轻量级 J2EE 框架应用

E 4　A Simple Controller with handling result view

学号：SA17225052

姓名：戴赛

报告撰写时间：2017/12/19

1.主题概述

主要任务是根据 result 中的 value 的结果找到对应的 xml 文件,并根据 xml 文件动态的生成 HTML 视图.

不过在作业中并没有使用 XSLT,而是通过 DOM4J 读取 XML 文件,并根据 XML 中的内容手动实现了生成 HTML 时的逻辑.

主要思路:

在 SimpleController 中开始执行 result 时总会读取到 value 值,此时执行一次判断:

若 value 中包含”_view.xml”则执行 doResultView()方法,找到 UseSC 下对应的 XML 文件,

依据该文件的内容生成相应的 HTML 文件,

并将 result 的 value 指向该 HTML 文件

再正常执行重定向或者转发的操作

2.假设

上次作业 E3
DOM4J

3.实现或证明

1.

定义 success_view.xml 视图

```
<?xml version="1.0" encoding="UTF-8" ?>
<view>
    <header>
        <title>a viewer in MVC</title>
    </header>
    <body>
        <form>
            <name>logoutForm</name>
            <action>logout.action</action>
            <method>post</method>
            <textView>
                <name>userName</name>
                <label>Login Name:</label>
                <value>Water</value>
            </textView>
            <textView>
                <name>userAge</name>
                <label>Age</label>
                <value>30</value>
            </textView>
            <buttonView>
                <name>logoutButton</name>
                <label>Logout</label>
            </buttonView>
        </form>
    </body>
</view>
```

2

修改 controller.xml

```
ontroller>
  <action name="loginAction.sc" class="water.ustc.action.LoginAction" method="handleLogin">
      <interceptro-ref name="log"></interceptro-ref>
      <result name="success" type="forward" value="/success_view.xml"></result>
      <result name="failure" type="redirect" value="/failure.jsp"></result>
  </action>
  <action name="registerAction.sc" class="water.ustc.action.RegisterAction" method="handleR
      <interceptro-ref name="log"></interceptro-ref>
```

3.
针对 result 的 value 是”*_view.xml”的情况动态生成 view 视图

3.1 判断 value 的内容,若包含*_view.xml.则执行 doResultView()方法,其结果是返回对应 value 的 html 文件名,将其重新赋值给原 value

```
String resultValue = result.attributeValue( s: "value");

// 判断一波resultvalue的值
if (resultValue.contains("_view.xml"))
{
    System.out.println("         result调用文件是视图文件:");
    System.out.println("             视图文件xml文件名是:"+resultValue+",,,,,,读取xml文件");

    // 生成html文件
    resultValue = doResultView(resultValue)+".html";

    System.out.println("视图HTML:"+resultValue);
}else System.out.println("          result调用文件是普通文件");
System.out.println("其value为:"+resultValue);
```

3.2 在 doResultView()中,新建 StringBuilder 对象,将 xml 中的对应内容存放在 StringBuilder 中,再用生成 web 目录下的 HTML 文件

```
private String doResultView(String resultValue)
{
    /*
    *  输入 原result *_view.xml
    *  返回 结果关键字
    * */
    String newResultValue = resultValue.split( regex: "_")[0]; // 提取关键字,这里是"success"
    System.out.println(newResultValue);
    try {
        System.out.println("**********ViewXML**********");

        // 创建HTML文件与StringBuilder
        StringBuilder sb = new StringBuilder();
        PrintStream printStream = null ;
        try {
            printStream= new PrintStream(new FileOutputStream( name: "F:\\J2EE\\UseSc\\web\\"+newResultValue+".html"));//路径默认在项目根目录下
            // F:\J2EE\UseSc\web\failure.jsp
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
```

3.3 利用 DOM4J 读取 xml,读取搭配 title 节点后,根据 title 内容动态生成对应 HTML 标题,将内容放入 StringBuilder 中

```
        InputStream inputStream = this.getClass().getResourceAsStream(resultValue); // 找到对应xml文件
        SAXReader saxReader = new SAXReader();
        Document document = saxReader.read(inputStream);

        // 获得(0)根节点,<view>
        Element rootElement = document.getRootElement();
        System.out.println("根节点名称:"+rootElement.getName()+",其内容为:"+rootElement.getTextTrim());

        /* HTML标题部分,无列表 */
        // 获得(1)层节点,<header>
        Element header = rootElement.element("header");
        System.out.println("    标题部分(1)层节点名称为"+header.getName());
        // System.out.println("    其属性个数为:"+header.attributeCount());
        // 获得(2)层节点,<title>
        Element title = header.element("title");
        String titleTextTrim = title.getTextTrim();
        System.out.println("        标题部分(2)层节点名称为"+title.getName());
        System.out.println("        其文本内容为"+titleTextTrim); // 标题实际内容
        // 这里可以输出HTML的标题部分
        sb.append("<!DOCTYPE html>\n");
        sb.append("<html lang=\"en\">\n");
        sb.append("<head>\n");
        sb.append("    <meta charset=\"UTF-8\">\n");
        sb.append("<title>"+titleTextTrim+"</title>\n");
        sb.append("</head>\n");
        sb.append("<body>\n");
```

3.4 读取到 body 节点后,动态生成 HTML 内容.
虽然这里只有内容只有一个表单 form,但是实际运用中可能会有多个内容.所以这里读取

内容时使用的是 LIst.遍历 List 内容,再利用 switch 根据情况实现相应逻辑
这里是构建表单

```
/* HTML内容部分,其子节点会有列表 */
// 获得(1)层节点,<body>
Element body = rootElement.element("body");
System.out.println("    内容部分(1)层节点名称为"+body.getName()); // 这层不需要列表
List<Element> bodyChildElementsList = body.elements(); // HTML页面内容有多种可能
for (Element bodyChild:bodyChildElementsList)
{
    switch (bodyChild.getName())
    {
        case "form":
        {
            // 获得(2)层节点,<form>
            Element form = body.element("form");
            System.out.println("        内容部分(2)层节点名称为"+form.getName()); // 这层也无列表,但是下一层有
            // 获得(3)层节点,完全遍历
            List<Element> formAttributesAndViewsList = form.elements(); // 获得表单的属性

            // <form name="logoutForm" action="logout.action" method="post">
            String formName = form.element("name").getTextTrim();
            String formAction = form.element("action").getTextTrim();
            String formMethod = form.element("method").getTextTrim();
            // System.out.println("<form name=\""+formName+"\" action=\""+formAction+"\" method=\""+formMethod+"\">");
            sb.append("<form name=\""+formName+"\" action=\""+formAction+"\" method=\""+formMethod+"\">\n");

            for(Element formAV:formAttributesAndViewsList)
```

3.5 思路同 3.4 遍历 List,利用 switch 根据情况生成表单中的具体内容,这里是 textView 和 buttonView

```
// System.out.println("<form name=\""+formName+"\" action=\""+formAction+"\" method=\""+formMethod+"\">");
sb.append("<form name=\""+formName+"\" action=\""+formAction+"\" method=\""+formMethod+"\">\n");

for(Element formAV:formAttributesAndViewsList)
{
    // 这里可以输出
    System.out.println("            内容部分(3)层节点名称"+formAV.getName());
    switch (formAV.getName())
    {
        case "textView":
        {
            String avValue = formAV.element("value").getTextTrim();
            String avName = formAV.element("name").getTextTrim();
            String avLabel = formAV.element("label").getTextTrim();
            // System.out.println(avValue);
            sb.append("  <input type=\"text\" name=\""+avName+"\" value=\""+avValue+"\" aria-label=\""+avLabel+"\">\n");
            System.out.println("写入内容为:");
            System.out.println("  <input type=\"text\" name=\""+avName+" value="+avValue+" aria-label="+avLabel+"\">\n");
            break;
        }
        case "buttonView":
        {
            String avName = formAV.element("name").getTextTrim();
            String avLabel = formAV.element("label").getTextTrim();
            sb.append("  <input type=\"button\" name=\""+avName+"\" aria-label=\""+avLabel+"\">\n") ;
            System.out.println("写入内容为:");
            System.out.println("  <input type=\"button\" name=\""+avName+"\" aria-label=\""+avLabel+"\">\n");
            break;
        }
    }

}
sb.append("</form>\n");
```

3.6

将 StringBuilder 写入 html 文件

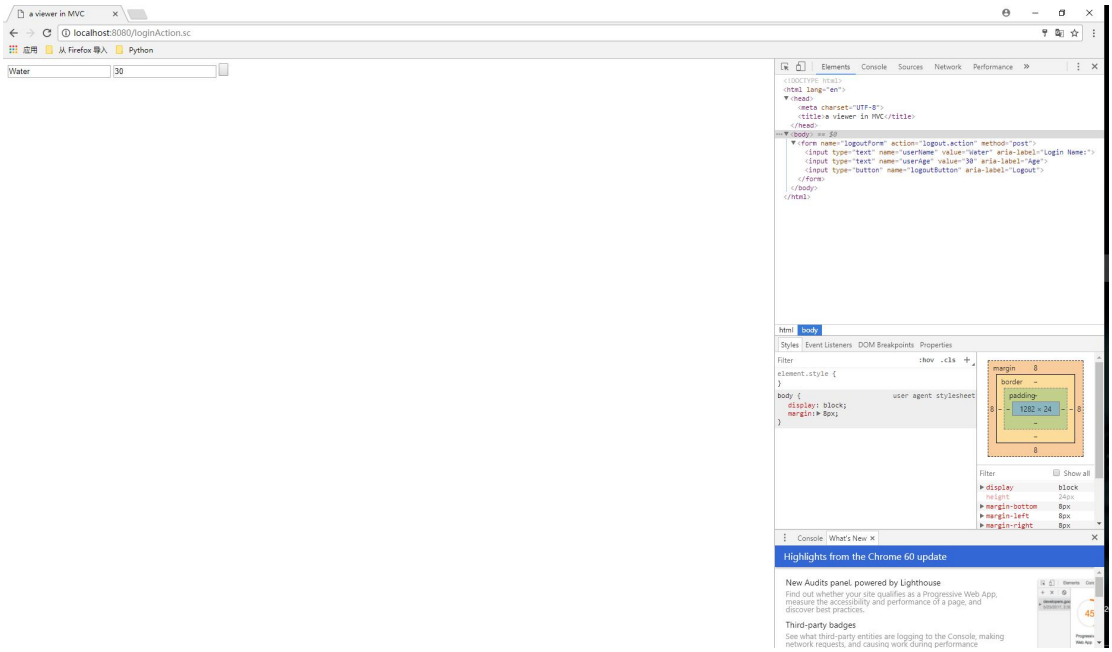
```
                }
                sb.append("</form>\n");
                break;
            }
        }
    }

    sb.append("</body>\n");
    sb.append("</html>\n");
    printStream.println(sb.toString());
    System.out.println("**********ViewXML**********");
} catch (DocumentException e) {
    e.printStackTrace();
}
return newResultValue;
```

生成的 HTML

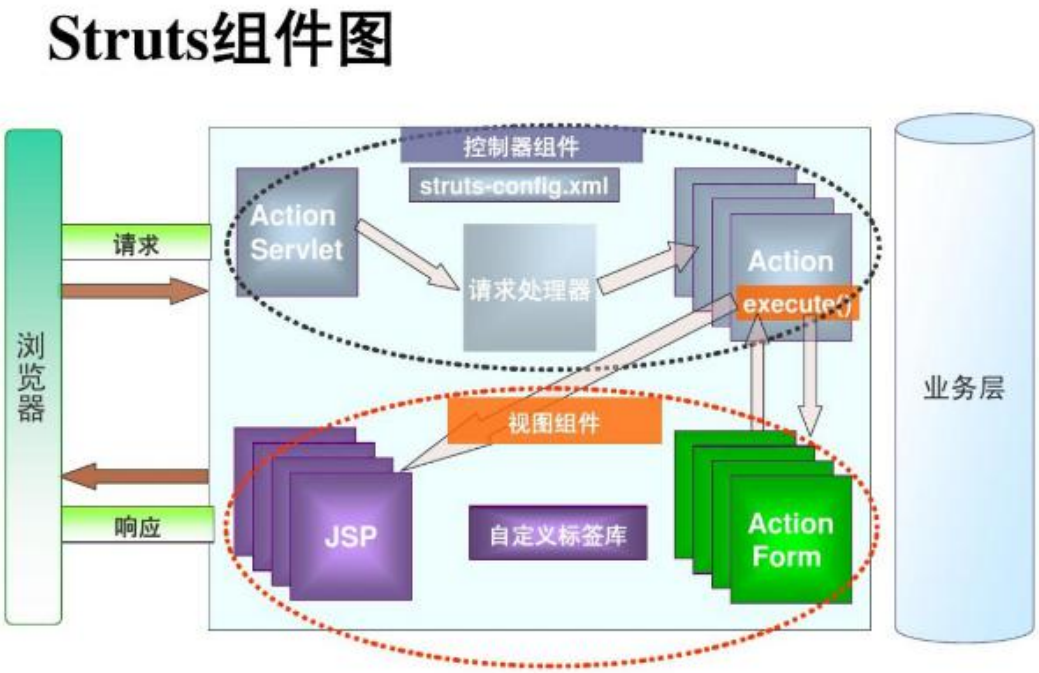
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
<title>a viewer in MVC</title>
</head>
<body>
<form name="logoutForm" action="logout.action" method="post">
  <input type="text" name="userName" value="Water" aria-label="Login Name:">
  <input type="text" name="userAge" value="30" aria-label="Age">
  <input type="button" name="logoutButton" aria-label="Logout">
</form>
</body>
</html>
```

运行 APP,登陆成功,查看页面源码



5.

问:描述 Structs 框架中视图组件的工作方式,如<s:form>/<s:submit>

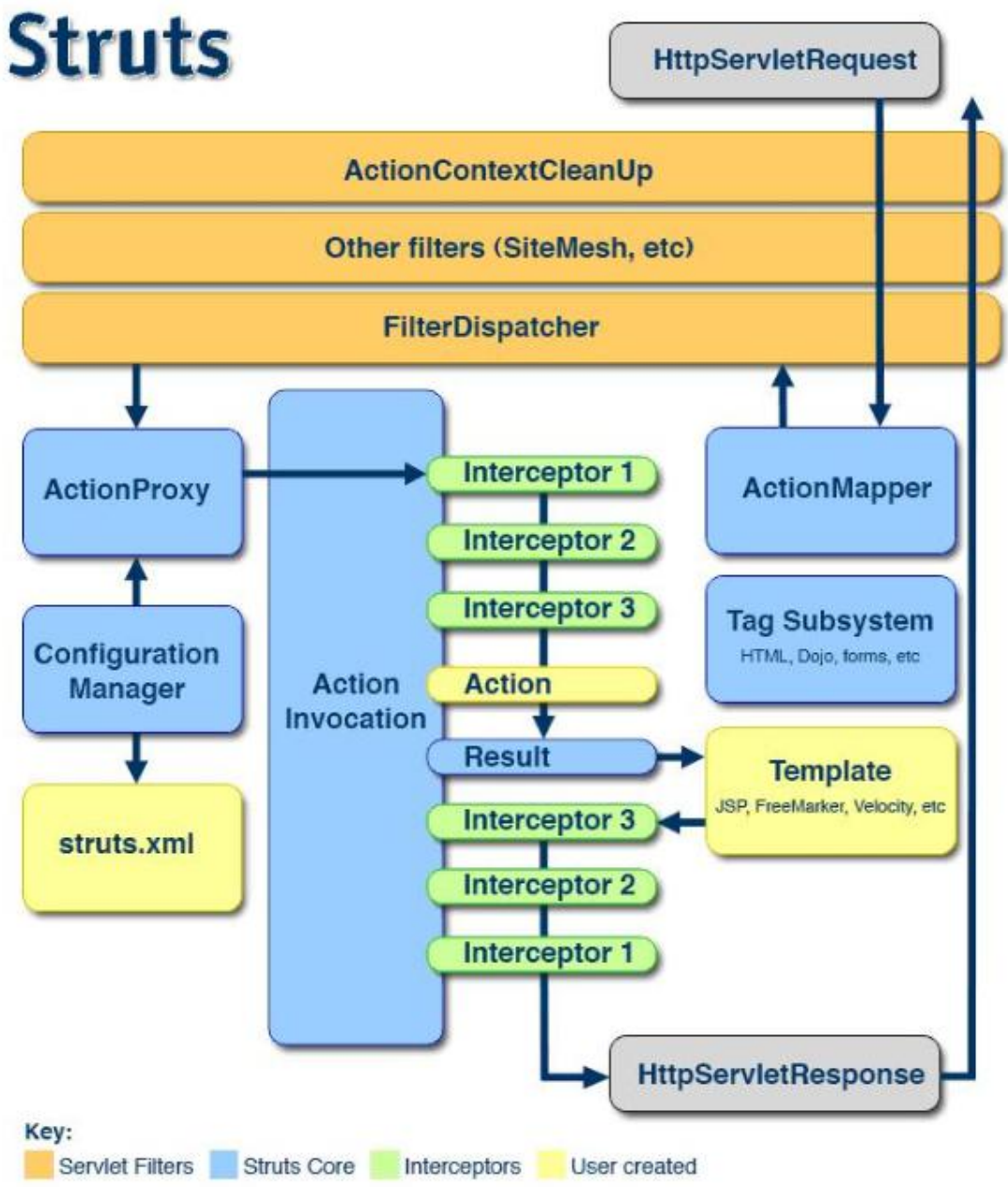


由上图,可知,Structs 视图组件的主要职责为接收控制器组件的数据请求,显示数据模型同时将输入的数据或请求传给控制器组件.

Structs 视图组件为分为三类,分别为自定义标签库,JSP,以及 ActionForm.

Struct1 只能支持 JSP 作为视图资源,虽然也可以使用自定义的标签库,但是配置比较复杂,学习成本高,且已经过时.

而 Struct2 中的视图组件已经不仅仅是这三类了,Struct2 的进步之处就是可以使用其他视图技术.具体如下



一旦 Action 执行完毕,ActionInvocation 负责根据 struct.xml 中的配置找到对应的返回结果.返回结果通是(但不总是,也可能是另外的一个 Action 链)一个需要被表示的 JSP 或者 FreeMarker 的模板.在表示的过程中可以使用 Structs2 框架中继承的标签.在这个过程中需要涉及到 ActionMapper.

在 struct.xml 配置文件中,每一个 Action 定义都有 name 和 class 属性,同时还要指定 result 元素.result 元素制定了逻辑视图名称和实际视图的对应关系.每个 result 都有一个 type 属性.

关于标签:

Struct2 中将所有的标签都定义在一个 s 标签库里,比如:<s:form>/<s:submit>,使用标签需要先导入标签库,代码为:<%@taglib prefix="s" uri="/struts-tags"%>其中 uri 就是 struct2 标签库的 URL,而 prefix 属性值是该标签库的前缀.这些标签或用于生成 html 元素

标签,或用于数据访问,控制逻辑.在数据上,可以使用 OGNL 表达式语言.

对主题内容进行实验实现，或例举证明，需描述实现过程及数据。如对 MVC 中 Controller 功能的实现及例证（图示、数据、代码等）

4.结论

对主题的总结，结果评论，发现的问题，或你的建议和看法。如 MVC 中 Controller 优点与缺点，个人看法（文字、图标、代码辅助等）

5.参考文献

以上内容的理论知识点或技术点如果参考了网上或印刷制品，请在这里罗列出来