptor                                                  name="checkbox"
class="org.apache.struts2.interceptor.CheckboxInterceptor" />
<interceptor name="profiling"
    class="org.apache.struts2.interceptor.ProfilingActivationInterceptor" />
<interceptor name="roles" class="org.apache.struts2.interceptor.RolesInterceptor" />
<interceptor name="jsonValidation"
    class="org.apache.struts2.interceptor.validation.JSONValidationInterceptor" />
<interceptor name="annotationWorkflow"
class="com.opensymphony.xwork2.interceptor.annotations.AnnotationWorkflowInter
ceptor" />
<interceptor                                                  name="multiselect"
class="org.apache.struts2.interceptor.MultiselectInterceptor" />
```

3. 使用拦截器

```
<action name="interceptorAction"
            class="day2.com.action.InterceptorAction">
            <interceptor-ref name="timer"></interceptor-ref>
            <interceptor-ref name="logger"></interceptor-ref>
```

```
            <result>/day2/login.jsp</result>
</action>
```

此时,struts-default.xml 中的 defaultStack 将会被覆盖，要手工引入

4. 所有 action 都用上同样的拦截器

<default-interceptor-ref name="mystack"></default-interceptor-ref> ， 此时,struts-default.xml 中的 defaultStack 将会被覆盖，要手工引入

5. 自定义拦截器：类级别拦截器，方法级别拦截器

1) 类级别拦截器：

A. 写一个类继承 AbstractInterceptor 或者实现 Interceptor

```
public class AuthorizationInterceptor extends AbstractInterceptor {

    @Override
    public String intercept(ActionInvocation invocation) throws Exception {
        System.out.println("AuthorizationInterceptor's intercept()...");
        return invocation.invoke();
    }
}
```

B. 在 struts.xml 声明该拦截器

```
<interceptors>
            <interceptor name="authorInterceptor"
                class="day2.com.interceptor.AuthorizationInterceptor">
            </interceptor>
            <interceptor-stack name="mystack">
                <interceptor-ref name="authorInterceptor"></interceptor-ref>
                <interceptor-ref name="defaultStack"></interceptor-ref>
            </interceptor-stack>
</interceptors>
```

C. 引用该拦截器：

```
<action name="interceptorAction"
            class="day2.com.action.InterceptorAction">
            <interceptor-ref name="authorInterceptor"></interceptor-ref>
            <result>/day2/login.jsp</result>
</action>
```

D. 拦截器生命方法：init(),intercept(ActionInvocation invocation),destroy()

E. 如果在 action 里有引用某些拦截器，则会覆盖<default-interceptor-ref name="mystack"></default-interceptor-ref>的拦截器，要手工引入

2) 方法级别拦截器：

A. 写一个类继承 MethodFilterInterceptor

```
public class MethodAuthorizationInterceptor extends MethodFilterInterceptor {

    @Override
    public String doIntercept(ActionInvocation invocation) throws Exception {
        System.out.println("AuthorizationInterceptor's intercept()...");
```

```
        return invocation.invoke();
    }
}
```

B. 在 struts.xml 声明该拦截器：

```
<interceptors>
        <interceptor name="methodAuthorInterceptor"
            class="day2.com.interceptor.MethodAuthorizationInterceptor">
        </interceptor>
</interceptors>
```

C. 引用该拦截器：

```
<action name="interceptorAction"
        class="day2.com.action.InterceptorAction">
        <interceptor-ref name="methodAuthorInterceptor">
                <!--需要拦截的方法-->
                <param name="includeMethods">m1,m2</param>
                <!--不需要拦截的方法-->
                <param name="excludeMethods">m1,m2</param>
        </interceptor-ref>
        <result>/day2/login.jsp</result>
</action>
```

D. 拦截器生命方法：init(),doIntercept(ActionInvocation invocation),destroy()

8. struts2 转换器

1. 已有的转化器：

1)预定义类型，八种基本类型+String；

2)日期类型， 使用当前区域（Locale）的短格式转换，即 DateFormat.getInstance(DateFormat.SHORT)；

3)集合（Collection）类型， 将request.getParameterValues(String arg)返回的字符串数据与java.util.Collection转换；

4)集合（Set）类型， 与List的转换相似，去掉相同的值；

5)数组（Array）类型， 将字符串数组的每一个元素转换成特定的类型，并组成一个数组。

2. 自定义转化器步骤：

1)编写一个转化器类

```
public class DateConverter extends DefaultTypeConverter {
    private SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    /**
     * map: ActionContext value: 需要转化的值：String[] toType:要转化成的目标类型
     */
    public Object convertValue(Map map, Object value, Class toType) {
        System.out.println("DateConverter's convertValue()被调用了===>" + this);
```

```
        if (toType == java.util.Date.class) {
            String strDate = ((String[]) value)[0];

            try {
                return sdf.parse(strDate);
            } catch (ParseException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                return null;
            }
        }
        return null;
    }
}
```

2)声明局部的：在要转换的action目录下新建一个XXX-conversion.properties 文件 birthday=day1.com.converter.DateConverter
3)声明全局的：在src下新建一个xwork-conversion.properties文件
java.util.Date=day1.com.converter.DateConverter

类型转换时的默认错误提示信息在 xwork-2.0.4.jar 中的 com.opensymphony.xwork2/xwork-messages.properties 文件上,
如果想要修改转换时的错误提示信息，要在对应的 action 同目录下新建一个 XXX(action 的名字).properties 文件,写上 invalid.fieldvalue.user.age(action 的属性名)=年龄输入格式不正确

9. Struts2-异常处理

9.1. 自定义异常类

```
package exception;
public class CustomException1 extends Exception {
   public CustomException1( String msg ){
      super( msg );
   }
}
```

9.2. 异常配置

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE struts PUBLIC
```

```
        "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
        "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="user" namespace="/user" extends="struts-default">

        <global-results>
            <result name="Welcome">Welcome.jsp</result>
            <result name="ERROR1">Error.jsp</result>
        </global-results>

        <global-exception-mappings>
            <exception-mapping result="ERROR1"
exception="exception.CustomException1"/>
            <exception-mapping result="ERROR2"
exception="exception.CustomException1"/>
        </global-exception-mappings>

        <action name="user"  class="controllers_models.User"/>
    </package>
</struts>
```

9.3. 异常显示页面

<s:property value="exception.message"/>

如果要改变错误的显示效果可以在其外添加显示效果标签如：

<s:property value="exception.message" />

10. Struts2-数据验证

10.1. 验证种类

1. validate 方法验证
2. 验证框架

腾科Java教学部

10.2. validate 方法验证

10.2.1. 国际化文件

在资源文件中添加以下一行

```
user.password.error=the userPassword1 is different from userPassword
```

注意：一定要有 Lomen.properties（注册时也是用这个）这个资源文件，否则可能找不到 user.password.error 的位置

10.2.2. Action 类:

```
public class User extends ActionSupport{

   ……
    public void validate() {

       // 前台验证

       if( this.userPassword != this.userPassword1 ){
          this.addFieldError( "userPassword",
this.getText( "user.password.error" ) );
       }
    }

   ……
}
```

10.2.3. 配置文件

- 配置文件中一定要有一个叫做 input 的 result，并且当前 action 可以访问这个 result，这是错误信息输送页面

```
<global-results>
    <result name="Welcome">Welcome.jsp</result>
    <result name="ERROR1">Error.jsp</result>
    <result name="input">login.jsp</result>
</global-results>
```

- 如果错误信息输送到数据录入页面，则错误信息会显示在数据错误的输入框的上方

10.2.4. 错误信息的显示

```
<s:form action="user!login">
 <s:textfield name="userName" label="用户名"/>
 <s:textfield name="userPassword" label="密码"/>
 <s:textfield name="userPassword1" label="密码确认"/>
 <s:submit value="登陆"/>
    </s:form>
```

哪项验证失败后，报错信息会自动显示到本那个输入框上方（这里是 userPassword 输入框的上方）

10.3. 验证框架

10.3.1. Action 类

```
public class User extends ActionSupport {
   ……
   }
```

10.3.2. 验证文件

10.3.2.1. 在被验证的 Action 的 class 文件目录下

User 类的 class 文件相同路径下建立一个“Action 类名-validation.xml”文件

如：User-validation.xml

10.3.2.2. 文件中的内容如下：

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
    "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
```

```
    <field name="userName">
        <field-validator type="requiredstring">
            <message key="user.name.exist"/>
        </field-validator>
        </field>
</validators>
```

10.4. 验证规则介绍

10.4.1. 非空验证

- 方法一：

```
<field name="userName">
    <field-validator type="required">
        <message key="user.name.exist"/>
    </field-validator>
</field>
```

- 方法二：

```
<field name="userName">
    <field-validator type="required">
        <message>${getText("user.name.exist")}</message>
    </field-validator>
</field>
```

10.4.2. 非空字符串验证

```
<field name="userName">
    <field-validator type="requiredstring">
        <message key="user.name.exist"/>
     </field-validator>
</field>
```

10.4.3. 字符串长度验证

```
<field name="userName">
        <field-validator type="stringlength">
```

```
        <param name="minLength">3</param>
           <param name="maxLength">6</param>
           <message key="user.name.length"/>
       </field-validator>
</field>
```

10.4.4. 正则表达式验证

```
<field name="userName">
   <field-validator type="regex">
       <param name="expression"><![CDATA[.*hello.*]]></param>
       <message>${getText("user.name.regex")}</message>
   </field-validator>
</field>
```

10.4.5. 整数验证-（控制整数的大小）

```
<field name="userAge">
  <field-validator type="int">
       <param name="min">3</param>
        <param name="max">6</param>
        <message>${getText("user.name.length")}</message>
  </field-validator>
</field>
```

10.4.6. 表达式验证

判断多个字段间的关系是否满足某个表达式，如：密码和密码确认输入的数据是否相同。

```
<field name="userPassword">
   <field-validator type="fieldexpression">
       <param name="expression">
           <![CDATA[userPassword==userPassword1]]>
       </param>
       <message>${getText("user.password.error")}</message>
   </field-validator>
</field>
```

腾科Java教学部

判断的表达式，直接写入要判断的各个属性

10.4.7. 日期验证

```
<field name="birthday">
    <field-validator type="date">
        <param name="min">06-1-1</param>
        <param name="max">08-1-1</param>
        <message>${getText("user.name.length")}</message>
    </field-validator>
</field>
```

日期格式按当前的 locale 的方式编写

11. struts2 token 机制

token 机制：防止表单重复提交

1. 原理：

在页面加载时，<s: token />产生一个 GUID（Globally Unique Identifier，全局唯一标识符）值的隐藏输入框如：

Java 代码

1)<input type="hidden" name="struts.token.name" value="struts.token"/>

2)<input type="hidden" name="struts.token" value="BXPNNDG6BB11ZXHPI4E106CZ5K7VNMHR"/>

同时，将 GUID 放到会话（session）中；在执行 action 之前，“token”拦截器将会话 token 与请求 token 比较，如果两者相同，则将会话中的 token 删除并往下执行，否则向 actionErrors 加入错误信息。如此一来，如果用户通过某种手段提交了两次相同的请求，两个 token 就会不同。

2. 使用步骤：

1） 在要提交的表单中加上 <s:token></s:token>，如：

```
<font color="red"><s:actionerror/></font>
<s:form action="tokenAction" method="POST" theme="simple">
        <s:token></s:token>
        <p>
            <s:text name="login.username"></s:text>
            <s:textfield name="name"></s:textfield>
        </p>
        <p>
            <s:submit value="提交"></s:submit>
        </p>
</s:form>
```

2） 在对应的 action 配置 token 拦截器

```
<action name="tokenAction" class="day3.com.action.TokenAction">
            <interceptor-ref name="defaultStack"></interceptor-ref>
             <!-- 注意 struts2 拦截器名字为 token-session struts2.1.2 已经更改为 tokenSession -->
                        token: 在活动中检查合法令牌(token), 防止表单的重复提交; 在<s:actionerror/>会产生提示信息
                        tokenSession: 同上, 但是在接到非法令牌时将提交的数据保存在 session 中; 不会在<s:actionerror/>会产生提示信息-->
             <interceptor-ref name="token">
                    <param name="excludeMethods">input</param>
            </interceptor-ref>
              <result>/day3/success.jsp</result>
              <result name="invalid.token">/day3/register_token.jsp</result>
</action>
```

12. Struts2 文件上传和下载

12.1. 文件上传

1. singlefileupload.jsp:

```
<s:form action="fileuploadAction.action" method="post"
enctype="multipart/form-data">
    <s:file name="myFile" label="选择文件"></s:file>
    <s:submit value="提交" align="center"></s:submit>
</s:form>
```

2. SingleFileUploadAction

```
package com.fileupload;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import com.util.BaseAction;
public class SingleFileUploadAction extends BaseAction {
    private File file;
```

```
private String fileName;
private String contentType;
public void setMyFile(File file){
    this.file = file;
}
public void setMyFileFileName(String fileName){
    this.fileName = fileName;
}
public void setMyFileContentType(String contentType){
    this.contentType = contentType;
}
public String execute() throws Exception{
    File destFile = new File(context.getRealPath("/uploadfiles")+"/"+fileName);
    copy(file,destFile);
    System.out.println("文件上传成功...");
    session.put("filePath", "/uploadfiles/"+fileName);
    return "singleFileUpload";
}

//拷贝文件
/*
 * srcFile：客户端传服务器的文件对象(struts2 帮我们完成)
 * destFile：将客户端传过来的文件写到的目标文件对象(需要我们完成)
 */
public void copy(File srcFile,File destFile) {
    System.out.println("srcFile==>"+srcFile.getAbsolutePath());
    System.out.println("destFile==>"+destFile.getAbsolutePath());
    BufferedInputStream bis = null;
    BufferedOutputStream bos = null;

    try {
        FileInputStream fis = new FileInputStream(srcFile);
        FileOutputStream fos = new FileOutputStream(destFile);
        bis = new BufferedInputStream(fis);
        bos = new BufferedOutputStream(fos);
        byte[] buf = new byte[1012];
        int length = 0;
        while((length = bis.read(buf))!=-1){
            bos.write(buf,0,length);
        }
        bos.flush();
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
```

腾科Java教学部

```
            }catch(IOException ex){
                ex.printStackTrace();
            }finally{
                if(bis!=null){
                    try {
                        bis.close();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
                if(bos!=null){
                    try {
                        bos.close();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }
        }
}
```

3. struts.xml 配置

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC

    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

    "http://struts.apache.org/dtds/struts-2.1.7.dtd">

<struts>

    <package name="fileupload_package" extends="struts-default">

        <action                                          name="fileuploadAction_*"

class="com.fileupload.SingleFileUploadAction" method="{1}">

            <interceptor-ref name="fileUpload">

                <param name="allowedTypes">
```

```
    image/bmp,image/png,image/jpg,image/pjpeg,image/gif,image/jpeg
                </param>
            </interceptor-ref>
            <interceptor-ref name="fileUploadStack"></interceptor-ref>
            <interceptor-ref name="defaultStack"></interceptor-ref>
            <result
name="singleFileUpload">/fileupload/single_result.jsp</result>
            <result name="input">/fileupload/singlefileupload.jsp</result>
        </action>
    </package>
</struts>
```

4. 上传的文件大小设置

默认最大的字节数是 87917472, 如果要想修改上传文件的最大字节数，则应在 struts.properties 配置如下信息：

```
struts.multipart.saveDir=/tmp
struts.multipart.maxSize=1021438150
```

5. 过滤上传的文件类型

```
<interceptor-ref name="fileUpload">
    <param name="allowedTypes">
    image/bmp,image/png,image/jpg,image/pjpeg,image/gif,image/jpeg
    </param>
</interceptor-ref>
```

12.2. 文件下载

1. Filedownload.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>文件下载列表</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
</head>
  <body>
    <table border="1">
        <c:forEach var="file" items="${files}">
            <tr>
                <td>${file.name }</td>
                <td><a
href="downfileAction_lookup.action?fileName=${file.name  }&realPath=${file.a
```

腾科Java教学部

```
bsolutePath} ">浏览</a></td>
                    <td><a
href= "download.action?fileName=${file.name }&realPath=${file.absolutePath} "
>下载</a></td>
                </tr>
            </c:forEach>
        </table>
    </body>
</html>
```

2. DownFileAction

```
package com.fileupload;
import java.io.File;
import java.io.InputStream;
import com.util.BaseAction;
public class DownfileAction extends BaseAction {
    public String findFiles() throws Exception {
        String fileDir = context.getRealPath("/uploadfiles");
        File file = new File(fileDir);
        File[] files = file.listFiles();
        request.getSession().setAttribute("files", files);
        return "fileList";
    }
    public String download() throws Exception {
        System.out.println("DownfileAction's download()...");
        return "filedownload";
    }
    public String lookup() throws Exception {
        System.out.println("DownfileAction's lookup()...");
        return "filedownload";
    }
    //返回显示在下载框中文件的默认名
    public String getFileName(){
```

腾讯Java教学部

```
            String fileName = request.getParameter("fileName");
            try {
            fileName = new String(fileName.getBytes("ISO-8859-1"), "UTF-8");

            } catch (Exception e) {
              e.printStackTrace();
            }
            return fileName;
      }
      //返回要下载的文件的输入流
      public InputStream getDownloadFile(){
            InputStream is = context.getResourceAsStream("/uploadfiles/"+getFileName());
            return is;
      }
}
```

3. Struts.xml 配置

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
      "http://struts.apache.org/dtds/struts-2.1.7.dtd">
<struts>
      <package name="fileupload_package" extends="struts-default">
                  <action                                    name="downfileAction_*"
class="com.fileupload.DownfileAction" method="{1}">
                  <result name="fileList">/fileupload/filedownload.jsp</result>
                  <result name="filedownload" type="stream">
                        <!--
                        contentDisposition文件下载的处理方式，包括内联(inline)和附件
```

(attachment)两种方式，

而附件方式会弹出文件保存对话框，否则浏览器会尝试直接显示文件。取值为：

attachment;filename="struts2.txt"，表示文件下载的时候保存的名字应为struts2.txt。

如果直接写filename="struts2.txt"，那么默认情况是代表inline，浏览器会尝试自动打开它，

等价于这样的写法：inline; filename="struts2.txt" 使用经过转码的文件名作为下载文件名，

fileName属性对应action类中的方法 getFileName()

downloadFile对应着action中的InputStream getDownloadFile()方法

-->

```
<param name="contentDisposition">filename="${fileName}"</param>
<param name="inputName">downloadFile</param>
</result>
</action>
<action name="download" class="com.fileupload.DownfileAction" method="download">
<result name="fileList">/fileupload/filedownload.jsp</result>
<result name="filedownload" type="stream">
```

腾科Java教学部

```
                <param name="contentType">
                 application/octet-stream;charset=GBK
                </param>
                <!--
                contentDisposition文件下载的处理方式，包括内联(inline)和附件(attachment)两种方式，
                而附件方式会弹出文件保存对话框，否则浏览器会尝试直接显示文件。取值为：
                attachment;filename="struts2.txt"，表示文件下载的时候保存的名字应为struts2.txt。
                如果直接写filename="struts2.txt"，那么默认情况是代表inline，浏览器会尝试自动打开它，
                等价于这样的写法：inline; filename="struts2.txt" 使用经过转码的文件名作为下载文件名，
                fileName属性对应action类中的方法 getFileName()

                downloadFile对应着action中的InputStream getDownloadFile()方法
                -->
                <param name="contentDisposition">attachment;filename="${fileName}"</param>
                <param name="inputName">downloadFile</param>
            </result>
```

腾科Java教学部

```
        </action>

    </package>

</struts>
```

腾科Java教学部