轻量级 J2EE 框架应用

E 2　A Simple Controller Based on Configuration File

学号：SA18225433

姓名：杨帆

报告撰写时间：2018/11/28

1.主题概述

1、 Action。

Strut2 中的 Action 是一种与 Servlet 类似的处理请求的 java 类，当接受一个请求，通过配置文件信息找到指定的 Action 类执行，Action 类执行完毕后会返回一个字符串，这个字符串与配置文件的 Action 里的 result 标签的 name 属性相对应，并且通过指定的方式跳转到对应的 jsp 视图中去。

2、 Xml 解析。

Java 中解析 xml 文件分为四种方式，包括 DOM 方式、SAX 方式、JDOM 方式、DOM4j 方式，DOM 方式是以层次结构组织的节点或信息片断的集合。这个层次结构允许开发人员在树中寻找特定信息。分析该结构通常需要加载整个文档和构造层次结构；SAX 方式使用 SAX 分析器对 XML 文档进行分析，分析同时会触发一系列事件，并激活相应的事件处理函数，应用程序通过这些事件处理函数实现对 XML 文档的访问；JDOM 方式基于树形结构，利用纯 java 的技术对 XML 文档实现解析，生成，序列化以及多种操作。它是直接为 java 编程服务，利用 java 语言的特性（方法重载，集合），把 SAX 和 DOM 的功能结合起来，尽可能的把原来解析 xml 变得简单，DOM4j 方式是一个易用的，开源的库，用于 XML,XPath,XSLT。它应用于 Java 平台，采用了 Java 集合框架并完全支持 DOM,SAX,JAXP。它提供了大量的接口，因此比 JDOM 更具有灵活性。

3、 Java 反射机制。

反射指的是在运行状态下，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为 java 语言的反射机制。首先介绍 Class 对象，每个类都有自己的 Class 对象，虚拟机在 class 文件加载阶段时，Class 对象就被创建了，这个对象保存在类同名的.class 文件里，在这个字节码文件被加载到内存后，即 Class 对象也就被载入到内存中了，因此可以获得这个 Class 对象，如何通过反射里得常用方法去获得类对象，类得构造方法，类的成员变量，类得成员方法等信息。

4、 描述你对 Strut2 控制器的理解，并参考资料，比较基于配置的控制器和注解的控制器各自的优缺点。

strut2 处理请求的流程理解，服务端接受一个 http 请求，这个请求经过一系列过滤器，其中会经过一个叫做 ActionContextCleanUp 的可选过滤器，用来对 strut2 等框架的集成提供帮助，后面会调用 FilterDispatcher，用来询问 ActionMapper 来决定这个请求是否需要调用某个 Action，在 FilterDispatcher 中会调用 doFilter 方法，对 req 和 resp 对象做封装，得到 ActionMapper 对象，调用 serviceAction 方法，该方法判断调用哪个 Action，并把请求处理交给 ActionProxy 对象，ActionProxy 通过 ConfigurationManager 询问框架的配置文件，找到需要调用的 Action 类，ActionProxy 会创建 ActionInvocation 对象来处理 Action 调用前后拦截器的的调用，当 Action 调用完毕，ActionInvocation 负责从 struts.xml 中的配置找到对应的返回结果。返回结果通常是 JSP 或其他模板文件。

使用 xml 配置文件的控制器是一直以来的一种规范，优点是配置比较集中、方便管

理，部分技术是其他技术配置无法替代的，缺点是配置冗长不简洁，，但是注解方式因为简洁的，简化开发的优点成为配置上的一种趋势，但缺点是配置嵌入到各个文件中，配置分散不容易管理。

2.假设

本次作业能够实现使用 IDEA 实现一个简单的模拟 Action 护理请求的过程，Servlet 监听请求，当接受到一个能够匹配的请求，获得 actionName 与配置文件中的 Action 标签项进行匹配，若成功则调用 Action 类对应的方法并返回一个字符串，与 Action 的子 result 标签进行匹配，若成功则跳转到 JSP 等模板页面中。

3.实现或证明

1. **实现成果**

   e2：https://github.com/saaaaaail/J2eee2

2. **将 E1 中的控制器修改为基于配置文件的控制器。**

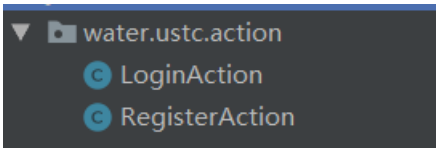
   2.1 将 E1 中的工程 UseSC 配置 web.xml 中定义的 servlet-mapping 修改为：可以拦截“*.sc”类型的请求，如 Http：//host/request_action.sc

   修改 url-pattern 标签，可以拦截“*.sc”标签，

```
<servlet-mapping>
    <servlet-name>sc</servlet-name>
    <url-pattern>*.sc</url-pattern>
</servlet-mapping>
```

   2.2 在 UseSC 工程中新建源码包 water.ustc.action，在该包下声名 POJO 类 LoginAction、RegisterAction 等，每个 POJO 类可以有不同功能的方法，所有方法均返回 String 类型的结果。针对 POJO 类方法的返回值，分别自定义对应的视图 jsp 或 html。

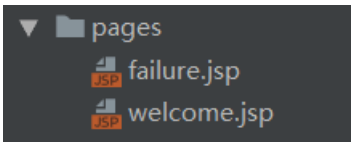
   创建 LoginAction 和 RegisterAction 类并按照下面的 controller.xml 文件，分别创建 handleLogin 方法，并返回”success”、”failure”字符串，创建 handleRegister 方法，并返回”success”字符串，




```
public class LoginAction {
    public String handleLogin() {
        System.out.println("执行handleLogin...");
        int count = (int)(Math.random()*10);
        if(count < 4){
            return "failure";
        }else{
            return "success";
        }
    }
}
```

```
public class RegisterAction {
    public String handleRegister(){
        System.out.println("执行handleRegister...");
        return "success";
    }
}
```

   并按照 controller.xml 文件配置 pages 目录，并定义 welcome.jsp 文件和 failure.jsp 文件，



failure.jsp

```
<%@ page language="java"
    pageEncoding="UTF-8"%
<!DOCTYPE html PUBLIC "-/
<html>
<head>
<meta http-equiv="Content
<title>failure</title>
</head>
<body>
失败
</body>
</html>
```

welcome.jsp

```
<%@ page language="java"
    pageEncoding="UTF-8"%
<!DOCTYPE html PUBLIC "-/
<html>
<head>
<meta http-equiv="Content
<title>success</title>
</head>
<body>
成功
</body>
</html>
```

2.3 在 UseSC 工程中 src 下新建 controller.xml，在其中配置若干<action>与<result>。示例如下：

```
<sc-configuration>

    <controller>
       <action name="login" class="water.ustc.action.LoginAction" method="handleLogin">
          <result name="success" type="forward" value="pages/welcome.jsp"></result>
          <result name="failure" type="redirect" value="pages/failure.jsp"></result>
          <!-- some results -->
       </action>
        <action name="register" class="water.ustc.action.RegisterAction" method="handleRegister">
          <result name="success" type="forward" value="pages/welcome.jsp"></result>
          <!-- some results -->
       </action>
       <!-- some actions -->
    </controller>

</sc-configuration>
```

按照上图要求，在 resourses 目录下创建 controller.xml 文件，并配置若干<action>与<result>标签，

controller.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<sc-configuration>
    <controller>
        <action name="login" class="water.ustc.action.LoginAction" method="handleLogin">
            <result name="success" type="forward" value="pages/welcome.jsp"></result>
            <result name="failure" type="redirect" value="pages/failure.jsp"></result>
        </action>
        <action name="register" class="water.ustc.action.RegisterAction" method="handleRegister">
            <result name="success" type="forward" value="pages/welcome.jsp"></result>
        </action>
    </controller>
</sc-configuration>
```

2.4 修改 SimpleController 工程的类 SimpleController 源码， 当一个 http request 请求访问 web container 资源时，由 doPost() 对请求进行处理：获取请

求（action）的名称。

当一个 http 请求访问 Servlet 容器时，在 Servlet 中通过 request 对象获得请求的 uri，并截取最后的/action.sc 中 action 名称，

```
private String getActionName(HttpServletRequest request) {
    String uri = request.getRequestURI();
    return uri.substring(uri.lastIndexOf( str: "/") + 1, uri.indexOf(".sc"));
}
```

2.5 SimpleController 获取请求 action 名称后，解析使用 simple-controller.jar 库的工程（对于当前练习即为 UseSC 工程） 配置文件 controller.xml（XML 解析，SAX、Dom 或其他），查找对应 name 的 action。如果在 controller.xml 中找到，则解析该 action 的配置。如果没有找到，响应客户端信息为：不可识别的 action 请求。

首先构建 xml 解析的工具类 XmlUtil，构建解析 xml 文件的方法，使用 DOM 来解析，首先获取 controller.xml 文件的所有 action 节点的 list，遍历所有 action 节点，获得每个节点所有属性，使用节点的 name 属性与请求的 antionName 进行比较，若相等则 action 匹配成功，

```
public String analyzeAction(String file, String actionName) {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(new File(file));
        NodeList actions = doc.getElementsByTagName("action");
        int actionsLength = actions.getLength();

        for (int i = 0; i < actionsLength; i++) {
            Node actionNode = actions.item(i);
            //获取Node节点所有属性值
            NamedNodeMap actionNodeMap = actionNode.getAttributes();
            String nameString = actionNodeMap.getNamedItem("name").getNodeValue();
            String methodString = actionNodeMap.getNamedItem("method").getNodeValue();
            String classString = actionNodeMap.getNamedItem("class").getNodeValue();
            String resultString;
            if (nameString.equals(actionName)) {
```

若 action 没有匹配，则返回”action:failure”字符串，

```
//没有对应的action
return "action:failure";
```

2.6 SimpleController 查找到 http request 请求的 action 后， 利用其 class 属性实例化所指向的类（ Java 反射机制，Reflection），并执行指定的 method 方法。

在解析 controller.xml 与 action 匹配成功后，获得了类地址 classString，类中某方法名 methodString，则使用 Class.forName 方法获得类的 Class 对象，调用 getMethod 方法获得 method 对象，调用方法的 invoke 方法执行该类方法，即执行了该 Action 方法，并获得了返回结果字符串保存到 resultString 中，

```
//使用反射调用目标方法，获取返回结果
Class clazz = Class.forName(classString);
Method method = clazz.getMethod(methodString);
Object obj = method.invoke(clazz.newInstance());
resultString = (String)obj;
```

2.7 method 方法执行完毕后，返回字符串作为请求结果。SimpleController 根据请求结果，查找此 action 中<result>结点的 name 属性，若找到，将 value 指向的资源按 type 所定义的方式返回到客户端。如果没有匹配的<result>，响应客户端为信息为：没有请求的资源。

调用了 Action 方法后，返回了执行结果的字符串 resultString，然后获得 Action 标签的 result 子标签 list，遍历子标签，保存 result 标签的 name、type、value 属性，使用 result 标签的 name 属性与 resultString 比较，若匹配，则返回 type 属性与 value 属性组成的字符串，

```
for (int j = 0; j < actionChildNodes.getLength(); j++) {
    if (actionChildNodes.item(j).getNodeType() == Node.ELEMENT_NODE) {
        if (actionChildNodes.item(j).getNodeName().toString().equals("result")) {
            NamedNodeMap resultMap = actionChildNodes.item(j).getAttributes();
            String resultName = resultMap.getNamedItem("name").getNodeValue();
            String resultType = resultMap.getNamedItem("type").getNodeValue();
            String resultValue = resultMap.getNamedItem("value").getNodeValue();

            if (resultName.equals(resultString)) {
                System.out.println(resultType + "," + resultValue);
                return resultType + "," + resultValue;
            }
        }
    }
}
```

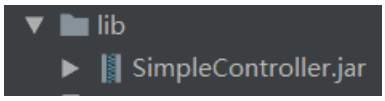
若没有匹配的 result name 属性，则返回”result:failure”字符串，

```
//result不匹配
return "result:failure";
```

至此 controller 解析完成，不论解析结果如何均返回了一个字符串，根据返回结果进行跳转操作，由于 Xml 解析方法中没有 request 和 response 对象，因此只完成返回字符串操作，在 SimpleController 类中接受 result 字符串，并解析是哪一种字符串，并执行相应的打印输出与跳转操作，

```
String[] tmp = result.split( regex: ",");
switch (tmp[0]){
    case "forward": out.println(tmp[1]);req.getRequestDispatcher(tmp[1]).forward(req,resp);break;
    case "redirect": out.println(tmp[1]);resp.sendRedirect(tmp[1]);break;
    case "action:failure": out.println("不可识别的action请求");break;
    case "result:failure": out.println("没有请求的资源");break;
}
```

2.8 为了使得 SimpleController 类的代码简洁，建议自主添加工具类或辅助类完成以上功能。重新打包导出 SimpleController 工程为 simple-controller.jar，并将该 jar 添加为 UseSC 的 lib 库。



2.9 将 UseSC 部署在 tomcat 中测试，验证当浏览器请求对应的 action 时，是否能够响应正确的视图。如果有错，调试程序直到输出期望结果

请求 http://localhost:8080/login.sc

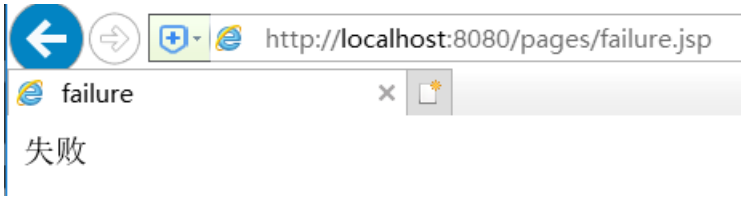
成功

返回 success 字符串

控制台依次打印：（actionName、获得 controller.xml 文件地址、执行 Action 对应方法、result 字符串返回结果）

```
login
controller.xml: /G:/IntelliJIDEA/workspaces_web/UseSC/out/artifacts/UseSC_war_exploded/WEB-INF/classes/controller.xml
执行handleLogin...
forward,pages/welcome.jsp
```

请求 http://localhost:8080/login.sc



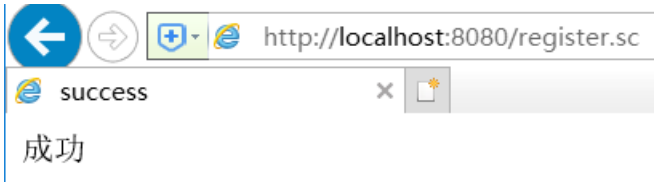
失败

当返回 failure 字符串，进行请求重定向，请求地址发生了变化，

控制台依次打印：（actionName、获得 controller.xml 文件地址、执行 Action 对应方法、result 字符串返回结果）

```
login
controller.xml: /G:/IntelliJIDEA/workspaces_web/UseSC/out/artifacts/UseSC_war_exploded/WEB-INF/classes/controller.xml
执行handleLogin...
redirect,pages/failure.jsp
```

请求 http://localhost:8080/register.sc



成功

返回 success 字符串

控制台依次打印：（actionName、获得 controller.xml 文件地址、执行 Action 对应方法、result 字符串返回结果）

```
register
controller.xml: /G:/IntelliJIDEA/workspaces_web/UseSC/out/artifacts/UseSC_war_exploded/WEB-INF/classes/controller.xml
执行handleRegister...
forward,pages/welcome.jsp
```

5.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。

本次作业学习了什么是 Action，了解了 action 与 servlet 的区别，学习了 servlet 在 web.xml 文件路径配置问题，了解了 java 中解析 xml 文件有四种方法，学习了如何使用 DOM 解析 xml 文件，学习了 java 里的反射机制，学习了 java 的 Class 对象，学习了如何使用反射调用执行一个类中的某个方法，学习了 Strut2 的生命周期，了解了 Strut2 的工作流程，了解了 xml 与注解两种方式的比较，优缺点。

第一个问题，servlet-mapping 的 url-pattern 地址配置问题：

在写 url 拦截地址时，写为了“/*.sc”，则编辑阶段就会报错，服务器无法正常启动，后查阅资料了解到，url-pattern 中不允许以“.xxx”的路径前加任何“/”。

第二个问题，jar 包中方法读取资源文件路径问题：

为了避免路径发生变化，应该使用绝对路径，

```
String file = this.getClass().getClassLoader().getResource( name: "controller.xml").getPath();
```

同时在绝对路径中不能有空格，若打印路径存 20%字符，表示路径中有空格，无法读取文件成功，应该去掉所有空格。

6.参考文献

1. java pojo 类
2. IDEA 报 .....系统找不到指定文件的解决办法
3. java DOM 解析
4. xml 简介
5. Javaweb 学习笔记——使用 Jdom 解析 xml
6. XML 基础+Java 解析 XML
7. javaweb（实用）-IDEA 下 resources 目录下 txt 文件读取写入引发的项目后台路径问题总结
8. servlet-mapping 之 url-pattern 详解
9. Servlet 路径问题
10. 浅谈 java 反射机制
11. 说说 Java 反射机制
12. Java Class 对象详解
13. 深入理解 Java 类型信息(Class 对象)与反射机制
14. Strut2 核心工作原理
15. spring 注解和 xml 配置的优缺点比较
16. spring 配置是用注解还是 XML？