第 1 章 Struts2_day01

案例：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：

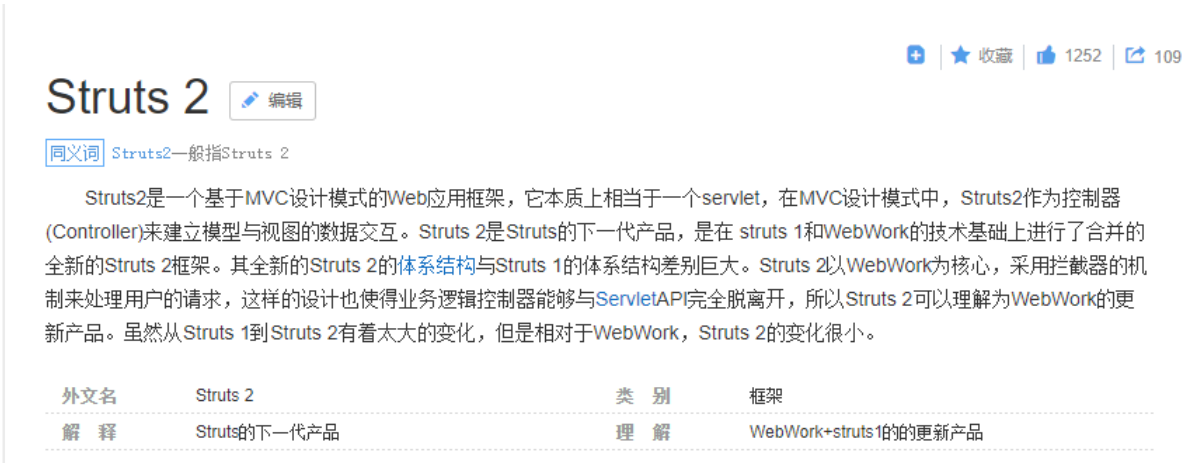
我们实际的开发中会使用Struts2作为WEB层的架构。

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



在介绍Struts2之前，先来认识一■下Struts 1。Struts 1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，Strutsl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Strutsl的下一代产品，是在Strutsl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Strutsl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?

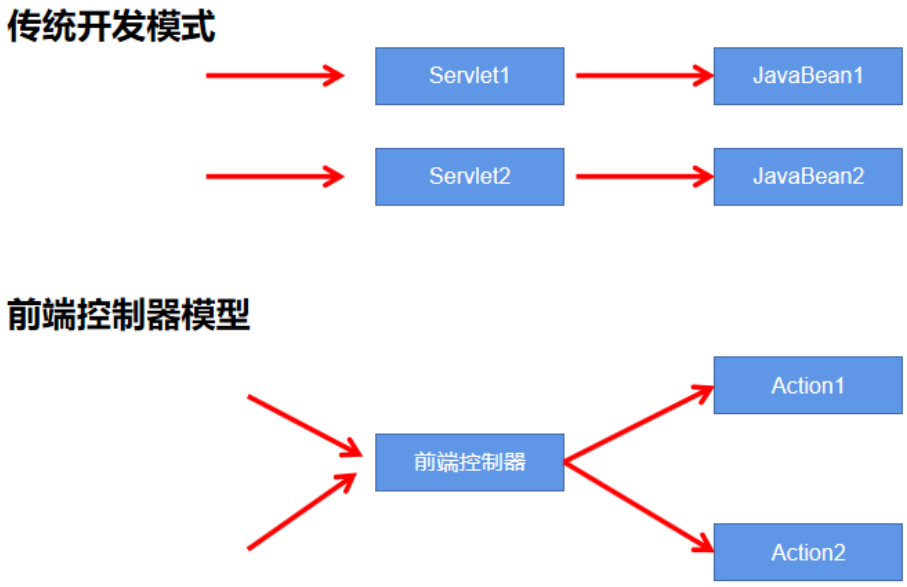
1.2.1.2 常见的WEB层的框架

- Struts2
- Strutsl
- Webwork
- SprmgMVC

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Servleto这样会导致出现很多Servleto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。



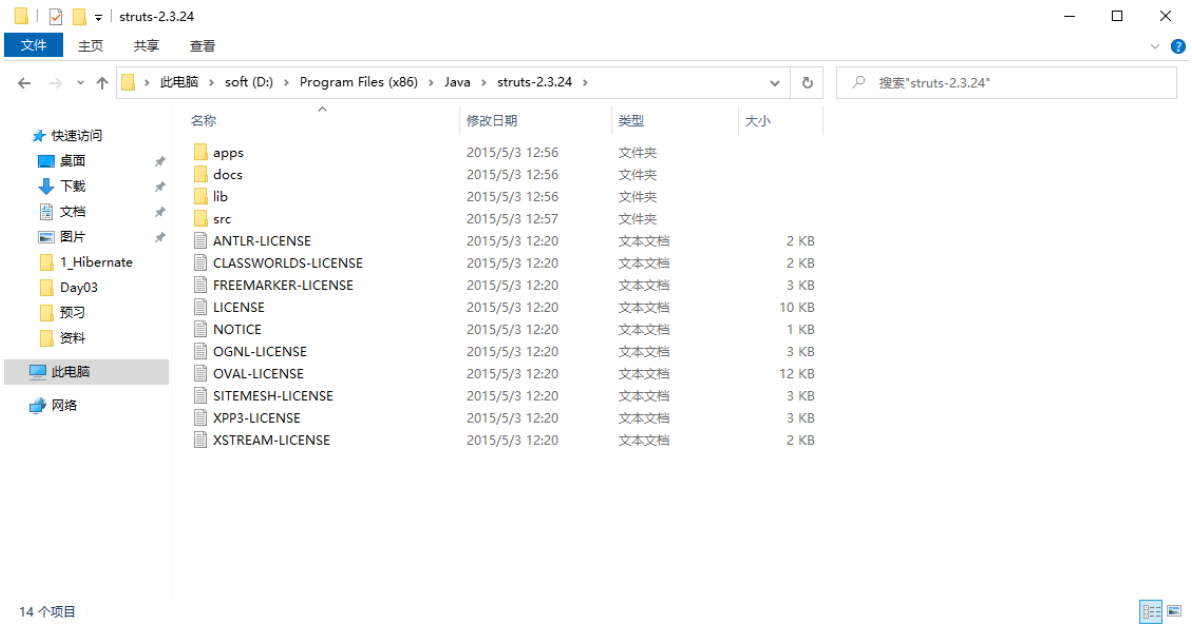
1.2.2 Struts2 快速入门

1.2.2.1下载 Struts2 的开发包

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

1.2.2.2 解压 Struts2 的开发包:

解压后的目录结构如下:



从图中可以看出，展示的是解压后的S truts2.3.24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：

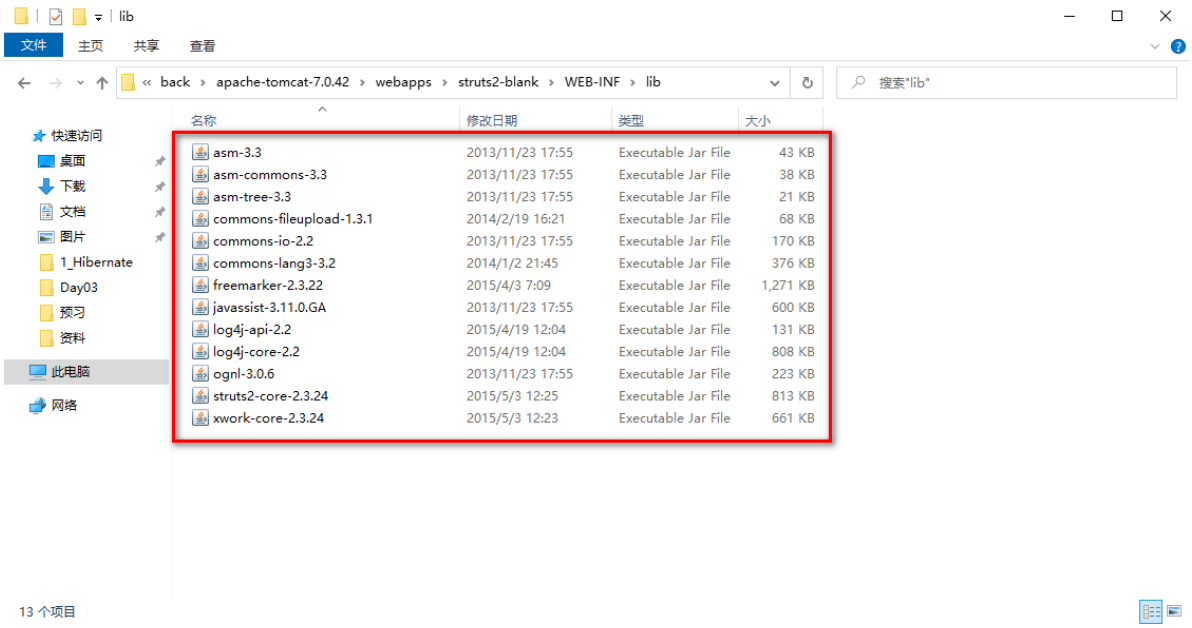
- apps：该文件夹存用于存放官方提供的Struts2示例程序，这些程序可以作为学习者的学习 资料，可为学习者提供很好的参照。各示例均为war文件，可以通过zip方式进行解压。
- docs：该文件夹用于存放官方提供的Struts2文档，包括Struts2的快速入门、Struts2的文档, 以及API文档等内容。
- lib：该文件夹用于存放Struts2的核心类库，以及Struts2的第三方插件类库。
- src：该文件夹用于存放该版本Struts2框架对应的源代码。

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hb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。



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 .j ar	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. j ar	Web Work核心库，Struts2的构建基础
ognl-3.0.6.jar	对象图导航语言(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，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demol,在demol下创建一个新的jsp。在jsp中编写 一个Action的访问路径。

```
<%--
  Created by IntelliJ IDEA.
  User: Administrator
  Date: 2020/9/25
  Time: 2:34
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Struts2 入门</title>
```

```
</head>
<body>
    <h3><a href="${ pageContext.request.contextPath }/hello.action">Struts2 入门
</a></h3>
</body>
</html>
```

点击该链接，需要提交请求到服务器的Actiono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StrutsDemol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

```
package com.admiral.demo1;

/**
 * Struts 入门的 Action
 */
public class HelloAction {

    /**
     * 提供一个方法
     * 方法名是固定的
     *      公有的,返回值是 String 方法名 execute 在这个方法中不能传递参数
     * @return
     */
    public String execute(){
        System.out.println("HelloAction 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。

```
<?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>
    <!--  Struts 为了管理 Action 的配置,通过包进行管理  -->
    <!--  配置 Struts2 的包  -->
    <package name="demo1" extends="struts-default" namespace="/">
        <!-- 配置 Action -->
        <action name="hello" class="com.admiral.demo1.HelloAction">

        </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。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <filter>
        <filter-name>struts</filter-name>
        <filter-
class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</fil
ter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

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。

```
package com.admiral.demo1;

/**
 * Struts 入门的 Action
 */
public class HelloAction {

    /**
     * 提供一个方法
     * 方法名是固定的
     *      公有的,返回值是 String 方法名 execute 在这个方法中不能传递参数
     * @return
     */
    public String execute(){
        System.out.println("HelloAction 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，对标签进行完善，在标签中配置一个标签，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successo标签内部就是跳转的 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>
    <!--  Struts 为了管理 Action 的配置,通过包进行管理  -->
    <!--  配置 Struts2 的包  -->
    <package name="demo1" extends="struts-default" namespace="/">
        <!-- 配置 Action -->
        <action name="hello" class="com.admiral.demo1.HelloAction">
            <result name="success">/demo1/success.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：

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

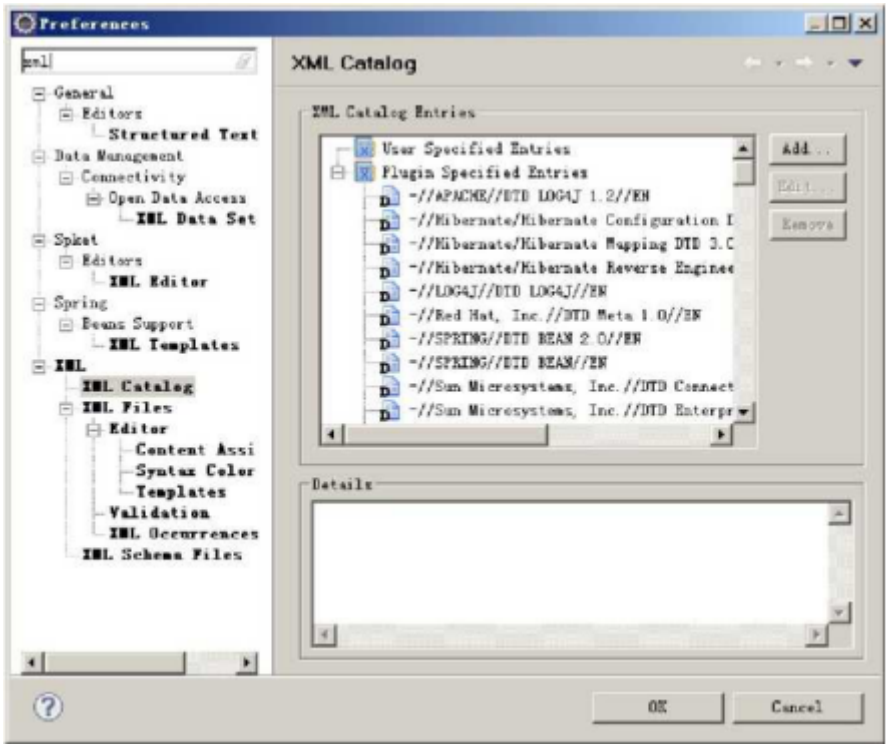
1.2.3.1 Struts2 的执行流程

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.

1.2.3.2 配置 struts.xml中 的提示（在不联网情况下）

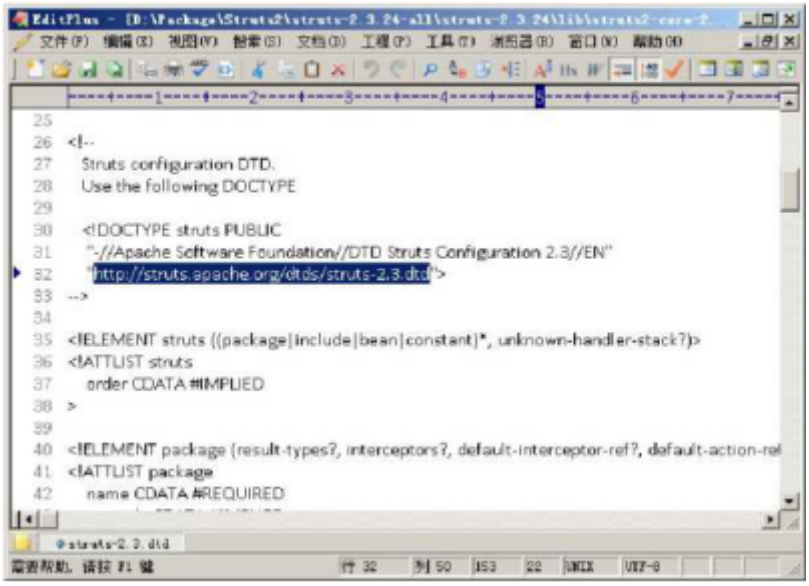
开发过程中如果可以上网，struts.xml会自动缓存dtd,提供提示功能。如果不能够上网，则 需要我们手动配制本地dtd,这样才能够使struts.xml产生提示。具体配置方法如下：

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。



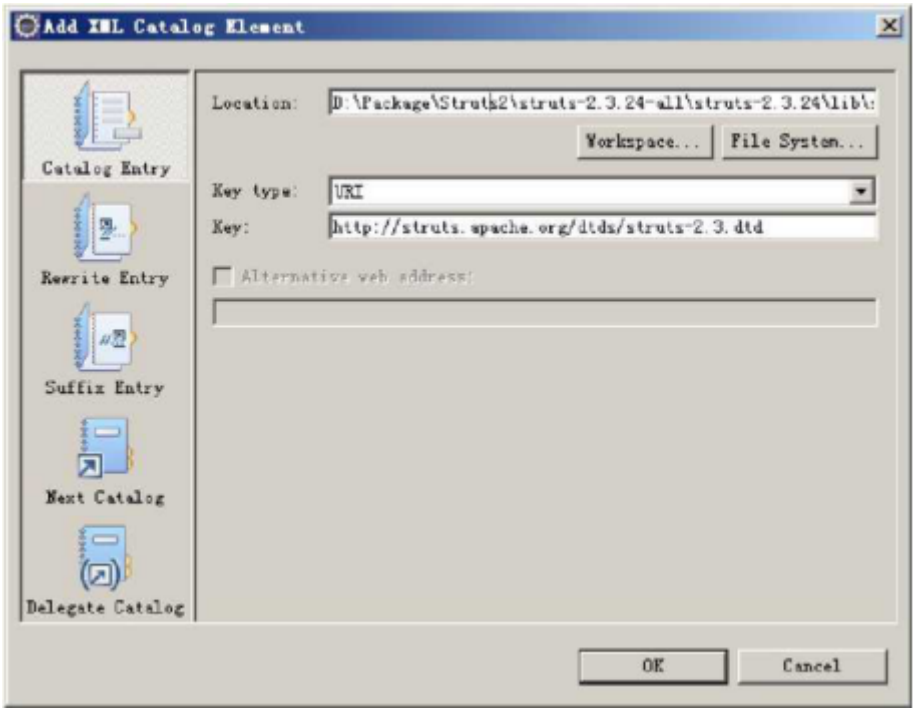
XML Catalog 窗口

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.dtdo
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



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 URL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。

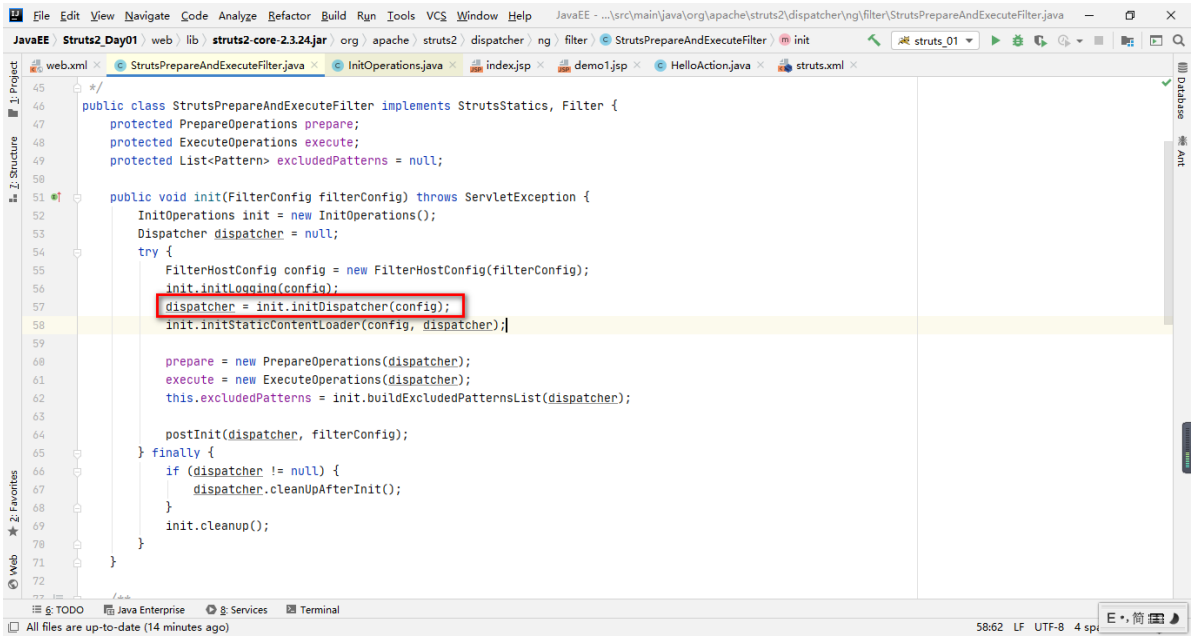
那么提示已经会配置了，也就可以进行Struts2的开发了，开发过程中需要自己配置struts.xml 文件，在这个文件中就有很多常见的配置。接下来我们就来学习Struts2的常见配置。

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。

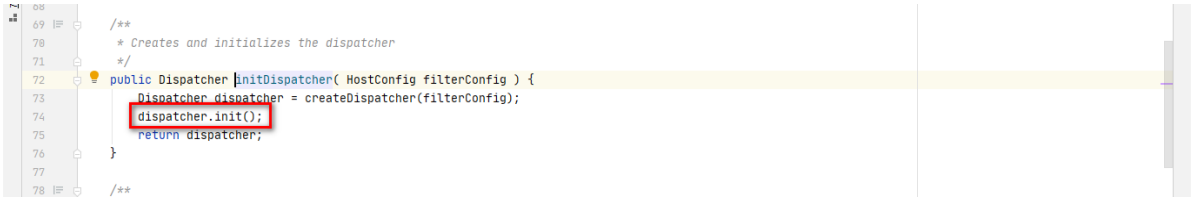
1.2.4.1 Struts2 的配置文件的加载顺序

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mit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。



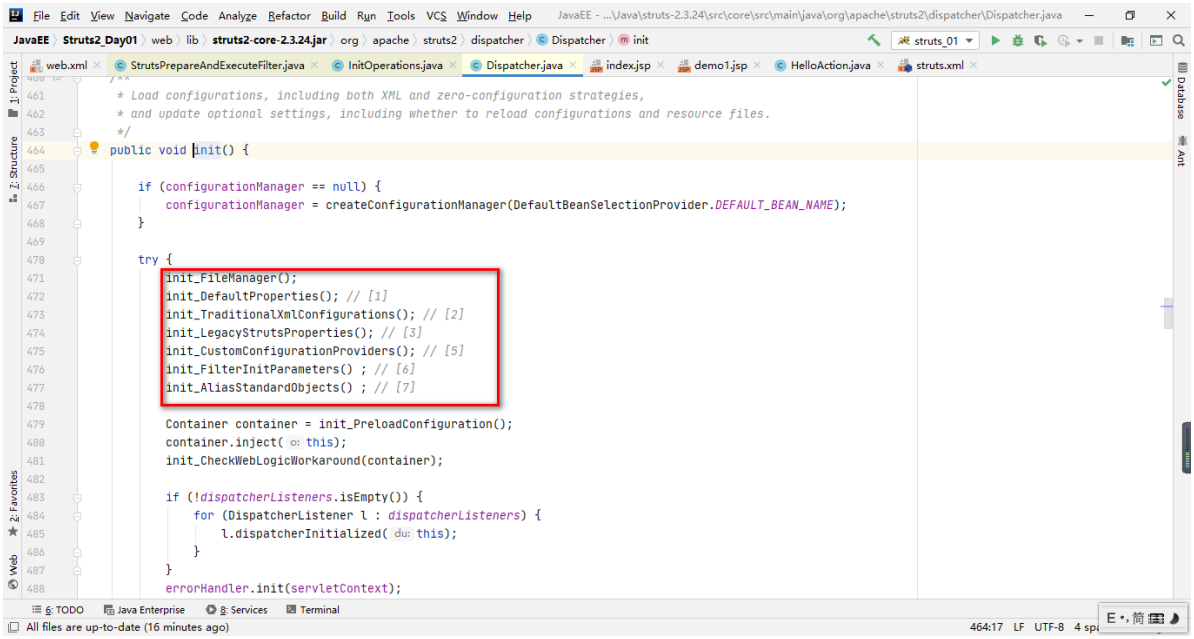
```java
*/
public class StrutsPrepareAndExecuteFilter implements StrutsStatics, Filter {
    protected PrepareOperations prepare;
    protected ExecuteOperations execute;
    protected List<Pattern> excludedPatterns = null;

    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mitDispatcher的方法来加载配置文件，进入到该代码中



```java
    /**
     * 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：



```java
     * Load configurations, including both XML and zero-configuration strategies,
     * and update optional settings, including whether to reload configurations and resource files.
     */
    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( o: this);
            init_CheckWebLogicWorkaround(container);

            if (!dispatcherListeners.isEmpty()) {
                for (DispatcherListener l : dispatcherListeners) {
                    l.dispatcherInitialized( du: this);
                }
            }
            errorHandler.init(servletContext);
```

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

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keyo
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。

- 默认命名空间 :默认的名称空间就是namespace=""
- 根命名空间 :跟名称空间就是namespace=,7"
- 带名称的命名空间 :带名称的名称空间就是namespace="/demo 1"

[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，通过元素对请求的Action和Action类进行配置。

元素中共有4个属性，这4个属性的说明如表所示。

属性	说明
name	必填属性，标识Action,指定了 Action所处理的请求 的 URLo
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，Method指定了执行Action的那个方法	生共同决定了访问路径，class对应的是Action类的全路径。 ,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valueo

• 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：

```
<?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>

    <constant name="struts.action.extension" value="abc"></constant>

    <!--  Struts 为了管理 Action 的配置,通过包进行管理  -->
    <!--  配置 Struts2 的包  -->
    <package name="demo1" extends="struts-default" namespace="/">
        <!-- 配置 Action -->
        <action name="hello" class="com.admiral.demo1.HelloAction">
            <result name="success">/demo1/success.jsp</result>
        </action>
    </package>
</struts>
```

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.il8n.encoding=UTF-8

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

###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.properti.es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〉元素的子元素指定，每个元素配置了一个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：

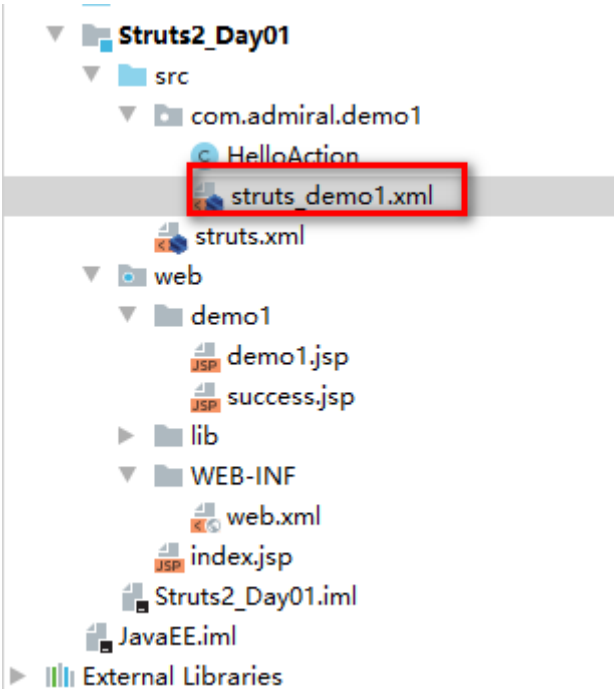
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <filter>
        <filter-name>struts</filter-name>
        <filter-
class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</fil
ter-class>
        <init-param>
            <param-name>struts.action.extension</param-name>
            <param-value>xyz</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>struts</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

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



struts_demo1.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>
```

```
    <!--  Struts 为了管理 Action 的配置,通过包进行管理  -->
    <!--  配置 Struts2 的包  -->
    <package name="demo1" extends="struts-default" namespace="/">
        <!-- 配置 Action -->
        <action name="hello" class="com.admiral.demo1.HelloAction">
            <result name="success">/demo1/success.jsp</result>
        </action>
    </package>
</struts>
```

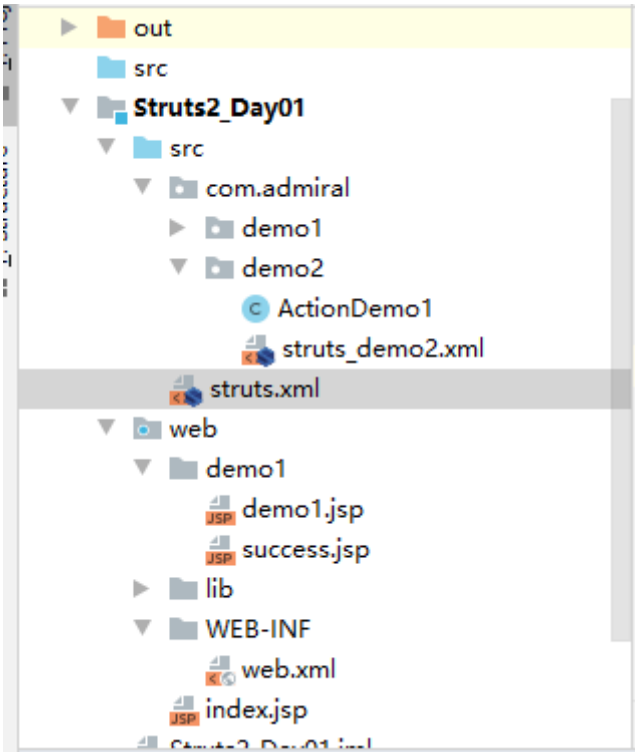
struts.xml

```
<?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>

    <constant name="struts.action.extension" value="action"></constant>

    <include file="com/admiral/demo1/struts_demo1.xml"></include>
</struts>
```

1.2.5 Struts2 的 Action 的访问

1.2.5.1 Action 的编写的方式

1.2.5.1.1Action类是POJO的类




```java
package com.admiral.demo2;

/**
 * Action的编写方式一:Action是一个普通的POJO类
 */
public class ActionDemo1 {

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

struts_demo2.xml

```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="demo2" extends="struts-default" namespace="/">
        <action name="actionDemo1" class="com.admiral.demo2.ActionDemo1">
</action>
    </package>
</struts>
```

struts.xml

```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>

    <constant name="struts.action.extension" value="action"></constant>

    <include file="com/admiral/demo1/struts_demo1.xml"></include>
    <include file="com/admiral/demo2/struts_demo2.xml"></include>
</struts>
```

1.2.5.1.2 Action类实现一个Action的接口

```java
package com.admiral.demo2;

import com.opensymphony.xwork2.Action;

/**
 * Action 的编写方式二:实现 Action接口
 * 实现接口的这种方式,提供给了五个常量(五个逻辑视图的名称)
 *  SUCCESS     :成功
 *  ERROR       :失败
 *  LOGIN       :登陆出错跳转页面
 *  INPUT       :表单校验的时候出错
 *  NONE        :不跳转
 */
public class ActionDemo2 implements Action {

    @Override
    public String execute() throws Exception {
        System.out.println("ActionDemo2 执行了....");
        return NONE;
    }
}
```

1.2.5.1.3 Action类继承ActionSupport类

```java
package com.admiral.demo2;

import com.opensymphony.xwork2.ActionSupport;

/**
 * Action 的编写方式三:继承 ActionSupport 类
 * 推荐使用这种方式:
 *      ActionSupport 类中提供了数据校验 国际化...一系列的操作方法.
 */
public class ActionDemo3 extends ActionSupport {
```

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

struts_demo2.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="demo2" extends="struts-default" namespace="/">
        <action name="actionDemo1" class="com.admiral.demo2.ActionDemo1">
</action>
        <action name="actionDemo2" class="com.admiral.demo2.ActionDemo2">
</action>
        <action name="actionDemo3" class="com.admiral.demo2.ActionDemo3">
</action>
    </package>
</struts>
```

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>

    <constant name="struts.action.extension" value="action"></constant>

    <include file="com/admiral/demo1/struts_demo1.xml"></include>
    <include file="com/admiral/demo2/struts_demo2.xml"></include>
</struts>
```

1.2.5.2 Action 的访问

1.2.5.2.1 通过method设置

在 web 目录下新建 demo2/demo1.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: Administrator
  Date: 2020/9/25
  Time: 4:48
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h1>Action的访问</h1>
    <h3>通过 method 方式</h3>
    <a href="${ pageContext.request.contextPath}/userFind.action">查找用户</a>
<br/>
    <a href="${ pageContext.request.contextPath}/userUpdate.action">修改用户</a>
<br/>
    <a href="${ pageContext.request.contextPath}/userDelete.action">删除用户</a>
<br/>
    <a href="${ pageContext.request.contextPath}/userSave.action">保存用户</a>
<br/>
</body>
</html>
```

在 src 下新建包 demo2 新建 UserAction

```
package com.admiral.demo3;

import com.opensymphony.xwork2.ActionSupport;

public class UserAction extends ActionSupport {

    public String find(){
        System.out.println("查找用户...");
        return NONE;
    }
    public String update(){
        System.out.println("更新用户...");
        return NONE;
    }
    public String delete(){
        System.out.println("删除用户...");
        return NONE;
    }
    public String save(){
        System.out.println("保存用户...");
        return NONE;
    }
}
```

新建 struts_demo3.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="demo3" extends="struts-default" namespace="/">
        <action name="userFind" class="com.admiral.demo3.UserAction"
method="find"></action>
        <action name="userUpdate" class="com.admiral.demo3.UserAction"
method="update"></action>
        <action name="userDelete" class="com.admiral.demo3.UserAction"
method="delete"></action>
        <action name="userSave" class="com.admiral.demo3.UserAction"
method="save"></action>
    </package>
</struts>
```

在 struts.xml 中引入

```
<?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>

    <constant name="struts.action.extension" value="action"></constant>

    <include file="com/admiral/demo1/struts_demo1.xml"></include>
    <include file="com/admiral/demo2/struts_demo2.xml"></include>
    <include file="com/admiral/demo3/struts_demo3.xml"></include>
</struts>
```

1.2.5.2.2 通过通配符的方式进行配置

更改 /web/demo2/demo1.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: Administrator
  Date: 2020/9/25
  Time: 4:48
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
```

```
<body>
    <h1>Action的访问</h1>
    <h3>通过 method 方式</h3>
    <a href="${ pageContext.request.contextPath}/userFind.action">查找用户</a>
<br/>
    <a href="${ pageContext.request.contextPath}/userUpdate.action">修改用户</a>
<br/>
    <a href="${ pageContext.request.contextPath}/userDelete.action">删除用户</a>
<br/>
    <a href="${ pageContext.request.contextPath}/userSave.action">保存用户</a>
<br/>

    <h3>通过通配符方式</h3>
    <a href="${ pageContext.request.contextPath}/product_find.action">查找商品</a>
<br/>
    <a href="${ pageContext.request.contextPath}/product_update.action">修改商品
</a><br/>
    <a href="${ pageContext.request.contextPath}/product_delete.action">删除商品
</a><br/>
    <a href="${ pageContext.request.contextPath}/product_save.action">保存商品</a>
<br/>



</body>
</html>
```

新建 ProductAction

```
package com.admiral.demo3;

import com.opensymphony.xwork2.ActionSupport;

public class ProductAction extends ActionSupport {

    public String find(){
        System.out.println("查找商品...");
        return NONE;
    }
    public String update(){
        System.out.println("更新商品...");
        return NONE;
    }
    public String delete(){
        System.out.println("删除商品...");
        return NONE;
    }
    public String save(){
        System.out.println("保存商品...");
        return NONE;
    }
}
```

修改 struts_demo3.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="demo3" extends="struts-default" namespace="/">
        <action name="userFind" class="com.admiral.demo3.UserAction"
method="find"></action>
        <action name="userUpdate" class="com.admiral.demo3.UserAction"
method="update"></action>
        <action name="userDelete" class="com.admiral.demo3.UserAction"
method="delete"></action>
        <action name="userSave" class="com.admiral.demo3.UserAction"
method="save"></action>

        <!-- 通过通配符的方式配置Action的访问 -->
        <action name="product_*" class="com.admiral.demo3.ProductAction"
method="{1}"></action>
    </package>
</struts>
```

1.2.5.2.3 动态方法访问

修改 /web/demo2/demo1.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: Administrator
  Date: 2020/9/25
  Time: 4:48
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h1>Action的访问</h1>
    <h3>通过 method 方式</h3>
    <a href="${ pageContext.request.contextPath}/userFind.action">查找用户</a>
<br/>
    <a href="${ pageContext.request.contextPath}/userUpdate.action">修改用户</a>
<br/>
    <a href="${ pageContext.request.contextPath}/userDelete.action">删除用户</a>
<br/>
    <a href="${ pageContext.request.contextPath}/userSave.action">保存用户</a>
<br/>
```

```
    <h3>通过通配符方式</h3>
    <a href="${ pageContext.request.contextPath}/product_find.action">查找商品</a>
<br/>
    <a href="${ pageContext.request.contextPath}/product_update.action">修改商品
</a><br/>
    <a href="${ pageContext.request.contextPath}/product_delete.action">删除商品
</a><br/>
    <a href="${ pageContext.request.contextPath}/product_save.action">保存商品</a>
<br/>

    <h3>通过动态方法访问方式</h3>
    <a href="${ pageContext.request.contextPath}/customer!find.action">查找客户
</a><br/>
    <a href="${ pageContext.request.contextPath}/customer!update.action">修改客户
</a><br/>
    <a href="${ pageContext.request.contextPath}/customer!delete.action">删除客户
</a><br/>
    <a href="${ pageContext.request.contextPath}/customer!save.action">保存客户
</a><br/>



</body>
</html>
```

创建 CustomerAction

```
package com.admiral.demo3;

import com.opensymphony.xwork2.ActionSupport;

public class CustomerAction extends ActionSupport {

    public String find(){
        System.out.println("查找客户...");
        return NONE;
    }
    public String update(){
        System.out.println("更新客户...");
        return NONE;
    }
    public String delete(){
        System.out.println("删除客户...");
        return NONE;
    }
    public String save(){
        System.out.println("保存客户...");
        return NONE;
    }
}
```

开启动态方法访问

```
    <!-- 开启动态方法访问 -->
    <constant name="struts.enable.DynamicMethodInvocation" value="true">
</constant>
```

配置 struts_demo3.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>
    <!-- 开启动态方法访问 -->
    <constant name="struts.enable.DynamicMethodInvocation" value="true">
</constant>

    <package name="demo3" extends="struts-default" namespace="/">
        <action name="userFind" class="com.admiral.demo3.UserAction"
method="find"></action>
        <action name="userUpdate" class="com.admiral.demo3.UserAction"
method="update"></action>
        <action name="userDelete" class="com.admiral.demo3.UserAction"
method="delete"></action>
        <action name="userSave" class="com.admiral.demo3.UserAction"
method="save"></action>

        <!-- 通过通配符的方式配置Action的访问 -->
        <action name="product_*" class="com.admiral.demo3.ProductAction"
method="{1}"></action>

        <!-- 通过动态方法访问的方式配置 Action 的访问 -->
        <action name="customer" class="com.admiral.demo3.CustomerAction">
</action>
    </package>
</struts>
```

1.3.1 搭建开发环境

1.3.1.0 创建数据库和表

```sql
CREATE TABLE `cst_customer` (
  `cust_id` bigint(32) NOT NULL AUTO_INCREMENT COMMENT '客户编号(主键)',
  `cust_name` varchar(32) NOT NULL COMMENT '客户名称(公司名称)',
  `cust_source` varchar(32) DEFAULT NULL COMMENT '客户信息来源',
  `cust_industry` varchar(32) DEFAULT NULL COMMENT '客户所属行业',
  `cust_level` varchar(32) DEFAULT NULL COMMENT '客户级别',
  `cust_phone` varchar(64) DEFAULT NULL COMMENT '固定电话',
  `cust_mobile` varchar(16) DEFAULT NULL COMMENT '移动电话',
  PRIMARY KEY (`cust_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

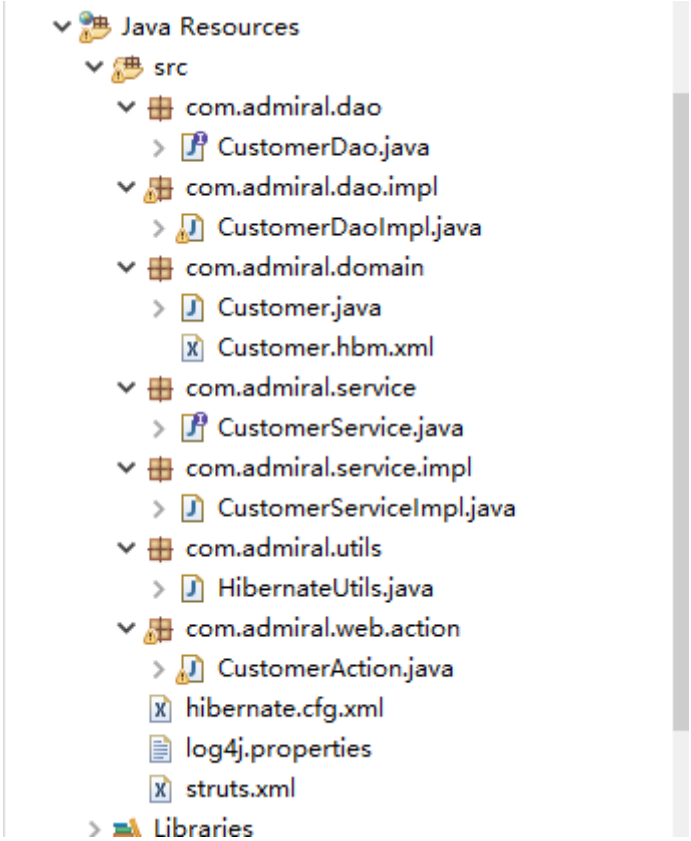
1.3.1.1 步骤一：创建 WEB 工程，引入 jar 包

- Struts2

- Hibernate

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

```
<?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" id="WebApp_ID" version="2.5">
  <display-name>Struts2_crm</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <filter>
    <filter-name>struts</filter-name>
    <filter-
class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</fil
ter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>
```

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

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

  </TBODY>
TABLE>
```

1.3.2.2 步骤七：编写Action中的代码

```
/**
* @Title: CustomerAction.java
* @Package com.admiral.web.action
* @Description:
* @author 白世鑫
* @date 2020-9-25
* @version V1.0
*/
package com.admiral.web.action;

import java.util.List;

import org.apache.struts2.ServletActionContext;

import com.admiral.domain.Customer;
import com.admiral.service.CustomerService;
import com.admiral.service.impl.CustomerServiceImpl;
import com.opensymphony.xwork2.ActionSupport;

public class CustomerAction extends ActionSupport {

    public String find() {
        CustomerService customerService = new CustomerServiceImpl();
        List<Customer> list = customerService.find();

        ServletActionContext.getRequest().setAttribute("list", list);

        return "findSuccess";
    }
}
```

1.3.2.3 步骤八：编写业务层的类

```
/**
* @Title: CustomerDaoImpl.java
* @Package com.admiral.dao.impl
* @Description:
```

```
* @author 白世鑫
* @date 2020-9-25
* @version V1.0
*/
package com.admiral.dao.impl;

import java.util.List;

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

import com.admiral.dao.CustomerDao;
import com.admiral.domain.Customer;
import com.admiral.utils.HibernateUtils;

public class CustomerDaoImpl implements CustomerDao{

    @Override
    public List<Customer> find() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

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

        transaction.commit();
        return customers;
    }

}
```

1.3.2.4 步骤九：引入Hibernate的jar包和配置文件

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>
    <!--
        建立类和表的映射关系
            name属性:     类中的全路径
            table属性:    表名(如果类名和表名是一致的,那么表名可以省略)
            catalog属性:  数据库名,可以省略
    -->
    <class name="com.admiral.domain.Customer" table="cst_customer">
        <!--建立类中的属性与表中的主键的映射关系 -->
        <id name="cust_id" column="cust_id">
            <!--主键的生成策略 -->
            <generator class="native"></generator>
        </id>
```

```
        <!-- 建立类中的普通属性与表中的字段的映射 -->
        <property name="cust_name" column="cust_name" not-null="false"
unique="false"/>
        <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

```
```

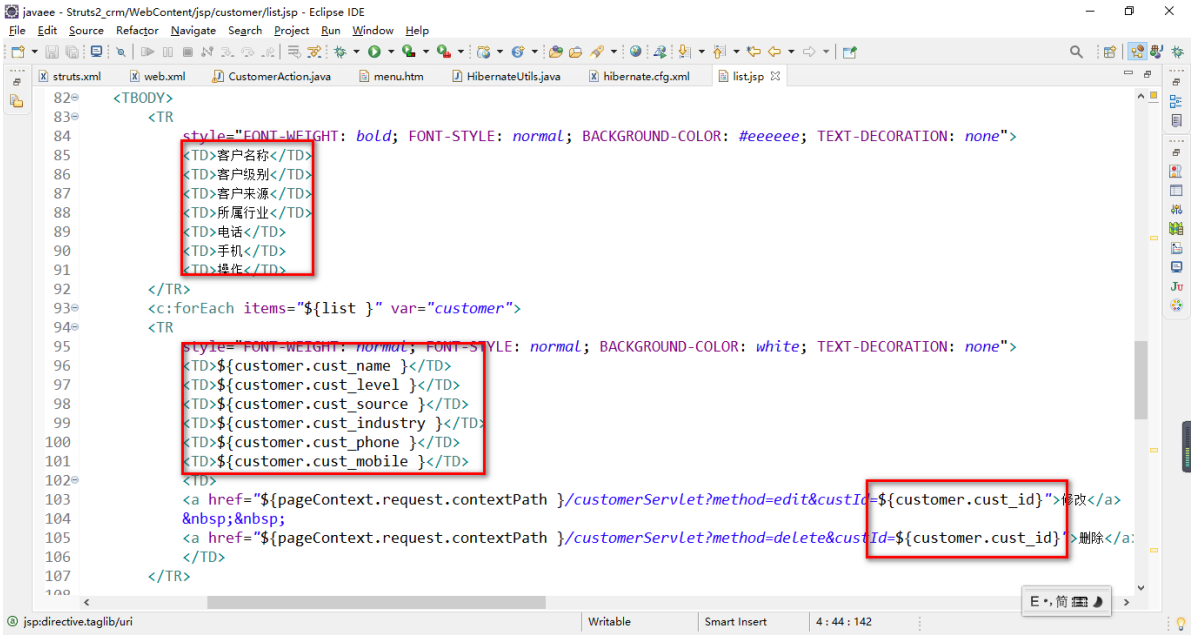
1.3.2.7 步骤十二：编写DAO

```
/**
* @Title: CustomerDaoImpl.java
* @Package com.admiral.dao.impl
* @Description:
* @author 白世鑫
* @date 2020-9-25
* @version V1.0
*/
package com.admiral.dao.impl;

import java.util.List;

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

import com.admiral.dao.CustomerDao;
import com.admiral.domain.Customer;
import com.admiral.utils.HibernateUtils;

public class CustomerDaoImpl implements CustomerDao{

    @Override
    public List<Customer> find() {
        Session session = HibernateUtils.getCurrentSession();
        Transaction transaction = session.beginTransaction();

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

        transaction.commit();
        return customers;
    }

}
```

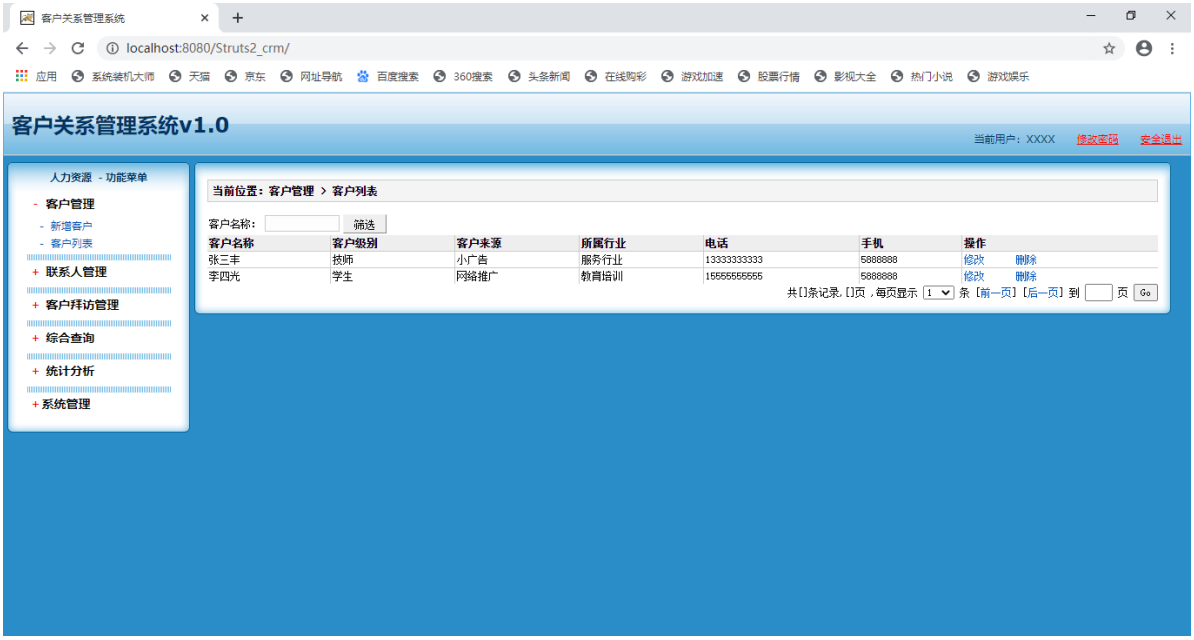
1.3.2.8 步骤十三：配置Action的有页面跳转

```
<?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="com.admiral.web.action.CustomerAction"
method="{1}">
            <result name="findSuccess">/jsp/customer/list.jsp</result>
        </action>
    </package>

</struts>
```

13.2.9 步骤十四：在页面中显示相应的数据



1.3.2.10 步骤十五：测试程序


客户关系管理系统
localhost:8080/Struts2_crm/
应用
系统装机大师
天猫
京东
网址导航
百度搜索
360搜索
头条新闻
在线购彩
游戏加速
股票行情
影视大全
热门小说
游戏娱乐
客户关系管理系统v1.0
当前用户：XXXX
修改密码
安全退出
人力资源 - 功能菜单
- 客户管理
- 新增客户
- 客户列表
+ 联系人管理
+ 客户拜访管理
+ 综合查询
+ 统计分析
+ 系统管理
当前位置：客户管理 > 客户列表
客户名称：
筛选
客户名称 客户级别 客户来源 所属行业 电话 手机 操作
张三丰 技师 小广告 服务行业 13333333333 5888888 修改 删除
李四光 学生 网络推广 教育培训 15555555555 5888888 修改 删除
共[]条记录,[]页，每页显示 1 条 [前一页] [后一页] 到 页 Go