

轻量级 J2EE 框架应用

E 1 A Simple Controller with DI

学号：SA17225052

姓名：戴赛

报告撰写时间：2018/1/9

1.主题概述

主要需要增加一个读取 `di.xml` 的方法,返回 `di` 的配置信息

跟据配置信息,大量的使用反射与内省动态的实现 `bean` 与 `action` 类属性的初始化

2.假设

上次作业 E6

Tomcat

MySQL

3. 实现或证明

1.2.

配置 di.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<sc-di>
    <bean id="user" class="UserBean"></bean>
    <bean id="loginAction.sc" class="LoginAction">
        <field name="UserBean" bean-ref="user"></field>
    </bean>
</sc-di>
```

3.

将拦截请求与 controller.xml 匹配,匹配失败则响应客户端

这部分内容其实已经在 E2 中实现了

先将 post 拦截下来的内容与 controller.xml 比对,将结果设置到对应的 boolean 上

```
// 获得(2)层节点<action>的List
List<Element> actionList = controller.elements( s: "action");
boolean actionNotInXml = true; // 是否有在xml中找到对应action
for (Element action:actionList)
{
    String xmlActionName = action.attributeValue( s: "name");
    if (xmlActionName.equals(trueReqActionName))
    {
        // 请求中的action的名称能在xml中找到
        actionNotInXml = false;
        String xmlActionClass = action.attributeValue( s: "class");
    }
}
```

匹配成功则会进行相应的操作

匹配失败则根据 boolean 执行客户端响应

```
}
if (resultNotInXML) response.sendError( 501, s: "没有请求的资源");
} catch (ServletException e) {
    e.printStackTrace();
}
```

4.

找到后再检查 di.xml 中是否有 action 信息,如果没有的话直接反射构造 Action

```
String returnResult;
// 此处进行依赖注入逻辑
HashMap<String, ArrayList<BeanField>> diInfo = readDI();
if (diInfo.get(trueReqActionName) == null)
{
    // 未在di.xml中找到action信息
    System.out.println("~~~~~依赖注入~~~~~");
    System.out.println(trueReqActionName + "在依赖注入配置文件中未查询到相关bean");
    System.out.println("*****开始执行正常的反射逻辑*****");
    // 反射
    Class actionClass = Class.forName(xmlActionClass); // 构建类
    Object actionObject = actionClass.newInstance(); // 实例类(无参)
    Method actionMethod = actionClass.getDeclaredMethod(xmlActionMethod, String.class, String.class); // 获取handle方法

    // 反射后,获取执行对应类的对应方法的结果
    returnResult = (String) actionMethod.invoke(actionObject, userName, userPass);
    // returnResult = temp;
    System.out.println("反射后,获取执行对应类的对应方法的结果为" + returnResult);
}
else
{
    // di.xml中有action信息
    returnResult = doDI(trueReqActionName, xmlActionClass, diInfo, userName, userPass, xmlActionMethod);
}
}
```

读取 di.xml 的 readDI()方法

```
private HashMap<String, ArrayList<BeanField>> readDI()
{
    HashMap<String, ArrayList<BeanField>> returnHashMap = new HashMap<>();

    try {
        InputStream inputStream = this.getClass().getResourceAsStream( "name: "/di.xml");
        SAXReader saxReader = new SAXReader();
        Document document = saxReader.read(inputStream);

        // 获取根节点
        Element rootElement = document.getRootElement();
        // System.out.println(rootElement.getName());

        // 获取bean节点
        List<Element> beansElementList = rootElement.elements( "bean");
        for (Element beanElement:beansElementList)
        {
            String beanId = beanElement.attributeValue( "id");
            String beanClass = beanElement.attributeValue( "class");
            // 存储field节点信息,因为可能有多个field,所以用List存储
            ArrayList<BeanField> beanFieldsList = new ArrayList<>();
            List<Element> fieldElementList = beanElement.elements( "field");
            for (Element fieldElement:fieldElementList)
            {
                String fieldName = fieldElement.attributeValue( "name");
                String fieldBean_ref = fieldElement.attributeValue( "bean-ref");
                BeanField beanField = new BeanField(fieldName, fieldBean_ref, beanClass);
                // System.out.println(fieldName+" "+fieldBean_ref);
                beanFieldsList.add(beanField);
            }
            returnHashMap.put(beanId, beanFieldsList);
        }
        // inputStream = null;
    } catch (DocumentException e) {
        System.out.println("di.xml读取失败"+e);
    }

    return returnHashMap;
}
```

其中用来保存 bean 的 field 信息的 BeanField 类

```
public class BeanField {
    private String name;
    private String bean_ref;

    public String getBeanClass() { return beanClass; }

    private String beanClass;

    public BeanField(String name, String bean_ref, String beanClass)
    {
        this.name = name;
        this.bean_ref = bean_ref;
        this.beanClass = beanClass;
    }

    public String getBean_ref() { return bean_ref; }

    public String getName() { return name; }
}
```

5.6.

di.xml 中有对应信息,则继续查询 field 节点下的依赖信息

首先依然是反射构建 Action 对象

```
System.out.println(trueReqActionName+"在依赖注入配置文件中查询到了相关bean");
System.out.println("#####开始进行依赖注入#####");

// 反射构建Action类
Class actionClass = Class.forName(xmlActionClass); // 构建类
Object actionObject = actionClass.newInstance(); // 实现类(无参)

// 构建以<bean对象名,bean对象>的HashMap
```

各种 bean 对象作为 Action 的一个属性,为了能将这些 bean 对象进行相应的赋值,也需要通过内省机制,因此我们需要将这些 bean 对象保存下来

```
// 构建以<bean对象名,bean对象>的HashMap
HashMap<String, Object> beanHashMap = new HashMap<>();

// 查找Action下的field
```

之后对 bean 对象中的各个属性进行赋值,进行多次的内省(作业需要进行的赋值只有 userName 和 userPass)

```
// 查找Action下的field
ArrayList<BeanField> beanFieldsList = diInfo.get(trueReqActionName);
for (BeanField beanField:beanFieldsList)
{
    String beanName = beanField.getName();
    String beanBean_ref = beanField.getBean_ref();

    // 拼接bean的完整名
    beanName = "water.ustc.bean."+beanName;

    // Bean的内省,传入username和pwd
    Class beanClass = Class.forName(beanName);
    Object beanObject = beanClass.newInstance();
    // 后面参数表示停止继承参数,表示不显示从父类继承下来的属性
    // 每个类都会从Object类继承下class属性,所以这里排除
    BeanInfo beanInfo = Introspector.getBeanInfo(beanClass, Object.class);

    // 对Bean的内省开始
    PropertyDescriptor[] properties = beanInfo.getPropertyDescriptors();
    for (PropertyDescriptor property:properties)
    {
        System.out.println("IIIIIIIIIIIIIIIIII");
        Method writeMethod = property.getWriteMethod();
        // 因为现在从页表上得到参数只有userName与userPass,所以先写成这样
        switch (property.getName())
        {
            case "userName":
            {
                writeMethod.invoke(beanObject,userName);
                break;
            }
            case "userPass":
            {
                writeMethod.invoke(beanObject,userPass);
                break;
            }
        }
        System.out.println(property.getName()+"的getter为"+writeMethod.getName());
        System.out.println("IIIIIIIIIIIIIIIIII");
    }
}
```


对进行完赋值的 bean 对象保存进 HashMap 中

```

        System.out.println("依赖注入成功");
    }

    // 将构建好的bean对象存入HashMap中
    beanHashMap.put(beanBean_ref, beanObject);
}
// 遍历HashMap，依格内省将bean注入Action

```

然后再用内省将各个 bean 注入 Action

```

// 遍历HashMap，依格内省将bean注入Action
BeanInfo actionInfo = Introspector.getBeanInfo(actionClass, Object.class);
PropertyDescriptor[] actionProperties = actionInfo.getPropertyDescriptors();
for (PropertyDescriptor actionProperty:actionProperties)
{
    // 获取Action中作为属性的bean对象
    Object beanObject = beanHashMap.get(actionProperty.getName());
    // 获取setter
    Method writeMethod = actionProperty.getWriteMethod();
    // 将bean注入Action
    writeMethod.invoke(actionObject, beanObject);
    System.out.println("依赖注入中注入Action的属性为"+actionProperty.getName());
}
// 执行Action
Method actionMethod = actionClass.getDeclaredMethod(xmlActionMethod); // 获取handle方法
return (String)actionMethod.invoke(actionObject);
catch (Exception e) {
    System.out.println("依赖注入失败:"+e);
    return null;
}

```

7.修改 LoginAction

```

/**
 * public class LoginAction {

    private UserBean user;

    public UserBean getUser() { return user; }

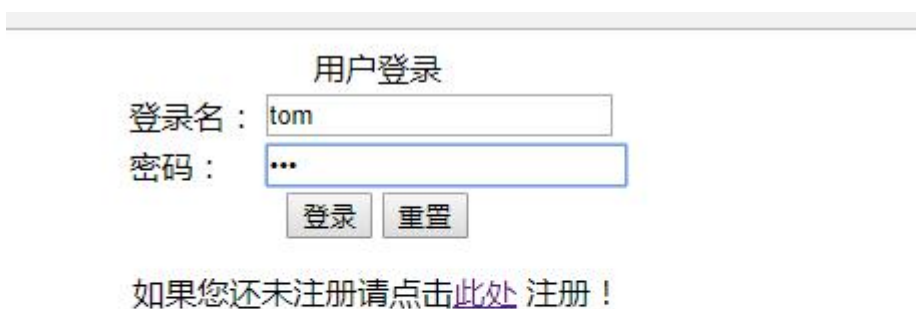
    public void setUser(UserBean user) { this.user = user; }

    public String handleLogin()
    {
        // UserBean userBean = new UserBean(name, pwd);
        if(user.signIn())
        {
            return "success";
        }else {
            return "failure";
        }
    }
}

```

4. 结论

1. 验证 进行登陆



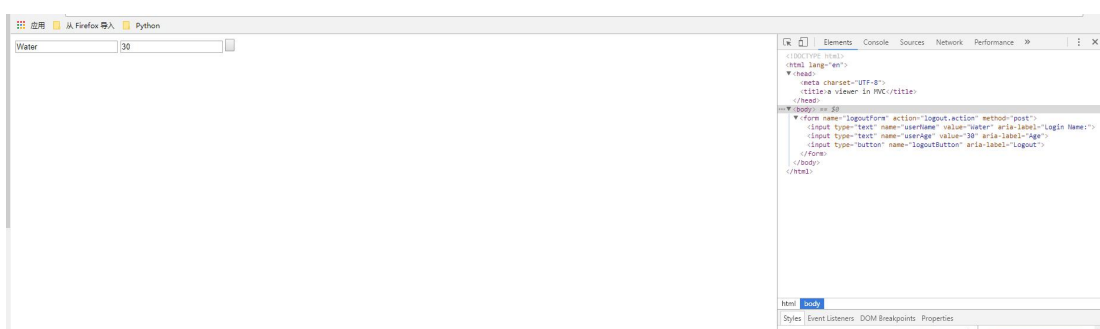
用户登录

登录名:

密码:

如果您还未注册请点击[此处](#)注册!

登陆正常



各个功能(数据库,依赖注入,拦截器,动态页面)正常执行

```
loginAction.sc的class为water.ustc.action.LoginAction
loginAction.sc的Method为handleLogin
~~~~~依赖注入~~~~~
loginAction.sc在依赖注入配置文件中查询到了相关bean
#####开始进行依赖注入#####
IIIIIIIIIIIIIIIIII
userName的getter为setUserName
IIIIIIIIIIIIIIIIII
IIIIIIIIIIIIIIIIII
userPass的getter为setUserPass
IIIIIIIIIIIIIIIIII
依赖注入中注入Action的属性为user
~~~~~以下是Conversation~~~~~
传入的类名为UserBean
根节点名称:OR-Mappings
driver class
```



```

/success
*****ViewXML*****
根节点名称:view,其内容为:
  标题部分(1)层节点名称为header
  标题部分(2)层节点名称为title
  其文本内容为a viewer in MVC
  内容部分(1)层节点名称为body
  内容部分(2)层节点名称为form
    内容部分(3)层节点名称name
    内容部分(3)层节点名称action
    内容部分(3)层节点名称method
    内容部分(3)层节点名称textView
  写入内容为:
    <input type="text" name="userName value=Water aria-label=Login Name:">
      内容部分(3)层节点名称textView
  写入内容为:
    <input type="text" name="userAge value=30 aria-label=Age">
      内容部分(3)层节点名称buttonView
  写入内容为:
    <input type="button" name="logoutButton" aria-label="Logout">
*****ViewXML*****
视图HTML:/success.html
其value为:/success.html
XML中loginAction.scAction的结果有failure
未在xml中找到对应的result结果,继续查询result
-----继续查询Result-----
action退出时的拦截器为:log
现在执行拦截器afterDo方法
此时时间为2018-01-09 17:33:07
在写入log前检查四个属性
name:log

```

2.

这次作业是 J2EE 课程的最后一次作业,感谢朱老师的课程,让我的 JAVA 编程能力有了极大的提高.

我上半学期曾选修过朱老师的安卓课程,作为一名跨专业学生,编程能力较为薄弱,在安卓课程上表现很不理想.于是为了锻炼自己的能力,在 J2EE 课程上我决定自己一个人一队,独自完成所有 J2EE 作业.最开始,我需要约一周的时间才能完成第一次作业 E1 时,但随着课程的进行,我完成作业的时间越来越短,部分功能甚至可以通过多个方法来实现.

当然,在作业中还是有些任务没有完成,如利用 XSLT 来动态的生成页面,使用 Invocation 动态代理实现懒加载.

如果有可能,我会在下学期选修朱老师的软件设计模式课程.最后,十分感谢朱老师的教导,谢谢老师.

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