Struts2学习日志二

一、结果页面配置

1.1 全局结果页面配置

```
    <package name="result"  extends="struts-default" namespace="/">
      <!-- 全局结果页面配置 -->
      <global-results>
         <result name="success" type="dispatcher">/hello.jsp</result>
      </global-results>
      <action name="book" class="cn.scct.actions.OrderAction" method="execute">
      </action>
      <action name="order" class="cn.scct.actions.BookAction" method="execute">
      </action>
    </package>
```

没有配置result，将会用`global-results`中的配置当作自己的配置。

1.2 结果跳转方式

1.2.1 转发

```
    <action name="result1" class="cn.scct.resultActions.ResultAction1" method="execute">
        <result name="success" type="dispatcher">/result.jsp</result>
    </action>
```

1.2.2 重定向

```
<!-- 重定向 -->
<action name="result2" class="cn.scct.resultActions.ResultAction2" method="execute" >
    <result name="success" type="redirect" >/result.jsp</result>
</action>
```

1.2.3 转发到Action(基本没意义)

```
<!-- 转发到Action -->
<action name="result3" class="cn.scct.resultActions.ResultAction3" method="execute" >
    <result name="success" type="chain">
            <!-- action的名字 -->
        <param name="actionName">result1</param>
            <!-- action所在的命名空间 -->
        <param name="namespace">/</param>
    </result>
</action>
```

1.2.4 重定向到Action

```
<!-- 重定向到Action -->
<action name="result4" class="cn.scct.resultActions.ResultAction4" method="execute" >
    <result  name="success"  type="redirectAction">
        <!-- action的名字 -->
        <param name="actionName">result1</param>
        <!-- action所在的命名空间 -->
        <param name="namespace">/</param>
    </result>
</action>
```

二、获取ServletAPI方法

在Struts2中，Action并没有直接和Servlet API进行耦合，也就是说Struts2的Action中不能直接访问Servlet API。虽然Struts2中的Action访问Servlet API麻烦一些，但是这却是Struts2中的重要改良之一，方便Action进行单元测试。
尽管Action和Servlet API解锁会带来很多好处，然而在Action中完全不访问Servlet时不可能的，在实现业务逻辑时，经常要访问Servlet中的对象，如session、request和application等。

2.1 通过ActionContext类访问(最常用)

ActionContext	Map	数据中心
原生request	HttpServletRequest	
原生response	HttpServletResponse	
原生ServletContext	ServletContext	
request域	Map	
session域	Map	
application域	Map	
param 参数	Map	
attr 域	Map 3个域合一	
ValueStack	值栈	
.....		

生命周期：每次请求时都会创建一个与请求对应的ActionContext对象.请求处理完ActionContext销毁.

如何获得ActionContext. struts2设计的是,将ActionContext对象创建好之后.将ActionContext与当前线程绑定.我们要获得ActionContext.只需要从ThreadLocal中获取即可.

显然ActionContext域的生命周期和request域一样。

获取ActionContext域对象

由于ActionContext对象与当前线程是绑定的，所以获取的方法必然是静态的。`ActionContext.getContext()`

```java
ActionContext context = ActionContext.getContext();
//request域=> map (struts2并不推荐使用原生request域)
//不推荐
Map<String, Object> requestScope = (Map<String, Object>) context.get("request");
//推荐
context.put("name", "requestTom");

//session域 => map
Map<String, Object> sessionScope = context.getSession();
sessionScope.put("name", "sessionTom");

//application域=>map
Map<String, Object> applicationScope = context.getApplication();
applicationScope.put("name", "applicationTom");
```

//静态方法获取context对象

//相当于setAttribute方法

ActionContext 类访问 Servlet API 的常用方法

方法声明	功能描述
void put(String key, Object value)	将 key-value 键值对放入 ActionContext 中，模拟 Servlet API 中的 HttpServletRequest 的 setAttribute()方法。
Object get(String key)	通过参数 key 来查找当前 ActionContext 中的值。
Map<String,Object> getApplication()	返回一个 Application 级的 Map 对象。
static ActionContext getContext()	获取当前线程的 ActionContext 对象
Map<String,Object> getParameters()	返回一个包含所有 HttpServletRequest 参数信息的 Map 对象。
Map<String,Object> getSession()	返回一个 Map 类型的 HttpSession 对象。
void setApplication(Map<String,Object> application)	设置 Application 上下文。
void setSession(Map<String,Object> session)	设置一个 Map 类型的 Session 值。

2.2 通过ServletActionContext类访问

```java
//如何在action中获得原生ServletAPI
public class Demo6Action extends ActionSupport {
    //并不推荐
    public String execute() throws Exception {
        //原生request
        HttpServletRequest request = ServletActionContext.getRequest();
        //原生session
        HttpSession session = request.getSession();
        //原生response
        HttpServletResponse response = ServletActionContext.getResponse();
        //原生servletContext
        ServletContext servletContext = ServletActionContext.getServletContext();
        return SUCCESS;
    }
}
```

原味生的

2.3 通过特定的接口访问

为了在Action中直接访问ServletAPI，Struts2还提供了一系列的接口：

ServletRequestAware：实现该接口的 Action 可以直接访问 Web 应用的 HttpServletRequest 实例。

ServletResponseAware：实现该接口的 Action 可以直接访问 Web 应用的 HttpServletResponse 实例。

SessionAware：实现该接口的 Action 可以直接访问 Web 应用的 HttpSession 实例。

ServletContextAware：实现该接口的 Action 可以直接访问 Web 应用的 ServletContext 实例。

```java
//如何在action中获得原生ServletAPI
public class Demo7Action extends ActionSupport implements ServletRequestAware {
    private HttpServletRequest request;
    public String execute() throws Exception {
        System.out.println("原生request:"+request);
        return SUCCESS;
    }
    @Override
    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
    }
}
```

三、从页面获取参数(参数封装)

```html
<body>
    <form action="${pageContext.request.contextPath }/form5.action" method="post">
      username:<input type="text" name="username"/>
      <br/>
      password:<input type="password" name="password"/>
      <br/>
      address:<input type="text" name="address"/>
      <br/>
      <input type="submit" value="提交">
</body>
```

3.1 原始方式封装

```java
    //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.封装到实体类对象中
    User user=new User();
    user.setUsername(username);
    user.setPassword(password);
    user.setAddress(address);
```

3.2 属性封装

其实只需要提供set方法，并不需要提供get方法。

```java
/*
 * 属性封装表单数据
 * */
//1.定义变量
//变量名称和表单输入项name属性值一样
//直接作为Action类的成员变量
//生成get和set方法
private String username;
private String password;
private String address;
public String getUsername() {
    return username;
}

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

属性封装

3.3 模型驱动封装(用的多)

```java
public class FormPackAction3 extends ActionSupport  implements ModelDriven<User>{
    //模型驱动封装
    //创建对象
    //前提要求：表单输入项的name属性值和实体类属性名称一样
    //不能同时使用属性封装和模型驱动封装
    private User user=new User();
    public String execute() throws Exception {
        System.out.println(user);
        return NONE;
    }
    public User getModel() {
        return user;
    }
}
```

第一步

第二步

但是只能封装一个对象

第三步

3.4 对象封装(也称为第二类属性封装)

其实是页面提供OGNL表达式进行封装

```html
<form action="${pageContext.request.contextPath }/data4.action" method="post">
  username:<input type="text" name="user.username"/>
  <br/>
  password:<input type="password" name="user.password"/>
  <br/>
  address:<input type="text" name="user.address"/>
  <br/>
  bookname:<input type="text" name="book.bname"/>
  <br/>
  <input type="submit" value="提交">
```

其实是OGNL表达式

```java
public class FormPackAction4 extends ActionSupport {
    //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;
    }

    public String execute() throws Exception {
        System.out.println(user);
        return NONE;
    }
}
```

3.5 封装数据到list集合

```jsp
    <form     action="${     pageContext.request.contextPath     }/strutsDemo4.action" method="post">
        名称:<input type="text" name="list[0].name"><br/>
        年龄:<input type="text" name="list[0].age"><br/>
        生日:<input type="text" name="list[0].birthday"><br/>
        名称:<input type="text" name="list[1].name"><br/>
        年龄:<input type="text" name="list[1].age"><br/>
        生日:<input type="text" name="list[1].birthday"><br/>
        <input type="submit" value="提交">
    </form>
```

list集合的多个对象及其属性

```java
public class StrutsDemo4 extends ActionSupport{
    private List<User> list;

    public List<User> getList() {
        return list;
    }

    public void setList(List<User> list) {
        this.list = list;
    }

    @Override
    public String execute() throws Exception {
        for (User user : list) {
            System.out.println(user);
        }
        return NONE;
    }
}
```

3.6 封装数据到Map集合

```
    <h1>批量插入用户:封装到 Map 集合</h1>
    <form     action="${     pageContext.request.contextPath     }/strutsDemo5.action"
method="post">
        名称:<input type="text" name="map['one'].name"><br/>
        年龄:<input type="text" name="map['one'].age"><br/>
        生日:<input type="text" name="map['one'].birthday"><br/>
        名称:<input type="text" name="map['two'].name"><br/>
        年龄:<input type="text" name="map['two'].age"><br/>
        生日:<input type="text" name="map['two'].birthday"><br/>
        <input type="submit" value="提交">
    </form>
```

```
public class StrutsDemo5 extends ActionSupport {
     private Map<String,User> map;
    public Map<String, User> getMap() {
        return map;
    }

    public void setMap(Map<String, User> map) {
        this.map = map;
    }

    @Override
    public String execute() throws Exception {
        for (String key : map.keySet()) {
            User user = map.get(key);
            System.out.println(key+"    "+user);
        }
        return NONE;
    }
}
```