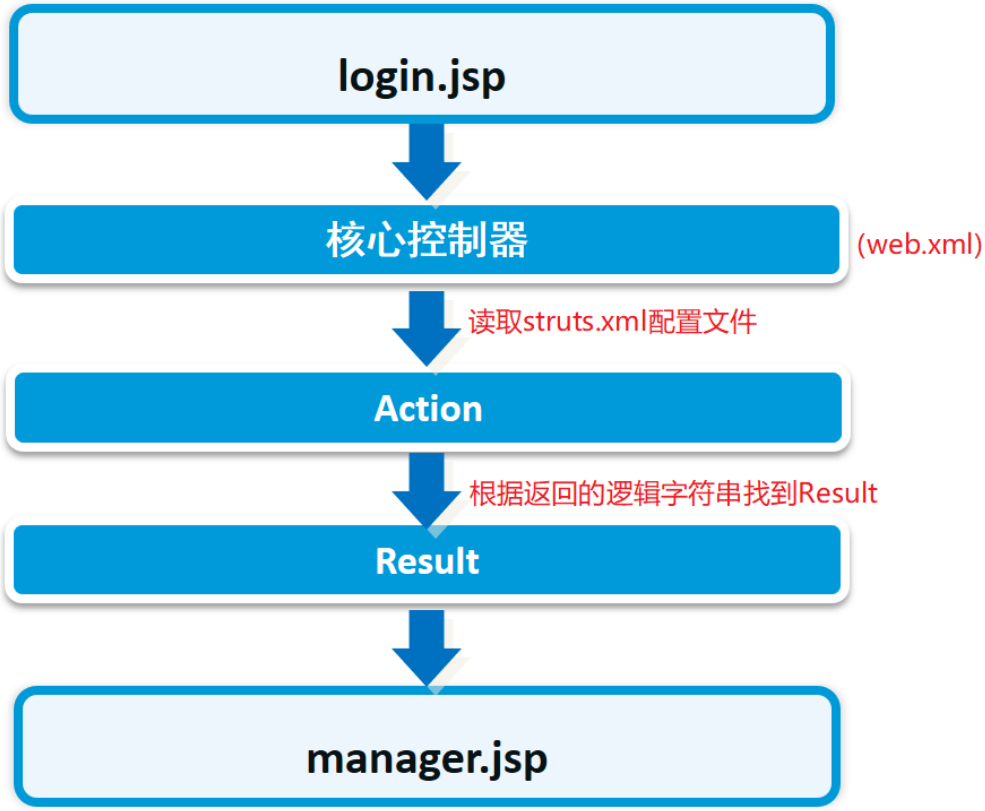
1 Struts2配置详解

1.1 Struts2 基本架构

1.1.1 Struts2执行流程

1. 浏览器将请求发送到服务器
2. 服务器接收请求，根据web.xml配置文件中，找到struts2的核心过滤器
3. 核心过滤器会将请求传递给struts.xml文件，struts2会根据action的配置找到相应的Action控制器，根据execute方法返回的结果字符串找result标签
4. 根据result标签的配置响应到jsp页面



1.1.2 web.xml

在项目的web.xml配置文件中加载struts2的核心控制器，代码如下：

```
<!-- struts2核心配置 -->
<filter>
  <filter-name>struts2</filter-name>
  <!-- struts核心控制器 -->
  <filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

1.1.3 Action

1. Action控制器由两部分组成，分别是：

   核心控制器(Filter):用于拦截用户请求，对请求进行处理

   业务控制器(Action):调用相应模型Model类实现业务处理，返回结果

2. 自定义的Action控制器可以实现Action接口或继承ActionSupport类

```
<package name="default" namespace="/" extends="struts-default">
    <action name="login" class="cn.houserent.action.LoginAction">
        <result name="success">/page/manage.jsp</result>
        <result name="input">/page/login.jsp</result>
        <result name="error">/page/error.jsp</result>
    </action>
</package>
```

1.1.4 Result

1. 实现对结果的调用
2. result元素的值指定对应的实际资源位置
3. name属性表示result逻辑名

```
<package name="default" namespace="/" extends="struts-default">
    <action name="login" class="cn.houserent.action.LoginAction">
        <result name="success">/page/manage.jsp</result>
        <result name="input">/page/login.jsp</result>
                    <result name="error">/page/error.jsp</result>
    </action>
</package>
```

1.2 Struts2 配置文件(struts.xml)

1.2.1 struts.xml

1. struts.xml配置文件是struts2框架的核心配置文件，主要负责管理Action
2. 通常放在WEB-INF/classes目录下，也可以在resources资源目录下，在该目录下的struts.xml文件可以被自动加载

```
<struts>
    <constant name="" value=""/>
    <package name="" namespace="" extends="">
        <action name="" class="">
            <result name=""></result>
        </action>
    </package>
</struts>
```

1.2.1.1 <constant>标签

1. 配置常量，可以改变Struts 2框架的一些行为
2. name属性表示常量名称，value属性表示常量值

```
<struts>
    <constant name="struts.i18n.encoding" value="UTF-8"/>
    <package name="" namespace="" extends="">
        <action name="" class="">
            <result name=""></result>
        </action>
    </package>
</struts>
```

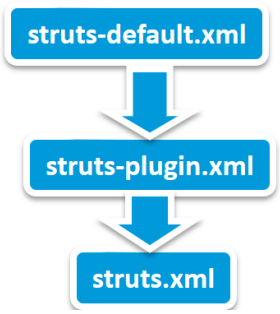
1.2.1.2 <package>标签

1. 包的作用：简化维护工作，提高重用性
2. 包可以“继承”已定义的包，并可以添加自己包的配置
3. name属性为必需的且唯一，用于指定包的名称

```
<struts>
	<constant name="" value=""/>
	<package name="default" namespace="/" extends="struts-default">
		<action name="" class="">
			<result name=""></result>
		</action>
	</package>
</struts>
```

1.2.2 struts配置文件加载顺序

- **struts-default.xml**
  - Struts 2默认配置文件，会自动加载
  - struts-default包在struts-default.xml文件中定义
- **struts-plugin.xml**
  - Struts 2插件使用的配置文件
- **加载顺序**



> 具体struts.xml文件如以下代码所示：

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

<!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"
        "http://struts.apache.org/dtds/struts-2.5.dtd">

<struts>

    <!-- 定义常量 -->
```

```
    <!-- 设置主题风格 -->
    <constant name="struts.ui.theme" value="simple"/>
    <!-- 设置编码格式 -->
    <constant name="struts.i18n.encoding" value="UTF-8"/>

    <!--
        package标签：用于区分功能模块，在struts2配置文件中可以有多个
            name属性：必填且唯一
            namespace属性：命名空间(一级目录)，默认为根目录"/"
            extends属性：用于继承struts的配置，可以继承struts-default,json-default,或自定义的
package
    -->
    <package name="default" namespace="/" extends="struts-default">
        <!--
            action标签：用于配置访问路径
                name属性：填写请求的访问路径，必填且唯一
                class属性：填写控制器的地址(全局路径)，该属性可以省略不写，省略则访问
ActionSupport类
                method属性：用于指定要执行的方法名称
        -->
        <action name="hello" class="com.bdqn.web.HelloAction" method="helloWorld">
            <!--
                result标签：配置返回的资源地址(可以返回到页面，也可以去到另外一个控制器)
                    name属性：配置返回的结果字符串(逻辑字符串)，默认为success
                    type属性：指定返回的方式，默认为dispatcher(转发)
            -->
            <result name="success" type="dispatcher">/success.jsp</result>
        </action>
    </package>



    <!-- 用户模块 -->
    <package name="user" namespace="/user" extends="default">
        <action name="hello"></action>
    </package>

</struts>
```

1.2.3 拆分配置文件

项目开发中，随着业务功能的新增，struts.xml配置文件的配置也会随之增加，可读性和维护性就会下降，为了增强后期的可维护性，可以通过拆分配置文件的方式解决问题。只需要将struts.xml业务模块的配置拆分成若干个struts-xxx.xm文件，最后在struts.xml主配置文件中通过 `<include>` 标签引入即可，代码如下：

```
<!-- 导入其他配置文件 -->
<include file="struts-house.xml"/>
<include file="struts-user.xml"/>
```

1.3 Action配置

1.3.1 Action的作用

1. 封装工作单元
2. 数据转移的场所
3. 返回结果字符串

1.3.2 访问Action的三种方式

1.3.2.1 method属性

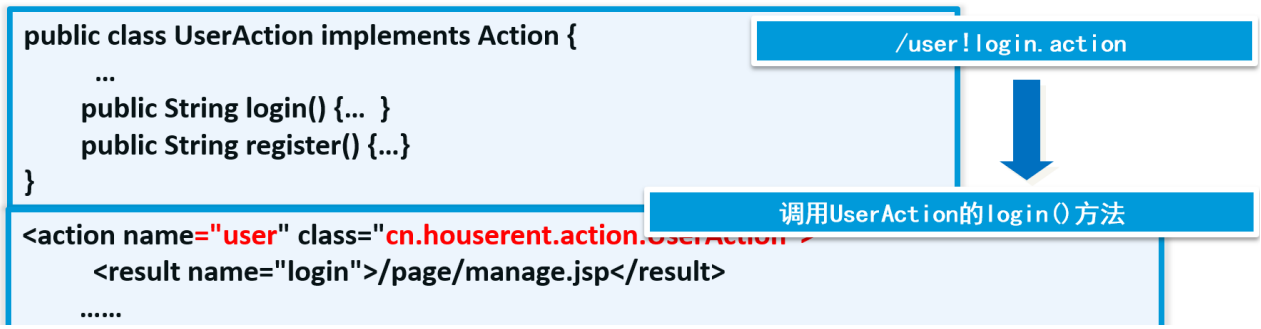
通过method属性实现在Action中不同方法的调用，该方式可以避免动态方法调用的安全隐患，但会导致大量的Action配置。

```
<action name="login"
        class="cn.houserent.action.UserAction" method="login">
      ......
</action>
```

1.3.2.2 Action动态方法调用

通过动态方法调用方法可以减少Action标签的配置数量，其用法为：`actionName!methodName.action`

禁用动态方法调用：将常量`struts.enable.DynamicMethodInvocation`设置为`false`。



> 使用动态方法调用，需要开启动态方法调用配置，以及设置白名单(struts2.5版本开始需要设置白名单)

`第1步：开启动态方法调用`

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

第2步：在<package>标签内设置白名单

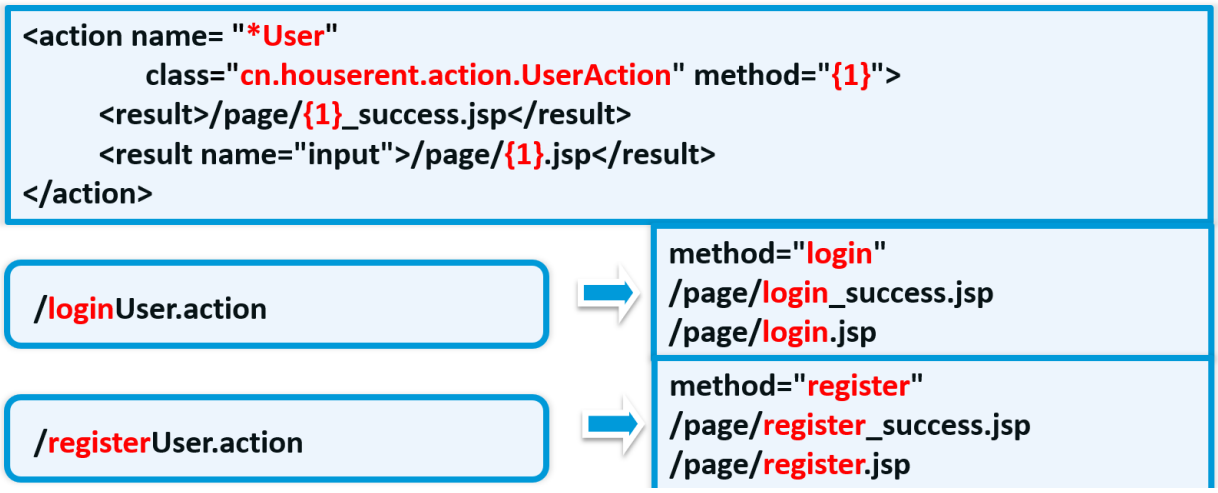
```xml
<!-- 设置白名单 -->
<global-allowed-methods>regex:.*</global-allowed-methods>
```

> 完整配置如下代码所示：

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

<!-- 公共配置信息 -->
<package name="default" extends="struts-default">

    <!-- 设置白名单 -->
    <global-allowed-methods>regex:.*</global-allowed-methods>

</package>
```

1.3.2.3 通配符(*)调用

作用：另一种形式的动态方法调用



注意：通配符方式调用也需要开启动态方法调用和设置白名单

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

<package name="user" namespace="/" extends="struts-default">
    <!-- 添加白名单 -->
    <global-allowed-methods>regex:.*</global-allowed-methods>
    <!-- 使用通配符 -->
    <action name="user_*" class="com.bdqn.web.UserAction" method="{1}">
        <result name="success">/success.jsp</result>
    </action>
</package>
```

1.3.3 配置默认Action

当struts2没有Action匹配请求时，默认Action将被执行

通过 `<default-action-ref ... />` 元素配置默认Action

```xml

<!-- 公共配置信息 -->
<package name="default" extends="struts-default">
    <!-- 引用默认的action -->
    <default-action-ref name="defaultAction"/>

    <!-- 404的action请求 -->
    <action name="defaultAction">
        <result name="success">/404.jsp</result>
    </action>
</package>
```

1.4 Result配置

1.4.1 常用的结果类型(type)

dispatcher类型：默认结果类型，后台使用RequestDispatcher转发请求

redirect类型 ： 后台使用的sendRedirect()将请求重定向至指定的URL

redirectAction类型 ： 主要用于重定向到Action

```xml
<package name="user" namespace="/user" extends="default">
    <action name="user" class="com.bdqn.web.UserAction">
        <result name="success" type="redirect">/success.jsp</result>
    </action>
```

```
5
6       <action name="login" class="com.bdqn.web.UserAction" method="login">
7           <!-- <result name="login_success" type="redirectAction">/user/find.action</result>--
    >
8           <result name="login_success" type="redirectAction">
9               <!-- 指定控制器名称 -->
10              <param name="namespace">/user</param>
11              <!-- 指定Action名称 -->
12              <param name="actionName">find.action</param>
13          </result>
14      </action>
15
16      <action name="find" class="com.bdqn.web.UserAction" method="find">
17          <result name="userlist">/userlist.jsp</result>
18      </action>
19
20  </package>
```

1.4.2 全局结果配置

概述：通过标签配置全局结果，注意该标签的顺序。

作用：定义公共的result返回页面，为了减少 `<result>` 标签配置的冗余

注意：当全局结果与局部结果配置冲突时，优先使用局部结果

```
1   <!-- 公共配置信息 -->
2   <package name="default" extends="struts-default">
3       <!-- 引用默认的action -->
4       <default-action-ref name="defaultAction"/>
5
6       <!-- 配置全局结果 -->
7       <global-results>
8           <result name="login">/login.jsp</result>
9           <result name="error">/error.jsp</result>
10      </global-results>
11
12      <!-- 设置白名单 -->
13      <global-allowed-methods>regex:.*</global-allowed-methods>
14      <!-- 404的action请求 -->
15      <action name="defaultAction">
16          <result name="success">/404.jsp</result>
17      </action>
18
19  </package>
```

1.5 Struts2封装获取表单数据方式

在struts2获取表单数据或提交路径的参数值的方式有4种，分别是：`原始Servlet方式`、`属性封装`、`表达式封装`、`模型驱动封装`

1.5.1 原始Servlet方式(了解)

使用Servlet的方式获取页面的数据，需要加入Servlet相关依赖：

```xml
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

> 使用原始Servlet封装不足之处：
>
> 1. 获取到所有的数据都是字符串类型，需要类型转换
> 2. 获取数据的代码复杂，啰嗦

```java
package com.bdqn.web;

import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.ServletActionContext;

import javax.servlet.http.HttpServletRequest;


/**
 * 原始封装：原生Servlet方式，获取到所有的数据都是String类型，需要类型转换
 */
public class UserAction1 extends ActionSupport {



    public String regsiter(){
        //获取Servlet对象
        HttpServletRequest request = ServletActionContext.getRequest();
        //获取用户名
        String userName = request.getParameter("userName");
        String password = request.getParameter("password");
        return SUCCESS;
    }

}
```

1.5.2 属性封装

1.5.2.1 概述

在类中定义成员变量，提供**get、set**方法

注意：必须保证成员变量名称与input表单中的name属性值、提交路径中的参数名称一致

该方式获取数据的方式比较简单，但是会造成属性过多，Action控制器代码冗余

1.5.2.2 控制器Action

```java
package com.bdqn.web;

import com.opensymphony.xwork2.ActionSupport;

public class UserAction2 extends ActionSupport {

    //必须与input标签的name属性值一致，或者与提交路径的参数名称一致
    private String userName;
    private String password;
    private Integer sex;
    private Integer age;
    private String phone;



    public String register(){
        System.out.println("用户名："+userName);
        System.out.println("密码："+password);
        System.out.println("性别："+sex);
        System.out.println("年龄："+age);
        System.out.println("电话："+phone);
        return SUCCESS;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
```

```
40
41      public Integer getSex() {
42          return sex;
43      }
44
45      public void setSex(Integer sex) {
46          this.sex = sex;
47      }
48
49      public Integer getAge() {
50          return age;
51      }
52
53      public void setAge(Integer age) {
54          this.age = age;
55      }
56
57      public String getPhone() {
58          return phone;
59      }
60
61      public void setPhone(String phone) {
62          this.phone = phone;
63      }
64  }
```

1.5.3 模型驱动封装

1.5.3.1 概述

> 使用模型驱动封装的步骤如下：
>
> - 第1步：控制器实现 `ModelDriven<T>` 接口
> - 第2步：重写 `getModel()` 方法

1.5.3.2 User

```
1  package com.bdqn.entity;
2
3  public class User {
4      private String userName;
5      private String password;
6      private Integer sex;
7      private Integer age;
8      private String phone;
9
```

```
    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Integer getSex() {
        return sex;
    }

    public void setSex(Integer sex) {
        this.sex = sex;
    }

    public Integer getAge() {
        return age;
    }

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

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public String toString() {
        return "User{" +
                "userName='" + userName + '\'' +
                ", password='" + password + '\'' +
                ", sex=" + sex +
                ", age=" + age +
                ", phone='" + phone + '\'' +
                '}';
    }
}
```

1.5.3.3 控制器Action代码

```java
package com.bdqn.web;

import com.bdqn.entity.User;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

/**
 * 模型驱动封装
 */
public class UserAction3 extends ActionSupport implements ModelDriven<User> {

    //创建用户对象
    private User user = new User();
    @Override
    public User getModel() {
        return user;//返回用户对象
    }

    /**
     * 注册
     * @return
     */
    public String register(){
        System.out.println(user);
        return SUCCESS;
    }

}
```

1.5.3.4 页面代码

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>用户注册</title>
</head>
<body>

<form action="/user/register" method="post">
    <div>
        <label>用户名: </label>
        <input type="text" name="userName">
    </div>
    <div>
        <label>密码: </label>
        <input type="password" name="password">
    </div>
    <div>
        <label>性别: </label>
        <input type="radio" name="sex" value="1">男
        <input type="radio" name="sex" value="2">女
    </div>
    <div>
        <label>年龄: </label>
```

```
        <input type="text" name="age">
    </div>
    <div>
        <label>电话: </label>
        <input type="text" name="phone">
    </div>
    <div>
        <input type="submit" value="注册">
    </div>
</form>

</body>
</html>
```

1.5.3.5 注意

1. input标签的name属性值、提交路径的参数名称必须与实体类的属性名相同(严格区分大小写)，控制器不需要提供模型对象的get、set方法
2. 模型驱动封装与属性封装一起使用时，同名优先使用模型驱动封装，属性封装无法获取数据

1.5.4 表达式封装

1.5.4.1 概述

使用表达式封装的方式获取表单数据，步骤如下：

- 第1步：在控制器中 定义类类型的成员变量 ，提供get、set方法。如：private User user;
- 第2步：页面使用 对象名.实体类属性名 ，如：

```
<input type="text" name="user.userName">
```

1.5.4.2 控制器Action

```
package com.bdqn.web;

import com.bdqn.entity.User;
import com.opensymphony.xwork2.ActionSupport;

public class UserAction4 extends ActionSupport {

```

```
    //表达式封装
    private User user;

    /**
     * 注册
     * @return
     */
    public String register(){
        System.out.println(user);
        return SUCCESS;
    }


    public User getUser() {
        return user;
    }

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

1.5.4.3 页面代码

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>用户注册</title>
</head>
<body>

<form action="/user/register" method="post">
    <div>
        <label>用户名: </label>
        <input type="text" name="user.userName">
    </div>
    <div>
        <label>密码: </label>
        <input type="password" name="user.password">
    </div>
    <div>
        <label>性别: </label>
        <input type="radio" name="user.sex" value="1">男
        <input type="radio" name="user.sex" value="2">女
    </div>
    <div>
        <label>年龄: </label>
        <input type="text" name="user.age">
    </div>
    <div>
        <label>电话: </label>
        <input type="text" name="user.phone">
    </div>
    <div>
        <input type="submit" value="注册">
    </div>
```

```
</form>

</body>
</html>
```

1.5.5 比较表达式封装和模型驱动封装

1.5.5.1 共同点

使用表达式封装和模型驱动封装都可以把数据封装到实体类对象里面

1.5.5.2 不同点

1. 使用模型驱动只能把数据封装到一个实体类对象里面

   注意：在一个action里面不能使用模型驱动把数据封装到不同的实体类对象里面

2. 使用表达式封装可以把数据封装到不同的实体类对象里面