1. 结果页面配置

1. 全局结果页面

1 result标签配置action方法的返回值到不同的路径里面

2 创建两个action，执行默认的方法execute方法，让两个action的方法都返回success，返回success之后，配置到同一个页面里面

（1）如果多个action，方法里面返回值相同的，到页面也是相同的，这个时候可以使用全局结果页面配置

```
<package name="demo1" extends="struts-default" namespace="/">
    <action name="book" class="cn.itcast.action.BookAction">
        <result name="success">/hello.jsp</result>
    </action>
    <action name="orders" class="cn.itcast.action.OrdersAction">
        <result name="success">/hello.jsp</result>
    </action>
</package>
```

（2）在package标签里面配置

```
<!-- 全局结果页面配置 -->
<global-results>
    <result name="success">/hello.jsp</result>
</global-results>
```

2. 局部结果页面

```
<action name="book" class="cn.itcast.action.BookAction">
    <result name="success">/hello.jsp</result>
</action>
```

（1）配置全局页面，也配置了局部页面，最终以局部配置为准

```
<!-- 全局结果页面配置 -->
<global-results>
    <result name="success">/hello.jsp</result>
</global-results>

<action name="book" class="cn.itcast.action.BookAction">
    <result name="success">/world.jsp</result>
</action>
```

2. Result标签的type属性

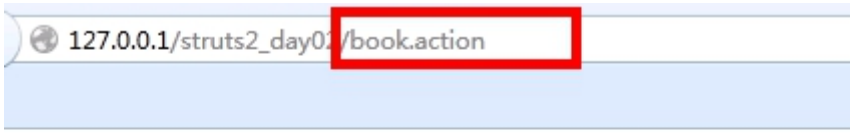
1 result标签里面除了name属性之外，还有一个属性 type属性

（1）type属性：如何到路径里面（转发还是重定向）

2 type属性值

（1）默认值，做转发操作，值是 dispatcher

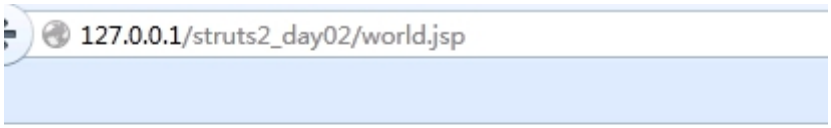
```
<result name="success" type="dispatcher">/hello.jsp</result>
```



orld struts2.....

（2）做重定向操作，值是 redirect

```
<result name="success"  type="redirect">
```



orld struts2.....

（2）上面两个值dispatcher、redirect，这两个值一般针对到页面中配置，配置到其他的action里面

- chain：转发到action，一般不用，缓存问题

```
<result name="success"  type="chain">orders</result>
```

- redirectAction：重定向到action

```
<!-- action访问名称 -->
<result name="success"  type="redirectAction">orders</result>
```

3. Action获取表单提交数据

1 之前web阶段，提交表单到servlet里面，在servlet里面使用request对象里面的方法获取，getParameter，getParameterMap

2 提交表单到action，但是action没有request对象，不能直接使用request对象

3 action获取表单提交数据主要三种方式

（1）使用ActionContext类

（2）使用ServletActionContext类

（3）使用接口注入方式

3-1. 使用ActionContext类获取

Map<String,Object> getParameters()	返回一个包含所有 HttpServletRequest 参数信

（1）因为方法不是静态的方法，需要创建ActionContext类的对象

（2）这个ActionContext类对象不是new出来的

static ActionContext getContext()	获取当前线程的 ActionContext 对象.

1 具体演示

（1）创建表单，提交表单到action里面

（2）在action使用ActionContext获取数据

```
//第一种方式 使用ActionContext类获取
//1 获取ActionContext对象
ActionContext context = ActionContext.getContext();
//2 调用方法得到表单数据
//  key是表单输入项name属性值，value是输入的值
Map<String, Object> map = context.getParameters();

Set<String> keys = map.keySet();
for (String key : keys) {
    //根据key得到value
    //数组形式：因为输入项里面可能有复选框情况
    Object[] obj = (Object[]) map.get(key);
    System.out.println(Arrays.toString(obj));
}
```

3-2. 使用ServletActionContext类获取

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

（1）调用类里面静态方法，得到request对象

```java
public String execute() throws Exception {
    //第一种方式 使用ServletActionContext类获取
    //1 使用ServletActionContext获取request对象
    HttpServletRequest request = ServletActionContext.getRequest();
    //2 调用request里面的方法得到结果
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String address = request.getParameter("address");

    System.out.println(username+":"+password+":"+address);

    return NONE;
}
```

3-3. 使用接口注入（了解）

1 让action实现接口，为了得到request对象

```java
public class Form3DemoAction extends ActionSupport implements ServletRequestAware {

    private HttpServletRequest request ;
    @Override
    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
    }

    public String execute() throws Exception {
        request.getParameter("");
```

3-4. 在action操作域对象

1 request、session、servletContext域对象

2 使用ServletActionContext类操作

```java
//操作三个域对象
//1 request域
HttpServletRequest request = ServletActionContext.getRequest();
request.setAttribute("req", "reqValue");

//2 session域
HttpSession session = request.getSession();
session.setAttribute("sess", "sessValue");

//3 ServletContext域
ServletContext context = ServletActionContext.getServletContext();
context.setAttribute("contextname", "contextValue");
```

4. Struts2封装获取表单数据方式

4-1. 原始方式获取表单封装到实体类对象

```java
public String execute() throws Exception {
    //1 获取表单数据
    HttpServletRequest request = ServletActionContext.getRequest();
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String address = request.getParameter("address");

    //2 封装到实体类对象里面
    User user = new User();
    user.setUsername(username);
    user.setPassword(password);
    user.setAddress(address);

    System.out.println(user);
    return NONE;
}
```

4-2. 属性封装（会用）

1 直接把表单提交属性封装到action的属性里面

2 实现步骤

（1）在action成员变量位置定义变量

- 变量名称和表单输入项的name属性值一样

（2）生成变量的set方法（把set和get方法都写出来）

```java
//1 定义变量
//变量的名称和表单输入项name属性值一样
private String username;
private String password;
private String address;

//2 生成变量的set和get方法
public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getPassword() {
    return password;
}
```

3 使用属性封装获取表单数据到属性里面，不能把数据直接封装到实体类对象里面

4-3. 模型驱动封装（重点）

1 使用模型驱动方式，可以直接把表单数据封装到实体类对象里面

2 实现步骤

（1）action实现接口 ModelDriven

```
class DataDemo2Action extends ActionSupport implements ModelDriven<User>{
```

（2）实现接口里面的方法 getModel方法

- 把创建对象返回

（3）在action里面创建实体类对象

```
//创建对象
private User user = new User();

public User getModel() {
    //返回创建user对象
    return user;
}
```

```
//前提要求：表单输入项name属性值 和 实体类属性名称一样
```

3 使用模型驱动和属性封装注意问题：

（1）在一个action中，获取表单数据可以属性封装，使用模型驱动封装，不能同时使用属性封装和模型驱动封装获取同一个表单数据

如果同时使用，只会执行模型驱动

4-4. 表达式封装（会用）

1 实现过程

（1）使用表达式封装可以把表单数据封装到实体类对象里面

第一步 在action里面声明实体类

第二步 生成实体类变量的set和get方法

```
//1 声明实体类
private User user;
//2 生成实体类变量的set和get方法
public User getUser() {
    return user;
}
public void setUser(User user) {
    this.user = user;
}
```

第三步 在表单输入项的name属性值里面写表达式形式

```
username:<input type="text" name="user.username"/>
<br/>
password:<input type="text" name="user.password"/>
<br/>
address:<input type="text" name="user.address"/>
<br/>
```

2 把表达式封装归类到属性封装里面

4-5. 比较表达式封装和模型驱动封装

1 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

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

- 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

```
//1 声明实体类
private User user;

private Book book;

public Book getBook() {
    return book;
}
public void setBook(Book book) {
    this.book = book;
```

```
username:<input type="text" name="user.username"/>
<br/>
password:<input type="text" name="user.password"/>
<br/>
address:<input type="text" name="user.address"/>
<br/>
bname: <input type="text" name="book.bname"/>
<br/>
```

5. 封装到集合里面

5-1. 封装数据到List集合

第一步 在action声明List

第二步 生成list变量的set和get方法

```
// 1 声明List变量
private List<User> list;
// 2 生成get和set方法
public List<User> getList() {
    return list;
}
public void setList(List<User> list) {
    this.list = list;
}
```

第三步 在表单输入项里面写表达式

```
<!-- list[0] : 表示list集合中第一个user对象 -->
username:<input type="text" name="list[0].username"/>
<br/>
password:<input type="text" name="list[0].password"/>
<br/>
address:<input type="text" name="list[0].address"/>

<br/><br/>

username:<input type="text" name="list[1].username"/>
<br/>
password:<input type="text" name="list[1].password"/>
<br/>
address:<input type="text" name="list[1].address"/>
<br/>
```

5-2. 封装数据到Map集合

第一步 声明map集合

第二步 生成get和set方法

```
// 1 声明map集合
private Map<String,User> map;
public Map<String, User> getMap() {
    return map;
}
public void setMap(Map<String, User> map) {
    this.map = map;
}
```

第三步 在表单输入项的name属性值里面写表达式

```
<!-- 设置key值 map['key值']
    设置value值
 -->
username:<input type="text" name="map['one'].username"/>
<br/>
password:<input type="text" name="map['one'].password"/>
<br/>
address:<input type="text" name="map['one'].address"/>
```