**jQuery3源码解读第1天课程**

【学习目标】

## 1，jQuery源码结构和作用

1. 说出jQuery源码模块组成
2. jQuery模块相互依赖关系
3. jQuery设计理念

## 2，能够理解NPM的用途及用法

1. npm用途
2. 使用npm进行包安装、删除、更新
3. 使用npm初始化项目

## 3，能够grunt作用并掌握配置

1. 说出说出grunt在项目管理中的作用
2. 掌握grunt的配置
3. 掌握grunt插件的配置

## 4，能够理解git作用及常用操作

1. 说出git在项目管理中的作用
2. 掌握git常用操作

# jQuery3框架结构及项目管理工具

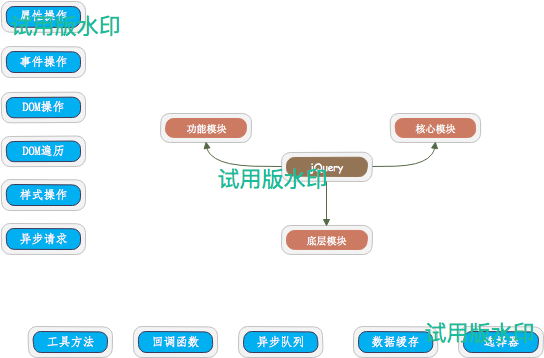
## jQuery是什么

jQuery是简化原生js操作dom元素的库。对于前端js来说，实现用户和页面元素的交互，提高用户体验是其根本任务。创建和获取DOM元素，操作元素的属性和样式，修改元素的内容，实现动画效果、事件处理，处理Ajax请求是前端js常见任务。要完成这些操作，不同厂商浏览器及不同版本的浏览器的API还有差异。jQuery他不仅解决了浏览器兼容性问题，还简化了代码开发。

## jQuery设计理念

* 轻量级
* **支持Css3**
* 支持各种主流浏览器

## jQuery3模块分类及依赖关系



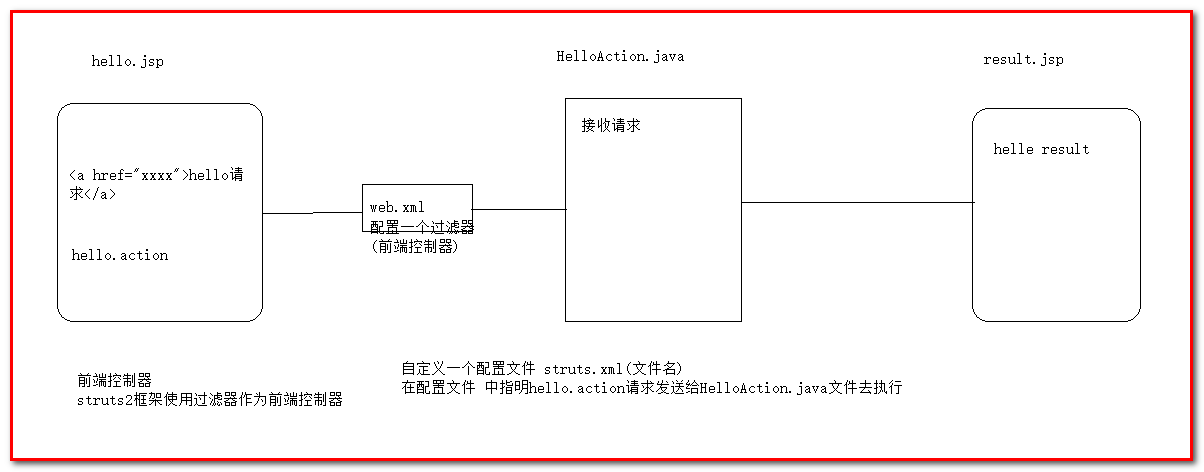
## jQuery3开发使用到的技术

* 面向对象编程
* requirejs
* NPM
* grunt
* git

**requirejs的使用**

# Struts2的HelloWorld

## Hello World（重点）



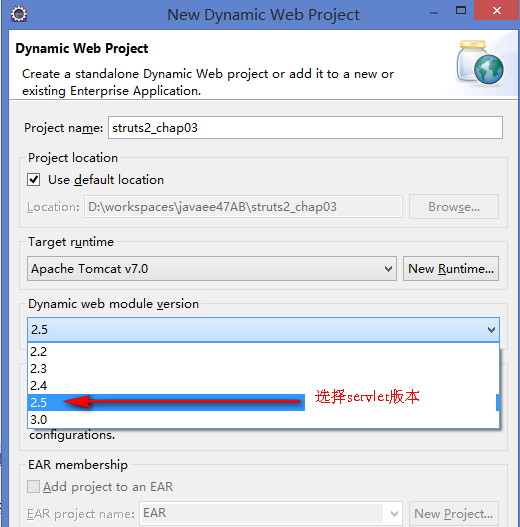
【练习题目】：使用struts2编写一个简单的请求和跳转的功能，从hello.jsp页面经过struts2，跳转到result.jsp页面

开发工具：eclipse Mars.2

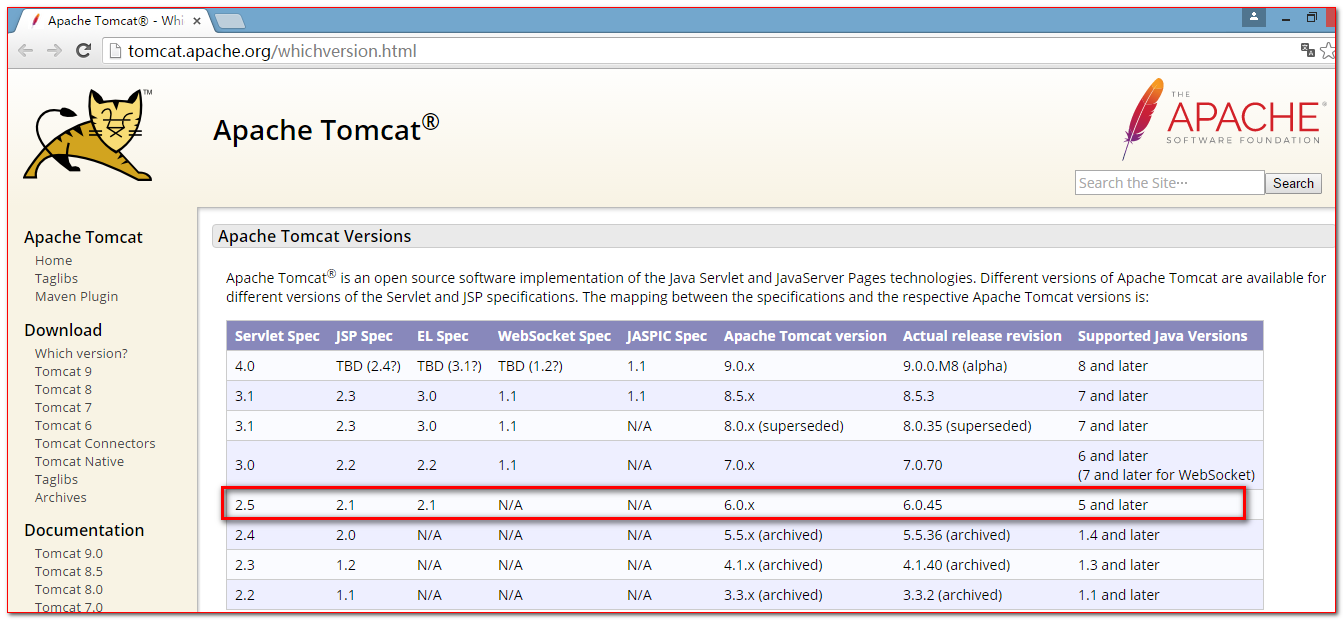
开发步骤：

|  |
| --- |
| 1. 创建WEB 工程 2. 导入必要jar包（struts2开发jar包） 3. 编写JSP 页面 4. 编写Action代码处理逻辑 5. 进行框架配置web.xml、struts.xml 6. 运行测试 |

【第一步】、创建Web工程：struts2\_chap03



注意：选择servlet2.5或以上版本，编译jdk选大于5的，比如6或7的。



【第二步】、导入jar包

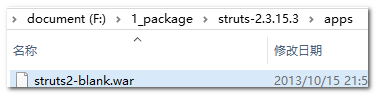
如何快速搭建一个应用？



开发时没必要将lib目录下jar文件全部复制到项目中，课程中将使用apps中struts2-blank.war中的jar。

导入Struts2需要的Jar包（13个）：apps/struts2\_blank.war/WEB-INF/lib/

找到war，解压：





Lib目录中的jar包介绍(重要程度排序)：

|  |  |  |
| --- | --- | --- |
| 1 | struts2-core-2.3.x.jar | Struts 2框架的核心类库 |
| 2 | xwork-core-2.3.x.jar | Command命令模式框架, Struts2和WebWork都基于和依赖xwork |
| 3 | ognl-3.0.x.jar | OGNL对象图导航语言(Object Graph Navigation Language), struts2框架通过其读写对象的属性。（类似于EL但更强大） |
| 4 | commons-fileupload-1.2.2.jar | 文件上传组件，2.1.6版本后需要加入此文件 |
| 5 | commons-io-2.0.1.jar | 上传文件依赖的jar包 |
| 6 | commons-lang-2.5.jar | 对java.lang包的增强，主要是提供字符串等操作的公用方法 |
| 7 | commons-logging-1.1.x.jar | ASF出品的日志包，Struts 2框架使用这个日志包来支持Log4J和JDK 1.4+的日志记录。（JCL，java通用日志接口） |
| 8 | log4j\*.jar | 一个日志实现 |
| 9 | freemarker-2.3.\*.jar | 模版引擎，struts2标签库，依赖freemarker ，Struts 2的UI标签的模板使用FreeMarker编写（页面静态化的技术，根据数据表的结构自动生成jsp页面、domain、dao、action...） |
| 10 | asm-\*.jar | 字节码操作类库 （.class文件） |
| 11 | asm-commons-3.3.jar | 提供了基于事件的表现形式 |
| 12 | asm-tree-3.3.jar | 提供了基于对象的表现形式 |
| 13 | javassist\*.jar | 生成代理的类库 |

【第三步】编写JSP页面：在WebContent路径下面，创建本项目的第一个文件夹01\_hello，在01\_hello中创建请求和响应页面：hello.jsp和result.jsp

在请求页面hello.jsp代码如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>My JSP 'hello.jsp' starting page</title>  </head>    <body>  <!--  struts2的请求默认以\*.aciton结尾（虽然可以不加.action，需要另外配置）  -->  <a href=*"*${pageContext.request.contextPath }*/hello.action"*>hello请求</a>  </body>  </html> |
| 【提示】  访问链接中的.action后缀是struts2的推荐写法。默认情况下框架接收以.action请求，并进行处理。 |

相应的result.jsp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>  hello result  </body>  </html> |

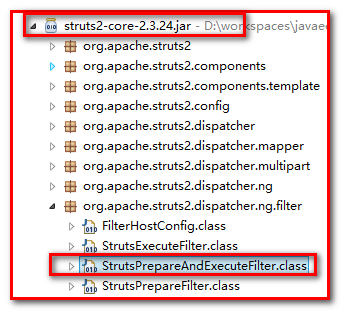
【第四步】编写服务端Action代码来处理请求：(相当于servlet)

在com.igeek\_01\_hello包中创建HelloAction.java，实现Action的接口

|  |
| --- |
| **package** com.igeek\_01\_hello;  **import** com.opensymphony.xwork2.Action;  /\*\*  \* 这个类相当于Servlet，处理具体的业务逻辑  \* 这个类的实现方式有N种，我们先采用一种进行实现：直接 implements Action  \* 实现com.opensymphony.xwork2.Action  \* 当 implements Action这个接口之后，我们发现系统报错  \* 得去实现接口中公共抽象未实现的方法  \* **@author** jerry  \*  \*/  **public** **class** HelloAction **implements** Action{  /\*\*  \* 实现Action必须要实现execute方法，在这个方法中，进行业务逻辑的处理  \* 包括调用service、调用dao、调用entity、domain....  \*/  **public** String execute() **throws** Exception {  //调用Service、dao...  System.*out*.println("HelloWorld");  //返回一个字符串  **return** "success";  }  } |

【第五步】、编写框架相关配置：web.xml和struts.xml

<1>在web.xml中配置前端控制器



|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app version=*"2.5"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*  *http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>  <display-name></display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  <!-- 配置struts2框架的过滤器 -->  <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>    </web-app> |

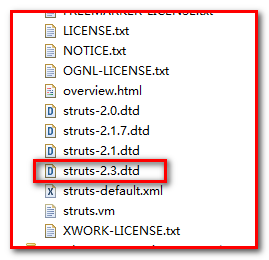
<2>在src中建立**struts.xml**（核心配置文件）

正常情况下，这个灵魂文件，就叫struts.xml， 没人去改这个名字，而且放置在src路径下面（系统自动加载）

【注意点1】

在struts-core.jar包中找到struts-2.3.dtd约束，打开：





约束如下：

<!DOCTYPE struts PUBLIC

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

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

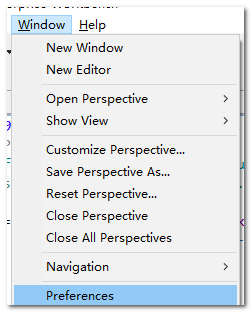
【注意点二：XML自动提示】

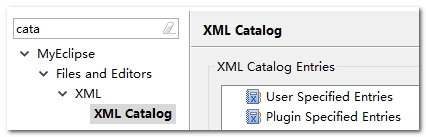
* 如果电脑可以联网，自动从网上下载 DTD，缓存提供提示。
* 如果电脑没有联网，在本地配置DTD文件位置，提供本地提示。

复制dtd链接：

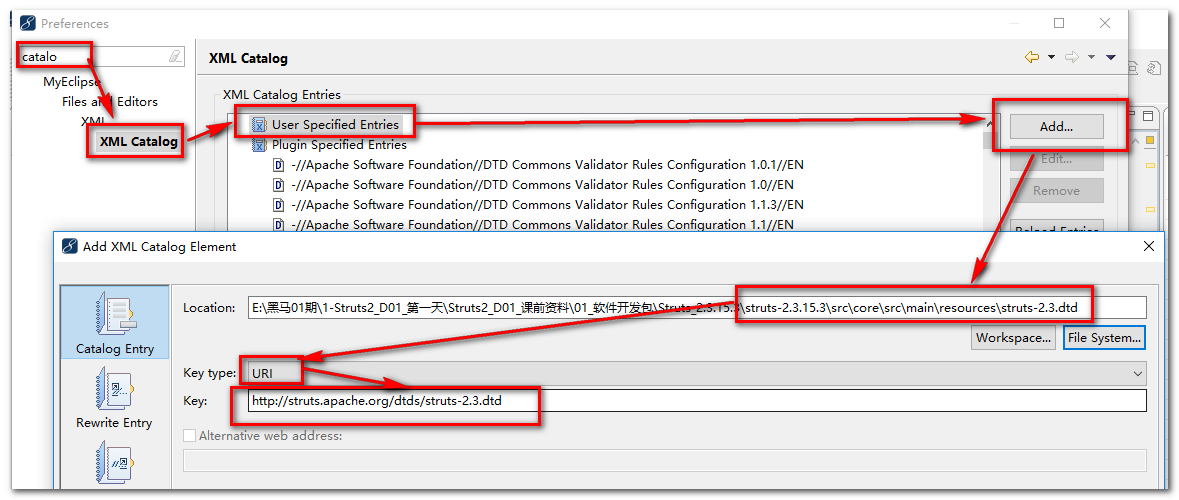


打开eclipse的：

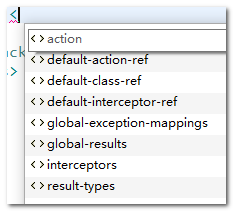




dtd文件可以从源码中获取，可以将其放到你喜欢的地方保存复用：



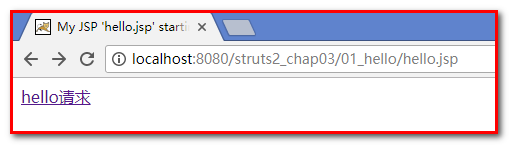
确定，必须要重新打开struts.xml。



【注意点三：struts.xml配置文件如下】

|  |
| --- |
| <?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>  <!--  把http://localhost:8080/struts2\_chap03/hello.action请求  与com.igeek\_01\_hello.HelloAction.java类匹配  分3步：  第一步:配置package节点，匹配项目后和action前的内容，  例如http://localhost:8080/struts2\_chap03/hello.action，匹配/  第二步:配置action节点，匹配action  例如http://localhost:8080/struts2\_chap03/hello.action，匹配hello.action  第三步:配置result节点，匹配结果页面  return "success"匹配 a\_hello/result.jsp  -->  <!--  package:包：实际开发中每个包表示不同的模块  name：包名，不同模块的名称，不能相同  extends：继承,必须继承struts-defalut，在struts-defalut包中实现了非常多的功能  namespace：名称空间，与访问路径相关的  例如:${pageContext.request.contextPath }/hello.action  namespace="/" 就是hello.action前的/  -->  <package name=*"default"* extends=*"struts-default"* namespace=*"/"*>  <!-- Hello的action  name:请求的名字  例如:${pageContext.request.contextPath }/hello.action  这里的hello.action对应到action节点中的name为hello,struts2框架会自动去掉后缀名.action  class:Action对应的完整的包路径  -->  <action name=*"hello"* class=*"com.igeek\_01\_hello.HelloAction"*>  <!-- 配置业务逻辑处理完成之后，跳转的action  name:与action中return的String类型的值对应  -->  <result name=*"success"*>/01\_hello/result.jsp</result>  </action>    </package>  </struts> |

第六步、部署测试：



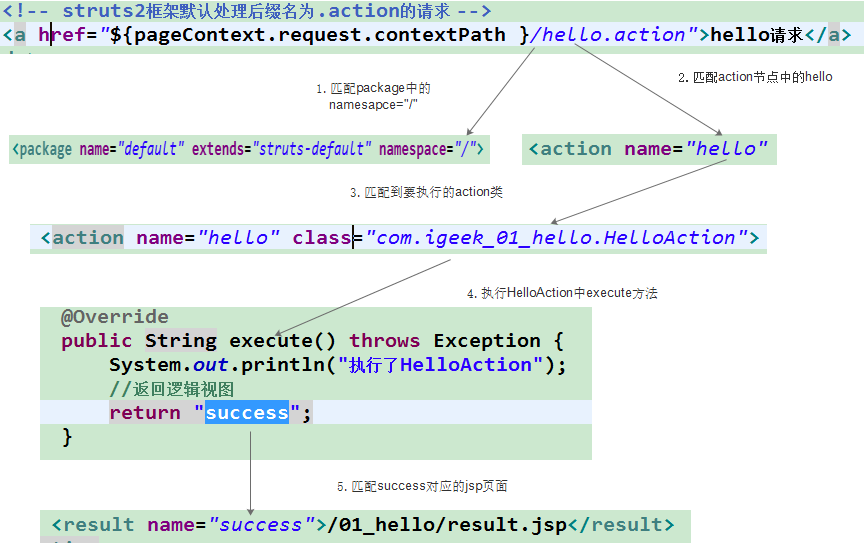
【回顾】

如何在jsp中发起请求：

* form表单提交，采用的是get或post请求
* 超链接<a>，采用的是get请求

关键字：请求方式hello.action、HelloAction(servlet)、Action接口、execute()、”success”、web.xml、struts.xml

## 程序基本运行流程分析（请求和响应）（重点）



## Config Brower插件---查看Action的配置信息加载信息(了解)

【需求】：当struts.xml文件中Action配置非常多的时候，你怎么能够清晰的看出Action与响应的路径之间的关系？

答：使用struts2的配置查看工具。

作用：查看Struts.xml文件中的配置：项目中的action及其与jsp页面之间的映射

知识点：如何使用struts2的插件。

strut2的插件的安装，只需要将插件复制到工程的lib中即可。



导入插件包 ，重新启动项目 ，访问:

localhost:8080/struts2\_chap03/config-browser/index.action

通过路径，查看struts2 Action 配置加载情况



# 常用配置

## 核心配置文件的分2类（两大类共6种）

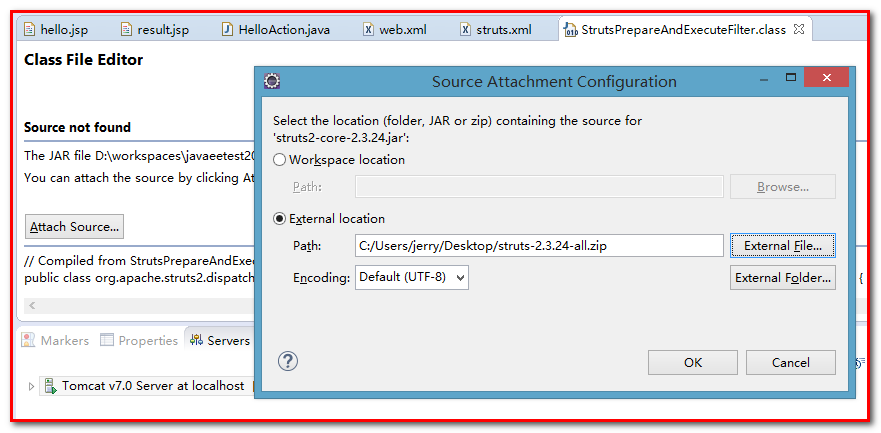
【问题】我们发现，在搭建第一个struts2项目的时候，我们配置了web.xml文件和struts.xml两个文件，那么

struts2在运行的过程中，到底要加载几个xml文件？大家想知道吗？

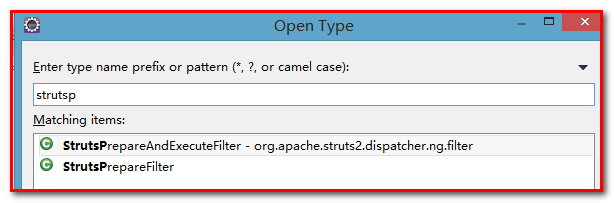
所以说：接下来，我们学习：核心配置文件的种类和加载顺序

【官方源码】

【源码关联】



Ctrl+Shift+T:搜索源码





StrutsPrepareAndExecuteFilter过滤器会加载哪些配置文件呢？  
 \* 通过源代码可以看到具体加载的配置文件和加载配置文件的顺序  
 \* init\_DefaultProperties();             -- 加载org/apache/struts2/default.properties  
 \* init\_TraditionalXmlConfigurations();        -- 加载struts-default.xml,struts-plugin.xml,struts.xml  
 \* init\_LegacyStrutsProperties();           -- 加载自定义的struts.properties.  
 \* init\_CustomConfigurationProviders();        -- 加载用户自定义配置提供者  
 \* init\_FilterInitParameters() ;           -- 加载web.xml的初始化信息

重点了解的配置文件  
 \* default.properties    -- 在org/apache/struts2/目录下，代表的是配置的是Struts2的常量的值  
 \* struts-default.xml    -- 在Struts2的核心包下，代表的是Struts2核心功能的配置（Bean、拦截器、结果类型等）  
 \* struts.xml        -- 重点中的重点配置，代表WEB应用的默认配置，在工作中，基本就配置它就可以了！！（可以配置常量）  
 \* web.xml          -- 配置前端控制器（可以配置常量）  
          
 \* 注意：  
 \* 前3个配置文件是struts2框架的默认配置文件，基本不用修改。  
 \* 后3个配置文件可以允许自己修改struts2的常量。但是有一个特点：后加载的配置文件修改的常量的值，会覆盖掉前面修改的常量的值。  
      
总结（重点掌握的配置文件）  
 \* 先加载default.properties文件，在org/apache/struts2/default.properties文件，都是常量。  
 \* 又加载struts-default.xml配置文件，在核心的jar包最下方，struts2框架的核心功能都是在该配置文件中配置的。  
 \* 再加载struts.xml的配置文件，在src的目录下，代表用户自己配置的配置文件  
 \* 最后加载web.xml的配置文件  
          
 \* 后加载的配置文件会覆盖掉之前加载的配置文件（在这些配置文件中可以配置常量）  
 \* 加载default.properties（默认常量值）  
 \* 加载struts-default.xml（核心的功能），编写包的配置都需要继承struts-default，但是配置文件中不能编写常量的  
 \* 加载struts.xml（开发必须要编写的配置文件），编写常量的（开发中编写常量的地方）  
 \* 加载web.xml，也可以编写常量（开发中尽量不要在该配置文件中编写常量值）  
      
注意一个问题  
 \* 哪些配置文件中可以配置常量？  
 \* default.properties        -- 默认值，咱们是不能修改的！！  
 \* struts.xml            -- 可以配置，开发中基本上都在该配置文件中配置常量  
 \* struts.properties         -- 可以配置，基本不会在该配置文件中配置  
 \* web.xml              -- 可以配置，基本不会在该配置文件中配置  
          
 \* 后加载的配置文件会覆盖掉之前加载的配置！！

## 灵魂文件struts.xml配置详解（重点）

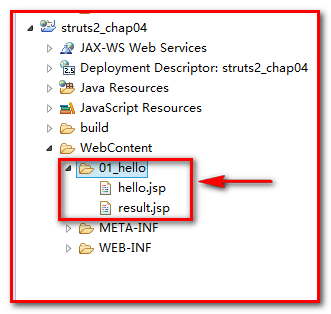
在本节中，我们将重点讲解package、action、result以及struts2内置常量的用法

### <package>详解

|  |
| --- |
| * package标签：   功能：用来定义不同的模块，管理和配置Action，并且实现包内配置复用 （通过包继承实现 ）   * name属性：包的名称，在struts容器中具有唯一性（在开发中可以用模块名称作为包名） * extends属性：继承父package中的功能，通常都继承struts-default。由于该默认包内定义了大量结果集类型和拦截器，所以struts强制你继承struts-default（不继承struts-defalut，就会报错）。 * namespace属性：名称空间用来标识一个路径，来区分不同action的访问路径。   Action的访问路径= namespace包名称空间 + action的name属性  默认为“”，在实际开发中，通常采用模块名字作为访问路径  【namespace知识点1】  当namespace="/xxx"的时候，访问路径路径就是/xxx/\*.action  如果namespace="/xxx/ooo"的时候，访问路径就是/xxx/ooo/\*.action  /xxx/ooo/yyy /xxx/ooo/yyy/\*.action  【namespace知识点2】当请求action路径是如下路径的时候  /xxx/a/b/c/d/e/hello.action，先去精确匹配，如果找不到  就去/xxx/a/b/c/d/\*.action中查找，如果还找不到  就去/xxx/a/b/c/\*.action中查找，如果还找不到  /xxx/a/b/\*.action  ....  一直找到/\*.action，如果这个里面还找不到。  就去namespace=""的里面查找 ,如果还找不到，就直接报错  【面试题】namespace="/"与namespace=""的区别：  /的优先级别比“”高    【注意点三】为什么要继承struts-default？  解答：  参考 struts-core.jar 提供 struts-default.xml 配置文件 ，在这个xml文件中定义了大量的结果集类型和拦截器    【查看类型代码】  通过快捷键shift+ctrl+T打开搜索框，然后输入      【说明】：在struts.xml文件中，可以配置多个package标签 |

【示例1】演示不同的包中相同的action通过不同的名称空间（namespace）来访问。

【第一步】：新建web项目struts2\_chap04,导入struts2架包，复制struts2\_chap03中的hello.jsp和result.jsp页面：



修改hello.jsp代码如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>My JSP 'hello.jsp' starting page</title>  </head>    <body>  <!--  struts2的请求默认以\*.aciton结尾（虽然可以不加.action，需要另外配置）  -->  <a href=*"*${pageContext.request.contextPath }*/xxx/hello.action"*>config请求</a>  </body>  </html> |

修改result.jsp，代码如下：

|  |
| --- |
| <%@ 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>  config result  </body>  </html> |

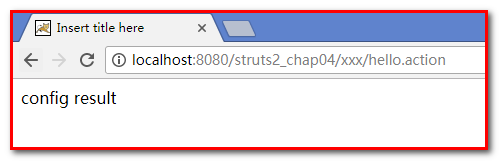
【第二步】：创建一个包com.igeek\_01\_config，新建HelloAction.Java，代码如下：

|  |
| --- |
| **package** com.igeek\_01\_config;  **import** com.opensymphony.xwork2.Action;  /\*\*  \* 这个类相当于Servlet，处理具体的业务逻辑  \* 这个类的实现方式有N种，我们先采用一种进行实现：直接 implements Action  \* 当 implements Action这个接口之后，我们发现系统报错  \* 得去实现接口中公共抽象未实现的方法  \* **@author** jerry  \*  \*/  **public** **class** HelloAction **implements** Action{  /\*\*  \* 实现Action必须要实现execute方法，在这个方法中，进行业务逻辑的处理  \* 包括调用service、调用dao、调用entity....  \*/  **public** String execute() **throws** Exception {  //调用Service、dao...  System.***out***.println("b-config中的HelloWorld...");  //返回一个字符串  **return** "success";  }  } |

【第三步】：配置web.xml代码与struts2\_chap03中相同, struts.xml中的代码

|  |
| --- |
| <?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>  <!-- 在struts.xml中可以定义多个package ，package对应不同的模块-->  <package name=*"anotherpackage"* extends=*"struts-default"* namespace=*"/xxx"*>  <action name=*"hello"* class=*"com.igeek\_01\_config.HelloAction"*>  <!-- 配置返回的结果  name:返回的结果  -->  <result name=*"success"*>/b\_config/result.jsp</result>  </action>  </package>  <!--  package:包，以前java中包的作用：  1 在不同包中可以存放同名的java对象  2 不同功能的类可以根据包名来区别  struts2中的package标签的功能类似（一样）  name：包的名字（正常包名都是以模块名字命名）  extends:继承 ：在struts2中必须直接(或者间接的)继承struts-default  namespace:访问路径  注意点1：  如果namespace="/",那么访问路径就是 /\*.action  如果namespace="/xxx",那么访问路径就是/xxx/\*.action  如果namespace="/xxx/ooo", 那么访问路径就是/xxx/ooo/\*.action  注意点2：  当访问路径是 /aa/bb/cc/xxx/\*.action的时候  如果 /aa/bb/cc/xxx/\*.action这个路径下面没有找到action，就找上一层目录/aa/bb/cc/\*.action  如果/aa/bb/cc/\*.action还没有找到，就继续找上一层/aa/bb/\*.action  如果/aa/bb/\*.action还没有找到，就继续找上一层/aa/\*.action  ..一直找到最后，如果还找不到，报错    -->  <package name=*"default"* extends=*"struts-default"* namespace=*"/"*>  <!-- 配置Action  name:action的名字，与请求URL对应  class:对应的action完整的包路径  -->  <action name=*"hello"* class=*"com.igeek\_01\_hello.HelloAction"*>  <!-- 配置返回的结果  name:返回的结果  -->  <result name=*"success"*>/a\_hello/result.jsp</result>  </action>  </package>  </struts> |

【第四步】：访问测试：

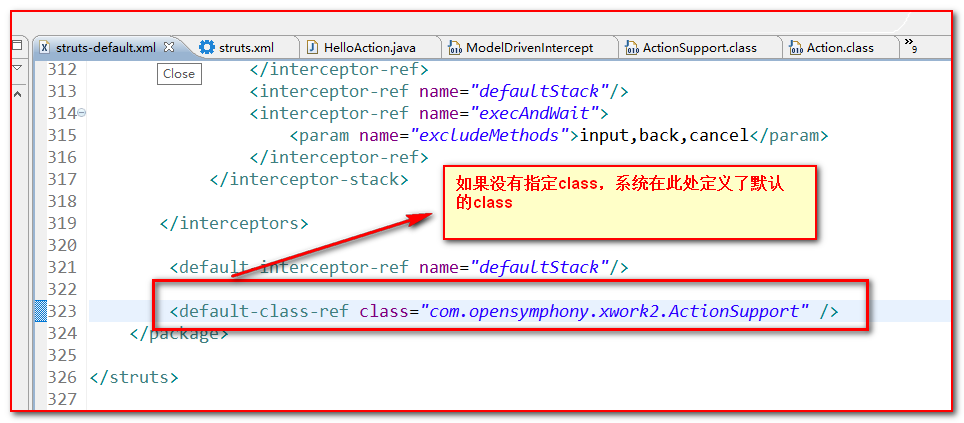


小结：namespace主要用来区分action，影响到action的访问路径。一般情况namespace为模块名，比如namespace=”/user”,访问的url为http://localhost:8080/project/user

### <action>详解

|  |
| --- |
| * action标签：用来管理具体的Action，负责接收客户端的请求，进行处理，并且完成响应  1. name属性：action的名字，用于配置请求的URL路径。 2. class属性：action对应的完整包路径，该类编写了action具体业务逻辑代码。 3. 找到action之后，系统会默认会执行Action类中的execute方法。 4. 若没有class属性，系统会默认执行ActionSupport中execute方法，而ActionSupport的execute会默认返回 success逻辑视图，这种处理方式在struts-default.xml 文件已经进行了规定。 5. 当访问路径中action的名字不存在的时候，系统会报错 |

快捷键的使用：ctrl+o 查找当前类中的属性和方法



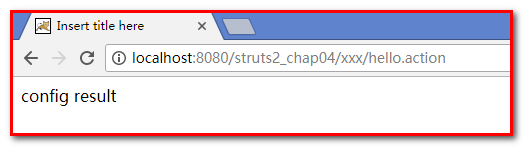
【示例1】

【需求说明】测试在Action的属性中，不添加class的时候，系统默认执行ActionSupport类中的execute方法，直接返回success

【第一步】：我们继续采用HelloAction.Java做测试，然后在struts.xml进行如下配置：

|  |
| --- |
| <?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=*"anotherPackage"* extends=*"struts-default"* namespace=*"/xxx"*>  <!-- 在同一个包中不能存在同名的action  action :  class:Action对应的完整的包路径 ,  可以省略，那么如果class属性省略了，那么系统会如何执行？  答： 如果没有class，会默认执行ActionSupport类的execute方法  -->  <action name=*"hello"*>  <result name=*"success"*>/01\_hello/result.jsp</result>  </action>  </package>  </struts> |

【第二步】：页面进访问http://localhost:8080/struts2\_chap04/01\_hello/hello.jsp，观察结果



【示例2】

【需求说明】

当页面访问不到对应的action的时候，我们尽量不要让找不到action这样的错误信息显示在用户面前，但是项目中难免会出现错误，所以这时候，我们可以进行如下设置，当找不到Action的时候，执行自定义的action。



如上图，当找不到对应的Action的时候，系统会报错，那么大家在项目上线之后，最忌讳的就是这样的错误出现在用户面前。

【第一步】：从资料中找到404-xsy.jpg放入img文件夹，在02\_action文件夹下面编写errorPage.jsp，

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP 'errorPage.jsp' starting page</title>  </head>    <body>  <img src=*"*${pageContext.request.contextPath }*/img/404-xsy.jpg"* />  </body>  </html> |

【第二步】：配置struts.xml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=*"anotherPackage"* extends=*"struts-default"* namespace=*"/xxx"*>  <!-- 在同一个包中不能存在同名的action  action :  class:Action对应的完整的包路径 ,  可以省略，那么如果class属性省略了，那么系统会如何执行？  答： 如果没有class，会默认执行ActionSupport类的execute方法  -->  <action name=*"hello"*>  <result name=*"success"*>/01\_hello/result.jsp</result>  </action>  </package>  <package name=*"default"* extends=*"struts-default"* namespace=*"/"*>  <!-- 为什么报错了之后，就要执行你的名为errorPage的action呢？ -->  <!-- 其实是需要在action中进行配置的 -->  <default-action-ref name=*"errorPage"*></default-action-ref>    <!-- 配置Action  name：这个name就是页面请求的action的名字  class:类对应的完整的包路径,这个class可以省略，  如果省略，系统会默认执行ActionSupport类中的execute方法，而这个方法直接返回success  -->  <action name=*"hello"*>  <result name=*"success"*>/01\_hello/result.jsp</result>  </action>  <!-- 当错误发生之后，由于这个action没有配置class，那么会默认执行ActionSupport类中的execute方法， 返回success -->  <action name=*"errorPage"*>  <result name=*"success"*>/02\_action/errorPage.jsp</result>  </action>    </package>  </struts> |

用户可以为每个package定义一个默认的Action，如果访问路径在package没有匹配<action> 就会执行默认action

【第三步】一定不能忘记配置默认的default-action-ref标签



【第四步】运行测试



小结：1.如果有对应action，正常运行

2.如果action只有name对应，没有class，默认执行ActionSupport类的execute方法

3.如果action的name对应不上，出错，我们应该给出一个友好的提示界面

### <constant>常量配置-（会配置即可）

* **struts.xml（推荐）**

<!-- 配置常量 -->

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

* **struts.properties**

key=value

* **web.xml**

<filter>

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

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

<init-param>

<param-name>struts.action.extension</param-name>

<param-value>xyz</param-value>

</init-param>

</filter>

**\*\*\*\*\*后加载配置文件中的常量的值会将先加载配置文件中常量的值覆盖！！！**

### 配置文件的分离（了解）

【开发场景】

在大部分应用里，随着应用规模的增加，系统中Action的数量也会大量增加，导致struts.xml配置文件变得非常臃肿。为了避免struts.xml文件过于庞大、臃肿，提高struts.xml文件的可读性，我们可以将一个struts.xml配置文件分解成多个配置文件，然后在struts.xml文件中包含其他配置文件。下面的struts.xml通过<include>元素指定多个配置文件：

<struts>

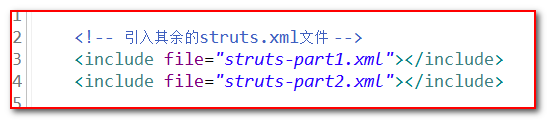
<include file="struts-part1.xml"/>

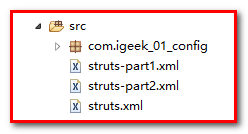
<include file="struts-part2.xml"/>

</struts>

通过这种方式，我们就可以将Struts 2的Action按模块添加在多个配置文件中。

实际开发过程中都是按照模块来进行划分。



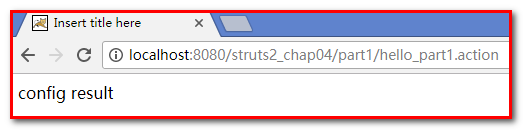


【struts-part1.xml】

|  |
| --- |
| <?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>  <!-- http://localhost:8080/struts2\_day01/part1/hello\_part1.action -->  <package name=*"part1"* extends=*"struts-default"* namespace=*"/part1"*>  <action name=*"hello\_part1"* class=*"com.igeek\_01\_config.HelloAction"*>  <result name=*"success"*>/01\_hello/result.jsp</result>  </action>  </package>  </struts> |

注意注意注意（重要的事情说三遍）：<include file="com/igeek\_01\_config/struts-part1.xml"/>

测试：<http://localhost:8080/struts2_chap04/part1/hello_part1.action>



# Action的编写方式和访问方式（重点）

## Action编写方式（3种）

Struts2中Action有三种实现方式：

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）

\* 对请求参数进行校验

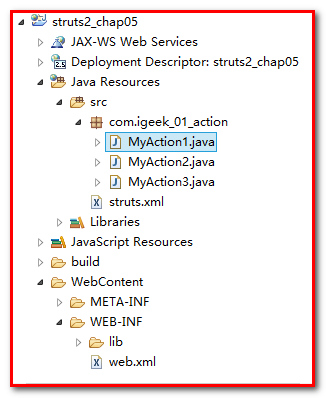
\* 设置错误信息

\* 读取国际化信息

第三种：pojo类（非侵入性、简单、干净）没有extends 父类，也没有implements 接口

说法：struts2是一个非侵入型的框架

【示例准备】



### 实现Action接口---第一种实现方式

Action 接口提供 execute 处理业务逻辑方法。

【第一步】在WebContent下面创建01\_action文件夹，在该文件夹中创建my.jsp页面中的代码如下：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>My JSP my.jsp' starting page</title>  </head>    <body>  <!-- 发起请求  struts2请求的时候，习惯以\*.action结尾  -->  <a href=*"*${pageContext.request.contextPath }*/my1.action"*>my1请求</a>  <br>  <a href=*"*${pageContext.request.contextPath }*/my2.action"*>my2请求</a>  <br>  <a href=*"*${pageContext.request.contextPath }*/my3.action"*>my3请求</a>  </body>  </html> |

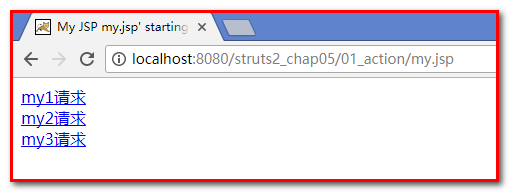
【第二步】：创建com.igeek\_01\_action包，然后编写MyAction1类，代码如下：

|  |
| --- |
| **package** com.igeek\_01\_action;  **import** com.opensymphony.xwork2.Action;  /\*\*  \* 这是Action的第一种实现方式--implements Action  \* 这时候，你可以使用接口中的5个常量和1个方法  \* 5个常量值：  \* SUCCESS  \* NONE:在struts.xml中就没有必要配置result标签了  \* INPUT:当struts运行的过程中，抛出异常的时候，会自动去查找名为input的结果集视图  \* LOGIN:默认查找名为login的结果集视图  \* ERROR:查找名为error的结果集视图  \* 1个方法：execute方法  \* **@author** jerry  \*  \*/  **public** **class** MyAction1 **implements** Action{  @Override  **public** String execute() **throws** Exception {  System.*out*.println("myAction1...");  **return** *NONE*;  }  } |

【第三步】：编写struts.xml，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=*"default"* extends=*"struts-default"* namespace=*"/"*>  <!-- action的实现方式一 -->  <action name=*"my1"* class=*"com.igeek\_01\_action.MyAction1"*>  </action>  </package>  </struts> |

【第四步】：访问测试：



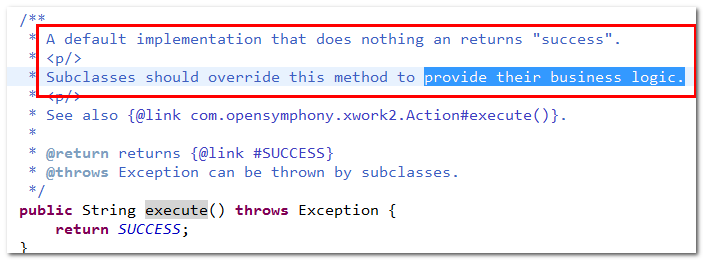
【内置的视图名称】

Action 接口提供一组 常用的内置的逻辑视图名称：

|  |
| --- |
| * SUCCESS 成功视图，默认值 * NONE 没有结果视图，用户自己生成响应数据 * ERROR 错误视图 * INPUT 输入视图 （数据输入非法，要求用户重新输入） * LOGIN 登陆视图 （如果用户未登陆，使用登陆视图） |

### 继承ActionSupport类---第二种实现方式

继承ActionSupport相当于间接实现Action接口，该类提供了更多功能，如数据校验、 国际化等，用的最多，使用的时候需要手动覆盖execute()。



【示例】

【第一步】：编写MyAction2类

|  |
| --- |
| **package** com.igeek\_01\_action;  **import** com.opensymphony.xwork2.ActionSupport;  /\*\*  \* Action的第二种实现方式，继承ActionSupport，这是推荐的方式！！！  \* 然后重写execute方法  \* 这种实现方式不光可以使用5个常量还可以使用ActionSupport类中实现的方法  \* **@author** jerry  \*  \*/  **public** **class** MyAction2 **extends** ActionSupport{    @Override  **public** String execute() **throws** Exception {  System.*out*.println("Action的第二种实现方式");  //这个时候为什么可以使用常量  **return** *NONE*;  }  } |

【第二步】：配置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=*"default"* extends=*"struts-default"* namespace=*"/"*>  <!-- action的实现方式一 -->  <action name=*"my1"* class=*"com.igeek\_01\_action.MyAction1"*>  </action>  <!-- action的实现方式二 -->  <action name=*"my2"* class=*"com.igeek\_01\_action.MyAction2"*>  </action>  </package>  </struts> |

【第三步】：访问 <http://localhost:8080/struts2_chap05/my2.action>



### 自定义POJO（了解）---第三种实现方式

POJO（Plain Old Java Object简单的Java对象），实际就是普通的JavaBean：没有父类，没有接口，干净

对于这样一个Action的执行，struts2使用反射的机制来进行调用执行。

【注意】**关于execute方法的编写要求，必须满足：**

* public修饰符
* String返回值
* 无参数
* throws Exception 可选

【示例】

【第一步】：编写MyAction3：

|  |
| --- |
| **package** com.igeek\_01\_action;  /\*\*  \* Action的第三种实现方式  \* 这种方式：不能使用常量也不能使用ActionSupport类中定义好的方法  \* 一个干净的java类  \* 这种实现方式既不能接口中常量，不能使用ActionSupport类中方法，所以说我们不推荐  \* **@author** jerry  \*  \*/  **public** **class** MyAction3 {    **public** String execute(){  System.*out*.println("Action的第三种实现方式");  //此处不能写大写的SUCCESS  **return** "none";  }  } |

【第二步】：编写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=*"default"* extends=*"struts-default"* namespace=*"/"*>  <!-- action的实现方式一 -->  <action name=*"my1"* class=*"com.igeek\_01\_action.MyAction1"*>  </action>  <!-- action的实现方式二 -->  <action name=*"my2"* class=*"com.igeek\_01\_action.MyAction2"*>  </action>  <!-- Action的实现方式三 -->  <action name=*"my3"* class=*"com.igeek\_01\_action.MyAction3"*>  </action>  </package>  </struts> |

其实，pojo的方式，底层使用反射机制，自动调用execute方法

【小结】

一般情况下推荐使用继承ActionSupport类的方法。

其他两种，大家了解就行了。

# Action的方法编写和调用执行

### Action的方法编写格式

总结：通过上面的方式，我们发现，每个类中都只有一个public String execute()的方法

问题：那么Action是不是默认只能执行execute方法？是不是只能写一个方法呢？

答：

一个action类中可以编写N个方法。

但是这些方法必须满足（方法名可以任意）：

|  |
| --- |
| * public修饰符 * String返回值 * 无参数 * throws Exception可选 |

那么如何调用这些方法呢？Struts2提供了三种方式来调用自定义的方法：

第一种方式：通过设置Method的属性来指定方法的名字

第二种方式：通过通配符的方式来指定方法的名字（推荐）

第三种方式：动态方法的调用：最大的特点：不安全

### 通过设置method属性调用Action业务方法

【需求信息】在一个Action业务中需要实现多个业务方法，如登录注册，如何实现？

【示例】创建一个Action，在Action中编写登录和注册两个方法，然后调用这两个方法

【第一步】：创建一个包：com.igeek\_02\_method,在这个包中，创建一个UserAction类型，继承ActionSupport类，

在里面创建login方法和regist方法，具体代码如下：

|  |
| --- |
| **package** com.igeek\_02\_method;  **import** com.opensymphony.xwork2.ActionSupport;  /\*\*  \* 通过Method指定调用的方法，不需要写execute方法了  \* **@author** jerry  \*  \*/  **public** **class** UserAction **extends** ActionSupport{    **public** String login(){  System.*out*.println("这是UserAction的login方法");  **return** *NONE*;  }    **public** String regist(){  System.*out*.println("这是UserAction的regist方法");  **return** *NONE*;  }    } |

【第二步】： 编写struts.xml:

【第三步】：访问测试：<http://localhost:8080/struts2_chap05/user_login.action>



【第四步】：总结

因此：可以在一个Action中编写多个业务方法，在访问时，通过method 区别不同业务。

缺点：调用一个方法，就要在struts.xml中配置一行，配置较多。

### 使用通配符来简化路径和Action业务方法的映射配置

通过通配符的使用简化Action的配置，从而实现不需要配置多个Action，就可以访问多个方法

【问题】单纯使用method属性来配置action的方法，调用不同的方法，需要配置不同的action。配置较多。

【解决方案】可以通过通配符的方式来解决。

【一个通配符的情况】重点掌握

* 在配置<action>元素的时候，允许在指定name属性的时候，使用模式字符串：\*代表一个或者多个任意字符
* 在class、method、result子元素中可以通过{N}形式代表前面的第N个\*匹配子串

访问规则：

{1} 对应 name中第一个\* 匹配的内容。 例如： 访问user\_login ------ \* 匹配 login -------- method就是login。

修改struts.xml中配置：

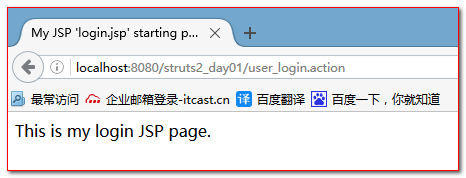
|  |
| --- |
| <!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*调用Action中自定义方法方式一：通过Method指定调用的方法\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->  <!-- <action name="user\_login" class="com.igeek\_02\_method.UserAction" method="login" />  <action name="user\_regist" class="com.igeek\_02\_method.UserAction" method="regist" />  -->  <!-- 调用Action中自定义方法方式二：通过通配符简化调用方式 -->  <action name=*"user\_\*"* class=*"com.igeek\_02\_method.UserAction"* method=*"{1}"*>  <result name=*"success"*>/02\_method/{1}.jsp</result>  </action> |

WebContent下新建02\_method文件夹，在此文件夹中新建login.jsp和regist.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>  login...  </body>  </html> |

|  |
| --- |
| <%@ 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>  result...  </body>  </html> |

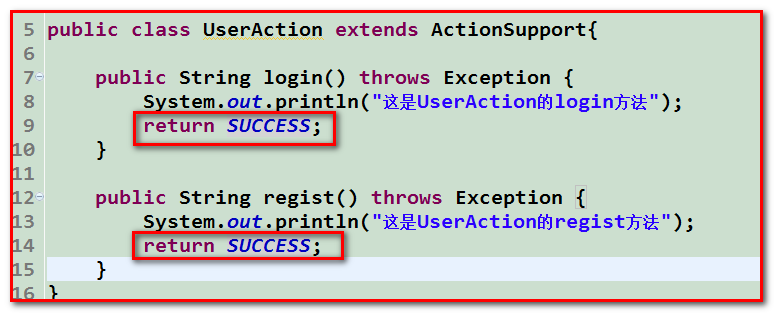
测试



【二个通配符的情况】（了解）

更复杂的情况：

修改Action中的返回结果为SUCCESS



实现N个action的N个方法的访问。

|  |
| --- |
| <!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*调用Action中自定义方法方式一：通过Method指定调用的方法\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->  <!-- <action name="user\_login" class="com.igeek\_02\_method.UserAction" method="login" />  <action name="user\_regist" class="com.igeek\_02\_method.UserAction" method="regist" />  -->  <!-- 调用Action中自定义方法方式二：通过通配符简化调用方式 -->  <!-- <action name="user\_\*" class="com.igeek\_02\_method.UserAction" method="{1}">  <result name="success">/02\_method/{1}.jsp</result>  </action> -->    <!-- 多个通配符的情况 -->  <action name=*"\*\_\*"* class=*"com.igeek\_02\_method.{1}"* method=*"{2}"*>  <result>/02\_method/{2}.jsp</result>  </action> |

访问测试：<http://localhost:8080/struts2_chap05/UserAction_regist.action>

注意：一个\*用多次的情况下，就要注意“约定”规则。

### 动态方法调用（了解）

【需求信息】单纯使用method属性来配置action的方法，调用不同的方法，需要配置不同的action。配置较多。

可以采用通配符的方式解决，也可以通过动态方法解决

优点：在配置action时无需配置method

缺点：不安全

【示例】

|  |
| --- |
| <!-- 调用Action中自定义方法方式三：通过动态方法调用  在默认情况之下，动态方法是关闭的，所以需要手动开启  -->  <action name=*"userDy"* class=*"com.igeek\_01\_method.UserAction"*>  </action> |

/user.action 执行 UserAction 的 execute

问题： 怎么执行login ？

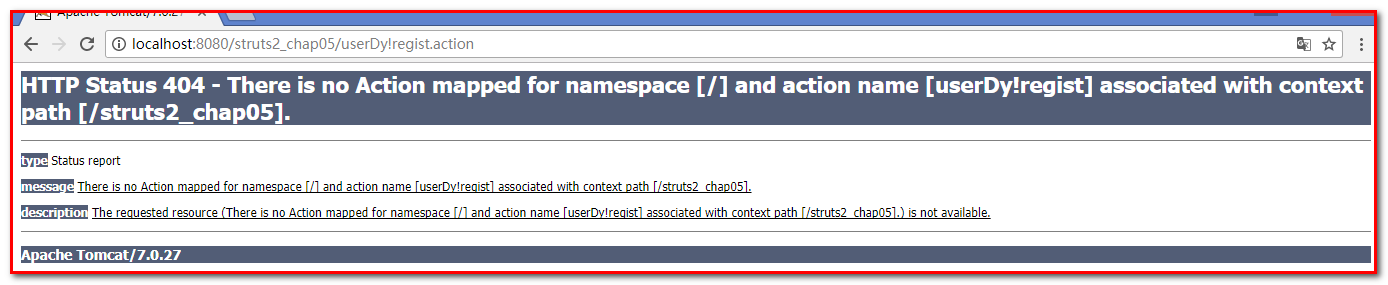
那么来看下这个方法在struts-default.properties文件中有这么一段话：



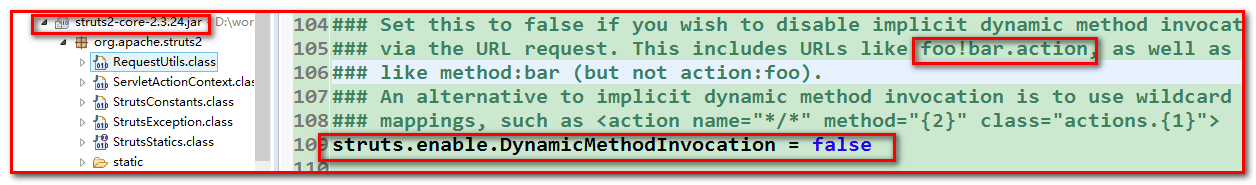
这句话的意思是：这个URL跟 foo!bar.action的方式差不多，其中bar是方法，foo是action的名字，下面我们来做一个案例



[http://localhost:8080/struts2\_chap05/userDy!regist.action](http://localhost:8080/struts2_chap05/userDy!.regist.action) 执行 UserAction的 regist 方法，报错：



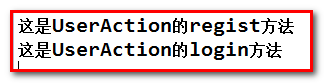
原因：在新版本struts2 中动态方法调用，默认是关闭的 ，在struts.xml文件中，将动态方法开启，具体配置如下





访问测试：<http://localhost:8080/struts2_chap05/userDy!regist.action>

<http://localhost:8080/struts2_chap05/userDy!login.action>



【小结】

action方法调用：

1. 默认调用execute方法。
2. method属性指定调用方法。虽可以任意调用任何的方法，但一个方法需要配置一个action。
3. method属性+通配符。可以一个配置，调用N方法。（推荐）
4. 动态方法调用，可以不配置method属性也能随意调用aciton中的任意方法。但必须开启动态方法调用。

重点和总结

1 必须完成HelloWord练习和运行流程图

2 Action的实现方式 ActionSupport

3 Action中的方法的调用方式 通配符

1. struts2 是一个MVC框架， 表现层框架， 和struts1、webwork关系
2. 开发struts2 应用，需要导入哪些jar包 （xml开发）
3. 如何配置struts.xml 配置文件提示 (重点)
4. struts2 运行流程
5. struts2 有哪些配置文件6个， 加载顺序
6. Action相关配置 ，访问Action路径 （package namespace + actionname）
7. 默认Action 和 Action 默认处理类
8. 常用常量配置 devMode、Action扩展名配置、动态方法调用开启
9. Action 三种实现方式 （掌握第二种：继承ActionSupport类)

10.Action 的方法调用 （method属性使用， 通配符使用， 动态方法调用（了解））