标题	**Struts 快速学习指南 (内部培训教材)** **-大部分素材来自于《Programming Jakarta Struts》一书**
	lzasp800（原作）
	copy form www.CSDN.net 2004-05-05
关键字	Struts MVC

1. Struts 简介

Struts 是一个技术框架，由 Craig R. McClanahan 编写，并且在 2000 年的时候捐献给了 ASF，目前，有很多组织和个人参与 Struts 框架的开发，使得 Struts 保持高速成长，同时，利用 Struts 开发的应用越来越多，使其成为 web 应用 MVC 模式中 VC 部分事实上的标准。

1.1 Web 技术历史

1.1.1 CGI

web 应用开发中历史上，**CGI(common gateway interface)**是最早使用的一种技术，通过为不同的平台，不同的 **web server** 编写插件编写应用接口，来满足通过 **web** 方式编写应用的需求。当时流行的方式包含 **NSAPI/ISAPI**，使用 **Perl** 来编写 **CGI** 程序。**CGI** 最大的问题就是线程并发的问题，当时给很多人的感觉是 **CGI** 访问速度慢，其主要原因是应用程序所编写的 **CGI** 没有考虑多线程。

1.1.2 Servlet

作为一种跨平台语言的服务器端技术，其一经产生就备受瞩目，采用 **Servlet** 开发的应用，不用考虑平台，多线程等让人头疼的问题，使得开发人员专注于业务逻辑的实现，大大解放了生产力。但是，在 **Servlet** 中嵌入 **html** 无疑是开发人员的噩梦，与同时期微软的 **ASP** 相比，**Servlet** 在开发效率方面让人不敢恭维。

1.1.3 Java Server Pages

JSP 从很大程度上时参考了 **ASP** 的想法，使得采用 **Java** 语言开发服务器端应用非常容易，同时因为 **java** 与生俱来的跨平台、安全性、易用性优势，当然，还有开发人员的高工资 **J,**使得 **JSP** 逐渐在 **Web** 服务器端应用开发中占据了主流位置。

2. Struts 安装

Struts 作为一个 **J2EE** 框架，很容易和你的 **web** 应用结合起来，你仅仅需要作以下几个步骤：

1、 下在 **Struts1.1** 二进制压缩包，将压缩包解压到**%STRUTS_HOME%**目录，目录结构如下如示：

2、 建立你的标准 **web** 应用程序，所谓标准应用程序是指在 **web** 应用程序的根目录下有一个 **WEB-INFO** 目录，**WEB-INF** 下有 **classes,lib** 目录,**classes** 下面有个 **web.xml** 文件。本文后续假设你的 **web** 应用在**%WEB_ROOT%**目录下。

3、 将**%STRUTS_HOME%/lib** 下所有文件 **copy** 到**%WEB_ROOT%/WEB-INF/lib** 下。

4、 配置**%WEB_ROOT%/WEB-INF/classes/web.xml** 以满足 **Struts** 需要，具体如下：

1、 在配置文件中映射 **ActionServlet,ActionServlet** 用于接受所有访问者的请求。在 **Struts** 应用中，所有对应用程序的请求，都会被 **WEB SERVER** 定向到 **ActionServlet** 进行统一控制、分配处理，**ActionServlet** 可以看作是 **Struts** 框架的核心，枢纽。

```
<web-app>
 <servlet>
  <servlet-name>controller</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
 </servlet>
</web-app>
```

2、 配置 **servlet** 映射，通过 **servlet** 映射可以将用户访问 **web** 应用的扩展名映射到具体处理的 **servlet**，例如，将所有以**.do** 为扩展名的页面的请求交给 **ActionServlet** 处理。

```
<web-app>
 <servlet>
  <servlet-name>controller</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>controller</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
</web-app>
```

另外，也可以采用如下方式进行映射，该方式将所有对**/action/**目录下文件的访问请求交给 **ActionServlet** 处理。

```
<web-app>
 <servlet>
  <servlet-name> controller </servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
 </servlet>
```

```
 <servlet-mapping>
  <servlet-name>controller</servlet-name>
  <url-pattern>>/action/*</url-pattern>
 </servlet-mapping>
</web-app>
```

3、 配置 **ActionServlet** 的初始化参数，**Struts1.1** 有一些指定的初始化参数，用于指明 **Struts** 应用所需要的配置文件，**debug** 等级等。

```
<web-app>
 <servlet>
  <servlet-name>controller</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
   <param-name>config</param-name>
   <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <init-param>
   <param-name>host</param-name>
   <param-value>localhost</param-value>
  </init-param>
  <init-param>
   <param-name>port</param-name>
   <param-value>7001</param-value>
  </init-param>
 </servlet>
 <servlet-mapping>
  <servlet-name> controller </servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
</web-app>
```

初始化参数利用**<init-param>**进行配置，配置采用名称-值对的方式，一个**<param-name>**对应一个**<param-value>**，初始化参数可以任意定义，例如 **host**,**port**，但是有一些在 **Struts1.1** 中是具有特别意义的，列举如下：

表 **2-1. Struts1.1** 中用到的初始化参数

参数名	含义/默认值
config	以相对路径的方式指明 Struts 应用程序的配置文件位置。如不设置，则默认值为 /WEB-INF/struts-config.xml。
config/sub1	以相对路径的方式指明子应用程序的配置文件位置，一般来说，很少用到子应用程序，在此不多描述。
debug	设置 Servlet 的 debug 级别，控制日志记录的详细程度。默认为 0，记录相对最少的日志信息。

detail	设置 Digester 的 debug 级别，Digester 是 Struts 框架所使用的用来解析 xml 配置文件的一个框架，通过该设置，可以查看不同详细等级的解析日志。默认为 0，记录相对最少的日志信息。

4、 配置标签库，标签库是 **Struts** 自带的一些组件库，采用 **JSP** 规范中 **Tag-lib** 的方式供大家使用，正是因为存在这么丰富的标签库，使得采用 **Struts** 的开发才显得这么方便，高效。

```
 <web-app>
<servlet>
 <servlet-name>controller</servlet-name>
 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
 <init-param>
  <param-name>config</param-name>
  <param-value>/WEB-INF/struts-config.xml</param-value>
 </init-param>
 <init-param>
  <param-name>host</param-name>
  <param-value>localhost</param-value>
 </init-param>
 <init-param>
  <param-name>port</param-name>
  <param-value>7001</param-value>
 </init-param>
</servlet>

<servlet-mapping>
 <servlet-name>controller</servlet-name>
 <url-pattern>*.do</url-pattern>
</servlet-mapping>

<taglib>
 <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
 <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>

<taglib>
 <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
 <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
```

```
 <taglib>
  <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
 </taglib>
</web-app>
```

标签库采用**<taglib>**定义，**<taglib>**含有两个子元素，**<taglib-uri>**和**<taglib-location>**，**<taglib-uri>**用户定义标签库的唯一表示符，可以理解为名字，以后要在 **jsp** 页面中使用这个标签库，靠的就是它。**<taglib-location>**指明标签库存在的物理路径，当然，和配置文件一样，也是相对路径。

5、 设置 **welcome** 文件列表**(**可选步骤**)**

```
<welcome-file-list>
 <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

6、 设置错误处理**(**可选步骤**)**，通常的 **http** 访问异常包含 **404 Not Found** 和 **500 Internal Error**，为了提供给用户更为友好的显示，可以做如下配置：

```
<web-app>
 <error-page>
  <error-code>404</error-code>
  <location>/common/404.jsp</location>
 </error-page>

 <error-page>
  <error-code>500</error-code>
  <location>/common/500.jsp</location>
 </error-page>
</web-app>
```

通过如上配置，当用户访问应用中不存在的页面时，将会将用户导向到**/common/404.jsp** 页面。同样地，当出现异常错误时，将会把**/common/500.jsp** 显示给用户。

7、 最后，一个完整的 **web.xml** 示例如下：

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

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
 <servlet>
  <servlet-name>storefront</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
   <param-name>config</param-name>
   <param-value>/WEB-INF/struts-config.xml</param-value>
```

```
  </init-param>
  <init-param>
   <param-name>debug</param-name>
   <param-value>3</param-value>
  </init-param>
  <init-param>
   <param-name>detail</param-name>
   <param-value>3</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
  <servlet-name>storefront</servlet-name>
  <url-pattern>/action/*</url-pattern>
 </servlet-mapping>

 <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>

 <error-page>
  <error-code>404</error-code>
  <location>/common/404.jsp</location>
 </error-page>
 <error-page>
  <error-code>500</error-code>
  <location>/common/500.jsp</location>
 </error-page>

 <taglib>
  <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
 </taglib>
 <taglib>
  <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
 </taglib>
 <taglib>
  <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
 </taglib>
</web-app>
```

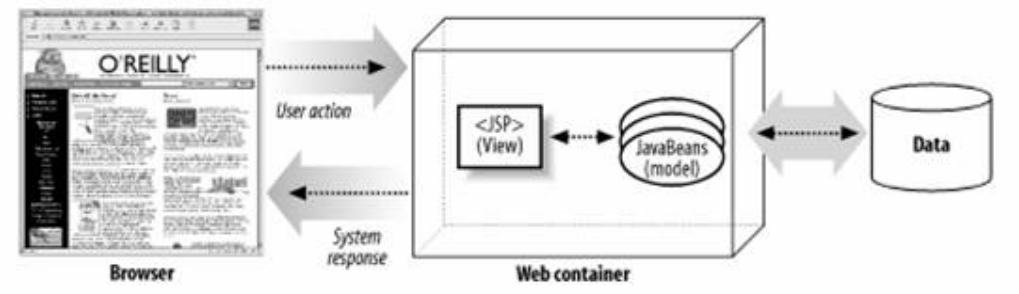
1、 到此为止，**Struts** 的开发环境安装算是告一段落。

1. Struts 框架

在介绍 Struts 框架之前，先来看看 web 开发的两种模式，这两种模式自 JSP 开发流行以来，就争论不断，它们分别是 JSP Model 1 和 JSP Model 2。

1.1 JSP Model 1

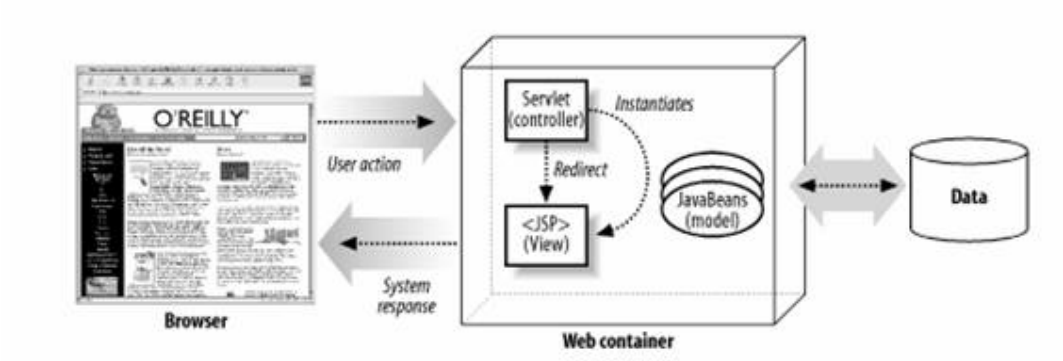
下图是 JSP Model 1 的构架示意图：



用户通过浏览器之间访问 web 应用的 JSP 页面， JSP 提供 UI 显示，JavaBeans 处理数据库访问和业务逻辑。这种开发方式最大的优势是直接、简单，对于小型应用，可以很方便、快速地进行开发。

1.2 JSP Model 2

下图是 JSP Model 2 的构架示意图：



JSP Model 2 和 JSP Model 1 最大的区别是引入了 MVC 模式的概念，即 M(Model:业务逻辑)，V(View:系统 UI),C(Controller:控制)分离，用户的所有请求提交给 Controller，由 Controller 进行统一分配，并且采用推的方式将不同的 UI 显示给用户。这样做得好处是：

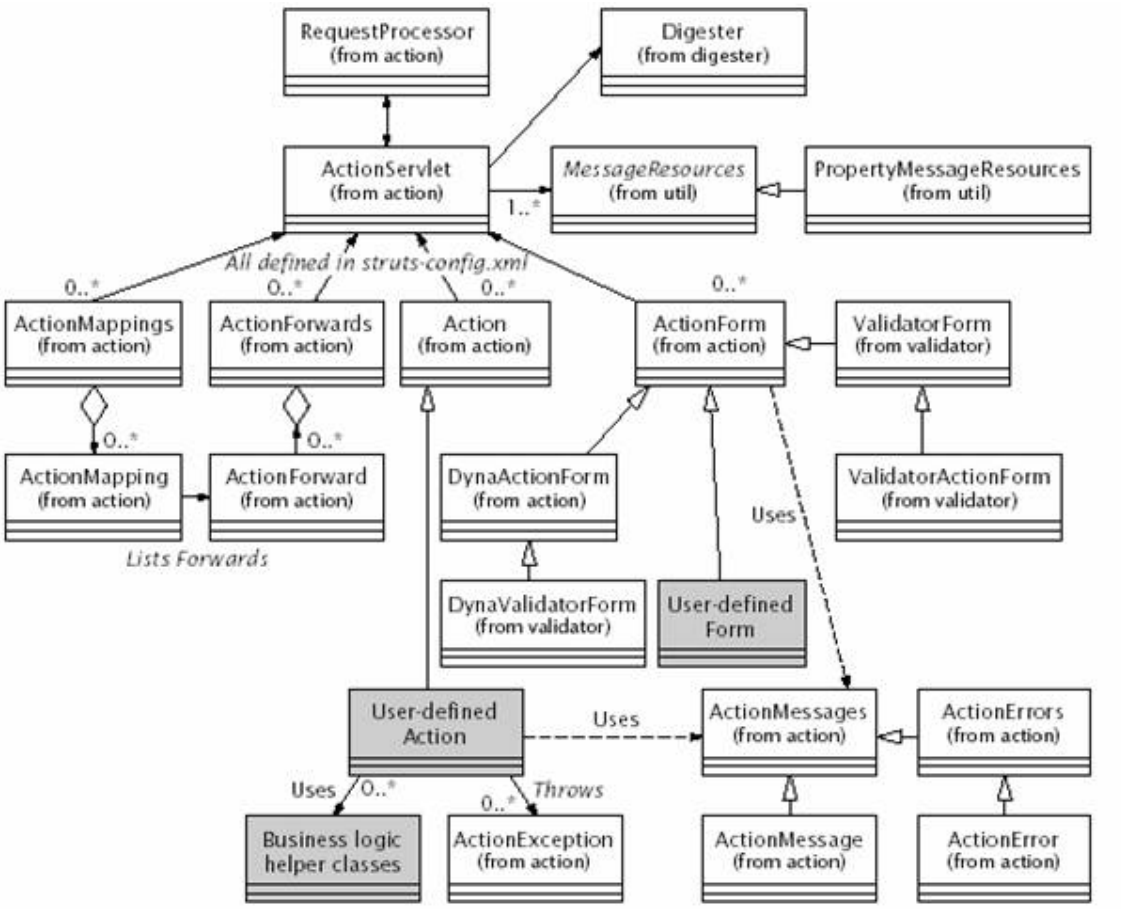
1、 可以统一控制用户的行为，例如在 Controller 中添加统一日志记录等功能是非常方便的。

2、 职责分离，有利于各部分的维护。用户不直接访问分散的 UI，这样可以通过配置文件或则流程定义的方式，在不同的环节、时间将不同的页面推向给用户。

1.3 Struts

通过了解 JSP Model 1 和 JSP Model 2，我想大家心里都已经有了选择，在这里，我不想说哪一种构架更好，在不同的环境中，使用恰到好处的技术才是最好的。普遍来说，MVC 分离是个不错的选择。

Struts 框架正是 MVC 分离的一个杰出作品。首先我们来看一下 Struts1.1 的 UML 图，以便于我们对 Struts 有个全局的了解：

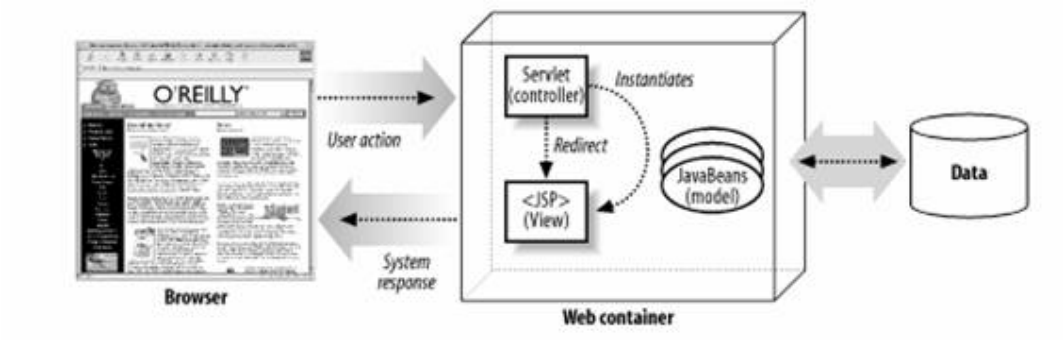


先不用急着看懂这张图，在下面的学习过程中，我们会慢慢地了解这张图中各个组件的含义。

接下来，我们从 MVC 的角度对 Struts 框架进行探索。

1.3.1 Controller

首先介绍 MVC 中的 C，上面提到了，JSP Model 1 和 JSP Model 2 最大的却别就是 C，那么在 Struts 中，这个 C 是什么呢？他是如何实现的呢？下面我们再来看看这个图：



这是 JSP Model 2 的构架图，也是 Struts 的构架图，Struts 使用一个 Servlet 作为 Controller，处理用户的请求，并分派给 Model 进行业务处理，在合适的时候将合适的 View 推向给用户。这个 Servlet 是 **org.apache.struts.action.ActionServlet** 或其子类。**ActionServlet** 类扩展自 **javax.servlet.http.HttpServlet** 类，其职责是将 http 请求提交给合适的处理器(**Processor**)进行处理。关于处理器我们在稍后会介绍，是o**rg.apache.struts.action.RequestProcessor** 或其子类的一个实例。

1.3.1.1 Controller(控制器)机制

J2EE 的前端控制器(Front Controller)设计模式中利用一个前端控制器来接受所有客户请求，为应用提供一个中心控制点，在该控制点上，可以很方便地添加一些全局性的，如加密、国际化、日志等通用操作。Controller 的实现机制正是建立在前端控制器的设计模式基础上。

前面我们介绍过，Struts 的控制器拥有一些职责，其中最主要的是以下几个：

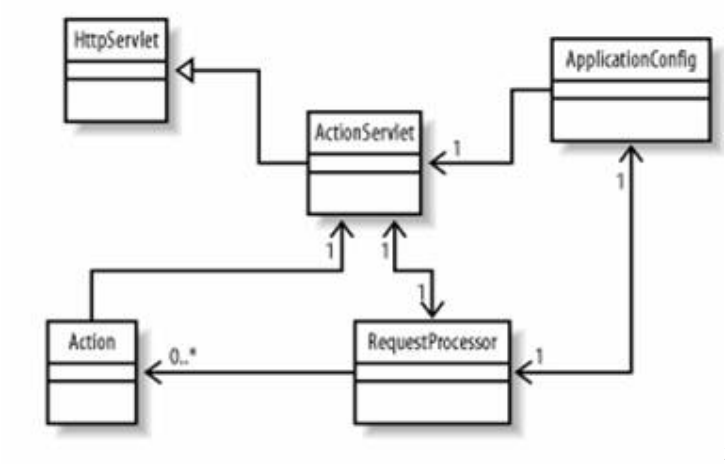
?/ΣΠAN>　　接收客户请求。

?/ΣΠAN>　　映射请求到指定的业务操作。

?/ΣΠAN>　　获取业务操作的结果并以有效的方式提供给客户。

?/ΣΠAN>　　根据业务操作的结果和当前的状态把不同的YI推向给客户。

在 Struts 框架中，控制器中不同的组件负责不同的控制职责，下图是 Struts 框架中关于控制器部分的一个组件图：



在上图中，很明显地可以看出，**ActionServlet** 处于核心位置，那么，我们就先来了解一下 **ActionServlet**。

1.3.1.2 ActionServlet 类

org.apache.struts.action.ActionServlet 在Στρυτσ应用程序中扮演接收器的角色，所有客户端的请求在被其它类处理之前都得通过 **ActionServlet** 的控制。

当 **ActionServlet** 的实例接收到一个HTTΠ请求，不管是通过γετ方法或ποστ方法，**ActionServlet** 的 **process( )**方法被调用并用以处理客户请求。**process( )**方法实现显示如下：

```
protected void process(HttpServletRequest request,HttpServletResponse response)
  throws IOException, ServletException {

  RequestUtils.selectApplication( request, getServletContext(  ) );
  getApplicationConfig( request ).getProcessor(  ).process( request, response );
}
```

该方法的实现很简单，`RequestUtils.selectApplication( request, getServletContext(  ) );`语句是用来根据用户访问的上下文路径来选择处理的应用，如果你只有一个 `Struts` 配置文件，就表示你只有一个 `Struts` 应用。关于如何建立多个 `Struts` 应用，本教程不作详细讲解，请参考相应资料。

`getApplicationConfig( request ).getProcessor(  ).process( request, response );`语句用来获取一个处理器，并将客户请求提交给处理器处理。

1.3.1.3 Struts 初始化处理流程

根据在ωεβ.ξμλ中配置的初始化参数，Σερϖλετ容器将决定在在容器的第一次启动，或第一次客户请求 **ActionServlet** 的时机加载 **ActionServlet**，不管哪种方式加载，和其它Σερϖλετ一样，**ActionServlet** 的 **init( )**方法将被调用，开始初始化过程。让我们来看看在初始化过程中将发生些什么，理解了这些，对于我们δεβυγ和扩展自己的应用更加得心应手。

1┌ 初始化框架的内部消息绑定，这些消息用来输出提示，警告，和错误信息到日志文件中。**org.apache.struts.action.ActionResources** 用来获取内部消息；

2┌ 加载*web.xml*中定义的不同参数，用以控制 **ActionServlet** 的不同行为，这些参数包括 **config**, **debug**, **detail**, ανδ **convertNull**；

3┌ 加载并初始化ωεβ.ξμλ中定义的σερϖλετ 名称和σερϖλετ映射信息。通过初始化，框架的各种ΔΤΔ被注册，ΔΤΔ用来在下一步校验配置文件的有效性；

4、 为默认应用加载并初始化Στρυτσ配置文件，配置文件即初始化参数 **config** 指定的文件。默认配置文件被解析，产生一个 **ApplicationConfig** 对象存于 **ServletContext** 中。可以通过关键字 **org.apache.struts.action.APPLICATION** 从 **ServletContext** 中获取 **ApplicationConfig**；

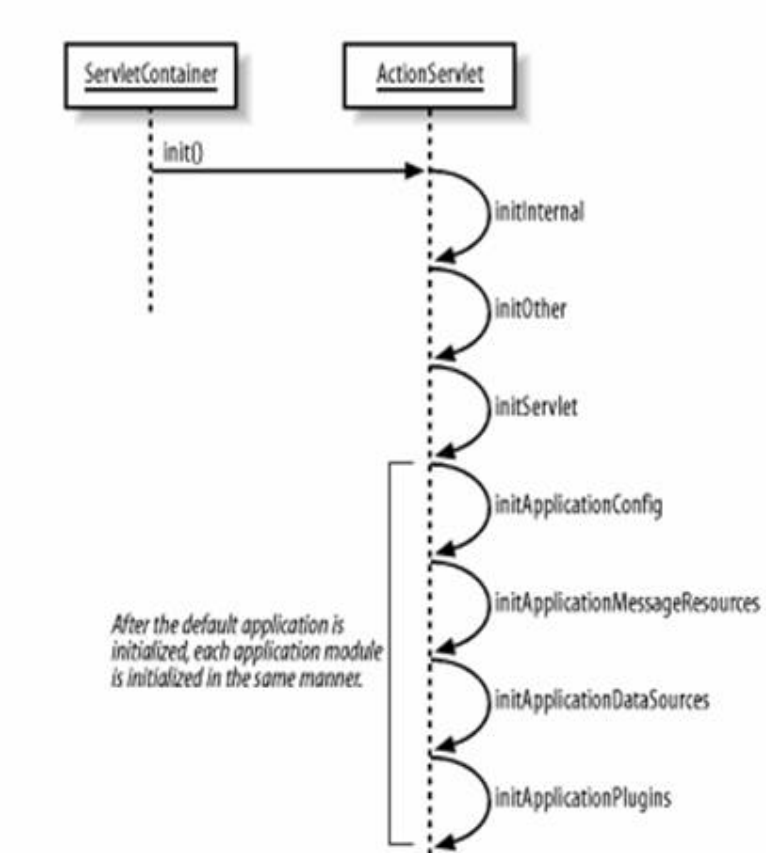
5┌ Στρυτσ配置文件中指定的每一个消息资源都被加载，初始化，并存在 **ServletContext** 的合适区域(基于每个 **message-resources** 元素的 **key** 属性)，如果 **key** 属性没有设置，则为 **org.apache.struts.action.MESSAGE**；

6┌ Στρυτσ配置文件中声明的每一个数据源被加载并且初始化，如果没有配置数据源，这一步跳过；

7┌ 加载并初始化Στρυτσ配置文件中指定的插件。每一个插件的 **init()**方法被调用；

8┌ 当默认应用加载完成，**init()**方法判断是否有应用模块需要加载，如果有，重复步骤4—7万成应用模块的加载。

下图是对上面文字说明的图形化表示：



RequestProcessor 类

前面提到过，当 **ActionServlet** 接收到客户请求后，会进行一连串的初始化操作，然后，就会将客户请求转交给合适的处理器进行处理，这个合适的处理器就是 **org.apache.struts.action.RequestProcessor** 或其子类的一个实例(根据 Struts 配置文件中的配置)。提供了默认实现，如果需要自定义这些行为，可以重载这个类定义自己的处理行为，当你想要自定义操作时，Struts 推荐你重载这个类而不是 **ActionServlet**。

下面的代码片断提供了 **RequestProcessor** 的默认行为实现代码：

```
public void process(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException {

    // Wrap multipart requests with a special wrapper
    request = processMultipart(request);

    // Identify the path component we will use to select a mapping
    String path = processPath(request, response);
    if (path == null) {
```

```
      return;
    }
    if (log.isInfoEnabled(  )) {
      log.info("Processing a '" + request.getMethod(  ) +
        "' for path '" + path + "'");
    }

    // Select a Locale for the current user if requested
    processLocale(request, response);

    // Set the content type and no-caching headers if requested
    processContent(request, response);
    processNoCache(request, response);

    // General-purpose preprocessing hook
    if (!processPreprocess(request, response)) {
      return;
    }

    // Identify the mapping for this request
    ActionMapping mapping = processMapping(request, response, path);
    if (mapping == null) {
      return;
    }

    // Check for any role required to perform this action
    if (!processRoles(request, response, mapping)) {
      return;
    }

    // Process any ActionForm bean related to this request
    ActionForm form = processActionForm(request, response, mapping);
    processPopulate(request, response, form, mapping);
    if (!processValidate(request, response, form, mapping)) {
      return;
    }

    // Process a forward or include specified by this mapping
    if (!processForward(request, response, mapping)) {
      return;
    }
    if (!processInclude(request, response, mapping)) {
      return;
    }
```

```
    // Create or acquire the Action instance to process this request
    Action action = processActionCreate(request, response, mapping);
    if (action == null) {
      return;
    }

    // Call the Action instance itself
    ActionForward forward =
      processActionPerform(request, response, action, form, mapping);

    // Process the returned ActionForward instance
    processActionForward(request, response, forward);
}
```

接下来，让我们一步一步地了解 process()方法到底做了什么。

1、 调用 **processMultipart( )**方法。如果 **HttpServletRequest** 是 POST 方式，且请求为 **multipart/form-data** ，Struts 框架将请求对象包装成处理 multipart 请求专用的请求对象，否则，只是简单地返回原有的请求对象。一般来说，除非需要处理文件上传，否则不用关心 multipart 功能的具体细节。

2、 调用 **processPath( )** 方法，该方法用来从请求 URL 中获应用取路径部分。获取到的信息在稍后的步骤中用于选择合适的 Struts **Action** 调用。

3、 调用 processLocale( ) 方法处理一些国际化的事务。

4、 调用方法来决定 **processContent( )**请求的 **content type** 编码(encoding)方式。**content type** 可以配合在配置文件中，也可以在 jsp 文件中配置，默认为 **text/html**。

5、 根据 **noCache** 属性的设置调用 **processNoCache( )** 方法，如果 **noCache** 设置为 **true**。则添加合适的响应头到响应对象中，使得页面保留在浏览器的 Cache 中。这些响应头包含 **Pragma**, **Cache-Control**, 和 **Expires** 。

6、 调用 **processPreprocess**( )方法，这个方法在这儿设置一个钩子，方法的默认实现只是简单地返回 **true**，这样给了自定义处理器的开发者提供了一个合适的地方让你添加自己的业务逻辑。因为这个方法在调用 **Action** 之前被调用，如果你重载这个方法，只需要返回 **false**，则 **Action** 就不会被调用。例如，你可以重载这个方法用户检查客户 session，如果不通过就返回 false。

7、 调用 **processMapping( )**方法，根据客户请求信息中的 path 信息来决定是否返回 **ActionMapping** 对象实例。如果不能够找到 **path** 的映射，则客户将会得到一个 **error** 响应。

8、 通过调用 **processRoles( )**方法检查是否为 **Action** 配置了安全角色。如果配置了角色要求，则请求对象的 **isUserInRole( )**方法被调用，如果用户属于这些角色，则客户会得到显示一个 error 响应。

9、 调用 **processActionForm( )**方法检查是否存在为 **ActionMapping** 配置的 **ActionForm** 。如果存在，则在有效区域内查找是否存在该 **ActionForm** 的实例，存在，则复用，不存在，则创建一个实例。然后将实例保存与再配置文件中配置好的有效区域(**request**,**session**,**application**)内，并用 **Action** 元素的 **name** 属性作为该实例的关键字。

10、 调用 **processPopulate( )**方法，如果存来存在为 **ActionMapping** 配置的 **ActionForm**，则封装请求对象中的数据到 **ActionForm** 中，在进行封装之前，先调用 **ActionForm** 的 **reset( )**方法进行属性值的默认化。

11、 调用 processValidate( )方法。如果 ActionForm 被配置好，并且 action 元素的属性 validate 被设置为 true ，则进一步调用 validate( )方法进行规则校验。如果 validate( )方法校验失败，就会保存一个 ActionErrors 对象到请求区域中，请求将会自动重定向到 action 映射的 input 属性所指定的页面中。如果校验通过或在 action 映射中没有配置 ActionForm，则继续处理请求。

12、 根据 action 映射是否配置了 **forward** 属性或 **include** 属性来决定下一步操作。如果配置了任意一个，则相应地调用 **RequestDispatcher** 对象的 **forward( )**方法或 **include( )**方法，调用后，对客户请求的处理结束。否则，继续处理请求。

13、 调用 **processActionCreate( )**方法创建或获取一个 **Action** 对象实例处理请求。**processActionCreate( )**方法会在缓存中查找是否存在已经创建好的 **Action** 实例，如果存在，则复用，否则，则重新创建并将其村于缓存中。

14、 调用 **processActionPerform( )**方法，该方法用于在一个 try/catch 代码块中调用 action 实例的 **execute( )**方法，这样确保 **action** 的 **execute( )**方法一旦发生执行异常能够被 **RequestProcessor** 捕获。

15、 调用 **processActionForward( )**方法，并传入 action 的 **execute( )**方法所返回的 **ActionForward** 对象实例，方法通过检查 **ActionForward** 对象实例，决定采用 **redirect** 或 **forword** 方式进行重定向。究竟采用 **redirect** 还是 **forword** 取决于 **forward** 元素的 **redirect** 属性值。

扩展 RequestProcessor

如果不想利用 Struts 提供的处理器，则可以扩展它。通过两个步骤即可实现：

1、 创建一个新的类，该类必须是 **org.apache.struts.action.RequestProcessor** 的子类；

2、 在 Struts 配置文件中进行声明，例如：(**粗体**部分为你的自定义处理器类)

```
<controller
  contentType="text/html;charset=UTF-8"
  debug="3"
  locale="true"
  nocache="true"
  processorClass="com.struts.framework.CustomRequestProcessor"/>
```

1.1.1.1 Action 类

如果说 **ActionServlet** 是 **Struts** 框架的入口，**RequestProcessor** 是消化过滤系统，则 **org.apache.struts.action.Action** 类可以说是整个框架的心脏。他是客户请求和业务操作的连接桥，也可以将其看作是业务操作的客户代理。

在前面对 **ReqeustProcessor** 类的学习中，我们了解到一旦确定并得到了一个 **action** 实例，**ReqeustProcessor** 会调用 **action** 的 **execute()** 方法处理客户请求，你需要扩展 **action** 类，并实现它的 **execute()** 方法，在此方法中添加你自己的处理代码。下面给出是一个示例，这个 **action** 用来处理用户的登录请求：

```
package com.oreilly.struts.storefront.security;


import java.util.Locale;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import com.oreilly.struts.storefront.customer.view.UserView;
import com.oreilly.struts.storefront.framework.exceptions.BaseException;
import com.oreilly.struts.storefront.framework.UserContainer;
import com.oreilly.struts.storefront.framework.StorefrontBaseAction;
import com.oreilly.struts.storefront.framework.util.IConstants;
import com.oreilly.struts.storefront.service.IStorefrontService;
```

```
/**
 * Implements the logic to authenticate a user for the Storefront application.
 */
public class LoginAction extends StorefrontBaseAction {
  /**
   * Called by the controller when the user attempts to log in to the
   * Storefront application.
   */
  public ActionForward execute( ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response )
  throws Exception{

    // The email and password should have already been validated by the ActionForm
    String email = ((LoginForm)form).getEmail(  );
    String password = ((LoginForm)form).getPassword(  );

    // Log in through the security service
    IStorefrontService serviceImpl = getStorefrontService(  );
    UserView userView = serviceImpl.authenticate(email, password);

    // Create a single container object to store user data
    UserContainer existingContainer = null;
    HttpSession session = request.getSession(false);
    if ( session != null ){
      existingContainer = getUserContainer(request);
      session.invalidate(  );
    }else{
```

```
      existingContainer = new UserContainer(  );
    }

    // Create a new session for the user
    session = request.getSession(true);

    // Store the UserView in the container and store the container in the session
    existingContainer.setUserView(userView);
    session.setAttribute(IConstants.USER_CONTAINER_KEY, existingContainer);

    // Return a Success forward
    return mapping.findForward(IConstants.SUCCESS_KEY);
  }
}
```

1.1.1.1.1 Action 类缓冲

Action 类被设计为线程安全的，在每个应用中每个 **Action** 类只会被实例化一次，供所有线程共享。**RequestProcessor** 利用一个 **HashMap** 用来保存 **Action** 实例。

思考题？

所有线程共享一个 **Action** 类实例意味着什么？我们在编程中需要注意些什么呢？

1.1.1.2 ActionForward 类

从前面的介绍我们已经了解到，**Action** 的 **execute( )** 方法返回一个 **ActionForward** 对象。**ActionForward** 对象是 JSP 页面、Java servlet 等 web 资源的抽象表现。

ActionForward 的用途是为了减少应用和物理资源(JSP 页面，Java servlet)的耦合，物理资源只需要在配置文件中指定(利用 `name`,`path` 属性和 `forward` 元素的 `redirect` 属性)，而不是在代码中指定。`RequestDispatcher` 利用 `ActionForward` 来执行重定向操作。

要在 `Action` 中返回一个 `ActionForward` 对象，你可以动态地创建一个 `ActionForward` 对象，不过更为通用的解决方案是，通过在 Struts 配置文件中进行 action 映射，然后通过关键字去查找一个 `ActionForward` 。下面是代码示例：

```
    return mapping.findForward( "Success" );
```

上面的代码中，"Success"作为参数被传递到 ActionMapping 的 findFoward( )方法中，findFoward( )方法在 Struts 配置文件的 global-forwards 区域，以及被调用的 action 的 forward 元素中查找名字和"Success"相匹配的元素。下面是 action 元素中的 forward 示例：

```
<action
   input="/security/signin.jsp"
   name="loginForm"
   path="/signin"
   scope="request"
   type="com.oreilly.struts.storefront.security.LoginAction"
   validate="true">
   <forward name="Success" path="/index.jsp" redirect="true"/>
   <forward name="Failure" path="/security/signin.jsp" redirect="true"/>
 </action>
```

1.1.1.1 Action 和业务逻辑

思考题?

Action 属于 MVC 中的 Controller 还是 Model？为什么？

1.1.1.2 使用 Struts 内置的 Action

Struts1.1 框架的 **org.apache.struts.actions** 包中包含了 5 个内置的 Action，用来执行一些通用的操作，你可以把它们用在你的项目中，以节省你的开发时间。接下来我们分别介绍这 5 个内置的 Action。

1.1.1.2.1 org.apache.struts.actions.ForwardAction 类

很多情况下，你仅仅需要引导客户从一个 JSP 页面跳转到另外一个 JSP 页面，按照我们通常的做法，可以做一个链接让用户直接访问要跳转到的页面。但是 MVC 模式不推荐你这么做，因为，**Controller** 的职责就是接收所有的客户请求，然后将客户请求提交给一个合适的模块进行处理，并将合适的 UI 推给用户，如果直接方式 JSP 页面，则跳过了 **Controller** 的控制，则无法享受 **Controller** 所提供的优点。为了解决这个问题，并且不用你去为了执行一个简单的重定向操作而创建一个 **Action** 类 ，Struts 框架提供了 **ForwardAction** 类，这个 **Action** 只是简单地执行一个重定向操作，重定向的目的地通过 **parameter** 属性配置。要使用 **ForwardAction** 类，只需要在 Struts 配置文件中将 **Action** 的 **type** 属性配置为 **org.apache.struts.actions.ForwardAction**:

```
<action
  input="/index.jsp"
  name="loginForm"
  path="/viewsignin"
  parameter="/security/signin.jsp"
  scope="request"
  type="org.apache.struts.actions.ForwardAction"
```

```
  validate="false"/>
</action>
```

当你访问**/viewsignin** 的时候，就会自动重定向到**/security/signin.jsp**。

1.1.1.2.2 org.apache.struts.actions.IncludeAction 类

暂略

1.1.1.2.3 org.apache.struts.actions.DispatchAction 类

暂略

1.1.1.2.4 org.apache.struts.actions.LookupDispatchAction 类

暂略

1.1.1.2.5 org.apache.struts.actions.SwitchAction 类

暂略

1.1.2 Model

Struts 没有定义具体的 Model 层的实现，Model 层通常是和业务逻辑紧密相关的，还通常有持续化的要求，Struts 目前没有考虑到这一层，但是，不管在开源世界还是商业领域，都有一些都别优秀的工具可以为 Model 层次的开发提供便利，例如优秀的 O/R Mapping 开源框架 Hibernate。

1.1.3 View

通常，Web 应用的 UI 由以下文件组成：

- HTML
- JSP

而 JSP 中通常包含以下组件：

- 自定义标签
- DTO(Data Transfer Object 数据传输对象)

在 Struts 中，还包含了以下两种常用的组件：

- Struts ActionForms
- 资源绑定(java resource bundles)，例如将标签的显示内容，错误提示的内容通过配置文件来配置，这样可以为实现国际化提供基础。

由此可见，Struts 对于传统的 Web UI 所作的扩充就是 Struts ActionForms 和资源绑定，接下来对其进行进一步描述。

1.1.3.1 使用 Struts ActionForm

在 Struts 框架中,**ActionForm** 负责在用户和业务逻辑层之间来回地传递用户输入的数据。框架会自动收集用户输入并以 **form bean** 的方式将这些数据传递给 **Action**，然后，**form bean** 可以被传递到业务层。不过，为了减少表示层和业务层的耦合，不建议将 **ActionForm** 直接传递给业务层，而建议代之为 **DTO**。即在 **Action** 中利用 **form bean** 的数据创建合适的 **DTO**，然后传递给业务层。下面的步骤描述了 **Struts** 框架在每一次请求中，是如何处理 **ActionForm** 的：

1、 检查是否已经配置 **ActionForm** 映射到 **Action**；

2、 如果某一个 **ActionForm** 被映射到 **Action**，利用配置文件中 **action** 元素的 **name** 属性查找相匹配的 **ActionForm** 配置信息；

3、 检查是否已经存在该 **ActionForm** 的实例**(instance);**

4、 如果存在该 **ActionForm** 的实例，并且符合当前请求的需要，则重用这个实例；

5、 否则，创建该 **ActionForm** 的实例，并且将其保存在合适的生存区域中**(**生存区域 **(scope)** 的设置请查看 **action** 元素，**scope** 表示该实例的生存期限**,**一般来说**,**有 **request,session,application** 等几种**)**；

6、 调用 **ActionForm** 实例的 **reset()**方法；

7、 遍历请求参数，根据不同的参数名，调用 **ActionForm** 实例和参数名相对应的 **setter** 方法，设置参数值到 **ActionForm** 实例中；

8、 最后，如果 **validate** 属性设置为 **true**，则调用 **ActionForm** 实例的 **validate()**方法，该方法可以返回任何错误，主要为校验错误。

对于每一个需要传递 form 数据的 HTML 页面，必须使用一个 **ActionForm**，同一个 **ActionForm** 可以被多个不同的页面使用，只要 **HTMLFORM** 域和 **ActionForm** 的属性相匹配即可。

下面是一个 **ActionForm** 的示例：

```
package com.oreilly.struts.banking.form;


import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.Action;
```

```
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.util.MessageResources;
/**
 * This ActionForm is used by the online banking appliation to validate
 * that the user has entered an accessNumber and a pinNumber. If one or
 * both of the fields are empty when validate( ) is called by the
 * ActionServlet, error messages are created.
 */
public class LoginForm extends ActionForm {
  // The user's private ID number
  private String pinNumber;
  // The user's access number
  private String accessNumber;

  public LoginForm(  ) {
    super(  );
    resetFields(  );
  }
  /**
   * Called by the framework to validate the user has entered values in the
   * accessNumber and pinNumber fields.
   */
  public ActionErrors validate(ActionMapping mapping, HttpServletRequest req ){
    ActionErrors errors = new ActionErrors(  );

    // Get access to the message resources for this application.
```

```
    // There's no easy way to access the resources from an ActionForm.
    MessageResources resources =
      (MessageResources)req.getAttribute( Action.MESSAGES_KEY );

    // Check and see if the access number is missing.
    if(accessNumber == null || accessNumber.length(  ) == 0) {
      String accessNumberLabel = resources.getMessage( "label.accessnumber" );
      ActionError newError =
        new ActionError("global.error.login.requiredfield", accessNumberLabel );
      errors.add(ActionErrors.GLOBAL_ERROR, newError);
    }

    // Check and see if the pin number is missing.
    if(pinNumber == null || pinNumber.length(  ) == 0) {
      String pinNumberLabel = resources.getMessage( "label.pinnumber" );
      ActionError newError =
        new ActionError("global.error.login.requiredfield", pinNumberLabel );
      errors.add(ActionErrors.GLOBAL_ERROR, newError);
    }
    // Return the ActionErrors, if any.
    return errors;
  }

  /**
   * Called by the framework to reset the fields back to their default values.
   */
  public void reset(ActionMapping mapping, HttpServletRequest request) {
    // Clear out the accessNumber and pinNumber fields.
    resetFields(  );
```

```
  }
  /**
   * Reset the fields back to their defaults.
   */
  protected void resetFields(   ) {
    this.accessNumber = "";
    this.pinNumber = "";
  }

  public void setAccessNumber(String nbr) {
    this.accessNumber = nbr;
  }

  public String getAccessNumber(   ) {
    return this.accessNumber;
  }

  public String getPinNumber(   ) {
    return this.pinNumber;
  }
  public void setPinNumber(String nbr) {
    this.pinNumber = nbr;
  }
}
```

Struts 框架提供的 ActionForm 实现了一些方法，到现在为止，最重要的两个方法是 reset()和 validator()：

```
public void reset( ActionMapping mapping, HttpServletRequest request );
public ActionErrors validate( ActionMapping mapping, HttpServletRequest request );
```

ActionForm 对于这两个方法的默认实现是不执行任何操作，你可以重载这两个方法来执行具体的逻辑。

当写好了 ActionForm 类之后，需要通知 Struts 应用程序它的存在，以及 action 和 ActionForm 之间的映射关系。在 Struts 中，是通过配置文件来实现这一目的的。第一步，将应用所需要用到的所有 ActionForm 在配置文件的 form-beans 这一段加以描述，下面的片断描述了如何通知 Struts 应用程序这三个 ActionForm 的存在。

```
<form-beans>
 <form-bean
   name="loginForm"
   type="com.oreilly.struts.banking.form.LoginForm"/>
 <form-bean
   name="accountInformationForm"
   type="org.apache.struts.action.DynaActionForm">
   <form-property name="accounts" type="java.util.ArrayList"/>
 </form-bean>
 <form-bean
   name="accountDetailForm"
   type="org.apache.struts.action.DynaActionForm">
   <form-property
     name="view"
     type="com.oreilly.struts.banking.view.AccountDetailView"/>
 </form-bean>
</form-beans>
```

每一个 form bean 的 **name** 属性必须唯一，**type** 属性定义了该 form bean 的类，该类必须实现 Struts **ActionForm** 类。下一步就是建立 **action** 和 **form-bean** 的联系，**form-bean** 可以和一到多个 **action** 建立联系，通过在 **action** 元素的属性中引用 **form-bean** 的 **name** 即可完成映射，下面的片断显示了 **LoginAction** 和 **form-bean** 的映射，我们之前已经看过这个片段：

```
<action
  path="/login"
  type="com.oreilly.struts.banking.action.LoginAction"
  scope="request"
  name="loginForm"
  validate="true"
  input="/login.jsp">
       <forward       name="Success"       path="/action/getaccountinformation"
redirect="true"/>
  <forward name="Failure" path="/login.jsp" redirect="true"/>
</action>
```

在 Struts1.1 中，添加了一种新类型的 action form，叫做 org.apache.struts.action.DynaActionForm，这种类型的 action form 可以配置为 action 的映射，它会自动处理 HTML form 中的数据并将其传递到 Action。DynaActionForm 如何做到自动处理 HTML form 数据的呢？DynaActionForm 内部使用一个 Map 来存放 HTML field 数据。

在接下来的一节中，我们详细了解一下 DynaActionForm。

1.1.1.1 使用 DynaActionForm

从上一节的介绍，我们可以看出，使用 ActionForm 和我们自己来编写类获取 HTML from 值，在进行处理相比，有不少优势。ActionForm 所封装的数据和行为时几乎每一个 web 应用程序都需要的，而且在一个应用中会多次用到，例如一个信息实体的增加和修改，可能从不同的角度，不同的页面实现信息实体的增、改，通过 ActionForm 就可以复用，复用可以统一规则，减少开发时间和维护工作量。但是，现在对 ActionForm 的使用越来越少，为什么呢？

第一，　也是一个最大的问题，会使得项目中存在很多 ActionForm 类，增加了整个项目类的数目和维护复杂度，有的开发人员为了避开这个问题，使用一个很大的，包含所有 HTML from 属性的 ActionForm 来和所有 action 映射，这种方式我认为问题更多，完全失去了封装的味道。

第二，　当需要添加或者删除一个 HTML from 属性时，如果 **ActionForm** 需要用到这些属性，就得修改 **ActionForm**，并且要重新编译。

基于这些原因，在 Struts1.1 框架中，添加了一种新类型的 **ActionForm**，这种 **ActionForm** 可以动态变化从而避免创建具体的 **ActionForm** 类。这种 **ActionForm** 的基类是 **org.apache.struts.action.DynaActionForm**，当然，**DynaActionForm** 是从 **ActionForm** 扩展而来的。对于应用来说，**DynaActionForm** 和 **ActionForm** 在以下三个方面会有些不同：

- **ActionForm** 的属性定义
- **validate()**方法
- **reset()**方法

DynaActionForm 的属性不同于 **ActionForm**，**ActionForm** 的属性是通过具体的类，具体的 **setter,getter** 方法来进行属性值的设置，获取，而 **DynaActionForm** 是在 **Struts** 的配置文件中定义的。

对 **reset()**方法调用的时机和 **ActionForm** 并无不同，只是相对来说，在 **reset()**方法被调用的时候，你拥有比较少的控制权。当然，可以通过扩展 **DynaActionForm**，重载 **reset()**方法。

对于从 UI 端输入的数据的校验，相对来说有些复杂，它是通过 Struts Validator 组件来实现的，稍后会详细介绍它。

1.1.1.1.1 配置 DynaActionForm

要使用 DynaActionForm，首先得在 Struts 配置文件中添加 form-bean 元素。在配置文件中，DynaActionForm 和 ActionForm 的不同之处在于，DynaActionForm 需要添加一些 form-property 元素，form-property 用来指定 HTML form 中的 field 名字，Struts 框架会通过这些名字的匹配，自动将 HTML form 各个 field 的值封装到 DynaActionForm 实例中。下面的片断是关于 DynaActionForm 的配置文件示例 ：

```
<form-beans>
  <form-bean
    name="loginForm"
    type="org.apache.struts.action.DynaActionForm">

    <!–在下面制定 Form 的属性 -->
    <form-property
      name="email"
      type="java.lang.String "/>
    <form-property
      name="password"
      type="java.lang.String "/>

    <!–可以为属性设置默认值 -->
    <form-property
      initial="false"
      name="rememberMe"
      type="java.lang.Boolean "/>
  </form-bean>
<form-beans>
```

当你 HTML form 中添加了一个属性，需要在 DynaActionForm 中添加一个属性时，就不需要去修改具体的 ActionForm 类，只需要在配置文件中添加一个 form-property 元素即可，大大提高了可扩展能力。

前面我们已经了解到，ActionForm 的 reset()方法默认不进行任何操作，在 DynaActionForm 中，reset()方法默认将所有属性设置为默认值，如果在配置文件中没有为该属性设置默认值，将会按照

java 编程语言的规范根据属性的类型为其进行初始化，例如：数字(int ,double,float)的将会初始化为 0，Ojbect 类型将为初始化为 null。

> 注意：在配置文件中定义的 form-property 的 type 属性，其值为一个 java 类名，因此对于 java 语言中的主类型，如 ini,long 必须定义为 java.lang.Int，java.lang.Long，其它主类型依次类推。

1.1.1.1.1 使用 DynaActionForm 执行校验规则

同 ActionForm 一样，DynaActionForm 也没有提供 validate()方法的默认操作，幸运的是，Struts 提供了另外一种框架来帮助大家解决校验的问题，这就是 Struts Validator 框架。Struts Validator 框架由 David Winterfeldt 编写，现在已经成为 Struts 主分发的一部分，它提供基于规则的校验，对于常用的校验，如必填项目，email，电话等等提供了现成的规则，只需要通过配置文件进行配置即可。关于 Struts Validator 框架的详细介绍，见后续章节。

1.1.1.2 Validator 框架

Struts 允许在 ActionForm 的 validator()方法中添加校验代码，对用户输入的数据进行规则校验，这能很好地工作，但是，这存在一些限制，举个简单的例子，一个必填项现在不是必填项了，要满足这个简单的需求就需要更改 ActionForm 类的 validator()方法，再进行重新编译，很麻烦。本节我们来学习 Validator 框架，看它是如何解决校验问题的。

1.1.1.2.1 Validator 的安装、配置

Validator 目前是 Jakarta Commons 项目的一部分，它也被包含在 Struts 主分发里面，可以直接使用 Struts 中自带的 Validator 库，也可以去网站上下载 http://jakarta.apache.org/commons/。Validator 需要一些其它库的支持，例如 Jakarta ORO，不过没关系，Struts 分发包里面都包含了，你只需要按照第 2 章介绍的将 Struts 安装到你的应用中，就一切 ok。

1.1.1.2.1.1 配置校验规则

前面提到过，Validator 是通过校验规则来实施校验，这些校验规则被配置在配置文件中，这意味着不需要修改源代码就可以方便地更改校验规则，是不是感觉很不错？Validator 所需要的配置文件有两个：*validation-rules.xml* 和 *validation.xml*。

1.1.1.2.1.2 validation-rules.xml

在 *validation-rules.xml* 文件中配置了一些全局性的校验规则，使得你在应用程序中使用校验而不用关注实现细节。这个配置文件是 **Validator** 框架自带的，可以用在所有 Struts 应用中。它默认配置了许多很常用的规则，一般来说，不用去更改它，除非需要扩展或修改这些默认的校验规则。

> 建议：即使你需要扩展一些规则，也不要去修改 *validation-rules.xml*，而是通过新的配置文件去定义你所扩展的校验规则。

validator-rules_1_1.dtd 定义了 *validation-rules.xml* 文件的结构，根元素是 `form-validation`，它包含一到多个 `global` 元素，`global` 元素包含一到多个 `validator` 元素。

每一个 `validator` 元素定义了一个唯一的校验规则。下面是 *validation-rules.xml* 文件中的一个片断，用来定义必填项(`required`)校验规则：

```
<validator
  name="required"
  classname="org.apache.struts.util.StrutsValidator"
  method="validateRequired"
  methodParams="java.lang.Object,
    org.apache.commons.validator.ValidatorAction,
    org.apache.commons.validator.Field,
    org.apache.struts.action.ActionErrors,
    javax.servlet.http.HttpServletRequest"
  msg="errors.required">
</validator>
```

下表详细介绍了 `validator` 元素每个属性的具体含义：

序号	属性	解释
1.	`name`	赋予校验规则一个唯一的名称，便于在 *validation-rules.xml* 文件和应用指定的其它校验文件中引用。
2.	`classname`	指定含有具体校验规则 **Java Class** 名，`org.apache.struts.util.StrutsValidator` 是 `Validator` 框架自带的一个 `Java Class`，它实现了一些很常用的校验规则。

3.	method	指定含有具体校验规则 **Java Class** 的具体方法，一个校验规则有实现校验的 **Java Class** 的一个方法来实现。
4.	methodParams	声明 method 属性所指定的方法的参数，参数之间用逗号分隔。
5.	msg	msg 是用来指定当校验不通过时，Validator 框架所给出的提示信息。它的值是应用所配置的资源文件中的一个关键字，当校验失败时，Validator 框架利用 msg 所指定的值到应用配置的资源文件中去查找匹配记录。Validator 框架默认使用以下提示信息： errors.required={0} is required. errors.minlength={0} cannot be less than {1} characters. errors.maxlength={0} cannot be greater than {1} characters. errors.invalid={0} is invalid. errors.byte={0} must be a byte. errors.short={0} must be a short. errors.integer={0} must be an integer. errors.long={0} must be a long. errors.float={0} must be a float. errors.double={0} must be a double. errors.date={0} is not a date. errors.range={0} is not in the range {1} through {2}. errors.creditcard={0} is not a valid credit card number. errors.email={0} is an invalid email address 可以将上面的这些信息添加到你的 Struts 应用所配置的资源文件(例如：ApplicationResources.properties)中，也可以修改这些值之后，将其添加到配置文件中，示例如下： errors.required={0} 是必填项。
6.	depends	depends 指定在本校验规则的前置校验规则，下面的片断定义了一个最小长度的校验规则，含义是在进行最小长度校验之前，会先调用 **required** 校验规则确保数据不为空： <validator name="minLength" classname="org.apache.struts.util.StrutsValidator" method="validateMinLength" methodParams="java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest" **depends="required"**

		msg="errors.minlength"> </validator> 如果存在多个前置校验规则，则可以用以下的方式进行声明，各校验规则之间用逗号分隔： **depends="required,integer"** 如果前置校验规则失败，则后续的校验规则不会被执行。
7.	**jsFunctionName**	可选属性。用来指定 **JavaScript** 函数的名字。 **The final attribute supported by the validator element is the jsFunctionName attribute. This optional attribute allows you to specify the name of the JavaScript function. By default, the Validator action name is used.**

前面已经介绍了，org.apache.struts.util.StrutsValidator 是 Validator 框架自带的一个校验规则类，其实现了一些常用的校验规则，其包含的校验方法(method)如下所列：

- **validateByte** 检查值能够安全地转换为 **byte**
- **validateCreditCard** 检查值是一个有效的信用卡号码
- **validateDate** 检查值是一个有效的日期
- **validateDouble** 检查值能够安全地转换为 **double**
- **validateEmail** 检查值是一个有效的 **Email** 地址
- **validateFloat** 检查值能够安全地转换为 **double**
- **validateInteger** 检查值能够安全地转换为 **int**
- **validateLong** 检查值能够安全地转换为 **long**
- **validateMask** 检查值符合掩码规则，掩码采用规则表达式的方式
- **validateMinLength** 检查值的长度大于等于指定长度
- **validateMaxLength** 检查值的长度小于指定长度

- **validateRange** 检查值的有效范围在指定范围内
- **validateRequired** 检查值不为 **null** 或长度**>0**
- **validateShort** 检查值能够安全地转换为 **short**

.1.1.1.1.1 validation.xml

Validator 框架所需要的第二个配置文件是 *validation.xml*，这个配置文件是具体应用(项目)所特定的，可以根据你的应用(项目)情况进行自定义配置。它描述了具体的 **ActionForm** 使用 *validation-rules.xml* 文件中的哪个校验规则进行校验。

validation_1_1.dtd 定义了 *validation.xml* 的结构，根元素为 **form-validation**，其包含 0 到多个 **global** 元素和一到多个 **formset** 元素：

```
<!ELEMENT form-validation (global*, formset+)>
```

global 元素包含 0 到多个 constant 子元素：

```
<!ELEMENT global (constant*)>
```

constant 子元素和 Java 里面常量的含义是一样的，下面的片断定义了两个常量：

```
<global>
 <constant>
  <constant-name>phone</constant-name>
  <constant-value>^\(?(\d{3})\)?[-| ]?(\d{3})[-| ]?(\d{4})$</constant-value>
 </constant>
 <constant>
  <constant-name>zip</constant-name>
  <constant-value>^\d{5}(-\d{4})?$</constant-value>
 </constant>
</global>
```

上面的片断包含了两个常量，**phone** 和 **zip**，这些常量在所有 **formset** 元素中有效，在 **formset** 中通过名称引用这些常量。

下面的片断展示了一个简单的 *validation.xml* 文件说明：

代码片断 3.3.3.3.1.3.1

```
<form-validation>
 <global>
  <constant>
   <constant-name>phone</constant-name>
   <constant-value>^\(?(\d{3})\)?[-| ]?(\d{3})[-| ]?(\d{4})$</constant-value>
  </constant>
 </global>
 <formset>
  <form name="checkoutForm">
   <field
     property="phone"
     depends="required,mask">
     <arg0 key="registrationForm.firstname.displayname"/>
     <var>
       <var-name>mask</var-name>
       <var-value>${phone}</var-value>
     </var>
    </field>
   </form>
  </formset>
</form-validation>
```

在上面的代码片断中，var 元素应用了在 global 中定义了 phone 常量，用来配合对 **phone** 属性的校验。

formset 元素可以包含两个子元素，constant 和 form。constant 元素和 global 区域定义的 constant 元素格式和用途一样，只不过作用范围不同，在 formset 中定义的 constant 元素其作用范围只限于该 formset 覆盖区域。Formset 元素中的 form 元素至少要出现一次。**DTD** 描述如下：

```
<!ELEMENT formset (constant*, form+)>
```

form 元素定义了需要进行校验的域，其 **name** 属性对应应用中分配给 form 的标识，在 Struts 框架中，就是在 Struts 配置文件中 **form-beans** 区域定义的 ActionForm 的 **name** 属性。

下面是 **form** 元素的 DTD 定义：

```
<!ELEMENT form (field+)>
```

field 元素指明了 JavaBean 中需要被校验的属性。在上面的代码片断中，在 Struts 中，ActionForm 就是这个需要被校验的 JavaBean。在代码片断 3.3.3.3.1.3.1 中，定义了对 Struts 配置文件中名称为

checkoutForm 的 ActionForm 所拥有的名称为 phone 的属性的校验说明，表示 checkoutForm 的 phone 属性为必填项而且符合**${phone}**所定义的正则表达式的掩码规则。field 元素的属性在下表中具体描述：

属性	描述
property	JavaBean(在 Struts 为 **ActionForm**)中需要被校验的属性的名称。
depends	应用于 **property** 指定属性的校验规则列表，多个校验规则之间用逗号分隔。
page	这个属性在应用于“向导”模式的 form 中，用来确保不会跳页访问。
indexedListProperty	不会用

表 3.3.3.3.1.3.1 **field** 元素的属性列表

field 元素包含以下几个子元素，DTD 定义如下：

```
<!ELEMENT field (msg?, arg0?, arg1?, arg2?, arg3?, var*)>
```

msg 子元素允许你为该 field 指定一个提示信息，校验规则将会使用这个指定的提示信息替代规则默认的提示信息，msg 子元素的值必须是应用资源文件的一个关键字(key)。例如：

```
<field property="phone" depends="required,mask">
  <msg name="mask" key="phone.invalidformat"/>
  <arg0 key="registrationForm.firstname.displayname"/>
  <var>
    <var-name>mask</var-name>
    <var-value>${phone}</var-value>
  </var>
</field>
```

msg 子元素支持三个属性，DTD 定义如下：

```
<!ATTLIST msg name     CDATA #IMPLIED
              key      CDATA #IMPLIED
              resource CDATA #IMPLIED >
```

name 属性指定了 **msg** 将使用的校验规则名称，属性值必须是在 *validation-rules.xml* 文件中定义的校验规则。

key 属性指定了一个资源文件的关键字，当校验失败是，该关键字所代表的信息将会添加到 **ActionError** 中。如果你想设置一个明确的文本而不是资源文件的关键字，则可以将 **resource** 属性设置位 **false**，这种情况下，可以将 **key** 属性设置为一个明确的文本，如“电话格式不正确**!**”。

field 元素可以包含至多四个额外的子元素，它们分别命名为 **arg0**, **arg1**, **arg2** 和 **arg3**，它们用来向提示信息中添加额外的信息，**arg0** 定义了第一个可替换的值，**arg1** 定义了第二个可替换的值，以此类推。每个 **arg** 元素支持三个属性，**name**, **key** 和 **resource**，其含义和之前我们看到的 msg 元素的同名属性含义一致。下面是一段应用 **arg** 元素的示例：

```
<field property="phone" depends="required,mask,minLength">
  <arg0 key="registrationForm.firstname.displayname"/>
  <arg1 name="minlength" key="${var:minLength}" resource="false"/>
  <var>
   <var-name>mask</var-name>
   <var-value>${phone}</var-value>
  </var>
  <var>
   <var-name>minLength</var-name>
   <var-value>5</var-value>
  </var>
</field>
```

field 元素包含的最后一个子元素是 var 元素，**field** 元素可以包含 0 到多个 var 元素，var 元素可以设置该 field 所用到的校验规则的参数，var-name 参数名，var-value 指定参数值。例如设置 mask 规则的具体的正则表达式：

```
 <var-name>mask</var-name>
 <var-value>${phone}</var-value>
```

设置 minLength 规则的最小长度：

```
<var-name>minLength</var-name>

<var-value>5</var-value>
```

当然，这个参数可以被 arg 元素应用，应用语法为：${var:var-name}。

1.1.1.1.1.1 插入 Validator

每一个 **Struts** 应用需要知道是否使用了 **Validator** 框架，可以通过 PlugIn(插件)机制将 **Validator** 框架配置到 **Struts** 应用中。

下面的代码演示了如何将 Validator 作为一个插件加入到 Struts 应用中，在 Struts 应用的配置文件 *Struts-config.xml* 中加入

如下代码片段：

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
    property="pathnames"
    value="/WEB-INF/validator-rules.xml,/WEB-INF/validator.xml"/>
</plug-in>
```

粗体部分 **pathnames** 属性的值用来指定 Validator 框架所使用的配置文件，多个配置文件之间用逗号分隔。

当应用启动的时候，**Struts** 框架将调用 ValidatorPlugIn 的 init()方法。**Validator** 框架的配置文件将会加载到内存中供应用使用。在 init()方法被调用之前，**pathnames** 所指定的值将会传递给 ValidatorPlugIn 的实例，ValidatorPlugIn 实例将会依据这个值去加载配置文件。

1.1.1.2 使用带校验的 ActionForm

你不能使用标准的 Struts ActionForm 去和 Validator 配合使用。你必须使用专门为 Validator 框架设计的 ActionForm 的子类。现在有两个子类可以选择，取决于你是否打算使用动态 ActionForms。下面的图直观地显示了 ActionForm 以及它的后代：

如果你打算使用动态 ActionForm，为了和 Validator 框架配合使用，你可以使用 DynaValidatorForm，否则，可以使用 ValidatorForm。从图上看出，DynaValidatorForm 有个子类叫做 DynaValidatorActionForm，ValidatorForm 有个子类叫做 ValidatorActionForm，这两个子类在其父类的名字中间加了个“Action”，这两个类有什么作用呢？

同样，根据你是否打算使用动态 ActionForm，你可以使用 DynaValidatorActionForm 或 ValidatorActionForm，来配合使用 Validator 框架，当使用这两个类时，它们将 action 的 path 属性传递给 Validator，Validator 使用 action 的名字去查找使用的校验规则。而使用 DynaValidatorForm 和 ValidatorForm，则是使用的 **ActionForm** 的 name 属性去查找匹配校验规则。(???)

1.1.1.1.1 示例

第一步，在 *Struts-config.xml* 中配置一个 ActionForm，示例如下：

```
<form-bean
  name="checkoutForm"
  type="org.apache.struts.validator.DynaValidatorForm">
  <form-property name="firstName" type="java.lang.String"/>
  <form-property name="lastName" type="java.lang.String"/>
  <form-property name="address" type="java.lang.String"/>
  <form-property name="city" type="java.lang.String"/>
  <form-property name="state" type="java.lang.String"/>
  <form-property name="postalCode" type="java.lang.String"/>
  <form-property name="country" type="java.lang.String"/>
  <form-property name="phone" type="java.lang.String"/>
</form-bean>
```

第二步，在 *Struts-config.xml* 中配置一个 Action，示例如下：

```
<action
   input="/checkout.jsp"
   name="checkoutForm"
   path="/checkout"
   scope="request"
   type="com.ort.struts.example.checkOutAction"
   validate="true">
 </action>
```

第三布，在 *validation.xml* 文件中定义如下：

```
<formset>
  <constant>
    <constant-name>phone</constant-name>
    <constant-value>^\(?(\d{3})\)?[-| ]?(\d{3})[-| ]?(\d{4})$</constant-value>
  </constant>
  <constant>
    <constant-name>zip</constant-name>
    <constant-value>^\d{5}(-\d{4})?$</constant-value>
  </constant>
```

```
  <form name="checkoutForm">
    <field
      property="firstName"
      depends="required,mask">
      <arg0 key="label.firstName"/>
      <var>
        <var-name>mask</var-name>
        <var-value>^[a-zA-Z]*$</var-value>
      </var>
    </field>
    <field
      property="postalCode"
      depends="required,mask">
      <arg0 key="registrationForm.zip"/>
      <var>
        <var-name>mask</var-name>
        <var-value>${zip}</var-value>
      </var>
    </field>
    <field
      property="phone"
      depends="required,mask">
      <arg0 key="registrationForm.phone"/>
      <var>
        <var-name>mask</var-name>
        <var-value>${phone}</var-value>
      </var>
    </field>
  </form>
 </formset>
</form-validation>
```

第四部，编写 HTML 页面如下：

暂略

1. JSP 自定义标签库

1.1 概述

在 **JSP** 开发中会遇到一些重复的工作。而使用自定义标签库是一种方法，可以用来将这些功能封装起来并在多个项目中重新用到它。此外，应用逻辑还可以包含在基于服务器的资源中，比如 **JavaBeans**。这种架构显示出使用自定义标签库可以更快更容易地开发基于 **Web** 的应用程序。

有关 JavaBeans 和自定义标签库的最初想法是：在程序员研究那些包含商务逻辑（business logic）的类的同时，Web 设计师可以同步进行页面设计。然后，Web 设计师可以通过使用简单的“连线”将 JSP 页面和这些类联系起来。尽管使用 JavaBean 会减少在 JSP 页面中写代码的数量，但你还是得写程序去使用这些 Beans。

然而使用自定义标签库则是一种完全无需在 JSP 中写代码的好办法。这并不是说自定义标签库会取代 JavaBeans，它们都是用来分离实际内容和显示形式的。JavaBeans 在用于商务逻辑被重用的设计中作用更为明显。JavaBeans 通常能在不同项目的各种页面中被用到。另一方面，自定义标签库则是一个特殊页面的自定义形式，即便如此，将它重新利用到其他程序中也是很常见的。得到自定义标签库的一种方法是自己建一个。但为什么不使用现成的呢？比如 Jakarta Taglibs 项目（源自 Apache Software Foundation）就提供了一些自定义标签库，它们可以在不同的 JSP 应用程序中重复使用。

1.2 Struts 包含的标签库

Struts 框架提供了一系列的框架组件，同时，他也提供了一系列的标签(Tag)用于和框架进行交互。Struts 提供的标签包含在以下四个标签库(Tag libraries)中：

- HTML
- Bean
- Logic
- Template

这四个标签库所包含的标签功能各自截然不同，从标签库的名字我们可以看出其功能，如，HTML 标签库是用来包装 HTML 控件的。

1.3 在 Struts 应用中使用标签库

和使用其它标签库一样，使用 Struts 提供的标签库只需要简单的两步：

1、 在 web.xml 中声明标签库：

```
<taglib>
  <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
```

```
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>
```

2、 在 JSP 页面中引入标签库：

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
```

1.4 Struts HTML 标签库

HTML 标签库中的标签列表

标签名	描述
base	包装 HTML 的 base 元素
button	包装 HTML 的 button 类型的 input 元素
cancel	包装 HTML cancel 按钮
checkbox	包装 HTML checkbox 类型的输入域
errors	有条件地显示一些 error 消息，显示 ActionErrors 信息
file	包装 HTML 文件上传输入域
form	定义 HTML form 元素
frame	包装 HTML frame 元素
hidden	包装 HTML hidden 输入域
html	包装 HTML 中的 html 元素
image	包装 "image"类型的输入域
img	包装 HTML 的 img 元素
javascript	包装根据 ValidatorPlugIn 提供的校验规则所提供的 javascript 校验脚本
link	包装超链接
messages	有条件地显示一些提示信息，显示 ActionMessages 信息
multibox	包装多选输入框
option	包装一个选择输入框
options	包装一批选择输入框

optionsCollection	包装一批选择输入框集
password	包装密文输入框
radio	包装单选输入框
reset	包装“重置”功能的按钮
rewrite	包装一个 URL
select	包装一个选择输入框
submit	包装一个提交按钮
text	包装一个文本输入框
textarea	包装一个备注输入框

在这里，不打算对每一个标签的使用进行详细说明，要想了解每一个标签的使用，请查看 Struts 官方文档。

接下来，我们着重学习一下几个非常重要的标签的使用，举一反三，通过这几个标签的使用，我想，即使不去看官方文档，也能够对其它标签的使用有个基本的了解。

1.1.1 form 标签

Struts 的 **form** 标签是最重要的标签之一，他包装了 HTML 的标准 **form** 标签，提供了将 HTML **form** 和 **ActionForm** 连接起来的功能。

HTML form 中的每一个域对应 **ActionForm** 的一个属性，当提交 HTML from 后，Struts 根据匹配关系，将 HTML **from** 域的值赋给 **ActionForm** 的同名属性。下表列举了 form 标签的属性，并且针对每一个属性加以详细说明：

Struts form 标签属性列表

Name	Description
action	form 提交的目标地址，action 用来选择一个 Struts **Action** 对提交后的客户请求进行处理。通过和 action 的 path 属性进行匹配来选择 Struts **Action**。 如果在 **web.xml** 中设置的 **servlet** 映射是扩展名映射，则可以用以下方式进行 **Action** 匹配(扩展名可以不作为匹配内容)： `<html:form action="login.do" focus="accessNumber">`

上面的语句表示 form 提交后，交给 path 属性值为 login 的 **Action** 进行处理

如果在 web.xml 中设置的 **servlet** 映射是路径映射，则 **action** 的值必须完全匹配 **Struts Action** 的 **path** 属性值，例如：

```
<html:form action="login" focus="accessNumber">
```

enctype	提交 form 使用的编码方式
focus	页面初始化时光标定位的输入控件名
method	提交请求的 HTTP 方式('POST' or 'GET')
name	对应的 **ActionForm** 的名字 ，如果不指定，则使用 Struts **Action** 在配置文件中 **name** 属性所指定的 **ActionForm**。
onreset	当 form 重置时执行的 Javascript
onsubmit	当 form 提交时执行的 javascript
scope	与 **form** 对应的 **ActionForm** 的有效区域，可以为 **request** 或 **session**
style	CSS 样式表 The CSS styles to be applied to this HTML element.
styleClass	CSS 样式类别
styleId	HTML 元素的 ID
target	form 提交的目标 frame
type	form 对应的 **ActionForm** 的类名

1.2 Struts Logic 标签库

Struts Logic 标签库中包含的标签列表

Tag name	Description
empty	如果标签 **parameter**，**propertie** 等属性所指定的变量值为 null 或空字符串，则处理标签包含的内容
equal	如果标签 **parameter**，**propertie** 等属性所指定的变量的值等于标签 **value** 属性所指定的值，则处理标签所包含的内容，如：

```
<logic:equal value="modify" property="action"   name="projectForm">
    <bean:message key="project.project_modify"/>
```

	</logic:equal> 上面的示例表示，如果 **projectForm** 的 **action** 属性等于 **modify**，则处理**<bean:message key="project.project_modify"/>**语句。
forward	Forward control to the page specified by the **ActionForward** entry.
greaterEqual	Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.
greaterThan	Evaluate the nested body content of this tag if the requested variable is greater than the specified value.
iterate	Repeat the nested body content of this tag over a specified collection.
lessEqual	Evaluate the nested body content of this tag if the requested variable is less than or equal to the specified value.
lessThan	Evaluate the nested body content of this tag if the requested variable is less than the specified value.
match	Evaluate the nested body content of this tag if the specified value is an appropriate substring of the requested variable.
messagesNotPresent	Generate the nested body content of this tag if the specified message is not present in this request.
messagesPresent	Generate the nested body content of this tag if the specified message is present in this request.
notEmpty	Evaluate the nested body content of this tag if the requested variable is neither null nor an empty string.
notEqual	Evaluate the nested body content of this tag if the requested variable is not equal to the specified value.
notMatch	Evaluate the nested body content of this tag if the specified value is not an appropriate substring of the requested variable.
notPresent	Generate the nested body content of this tag if the specified value is not present in this request.
present	Generate the nested body content of this tag if the specified value is present in this request.
redirect	Render an HTTP redirect.

执行比较功能的标签通用属性表

Name	Description
name	The name of a bean to use to compare against the **value** attribute. If the **property** attribute is used, the value compared against the property of the bean, instead of the bean itself.
parameter	The name of a request parameter to compare the **value** attribute against.
property	The variable to be compared is the property (of the bean specified by the **name** attribute) specified by this attribut The property reference can be simple, nested, and/or indexed.
scope	The scope within which to search for the bean named by the **name** attribute. All scopes will be searched if not specified.
value	The constant value to which the variable, specified by another attribute(s) of this tag, will be compared.

示例：

To check whether a particular request parameter is present, you can use the Logic `present` tag:

```
<logic:present parameter="id">
  <!-- Print out the request parameter id value -->
</logic:present>
```

To check whether a collection is empty before iterating over it, you can use the `notEmpty` tag:

```
<logic:notEmpty name="userSummary" property="addresses">
  <!-- Iterate and print out the user's addresses -->
</logic:notEmpty>
```

Finally, here's how to compare a number value against a property within an `ActionForm`:

```
<logic:lessThan property="age" value="21">
  <!-- Display a message about the user's age -->
</logic:lessThan>
```

1.1 Struts Bean 标签库

Struts Bean 标签库中的标签列表

Tag name	Description
`cookie`	Define a scripting variable based on the value(s) of the specified request cookie.
`define`	Define a scripting variable based on the value(s) of the specified bean property.
`header`	Define a scripting variable based on the value(s) of the specified request header.
`include`	Load the response from a dynamic application request and make it available as a bean.
`message`	Render an internationalized message string to the response.
`page`	Expose a specified item from the page context as a bean.
`parameter`	Define a scripting variable based on the value(s) of the specified request parameter.
`resource`	Load a web application resource and make it available as a bean.
`size`	Define a bean containing the number of elements in a `Collection` or `Map`.
`struts`	Expose a named Struts internal configuration object as a bean.
`write`	Render the value of the specified bean property.

1.2 Struts Template 标签库

Struts Template 标签库中的标签列表

Tag name	Description
`insert`	Retrieve (or include) the specified template file, and then insert the specified content into the template's layout. By changing the layout defined in the template file, any other file that inserts the template will automatically use the new layout.
`put`	Create a request-scope bean that specifies the content to be used by the `get` tag. Content can be printed directly or included from a JSP or HTML file.
`get`	Retrieve content from a request-scope bean, for use in the template layout.

1. 示例

示例实现了一个简单的项目信息 CRUD(添加、查询、更新、删除)+分页显示的功能。

其中项目有如下属性：

- 项目编号
- 项目名称
- 项目所属区域(区域存在与一个字典表中)
- 项目分成比例(0.00%--100.00%)

数据库结构定义如下所示(采用 Oracle8i 数据库,如果采用其它数据库请作相应变化)：

SITES ——区域表				
Name	**Datatype**	**Null Option**	**Comment**	**Is PK**
SITECODE	VARCHAR2(10)	NOT NULL	区域号	Yes
SITENAME	VARCHAR2(30)	NOT NULL	区域名称	No

PROJECTS ——项目表				
Name	**Datatype**	**Null Option**	**Comment**	**Is PK**
PROJECTCODE	VARCHAR2(10)	NOT NULL	项目编号	Yes
PROJECTNAME	VARCHAR2(30)	NOT NULL	项目名称	No
SITECODE	VARCHAR2(10)	NOT NULL	所属区域号	
DISCOUNT	NUMBER	NULL	项目分成比例	

附：在 ORACLE8i 中创建数据表的语句为：

```
--创建区域表
CREATE TABLE SITES (
   SITECODE VARCHAR2(10) NOT NULL,
   SITENAME VARCHAR2(30) NOT NULL
);
--添加主键
ALTER TABLE SITES
        ADD   ( CONSTRAINT SITES_PKSITECODE PRIMARY KEY (SITECODE)
```

```
        USING INDEX );
--创建项目信息表
CREATE TABLE PROJECTS(
   PROJECTCODE VARCHAR2(10) NOT NULL,
   PROJECTNAME VARCHAR2(30) NOT NULL,
   SITECODE VARCHAR2(10) NOT NULL,
   DISCOUNT NUMBER NULL
);
--添加主键索引
ALTER TABLE PROJECTS
        ADD   ( CONSTRAINT PROJECTS_PKPROJECTCODE PRIMARY KEY (PROJECTCODE)
        USING INDEX );
--添加外键索引
CREATE INDEX FK_PROJECTSSITECODE ON PROJECTS
(
        SITECODE   ASC
);
--添加外键约束
   ALTER TABLE PROJECTS
        ADD   ( CONSTRAINT FK_PROJECTSSITECODE
                FOREIGN KEY (SITECODE)
                        REFERENCES SITES ) ;
```

```
--创建区域表
CREATE TABLE SITES (
   SITECODE VARCHAR2(10) NOT NULL,
   SITENAME VARCHAR2(30) NOT NULL
);
```

```
--添加主键
ALTER TABLE SITES
        ADD   ( CONSTRAINT SITES_PKSITECODE PRIMARY KEY (SITECODE)
        USING INDEX );
--创建项目信息表
CREATE TABLE PROJECTS(
   PROJECTCODE VARCHAR2(10) NOT NULL,
   PROJECTNAME VARCHAR2(30) NOT NULL,
   SITECODE VARCHAR2(10) NOT NULL,
   DISCOUNT NUMBER NULL
);
--添加主键索引
ALTER TABLE PROJECTS
        ADD   ( CONSTRAINT PROJECTS_PKPROJECTCODE PRIMARY KEY (PROJECTCODE)
        USING INDEX );
--添加外键索引
CREATE INDEX FK_PROJECTSSITECODE ON PROJECTS
(
        SITECODE   ASC
);
--添加外键约束
   ALTER TABLE PROJECTS
        ADD   ( CONSTRAINT FK_PROJECTSSITECODE
                FOREIGN KEY (SITECODE)
                        REFERENCES SITES ) ;
```

1.1 命名规范

1、 所有 jsp,action 映射路径均为小写字母，如有需要可以使用小写字母+数字：例如：

/projectlist,/projetlist.jsp

2、 所有<html:form>中的元素(如文本框，列表框等)名称都使用 java 规范的变量命名方式(变量由一个或多个单词组成，第一个单词小写，第一个单词后的单词首字母大写)，例如：

```
<html:text styleClass="input" maxlength="10" property="projectCode" size="30"/>
```

3、 其它方面均遵守 SUN 推荐的编码规范。

1.2 文件

1.2.1 projectlist.jsp

该 jsp 页面用来显示项目信息列表，并提供查询功能。同时，提供按钮将用户导向到添加、修改、删除功能。

1.2.2 projectform.jsp

在执行添加、修改操作之前，需要提供一个 form 供用户输入数据，在执行删除操作之前，需要提供一个 form 将被删除数据显示出来，供用户确认。该 jsp 页面就是用来满足这些需要，提供对单条项目信息的显示，根据具体的操作类型(创建、修改、删除)，数据被显示在可编辑的输入控件中或不可编辑的 label(文本标签)上。

1.2.3 success.jsp

添加、修改、删除等操作正常完成后，提供一个页面显示“恭喜”信息，使得用户能够清楚知道自己的行为已经生效 J。

1.2.4 failed.jsp

添加、修改、删除等操作异常失败，提供一个页面显示“失败”信息，使得用户能够清楚知道自己的行为已经失败 L。

1.2.5 ProjectListSearchAction.java

“Project”+”List”+”Search”+”Action”，组成了这个 Action 的名字，这是我个人的命名风格，表示这个 Action 会处理项目列表和查询事务。在 projectlist.jsp 被装载之前，ProjectListSearchAction 需要将数据加载到 scope 指定的地方，供 projectlist.jsp 显示，当用户从 projectlist.jsp 中提交查询请求，该 Action 需要处理查询，并加载数据，供 projectlist.jsp 显示。

1.2.6 ProjectFormLoadAction

这个 Action 用来处理在显示 projectform.jsp 之前，将所需要的数据加载到 scope 指定的范围中，供 projectform 使用。

1.2.7 ProjectFormSaveAction.java

这个 Action 用来处理用户在 projectform.jsp 中提交的数据，根据用户的操作类型，完成具体的操作，并将合适的提示页面(success.jsp or failed.jsp)显示给用户。

1.2.8 web.xml

在 Struts 安装那一节，我们已经知道 web.xml 文件的作用，通过这个文件，我们可以将 ActionServlet 配置好，用以截获用户对 Struts 应用的访问请求。下面是示例程序所用的 web.xml 内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>simpledemo</display-name>
  <description>Demo for using STRUTS to do some thing about
CRUD(Create,Read,Update,Delete) and any more....</description>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>debugjsp</servlet-name>
    <description>Added to compile JSPs with debug info</description>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
    <init-param>
      <param-name>classdebuginfo</param-name>
```

```
      <param-value>true</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>debugjsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>20</session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>projectsearch.do</welcome-file>
  </welcome-file-list>
  <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
```

```
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-template.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-template.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-nested.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
  </taglib>
</web-app>
```

.1.1 资源文件

使用资源文件来放置标签显示值，提示信息等，如果你的 Struts 应用有国际化的要求，那么资源文件一定要好好地利用，就算没有国际化需求，使用资源文件也可以统一应用程序用语(例如统一的提示信息，标签信息等)，而且当用语发生变化（如“小时”变成“钟头”J），很容易统一进行修改。下面是示例程序所用到的资源文件内容(application.properties)：

```
#System global labels
button_cancel = 取消
button_edit = 修改
button_delete = 删除
button_save = 保存
button_submit = 确认


#lables for project
projectcontroller.title = 管理项目
project.project_create = 添加项目
```

```
project.project_modify = 修改项目
project.project_list = 已添加项目列表
project.projectcode = 项目代码
project.projectname = 项目名称
project.discount = 项目分成比例
project.site = 所属小区
# Standard error messages for validator framework checks
errors.required={0} is required.
errors.minlength={0} cannot be less than {1} characters.
errors.maxlength={0} cannot be greater than {2} characters.
errors.invalid={0} is invalid.
errors.byte={0} must be an byte.
errors.short={0} must be an short.
errors.integer={0} must be an integer.
errors.long={0} must be an long.
errors.float={0} must be an float.
errors.double={0} must be an double.
errors.date={0} is not a date.
errors.range={0} is not in the range {1} through {2}.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is an invalid e-mail address.
```

那么，如何在 Struts 应用中引用资源文件呢？

首先需要在 Struts 配置文件(Struts-config.xml)中指明配置文件的路径，在配置文件中添加如下一行信息：

```
<message-resources parameter="ApplicationResources_CN" />
```

parameter 所指的就是配置文件，注意，为什么这里指明的是 applicationResources_CN，而不是上面提到的 application.properties？这是为了能在 Struts 中正确显示中文，利用 jdk 自带的 native2ascii 程序对 application.properties 作了编码转换，编码转换后的文件名为 ApplicationResources_CN.properties，扩展名可以省略。

需要注意的是，改配置文件一定要放在 classpath 范围内，一般放置在 WEB-INF/classes 目录下，如果放在 classes 的子目录下，其指引方式和 java 包一样，例如在 WEB-INF/classes/com 目录下，则应该用如下语句指引：

```
<message-resources parameter="com.ApplicationResources_CN" />
```

小技巧

进行中文编码转换的命令如下：

native2ascii -encoding gb2312 application.properties ApplicationResources_CN.properties

在配置文件声明了对资源文件的引用之后，就可以在 **Struts** 提供的标签以及校验框架等其它地方使用这些资源，具体使用方法请查看相关标签和配置文件说明。

1.1.2 struts-config.xml

该文件定义了 **Struts** 应用中的 **Action,ActionForm,**插件，资源引用等信息，示例程序 **struts-config.xml** 文件内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
  <form-beans>
      <!--项目增、删、改 form 定义 begin-->
    <form-bean name="projectForm" type="com.ort.strutsdemo.simpledemo.ui.ProjectForm">
    </form-bean>
    <!--项目增、删、改 form 定义 结束-->
    <!--进入项目增、删、改界面之前参数传递 Form begin-->
    <form-bean name="projectLoadForm" type="org.apache.struts.action.DynaActionForm">
      <form-property name="action" size="10" type="java.lang.String" initial="create"/>
      <form-property name="projectCode" size="30" type="java.lang.String" initial=""/>
    </form-bean>
    <!--进入项目增、删、改界面之前参数传递 Form end-->
    <!--项目查询 Form 定义 Begin-->
    <form-bean name="projectSearchForm" type="org.apache.struts.action.DynaActionForm">
        <form-property name="projectCode" size="10" type="java.lang.String" initial=""/>
        <form-property name="projectCodeSign" size="10" type="java.lang.String" initial=""/>
        <form-property name="projectName" size="10" type="java.lang.String" initial=""/>
        <form-property name="projectNameSign" size="10" type="java.lang.String" initial=""/>
```

```
        <form-property name="page" size="10" type="java.lang.Integer" initial="1"/>
        <form-property name="pageCount" size="10" type="java.lang.Integer" initial="1"/>
    </form-bean>
    <!--项目查询 Form 定义 End-->
  </form-beans>
  <global-forwards>
    <forward name="projectlist" path="/projectlist.jsp" />
    <forward name="projectform" path="/projectform.jsp" />
    <forward name="failed" path="/failed.jsp" />
    <forward name="success" path="/success.jsp" />
  </global-forwards>

  <action-mappings>
    <action input="projectform" name="projectForm" path="/projectformsave" scope="request"
type="com.ort.strutsdemo.simpledemo.controller.ProjectFormSaveAction" validate="true">
      <forward name="success" path="/projectsearch.do" redirect="true" />
      <forward name="success.return" path="/projectsearch.do" redirect="true" />
      <forward name="cancel" path="/projectsearch.do" redirect="true" />
      <forward name="failed" path="/failed.jsp" />
      <forward name="failed.return" path="/projectsearch.do" />
    </action>
    <action input="projectlist" name="projectLoadForm" path="/projectformload" scope="request"
type="com.ort.strutsdemo.simpledemo.controller.ProjectFormLoadAction" validate="false">
      <forward name="success" path="/projectform.jsp" />
      <forward name="failed" path="/failed.jsp" />
      <forward name="failed.return" path="/projectsearch.do" redirect="true" />
    </action>
     <action input="projectlist" name="projectSearchForm" path="/projectsearch" scope="request"
type="com.ort.strutsdemo.simpledemo.controller.ProjectListSearchAction" validate="false">
```

```
        <forward name="success" path="/projectlist.jsp"/>
        <forward name="failed" path="/failed.jsp" />
        <forward name="failed.return" path="/projectsearch.do" redirect="true" />
      </action>
    </action-mappings>
    <controller>
      <set-property property="inputForward" value="true" />
    </controller>
    <message-resources parameter="ApplicationResources_CN" />
    <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
      <set-property property="pathnames"
value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml" />
    </plug-in>
</struts-config>
```

1.1 CRUD

CRUD(Create，Retrieve or Read?，Update，Delete)是信息管理系统中经常要执行的操作的简称。接下来我们分别描述这几种操作的执行流程。

1.1.1 查询项目信息

执行项目信息查询列表显示的文件请求处理顺序如下：

1、 ProjectListSearchAction

2、 projectlist.jsp

ProjectListSearchAction 代码如下：

```
package com.ort.strutsdemo.simpledemo.controller;


/**
```

```
 * <p>Title: Struts Training </p>
 * <p>Description: Struts 内部培训 Demo</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author Liuz
 * @version 1.0
 */
import org.apache.struts.action.*;
import javax.servlet.http.*;
import com.ort.strutsdemo.simpledemo.business.BusinessDelegate;
import com.boss.module.operation.object.Project;
import com.ort.strutsdemo.simpledemo.Constants;
import com.ort.strutsdemo.simpledemo.controller.exception.ExceptionBean;
import com.boss.module.operation.object.searchresult.help.ResultSetIterator;
import com.ort.strutsdemo.simpledemo.ui.ProjectForm;
import com.boss.module.operation.object.searchgene.ProjectSearchGene;

/**
 *
 * <p>Title: Struts Training </p>
 * <p>Description: 项目查询结果数据的装载</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: ORT</p>
 * @author Liuz
 * @version 1.0
 */
public class ProjectListSearchAction
    extends Action {
    BusinessDelegate businessDelegate = null;
```

```
    public ActionForward execute(ActionMapping actionMapping,
                                 ActionForm actionForm,
                                 HttpServletRequest request,
                                 HttpServletResponse response) {
        try {
            DynaActionForm form = (DynaActionForm) actionForm;
            //定义分页所需要变量
            int page= ((Integer)form.get("page")).intValue();
            int pageSize = 5;
            int allSize = 0;
            int pageCount = 0;
            //获取用户输入查询值,并形成查询条件
            String projectCode = (String)form.get("projectCode");
            String projectCodeSign = (String)form.get("projectCodeSign");
            String projectName = (String)form.get("projectName");
            String projectNameSign = (String)form.get("projectNameSign");
            ProjectSearchGene searchGene = new ProjectSearchGene(); //searchGene 为一查询精
灵，用以处理查询操作，在此不用过多关注，有机会在另文介绍
            searchGene.setProjectCode(projectCode, projectCodeSign);
            searchGene.setProjectName(projectName, projectNameSign);



            //装载当前页面所需要显示项目列表数据
             BusinessDelegate businessDelegate = BusinessDelegate.getInstance(); //业务层操
作，不用关注
             ResultSetIterator projectIterator = businessDelegate.
                 getProjectIterator(searchGene,pageSize);
             java.util.List projects = projectIterator.getElements(page);
             if (projects == null) {
```

```
                projects = new java.util.ArrayList();
            }
            //将项目列表查询结果放置到请求对象中
            request.removeAttribute(Constants.PROJECT_LISTFORM_KEY);
            request.setAttribute(Constants.PROJECT_LISTFORM_KEY, projects);
            //计算总页数
            allSize = projectIterator.getAllSize();
            pageCount = (allSize % pageSize == 0) ? allSize / pageSize :
                allSize / pageSize + 1;
           // System.err.print(pageCount);
            form.set("pageCount", new Integer(pageCount));

            //装载当前页面所需要现实小区信息
            ResultSetIterator siteIterator = businessDelegate.getSitesIterator();
            java.util.List sites = siteIterator.getElements(1);
            request.getSession().setAttribute(Constants.SITE_LISTFORM_KEY,
                                              sites);
            //重定向到 mapping 中配置的 input 页面
//              return actionMapping.findForward("success");
            return actionMapping.getInputForward();
        }
        catch (Exception ex) {
            com.ipbs.util.Log.println(
                "[ProjectListSearchAction.java][Exception]:" + ex.getMessage());
            ExceptionBean exception = new ExceptionBean();
            exception.setErrorMsg(Constants.getExceptionMsg(ex));
            exception.setReturnPath(actionMapping.findForward("failed.return").
                                    getPath());
            request.setAttribute(Constants.EXCEPTION_BEAN, exception);
```

```
            return actionMapping.findForward("failed");
        }
    }
}
```

在上面的代码中，应用了一个常量类(Constants)，为了让大家看得更加明白，将其代码显示如下：

```
package com.ort.strutsdemo.simpledemo;


/**
 * <p>Title: Struts Training </p>
 * <p>Description: Struts 内部培训 Demo</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author Liuz
 * @version 1.0
 */


public class Constants {
  public static final String PROJECT_SINGLEFORM_KEY = "projectForm";
  public static final String PROJECT_LISTFORM_KEY = "PROJECTS";
  public static final String PROJECT_SEARCHFORM_KEY = "SEARCHPROJECTS";
  public static final String PROJECT_CONTROLLERFORM_KEY = "PROJECTCONTROLLER";
  public static final String SITE_LISTFORM_KEY = "SITES";
  public static final String SITE_SINGLEFORM_KEY = "SITE";


  public static final String EXCEPTION_BEAN = "EXCEPTIONBEAN";
  /**
   * 通过识别异常的基础类型，返回易懂的提示信息
   * @param ex Exception 异常
   * @return String 异常的描述信息
```

```
     */
    public static final String getExceptionMsg(Exception ex){
        String returnMsg="";
        if(ex!=null){
            if (ex instanceof com.boss.module.operation.command.exception.AlreadyExistException){
                returnMsg = "数据已存在!";
                return returnMsg;
            }else if (ex instanceof com.boss.module.operation.command.exception.DbException){
                returnMsg = "数据库错误!";
                return returnMsg;
            }else if(ex instanceof
com.boss.module.operation.command.exception.InvalidObjectException){
                returnMsg = "无效的数据!";
                return returnMsg;
            }else if(ex instanceof com.boss.module.operation.command.exception.InvalidPkException){
                returnMsg = "无效的主键!";
                return returnMsg;
            }else if(ex instanceof com.boss.module.operation.command.exception.NotFoundException){
                returnMsg = "数据不存在!";
                return returnMsg;
            }else if(ex instanceof
com.boss.module.operation.command.exception.UnAuthorizationException){
                returnMsg = "无权限!";
                return returnMsg;
            }else if(ex instanceof
com.boss.module.operation.object.searchresult.exception.IteratorException){
                returnMsg = "获取列表数据异常!";
                return returnMsg;
            }
```

```
        }
      return returnMsg;
    }
}
```

projectlist.jsp 内容如下：

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<%@ taglib uri="/WEB-INF/struts-nested.tld" prefix="nested" %>
<%@ taglib uri="/WEB-INF/struts-template.tld" prefix="template" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ page contentType="text/html; charset=GB2312" %>

<html:html>
<head>
<!—projectlist.title 为资源引用-->
<title><bean:message key="projectlist.title"/></title>
<link href="css/main.css" rel="stylesheet" type="text/css">
<meta http-equiv="Content-Type" content="text/html; charset=gb2312"></head>
<body bgcolor="#ffffff">
<html:form action="projectsearch">
    <table width="70%"  border="0" align="center" cellpadding="3" cellspacing="1"
class="tablebodycolor">
    <tr class="tdbodycolor">
      <td colspan="2" class="tdheadcolor">
            <bean:message key="project.project_search"/>
      </td>
```

```
    </tr>
    <tr class="tdbodycolor">
      <td width="15%"><div align="right"><bean:message
key="project.projectcode"/></div></td>
      <td width="85%">
          <html:select property="projectCodeSign">
              <html:option value="like">like</html:option>
              <html:option value="not like">not like</html:option>
              <html:option value="=">equal</html:option>
              <html:option value="<>">not equal</html:option>
          </html:select>
     <html:text styleClass="input" maxlength="30" property="projectCode" size="30"/>
      </td>
    </tr>
    <tr class="tdbodycolor">
      <td><div align="right"><bean:message key="project.projectname"/></div></td>
      <td>
          <html:select property="projectNameSign">
              <html:option value="like">like</html:option>
              <html:option value="not like">not like</html:option>
              <html:option value="=">equal</html:option>
              <html:option value="<>">not equal</html:option>
          </html:select>
          <html:text styleClass="input" maxlength="30" property="projectName" size="30"/>
      </td>
    </tr>
    <tr class="tdbodycolor">


   <tr class="tdbodycolor">
```

```
      <td colspan="2"><div align="center">
          <html:hidden property="page"/>
          <html:hidden property="pageCount"/>
        <html:submit><bean:message key="button_search"/></html:submit>
      </div></td>
    </tr>
    </tr>
  </table>
</html:form>
<table width="70%" border="0" align="center" cellpadding="2" cellspacing="1"
class="tablebodycolor">
  <tr class="tdheadcolor">
    <td height="19" colspan="6" class="tdbodycolor">
      <input type="button" name="button_create"
          value="<bean:message key="button_create"/>"
onclick="javascript:window.location='projectformload.do?action=create'"/>
    </td>
  </tr>
</table>
<table width="70%" border="0" align="center" cellpadding="2" cellspacing="1"
class="tablebodycolor">
  <tr class="tdheadcolor">
    <td height="19" colspan="6" class="tdbodycolor"><bean:message
key="project.project_list"/></td>
  </tr>
  <tr class="tdheadcolor">
    <td width="92"><div align="center"><bean:message key="project.projectcode"/></div></td>
    <td width="182"><div align="center"><bean:message
key="project.projectname"/></div></td>
```

```
    <td width="70"><div align="center"><bean:message key="project.discount"/></div></td>
    <td width="139"><div align="center"><bean:message key="project.site"/></div></td>
    <td width="83"><div align="center"><bean:message key="button_edit" /></div></td>
    <td width="83"><div align="center"><bean:message key="button_delete"/></div></td>
  </tr>
  <logic:iterate id="project" name="PROJECTS">
  <tr class="tdbodycolor">
    <td><div align="center"><bean:write name="project" property="projectCode"/></div></td>
    <td><div align="center"><bean:write name="project" property="projectName"/></div></td>
    <td><div align="center"><bean:write name="project" property="discount"/></div></td>
    <td><div align="center"><bean:define id="site" name="project" property="site">
                            </bean:define>
                            <bean:write name="site" property="siteName"/></div></td>
    <td><div align="center">
        <input type="button" name="button_edit"
            value="<bean:message key="button_edit"/>"
onclick="javascript:window.location='projectformload.do?action=modify&projectCode=<bean:write
name="project" property="projectCode"/>'"/>
    </div></td>
    <td><div align="center">
       <input type="button" name="button_delete"
            value="<bean:message key="button_delete"/>"
onclick="javascript:window.location='projectformload.do?action=delete&projectCode=<bean:write
name="project" property="projectCode"/>'"/>
    </div></td>
  </tr>
  </logic:iterate>
</table>
```

```
<!------分页----->
<script language="javascript">
    function firstPage(){
            document.projectSearchForm.page.value = 1;
            document.projectSearchForm.submit();
    }
        function previousPage(){
            document.projectSearchForm.page.value =
parseInt(document.projectSearchForm.page.value)-1;
            document.projectSearchForm.submit();
        }
        function nextPage(){
            document.projectSearchForm.page.value =
parseInt(document.projectSearchForm.page.value)+1;
            document.projectSearchForm.submit();
        }
        function lastPage(){
            document.projectSearchForm.page.value =
document.projectSearchForm.pageCount.value;
            document.projectSearchForm.submit();
        }
</script>
<table width=600 cellspacing="0" cellpadding="0" align="center">
        <tr>
          <td colspan=2><br>
              <div align="center">
         <bean:define id="currPageOb" property="page" name="projectSearchForm"/>
                <bean:define id="pageCountOb" property="pageCount" name="projectSearchForm"/>
                <%int currPage = ((Integer)(currPageOb)).intValue();
```

```
                int pageCount = ((Integer)(pageCountOb)).intValue();%>
                <%if(currPage>1){%>
                        <a href="javascript:firstPage();">首页</a>
                    <a href="javascript:previousPage();">上一页</a>
                <%}%>
                <%if(pageCount>currPage){%>
                    <a href="javascript:nextPage();">下一页</a>
                        <a href="javascript:lastPage();">末页</a>
                <%}%>
                当前页:<%=currPage%>,
                共<%=pageCount%>页
              </div>
            </td>
        </tr>
</table>
<!------分页----->


</body>
</html:html>
```

界面显示如下：

1.1.1 创建项目信息

执行项目信息查询列表显示的文件请求处理顺序如下：

1、 **ProjectListSearchAction**

2、 **projectlist.jsp**

3、 **ProjectFormLoadAction**

4、 **projectform.jsp**

5、 **ProjectFormSaveAction**

6、 **success.jsp or failed.jsp**

7、 **ProjectListSearchAction**

ProjectFormLoadAction 内容如下：

```
package com.ort.strutsdemo.simpledemo.controller;



/**
 * <p>Title: Struts Training </p>
 * <p>Description: Struts 内部培训 Demo</p>
 * <p>Copyright: Copyright (c) 2004</p>
```

```
 * <p>Company: </p>
 * @author Liuz
 * @version 1.0
 */
import org.apache.struts.action.*;
import javax.servlet.http.*;
import com.ort.strutsdemo.simpledemo.business.BusinessDelegate;
import com.boss.module.operation.object.Project;
import com.ort.strutsdemo.simpledemo.Constants;
import com.ort.strutsdemo.simpledemo.controller.exception.ExceptionBean;
import com.ort.strutsdemo.simpledemo.ui.ProjectForm;
import com.boss.module.operation.object.searchresult.help.ResultSetIterator;


public class ProjectFormLoadAction
    extends Action {
  BusinessDelegate businessDelegate = null;
  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) {
    businessDelegate = BusinessDelegate.getInstance();
    DynaActionForm form = (DynaActionForm)actionForm;
    String action = (String)form.get("action");
    String projectCode = (String)form.get("projectCode");
    try {
        Project project = null;
        if ( ("create".equals(action))) {
            project = new Project();
        }
```

```
        else {
            project = businessDelegate.getProject(projectCode);
        }
        project.setAction(action);
        ProjectForm modifyForm = new ProjectForm();
        org.apache.commons.beanutils.PropertyUtils.copyProperties(
            modifyForm, project);
        if (! ("create".equals(action))) {
            modifyForm.setSiteCode(project.getSite().getSiteCode());
        }
        request.setAttribute(Constants.PROJECT_SINGLEFORM_KEY, modifyForm); //为下一个页面，
即 projectform.jsp 的显示提供数据
        //装载当前页面所需要显示小区信息
        ResultSetIterator siteIterator = businessDelegate.getSitesIterator();
        java.util.List sites = siteIterator.getElements(1);
        request.getSession().setAttribute(Constants.SITE_LISTFORM_KEY,
                                          sites);

        return actionMapping.findForward("success");
    }
    catch (Exception ex) {
        com.ipbs.util.Log.println("[ProjectFormLoadAction.java][Exception]:"+ex.getMessage());
        ExceptionBean exception = new ExceptionBean();
        exception.setErrorMsg(Constants.getExceptionMsg(ex));
        exception.setReturnPath(actionMapping.findForward("failed.return").getPath());
        request.setAttribute(Constants.EXCEPTION_BEAN,exception);
        return actionMapping.findForward("failed");
    }
  }
```

```
}
```

projectform.jsp 内容如下：

```
<%
/**
 * 文件名:projectform.jsp
 * 描述:在执行添加、修改操作之前，需要提供一个 form 供用户输入数据，在执行删除操作之前，
 *       需要提供一个 form 将被删除数据显示出来，供用户确认。该 jsp 页面就是用来满足这些需要，
 *       提供对单条项目信息的显示，根据具体的操作类型(创建、修改、删除)，数据被显示在可编
 *       辑的输入控件中或不可编辑的
 *
 *+------------------------------------------------------------------------
 * 更改历史
 * 更改时间                    更改人          目标版本              更改内容
 *+------------------------------------------------------------------------
 * 2004-04-21 16:09           liuz          1.00.000              创建
 *
 *
 */
%>
<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<html>
<head>
<title>
<logic:equal value="modify" property="action"   name="projectForm">
    <bean:message key="project.project_modify"/>
```

```
</logic:equal>
<logic:equal value="create" property="action" name="projectForm">
  <bean:message key="project.project_create"/>
</logic:equal>
<logic:equal value="delete" property="action" name="projectForm">
  <bean:message key="project.project_delete"/>
</logic:equal>
</title>
<link href="css/main.css" rel="stylesheet" type="text/css">
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body bgcolor="#ffffff">


<html:form action="/projectformsave" method="post" onsubmit="return validateProjectForm(this);">
  <table width="70%"  border="0" align="center" cellpadding="3" cellspacing="1"
class="tablebodycolor">
    <tr class="tdbodycolor">
      <td colspan="2" class="tdheadcolor">
          <logic:equal value="modify" property="action" name="projectForm">
              <bean:message key="project.project_modify"/>
          </logic:equal>
          <logic:equal value="create" property="action" name="projectForm">
              <bean:message key="project.project_create"/>
          </logic:equal>
          <logic:equal value="delete" property="action" name="projectForm">
              <bean:message key="project.project_delete"/>
          </logic:equal>
      </td>
    </tr>
```

```
    <tr class="tdbodycolor">
      <td width="15%"><div align="right"><bean:message
key="project.projectcode"/></div></td>
      <td width="85%">
          <logic:equal value="modify" property="action" name="projectForm">
              <html:hidden property="projectCode" write="true"/>
          </logic:equal>
          <logic:notEqual value="modify" property="action" name="projectForm">
              <html:text styleClass="input" maxlength="10" property="projectCode" size="30"/>
          </logic:notEqual>
          </td>
    </tr>
    <tr class="tdbodycolor">
      <td><div align="right"><bean:message key="project.projectname"/></div></td>
      <td><html:text styleClass="input" maxlength="30" property="projectName" size="30"/></td>
    </tr>
    <tr class="tdbodycolor">
      <td><div align="right"><bean:message key="project.discount"/></div></td>
      <td><html:text styleClass="input" maxlength="10" property="discount" size="10"/>
      %</td>
    </tr>
    <tr class="tdbodycolor">
      <td><div align="right"><bean:message key="project.site"/></div></td>
      <td>
          <html:select property="siteCode">
              <html:options  collection="SITES"  labelProperty="siteName"  property="siteCode"/>
          </html:select>
      </td>
    </tr>
```

```
    <tr class="tdbodycolor">
      <td colspan="2"><div align="center">
          <html:hidden property="action" />
        <html:submit><bean:message key="button_submit"/></html:submit>
            <html:cancel><bean:message key="button_cancel"/>
            </html:cancel>
      </div></td>
    </tr>
  </table>
</html:form>
<html:javascript formName="projectForm"
        dynamicJavascript="true"
         staticJavascript="false"/>
<script language="Javascript1.1" src="staticJavascript.jsp"></script>
</body>
</html>
```

界面显示效果如下：

图 **6.3.2.1** 修改项目图

图 **6.3.2.2** 添加项目图

图 6.3.2.3 删除确认图

最后,信息新增、修改或删除确认后，需要提交给 ProjectFormSaveAction 进行，ProjectFormSaveAction 内容如下：

```
package com.ort.strutsdemo.simpledemo.controller;


import org.apache.struts.action.*;
import javax.servlet.http.*;
import com.ort.strutsdemo.simpledemo.business.BusinessDelegate;
import com.boss.module.operation.object.Project;
import com.boss.module.operation.object.Site;
import com.ort.strutsdemo.simpledemo.Constants;
import com.ort.strutsdemo.simpledemo.ui.ProjectForm;
import com.ort.strutsdemo.simpledemo.controller.exception.ExceptionBean;
import com.ipbs.util.web.ParamUtils;
/**
 *
 * <p>Title: Struts Training </p>
 * <p>Description: 项目管理功能页面的导向,以及页面所需要数据的初始化,同时,处理删除操作</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: </p>
 * @author Liuz
 * @version 1.0
 */
public class ProjectFormSaveAction
    extends Action {
  BusinessDelegate businessDelegate = null;
  public ActionForward execute(ActionMapping actionMapping,
```

```
                                    ActionForm actionForm,
                                    HttpServletRequest request,
                                    HttpServletResponse response) {
    businessDelegate = BusinessDelegate.getInstance();
    ProjectForm form = (ProjectForm)actionForm;
    ActionForward forward = null;
    String action = form.getAction();
    if (this.isCancelled(request)) {
         return (actionMapping.findForward("cancel"));
    }
    if("modify".equals(action)){
         forward = modifyProject(actionMapping, actionForm,
                                              request, response);
    }
    else if ("create".equals(action)) {
      forward = createProject(actionMapping, actionForm,request, response);
    }else if("delete".equals(action)){
        forward = deleteProject(actionMapping, actionForm,request, response);
    }
    return forward;
  }



  public ActionForward modifyProject(ActionMapping actionMapping,
                                       ActionForm actionForm,
                                     HttpServletRequest request,
                                     HttpServletResponse response
                                     )
{
```

```
    ProjectForm form = (ProjectForm) actionForm;
    String projectCode = form.getProjectCode();
    String projectName = form.getProjectName();
    String siteCode = form.getSiteCode();
    double discount = form.getDiscount();
     try{
          Site site = businessDelegate.getSite(siteCode);
          Project project = new Project();
          project.setProjectCode(projectCode);
          project.setProjectName(projectName);
          project.setDiscount(discount);
          project.setSite(site);
          businessDelegate.modifyProject(project);
          return actionMapping.findForward("success");
     }catch(Exception ex){
          com.ipbs.util.Log.println(
               "[ProjectFormSaveAction.java][Exception]:" + ex.getMessage());
          ExceptionBean exception = new ExceptionBean();
          exception.setErrorMsg(Constants.getExceptionMsg(ex));
          exception.setReturnPath(actionMapping.getPath());
          request.setAttribute(Constants.EXCEPTION_BEAN, exception);
          return actionMapping.findForward("failed");
     }
}


public ActionForward createProject(ActionMapping actionMapping,
                                    ActionForm actionForm,
                                   HttpServletRequest request,
                                   HttpServletResponse response
```

```
                                    ) {
    ProjectForm form = (ProjectForm) actionForm;
    String projectCode = form.getProjectCode();
    String projectName = form.getProjectName();
    String siteCode = form.getSiteCode();
    double discount = form.getDiscount();
    try{
        Site site = businessDelegate.getSite(siteCode);
        Project project = new Project();
        project.setProjectCode(projectCode);
        project.setProjectName(projectName);
        project.setDiscount(discount);
        project.setSite(site);
        businessDelegate.createProject(project);
        return actionMapping.findForward("success");
    }catch(Exception ex){
        com.ipbs.util.Log.println(
             "[ProjectFormSaveAction.java][Exception]:" + ex.getMessage());
         ExceptionBean exception = new ExceptionBean();
         exception.setErrorMsg(Constants.getExceptionMsg(ex));
         exception.setReturnPath(actionMapping.getPath());
         request.setAttribute(Constants.EXCEPTION_BEAN, exception);
         return actionMapping.findForward("failed");
    }
}


public ActionForward deleteProject(ActionMapping actionMapping,
                                   ActionForm actionForm,
                                   HttpServletRequest request,
```

```
                                    HttpServletResponse response
                                    ) {
        ProjectForm form = (ProjectForm)actionForm;
        String projectCode = form.getProjectCode();
        try{
            businessDelegate.deleteProject(projectCode);
            request.removeAttribute(Constants.PROJECT_SINGLEFORM_KEY);
            return actionMapping.findForward("success");
        }catch(Exception ex){
            com.ipbs.util.Log.println(
                "[ProjectControllerAction.java][Exception]:" + ex.getMessage());
            ExceptionBean exception = new ExceptionBean();
            exception.setErrorMsg(Constants.getExceptionMsg(ex));
            exception.setReturnPath(actionMapping.findForward("failed.return").getPath());
            request.setAttribute(Constants.EXCEPTION_BEAN,exception);
            return actionMapping.findForward("failed");

        }
    }

}
```

至此为止，该示例项目所使用的绝大部分内容已经介绍完了，因为时间问题，没有对示例程序进行很详细的一一讲解，代码上有些简单的注释，希望你能够看明白，如果大家对以上代码有疑问会发现什么问题，可以发邮件和我沟通(lzasp800@sina.com)，另外，我保证上面的代码是真实可运行的,如果用心看，也是可以看明白的(当然只有文章中所列的代码是无法通过编译的)。

个人觉得，做 Struts 项目非常重要的一个环节就是请求处理流程设计，这也是本示例强调的重点，弄清楚了这一点，看代码也会容易很多。

1.1.1 修改项目信息

请求处理流程如下,具体文件内容请查看前述内容：

1、 **ProjectListSearchAction**

2、 **projectlist.jsp**

3、 **ProjectFormLoadAction**

4、 **projectform.jsp**

5、 **ProjectFormSaveAction**

6、 **success.jsp or failed.jsp**

7、 **ProjectListSearchAction**

1.1.2 删除项目信息

请求处理流程如下,具体文件内容请查看前述内容:

1、 **ProjectListSearchAction**

2、 **projectlist.jsp**

3、 **ProjectFormLoadAction**

4、 **projectform.jsp**

5、 **ProjectFormSaveAction**

6、 **success.jsp or failed.jsp**

7、 **ProjectListSearchAction**

2. 附录

无。

3. 索引