

第1章 Struts2_day01

今日任务

- 使用 Struts2 完成客户列表显示的功能

教学导航

教学目标	
教学方法	案例驱动法

案例一：使用**Struts2**完成客户列表显示的功能.

1.1 案例需求

1.1.1 需求概述

CRM 系统中有客户的显示的功能，效果如图：



The screenshot shows a web-based CRM application interface. At the top, there is a header bar with the text "当前位置: 客户管理 > 客户列表". Below the header, there is a search/filter section with fields for "客户名称", "客户来源", "客户行业", "客户级别", and a "筛选" button. Below this is a table displaying client information. The table has columns: "客户名称", "客户级别", "客户来源", "客户行业", "电话", "手机", and "操作". There are three rows of data: 1. 张三, VIP客户, 电话营销, 电子商务, 01098745678, 13543568276, [修改, 删除]. 2. 李四, VIP客户, 电话营销, 酒店旅游, 110, 120, [修改, 删除]. 3. 王五, 普通客户, 网络营销, 教育培训, 111111, 222222, [修改, 删除]. At the bottom of the table, there is a pagination control with the text "共[7]条记录 [1]页, 每页显示 3 条 [前一页] [后一页] 到 [] 页 Go".

我们实际的开发中会使用 Struts2 作为 WEB 层的架构。

1.2 相关知识点

1.2.1 Struts2 框架的概述

Struts2 是一种基于 MVC 模式的轻量级 Web 框架，它自问世以来，就受到了广大 Web 开发者的关注，并广泛应用于各种企业系统的开发中。目前掌握 Struts2 框架几乎成为 Web 开发者的必备技能之一。接下来将针对 Struts2 的特点、安装以及执行流程等内容进行详细的讲解。



1.2.1.1 什么是 Struts2

Struts 2

[编辑](#)[同义词](#) [Struts2一般指Struts 2](#)

Struts2是一个基于MVC设计模式的Web应用框架，它本质上相当于一个servlet，在MVC设计模式中，Struts2作为控制器(Controller)来建立模型与视图的数据交互。Struts 2是Struts的下一代产品，是在 struts 1和WebWork的技术基础上进行了合并的全新的Struts 2框架。其全新的Struts 2的体系结构与Struts 1的体系结构差别巨大。Struts 2以WebWork为核心，采用拦截器的机制来处理用户的请求，这样的设计也使得业务逻辑控制器能够与ServletAPI完全脱离开，所以Struts 2可以理解为WebWork的更新产品。虽然从Struts 1到Struts 2有着太大的变化，但是相对于WebWork，Struts 2的变化很小。

在介绍 Struts2 之前，先来认识一下 Struts1。Struts1 是最早的基于 MVC 模式的轻量级 Web 框架，它能够合理的划分代码结构，并包含验证框架、国际化框架等多种实用工具框架。但是随着技术的进步，Struts1 的局限性也越来越多的暴露出来。为了符合更加灵活、高效的开发需求，Struts2 框架应运而生。

Struts2 是 Struts1 的下一代产品，是在 Struts1 和 WebWork 技术的基础上进行合并后的全新框架（WebWork 是由 OpenSymphony 组织开发的，致力于组件化和代码重用的 J2EE Web 框架，它也是一个 MVC 框架）。虽然 Struts2 的名字与 Struts1 相似，但其设计思想却有很大不同。实质上，Struts2 是以 WebWork 为核心的，它采用拦截器的机制来处理用户的请求。这样的设计也使得业务逻辑控制器能够与 ServletAPI 完全脱离开，所以 Struts2 可以理解为 WebWork 的更新产品。

Struts2 拥有优良的设计和功能，其优势具体如下：

- 项目开源，使用及拓展方便，天生优势。
- 提供 Exception 处理机制。
- Result 方式的页面导航，通过 Result 标签很方便的实现重定向和页面跳转。
- 通过简单、集中的配置来调度业务类，使得配置和修改都非常容易。
- 提供简单、统一的表达式语言来访问所有可供访问的数据。
- 提供标准、强大的验证框架和国际化框架。
- 提供强大的、可以有效减少页面代码的标签。
- 提供良好的 Ajax 支持。
- 拥有简单的插件，只需放入相应的 JAR 包，任何人都可以扩展 Struts2 框架，比如自定义拦截器、自定义结果类型、自定义标签等，为 Struts2 定制需要的功能，不需要什么特殊配置，并且可以发布给其他人使用。
- 拥有智能的默认设置，不需要另外进行繁琐的设置。使用默认设置就可以完成大多数项目程序开发所需要的功能。

上面列举的就是 Struts2 的一系列技术优势，只需对它们简单了解即可，在学习了后面的知识后，会慢慢对这些技术优势有更好的理解和体会。

那么除了 Struts2 之外，还有那些优秀的 WEB 层框架呢？

1.2.1.2 常见的 WEB 层的框架

- Struts2
- Struts1



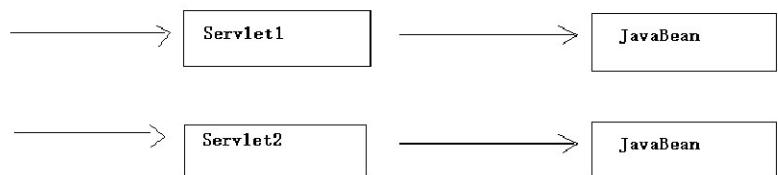
- Webwork
- SpringMVC

WEB 层框架都会有一个特点，就是基于前端控制器模式实现的。

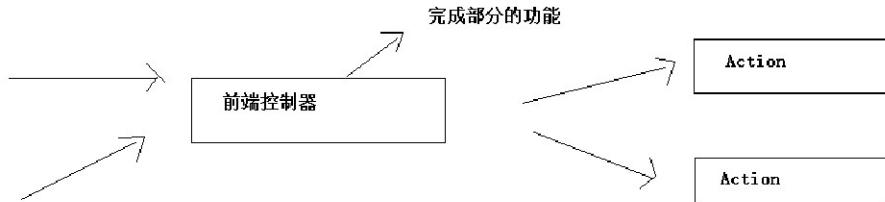
1.2.1.3 WEB 层的框架都会基于前端控制器的模式：

什么是前端控制器模式呢？我们来看下图，在图中传统方式的开发，有一次请求就会对应一个 Servlet。这样会导致出现很多 Servlet。而 Struts2 将所有的请求都先经过一个前端控制器，在前端控制器中实现框架的部分功能，剩下具体操作要提交到具体的 Action 中。那么所有的请求都会经过前端控制器，那用什么来实现前端控制器呢？过滤器就是最好的一个实现方式，因为需要所有的请求都可以被过滤器拦截，然后在过滤器中实现部分的功能。所以 Struts2 的前端控制器也是有过滤器来实现的。

传统方式开发：



前端控制器模型



了解这些之后，我们来进行一个 Struts2 的快速入门，来感受一下 Struts2 的优势吧。

1.2.2 Struts2 快速入门：

1.2.2.1 下载 Struts2 的开发包：

Struts2 的官网: <https://struts.apache.org/>



我们这里使用 struts-2.3.24 版本的 Struts2 为大家讲解，这个版本可以在当日授课资料中找到。下载好了以后需要进行解压。

1.2.2.2 解压 Struts2 的开发包：

解压后的目录结构如下：

名称	修改日期	类型	大小
apps	2016/7/20 8:55	文件夹	
docs	2015/5/3 12:56	文件夹	
lib	2015/5/3 12:56	文件夹	
src	2015/5/3 12:57	文件夹	
ANTLR-LICENSE.txt	2015/5/3 12:20	文本文档	2 KB
CLASSWORLDS-LICENSE.txt	2015/5/3 12:20	文本文档	2 KB
FREEMARKER-LICENSE.txt	2015/5/3 12:20	文本文档	3 KB
LICENSE.txt	2015/5/3 12:20	文本文档	10 KB
NOTICE.txt	2015/5/3 12:20	文本文档	1 KB
OGNL-LICENSE.txt	2015/5/3 12:20	文本文档	3 KB
OVAL-LICENSE.txt	2015/5/3 12:20	文本文档	12 KB
SITEMESH-LICENSE.txt	2015/5/3 12:20	文本文档	3 KB
XPP3-LICENSE.txt	2015/5/3 12:20	文本文档	3 KB
XSTREAM-LICENSE.txt	2015/5/3 12:20	文本文档	2 KB

Struts2 的文件夹结构

从图中可以看出，展示的是解压后的 Struts2.3.24 的目录结构，为了让大家对每个目录的内容和作用有一定的了解，接下来针对这些目录进行简单介绍，具体如下：

- **apps:** 该文件夹存用于存放官方提供的 Struts2 示例程序，这些程序可以作为学习者的学习资料，可为学习者提供很好的参照。各示例均为 war 文件，可以通过 zip 方式进行解压。
- **docs:** 该文件夹用于存放官方提供的 Struts2 文档，包括 Struts2 的快速入门、Struts2 的文档，以及 API 文档等内容。
- **lib:** 该文件夹用于存放 Struts2 的核心类库，以及 Struts2 的第三方插件类库。



- src: 该文件夹用于存放该版本 Struts2 框架对应的源代码。
有了 Struts2 的开发环境，接下来我们可以进行 Struts2 的开发了。

1.2.2.3 创建一个 web 工程引入相应 jar 包：

首先，需要我们创建一个 WEB 工程，引入相关的 jar 包文件。引入哪些 jar 包呢？将 struts-2.3.24 框架目录中的 lib 文件夹打开，得到 Struts2 开发中可能用到的所有 JAR 包（此版本有 107 个 JAR 包）。实际的开发中，我们根本不用引入这么多的 jar 包。

要进行 struts2 的基本的开发，可以参考 struts-2.3.24 中的 apps 下的一些示例代码，其中 struts2-blank.war 是一个 struts2 的空的工程。我们只需要将 struts2-blank.war 解压后进入到 WEB-INF 下的 lib 中查看。

名称	修改日期	类型	大小
asm-3.3.jar	2013/11/23 17:55	Executable Jar File	43 KB
asm-commons-3.3.jar	2013/11/23 17:55	Executable Jar File	38 KB
asm-tree-3.3.jar	2013/11/23 17:55	Executable Jar File	21 KB
commons-fileupload-1.3.1.jar	2014/2/19 16:21	Executable Jar File	68 KB
commons-io-2.2.jar	2013/11/23 17:55	Executable Jar File	170 KB
commons-lang3-3.2.jar	2014/1/2 21:45	Executable Jar File	376 KB
freemarker-2.3.22.jar	2015/4/3 7:09	Executable Jar File	1,271 KB
javassist-3.11.0.GA.jar	2013/11/23 17:55	Executable Jar File	600 KB
log4j-api-2.2.jar	2015/4/19 12:04	Executable Jar File	131 KB
log4j-core-2.2.jar	2015/4/19 12:04	Executable Jar File	808 KB
ognl-3.0.6.jar	2013/11/23 17:55	Executable Jar File	223 KB
struts2-core-2.3.24.jar	2015/5/3 12:25	Executable Jar File	813 KB
xwork-core-2.3.24.jar	2015/5/3 12:23	Executable Jar File	661 KB

这些包就是 struts2 的基本的开发包了，那么这些包都是什么含义呢？

Struts2 项目依赖的基础 JAR 包说明

文件名	说明
asm-3.3.jar	操作 java 字节码的类库
asm-commons-3.3.jar	提供了基于事件的表现形式
asm-tree-3.3.jar	提供了基于对象的表现形式
struts2-core-2.3.24.jar	Struts2 框架的核心类库
xwork-core-2.3.24.jar	WebWork 核心库，Struts2 的构建基础
ognl-3.0.6.jar	对像图导航语言(Object Graph Navigation Language)，struts2 框架通过其读写对象的属性
freemarker-2.3.22.jar	Struts2 标签模板使用的类库
javassist-3.11.0.GA.jar	javaScript 字节码解释器
commons-fileupload-1.3.1.jar	Struts2 文件上传组件依赖包
commons-io-2.2.jar	Struts2 的输入输出，传文件依赖的 jar
commons-lang-2.4.jar	包含一些数据类型工具，是对 java.lang 包的增强
log4j-api-2.2.jar	Struts2 的日志管理组件依赖包的 api
log4j-core-2.2.jar	Struts2 的日志管理组件依赖包



从表可以看出，此版本的 Struts2 项目所依赖的基础 JAR 包共 13 个。Struts2 根据版本的不同所依赖的基础 JAR 包可能不完全相同，不过基本上变化不大，读者可以视情况而定。

需要注意的是，通常使用 Struts2 的 Web 项目并不需要利用到 Struts2 的全部 JAR 包，因此没有必要一次将 Struts2 的 lib 目录下的全部 JAR 包复制到 Web 项目的 WEB-INF/lib 路径下，而是根据需要，再添加相应的 JAR 包。

那么 Struts2 的基本 jar 包已经引入完成了，我们使用 Struts2 都是从页面发起请求到服务器，再由服务器处理请求，响应到页面的这个过程。接下来我们就从页面开发进行 Struts2 的开发吧。

1.2.2.4 创建一个页面：放置一个链接。

首先需要在 WebContent 下创建一个目录 demo1，在 demo1 下创建一个新的 jsp。在 jsp 中编写一个 Action 的访问路径。

```
<h1>Struts2 的入门案例</h1>
<a href="#">#{pageContext.request.contextPath }/strutsDemo1.action">访问 Struts2 的
Action.</a>
```

点击该链接，需要提交请求到服务器的 Action。那么接下来需要编写 Action 去处理请求。

1.2.2.5 编写一个 Action：

在 src 下创建一个包 cn.itcast.struts2.action，在该包下新建一个 StrutsDemo1 的类。在这个类中编写一个公有的，返回值为 String 类型的方法，这个方法的名称叫做 execute，且该方法没有任何参数。（因为这个方法最终要被反射执行）

```
public class StrutsDemo1 {

    /**
     * 提供一个默认的执行的方法：execute
     */
    public String execute(){
        System.out.println("StrutsDemo1 中的 execute 执行了... ");
        return null;
    }
}
```

Action 类编写好了以后，Struts2 框架如何识别它就是一个 Action 呢，那么我们需要对 Action 类进行配置。

1.2.2.6 完成 Action 的配置：

这个时候，我们还需要观察 apps 中的示例代码，在 WEB-INF 的 classes 中，有一个名称为 struts.xml 的文件，这个文件就是 struts2 的配置文件。

我们在开发中需要将 struts.xml 文件引入到工程的 src 下，因为 src 下内容发布到 web 服务器中就是 WEB-INF 下的 classes 中。将 struts.xml 中的原有的内容删除掉就，然后配置上自己编写的 Action 类就可以了。



配置内容如下：里面的具体的标签，我们会在后面的地方详细介绍。

```
<struts>
    <!-- 配置一个包:package -->
    <package name="demo1" extends="struts-default" namespace="/">
        <!-- 配置 Action -->
        <action name="strutsDemo1" class="cn.itcast.struts2.action.StrutsDemo1">

            </action>
        </package>
    </struts>
```

Action 类已经配置好了，配置好了以后大家考虑一下，现在是否可以执行呢？其实现在还不行，因为之前我们介绍过，WEB 层的框架都有一个特点就是基于前端控制器的模式，这个前端控制器是由过滤器实现的，所以我们需要配置 Struts2 的核心过滤器。这个过滤器的名称是 StrutsPrepareAndExecuteFilter。

1.2.2.7 配置核心过滤器：

Struts2 框架要想执行，所有的请求都需要经过这个前端控制器（核心过滤器），所以需要配置这个核心过滤器。因为这个过滤器完成了框架的部分的功能。那么我们接下来对过滤器进行配置。我们打开 web.xml，在 web.xml 中进行如下配置。

```
<!-- 配置 Struts2 的核心过滤器：前端控制器 -->
<filter>
    <filter-name>struts2</filter-name>

    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

那么到这，我们程序就可以执行了，但是到了 Action 以后，页面并没有跳转，只是会在控制台输出 Action 中的内容，往往我们在实际的开发中，处理了请求以后，还需要进行页面的跳转，如何完成 Struts2 的页面的跳转呢？

这个时候我们需要修改 Action 类中的 execute 方法的返回值了，这个方法返回了一个 String 类型，这个 String 类型的值就是一个逻辑视图（逻辑视图：相当于对一个真实的页面，取了一个别名。），那么我们来修改一个 Action 类。

1.2.2.8 修改 Action，将方法设置一个返回值：

修改 Action 中的 execute 方法的返回值，我们先任意给其返回一个字符串，比如返回一个 success 的字符串。这个字符串就作为一个逻辑视图名称。



```
public class StrutsDemo1 {  
  
    /**  
     * 提供一个默认的执行的方法:execute  
     */  
  
    public String execute(){  
        System.out.println("StrutsDemo1 中的 execute 执行了...");  
        return "success";  
    }  
}
```

返回一个 success 的字符串了，这个 success 的字符串又怎么能代表一个页面呢？这个时候我们又要对 struts2 进行配置了。这个时候需要修改 struts.xml，对 Action 的配置进行完善。

1.2.2.9 修改 struts.xml

打开 struts.xml 文件，对<action>标签进行完善，在<action>标签中配置一个<result>标签，这个标签中的 name 属性就是之前方法返回的那个字符串的逻辑视图名称 success。标签内部就是跳转的页面。

```
<struts>  
  
    <!-- 配置一个包:package -->  
    <package name="demo1" extends="struts-default" namespace="/">  
  
        <!-- 配置 Action -->  
        <action name="strutsDemo1" class="cn.itcast.struts2.action.StrutsDemo1">  
            <!-- 配置结果页面的跳转 -->  
            <result name="success"/>/demo1/demo2.jsp</result>  
        </action>  
    </package>  
</struts>
```

到这，我们的整个程序就执行完毕了。我们可以启动服务器并且测试项目。

打开页面：



Struts2的入门案例

访问Struts2的Action 点击链接

页面跳转：



成功跳转页面！！！

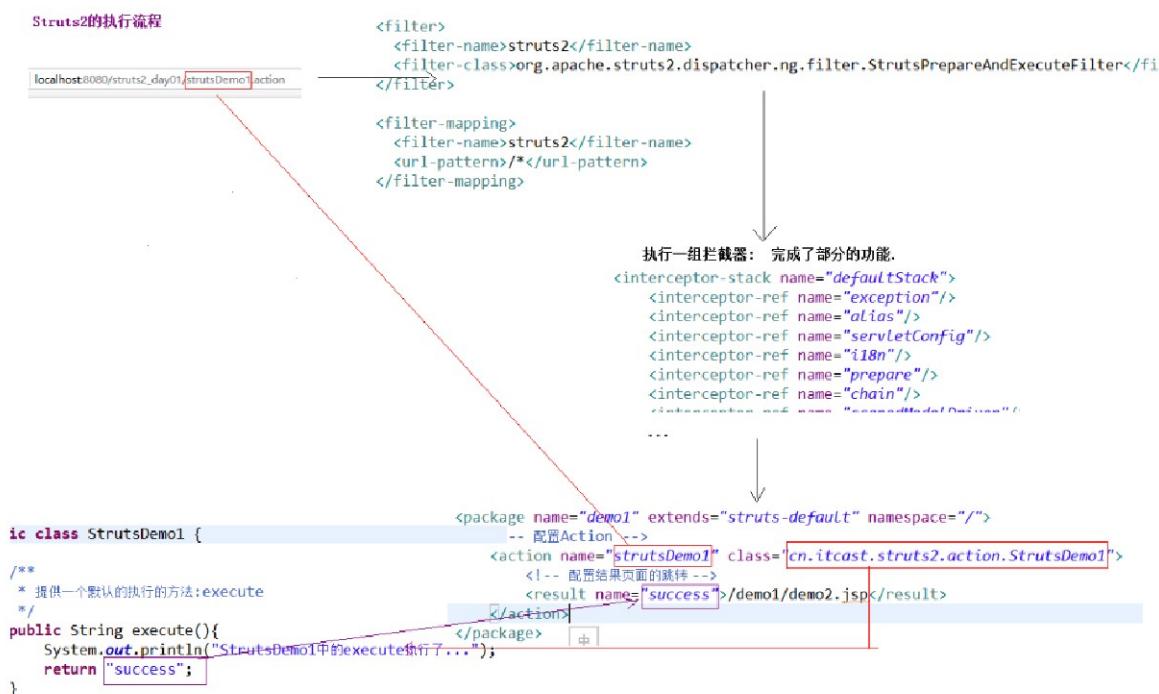
到这里 Struts2 的入门案例已经编写完成了，那么我们来总结下 Struts2 的整个流程。



1.2.3 Struts2 开发流程分析

1.2.3.1 Struts2 的执行流程:

从客户端发送请求过来 先经过前端控制器（核心过滤器 StrutsPrepareAndExecuteFilter）过滤器中执行一组拦截器（一组拦截器 就会完成部分功能代码），到底哪些拦截器执行了呢，在 Struts2 中定义很多拦截器，在其默认栈中的拦截器会得到执行，这个我们可以通过断点调试的方式测试，拦截器执行完成以后，就会执行目标 Action，在 Action 中返回一个结果视图，根据 Result 的配置进行页面的跳转。

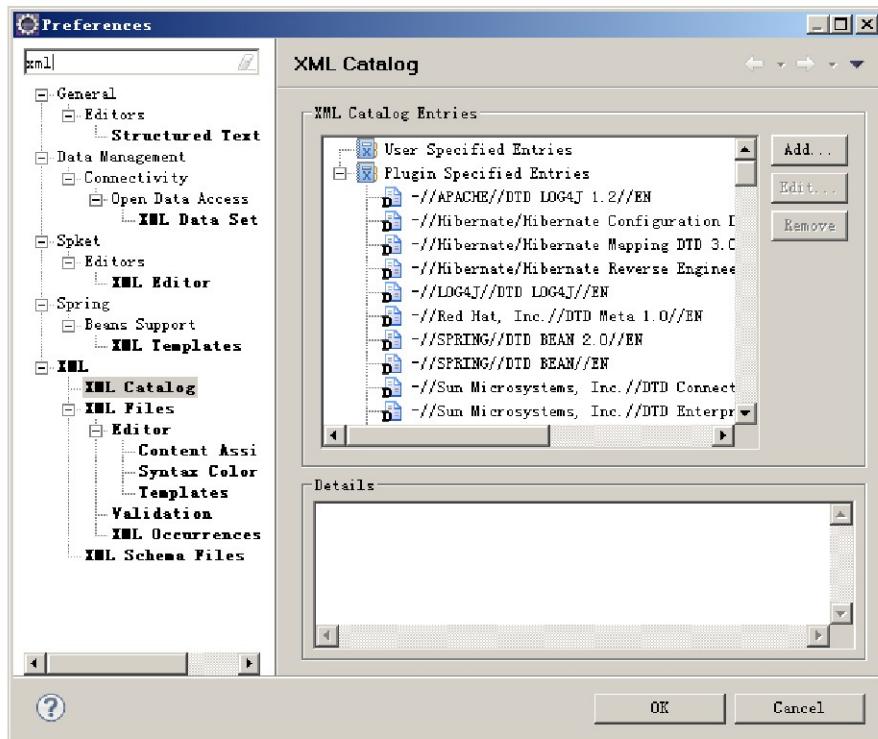


以上内容是 Struts2 的执行流程，但是可能大家在编写测试的时候会遇到一些问题，比如在编写 XML 配置文件的时候没有提示。

1.2.3.2 配置 struts.xml 中的提示(在不联网情况下)

开发过程中如果可以上网，struts.xml 会自动缓存 dtd，提供提示功能。如果不能够上网，则需要我们手动配制本地 dtd，这样才能够使 struts.xml 产生提示。具体配置方法如下：

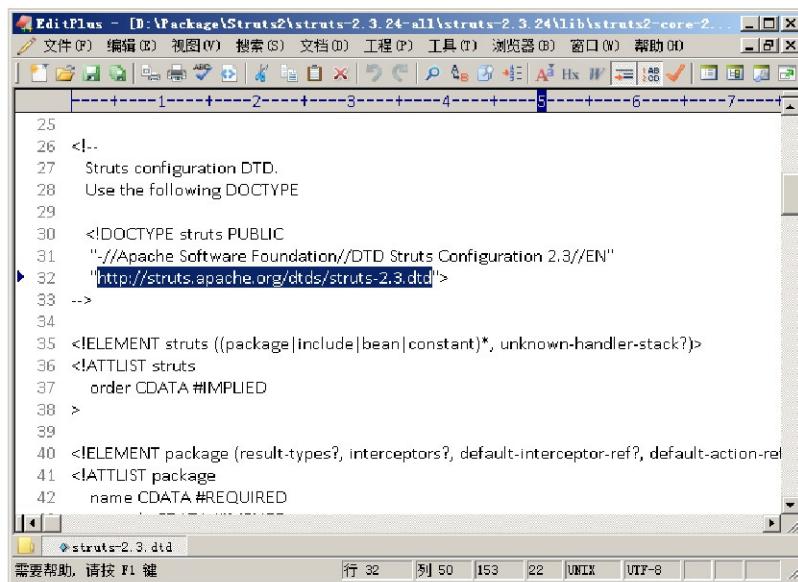
1.首先，在 Eclipse 中，依次点击工具栏中的 window 和下方的 Preferences 弹出对话框。然后在左侧的搜索框中输入 xml，显示出所有与 xml 有关的选项后，点击 XML Catalog，会出现如图 1-11 所示界面。



XML Catalog 窗口

2. 接下来在已下载的 Struts2 解压包中的 lib 包中找到其核心包 struts2-core-2.3.24.jar，使用解压工具将其解压成文件夹形式。解压后，我们会看到文件夹中有几个以.dtd 结尾的文件。我们所使用的是 struts-2.3.dtd。

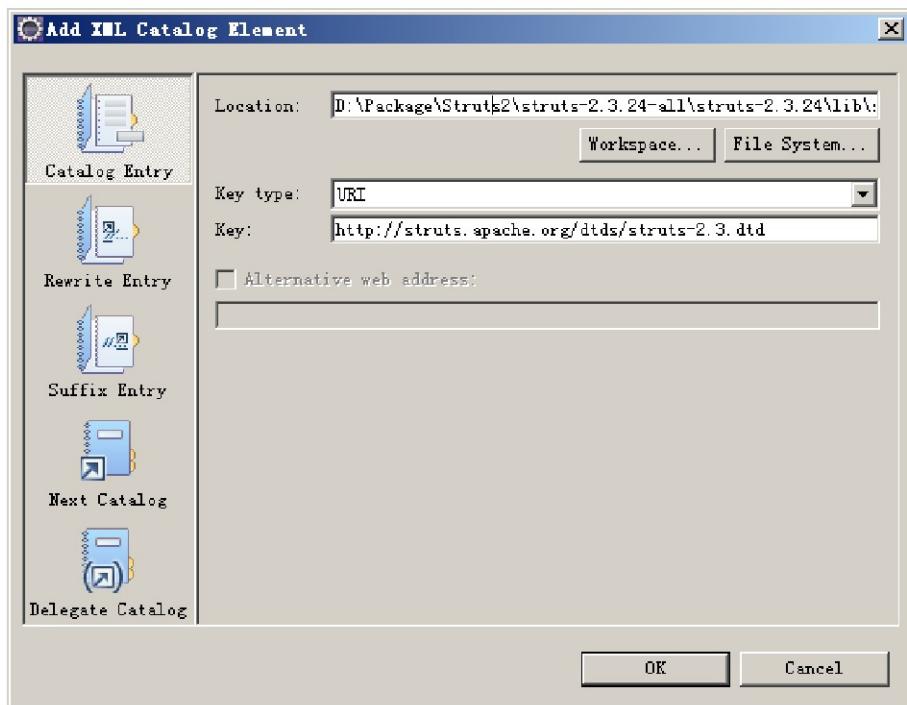
3. 将此 dtd 使用 EditPlus 等文本工具打开后，找到图中选中内容，将其 http 地址复制。如图 1-12 所示。



```
25
26 <!--
27 Struts configuration DTD.
28 Use the following DOCTYPE
29
30 <!DOCTYPE struts PUBLIC
31 "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
32 "<http://struts.apache.org/dtds/struts-2.3.dtd">
33 -->
34
35 <!ELEMENT struts ((package|include|bean|constant)*, unknown-handler-stack?)>
36 <!ATTLIST struts
37   order CDATA #IMPLIED
38 >
39
40 <!ELEMENT package (result-types?, interceptors?, default-interceptor-ref?, default-action-ref?,
41 <!ATTLIST package
42   name CDATA #REQUIRED
```

struts-2.3.dtd 文件

4. 点击 Eclipse 中弹出对话框中右侧的 Add 按钮，此时会弹出 Add XML Catalog Element 界面。点击 File System 按钮，找到本地刚才解压文件夹中的 struts-2.3.dtd，然后将界面中的 Key type 改为 URI，并将刚才复制的地址黏贴到 Key 中。如图所示。



Add XML Catalog Element 窗口

在图中点击 OK 后，关闭已打开的 struts.xml，然后再重新打开 struts.xml，此时再编写 struts.xml 内容的时候，就会有提示了。

那么提示已经会配置了，也就可以进行 Struts2 的开发了，开发过程中需要自己配置 struts.xml 文件，在这个文件中就有很多常见的配置。接下来我们就来学习 Struts2 的常见配置。

1.2.4 Struts2 的常见配置

通过前面的学习，我们已对 Struts2 框架有了一定的了解，但是对于各知识点的细节，还需要进一步地学习。接下来，将针对 Struts2 中 struts.xml 文件的配置等内容进行详细的讲解。在学习具体详细的配置之前，要对 Struts2 的配置文件的加载顺序有一定的了解，这样对后面学习 Struts2 的配置都是有帮助的。

1.2.4.1 Struts2 的配置文件的加载顺序：

每次从客户端发送请求到服务器都要先经过 Struts2 的核心过滤器 StrutsPrepareAndExecuteFilter，这个过滤器有两个功能：预处理和执行。在预处理中主要就是来加载配置文件的。对应的是过滤器中的 init 方法，而执行是用来执行一组拦截器完成部分功能的，对应的是过滤器的 doFilter 方法。所以我们如果要去了解 Struts2 的配置文件的加载顺序，那么我们需要查询过滤器的 init 方法。



```
public void init(FilterConfig filterConfig) throws ServletException {
    InitOperations init = new InitOperations();
    Dispatcher dispatcher = null;
    try {
        FilterHostConfig config = new FilterHostConfig(filterConfig);
        init.initLogging(config);
        dispatcher = init.initDispatcher(config);
        init.initStaticContentLoader(config, dispatcher);

        prepare = new PrepareOperations(dispatcher);
        execute = new ExecuteOperations(dispatcher);
        this.excludedPatterns = init.buildExcludedPatternsList(dispatcher);

        postInit(dispatcher, filterConfig);
    } finally {
        if (dispatcher != null) {
            dispatcher.cleanUpAfterInit();
        }
        init.cleanup();
    }
}
```

在 init 方法中，调用了 init 的 initDispatcher 的方法来加载配置文件，进入到该代码中：

```
/*
 * Creates and initializes the dispatcher
 */
public Dispatcher initDispatcher( HostConfig filterConfig ) {
    Dispatcher dispatcher = createDispatcher(filterConfig);
    dispatcher.init();
    return dispatcher;
}
```

我们会发现这个方法又调用了 dispatcher 的 init 方法。进入 init 方法内部：

```
public void init() {
    if (configurationManager == null) {
        configurationManager = createConfigurationManager(DefaultBeanSelectionProvider.DEFAULT_BEAN_NAME);
    }

    try {
        init_FileManager();
        init_DefaultProperties(); // [1]
        init_TraditionalXmlConfigurations(); // [2]
        init_LegacyStrutsProperties(); // [3]
        init_CustomConfigurationProviders(); // [5]
        init_FilterInitParameters(); // [6]
        init_AliasStandardObjects(); // [7]

        Container container = init_PreloadConfiguration();
        container.inject(this);
        init_CheckWebLogicWorkaround(container);
    }
}
```

这一系列的代码就是用来加载 Struts2 的配置文件的。

```
init_DefaultProperties(); // [1]
```

加载 org.apache.struts.default.properties 配置的是 struts2 的所有常量。

```
init_TraditionalXmlConfigurations(); // [2]
```

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

```
init_LegacyStrutsProperties(); // [3]
```

加载用户自定义 struts.properties

```
init_CustomConfigurationProviders(); // [5]
```

加载用户配置的提供对象

```
init_FilterInitParameters(); // [6]
```

加载 web.xml

```
init_AliasStandardObjects(); // [7]
```

加载标准对象。

根据上面的代码我们可以得出配置文件的加载顺序如下

```
* default.properties  
* struts-default.xml  
* struts-plugin.xml  
* struts.xml          ----- 配置 Action 以及常量. (*****)  
* struts.properties   ----- 配置常量  
* web.xml             ----- 配置核心过滤器及常量.
```

前三个配置文件我们不用关心，是 Struts2 内部的配置文件，我们无法修改，能修改的文件就是 struts.xml, struts.properties, web.xml 配置文件。这几个配置文件的加载是有一定的顺序的。这三个配置文件都可以修改 Struts2 的常量的值，要记住的是，后加载配置文件中常量的值会将先加载的配置文件中常量的值给覆盖。

知道了这些我们就可以来看 Struts2 的详细的配置了。首先来看 Action 的配置。

1.2.4.2 Action 的配置：

Struts2 框架的核心配置文件是 struts.xml 文件，该文件主要用来配置 Action 和请求的对应关系。

【<package>的配置】

Struts2 框架的核心组件是 Action 和拦截器，它使用包来管理 Action 和拦截器。每个包就是多个 Action、多个拦截器、多个拦截器引用的集合。在 struts.xml 文件中， package 元素用于定义包配置，每个 package 元素定义了一个包配置。 package 元素的常用属性，如表所示。

package 元素的常用属性

属性	说明
name	必填属性，它指定该包的名字，此名字是该包被其他包引用的 key。
namespace	可选属性，该属性定义该包的命名空间。
extends	可选属性，它指定该包继承自其他包。继承其他包，可以继承其他包中的 Action 定义、拦截器定义等。
abstract	可选属性，它指定该包是否是一个抽象包，抽象包中不能包含 Action 定义。

表中就是 package 元素的常用属性，其中，在配置包时，必须指定 name 属性，就是包的标识。除此之外，还可以指定一个可选的 extends 属性， extends 属性值必须是另一个包的 name 属性值，但该属性值通常都设置为 struts-default，这样该包中的 Action 就具有了 Struts2 框架默认的拦截器等功能了。除此之外，Struts2 还提供了一种所谓的抽象包，抽象包不能包含 Action 定义。为了显式指定一个包是抽象包，可以为该 package 元素增加 abstract="true" 属性。

在 package 中还有 namespace 的配置， namespace 属性与 action 标签的 name 属性共同决定了访



问路径。namespace 有如下三种配置。

- 默认名称空间 : 默认的名称空间就是 namespace=""
- 跟名称空间 : 跟名称空间就是 namespace="/"
- 带名称的名称空间 : 带名称的名称空间就是 namespace="/demo1"

【Action 的配置】

Action 映射是框架中的基本“工作单元”。Action 映射就是将一个请求的 URL 映射到一个 Action 类，当一个请求匹配某个 Action 名称时，框架就使用这个映射来确定如何处理请求。在 struts.xml 文件中，通过<action>元素对请求的 Action 和 Action 类进行配置。

<action>元素中共有 4 个属性，这 4 个属性的说明如表所示。

action 元素属性说明

属性	说明
name	必填属性，标识 Action，指定了 Action 所处理的请求的 URL。
class	可选属性，指定 Action 对应 Action 类。
method	可选属性，指定请求 Action 时调用的方法。
converter	可选属性，指定类型转换器的类。

其中 name 属性和 namespace 属性共同决定了访问路径，class 对应的是 Action 类的全路径。Method 指定了执行 Action 的那个方法，默认是 execute 方法。

基本的 Struts2 的配置我们已经了解了，在实际的开发中我们需要大量的用到 Struts2 的常量，那么我们接下来学习一下 Struts2 的常量。

1.2.4.3 Struts2 常量的配置：

Struts2 的这些常量大多在默认的配置文件中已经配置好，但根据用户需求的不同，开发的要求也不同，可能需要修改这些常量值，修改的方法就是在配置文件对常量进行重新配置。

Struts2 常量配置共有 3 种方式，分别如下：

- 在 struts.xml 文件中使用<constant>元素配置常量。
- 在 struts.properties 文件中配置常量。
- 在 web.xml 文件中通过< init-param>元素配置常量。

为了让大家更好地掌握这 3 种 Struts2 常量配置的方式，接下来分别对它们进行讲解，具体如下。

1、在 struts.xml 文件中通过<constant>元素配置常量

在 struts.xml 文件中通过<constant>元素来配置常量，是最常用的方式。在 struts.xml 文件中通过<constant.../>元素来配置常量时，需要指定两个必填的属性 name 和 value。

- name: 该属性指定了常量的常量名。
- value: 该属性指定了常量的常量值。

在 struts.xml 文件中配置的示例代码如下：

```
<struts>
    <!-- 设置默认编码集为 UTF-8 -->
    <constant name="struts.i18n.encoding" value="UTF-8" />
    <!-- 设置使用开发模式 -->
    <constant name="struts.devMode" value="true" />
```



```
</struts>
```

在上述示例代码中，配置了常量 struts.i18n.encoding 和 struts.devMode，用于指定 Struts2 应用程序的默认编码集为 UTF-8，并使用开发模式。值得一提的是，struts.properties 文件能配置的常量都可以在 struts.xml 文件中用<constant>元素来配置。

2、在 struts.properties 文件中配置常量

struts.properties 文件是一个标准的 properties 文件，其格式是 key-value 对，即每个 key 对应一个 value，key 表示的是 Struts2 框架中的常量，而 value 则是其常量值。在 struts.properties 文件中配置常量的方式，具体如下所示：

```
### 设置默认编码集为 UTF-8
struts.i18n.encoding=UTF-8

### 设置 action 请求的扩展名为 action 或者没有扩展名
struts.action.extension=action,,

### 设置不使用开发模式
struts.devMode=false

### 设置不开启动态方法调用
struts.enable.DynamicMethodInvocation=false
```

在上述代码片段中，“=”号左边的是 key，右边的是每个 key 对应的 value，另外，代码片段中的“###”表示的是 properties 文件中的注释信息，用于解释说明。

需要注意的是，和 struts.xml 文件一样，struts.properties 文件也应存放于 WEB-INF/classes 路径下。

3、在 web.xml 文件中通过初始化参数配置常量

在 web.xml 文件中配置核心过滤器 StrutsPrepareAndExecuteFilter 时，通过初始化参数来配置常量。通过<filter>元素的<init-param>子元素指定，每个<init-param>元素配置了一个 Struts2 常量。在 web.xml 文件中通过初始化参数配置常量方式，具体如以下代码片段所示：

```
<filter>
    <!-- 指定 Struts2 的核心过滤器 -->
    <filter-name>struts2</filter-name>
    <filter-class>
        org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
    <!--通过 init-param 元素配置 Struts2 常量，配置默认编码集为 UTF-8 -->
    <init-param>
        <param-name>struts.i18n.encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
```

在上述 web.xml 文件的代码片段中，当配置 StrutsPrepareAndExecuteFilter 时，还通过<init-param>子元素配置了常量 struts.i18n.encoding，指定其值为 UTF-8。需要注意的是，在 web.xml 文件中配置常量时，<init-param>标签必须放在<filter>标签下。

Struts2 所支持的常量数量众多，在 struts2-core-2.3.24.jar 压缩文件的 org/apache/struts2 路径下有一个 default.properties 文件，该文件里为 Struts2 的所有常量都指定了默认值，读者可以通过查看该文件来了解 Struts2 所支持的常量。

之前我们就已经介绍过了 Struts2 的配置文件的加载顺序，**后加载的配置文件的常量的值会覆盖先加载的配置文件中常量的值。所以这个地方大家要注意。**



在实际的开发中我们更习惯使用 struts.xml 修改 struts2 的常量。但是在实际开发中还会有一个问题，就是如果一个项目是团队开发的，也就是很多人开发的，那么团队中的很多人就都需要去修改 struts.xml。那么最后在项目整合的时候就会很麻烦。所以 Struts2 中也支持分模块开发的配置。

1.2.4.4 分模块开发的配置

在实际开发中，我们通常很多人都需要修改同一个配置文件就是 struts.xml。因为这个文件是 Struts2 的核心配置文件。而且这个文件一旦改错了一点，那么会导致整个项目都会出现问题，所以 Struts2 提供了<include>标签解决这个问题。

<include>元素用来在一个 struts.xml 配置文件中包含其他的配置文件，包含配置体现的是软件工程中的“分而治之”原则。Struts2 允许将一个配置文件分解成多个配置文件，从而提高配置文件的可读性。Struts2 默认只加载 WEB-INF/classes 下的 struts.xml 文件，但一旦通过多个 xml 文件来配置 Action，就必须通过 struts.xml 文件来包含其他配置文件。

为了让大家更直观地理解如何在 struts.xml 文件中进行包含配置，接下来通过一段示例代码来说明，具体如下：

```
<struts>
    <!-- 包含了 4 个配置文件 -->
    <!-- 不指定路径默认在 src 下时的方式 -->
    <include file="struts-shop.xml"/>
    <include file="struts-user.xml"/>
    <include file="struts-shoppingcart.xml"/>
    <!-- 配置文件在具体包中时的方式 -->
    <include file="cn/itcast/action/struts-product.xml">
</struts>
```

在上述代码片段中，struts.xml 文件包含了 4 个配置文件，这 4 个配置文件都包含在<include>元素中。配置<include>元素时，指定了一个必需的 file 属性，该属性指定了被包含配置文件的文件名。上述 include 元素的 file 属性中，前 3 个没有指定文件所在路径时，表示该文件在项目的 src 路径下，如果配置文件在具体的包中，那么引入配置文件时，需要包含文件所在包的路径。

需要注意的是，每一个被包含的配置文件都是标准的 Struts2 配置文件，一样包含 DTD 信息、Struts2 配置文件的根元素等信息。通过将 Struts2 的所有配置文件都放在 Web 项目的 WEB-INF/classes 路径下，struts.xml 文件包含了其他的配置文件，在 Struts2 框架自动加载 struts.xml 文件时，完成加载所有的配置信息。

Action 的基本配置以及熟悉了，那么通过配置 Struts2 框架就可以找到具体的 Action 类了，那么 Action 类又要如何编写呢？接下来学习 Action 类的编写的方式。

1.2.5 Struts2 的 Action 的访问：

在 Struts2 的应用开发中，Action 作为框架的核心类，实现对用户请求的处理，Action 类被称为业务逻辑控制器。一个 Action 类代表一次请求或调用，每个请求的动作都对应于一个相应的 Action 类，一个 Action 类是一个独立的工作单元。也就是说，用户的每次请求，都会转到一个相应的 Action 类里面，由这个 Action 类来进行处理。简而言之，Action 就是用来处理一次用户请求的对象。

实现 Action 控制类共有 3 种方式，接下来，分别对它们进行讲解，具体如下。



1.2.5.1 Action 的编写的方式:

【Action 的是一个 POJO 的类】

在 Struts2 中，Action 可以不继承特殊的类或不实现任何特殊的接口，仅仅是一个 POJO。POJO 全称 Plain Ordinary Java Object（简单的 Java 对象），只要具有一部分 getter/setter 方法的那种类，就可以称作 POJO。一般在这个 POJO 类中，要有一个公共的无参的构造方法（采用默认的构造方法就可以）和一个 execute()方法。定义格式如下：

```
public class ActionDemo1 {  
  
    public String execute(){  
        System.out.println("ActionDemo1 执行了...");  
        return null;  
    }  
}
```

execute()方法的要求如下：

- 方法的权限修饰符为 public。
- 返回一个字符串，就是指示的下一个页面的 Result。
- 方法没有参数。

也就是说，满足上述要求的 POJO 都可算作是 Struts2 的 Action 实现。

通常会让开发者自己编写 Action 类或者实现 Action 接口或者继承 ActionSupport 类。

【Action 类实现一个 Action 的接口】

为了让用户开发的 Action 类更规范，Struts2 提供一个 Action 接口，用户在实现 Action 控制类时，可以实现 Struts2 提供的这个 Action 接口。

Action 接口定义了 Struts 的 Action 处理类应该实现的规范，Action 接口中的具体代码如下所示。

```
public class ActionDemo2 implements Action{  
  
    @Override  
    public String execute() throws Exception {  
        System.out.println("ActionDemo2 执行了...");  
        return null;  
    }  
}
```

从上述代码中可以看出，Action 接口位于 com.opensymphony.xwork2 包中。这个接口里只定义了一个 execute()方法，该接口的规范规定了 Action 处理类应该包含一个 execute()方法，该方法返回一个字符串。除此之外，该接口还定义了 5 个字符串常量，它们的作用是统一 execute()方法的返回值。

Action 接口中提供了 5 个已经定义的常量如下：

* SUCCESS	:success，代表成功。
* NONE	:none，代表页面不跳转
* ERROR	:error，代表跳转到错误页面。
* INPUT	:input，数据校验的时候跳转的路径。
* LOGIN	:login，用来跳转到登录页面。

由于 Xwork 的 Action 接口简单，为开发者提供的帮助较小，所以在实际开发过程中，Action 类



很少直接实现 Action 接口，通常都是从 ActionSupport 类继承。

【Action 类继承 ActionSupport 类】（推荐）

```
public class ActionDemo3 extends ActionSupport {  
  
    @Override  
    public String execute() throws Exception {  
        System.out.println("ActionDemo3 执行了...");  
        return NONE;  
    }  
}
```

ActionSupport 类本身实现了 Action 接口，是 Struts2 中默认的 Action 接口的实现类，所以继承 ActionSupport 就相当于实现了 Action 接口。ActionSupport 类还实现了 Validateable、ValidationAware、TextProvider、LocaleProvider 和 Serializable 等接口，来为用户提供更多的功能。

ActionSupport 类中提供了许多默认方法，这些默认方法包括获取国际化信息的方法、数据校验的方法、默认的处理用户请求的方法等。实际上，ActionSupport 类是 Struts2 默认的 Action 处理类，如果让开发者的 Action 类继承该 ActionSupport 类，则会大大简化 Action 的开发。

Action 的类已经会编写了，如果要执行这个 Action，可以通过前面的配置来完成，但是之前的方式有一个缺点，就是多次请求不能对应同一个 Action，因为实际的开发中，一个模块的请求通常由一个 Action 类处理就好了，否则会造成 Action 类过多。那么接下来我们讲一下 Action 访问的一些细节。

1.2.5.2 Action 的访问：

Action 的访问不是难题，因为之前已经访问过了，但是出现一个问题一次请求现在对应一个 Action，那么如果请求很多对应很多个 Action。现在要处理的问题就是要让一个模块的操作提交到一个 Action 中。

其实我们学过在<action>的标签中有一个属性 method，通过 method 的配置来指定 Action 中的某个方法执行。

【解决 Action 的访问的问题的方式一：通过配置 method 属性完成】

编写页面：

```
<h3>客户的管理</h3>  
<a href="${ pageContext.request.contextPath }/saveCustomerAction.action">添加客户</a> <br/>  
<a href="${ pageContext.request.contextPath }/updateCustomerAction.action">修改客户</a> <br/>  
<a href="${ pageContext.request.contextPath }/deleteCustomerAction.action">删除客户</a> <br/>  
<a href="${ pageContext.request.contextPath }/findCustomerAction.action">查询客户</a> <br/>
```

编写 Action：

```
public class CustomerAction extends ActionSupport {  
  
    public String save() {
```



```
System.out.println("CustomerAction 中 save 方法执行了...");  
    return NONE;  
}  
  
public String update(){  
    System.out.println("CustomerAction 中 update 方法执行了...");  
    return NONE;  
}  
  
public String delete(){  
    System.out.println("CustomerAction 中 delete 方法执行了...");  
    return NONE;  
}  
  
public String find(){  
    System.out.println("CustomerAction 中 find 方法执行了...");  
    return NONE;  
}  
}
```

配置 Action:

```
<package name="demo3" extends="struts-default" namespace="/">  
    <action  
        name="saveCustomerAction"  
        class="cn.itcast.struts2.demo3.CustomerAction" method="save"></action>  
        <action  
            name="updateCustomerAction"  
            class="cn.itcast.struts2.demo3.CustomerAction" method="update"></action>  
            <action  
                name="deleteCustomerAction"  
                class="cn.itcast.struts2.demo3.CustomerAction" method="delete"></action>  
                <action  
                    name="findCustomerAction"  
                    class="cn.itcast.struts2.demo3.CustomerAction" method="find"></action>  
    </package>
```

但是这种方式我们会发现，同一个 Action 类就被配置了很多次，只是修改了后面的 method 的值。那么能不能配置简单化呢？也就是一个 Action 类，只配置一次就好了？这个时候我们就需要使用通配符的配置方式了。

【解决 Action 的访问的问题的方式二：通过通配符的配置完成】

编写页面:

```
<h3>联系人的管理</h3>  
    <a href="${ pageContext.request.contextPath }/linkman_save.action">添加联系人</a>  
    <br/>  
    <a href="${ pageContext.request.contextPath }/linkman_update.action">修改联系人</a>  
    <br/>  
    <a href="${ pageContext.request.contextPath }/linkman_delete.action">删除联系人</a>  
    <br/>  
    <a href="${ pageContext.request.contextPath }/linkman_find.action">查询联系人</a>  
    <br/>
```

编写 Action:



```
public class LinkManAction extends ActionSupport{  
  
    public String save(){  
        System.out.println("保存联系人...");  
        return NONE;  
    }  
  
    public String update(){  
        System.out.println("修改联系人...");  
        return NONE;  
    }  
  
    public String delete(){  
        System.out.println("删除联系人...");  
        return NONE;  
    }  
  
    public String find(){  
        System.out.println("查询联系人...");  
        return NONE;  
    }  
}
```

配置 Action:

```
<!-- 通配符的配置 -->  
<action name="linkman_*" class="cn.itcast.struts2.demo3.LinkManAction"  
method="{1}"></action>
```

在<action>的 name 属性中使用的*代表任意字符，method 中的{1}代表 name 属性中的出现的第一个*所代替的字符。

这个时候我们就只配置一个就可以了，在上述代码中，当客户端发送/linkman_save.action 这样的请求时，action 元素的 name 属性就被设置成 linkman_save，method 属性就被设置成 save。当客户端发送/linkman_update.action 这样的请求时，action 元素的 name 属性就被设置为 linkman_update，method 属性也被设置成 update。

当然使用通配符是开发中常用的方式，Struts2 还有一种解决这类问题的办法，这种大家可以作为扩展内容来学习。

【解决 Action 的访问的问题的方式三：动态方法访问】

动态方法访问在 Struts2 中默认是不开启的，如果想要使用需要先去开启一个常量。

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

动态方法访问主要的控制是在页面端，所以编写 Action 和配置 Action 都很简单，关键是访问路径的编写。

编写 Action:

```
public class UserAction extends ActionSupport{  
  
    public String save(){  
        System.out.println("保存用户...");  
        return NONE;  
    }  
}
```



```
public String update(){
    System.out.println("修改用户..."); 
    return NONE;
}

public String delete(){
    System.out.println("删除用户..."); 
    return NONE;
}

public String find(){
    System.out.println("查询用户..."); 
    return NONE;
}
```

配置 Action:

```
<!-- 动态方法访问的配置 -->
<action name="userAction" class="cn.itcast.struts2.demo3.UserAction"></action>
```

页面路径写法:

```
<h3>用户的管理</h3>
<a href="${ pageContext.request.contextPath }/userAction!save.action">添加用户</a>
<br/>

<a href="${ pageContext.request.contextPath }/userAction!update.action">修改用户
</a> <br/>

<a href="${ pageContext.request.contextPath }/userAction!delete.action">删除用户
</a> <br/>

<a href="${ pageContext.request.contextPath }/userAction!find.action">查询用户</a>
<br/>
```

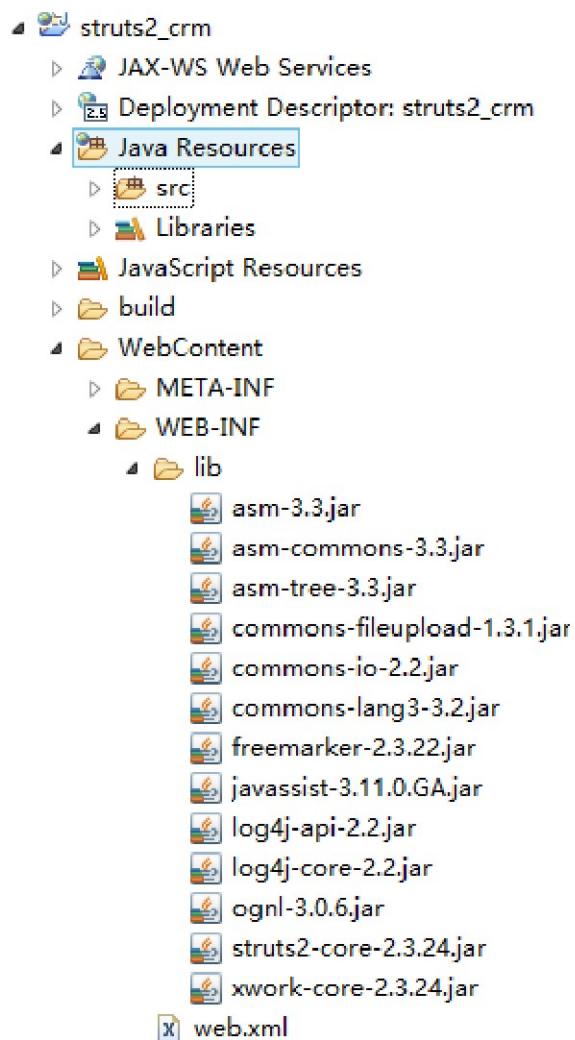
到这对 Struts2 基本的开发和流程应该没有问题了，我们可以通过一个案例对今天的内容进行一个总结。完成客户列表的显示操作。



1.3 案例实现

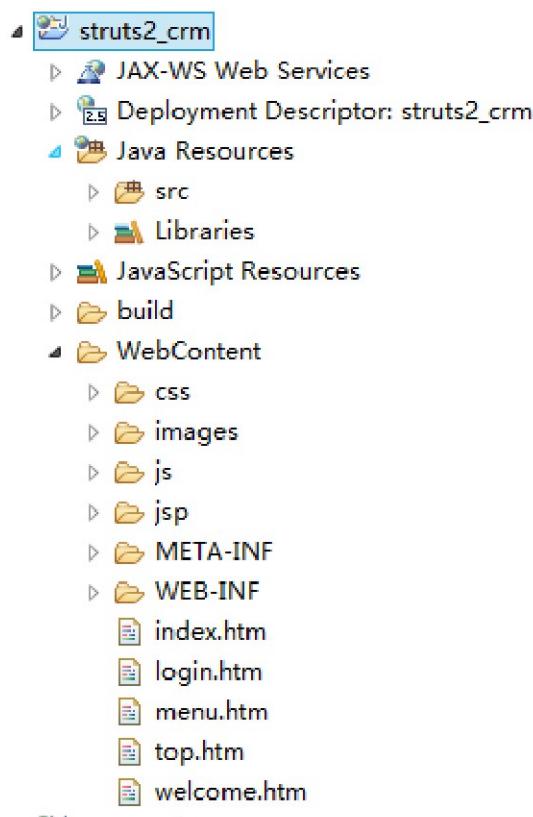
1.3.1 搭建开发环境

1.3.1.1 步骤一：创建 WEB 工程，引入 jar 包

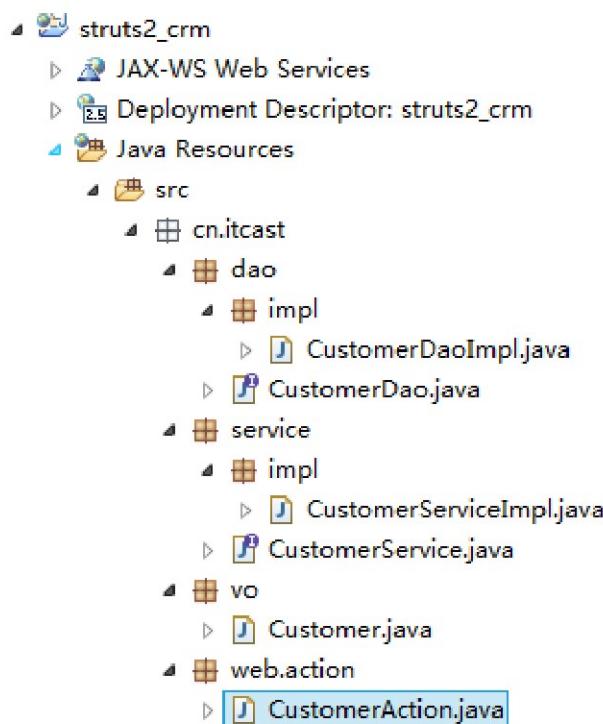




1.3.1.2 步骤二：引入相应的页面



1.3.1.3 步骤三：创建包和相关的类



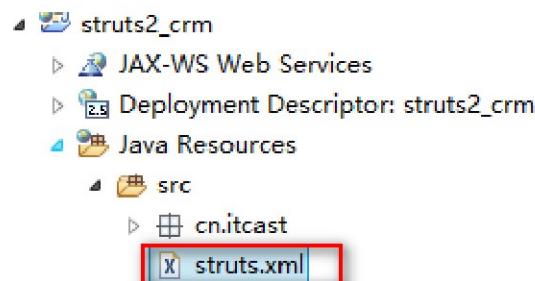


1.3.1.4 步骤四：配置核心过滤器

```
<!-- 配置Struts2的核心过滤器 -->
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

1.3.1.5 步骤五：引入相应的配置文件



1.3.2 案例代码实现

1.3.2.1 步骤六：修改菜单页面修改提交路径

```
<TR>
    <TD class=menuSmall><A class=style2 href="/struts2_crm/customer_findAll.action"
        target=main>- 客户列表</A></TD>
</TR>
```

1.3.2.2 步骤七：编写 Action 中的代码

```
/*
 * 客户管理的 Action 类
 * @author jt
 */
public class CustomerAction extends ActionSupport{

    /**
     * 查询客户列表的方法:
     * @return

```



```
* /  
public String findAll(){  
    // 创建业务层的类的对象  
    CustomerService customerService = new CustomerServiceImpl();  
    // 调用业务层的方法查询所有客户  
    List<Customer> list = customerService.findAll();  
    // 获得 request 对象，并且保存到 request 中。  
    ServletActionContext.getRequest().setAttribute("list", list);  
    return "findAll";  
}  
}
```

1.3.2.3 步骤八：编写业务层的类

业务层的接口

```
/**  
 * 客户管理业务层的接口  
 * @author jt  
 *  
 */  
public interface CustomerService {  
  
    List<Customer> findAll();  
  
}
```

业务层的实现类

```
/**  
 * 客户管理业务层的实现类  
 * @author jt  
 *  
 */  
public class CustomerServiceImpl implements CustomerService {  
  
    @Override  
    public List<Customer> findAll() {  
        CustomerDao customerDao = new CustomerDaoImpl();  
        return customerDao.findAll();  
    }  
  
}
```

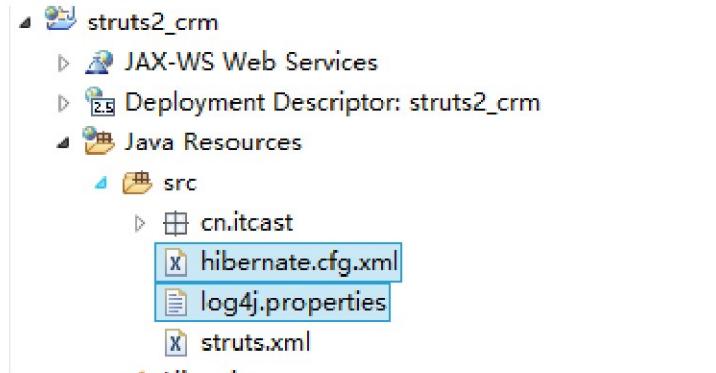


1.3.2.4 步骤九：引入 Hibernate 的 jar 包和配置文件

引入 jar 包

名称	修改日期	类型	大小
antlr-2.7.7.jar	2016/7/7 9:19	Executable Jar File	435 KB
c3p0-0.9.2.1.jar	2016/7/7 11:18	Executable Jar File	414 KB
dom4j-1.6.1.jar	2016/7/7 9:19	Executable Jar File	307 KB
geronimo-jta_1.1_spec-1.1.1.jar	2016/7/7 9:19	Executable Jar File	16 KB
hibernate-c3p0-5.0.7.Final.jar	2016/7/7 11:18	Executable Jar File	12 KB
hibernate-commons-annotations-5.0....	2016/7/7 9:19	Executable Jar File	74 KB
hibernate-core-5.0.7.Final.jar	2016/7/7 9:19	Executable Jar File	5,453 KB
hibernate-jpa-2.1-api-1.0.0.Final.jar	2016/7/7 9:19	Executable Jar File	111 KB
jandex-2.0.0.Final.jar	2016/7/7 9:19	Executable Jar File	184 KB
javassist-3.18.1-GA.jar	2016/7/7 9:19	Executable Jar File	698 KB
jboss-logging-3.3.0.Final.jar	2016/7/7 9:19	Executable Jar File	66 KB
log4j-1.2.16.jar	2016/7/7 9:19	Executable Jar File	471 KB
mchange-commons-java-0.2.3.4.jar	2016/7/7 11:18	Executable Jar File	568 KB
mysql-connector-java-5.1.7-bin.jar	2016/7/7 9:20	Executable Jar File	694 KB
slf4j-api-1.6.1.jar	2016/7/7 9:19	Executable Jar File	25 KB
slf4j-log4j12-1.7.2.jar	2016/7/7 9:19	Executable Jar File	9 KB

引入配置文件



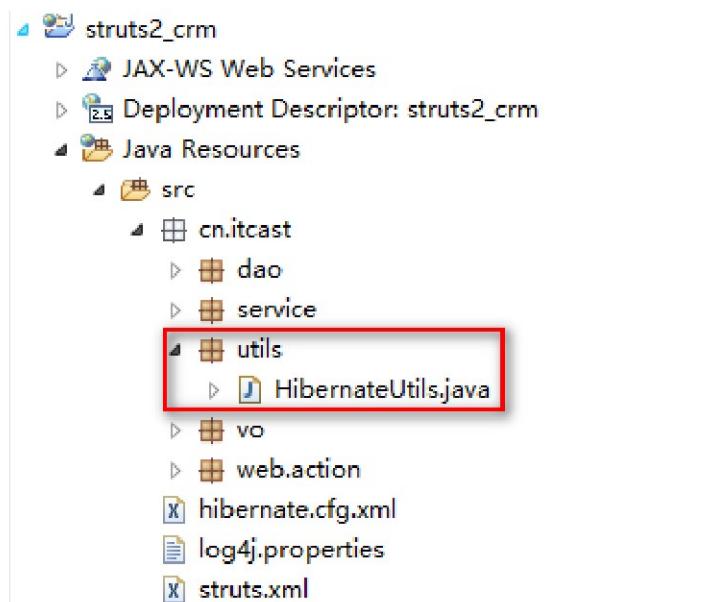
1.3.2.5 步骤十：添加映射文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="cn.itcast.vo.Customer" table=" cst_customer">
        <id name="cust_id" column="cust_id">
            <!-- 主键生成策略 -->
            <generator class="native"/>
```



```
</id>
<property name="cust_name" column="cust_name"/>
<property name="cust_source" column="cust_source"/>
<property name="cust_industry" column="cust_industry"/>
<property name="cust_level" column="cust_level"/>
<property name="cust_phone" column="cust_phone"/>
<property name="cust_mobile" column="cust_mobile"/>
</class>
</hibernate-mapping>
```

1.3.2.6 步骤十一：引入工具类并修改配置文件



修改配置文件

```
<!-- 必要的配置信息：连接数据库的基本参数 -->
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql:///struts2_day01</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">123</property>

<!-- Hibernate加载映射 -->
<mapping resource="cn/itcast/vo/Customer.hbm.xml"/>
```

1.3.2.7 步骤十二：编写 DAO

编写 DAO 接口

```
/*
 * 客户管理的 DAO 的接口
 * @author jt
 */
```



```
/*
public interface CustomerDao {

    List<Customer> findAll();

}
```

编写 DAO 实现类

```
/**
 * 客户管理的 DAO 的实现类
 * @author jt
 *
 */
public class CustomerDaoImpl implements CustomerDao {

    @Override
    /**
     * DAO 中查询所有客户的方法
     */
    public List<Customer> findAll() {
        Session session = HibernateUtils.openSession();
        Transaction tx = session.beginTransaction();

        List<Customer> list = session.createQuery("from Customer").list();

        tx.commit();
        session.close();
        return list;
    }
}
```

1.3.2.8 步骤十三：配置 Action 的有页面跳转

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
"http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>

    <package name="crm" extends="struts-default" namespace="/">
        <action    name="customer_*"    class="cn.itcast.web.action.CustomerAction"
method="{1}">
            <result name="findAll"/>/jsp/customer/list.jsp</result>
        </action>
    </package>
</struts>
```



```
</package>  
</struts>
```

1.3.2.9 步骤十四：在页面中显示相应的数据

引入 jstl 的 jar 包

名称	修改日期	类型	大小
jstl-1.2.jar	2014/6/27 18:51	Executable Jar File	405 KB

编写显示数据的代码

```
<TBODY>  
    <TR>  
        style="FONT-WEIGHT: bold; FONT-STYLE: normal; BACKGROUND-COLOR: #eeeeee; TEXT-DECORATION: none">  
            <TD>客户名称</TD>  
            <TD>客户级别</TD>  
            <TD>客户来源</TD>  
            <TD>电话</TD>  
            <TD>手机</TD>  
            <TD>操作</TD>  
    </TR>  
    <c:forEach items="${list }" var="customer">  
        <TR>  
            style="FONT-WEIGHT: normal; FONT-STYLE: normal; BACKGROUND-COLOR: white; TEXT-DECORATION: none">  
                <TD>${customer.cust_name }</TD>  
                <TD>${customer.cust_level }</TD>  
                <TD>${customer.cust_source }</TD>  
                <TD>${customer.cust_phone }</TD>  
                <TD>${customer.cust_mobile }</TD>  
                <TD>  
                    <a href="#">修改</a>  
                    &nbsp;&nbsp;  
                    <a href="#">删除</a>  
                </TD>  
        </TR>  
    </c:forEach>  
</TBODY>
```

1.3.2.10 步骤十五：测试程序

启动服务器访问页面

The screenshot shows the '客户关系管理系统v1.0' (Customer Relationship Management System v1.0) interface. The left sidebar has a tree menu with '客户管理' expanded, showing '新老客户' and '客户列表'. The main content area displays a table titled '客户列表' with columns: 客户名称 (Customer Name), 客户级别 (Customer Level), 客户来源 (Customer Source), 电话 (Phone), 手机 (Mobile), and 操作 (Operation). Two rows are shown: 张总 (Manager) and 刘总 (Manager). At the bottom right of the table, there is a pagination bar with '共 0 条记录, 0 页, 每页显示 1 页, 共 [前一页] [后一页] 到 [输入框] 页'.



第1章 Struts2_day02

今日任务

- 使用 Struts2 完成对客户的新增操作

教学导航

教学目标	
教学方法	案例驱动法

案例一：使用 **Struts2** 完成对客户的新增的优化操作

1.1 案例需求

1.1.1 需求概述

CRM 系统中客户信息管理模块功能包括：

新增客户信息

客户信息查询

修改客户信息

删除客户信息

本功能要实现新增客户，页面如下：



当前位置：客户管理 > 添加客户

客户名称：	<input type="text"/>	客户级别：	<input type="text"/>
信息来源：	<input type="text"/>	联系人：	<input type="text"/>
固定电话：	<input type="text"/>	移动电话：	<input type="text"/>
联系地址：	<input type="text"/>	邮政编码：	<input type="text"/>
客户传真：	<input type="text"/>	客户网址：	<input type="text"/>

1.2 相关知识点

1.2.1 Struts2 访问 Servlet 的 API:

前面已经对 Struts2 的流程已经执行完成了，但是如果表单中有参数如何进行接收又或者我们需要向页面保存一些数据，又要如何完成呢？我们可以通过学习 Struts2 访问 Servlet 的 API 来实现这样的功能。

在 Struts2 中，Action 并没有直接和 Servlet API 进行耦合，也就是说在 Struts2 的 Action 中不能直接访问 Servlet API。虽然 Struts2 中的 Action 访问 Servlet API 麻烦一些，但是这却是 Struts2 中 Action 的重要改良之一，方便 Action 进行单元测试。

尽管 Action 和 Servlet API 解耦会带来很多好处，然而在 Action 中完全不访问 Servlet API 几乎是不可能的，在实现业务逻辑时，经常要访问 Servlet 中的对象，如 session、request 和 application 等。在 Struts2 中，访问 Servlet API 有 3 种方法，具体如下：

1.2.1.1 通过 ActionContext 类访问

Struts2 框架提供了 ActionContext 类来访问 Servlet API，ActionContext 是 Action 执行的上下文对象，在 ActionContext 中保存了 Action 执行所需要的所有对象，包括 parameters, request, session, application 等。下面列举 ActionContext 类访问 Servlet API 的几个常用方法，具体如表所示。

ActionContext 类访问 Servlet API 的常用方法

方法声明	功能描述
void put(String key, Object value)	将 key-value 键值对放入 ActionContext 中，模拟 Servlet API 中的 HttpServletRequest 的 setAttribute() 方法。
Object get(String key)	通过参数 key 来查找当前 ActionContext 中的值。
Map<String, Object> getApplication()	返回一个 Application 级的 Map 对象。
static ActionContext getContext()	获取当前线程的 ActionContext 对象。
Map<String, Object> getParameters()	返回一个包含所有 HttpServletRequest 参数信



	息的 Map 对象。
Map<String, Object> getSession()	返回一个 Map 类型的 HttpSession 对象。
void setApplication(Map<String, Object> application)	设置 Application 上下文。
void setSession(Map<String, Object> session)	设置一个 Map 类型的 Session 值。

列举的是 ActionContext 类访问 Servlet API 的常用方法，要访问 Servlet API，可以通过如下方式进行，具体示例代码如下：

```
ActionContext context = ActionContext.getContext();
context.put("name", "itcast");
context.getApplication().put("name", "itcast");
context.getSession().put("name", "itcast");
```

在上述示例代码中，通过 ActionContext 类中的方法调用，分别在 request、application 和 session 中放入了("name", "itcast")对。可以看到，通过 ActionContext 类可以非常简单地访问 JSP 内置对象的属性。

为了让大家更好地掌握如何通过 ActionContext 类来访问 Servlet API，接下来通过一个具体的案例来演示 ActionContext 的使用。

(1) 在 Eclipse 中创建一个名称为 struts2_day02 的 Web 项目，将 Struts2 所需的 jar 包复制到项目的 lib 目录中，并发布到类路径下。在 WebContent 目录下编写一个简单的登录页面 demo1.jsp，如文件所示。

```
demo1.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>访问 Servlet 的 API 方式一</h1>
<form action="${pageContext.request.contextPath}/requestDemo1.action"
method="post">
    姓名:<input type="text" name="name"/><br/>
    年龄:<input type="text" name="age"><br/>
    <input type="submit" value="提交">
</form>
</body>
</html>
```

在文件中，编写了姓名和年龄输入框和一个登录提交按钮。

(2) 在 WEB-INF 目录下创建一个名称为 web.xml 的文件，在该文件中配置 Struts2 的核心控制器，如文件所示。

```
web.xml
```



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">

    <filter>
        <filter-name>struts2</filter-name>

        <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

(3) 在 src 目录下创建 struts.xml 文件，如下所示。

```
struts.xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>
    <package name="demo1" extends="struts-default" namespace="/">
        <action
            name="requestDemo1"
            class="cn.itcast.struts.demo1.RequestDemo1Action">
            <result name="success">/demo2.jsp</result>
        </action>
    </package>
</struts>
```

(4) 在 src 目录下创建一个 cn.itcast.struts.demo1 包，再在 cn.itcast.struts.demo1 包中创建 RequestActionDemo1 类，进行业务逻辑处理，如下所示。

```
RequestActionDemo1.java
public class RequestActionDemo1 extends ActionSupport {

    @Override
    public String execute() throws Exception {
        // 接收表单的参数:
        // 使用的是 Struts2 中的一个对象 ActionContext 对象.
```



```
ActionContext actionContext = ActionContext.getContext();
// 接收参数:
Map<String, Object> paramsMap = actionContext.getParameters();
for (String key : paramsMap.keySet()) {
    String[] value = (String[]) paramsMap.get(key);
    System.out.println(key + " " + value[0]);
}

// 向 request 中存入数据 request.setAttribute(String name, Object value);
actionContext.put("requestName", "张三");
// 向 session 中存入数据 request.getSession().setAttribute(String name, Object value);
actionContext.getSession().put("sessionName", "李四");
// 向 application 中存入数据 this.getServletContext().setAttribute(String name, Object value);
actionContext.getApplication().put("applicationName", "王五");

return SUCCESS;
}
}
```

(5) 在 WebContent 目录下创建 demo2.jsp，如下所示。

```
demo2.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>获取数据</h1>
${ reqName }
${ sessName }
${ appName }
</body>
</html>
```

1.2.1.2 通过特定接口访问

Struts2 框架提供了 ActionContext 类来访问 Servlet API，虽然这种方法可以访问 Servlet API，但



是无法直接获得 Servlet API 实例。为了在 Action 中直接访问 Servlet API，Struts2 还提供了一系列接口，具体如下：

- **ServletRequestAware**: 实现该接口的 Action 可以直接访问 Web 应用的 HttpServletRequest 实例。
- **ServletResponseAware**: 实现该接口的 Action 可以直接访问 Web 应用的 HttpServletResponse 实例。
- **SessionAware**: 实现该接口的 Action 可以直接访问 Web 应用的 HttpSession 实例。
- **ServletContextAware**: 实现该接口的 Action 可以直接访问 Web 应用的 ServletContext 实例。

下面以 **ServletRequestAware** 为例，讲解如何在 Action 中访问 HttpServletRequest 实例。

(1) 在 cn.itcast.struts.demo1 包中创建 RequestDemo2Action 类，如下所示。

RequestDemo2Action.java

```
1 package cn.itcast.struts.demo1;
2 import javax.servlet.http.HttpServletRequest;
3 import org.apache.struts2.interceptor.ServletRequestAware;
4 import com.opensymphony.xwork2.ActionSupport;
5 public class RequestDemo2Action extends ActionSupport implements
6 ServletRequestAware{
7     HttpServletRequest request;
8     @Override
9     public void setServletRequest(HttpServletRequest request) {
10         this.request = request;
11     }
12     @Override
13     public String execute() throws Exception {
14         request.setAttribute("message",
15             "通过ServletRequestAware 接口实现了访问 Servlet API");
16         return SUCCESS;
17     }
18 }
```

在上述文件中，自定义的 RequestDemo2Action 类实现了 Action 的 **ServletRequestAware** 接口。需要注意的是，RequestDemo2Action 类中必须实现 **setServletRequest()**方法和 **execute()**方法，通过 **setServletRequest()**方法，可以得到 **HttpServletRequest** 的实例，这是在调用 **execute()**方法或者其他自定义的方法之前就调用的，然后在 **execute()**方法中，就可以访问 **HttpServletRequest** 的属性内容了。

(2) 修改 struts.xml 配置文件，在其 package 元素中添加一个名称为 aware 的 action 配置信息，所需添加的代码如下所示：

```
<action name="requestDemo2" class="cn.itcast.struts.demo1.RequestDemo2Action">
    <result name="success"/>/message.jsp</result>
</action>
```

(3) 在 WebContent 目录下新建一个 message.jsp 页面，通过 EL 表达式去访问存放在 **request** 对象中的键为 **message** 的值，在页面**<body>**元素中所需添加的代码代码如下：

```
<div align=center>${requestScope.message}</div>
```

(4) 重新发布项目 struts2_day02，在浏览器地址栏中输入“http://localhost:8080/struts2_day02/requestDemo2.action”，成功访问后可以看出，使用 **ServletRequestAware** 接口顺利访问了 Servlet API。



1.2.1.3 通过 ServletActionContext 访问

为了直接访问 Servlet API，Struts2 框架还提供了 `ServletActionContext` 类，该类包含了几个常用的静态方法，具体如下：

- `static HttpServletRequest getRequest()`: 获取 Web 应用的 `HttpServletRequest` 对象。
- `static HttpServletResponse getResponse()`: 获取 Web 应用的 `HttpServletResponse` 对象。
- `static ServletContext getServletContext()`: 获取 Web 应用的 `ServletContext` 对象。
- `static PageContext getPageContext()`: 获取 Web 应用的 `PageContext` 对象。

接下来，讲解如何通过 `ServletActionContext` 访问 Servlet API。

(1) 在 `cn.itcast.action` 包中创建 `RequestDemo3Action` 类，该类中的代码，如下所示。

RequestDemo3Action.java

```
package cn.itcast.struts.demol;

import java.util.Arrays;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class RequestDemo3Action extends ActionSupport {

    @Override
    public String execute() throws Exception {
        // 接收参数:
        HttpServletRequest request = ServletActionContext.getRequest();
        Map<String, String[]> map = request.getParameterMap();
        for (String key : map.keySet()) {
            String[] value = map.get(key);
            System.out.println(key + " " + Arrays.toString(value));
        }

        // 向 request 域中存值:
        request.setAttribute("reqName", "reqValue");
        // 向 session 域中存值:
        request.getSession().setAttribute("sessName", "sessValue");
        // 向 application 域中存值:
        ServletActionContext.getServletContext().setAttribute("appName",
        "appValue");
        return SUCCESS;
    }
}
```

(2) 配置 `struts.xml`，在 `package` 元素中添加一个名称为 `requestDemo3` 的 action 配置信息，添加代码如下所示：



```
<action name="requestDemo3" class="cn.itcast.struts.demo1.RequestDemo3Action">
    <result name="success"/>/demo2.jsp</result>
</action>
```

(3) 重新运行 struts2_day02 项目，在浏览器的地址栏中输入“http://localhost:8080/struts2_day02/requestDemo3”。

从以上内容可以看出，借助于 `ServletActionContext` 类的帮助，开发者也可以在 Action 中直接访问 Servlet API，避免了 Action 类实现 `ServletRequestAware`、`ServletResponseAware`、`SessionAware` 和 `ServletContextAware` 等 `XxxAware` 接口。虽然如此，该 Action 依然与 Servlet API 直接耦合，一样不利于程序解耦。综上三种访问 Servlet API 的方式，建议在开发中优先使用 `ActionContext`，以避免和 Servlet API 耦合。

Servlet 的 API 已经可以访问了，那么在页面跳转的时候我们会发现其实 Struts2 默认使用的都是转发，那如果我们要使用重定向，应该怎么办呢？接下来我们来看下结果页面的配置。

1.2.2 结果页面的配置：

在 `struts.xml` 文件中，Result 的配置非常简单，使用 `<result>` 元素来配置 Result 逻辑视图与物理视图之间的映射，`<result>` 元素可以有 `name` 和 `type` 属性，但这两种属性都不是必选的。

- `name` 属性：指定逻辑视图的名称，默认值为 `success`。
- `type` 属性：指定返回的视图资源的类型，不同的类型代表不同的结果输出，默认值是 `dispatcher`。

`struts.xml` 文件中的 `<result>` 元素配置代码如下所示：

```
<action name="requestDemo1" class="cn.itcast.struts.demo1.RequestDemo1Action">
    <result name="success" type="dispatcher"/>/demo2.jsp</result>
</action>
```

在上述配置中，使用了 `<result>` 元素的 `name`、`type` 属性。其中，为 Action 配置了 `name` 为 `success` 的 Result 映射，该映射的值可以是 JSP 页面，也可以是一个 Action 的 `name` 值用 `type` 属性指定了该 Result 的结果类型为 `dispatcher`，它也是默认的结果类型。

在结果页面的配置中，Struts2 有两种配置的方式，一种称为全局结果页面，一种称为局部结果页面。全局结果是指在这个包下的所有返回相同字符串的值，都可以向这个页面来进行跳转。局部结果是指在某个 Action 中返回的字符串的值，会向这个页面跳转。

1.2.2.1 全局结果页面

全局结果页面是指在同一个包下面配置的 Action 中返回相同的字符串的值，都可以跳转到该页面。需要通过 `<global-results>` 进行配置。

```
<global-results>
    <result name="success"/>/success.jsp</result>
</global-results>
```

1.2.2.2 局部结果页面

局部结果页面是指在某个 Action 中根据该字符串的值进行页面的跳转。只对这个 Action 有效。



```
<action name="requestDemo1" class="cn.itcast.struts.demo1.RequestDemo1Action">
    <result name="success" type="dispatcher"/>/demo2.jsp</result>
</action>
```

在 Struts2 中，当框架调用 Action 对请求进行处理后，就要向用户呈现一个结果视图。在 Struts2 中，预定义了多种 ResultType，其实就是定义了多种展示结果的技术。

一个结果类型就是实现了 com.opensymphony.xwork2.Result 接口的类，Struts2 把内置的 <result-type> 都放在 struts-default 包中，struts-default 包就是配置包的父包，这个包定义在 struts2-core-2.3.24.jar 包中的根目录下的 struts-default.xml 文件中，可以找到相关的<result-type>的定义。

每个<result-type>元素都是一种视图技术或者跳转方式的封装，其中的 name 属性指出在<result>元素中如何引用这种视图技术或者跳转方式，对应着<result>元素的 type 属性。Struts2 中预定义的 ResultType 如表所示。

Struts2 中预定义的 ResultType

属性	说明
chain	用来处理 Action 链，被跳转的 Action 中仍能获取上个页面的值，如 request 信息。
dispatcher	用来转向页面，通常处理 JSP，是默认的结果类型。
freemarker	用来整合 FreeMarker 模板结果类型。
httpheader	用来处理特殊的 HTTP 行为结果类型。
redirect	重定向到一个 URL，被跳转的页面中丢失传递的信息。
redirectAction	重定向到一个 Action，跳转的页面中丢失传递的信息。
stream	向浏览器发送 InputStream 对象，通常用来处理文件下载，还可用于 Ajax 数据。
velocity	用来整合 Velocity 模板结果类型。
xslt	用来整合 XML/XSLT 结果类型。
plainText	显示原始文件内容，例如文件源代码。
postback	使得当前请求参数以表单形式提交

其中红色的几个值比较常用，需要重点记忆。其他的了解即可，到这我们已经了解了 Struts2 的结果页面的配置了，也知道如何接收数据了，但是接收过来的数据，往往需要进行封装才会向业务层进行传递，那么作为一个框架，如果连这点功能都没有，那就太不像一个“框架”了。那么在 Struts2 中提供了对于数据封装的几种方式。接下来我们就来学习一下。

1.2.3 Struts 的数据封装

在很多的实际开发的场景中：页面提交请求参数到 Action，在 Action 中接收参数并且对请求参数需要进行数据的封装。封装到一个 JavaBean 中，然后将 JavaBean 传递给业务层。那么这些操作 Struts2 已经替我们都想好了。Struts2 将数据的封装分成两大类，一类被称为是属性驱动，一类被称为是模型驱动。我们先来看第一种：属性驱动。

属性驱动可以细分成两种，一种只需要提供属性的 set 方法即可。另一种可以通过表达式方式直



接封装到对象中。

1.2.3.1 属性驱动

在 Struts2 中，可以直接在 Action 中定义各种 Java 基本数据类型的字段，使这些字段与表单数据相对应，并利用这些字段进行数据传递。

【属性驱动方式一：提供属性的 set 方法的方式】

编写页面

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>数据的封装方式一：提供 set 方法的方式</h1>
<form action="${pageContext.request.contextPath}/actionDemo1.action"
method="post">
    姓名:<input type="text" name="name"/><br/>
    年龄:<input type="text" name="age"/><br/>
    生日:<input type="text" name="birthday"/><br/>
    工资:<input type="text" name="salary"/><br/>
    <input type="submit" value="提交">
</form>

</body>
</html>
```

编写 Action 类

```
package cn.itcast.struts.demo2;

import java.util.Date;

import com.opensymphony.xwork2.ActionSupport;

import cn.itcast.struts.domain.User;
/**
 * 参数封装的属性驱动的方式一：提供属性的 set 方法.
 */
public class ActionDemo1 extends ActionSupport{
    // 接收参数：
    private String name;
```



```
private Integer age;
private Date birthday;
private Double salary;

public void setName(String name) {
    this.name = name;
}

public void setAge(Integer age) {
    this.age = age;
}

public void setBirthday(Date birthday) {
    this.birthday = birthday;
}

public void setSalary(Double salary) {
    this.salary = salary;
}

@Override
public String execute() throws Exception {
    System.out.println(name);
    System.out.println(age);
    System.out.println(birthday);
    System.out.println(salary);

    return NONE;
}
}
```

以上这种方式需要通过在 Action 中定义属性，并且提供属性的 set 方法来完成。这种方式只需要提供 set 方法即可。但若需要传入的数据很多的话，那么 Action 的属性也会变得很多。再加上属性有对应的 getter/setter 方法，Action 类的代码会很庞大，在 Action 里编写业务的代码时，会使 Action 非常臃肿，不够简洁。那么要怎样解决这个问题呢？

把属性和相应的 getter/setter 方法从 Action 里提取出来，单独作为一个值对象，这个对象就是用来封装这些数据的，在相应的 Action 里直接使用这个对象，而且可以在多个 Action 里使用。采用这种方式，一般以 JavaBean 来实现，所封装的属性和表单的属性一一对应，JavaBean 将成为数据传递的载体。

【属性驱动方式二：页面提供表达式方式】

编写页面

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```



```
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>数据的封装方式二：页面提供表达式的方式.</h1>
<form      action="${pageContext.request.contextPath }/actionDemo2.action"
method="post">
    姓名:<input type="text" name="user.name"/><br/>
    年龄:<input type="text" name="user.age"><br/>
    生日:<input type="text" name="user.birthday"><br/>
    工资:<input type="text" name="user.salary"/><br/>
    <input type="submit" value="提交">
</form>
</body>
</html>
```

编写 Action:

```
package cn.itcast.struts.demo2;

import com.opensymphony.xwork2.ActionSupport;

import cn.itcast.struts.domain.User;
/**
 * 数据封装属性驱动的方式二：页面使用 OGNL 表达式的方式.
 */
public class ActionDemo2 extends ActionSupport{
    private User user;
    // 必须提供对象的 get 方法
    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    @Override
    public String execute() throws Exception {
        System.out.println(user);

        return NONE;
    }
}
```



}

以上这种方式需要提供对 user 的 get 方法，如果没有提供 get 方法，在 Struts2 的底层就没有办法获得该对象，那么在 user 中只会有一个属性被封装进去，而其他的属性都是 null。

1.2.3.2 模型驱动

在 Struts2 中，Action 处理请求参数还有另外一种方式，叫做模型驱动（ModelDriven）。通过实现 ModelDriven 接口来接收请求参数，Action 类必须实现 ModelDriven 接口，并且要重写 getModel() 方法，这个方法返回的就是 Action 所使用的数据模型对象。

模型驱动方式通过 JavaBean 模型进行数据传递。只要是普通的 JavaBean，就可以充当模型部分。采用这种方式，JavaBean 所封装的属性与表单的属性一一对应，JavaBean 将成为数据传递的载体。代码如下所示。

【模型驱动】

编写页面:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>

<h1>数据的封装方式三：使用模型驱动的方式.</h1>
<form action="${pageContext.request.contextPath }/actionDemo3.action"
method="post">
    姓名:<input type="text" name="name"/><br/>
    年龄:<input type="text" name="age"><br/>
    生日:<input type="text" name="birthday"><br/>
    工资:<input type="text" name="salary"/><br/>
    <input type="submit" value="提交">
</form>
</body>
</html>
```

编写 Action

```
package cn.itcast.struts.demo2;

import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

import cn.itcast.struts.domain.User;
/**
```



```
* 数据封装的模型驱动的方式：模型驱动的方式。
* @author jt
*
*/
public class ActionDemo3 extends ActionSupport implements ModelDriven<User>{
    // 手动构建对象
    private User user =new User();
    @Override
    public User getModel() {
        return user;
    }

    @Override
    public String execute() throws Exception {
        System.out.println(user);
        return NONE;
    }
}
```

到这我们已经能够将数据封装到一个 Java 对象中了，大部分我们会优先使用模型驱动的方式，因为 Struts2 内部有很多结果是围绕模型驱动设计的。但如果页面向多个对象中封装，那么就需要使用属性驱动的方式二了。这些都是像某个对象中封装数据，那么如果 Action 中需要一个对象的集合呢？又应该如何进行数据的封装呢？那么接下来我们来了解一下 Struts2 中复杂类型数据的封装。

1.2.4 Struts2 中封装集合类型的数据：

在实际的开发中，有些时候我们需要批量插入用户或者批量插入其他的对象，在 Action 中需要接受到这多个 Action 中封装的对象，然后传递给业务层。那么这个时候就需要将表单的数据封装到集合中。一般我们通常使用的集合无非是 List 或者是 Map 集合。下面就以这两种集合进行数据的封装的示例演示。

1.2.4.1 封装到 List 集合中：

编写页面：

```
<form      action="${pageContext.request.contextPath} /strutsDemo4.action"
method="post">
    名称:<input type="text" name="list[0].name"><br/>
    年龄:<input type="text" name="list[0].age"><br/>
    生日:<input type="text" name="list[0].birthday"><br/>
    名称:<input type="text" name="list[1].name"><br/>
    年龄:<input type="text" name="list[1].age"><br/>
    生日:<input type="text" name="list[1].birthday"><br/>
```



```
<input type="submit" value="提交">  
</form>
```

编写 Action:

```
public class StrutsDemo4 extends ActionSupport {  
    private List<User> list;  
  
    public List<User> getList() {  
        return list;  
    }  
  
    public void setList(List<User> list) {  
        this.list = list;  
    }  
  
    @Override  
    public String execute() throws Exception {  
        for (User user : list) {  
            System.out.println(user);  
        }  
        return NONE;  
    }  
}
```

List 集合有下标，所以可以通过 list[0], list[1]。那么如果是 Map 集合又应该如何进行封装呢？那么接下来我们看如何将数据封装到 Map 集合中。

1.2.4.2 封装数据到 Map 集合:

编写页面:

```
<h1>批量插入用户:封装到 Map 集合</h1>  
<form action="${pageContext.request.contextPath}/strutsDemo5.action"  
method="post">  
    名称:<input type="text" name="map['one'].name"><br/>  
    年龄:<input type="text" name="map['one'].age"><br/>  
    生日:<input type="text" name="map['one'].birthday"><br/>  
    名称:<input type="text" name="map['two'].name"><br/>  
    年龄:<input type="text" name="map['two'].age"><br/>  
    生日:<input type="text" name="map['two'].birthday"><br/>  
    <input type="submit" value="提交">  
</form>
```

编写 Action:

```
public class StrutsDemo5 extends ActionSupport {  
    private Map<String, User> map;
```



```
public Map<String, User> getMap() {
    return map;
}

public void setMap(Map<String, User> map) {
    this.map = map;
}

@Override
public String execute() throws Exception {
    for (String key : map.keySet()) {
        User user = map.get(key);
        System.out.println(key+" "+user);
    }
    return NONE;
}
}
```

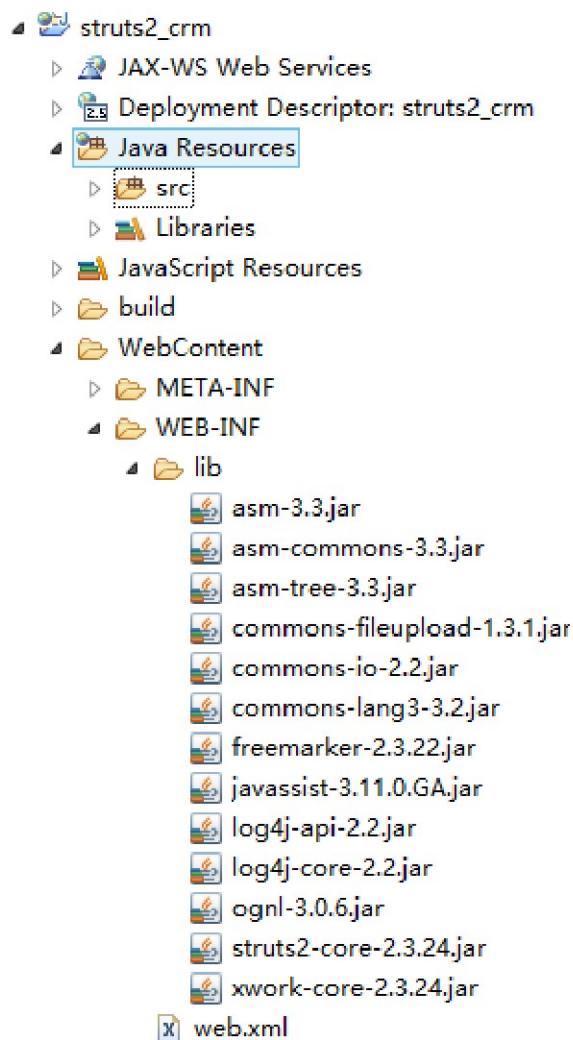
到这数据已经可以进行封装了，那么我们来编写一个案例，对今天的内容进行一下总结。我们就将 CRM 中的客户的保存来进行实现。



1.3 案例代码：

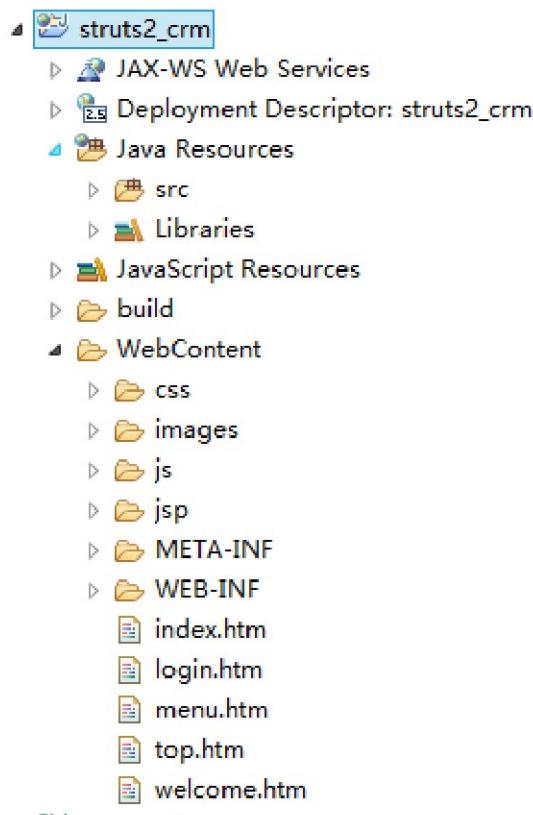
1.3.1 环境准备

1.3.1.1 步骤一：创建 WEB 工程，引入 jar 包

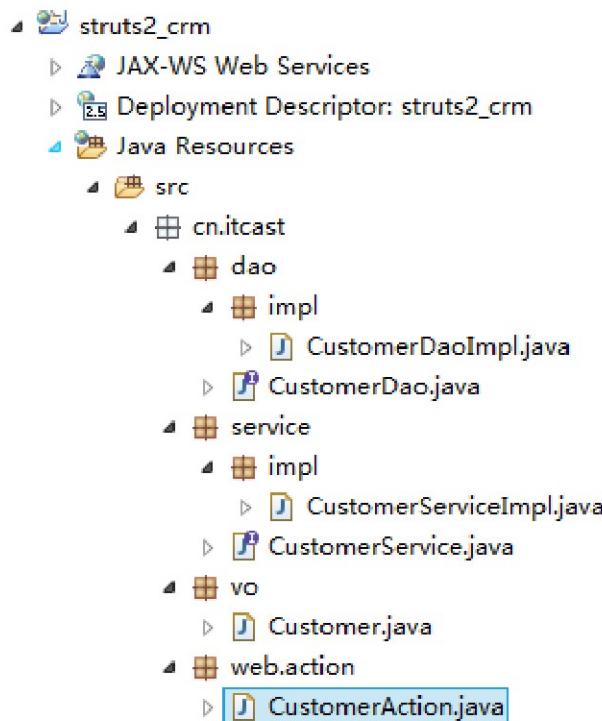




1.3.1.2 步骤二：引入相应的页面



1.3.1.3 步骤三：创建包和相关的类



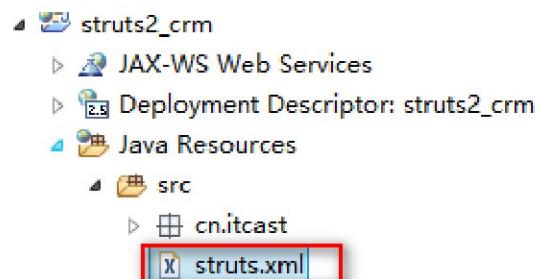


1.3.1.4 步骤四：配置核心过滤器

```
<!-- 配置Struts2的核心过滤器 -->
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

1.3.1.5 步骤五：引入相应的配置文件



1.3.2 代码实现

1.3.2.1 步骤六：修改菜单页面修改提交路径

```
<TR>
    <TD class=menuSmall><A class=style2 href="/struts2_crm/customer_saveUI.action"
        target=main>- 新增客户</A></TD>
</TR>
```

1.3.2.2 步骤七：编写 Action 类

```
package cn.itcast.web.action;

import java.util.List;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;
import cn.itcast.service.CustomerService;
import cn.itcast.service.impl.CustomerServiceImpl;
import cn.itcast.vo.Customer;
/**
 * 客户管理的 Action 类

```



```
*  
*/  
  
public class CustomerAction extends ActionSupport{  
    /**  
     * 跳转到客户的添加页面  
     * @return  
     */  
  
    public String saveUI(){  
        return "saveUI";  
    }  
}
```

1.3.2.3 步骤八：配置 Action

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE struts PUBLIC  
"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"  
"http://struts.apache.org/dtds/struts-2.3.dtd">  
<struts>  
  
<package name="crm" extends="struts-default" namespace="/">  
    <action name="customer_*" class="cn.itcast.web.action.CustomerAction"  
method="{1}">  
        <result name="saveUI"/>/isp/customer/add.jsp</result>  
    </action>  
    </package>  
</struts>
```

1.3.2.4 步骤九：启动服务器测试



1.3.2.5 步骤十：修改表单提交路径

```
<FORM id=form1 name=form1  
action="${pageContext.request.contextPath }/customer_save.action"  
method=post>
```

1.3.2.6 步骤十一：编写 Action 中的 save 方法

```
package cn.itcast.web.action;  
  
import java.util.List;  
  
import org.apache.struts2.ServletActionContext;  
  
import com.opensymphony.xwork2.ActionSupport;  
import com.opensymphony.xwork2.ModelDriven;  
  
import cn.itcast.service.CustomerService;  
import cn.itcast.service.impl.CustomerServiceImpl;  
import cn.itcast.vo.Customer;  
  
/**  
 * 客户管理的 Action 类  
 *  
 */  
  
public class CustomerAction extends ActionSupport implements ModelDriven<Customer>{  
    // 模型驱动使用的对象  
    private Customer customer = new Customer();  
    @Override  
    public Customer getModel() {  
        return customer;  
    }  
  
    /**  
     * 保存客户的执行的方法  
     */  
    public String save(){  
        CustomerService customerService = new CustomerServiceImpl();  
        customerService.save(customer);  
        return "saveSuccess";  
    }  
  
    /**  
     * 跳转到客户的添加页面  
     */
```



```
* @return
*/
public String saveUI(){
    return "saveUI";
}

}
```

1.3.2.7 步骤十二：编写业务层的类

业务层接口

```
package cn.itcast.service;

import java.util.List;

import cn.itcast.vo.Customer;
/**
 * 客户管理业务层的接口
 *
 */
public interface CustomerService {

    void save(Customer customer);

}
```

业务层实现类

```
package cn.itcast.service.impl;

import java.util.List;

import cn.itcast.dao.CustomerDao;
import cn.itcast.dao.impl.CustomerDaoImpl;
import cn.itcast.service.CustomerService;
import cn.itcast.vo.Customer;
/**
 * 客户管理业务层的实现类
 *
 */
public class CustomerServiceImpl implements CustomerService {

    @Override
    public void save(Customer customer) {
        CustomerDao customerDao = new CustomerDaoImpl();
        customerDao.save(customer);
    }
}
```



```
}
```

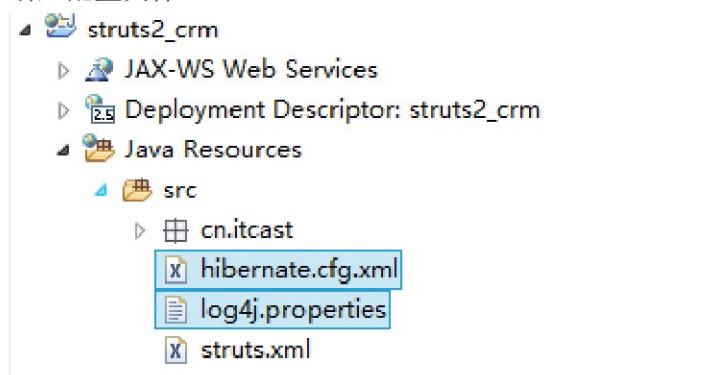
```
}
```

1.3.2.8 步骤十三：引入 Hibernate 的 jar 包和配置文件

引入 jar 包

名称	修改日期	类型	大小
antlr-2.7.7.jar	2016/7/7 9:19	Executable Jar File	435 KB
c3p0-0.9.2.1.jar	2016/7/7 11:18	Executable Jar File	414 KB
dom4j-1.6.1.jar	2016/7/7 9:19	Executable Jar File	307 KB
geronimo-jta_1.1_spec-1.1.1.jar	2016/7/7 9:19	Executable Jar File	16 KB
hibernate-c3p0-5.0.7.Final.jar	2016/7/7 11:18	Executable Jar File	12 KB
hibernate-commons-annotations-5.0....	2016/7/7 9:19	Executable Jar File	74 KB
hibernate-core-5.0.7.Final.jar	2016/7/7 9:19	Executable Jar File	5,453 KB
hibernate-jpa-2.1-api-1.0.0.Final.jar	2016/7/7 9:19	Executable Jar File	111 KB
jandex-2.0.0.Final.jar	2016/7/7 9:19	Executable Jar File	184 KB
javassist-3.18.1-GA.jar	2016/7/7 9:19	Executable Jar File	698 KB
jboss-logging-3.3.0.Final.jar	2016/7/7 9:19	Executable Jar File	66 KB
log4j-1.2.16.jar	2016/7/7 9:19	Executable Jar File	471 KB
mchange-commons-java-0.2.3.4.jar	2016/7/7 11:18	Executable Jar File	568 KB
mysql-connector-java-5.1.7-bin.jar	2016/7/7 9:20	Executable Jar File	694 KB
slf4j-api-1.6.1.jar	2016/7/7 9:19	Executable Jar File	25 KB
slf4j-log4j12-1.7.2.jar	2016/7/7 9:19	Executable Jar File	9 KB

引入配置文件



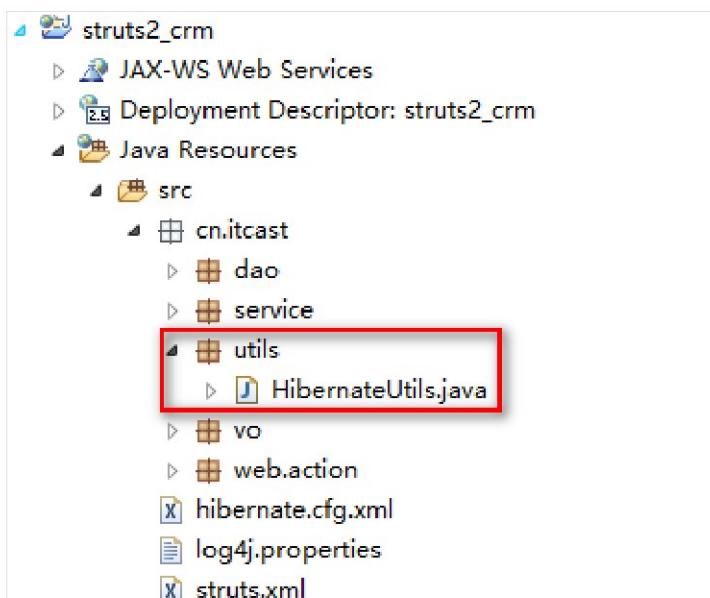
1.3.2.9 步骤十四：添加映射文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
```



```
<class name="cn.itcast.vo.Customer" table="cst_customer">
    <id name="cust_id" column="cust_id">
        <!-- 主键生成策略 -->
        <generator class="native"/>
    </id>
    <property name="cust_name" column="cust_name"/>
    <property name="cust_source" column="cust_source"/>
    <property name="cust_industry" column="cust_industry"/>
    <property name="cust_level" column="cust_level"/>
    <property name="cust_phone" column="cust_phone"/>
    <property name="cust_mobile" column="cust_mobile"/>
</class>
</hibernate-mapping>
```

1.3.2.10 步骤十五：引入工具类并修改配置文件



修改配置文件

```
<!-- 必要的配置信息：连接数据库的基本参数 -->
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql:///struts2_day01</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">123</property>

<!-- Hibernate加载映射 -->
<mapping resource="cn/itcast/vo/Customer.hbm.xml"/>
```

1.3.2.11 步骤十六：编写 DAO

DAO 接口



```
package cn.itcast.dao;

import java.util.List;

import cn.itcast.vo.Customer;
/***
 * 客户管理的 DAO 的接口
 *
 */
public interface CustomerDao {

    void save(Customer customer);

}
```

DAO 的实现类

```
package cn.itcast.dao.impl;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.Transaction;

import cn.itcast.dao.CustomerDao;
import cn.itcast.utils.HibernateUtils;
import cn.itcast.vo.Customer;
/***
 * 客户管理的 DAO 的实现类
 *
 */
public class CustomerDaoImpl implements CustomerDao {

    @Override
    public void save(Customer customer) {
        Session session = HibernateUtils.openSession();
        Transaction tx = session.beginTransaction();

        session.save(customer);

        tx.commit();
        session.close();
    }

}
```

1.3.2.12 配置保存后的跳转页面

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

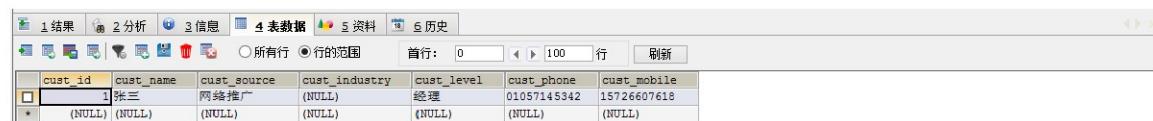
    <package name="crm" extends="struts-default" namespace="/">
        <action name="customer_*" class="cn.itcast.web.action.CustomerAction"
method="{1}">
            <result name="saveUI"/>/jsp/customer/add.jsp</result>
            <result
                name="saveSuccess"
type="redirect">customer_findAll.action</result>
        </action>
    </package>
</struts>
```

其中这里要注意，如果有查询列表的 Action，保存以后应该重定向到查询客户列表的 Action 中，如果没有可以配置一个跳转成功的页面即可。

1.3.2.13 步骤十七：测试程序



到数据库中查看



cust_id	cust_name	cust_source	cust_industry	cust_level	cust_phone	cust_mobile
1	张三	网络推广	(NULL)	经理	01057145342	15726607618

数据库中已经有记录了，说明我们的保存操作是 OK 的。

第1章 Struts2_day03

今日任务

- 使用 Struts2 完成对客户查询的优化操作

教学导航

教学目标	
教学方法	案例驱动法

案例一：使用 **Struts2** 完成对客户查询的优化操作

1.1 案例需求

1.1.1 需求概述

CRM 系统中客户信息管理模块功能包括：

新增客户信息

客户信息查询

修改客户信息

删除客户信息

本功能要实现查询客户，页面如下：



当前位置：客户管理 > 客户列表

客户名称	客户级别	客户来源	联系人	电话	手机	操作
王五	VIP客户	网络营销	李秘书	010-87659823	15726609865	修改 删除
李四	VIP客户	网络营销	张秘书	010-87659823	15726609865	修改 删除
小王	普通客户	电话营销	李秘书	010-87659823	15726609865	修改 删除

共[18]条记录 [6]页,每页显示 3 条 [前一页] [后一页] 到 页 Go

之前我们的查询列表已经完成过了，但是用的是 EL 和 JSTL 来完成，其实这样并不好，因为 Struts2 有自己的表达式语言和存取值的方式。那么接下来我们就来学习 OGNL 和值栈。



1.2 相关知识点

1.2.1 OGNL 的概述

1.2.1.1 什么是 OGNL

OGNL

[编辑](#)

OGNL是Object-Graph Navigation Language的缩写，它是一种功能强大的表达式语言，通过它简单一致的表达式语法，可以存取对象的任意属性，调用对象的方法，遍历整个对象的结构图，实现字段类型转化等功能。它使用相同的表达式去存取对象的属性。

OGNL 的全称是对象图导航语言（Object-Graph Navigation Language），它是一种功能强大的开源表达式语言，使用这种表达式语言，可以通过某种表达式语法，存取 Java 对象的任意属性，调用 Java 对象的方法，同时能够自动实现必要的类型转换。如果把表达式看作是一个带有语义的字符串，那么 OGNL 无疑成为了这个语义字符串与 Java 对象之间沟通的桥梁。

1.2.1.2 OGNL 的作用

Struts2 默认的表达式语言就是 OGNL，它具有以下特点：

- 支持对象方法调用。例如：objName.methodName()。
- 支持类静态方法调用和值访问，表达式的格式为@[类全名(包括包路径)]@[方法名 | 值名]。例如：@java.lang.String@format('foo %s', 'bar')。
- 支持赋值操作和表达式串联。例如：price=100, discount=0.8, calculatePrice()，在方法中进行乘法计算会返回 80。
- 访问 OGNL 上下文（OGNL context）和 ActionContext。
- 操作集合对象。

1.2.1.3 OGNL 的要素

了解了什么是 OGNL 及其特点后，接下来，分析一下 OGNL 的结构。OGNL 的操作实际上就是围绕着 OGNL 结构的三个要素而进行的，分别是表达式（Expression）、根对象（Root Object）、上下文环境（Context），下面分别讲解这三个要素，具体如下。

1、表达式

表达式是整个 OGNL 的核心，OGNL 会根据表达式去对象中取值。所有 OGNL 操作都是针对表达式解析后进行的。它表明了此次 OGNL 操作要“做什么”。表达式就是一个带有语法含义的字符串，这个字符串规定了操作的类型和操作的内容。OGNL 支持大量的表达式语法，不仅支持这种“链式”对象访问路径，还支持在表达式中进行简单的计算。

2、根对象（Root）



Root 对象可以理解为 OGNL 的操作对象，表达式规定了“做什么”，而 Root 对象则规定了“对谁操作”。OGNL 称为对象图导航语言，所谓对象图，即以任意一个对象为根，通过 OGNL 可以访问与这个对象关联的其它对象。

3、Context 对象

实际上 OGNL 的取值还需要一个上下文环境。设置了 Root 对象，OGNL 可以对 Root 对象进行取值或写值等操作，Root 对象所在环境就是 OGNL 的上下文环境（Context）。上下文环境规定了 OGNL 的操作“在哪里进行”。上下文环境 Context 是一个 Map 类型的对象，在表达式中访问 Context 中的对象，需要使用“#”号加上对象名称，即“#对象名称”的形式。

1.2.1.4 OGNL 的入门

前面已经提到过 OGNL 支持方法的调用，比如访问对象方法和访问静态方法，这里结合一些案例来演示 OGNL 是如何调用方法的。

演示 OGNL 如何访问对象的方法：

```
@Test
// OGNL 调用对象的方法:
public void demo1() throws OgnlException{
    OgnlContext context = new OgnlContext();
    Object obj = Ognl.getValue("helloworld.length()", context,
context.getRoot());
    System.out.println(obj);
}
```

在上面的测试案例中，我们调用了字符串的 length 方法，其中就是 OGNL 的表达式，最终的输出结果为 10。

OGNL 除了可以访问对象的方法，还可以访问对象的静态方法。其语法格式如下：

@类的全路径名@方法名称（参数列表）

@类的全路径名@属性名称

演示入门案例

```
@Test
// OGNL 调用对象静态的方法:
public void demo2() throws OgnlException{
    OgnlContext context = new OgnlContext();
    Object obj = Ognl.getValue("@java.lang.Math@random()", context,
context.getRoot());
    System.out.println(obj);
}

@Test
// OGNL 获取数据:
public void demo3() throws OgnlException{
    OgnlContext context = new OgnlContext();

    // 获取 OgnlContext 中的数据:
```



```
/*context.put("name", "张三");  
String name = (String) Ognl.getValue("#{name}", context, context.getRoot());  
System.out.println(name);*/  
  
// 获得 Root 中的数据  
User user = new User();  
user.setName("李四");  
context.setRoot(user);  
  
String name = (String) Ognl.getValue("name", context, context.getRoot());  
System.out.println(name);  
}
```

1.2.2 值栈的概述

1.2.2.1 什么是值栈？

ValueStack 是 Struts 的一个接口，字面意义为值栈，OgnlValueStack 是 ValueStack 的实现类，客户端发起一个请求 struts2 架构会创建一个 action 实例同时创建一个 OgnlValueStack 值栈实例，OgnlValueStack 贯穿整个 Action 的生命周期，struts2 中使用 OGNL 将请求 Action 的参数封装为对象存储到值栈中，并通过 OGNL 表达式读取值栈中的对象属性值。

1.2.2.2 值栈的内部结构：

在 OgnlValueStack 中包括两部分，值栈和 map（即 ognl 上下文）

```
public class OgnlValueStack implements Serializable, ValueStack, ClearableValueStack, MemberAccessVa  
public static final String ION_O  
private static final long JID =  
private static final String IER_K  
private static final long rFac  
CompoundRoot root;  
transient Map<String, Object> context;
```

继承ArrayList
实现压栈和出
栈功能。
存储action实例
及请求的参数
等

即OgnlContext，上下文
中存储了一些引用：
parameters、request、
response、session、
application等。
最重要的是上下文的
Root为CompoundRoot

stack.getName() + ".throw
.xwork2.util.OgnlValueStack.MA
eStack.class);

- Context: 即 OgnlContext 上下文，它是一个 map 结构，上下文中存储了一些引用，parameters、request、session、application 等，上下文的 Root 为 CompoundRoot。

OgnlContext 中的一些引用：

parameters: 该 Map 中包含当前请求的请求参数

request: 该 Map 中包含当前 request 对象中的所有属性

session: 该 Map 中包含当前 session 对象中的所有属性



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

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

- CompoundRoot: 存储了 action 实例, 它作为 OgnlContext 的 Root 对象。

CompoundRoot 继承 ArrayList 实现压栈和出栈功能, 拥有栈的特点, 先进后出, 后进先出, 最后压进栈的数据在栈顶。我们把它称为对象栈。

struts2 对原 OGNL 作出的改进就是 Root 使用 CompoundRoot(自定义栈), 使用 OgnlValueStack 的 findValue 方法可以在 CompoundRoot 中从栈顶向栈底找查找的对象的属性值。

CompoundRoot 作为 OgnlContext 的 Root 对象, 并且在 CompoundRoot 中 action 实例位于栈顶, 当读取 action 的属性值时会先从栈顶对象中找对应的属性, 如果找不到则继续找栈中的其它对象, 如果找到则停止查找。

1.2.2.3 ActionContext 和 ValueStack 的关系:

通过源码查询:

```
public ActionContext createActionContext(HttpServletRequest request,
HttpServletResponse response) {
    ActionContext ctx;
    Integer counter = 1;
    Integer oldCounter = (Integer)
request.getAttribute(CLEANUP_RECURSION_COUNTER);
    if (oldCounter != null) {
        counter = oldCounter + 1;
    }

    ActionContext oldContext = ActionContext.getContext();
    if (oldContext != null) {
        // detected existing context, so we are probably in a forward
        ctx = new ActionContext(new HashMap<String,
Object>(oldContext.getContextMap()));
    } else {
        ValueStack stack = dispatcher.getContainer().getInstance(ValueStackFactory.class).createValueStack();
        stack.getContext().putAll(dispatcher.createContextMap(request,
response, null));
        ctx = new ActionContext(stack.getContext());
    }
    request.setAttribute(CLEANUP_RECURSION_COUNTER, counter);
    ActionContext.setContext(ctx);
    return ctx;
}
```



- * 在创建 ActionContext 的时候 创建 ValueStack 的对象，将 ValueStack 对象给 ActionContext.
- * ActionContext 中有一个 ValueStack 的引用。 ValueStack 中也有一个 ActionContext 的引用.
- * ActionContext 获取 ServletAPI 的时候, 依赖值栈了.

1.2.2.4 获取值栈对象:

【通过 ActionContext 对象获取值栈.】

```
ValueStack stack1 =ActionContext.getContext().getValueStack();
```

【通过 request 域获取值栈.】

```
ValueStack stack2 = (ValueStack) ServletActionContext.getRequest().getAttribute(ServletActionContext.STRUTS_VALUESTACK_KEY);
```

1.2.2.5 操作值栈:

【对 Action 中的属性提供 get 方法的方式】

因为 Action 本身在值栈中， Action 中的属性也就默认在值栈中了，所以我们通过对 Action 的属性提供 get 方法的方式来操作值栈。

【手动操作值栈】

调用值栈的 push 和 set 方法对值栈进行操作

1.2.2.6 从值栈中获取数据

1.2.2.7 EL 能够访问值栈

底层对 Request 对象的 getAttribute 方法进行增强：

```
public class StrutsRequestWrapper extends HttpServletRequestWrapper {  
  
    private static final String REQUEST_WRAPPER_GET_ATTRIBUTE = "__requestWrapper.getAttribute";  
    private final boolean disableRequestAttributeValueStackLookup;  
  
    /**  
     * The constructor  
     * @param req The request  
     */  
    public StrutsRequestWrapper(HttpServletRequest req) {  
        this(req, false);  
    }
```



```
}

/**
 * The constructor
 * @param req The request
 * @param disableRequestAttributeValueStackLookup flag for disabling request
attribute value stack lookup (JSTL accessibility)
 */

public StrutsRequestWrapper(HttpServletRequest req, boolean
disableRequestAttributeValueStackLookup) {
    super(req);
    this.disableRequestAttributeValueStackLookup = =
disableRequestAttributeValueStackLookup;
}

/**
 * Gets the object, looking in the value stack if not found
 *
 * @param key The attribute key
 */
public Object getAttribute(String key) {
    if (key == null) {
        throw new NullPointerException("You must specify a key value");
    }

    if ((disableRequestAttributeValueStackLookup ||
key.startsWith("javax.servlet")) {
        // don't bother with the standard javax.servlet attributes, we can
short-circuit this
        // see WW-953 and the forums post linked in that issue for more info
        return super.getAttribute(key);
    }

    ActionContext ctx = ActionContext.getContext();
    Object attribute = super.getAttribute(key);

    if (ctx != null && attribute == null) {
        boolean alreadyIn =.isTrue((Boolean)
ctx.get(REQUEST_WRAPPER_GET_ATTRIBUTE));

        // note: we don't let # come through or else a request for
        // #attr.foo or #request.foo could cause an endless loop
        if (!alreadyIn && !key.contains("#")) {
            try {

```



```
// If not found, then try the ValueStack
ctx.put(REQUEST_WRAPPER_GET_ATTRIBUTE, Boolean.TRUE);
ValueStack stack = ctx.getValueStack();
if (stack != null) {
    attribute = stack.findValue(key);
}
} finally {
    ctx.put(REQUEST_WRAPPER_GET_ATTRIBUTE, Boolean.FALSE);
}
}
return attribute;
}
```

1.2.3 EL 的特殊字符的使用:

1.2.3.1 #号的使用:

【获取 context 的数据】

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

【用于构建一个 Map 集合】：使用 struts 的 UI 标签的时候。

```
<s:iterator value="#{'aaa':'111','bbb':'222','ccc':'333'}" var="entry">
    <s:property value="key"/>---<s:property value="value"/><br/>
    <s:property value="#entry.key"/>---<s:property value="#entry.value"/><br/>
</s:iterator>

<s:radio list="#{'1':'男','2':'女'}" name="sex"></s:radio>
```

1.2.3.2 %号的使用:

【%强制解析 OGNL 表达式】

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

【%强制不解析 OGNL 表达式】

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

1.2.3.3 \$号的使用:

【在配置文件中使用 OGNL 表达式】

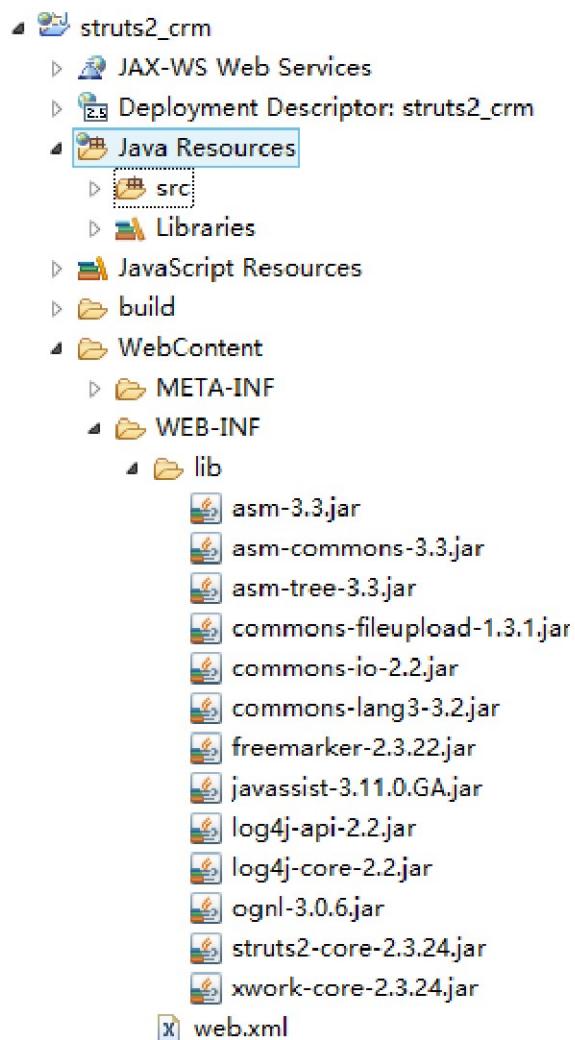
在 struts 的配置文件中使用 .XML 文件 或者 是属性文件。



1.3 案例实现

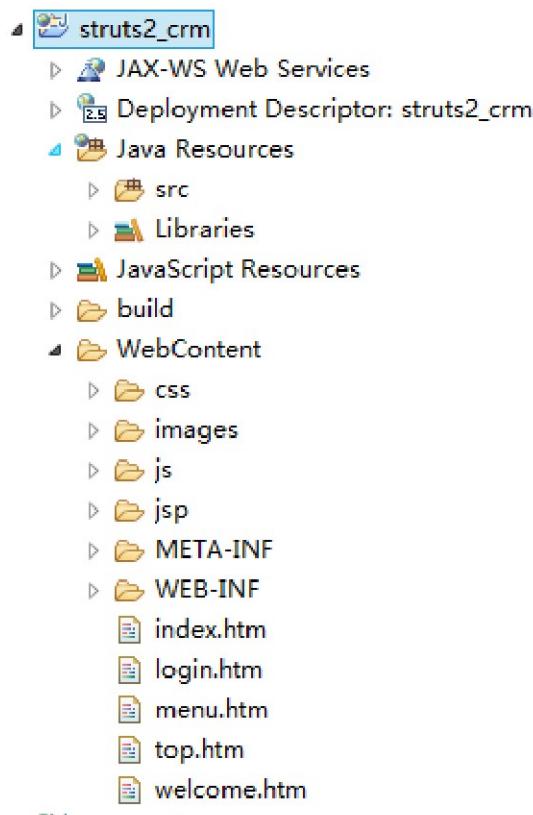
1.3.1 环境准备

1.3.1.1 步骤一：创建 WEB 工程，引入 jar 包

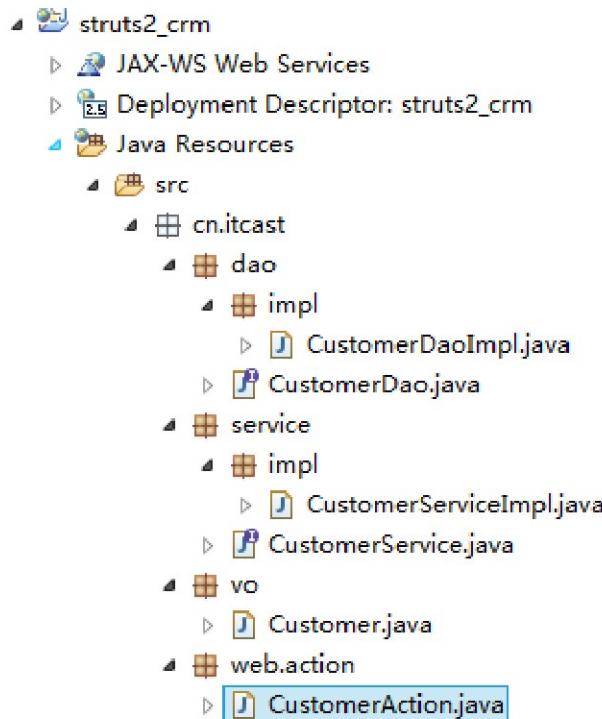




1.3.1.2 步骤二：引入相应的页面



1.3.1.3 步骤三：创建包和相关的类



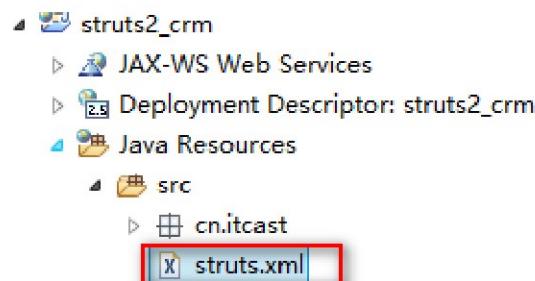


1.3.1.4 步骤四：配置核心过滤器

```
<!-- 配置Struts2的核心过滤器 -->
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

1.3.1.5 步骤五：引入相应的配置文件



1.3.2 代码实现

1.3.2.1 步骤六：修改菜单页面修改提交路径

```
<TR>
    <TD class=menuSmall><A class=style2 href="/struts2_crm/customer_findAll.action"
        target=main>- 客户列表</A></TD>
</TR>
```

1.3.2.2 步骤七：编写 Action 中的代码

```
/*
 * 客户管理的 Action 类
 * @author jt
 *
 */
public class CustomerAction extends ActionSupport{

    /**
     * 查询客户列表的方法:
     * @return
     */
}
```



```
public String findAll(){
    // 创建业务层的类的对象
    CustomerService customerService = new CustomerServiceImpl();
    // 调用业务层的方法查询所有客户
    List<Customer> list = customerService.findAll();
    // 获取值栈并且将数据保存到值栈中
    ActionContext.getContext().getValueStack().set("list", list);
    return "findAll";
}
```

1.3.2.3 步骤八：编写业务层的类

业务层的接口

```
/**
 * 客户管理业务层的接口
 * @author jt
 *
 */
public interface CustomerService {

    List<Customer> findAll();

}
```

业务层的实现类

```
/**
 * 客户管理业务层的实现类
 * @author jt
 *
 */
public class CustomerServiceImpl implements CustomerService {

    @Override
    public List<Customer> findAll() {
        CustomerDao customerDao = new CustomerDaoImpl();
        return customerDao.findAll();
    }

}
```

1.3.2.4 步骤九：引入 Hibernate 的 jar 包和配置文件

引入 jar 包



名称	修改日期	类型	大小
antlr-2.7.7.jar	2016/7/7 9:19	Executable Jar File	435 KB
c3p0-0.9.2.1.jar	2016/7/7 11:18	Executable Jar File	414 KB
dom4j-1.6.1.jar	2016/7/7 9:19	Executable Jar File	307 KB
geronimo-jta_1.1_spec-1.1.1.jar	2016/7/7 9:19	Executable Jar File	16 KB
hibernate-c3p0-5.0.7.Final.jar	2016/7/7 11:18	Executable Jar File	12 KB
hibernate-commons-annotations-5.0....	2016/7/7 9:19	Executable Jar File	74 KB
hibernate-core-5.0.7.Final.jar	2016/7/7 9:19	Executable Jar File	5,453 KB
hibernate-jpa-2.1-api-1.0.0.Final.jar	2016/7/7 9:19	Executable Jar File	111 KB
jandex-2.0.0.Final.jar	2016/7/7 9:19	Executable Jar File	184 KB
javassist-3.18.1-GA.jar	2016/7/7 9:19	Executable Jar File	698 KB
jboss-logging-3.3.0.Final.jar	2016/7/7 9:19	Executable Jar File	66 KB
log4j-1.2.16.jar	2016/7/7 9:19	Executable Jar File	471 KB
mchange-commons-java-0.2.3.4.jar	2016/7/7 11:18	Executable Jar File	568 KB
mysql-connector-java-5.1.7-bin.jar	2016/7/7 9:20	Executable Jar File	694 KB
slf4j-api-1.6.1.jar	2016/7/7 9:19	Executable Jar File	25 KB
slf4j-log4j12-1.7.2.jar	2016/7/7 9:19	Executable Jar File	9 KB

引入配置文件

```
struts2_crm
  └── JAX-WS Web Services
  └── Deployment Descriptor: struts2_crm
  └── Java Resources
    └── src
      └── cn.itcast
        ├── hibernate.cfg.xml
        ├── log4j.properties
        └── struts.xml
```

1.3.2.5 步骤十：添加映射文件

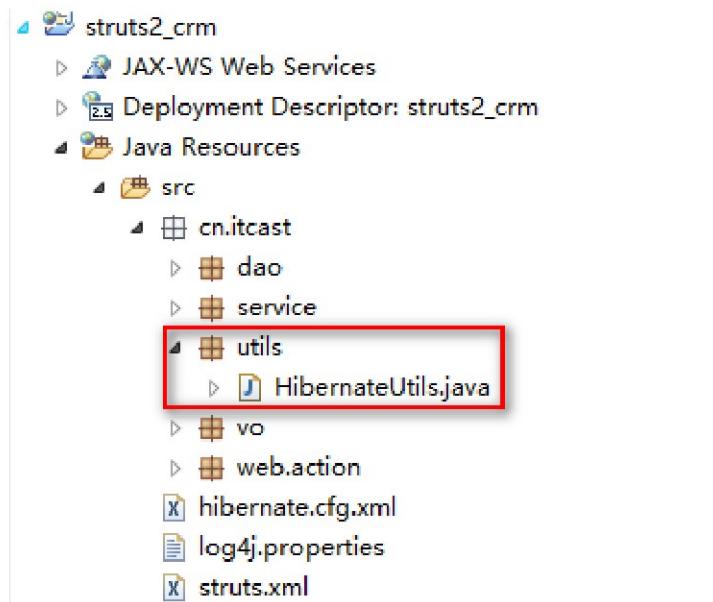
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="cn.itcast.vo.Customer" table="cst_customer">
    <id name="cust_id" column="cust_id">
      <!-- 主键生成策略 -->
      <generator class="native"/>
    </id>
    <property name="cust_name" column="cust_name"/>
    <property name="cust_source" column="cust_source"/>
    <property name="cust_industry" column="cust_industry"/>
  </class>
</hibernate-mapping>
```



```
<property name="cust_level" column="cust_level"/>
<property name="cust_phone" column="cust_phone"/>
<property name="cust_mobile" column="cust_mobile"/>
</class>
</hibernate-mapping>
```

1.3.2.6 步骤十一：引入工具类并修改配置文件



修改配置文件

```
<!-- 必要的配置信息：连接数据库的基本参数 -->
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql:///struts2_day01</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">123</property>

<!-- Hibernate加载映射 -->
<mapping resource="cn/itcast/vo/Customer.hbm.xml"/>
```

1.3.2.7 步骤十二：编写 DAO

编写 DAO 接口

```
/*
 * 客户管理的 DAO 的接口
 * @author jt
 *
 */
public interface CustomerDao {

    List<Customer> findAll();
```



}

编写 DAO 实现类

```
/*
 * 客户管理的 DAO 的实现类
 * @author jt
 *
 */
public class CustomerDaoImpl implements CustomerDao {

    @Override
    /**
     * DAO 中查询所有客户的方法
     */
    public List<Customer> findAll() {
        Session session = HibernateUtils.openSession();
        Transaction tx = session.beginTransaction();

        List<Customer> list = session.createQuery("from Customer").list();

        tx.commit();
        session.close();
        return list;
    }

}
```

1.3.2.8 步骤十三：配置 Action 的有页面跳转

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

    <package name="crm" extends="struts-default" namespace="/">
        <action    name="customer_*"    class="cn.itcast.web.action.CustomerAction"
method="{1}">
            <result name="findAll"/>/jsp/customer/list.jsp</result>
        </action>
    </package>
</struts>
```



1.3.2.9 步骤十四：在页面中显示相应的数据

编写显示数据的代码

```
<TR
    style="FONT-WEIGHT: bold; FONT-STYLE: normal; BACKGROUND-COLOR: #eeeeee; TEXT-DECORATION: none">
    <TD>客户名称</TD>
    <TD>客户级别</TD>
    <TD>客户来源</TD>
    <TD>电话</TD>
    <TD>手机</TD>
    <TD>操作</TD>
</TR>
<s:iterator value="list" var="c">
<TR
    style="FONT-WEIGHT: normal; FONT-STYLE: normal; BACKGROUND-COLOR: white; TEXT-DECORATION: none">
    <TD><s:property value="cust_name"/>-<s:property value="#c.cust_name"/></TD>
    <TD><s:property value="cust_level"/></TD>
    <TD><s:property value="cust_source"/></TD>
    <TD><s:property value="cust_phone"/></TD>
    <TD><s:property value="cust_mobile"/></TD>
    <TD>
        <a href="#">修改</a>
        &nbsp;&nbsp;
        <a href="#">删除</a>
    </TD>
</TR>
</s:iterator>
```

1.3.2.10 步骤十五：测试程序

启动服务器访问页面

The screenshot shows the '客户关系管理系统v1.0' (Customer Relationship Management System v1.0) interface. The left sidebar has a tree menu with '客户管理' expanded, showing '客户列表'. The main content area displays a table of customer information with two entries:

客户名称	客户级别	客户来源	电话	手机	操作
张总	经理	网络推广	15726607618	01055675634	修改 删除
刘总	经理	电话营销	15726607618	01055675634	修改 删除

At the bottom right of the table, there is a pagination control: '共 2 条记录, 0 页, 每页显示 1 条 [前一页] [后一页] 跳到 页 6'.



第1章 Struts2_day04

今日任务

- 使用 Struts2 完成用户登录的权限拦截器的代码编写

教学导航

教学目标	
教学方法	案例驱动法

案例一：

使用**Struts2**完成用户登录的权限拦截器的代码编写

1.1 案例需求

1.1.1 需求的概述

在 CRM 的系统中，有用户登录的功能，如果访问者知道了后台的访问页面的路径，有可能没有进行登录直接就进入到 CRM 系统中。那么对于这类的账号需要进行拦截。

首先需要有登录的功能的实现，如果对于没有登录的用户直接访问后台的 Action，可以实施拦截。



1.2 相关知识点：

1.2.1 Struts2 的拦截器：

1.2.1.1 拦截器的概述

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

在 Webwork 的中文文档的解释为——拦截器是动态拦截 Action 调用的对象。它提供了一种机制可以使开发者可以定义在一个 action 执行的前后执行的代码，也可以在一个 action 执行前阻止其执行。同时也是提供了一种可以提取 action 中可重用的部分的方式。

谈到拦截器，还有一个词大家应该知道——拦截器链（Interceptor Chain，在 Struts 2 中称为拦截器栈 Interceptor Stack）。拦截器链就是将拦截器按一定的顺序联结成一条链。在访问被拦截的方法或字段时，拦截器链中的拦截器就会按其之前定义的顺序被调用。

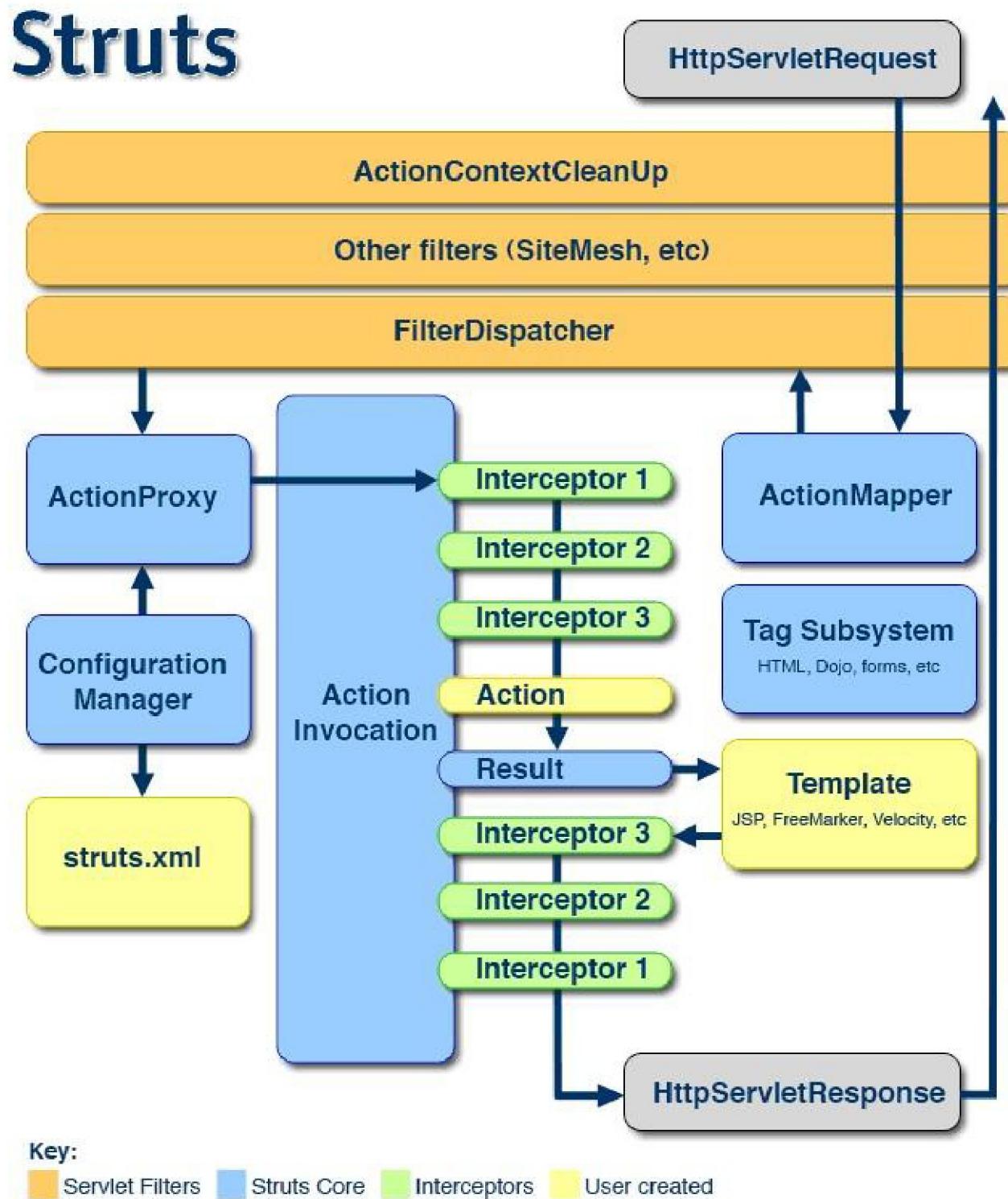
1.2.1.2 拦截器的实现原理：

大部分时候，拦截器方法都是通过代理的方式来调用的。Struts 2 的拦截器实现相对简单。当请求到达 Struts 2 的 ServletDispatcher 时，Struts 2 会查找配置文件，并根据其配置实例化相对的拦截器对象，然后串成一个列表，最后一个一个地调用列表中的拦截器。

Struts2 拦截器是可插拔的，拦截器是 AOP 的一种实现。Struts2 拦截器栈就是将拦截器按一定的顺序联结成一条链。在访问被拦截的方法或字段时，Struts2 拦截器链中的拦截器就会按其之前定义的顺序被调用。



1.2.1.3 Struts2 的执行流程:



1.2.1.4 自定义拦截器

在程序开发过程中，如果需要开发自己的拦截器类，就需要直接或间接的实现



com.opensymphony.xwork2.interceptor.Interceptor 接口。其定义的代码如下：

```
public interface Interceptor extends Serializable {  
    void init();  
    void destroy();  
    String intercept(ActionInvocation invocation) throws Exception;  
}
```

该接口提供了三个方法，其具体介绍如下。

- void init(): 该方法在拦截器被创建后会立即被调用，它在拦截器的生命周期内只被调用一次。可以在该方法中对相关资源进行必要的初始化。
- void destroy(): 该方法与 init 方法相对应，在拦截器实例被销毁之前，将调用该方法来释放和拦截器相关的资源。它在拦截器的生命周期内，也只被调用一次。
- String intercept(ActionInvocation invocation) throws Exception: 该方法是拦截器的核心方法，用来添加真正执行拦截工作的代码，实现具体的拦截操作。它返回一个字符串作为逻辑视图，系统根据返回的字符串跳转到对应的视图资源。每拦截一个动作请求，该方法就会被调用一次。该方法的 ActionInvocation 参数包含了被拦截的 Action 的引用，可以通过该参数的 invoke()方法，将控制权转给下一个拦截器或者转给 Action 的 execute()方法。

如果需要自定义拦截器，只需要实现 Interceptor 接口的三个方法即可。然而在实际开发过程中，除了实现 Interceptor 接口可以自定义拦截器外，更常用的一种方式是继承抽象拦截器类 AbstractInterceptor。该类实现了 Interceptor 接口，并且提供了 init()方法和 destroy()方法的空实现。使用时，可以直接继承该抽象类，而不用实现那些不必要的方法。拦截器类 AbstractInterceptor 中定义的方法如下所示：

```
public abstract class AbstractInterceptor implements Interceptor {  
    public void init() {}  
    public void destroy() {}  
    public abstract String intercept(ActionInvocation invocation)  
        throws Exception;  
}
```

从上述代码中可以看出，AbstractInterceptor 类已经实现了 Interceptor 接口的所有方法，一般情况下，只需继承 AbstractInterceptor 类，实现 intercept()方法就可以创建自定义拦截器。

只有当自定义的拦截器需要打开系统资源时，才需要覆盖 AbstractInterceptor 类的 init()方法和 destroy()方法。与实现 Interceptor 接口相比，继承 AbstractInterceptor 类的方法更为简单。

1.2.1.5 拦截器的配置

1、拦截器

要想让拦截器起作用，首先要对它进行配置。拦截器的配置是在 struts.xml 文件中完成的，它通常以<interceptor>标签开头，以</interceptor>标签结束。定义拦截器的语法格式如下：

```
<interceptor name="interceptorName" class="interceptorClass">  
    <param name="paramName">paramValue</param>  
</interceptor>
```

上述语法格式中，name 属性用来指定拦截器的名称，class 属性用于指定拦截器的实现类。有时，在定义拦截器时需要传入参数，这时需要使用<param>标签，其中 name 属性用来指定参数的名称，paramValue 表示参数的值。



2、拦截器栈

3、在实际开发中，经常需要在 Action 执行前同时执行多个拦截动作，如：用户登录检查、登录日志记录以及权限检查等，这时，可以把多个拦截器组成一个拦截器栈。在使用时，可以将栈内的多个拦截器当成一个整体来引用。当拦截器栈被附加到一个 Action 上时，在执行 Action 之前必须先执行拦截器栈中的每一个拦截器。

定义拦截器栈使用<interceptors>元素和<interceptor-stack>子元素，当配置多个拦截器时，需要使用<interceptor-ref>元素来指定多个拦截器，配置语法如下：

```
<interceptors>
    <interceptor-stack name="interceptorStackName">
        <interceptor-ref name="interceptorName" />
        ...
    </interceptor-stack>
</interceptors>
```

在上述语法中，interceptorStackName 值表示配置的拦截器栈的名称；interceptorName 值表示拦截器的名称。除此之外，在一个拦截器栈中还可以包含另一个拦截器栈，示例代码如下：

```
<package name="default" namespace="/" extends="struts-default">
    <!-- 声明拦截器 -->
    <interceptors>
        <interceptor name="interceptor1" class="interceptorClass"/>
        <interceptor name="interceptor2" class="interceptorClass"/>
        <!-- 定义一个拦截器栈 myStack，该拦截器栈中包含两个拦截器和一个拦截器栈 -->
        <interceptor-stack name="myStack">
            <interceptor-ref name="defaultStack" />
            <interceptor-ref name="interceptor1" />
            <interceptor-ref name="interceptor2" />
        </interceptor-stack>
    </interceptors>
</package>
```

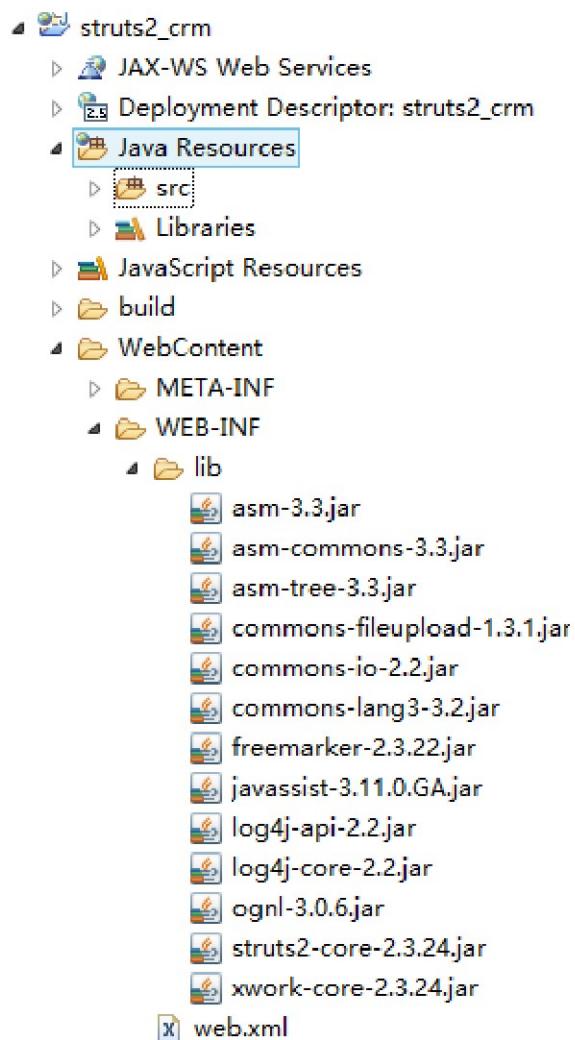
在上述代码中，定义的拦截器栈是 myStack，在 myStack 栈中，除了引用了两个自定义的拦截器 interceptor1 和 interceptor2 外，还引用了一个内置拦截器栈 defaultStack，这个拦截器是必须要引入的。



1.3 案例实现

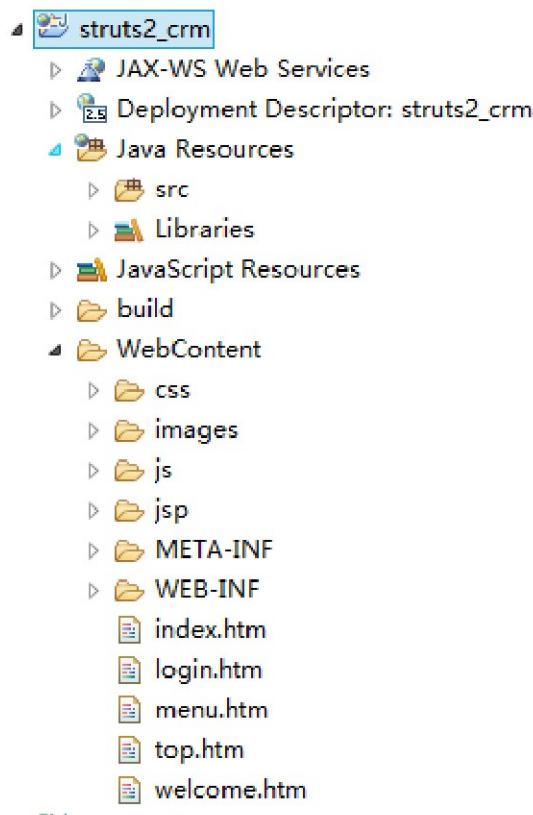
1.3.1 环境搭建

1.3.1.1 步骤一：创建 WEB 工程，引入 jar 包

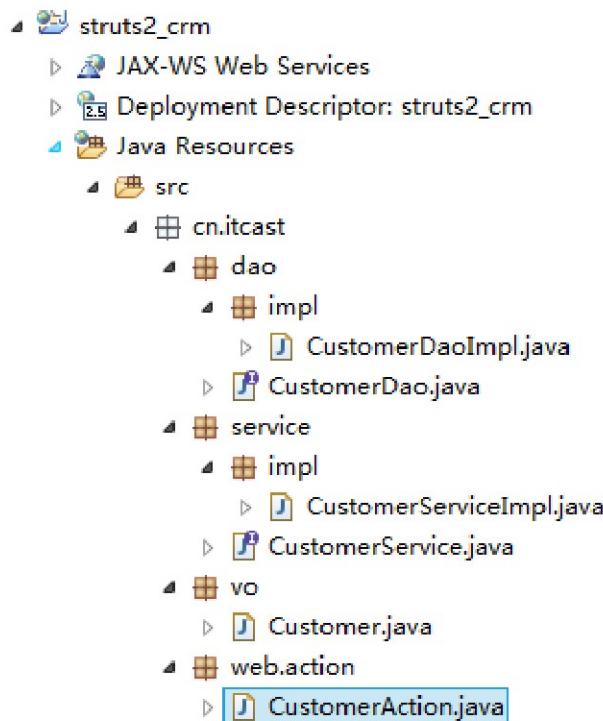




1.3.1.2 步骤二：引入相应的页面



1.3.1.3 步骤三：创建包和相关的类



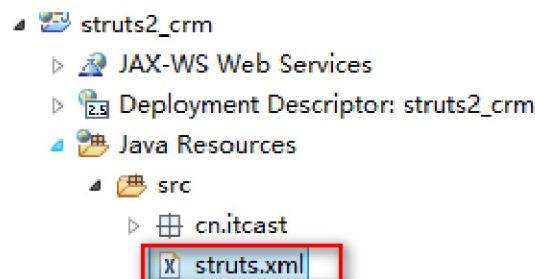


1.3.1.4 步骤四：配置核心过滤器

```
<!-- 配置Struts2的核心过滤器 -->
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

1.3.1.5 步骤五：引入相应的配置文件



1.3.2 代码实现

1.3.2.1 修改登录的页面

```
<FORM id=form1 name=form1 action="${ pageContext.request.contextPath }/user_login.action" method=post target=_parent>

<TR>
    <TD style="HEIGHT: 28px" width=80>登录名: </TD>
    <TD style="HEIGHT: 28px" width=150><INPUT id=txtName
        style="WIDTH: 130px" name="user_name"></TD>
    <TD style="HEIGHT: 28px" width=370><SPAN
        id=RequiredFieldValidator3
        style="FONT-WEIGHT: bold; VISIBILITY: hidden; COLOR: white">请输入登录名</SPAN></TD></TR>

<TR>
    <TD style="HEIGHT: 28px">登录密码: </TD>
    <TD style="HEIGHT: 28px"><INPUT id=txtPwd style="WIDTH: 130px"
        type=password name="user_pass"></TD>
    <TD style="HEIGHT: 28px"><SPAN id=RequiredFieldValidator4
        style="FONT-WEIGHT: bold; VISIBILITY: hidden; COLOR: white">请输入密码</SPAN></TD></TR>
```

1.3.2.2 编写 Action

```
package cn.itcast.crm.web.action;
```



```
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

import cn.itcast.crm.domain.User;
import cn.itcast.crm.service.UserService;
import cn.itcast.crm.service.impl.UserServiceImpl;

public class UserAction extends ActionSupport implements ModelDriven<User>{
    // 模型驱动需要使用的对象:
    private User user = new User();

    @Override
    public User getModel() {
        // TODO Auto-generated method stub
        return user;
    }

    /**
     * 编写一个执行的方法:login
     */
    public String login(){
        UserService userService = new UserServiceImpl();
        User existUser = userService.login(user);
        // 判断是否登录成功:
        if(existUser == null){
            // 登录失败
            this.addActionError("用户名或密码错误!");
            return LOGIN;
        }else{
            // 登录成功.将用户信息存入到 session 中.
            ActionContext.getContext().getSession().put("existUser", existUser);
            return SUCCESS;
        }
    }
}
```

1.3.2.3 编写业务层

【业务层接口】

```
package cn.itcast.crm.service;

import cn.itcast.crm.domain.User;
```



```
public interface UserService {  
  
    User login(User user);  
  
}
```

【业务层实现类】

```
package cn.itcast.crm.service.impl;  
  
import cn.itcast.crm.dao.UserDao;  
import cn.itcast.crm.dao.impl.UserDaoImpl;  
import cn.itcast.crm.domain.User;  
import cn.itcast.crm.service.UserService;  
  
public class UserServiceImpl implements UserService {  
  
    @Override  
    public User login(User user) {  
        UserDao userDao = new UserDaoImpl();  
        return userDao.login(user);  
    }  
  
}
```

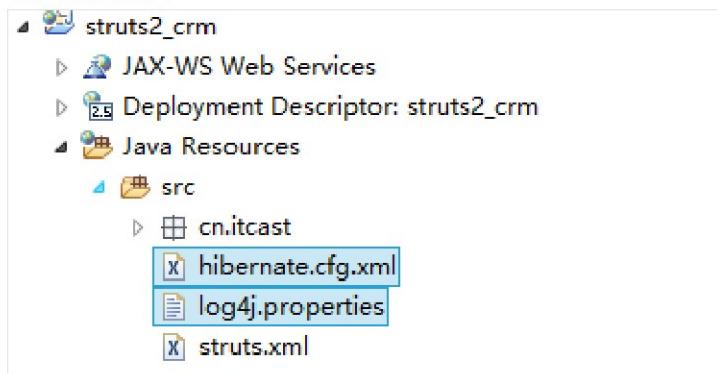
1.3.2.4 步骤九：引入 Hibernate 的 jar 包和配置文件

引入 jar 包

名称	修改日期	类型	大小
antlr-2.7.7.jar	2016/7/7 9:19	Executable Jar File	435 KB
c3p0-0.9.2.1.jar	2016/7/7 11:18	Executable Jar File	414 KB
dom4j-1.6.1.jar	2016/7/7 9:19	Executable Jar File	307 KB
geronimo-jta_1.1_spec-1.1.1.jar	2016/7/7 9:19	Executable Jar File	16 KB
hibernate-c3p0-5.0.7.Final.jar	2016/7/7 11:18	Executable Jar File	12 KB
hibernate-commons-annotations-5.0....	2016/7/7 9:19	Executable Jar File	74 KB
hibernate-core-5.0.7.Final.jar	2016/7/7 9:19	Executable Jar File	5,453 KB
hibernate-jpa-2.1-api-1.0.0.Final.jar	2016/7/7 9:19	Executable Jar File	111 KB
jandex-2.0.0.Final.jar	2016/7/7 9:19	Executable Jar File	184 KB
javassist-3.18.1-GA.jar	2016/7/7 9:19	Executable Jar File	698 KB
jboss-logging-3.3.0.Final.jar	2016/7/7 9:19	Executable Jar File	66 KB
log4j-1.2.16.jar	2016/7/7 9:19	Executable Jar File	471 KB
mchange-commons-java-0.2.3.4.jar	2016/7/7 11:18	Executable Jar File	568 KB
mysql-connector-java-5.1.7-bin.jar	2016/7/7 9:20	Executable Jar File	694 KB
slf4j-api-1.6.1.jar	2016/7/7 9:19	Executable Jar File	25 KB
slf4j-log4j12-1.7.2.jar	2016/7/7 9:19	Executable Jar File	9 KB



引入配置文件



1.3.2.5 步骤十：添加映射文件

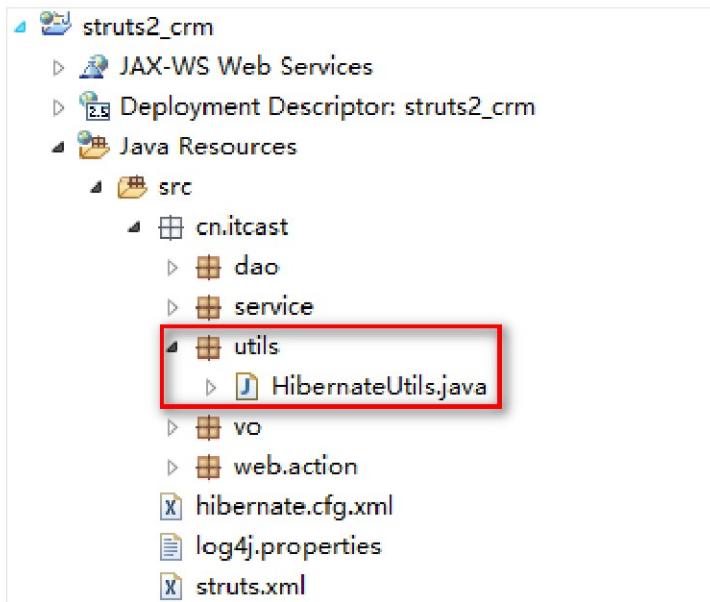
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <!-- 建立类和表的映射 -->
    <!--
        class 标签:
            * name 属性:类的全路径
            * table 属性: 表名
    -->
    <class name="cn.itcast.crm.domain.User" table="cst_user">
        <!-- 类中的属性与表中的字段的映射 -->
        <!-- id 标签: 用来映射 类中的标识属性 与 表中主键建立映射 -->
        <!-- name: 类中的属性名 column: 表中的字段名称 -->
        <id name="user_id" column="user_id">
            <!-- 主键生成策略 -->
            <generator class="native"/>
        </id>

        <!-- 建立普通属性的映射 -->
        <!-- name: 类中的属性名 column: 表中的字段名称 -->
        <!--
            type 属性:数据类型
            1. hibernate 类型:string
            2. Java 类型 :java.lang.String
            3. SQL 类型      :<column name="cust_name" sql-type="varchar"></column>
        -->
        <property name="user_name" column="user_name" length="32"/>
        <property name="user_pass" column="user_pass" length="32"/>
    </class>
```



```
</hibernate-mapping>
```

1.3.2.6 步骤十一：引入工具类并修改配置文件



修改配置文件

```
<!-- 必要的配置信息：连接数据库的基本参数 -->
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql:///struts2_day01</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">123</property>

<!-- 加载映射文件 -->
<mapping resource="cn/itcast/crm/domain/User.hbm.xml"/>
```

1.3.2.7 编写 DAO

【DAO 的接口】

```
package cn.itcast.crm.dao;

import cn.itcast.crm.domain.User;

public interface UserDao {

    User login(User user);

}
```

【DAO 的实现类】

```
package cn.itcast.crm.dao.impl;
```



```
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import cn.itcast.crm.dao.UserDao;
import cn.itcast.crm.domain.User;
import cn.itcast.crm.utils.HibernateUtils;

public class UserDaoImpl implements UserDao {

    @Override
    public User login(User user) {
        Session session = HibernateUtils.openSession();
        Transaction tx = session.beginTransaction();

        Query query = session.createQuery("from User where user_name = ? and
user_pass = ?");
        query.setParameter(0, user.getUser_name());
        query.setParameter(1, user.getUser_pass());
        User existUser = (User) query.uniqueResult();

        tx.commit();
        session.close();
        return existUser;
    }

}
```

1.3.2.8 配置 Action

```
<action name="user_*" class="cn.itcast.crm.web.action.UserAction" method="{1}">
    <result name="success">/index.htm</result>
</action>
```

1.3.2.9 测试登录功能

启动服务器，访问页面：



登录成功后：



1.3.2.10 编写登录权限拦截器

```
package cn.itcast.crm.web.interceptor;

import org.apache.struts2.ServletActionContext;

import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.interceptor.MethodFilterInterceptor;

import cn.itcast.crm.domain.User;

public class PrivilegeInterceptor extends MethodFilterInterceptor {

    @Override
    protected String doIntercept(ActionInvocation invocation) throws Exception {
        // 判断是否已经登录了.
        User existUser = (User)
ServletActionContext.getRequest().getSession().getAttribute("existUser");
    }
}
```



```
if(existUser == null){  
    // 没有登录:  
    ActionSupport actionSupport = (ActionSupport) invocation.getAction();  
    actionSupport.addActionError("没有登录，没有权限访问！");  
    return actionSupport.LOGIN;  
}  
else{  
    // 已经登录:放行.  
    return invocation.invoke();  
}  
}  
}
```

1.3.2.11 配置拦截器

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE struts PUBLIC  
"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"  
"http://struts.apache.org/dtds/struts-2.3.dtd">  
  
<struts>  
  
<package name="crm" extends="struts-default" namespace="/">  
    <!-- 定义拦截器 -->  
    <!-- <interceptors>  
        <interceptor  
            name="privilegeInterceptor"  
            class="cn.itcast.crm.web.interceptor.PrivilegeInterceptor"/>  
    </interceptors> -->  
  
    <interceptors>  
        <interceptor  
            name="privilegeInterceptor"  
            class="cn.itcast.crm.web.interceptor.PrivilegeInterceptor"/>  
  
        <!-- 定义拦截器栈 -->  
        <interceptor-stack name="myStack">  
            <interceptor-ref name="privilegeInterceptor"/>  
            <interceptor-ref name="defaultStack"/>  
        </interceptor-stack>  
    </interceptors>  
  
<global-results>  
    <result name="login"/>/login.jsp</result>  
</global-results>
```



```
<action name="customer_*" class="cn.itcast.crm.web.action.CustomerAction"
method="{1}">
    <result name="findAll"/>/jsp/customer/list.jsp</result>
    <result name="saveSuccess"
type="redirectAction">customer_findAll.action</result>

    <!-- 如果 Action 中定义了自定义拦截器，那么默认的拦截器就都不执行了。 -->
    <interceptor-ref name="myStack"/>
    <!-- <interceptor-ref name="privilegeInterceptor"/>
    <interceptor-ref name="defaultStack"/> -->
</action>

<action name="user_*" class="cn.itcast.crm.web.action.UserAction"
method="{1}">
    <result name="success"/>/index.htm</result>
</action>
</package>
</struts>
```

第2章 总结

2.1 Struts2 的标签库

对于一个 MVC 框架而言，重点是实现两部分：业务逻辑控制器部分和视图页面部分。Struts2 作为一个优秀的 MVC 框架，也把重点放在了这两部分上。控制器主要由 Action 来提供支持，而视图则是由大量的标签来提供支持。接下来将针对 Struts2 标签库的构成和常用标签的使用进行详细的讲解。

2.1.1 Struts2 标签库概述

2.1.1.1 Struts2 标签库概述

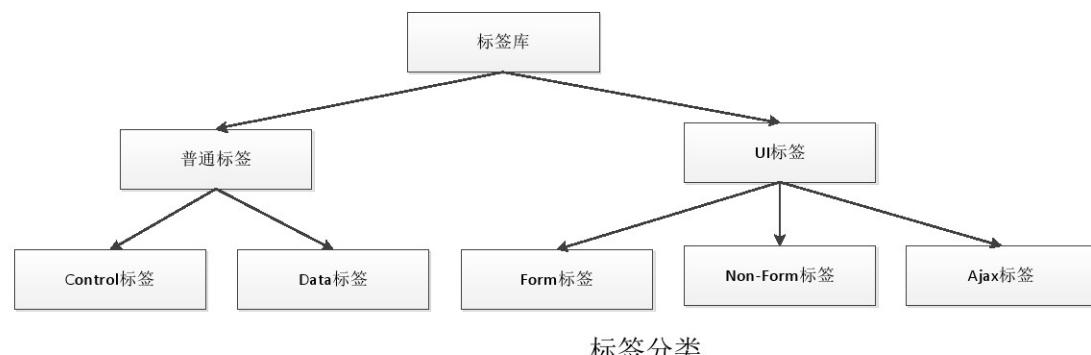
在 JavaWeb 中，Struts2 标签库是一个比较完善，而且功能强大的标签库，它将所有标签都统一到一个标签库中，从而简化了标签的使用，它还提供主题和模板的支持，极大地简化了视图页面代码的编写，同时它还提供对 Ajax 的支持，大大的丰富了视图的表现效果。与 JSTL(JSP Standard Library，JSP 标准标签库)相比，Struts2 标签库更加易用和强大。



2.1.1.2 Struts2 标签库的分类

早期的 JSP 页面需要嵌入大量的 Java 脚本来进行输出，这样使得一个简单的 JSP 页面加入了大量的代码，不利于代码的可维护性和可读性。随着技术的发展，逐渐的采用标签库来进行 JSP 页面的开发，这使得 JSP 页面能够在很短的时间内开发完成，而且代码通俗易懂，大大的方便了开发者，Struts2 的标签库就是这样发展起来的。

Struts2 框架对整个标签库进行了分类，按其功能大致可分为两类，如图所示。



由图中可以看出，Struts2 标签库主要分为两类：普通标签和 UI 标签。普通标签主要是在页面生成时，控制执行的流程。UI 标签则是以丰富而可复用的 HTML 文件来显示数据。

普通标签又分为控制标签（Control Tags）和数据标签（Data Tags）。控制标签用来完成条件逻辑、循环逻辑的控制，也可用来做集合的操作。数据标签用来输出后台的数据和完成其他数据访问功能。

UI 标签又分为表单标签（Form Tags）、非表单标签（Non-Form Tags）和 Ajax 标签。表单标签主要用来生成 HTML 页面中的表单元素，非表单标签主要用来生成 HTML 的<div>标签及输出 Action 中封装的信息等。Ajax 标签主要用来提供 Ajax 技术支持。

2.1.1.3 Struts2 标签的使用

Struts2 标签库被定义在 struts-tags.tld 文件中，我们可以在 struts-core-2.3.24.jar 中的 META-INF 目录下找到它。要使用 struts2 的标签库，一般只需在 JSP 文件使用 taglib 指令导入 Struts2 标签库，具体代码如下：

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

在上述代码中，taglib 指令的 uri 属性用于指定引入标签库描述符文件的 URI，prefix 属性用于指定引入标签库描述符文件的前缀。需要注意的是，在 JSP 文件中，所有的 Struts2 标签库的使用“s”前缀。

2.1.2 Struts2 的控制标签

在程序开发中，经常要用流程控制实现分支、循环等操作，为此，Struts2 标签库中提供了控制标签，常用的逻辑控制标签主要包括：<s:if>、<s:elseif>、<s:else>和<s:iterator>等。本节将针对这四个常用标签进行详细的讲解。



2.1.2.1 <s:if>、<s:elseif>、<s:else>标签

与多数编程语言中的 if、elseif 和 else 语句的功能类似，<s:if>、<s:elseif>、<s:else>这三个标签用于程序的分支逻辑控制。其中，只有<s:if>标签可以单独使用，而<s:elseif>、<s:else>都必须与<s:if>标签结合使用，其语法格式如下所示：

```
<s:if test = "表达式 1">
    标签体
</s:if>
<s: elseif test = "表达式 2">
    标签体
</s: elseif >
<s: else >
    标签体
</s :else >
```

上述语法格式中，<s:if>、和<s:elseif>标签必须指定 test 属性，该属性用于设置标签的判断条件，其值为 boolean 型的条件表达式。

2.1.2.2 <s:iterator>标签

<s:iterator>标签主要用于对集合中的数据进行迭代，它可以根据条件遍历集合中的数据。
<s:iterator>标签的属性及相关说明如表所示。

Iterator 标签的属性

属性	是否必须	默认值	类型	描述
begin	否	0	Integer	迭代数组或集合的起始位置
end	否	数组或集合的长度大小减 1，假如 step 为负则为 0	Integer	迭代数组或集合的结束位置
status	否	False	Boolean	迭代过程中的状态
step	否	1	Integer	指定每一次迭代后索引增加的值
value	否	无	String	迭代的数组或集合对象
var	否	无	String	将生成的 Iterator 设置为 page 范围的属性
id	否	无	String	指定了集合元素的 id，现已用 var 代替

在表中，如果在<s:iterator>标签中指定 status 属性，那么通过该属性可以获取迭代过程中的状态信息，如：元素数、当前索引值等。通过 status 属性获取信息的方法如表 4-2 所示（假设其属性值为 st）。

status 获取状态信息

方法	说明
st.count	返回当前已经遍历的集合元素的个数
st.first	返回当前遍历元素是否为集合的第一个元素
st.last	返回当前遍历元素是否为集合的最后一个元素



st.index	返回遍历元素的当前索引值
st.even	返回当前遍历的元素的索引是否为偶数
st.odd	返回当前遍历的元素的索引是否为奇数

为了让大家更好的掌握<s:if>、<s:else>和<s:iterator>标签的使用，接下来，通过一个案例来演示。

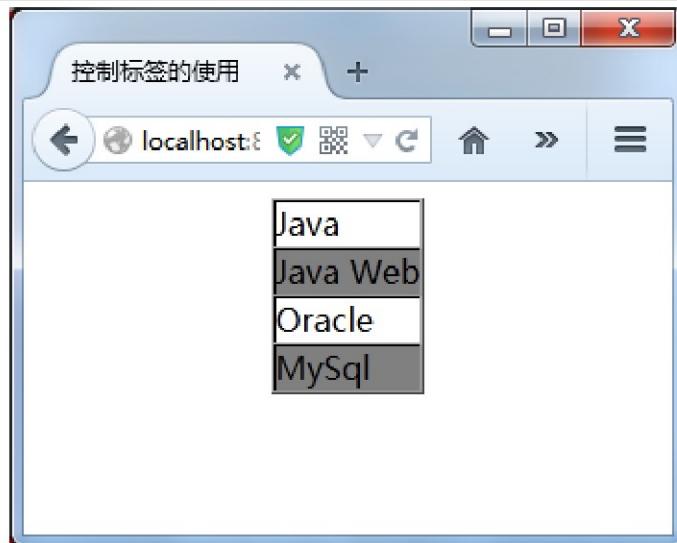
在 Eclipse 中创建 Web 项目，将 Struts2 框架所需的 jar 包添加到 WEB-INF 目录下的 lib 文件夹中，然后在 WEB-INF 目录下添加 web.xml 文件，并在其中注册核心过滤器和首页信息，在 WebContent 下创建一个名称为 iteratorTags.jsp 文件，如文件所示。

iteratorTags.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib prefix="s" uri="/struts-tags" %>
4 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5             "http://www.w3.org/TR/html4/loose.dtd">
6 <html>
7 <head>
8 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9 <title>控制标签的使用</title>
10 </head>
11 <body>
12 <center>
13     <table border="1px" cellpadding="0" cellspacing="0">
14         <s:iterator var = "name"
15             value="{ 'Java', 'Java Web', 'Oracle', 'MySql' }"
16             status="st">
17             <s:if test="#st.odd">
18                 <tr style="background-color: white;">
19                     <td><s:property value = "name"/></td>
20                 </tr>
21             </s:if>
22             <s:else>
23                 <tr style="background-color: gray;">
24                     <td><s:property value = "name"/></td>
25                 </tr>
26             </s:else>
27         </s:iterator>
28     </table>
29 </center>
30 </body>
31 </html>
```

在文件的 table 标签内，首先使用<s:iterator>标签来循环输出集合中的值，然后使用 status.odd 方法获取的值，作为<s:if>、<s:else>标签的判断条件，来对行数显示进行控制。

在浏览器地址栏中输入“<http://localhost:8080/chapter04/iteratorTags.jsp>”，成功访问后，浏览器的显示效果如图所示。



<s:if>、<s:else>和<s:iterator>标签的使用效果

从图中的运行结果可以看出，表格的奇数行变为了白色，偶数行变为灰色。这是因为在文件中，使用<s:iterator>遍历新创建的 List 集合时，通过判断其所在索引的奇偶来决定表格的颜色。

2.1.3 Struts2 的数据标签

在 Struts2 标签库中，数据标签主要用于各种数据访问相关的功能以及 Action 的调用等。常用的数据标签有<s:property>、<s:a>、<s:debug>、<s:include>、<s:param>等。

2.1.3.1 <s:property>标签

<s:property>标签用于输出指定的值，通常输出的是 value 属性指定的值，<s:property>标签的属性及属性说明如下所示：

- id: 可选属性，指定该元素的标识。
- default: 可选属性，如果要输出的属性值为 null，则显示 default 属性的指定值。
- escape: 可选属性，指定是否忽略 HTML 代码。
- value: 可选属性，指定需要输出的属性值，如果没有指定该属性，则默认输出 ValueStack 栈顶的值（关于值栈内容会在下一章节进行讲解）。

接下来，编写一个 propertyTags.jsp 页面，来演示 property 标签的使用，如文件 4-2 所示。

propertyTags.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib prefix="s" uri="/struts-tags" %>
4 <html>
5 <head>
6 <title>property 标签</title>
7 </head>
8 <body>
9   输出字符串:
10  <s:property value=""www.itcast.cn""/><br>
```



```
11 忽略 HTML 代码:  
12 <s:property value=''<h3>www.itcast.cn</h3>'' escape="true"/><br>  
13 不忽略 HTML 代码:  
14 <s:property value=''<h3>www.itcast.cn</h3>'' escape="false"/><br>  
15 输出默认值:  
16 <s:property value="" default="true"/><br>  
17 </body>  
18 </html>
```

在文件中，定义了四个不同属性的<s:property>标签。第一个标签中，只有一个 value 属性，所以会直接输出 value 值；第二个标签，使用了 value 和 escape 两个属性，其中 value 属性值中包含了 html 标签，但其 escape 属性值为 true，表示忽略 HTML 代码，所以会直接输出 value 值；第三个标签，同样使用了 value 和 escape 两个属性，其 value 属性值中依然包含了 html 标签，但其 escape 属性值为 false，表示不能忽略 HTML 代码，所以输出的 value 值为 3 号标题的值；最后一个标签，使用了 value 和 default 两个属性，但 value 的值为空，并且指定了 default 属性，所以最后会输出 default 属性指定的值。

在浏览器地址栏中输入“<http://localhost:8080/chapter04/propertyTags.jsp>”，成功访问后，浏览器的显示结果如图所示。



property 标签的输出

从图中可以看出，property 标签的属性不同，其输出的结果也不同，大家在使用此标签时一定要注意其属性的设置。

2.1.3.2 <s:a>标签

<s:a>标签用于构造 HTML 页面中的超链接，其使用方式与 HTML 中的<a>标签类似。<s:a>标签的属性及相关说明如表 4-3 所示。

表1-1 a 标签的常用属性及描述

属性	是否必须	类型	描述
action	否	String	指定超链接 Action 地址
href	否	String	超链接地址



namespace	否	String	指定 Action 地址
id	否	String	指定其 id
method	否	String	指定 Action 调用方法

<s:a>标签的使用格式，如下所示：

```
<s:a href="链接地址"></s:a>
<s:a namespace="" action="">itcast.cn</s:a>
```

2.1.3.3 <s:debug>标签

<s:debug>标签用于在调试程序时输出更多的调试信息，主要输出 ValueStack 和 StackContext 中的信息，该标签只有一个 id 属性，且一般不使用。

在使用 debug 标签后，网页中会生成一个[Debug]的链接，单击该链接，网页中将输出各种服务器对象的信息，如图 4-4 所示。

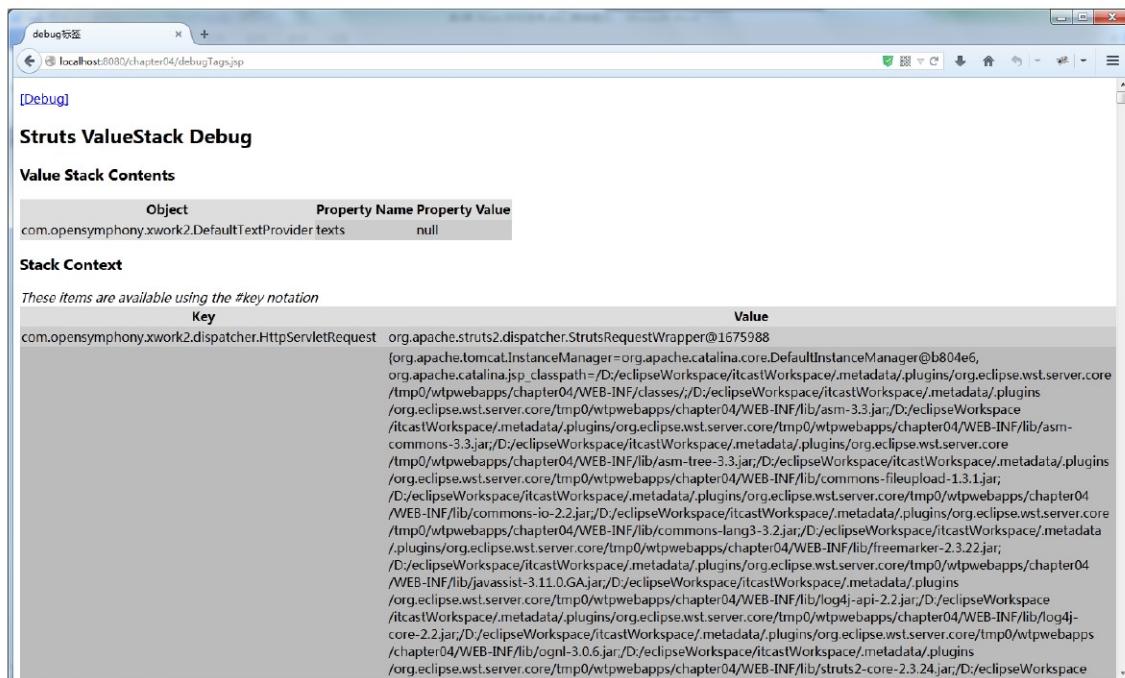


图4-1 debug 标签

2.1.4 Struts2 的模板和主题

Struts2 的 UI 标签都是基于模板和主题的。所谓模板，就是一些代码，Struts2 标签使用这些代码渲染生成相应的 HTML 代码。模板是一个 UI 标签的外在表现形式，并且每个标签都会有自己的对应的模板。如果为所有的 UI 标签提供样式和视觉效果相似的模板，那么这一系列的模板就形成了一个主题。

Struts2 默认提供了 4 种主题，分别为 simple、xhtml、css_xhtml 和 Ajax。

- **simple 主题：**这是最简单的主题，使用该主题时，每个 UI 标签只生成最基本的 HTML 元素，没有任何附加功能。
- **xhtml 主题：**这是 Struts2 的默认主题，它对 simple 主题进行了扩展，提供了布局功能、Label 显示名称、以及与验证框架和国际化框架的集成。



- `css_xhtml`：该主题是对 `xhtml` 的扩展，在 `xhtml` 的基础之上添加对 CSS 的支持和控制。
- `Ajax`：继承自 `xhtml`，提供 Ajax 支持。

这 4 种内建主题中，`xhtml` 为默认主题，但 `xhtml` 有一定的局限性。因为它使用表格进行布局，并且只支持每一行放一个表单项，这样一来，一旦遇到复杂的页面布局，`xhtml` 就难以胜任了。此时，就需要改变 Struts2 的默认主题。

通常，通过设置常量 `struts.ui.theme`，来改变默认主题，具体做法是在 `struts.xml` 或者 `struts.properties` 文件中增加相应的配置。比如想要设置使用 `simple` 的主题，那么需要在 `struts.xml` 中增加如下配置：

```
<constant name="struts.ui.theme" value="simple"/>
```

或者在 `struts.properties` 文件中增加如下配置：

```
struts.ui.theme = simple
```

2.1.5 Struts2 的表单标签

Struts2 的表单标签用来向服务器提交用户输入的信息，绝大多数的表单标签都有其对应的 HTML 标签，通过表单标签可以简化表单开发，还可以实现 HTML 中难以实现的功能。大家可以结合 HTML 的标签对比学习 Struts2 的表单标签。

2.1.5.1 表单标签的公共属性

Struts2 的表单标签用来向服务器提交用户输入信息，在 `org.apache.struts2.components` 包中都有一个对应的类，所有表单标签对应的类都继承自 `UIBean` 类。`UIBean` 类提供了一组公共属性，这些属性是完全通用的。如表 4-4 所示。

表1-2 表单标签的通用属性

属性名	主题	数据类型	说明
<code>title</code>	<code>simple</code>	<code>String</code>	设置表单元素的 <code>title</code> 属性
<code>disabled</code>	<code>simple</code>	<code>String</code>	设置表单元素是否可用
<code>label</code>	<code>xhtml</code>	<code>String</code>	设置表单元素的 <code>label</code> 属性
<code>labelPosition</code>	<code>xhtml</code>	<code>String</code>	设置 <code>label</code> 元素的显示位置，可选值： <code>top</code> 和 <code>left</code> (默认)
<code>name</code>	<code>simple</code>	<code>String</code>	设置表单元素的 <code>name</code> 属性，与 Action 中的属性名对应
<code>value</code>	<code>simple</code>	<code>String</code>	设置表单元素的值
<code>cssClass</code>	<code>simple</code>	<code>String</code>	设置表单元素的 <code>class</code>
<code>cssStyle</code>	<code>simple</code>	<code>String</code>	设置表单元素的 <code>style</code> 属性
<code>required</code>	<code>xhtml</code>	<code>Boolean</code>	设置表单元素为必填项
<code>requiredposition</code>	<code>xhtml</code>	<code>String</code>	设置必填标记(默认为*)相对于 <code>label</code> 元素的位置，可选值： <code>left</code> 和 <code>right</code> (默认)
<code>tabindex</code>	<code>simple</code>	<code>String</code>	设置表单元素的 <code>tabindex</code> 属性

除了这些常用的通用属性外，还有很多其它属性。由于篇幅有限，这里就不一一列举。需要注意的是，表单标签的 `name` 和 `value` 属性基本等同于 HTML 组件的 `name` 和 `value`，但是也有些不同的地方：表单标签在生成 HTML 的时候，如果标签没有设置 `value` 属性的话，就会从值栈中按照 `name`



获取相应的值，把这个值设置成的 HTML 组件的 value。简单的说，就是表单标签的 value 在生成 HTML 的时候会自动设置值，其值从值栈中获取。关于值栈问题将在下一章节进行讲解。

2.1.5.2 <s:form>标签

<s:form>标签用来呈现 HTML 语言中的表单元素，其常用属性如表所示。

<form>标签的常用属性及描述

属性名	是否必填	类型	说明
action	否	String	指定提交时对应的 action，不需要.action 后缀
enctype	否	String	HTML 表单 enctype 属性
method	否	String	HTML 表单 method 属性
namespace	否	String	所提交 action 的命名空间

在使用<s:form>标签时，一般会包含其它的表单元素，如 textfield, radio 等标签，通过这些表单元素对应的 name 属性，在提交表单时，将其作为参数传入 Struts2 框架进行处理。

2.1.5.3 <s:submit>标签

<s:submit>标签主要用于产生 HTML 中的提交按钮，该表单元素中，可以指定提交时的 Action 对应的方法。通常与<s:form>标签一起使用，该标签的常用属性如表所示。

<s:submit>标签的常用属性

属性名	是否必填	类型	说明
action	否	String	指定提交时对应的 action
method	否	String	指定 action 中调用的方法

2.1.5.4 <s:textfield>和<s:textarea>标签

<s:textfield>和<s:textarea>标签的作用比较相似，都用于创建文本框，区别在于<s:textfield>创建的是单行文本框，而<textarea>创建的是多行文本框。二者使用也比较简单，一般指定其 label 和 name 属性即可。两个标签的用法如下所示：

<s:textfield>标签的用法：

```
<s:textfield label="用户名" name="username"/>
```

<s:textarea>标签的用法：

```
<s:textarea label="描述" name="description"/>
```

name 属性用来指定单行/多行文本框的名称，在 Action 中，通过该属性获取单行/多行文本框的值。其 value 属性用来指定单行/多行文本框的当前值。

此外，<textarea>标签可以通过使用 cols 和 rows 属性分别指定多行文本框的列数和行数。

2.1.5.5 <s:password>标签

<s:password>标签用于创建一个密码输入框，它可以生成 HTML 中的<input type="password"/>标签，常用于在登录表单中输入用户的登录密码。<s:password>标签的常用属性说明如表所示。

<s:password>标签的常用属性说明

属性名	说明
Name	用于指定密码输入框的名称
Size	用于指定密码输入框的显示宽度，以字符数为单位
Maxlength	用于限定密码输入框的最大输入字符串个数
showPassword	是否显示初始值，即使显示也仍为密文显示，用掩码代替

<s:password>标签的使用方法如下所示：

```
<s:password label="password" name="password" maxlength="15"/>
```

需要注意的是 Struts2 的 password 标签与 HTML 的<input type="password"/>标签有小小的不同，<input type="password"/>标签只要设置 value 属性就可以将 value 属性的值作为默认显示值；而 Struts2 的<s:password>标签除了要设置 value 属性，还要设置 showPassword 属性为 true。

2.1.5.6 <s:radio>标签

<s:radio>标签用于创建单选按钮，生成 HTML 中的<input type="radio"/>标签。<s:radio>标签的常用属性说明如表所示。

<s:radio>标签的属性及说明

属性名	是否必填	类型	说明
List	是	Collection,Map Enumeration, Iterator,array	用于生成单选框中的集合
listKey	否	String	指定集合对象中的哪个属性作为选项的 value
listValue	否	String	指定集合对象中的哪个属性作为选项的内容

表中的三个属性必须要配合使用，由 list 属性指定从集合中获得元素，由 listKey 属性指定获得元素之后使用元素的哪个属性作为生成<input type="radio"/>的 value 属性，由 listValue 属性指定生成的<input type="radio"/>后给用户看的文字。

接下来，通过一个简单的用户注册案例，来演示<s:form>标签、<s:textfield>标签、<s:textarea>标签和<s:radio>标签的使用。

在 chapter04 项目中，创建 register.jsp 页面，如文件 register.jsp 所示。

register.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib prefix="s" uri="/struts-tags"%>
4 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5         "http://www.w3.org/TR/html4/loose.dtd">
6 <html>
7 <head>
```



```
8 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9 <title>注册</title>
10 </head>
11 <body>
12 <s:form action="login" >
13     <s:textfield label="用户名" name="username"/>
14     <s:password label="密码" name="password1"/>
15     <s:password label="确认密码" name="password2"/>
16 <s:radio name="sex" label="性别" list="#{'0':'男','1':'女'}" value="0"/>
17     <s:textarea label="个性签名" name="description" rows="5" cols="15"/>
18     <s:submit value="提交"/>
19 </s:form>
20 </body>
21 </html>
```

在文件中，分别使用了<s:form>、<s:textfield>、<s:textarea>、<s:password>、<s:submit>和<s:radio>六种标签。其中<s:radio>标签中，使用list元素定义了一个Map集合，并使用value元素指定其默认显示值为“男”，其0值代表集合中key为0的元素。在浏览器地址栏输入“<http://localhost:8080/chapter04/register.jsp>”，成功访问后，浏览器的显示结果如图所示。

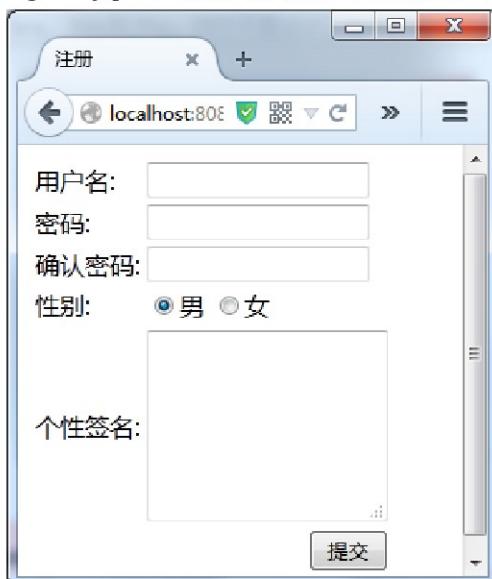


图4-2 注册页

2.1.5.7 <s:checkboxlist>标签

<s:checkboxlist>标签用于一次性创建多个复选框，用户可以选择创建零到多个，它用来产生一组<input type="checkbox"/>标签，<s:checkboxlist>标签的常用属性说明如表所示。

<s:checkboxlist>标签的常用属性及说明

属性名	是否必填	类型	说明
Name	否	String	指定该元素的name
List	是	Collection,Map	用于生成多选框的集合

		Enmumeration, Iterator,array	
listKey	否	String	生成 checkbox 的 value 属性。
listValue	否	String	生成 checkbox 后面显示的文字

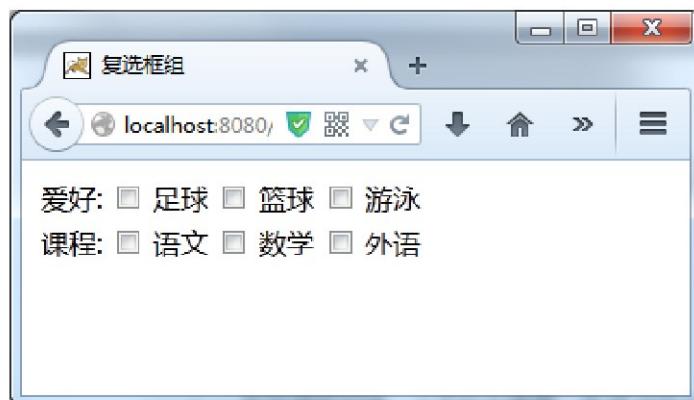
需要注意的是，listKey 和 listValue 属性主要用在集合中，其中存放的是 javabean，可以使用这两个属性从 javabean 众多属性中筛选需要的值。

接下来，在 chapter04 项目中创建一个名称为 checkboxlistTags.jsp 的页面，演示<s:checkboxlist>标签的使用，如文件所示。

checkboxlistTags.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib prefix="s" uri="/struts-tags"%>
4 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5         "http://www.w3.org/TR/html4/loose.dtd">
6 <html>
7 <head>
8 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9 <title>注册</title>
10 </head>
11 <body>
12     <s:form>
13         <s:checkboxlist label="爱好" name="interesters"
14             list="{'足球','篮球','游泳'}" labelposition="left"/>
15         <s:checkboxlist label="课程" name="education"
16             list="#{'a':'语文','b':'数学','c':'外语'}"
17             labelposition="left" listKey="key" listValue="value"/>
18     </s:form>
19 </body>
20 </html>
```

在文件中，创建了两个<s:checkboxlist>标签，标签中，list 表示要显示的集合元素，通过 labelposition 属性将 label 属性的文字内容显示在标签左侧。在浏览器地址栏输入“<http://localhost:8080/chapter04/checkboxlistTags.jsp>”，成功访问后，浏览器的运行效果如图所示。



<s:checkboxlist>标签的使用



2.1.5.8 <s:select>标签

<s:select>标签用于创建一个下拉列表框，生成 HTML 中的<select>标签。<s:select>标签的常用属性说明所示。

select 标签的常用属性及说明

属性名	是否必填	类型	说明
list	是	Cellection,Map Enmumeration, Iterator,array	用于生成下拉框的集合
listKey	否	String	生成选项的 value 属性
listValue	否	String	生成选项的显示文字
headerKey	否	String	在所有的选项前再加额外的一个选项作为其标题的 value 值
headerValue	否	String	显示在页面中 header 选项的内容
Multiple	否	Boolean	指定是否多选，默认为 false
emptyOption	否	Boolean	是否在标题和真实的选项之间加一个空选项
Size	否	Int	下拉框的高度，即最多可以同时显示多少个选项

在表中，headerKey 和 headerValue 属性需要同时使用，可以在所有的真实选项之前加一项作为标题项。比如选择省份的时候，可以在所有具体省份之前加一项“请选择”，这个项不作为备选的值。

multiple 属性和 size 属性类似于 HTML 的<select>标签，size 属性可以让下拉框同时显示多个值，multiple 属性让用户同时选择多个值，只是在后台的 Action 接收下拉框值的时候，不能使用 String 类型，而应该使用 String[] 或者 List<String>。

2.1.5.9 <s:hidden>标签

<s:hidden>标签用于创建隐藏表单元素，生成 HTML 中的隐藏域标签<input type="hidden">。该标签在页面上没有任何显示，可以保存或交换数据。其使用也比较简单，通常只设置其 name 和 value 属性即可。其一般用法如下所示：

```
<s:hidden name="id" value="%{id}"/>
```

该标签主要用来需要提交的表单传值时使用，比如需要提交表单时，要传一个值到请求参数中去，就可以使用该标签。

2.1.5.10 <s:reset>标签

<s:reset>标签用来创建一个重置按钮，会生成 HTML 中的<input type="reset"/>标签，该标签的使用比较简单，其常用属性为 name 和 value。其中，name 属性用于指定重置按钮的名称，在 Action 中，可以通过 name 属性来获取重置按钮的值，value 属性用于显示按钮的值。该标签的用法如下所示：

```
<s:reset value="Reset"/>
```



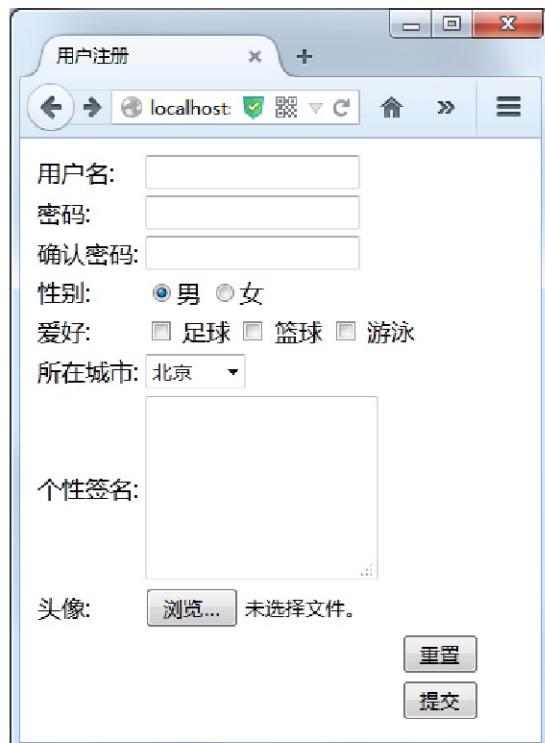
```
<s:reset name="reset" value="重置"/>
```

接下来，通过一个页面注册的案例来演示 Struts2 中表单标签的使用。在 chapter04 项目的 WebContent 目录下创建一个名称为 userRegister.jsp 的文件，如所示。

userRegister.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3 <%@ taglib prefix="s" uri="/struts-tags"%>
4 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5          "http://www.w3.org/TR/html4/loose.dtd">
6 <html>
7 <head>
8 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9 <title>用户注册</title>
10 </head>
11 <body>
12 <s:form action="register">
13     <s:hidden name="user" value="username"/>
14     <s:textfield label="用户名" name="username" />
15     <s:password label="密码" name="password1" />
16     <s:password label="确认密码" name="password2" />
17     <s:radio name="sex" label="性别"
18             list="#{'0':'男','1':'女'}" value="0" />
19     <s:checkboxlist label="爱好" name="interesters"
20             list="{'足球','篮球','游泳'}" labelposition="left" />
21     <s:select label="所在城市" name="city"
22             list="#{'beijing':'北京','shanghai':'上海市','guangzhou':'广州'}"
23             listKey="key" listValue="value">
24     </s:select>
25     <s:textarea label="个性签名" name="description" rows="5" cols="15" />
26     <s:file name="upLoadFile" label="头像"/>
27     <s:reset value="重置"/>
28     <s:submit value="提交" />
29 </s:form>
30 </body>
31 </html>
```

启动 Tomcat 服务器，在浏览器地址栏中输入 “<http://localhost:8080/chapter04/userRegister.jsp>” ，成功访问后，浏览器的显示效果如图所示。



用户名:

密码:

确认密码:

性别: 男 女

爱好: 足球 篮球 游泳

所在城市: 北京

个性签名:

头像: 未选择文件。

表单标签的使用