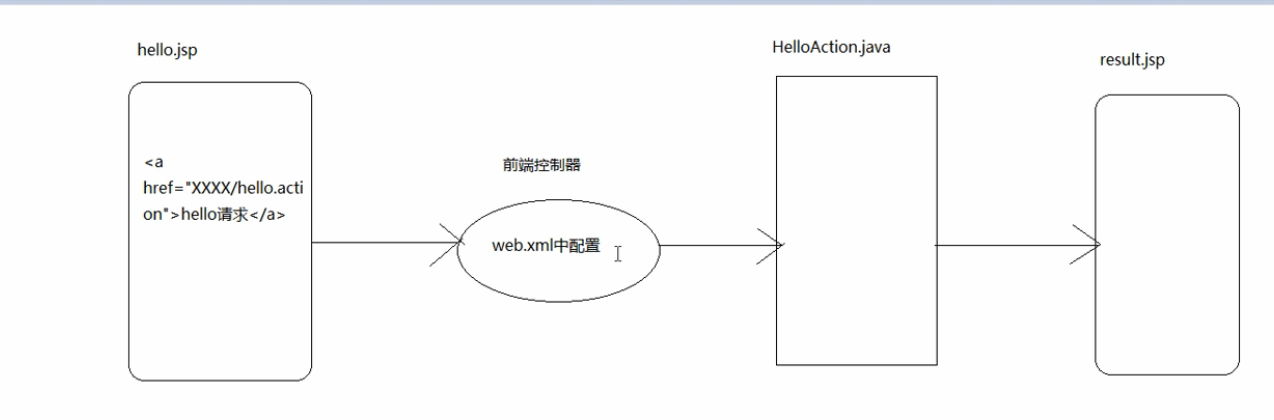
**Struts2**

Struts2的简单操作过程：



1. 首先在hello.jsp页面中编写好页面。
2. 将hello.jsp页面交给前段控制器。
3. 前段控制器中经过配置找到HelloAction.java。
4. 然后通过HelloAction中的execute()方法的return值找到前段控制器中的相应的跳转页面。
5. 跳转到result.jsp页面。

案例分析：



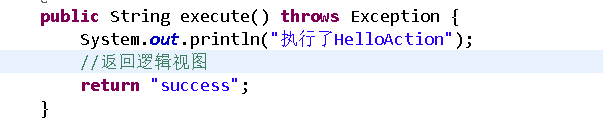
匹配package中的namespace=”x” 匹配action节点中的hello

匹配到要执行的action类



执行HelloAction中的execute方法



匹配success对应的结果页面





以上就是简单的struts2案例分析：

1. 首先由hello.jsp中的超链接中的href中的/匹配package中的namespace中的“/”.再由/后的hello名称匹配到action中的hello
2. 再由action节点中的name属性值hello找到前端控制器中的要执行的action类
3. 执行action类中的execute方法
4. 根据返回值匹配result节点中的name属性值后所对应的页面。
5. 跳转到该页面。

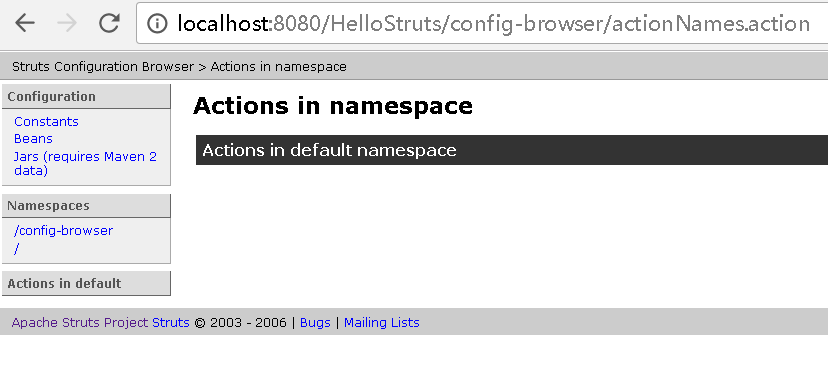
导入插件包：用于查看struts.xml文件中的配置。项目只给你action及其与jsp页面之间的映射。

直接将该插件复制到工程的lib中即可。



此包在struts下载文件的的lib目录下。

导入插件包，重新启动项目，访问：localhost:8080/HelloStruts/config-browser/index.action通过路径，查看struts2 Action 的配置情况：



**Struts.xml配置文件：**

**1.package节点**

在<package name = “default” extends = “struts-default” namespace = “/”></package>中

Name表示包名。

Extends：是继承。

Namespace：是指访问路径。

**Struts.xml配置文件：**

<?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=*"default"* extends = *"struts-default"* namespace = *"/xxx"*>

<action name = *"hello"* class = *"wp.struts.FirstAction"*>

<result name = *"success"*>/hello\_02/result.jsp</result><!--要跳转的页面路径-->

</action>

</package>

</struts>

在该配置文件中namespace：访问路径

【面试知识点一】

如果url是 [http://项目路径/xxx/yyy.action。那么Namespace](http://项目路径/xxx/yyy.action。Namespace) = “/xxx”

如果url是 [http://项目路径/xxx/yyy/zzz.action。那么Namespace](http://项目路径/xxx/yyy.action。Namespace) = “/xxx/zzz”

如果url是 [http://项目路径/xxx/yyy/zzz/aaa.action。那么Namespace](http://项目路径/xxx/yyy.action。Namespace) = “/xxx/yyy/zzz”

【面试知识点二】

如果url是http://localhost:8080/项目名称/xxx/aaa/bbb/cc/dd/ee/hello.action那么会

首先去收索namespace为/xxx/aaa/bbb/cc/dd/ee的package

如果找不到，去收索/xxx/aaa/bbb/cc/dd的package

如果找不到，去收索/xxx/aaa/bbb/cc的package

如果找不到，去收索/xxx/aaa/bbb的package

如果找不到，去收索/xxx/aaa的package

如果找不到，去收索/xxx的package

如果找不到回去收索/

如果再找不到会报错

**Namespace是逐级向上收索。所以即使在struts.xml文件中的namespace = “/”程序也可以正常运行。**

【面试知识点三】

Namespace = “” 和namespace = “/”的区别：

Namespace = “/” 的优先级比namespace = “”的优先级高。

1. **action节点**

<action name = “” class = “”></action>

如果不写class = “...”，那么程序将直接找到ActionSupport类中的execute()方法进行执行，然后返回”success”。所以即使不写class程序也会正确的执行。

Struts2框架在执行action时，找不到对应的Action类，会去找struts.default.xml中的配置信

息。在D:\eclipse\eclipse项目\Struts2\_chapter04\WebContent\WEB-INF\lib\struts2-core-2.3.34.jar/struts-default.xml文件中最后一句**<default-class-ref class = “com.opensymphony.xwork2.ActionSupport”/>** 由此可知，默认找的类是ActionSupport类。

当找不到action节点中对应的name后，会报404错误，可以自己设定一个错误页面，当找不到action节点中对应的name以后就跳转到指定的错误页面。范例如下：

在package节点下配置一个出错要执行的action。然后在package的首行引入

<!-- 告诉struts框架，如果action找不到，要跳转到指定的action -->

<default-action-ref name = *"errorPag"*></default-action-ref>这段代码告诉struts框架。

**范例：**

<struts>

<package name=*"default"* extends = *"struts-default"* namespace = *"/xxx/aaa"*>

<!-- 告诉struts框架，如果action找不到，要跳转到指定的action -->

**<default-action-ref name = *"errorPag"*></default-action-ref>**

<action name = *"hello1"* class = *"wp.struts.FirstAction"*> <!--此句找不到hello1 出错-->

<result name = *"success"*>/hello\_02/result.jsp</result>

</action>

<!-- 配置一个出错要执行的action -->

**<action name = *"errorPag"*>**

**<result name = *"success"*>/hello\_02/errorPage.jsp</result>**

**</action>**

</package>

</struts>

如果找不到页面程序就执行**<default-action-ref name = *"errorPag"*></default-action-ref>**

这句话，通过name = “errorPag”找到对应的action节点中name=”errorPag”的节点，然后执行默认的class类（ActionSupport类）通过返回的success执行result节点中的页面。

1. **constant常量配置,文件分离**
2. 常量配置：

在Struts中默认的后缀名是.action可以根据常量的配置，设置自己想要的后缀名称。

在主struts.xml配置文件中添加<constant name=*"struts.action.extension"* value=*"ABC"*></constant>这样一段代码，然后再对各个节点进行配置。此时就可以将后缀名称改变为.ABC了。

范例：

<!-- 配置后缀名常量 -->

<!-- 即.ABC的后缀也可以访问 -->

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

<package name = *"mypart\_1"* extends = *"struts-default"* namespace = *"/part1"*>

<action name = *"a"* >

<result name = *"success"*>/01\_hello/result.jsp</result>

</action>

</package>

当然也可以通过web.xml文件进行常量的配置，但是最好是在Struts.xml文件中进行常量的配置。

**加载配置文件是有优先级的，先加载struts.xml文件，再加载web.xml文件。**

1. 文件分离：（主要是指在主配置文件struts.xml中引入其他子文件，方便管理配置文件）

将每个package模块放在不同的Struts.xml配置文件中

**Eg:在src下新建一个struts.xml的配置文件名为struts-part1.xml,在配置文件中配置好package，action，result等节点后。在住struts.xml文件中引入子配置文件：<include file = “子配置文件的全路径”></include>**

**配置文件格式：**

<?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=*"default"* extends = *"struts-default"* namespace = *""*>

<action name = *""* class = *""*>

<result name = *"success"*>要执行的页面</result>

</action>

</package>

</struts>

在配置文件中**<!DOCTYPE struts PUBLIC**

**"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"**

**"http://struts.apache.org/dtds/struts-2.3.dtd">**

这段代码是在 **Libraries-->web应用程序-->struts2-core-2.2.34.jar-->struts-default.xml**中找到复制过来的。

**Action的三种编写方式：**

第一种：实现Action接口（可以使用结果集常量字符串）

第二种：继承ActionSupport类：（重点中的重点，推荐的方式）

1. 对请求参数进行校验。
2. 设置错误信息。

第三种：pojo类（非入侵，简单，干净）没有extends父类，也没有implements接口。

配置文件的配置格式都相同。

在Action接口中有五个常量和一个execute方法：

五个常量分别是：

Public static final String SUCCESS = “success” ; 运行成功时返回的逻辑视图

Public static final String NONE = “none” ; 不返回任何结果页面

Public static final String ERROR = “error” ; 运行失败时返回的逻辑视图

Public static final String INPUT = “input” ; 校验失败时返回的逻辑视图

Public static final String LOGIN = “login” ; 登录失败时返回的逻辑视图

**解耦合方式调用servlet的API:**

解耦合的概念就是代码中不直接依赖于request对象和response对象。

Struts2设计的思想就是与servlet API解耦合，Action中不再依赖Servlet的

那么如何调用这些API呢？Struts2为我们提供了一个API，可以间接调用servlet api焦作ActionContext类（Action上下文）。（可以理解为该类就是一个工具类）

该API提供大量间接操作servlet API 方法：

ActionContext：

getContext()返回ActionContext实例对象

Get(key)相当于HttpServletRequest的getAttribute(String name)方法

getApplication()：返回一个Map对象，存取servletContext属性

getSession()：返回一个Map对象，存取session属性。

getParameters()：类似调用HttpServletRequest的getParamterMap()方法。

SetSession(Map):将Map实例里的key-value保持为HttpSession的属性名称，属性值。

范例：

package wp.struts\_01;

import com.opensymphony.xwork2.ActionContext;

import com.opensymphony.xwork2.ActionSupport;

public class IndirectServletAPIAction extends ActionSupport{

@Override

public String execute() throws Exception {

//接收参数

ActionContext context = ActionContext.getContext() ; //取得ActionContext对象

java.util.Map<String, Object> map = context.getParameters() ; //类似调用HttpServletRequest的getParamterMap()方法。

String[] names = (String[])map.get("name") ;//map.get(key)实际上是返回的一个字符串数组，所以用数组接收

System.out.println(names[0]) ;

return NONE;

}

}

<!-- 访问连接http://localhost:8080/Struts2\_chapter04/indirect?name=admin -->

<action name = *"indirect"* class = *"wp.struts\_01.IndirectServletAPIAction"*>

<result name = *"success"*></result>

Servlet API 直接调用（耦合方式）：

1. 接口注入方式操作Servlet API(了解)

实现接口，访问Action时完成注入

实现ServletRequestAware接口：可获取到request对象。

实现ServletResponseAware接口：可获取到response对象。

实现ServletContextAware接口：可获取到ServletContext对象。

范例：取得request对象

**package** InjectServletAPI;

**import** javax.servlet.http.HttpServletRequest;

**import** org.apache.struts2.interceptor.ServletRequestAware;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** InjectServletAPIAction **extends** ActionSupport **implements** ServletRequestAware{

/\*\*

\* 自动获取request对象

\* \*/

HttpServletRequest request ;

@Override

**public** **void** setServletRequest(HttpServletRequest arg0) {

**this**.request = arg0 ; //通过setServletRequest(HttpServletRequest arg0)方法取得request对象

}

**public** String execute()**throws** Exception{

String name = request.getParameter("name") ; //通过request对象取得属性。

System.***out***.println(name);

request.setAttribute("info", name) ;

**return** ***SUCCESS*** ;

}

}

访问路径为：http://localhost:8080/Struts2\_chapter04/in?name=admin

配置文件：</action>

<!-- 访问连接http://localhost:8080/Struts2\_chapter04/in?name=admin -->

<action name = *"in"* class = *"InjectServletAPI.InjectServletAPIAction"*>

<result name = *"success"*>/myaction/indirect.jsp</result>

</action>

**2.通过ServletActionContext类的静态方法直接获取Servlet API(推荐)**

使用ServletActionContext：

Static PageContext getPageContext() ; 取得pageContext对象

Static HttpServletRequeset getRequest() ; 取得request对象

Static HttpServletResponse getResponse() ; 取得response对象

Static ServletContext getServletContext() ; 取得servletContext对象

范例：通过ServletActionContext类的静态方法直接获取Servlet API

**package** InjectServletAPI;

**import** javax.servlet.http.HttpServletRequest;

**import** org.apache.struts2.ServletActionContext;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** DirectServletAPIAction **extends** ActionSupport{

@Override

**public** String execute() **throws** Exception {

HttpServletRequest request = ServletActionContext.*getRequest*() ; //取得request对象

String name = request.getParameter("name") ;

System.***out***.println(name) ;

**return** ***NONE***;

}

}

在以上程序中直接通过ServletActionContext类的静态方法直接获取了servlet的内置对象。

配置文件：

<!-- 通过ServletActionContext类的静态方法来获取servlet的API -->

<action name=*"direct"* class = *"InjectServletAPI.DirectServletAPIAction"*>

</action>

访问路径：<http://localhost/8080/Struts2_chapter04/direct?name>= admin

比较以上三种方式：第三种方式最为方便。