

轻量级 J2EE 框架应用

E 7 A Simple Controller With DI

学号: SA16225221

姓名: 欧勇

报告撰写时间: 2017/1/8

1. 主题概述

简要介绍主题的核心内容，如 MVC，及 MVC 中 Controller 的作用与实现作业内容：

1. 基于 E2。新建一个 XML 文件名为 di.xml，格式可参考如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<dependency-mappings>
  <bean>
    <name>user</name>
    <class>water.model.UserBean</class>
    <property>
      <name>userName</name>
    </property>
    <property>
      <name>userPass</name>
    </property>
    <property>
      <name>userID</name>
    </property>
  </bean>
  <bean>
    <name>loginAction</name>
    <class>water.action.LoginAction</class>
    <property>
      <name>userBean</name>
      <ref-class>user</ref-class>
    </property>
  </bean>
  <!-- other beans -->
</dependency-mappings>
```

2. 在 di.xml 中定义<bean>节点。<bean>节点中指明当前 bean 的名字、class 类型及 bean 属性是否引用另一个 bean。如示例中 loginAction 的属性 userBean 引用名为 user 的 bean。
3. 修改 E2 中的控制器代码。当有 http 登录请求被拦截后，在 controller.xml 中查找是否有对应的 Action 请求。如果无，直接响应客户端“无法识别该请求”。
4. 如果找到该请求对应的 Action，则在 di.xml 查找与请求 Action 同名的 bean 节点。如果 di.xml 中无指定<bean>节点，Controller 直接通过反射构造 Action 实例，并将请求分发给 Action 实例进行处理。
5. 如果在 di.xml 中找到指定的<bean>节点，则查看该节点是否有属性依赖其他

<bean>节点。如果无依赖，则直接通过反射构造该<bean>实例，并将请求分发至其处理。

6. 如果有依赖，则需先通过反射构造 被依赖的<bean>实例，之后再构造 依赖<bean>实例；并通过属性的 setter 方法（Java 内省机制，Introspector）将 被依赖的<bean>实例注入依赖<bean>实例。

7. 完成以上工作后，修改 E 2 中的 LoginAction 的代码，将对象属性初始化代码语句移除。重新打包工程测试。

8. 基于以上内容，修改 E 3-E 6 中关于对象依赖的代码，将对象依赖关系通过 di.xml 进行管理。重新打包工程，测试是否能够正确运行。如修改 DAO 的依赖代码，修改 Model 的依赖代码等。

2. 假设

主题内容所参照的假设条件，或假定的某故事场景，如调试工具或软硬件环境

开发环境：

Win10

Eclipse kepler

JDK 1.8

Tomcat 7.0

3. 实现或证明

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）

本次 E7 作业报告基于 E2 作业报告

本次作业流程：

假设用户名为 world，密码为 hello

为了方便查看，采用 get 方式提交，可以通过浏览器 url 看到输入的用户名和密码

（因为若采用 post 方式则无法通过 url 看到用户名和密码，所以采用 get 方式提交）

若登录成功则跳转 login_success.jsp 页面，页面显示 Login Success 的字符串

若登录失败则跳转 login_fail.jsp 页面，页面显示 Login Fail 的字符串

若使用未知的 action 提交，既 action="unknow.scaction"则无法找到相应的方法处理，则页面显示“不可识别的 action 请求”

若返回的是未知的处理结果，则返回 error_result.jsp 页面，页面显示“没有请求的资源”

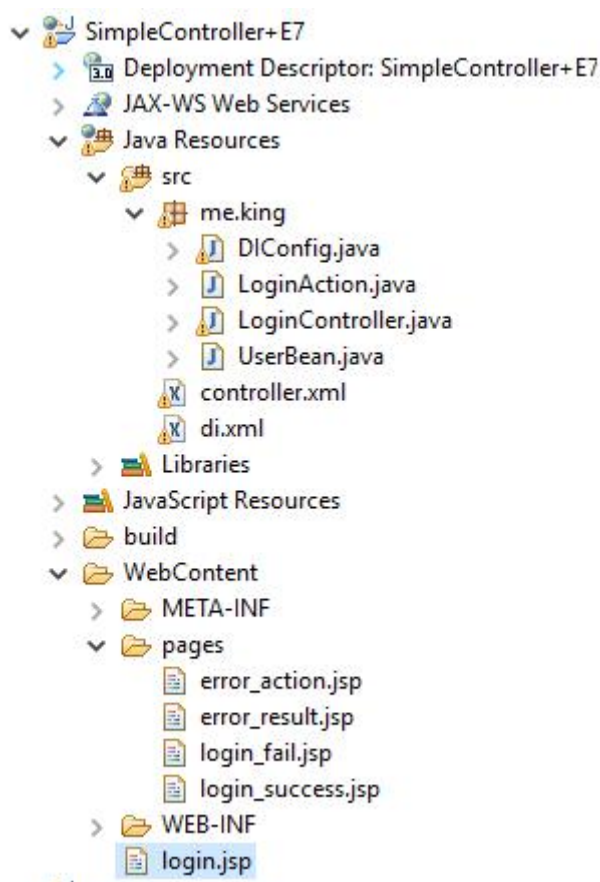


图 1：项目目录结构

图 1：项目目录结构，可以看出项目名称为 SimpleController，src 文件夹下有 controller.xml

配置文件，其中记录有 action 和 result 相关的配置，di.xml 配置文件配置了所有 bean。同时还有名为 me.king 的包，其下有 LoginController 作为控制层，UserBean 作为模型层，LoginAction 为处理登录请求的类，DIConfig 为处理配置文件 di.xml 的类，然后还有 5 个 jsp 页面作为视图层，分别是 login_fail.jsp, login_success.jsp, error_action.jsp（本次作业未使用），error_result.jsp 和 login.jsp。

注意在 WEB-INF/lib 下需要导入 dom4j 的 jar 包，若仅仅只是将 jar 包放入 Java Resources/Libraries 中，则在编译时能通过，但是却无法完成处理，因为会在执行到语句 `new SAXReader()` 时报如下错误，提示找不到对应的类。

```

root cause
java.lang.NoClassDefFoundError: org/dom4j/io/SAXReader
    me.king.LoginController.doGet(LoginController.java:48)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:621)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
    org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:51)

```

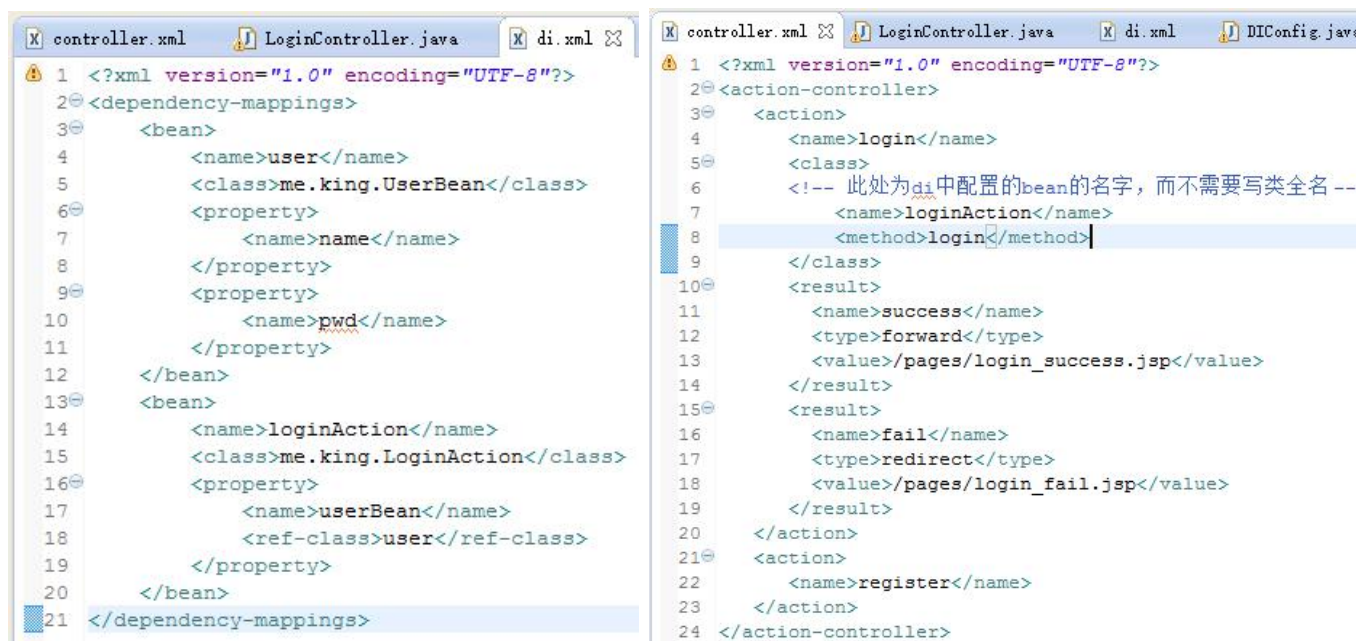


图 2: di.xml(左)和 controll.xml(右)配置文件截图

图 2: di.xml(左)和 controll.xml(右)配置文件截图

如图，在 di.xml 中配置了 user 类和 loginAction 类两个 bean，loginAction 依赖于 user


A screenshot of an IDE showing the configuration of a web application in web.xml. The file is named 'controller.xml' and contains XML tags for a web-app, welcome-file-list, and servlet-mapping. A red arrow points to the url-pattern attribute in the servlet-mapping tag, which is set to *.saction.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <display-name>SimpleController</display-name>
  <welcome-file-list>
    <welcome-file>login.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>LoginController</servlet-name>
    <servlet-class>me.king.LoginController</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginController</servlet-name>
    <url-pattern>*.saction</url-pattern>
  </servlet-mapping>
</web-app>
```

图 2.1: web.xml 配置文件截图

图 2.1: web.xml 配置文件截图

不使用注解的方式告知容器，而是用配置文件的方式配置控制器 LoginController 的映射路径,其中，将 login.jsp 配置为默认页面，将 servlet 控制层类 LoginController 映射名为同类名，同时，对所有以.saction 结尾的 url 请求进行转发和控制。

A screenshot of an IDE showing the Java code for LoginController. The code includes package, import, and class declarations. The doGet method is highlighted with a red box, showing it throws ServletException and IOException.

```
1 package me.king;
2
3 import java.io.File;
4
21
23 * Servlet implementation class LoginController
25 @@WebServlet("/LoginController")
26 public class LoginController extends HttpServlet {
27     private static final long serialVersionUID = 1L;
28
30 * @see HttpServlet#HttpServlet()
32 public LoginController() {
33     super();
34     // TODO Auto-generated constructor stub
35 }
36
38 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
40 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
90 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
91     // TODO Auto-generated method stub
92     doGet(request, response);
93 }
94 }
```

图 3.: LoginController 代码截图

图 3.: LoginController 代码截图

在 doGet 方法中进行控制转发，其他采用默认，若前端页面采用 post 方式提交，则在 doPost 方法中也需要进行转发处理，本次采用直接调用 doGet()方法进行处理。

```

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
    ServletContext sc = getServletContext();
    //获取请求的action
    String actStr = request.getRequestURL().toString();
    String actionName = actStr.substring(actStr.lastIndexOf('/')+1,actStr.indexOf(".saction")).toString();
    InputStream is = this.getClass().getResourceAsStream("/controller.xml");//获取controller.xml配置文件，并以流的形式读入
    try {
        Document dc = (new SAXReader()).read(is);
        List<Element> actions = dc.getRootElement().elements(); //获取所有的action标签并加入到list中
        boolean isFindAction = false, isFindResult = false;
        for(Element ele: actions){
            if( actionName.equals(ele.elementText("name"))){
                isFindAction = true; //将标志位设置为成功找到该action
                String className = ele.element("class").elementText("name"),
                    classMethod = ele.element("class").elementText("method");

                //利用反射机制实例化对应的类，对应于原Class.forName(className);
                Object o = DIConfig.getBean(className); //获取className对应的对象
                Class<?> cls = o.getClass();
                //指定获取当前类的classMethod方法，同时指定参数列表的类型为string, string
                Method m = cls.getDeclaredMethod(classMethod, new Class[]{String.class,String.class});
                //调用该方法并将返回结果存入rstName中，在本例中，这里调用的是LoginAction类的login方法
                String rstName = (String) m.invoke(o,request.getParameter("name"),request.getParameter("pwd"));

                List<Element> rstes = ele.elements("result");//所有的result标签
                for(Element rstEle: rstes){ //一个rstEle对应一个result节点
                    if(rstName.equals( rstEle.elementText("name"))){ //当存在对应result返回结果的配置信息的时候
                        isFindResult = true; //将标志位设置为成功找到该result
                        String rstValue = rstEle.elementText("value");
                        if(rstEle.elementText("type").equals("forward"))
                            sc.getRequestDispatcher(rstValue).forward(request, response); //forward为内部重定向
                        else response.sendRedirect('.'+rstValue); //重定向根据相对地址跳转
                    }
                }
                if(!isFindResult)
                    response.sendRedirect("./pages/error_result.jsp"); //不存在对应result的时候跳转至error_result页面
            }
        }
        if(!isFindAction) //当找不到对应action请求的时候直接响应客户端“无法识别该请求”
            response.getWriter().write("Sorry, this is a unrecognized action request.");
            //response.sendRedirect("./pages/error_action.jsp");
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}

```

图 3.1: LoginController 类中 doGet 方法代码截图

图 3.1: LoginController 类中 doGet 方法代码截图

具体代码的解释已经写在注释中，对 doGet 中主要控制转发流程解释如下：

1. 获取 request 中的 action 请求名，获取 controller.xml 配置文件
2. 解析配置文件并将所有的 action 标签都加入到 list 中
3. 设置标志位用于标志是否找到对应 action
4. 遍历 action 标签 list 并判断 name 标签和 method 标签
5. 使用 DIConfig 类获取对应类的对象 o，获取实例化对应的类为 cls 对象
6. 获取对应的 method 方法，调用实例化后 cls 的 method 方法并获取返回结果
7. 将返回结果与 result 标签 list 中的对应 name 对比
8. 获取对应 result 标签中的 value（返回地址），type（跳转还是内部重定向）
9. 执行返回操作

同时，在最后执行转发操作中，使用 response.sendRedirect()方法时，需要使用相对定位，否则会报 `:Path pages/login_success.jsp does not start with a "/" character` 错误，并且页面显示 404 错误码，找不到资源。

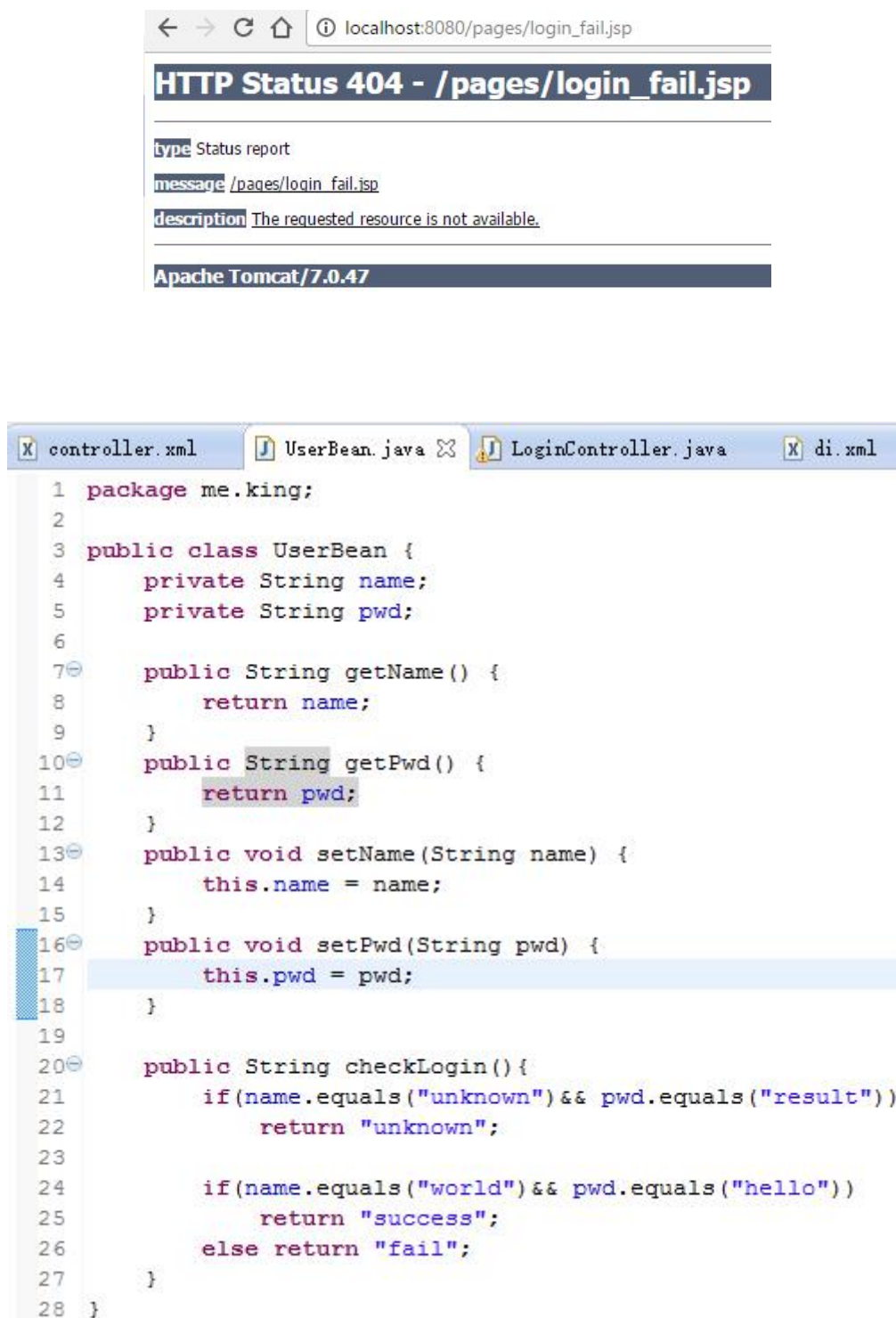


图 4: UserBean 代码截图

图 4: UserBean 代码截图，定义一个简单的 `checkLogin` 方法，参数为 `name` 和 `pwd`，函数体内直接对比两个字符串，若正确则返回“success”，否则返回“fail”。同时为了测试方便，当 `name` 为 `unknown` 并且 `pwd` 为 `result` 的时候返回“unknown”。

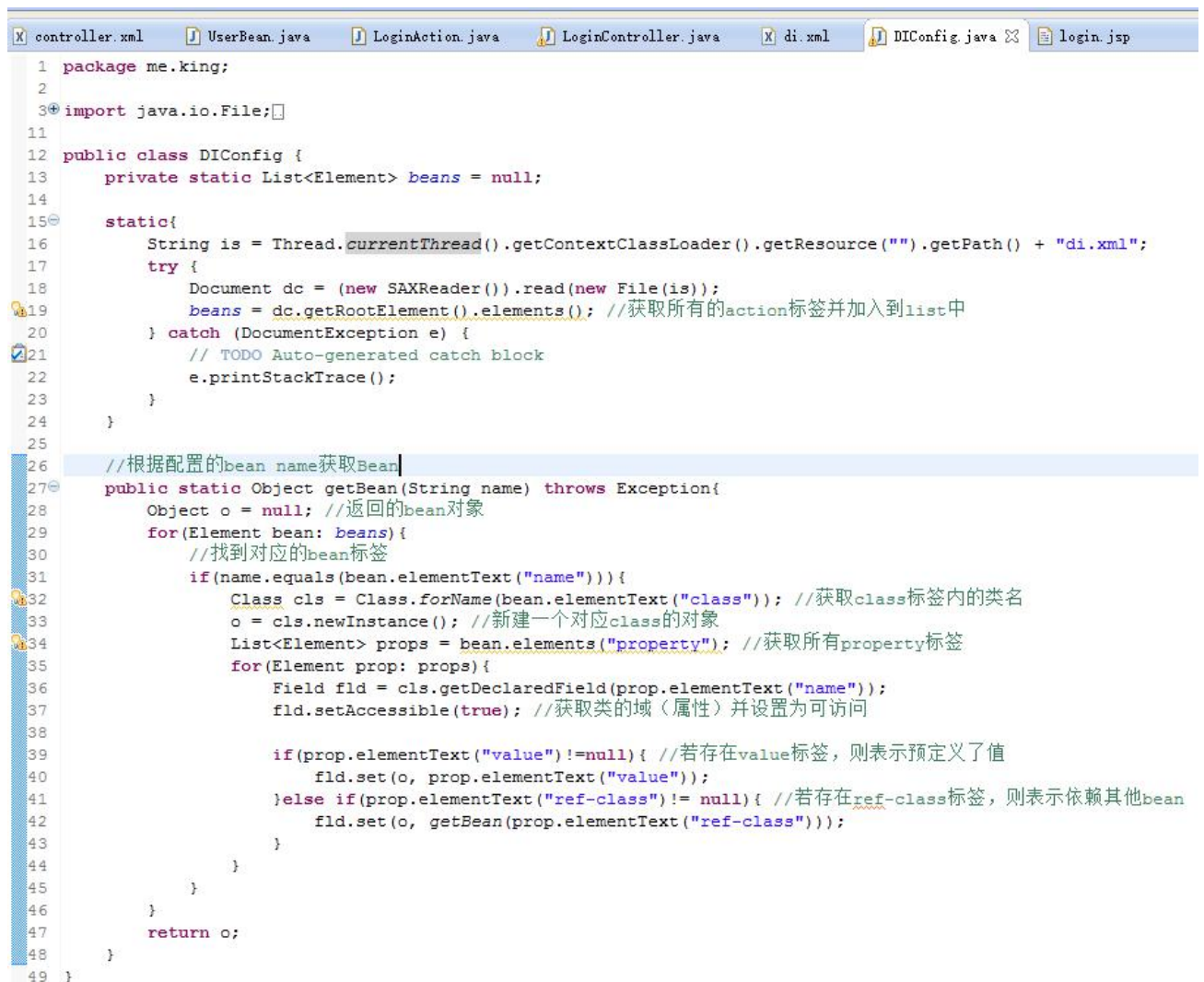


```

1 package me.king;
2
3 public class LoginAction {
4     private UserBean userBean = null;
5
6     //将name和pwd都设置到userBean中，然后调用userBean的checkLogin方法判断
7     public String login(String name, String pwd){
8         userBean.setName(name);
9         userBean.setPwd(pwd);
10        return userBean.checkLogin();
11    }
12
13    public UserBean getUserBean() {
14        return userBean;
15    }
16
17    public void setUserBean(UserBean userBean) {
18        this.userBean = userBean;
19    }
20 }

```

图 5: LoginAction 代码截图



```

1 package me.king;
2
3 import java.io.File;
4
11
12 public class DIConfig {
13     private static List<Element> beans = null;
14
15     static{
16         String is = Thread.currentThread().getContextClassLoader().getResource("").getPath() + "di.xml";
17         try {
18             Document dc = (new SAXReader()).read(new File(is));
19             beans = dc.getRootElement().elements(); //获取所有的action标签并加入到list中
20         } catch (DocumentException e) {
21             // TODO Auto-generated catch block
22             e.printStackTrace();
23         }
24     }
25
26     //根据配置的bean name获取Bean
27     public static Object getBean(String name) throws Exception{
28         Object o = null; //返回的bean对象
29         for(Element bean: beans){
30             //找到对应的bean标签
31             if(name.equals(bean.elementText("name"))){
32                 Class cls = Class.forName(bean.elementText("class")); //获取class标签内的类名
33                 o = cls.newInstance(); //新建一个对应class的对象
34                 List<Element> props = bean.elements("property"); //获取所有property标签
35                 for(Element prop: props){
36                     Field fld = cls.getDeclaredField(prop.elementText("name"));
37                     fld.setAccessible(true); //获取类的域（属性）并设置为可访问
38
39                     if(prop.elementText("value")!=null){ //若存在value标签，则表示预定义了值
40                         fld.set(o, prop.elementText("value"));
41                     }else if(prop.elementText("ref-class")!= null){ //若存在ref-class标签，则表示依赖其他bean
42                         fld.set(o, getBean(prop.elementText("ref-class")));
43                     }
44                 }
45             }
46         }
47         return o;
48     }
49 }

```

图 6: DIConfig 代码截图

图 6: DIConfig 代码截图

整个 DIConfig 类都是静态的方法和属性，默认先获取配置文件并使用 beans 保存所有 bean 标签

getBean 方法流程如下：

1. 根据 name 参数从 beans 链表中找到对应的 bean
2. 利用反射机制新建一个对应的类
3. 遍历 bean 下的 property 标签（即对应类的域），并设置对应的域为可访问
4. 若遇到 property 标签下配置了 value 标签（内置类型使用 value 预设），则初始化该域 的值为 value 标签中的值
5. 若配置了 ref-class 标签则调用 getBean，将返回的对象设置为该域的值（即递归调用 并反射出对应的域）

```

1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7 <title>Insert title here</title>
8 </head>
9 <body>
10 <form action="login.scaction">
11   <label>NAME:</label><input type="text" name="name"><br>
12   <label>PASSWORD:</label><input type="password" name="pwd"><br>
13   <input type="submit" value="LOGIN">
14 </form>
15 <br><br><br>
16 <form action="unknown.scaction">
17   <label>This is a unknown action request</label><br>
18   <input type="submit" value="unknown">
19 </form>
20
21 </body>
22 </html>

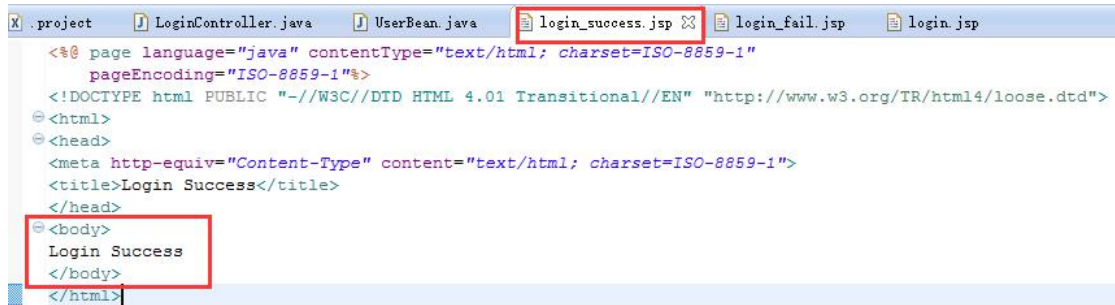
```

图 7.1: login.jsp 代码截图

图 7.1: login.jsp 代码截图，

编写 2 个 form 表单，第一个为正常登录表单 action 设置为“login.scaction”（必须设置为 后缀.scaction，具体配置在 controller.xml 中已经设置完成），设置用户名和密码的 name 属性分别为“name”和“pwd”。

同时为了测试方便，定义一个 action 为“unknown.scaction”的表单，用于测试当 action 为 未知的情况。



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login Success</title>
</head>
<body>
Login Success
</body>
</html>
```

图 7.2: login_success.jsp 代码截图

图 7.2: login_success.jsp 代码截图，简单的在 body 中写入 Login Success。



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login Fail</title>
</head>
<body>
Login Fail
</body>
</html>
```

图 7.3: login_fail.jsp 代码截图

图 7.3: login_fail.jsp 代码截图，简单的在 body 中写入 Login Fail。



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>ERROR ACTION</title>
</head>
<body>
Sorry, this is a unrecognized action request.
</body>
</html>
```

图 7.4: error_action.jsp 代码截图

图 7.4: error_action.jsp 代码截图，简单的在 body 中写入 Sorry, this is a unrecognized action request. error_action.jsp 在 E2 中使用，但在本次作业中并没有用到。

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7 <title>ERROR RESULT</title>
8 </head>
9 <body>
10 <div style="border: 1px solid red; padding: 2px;>Sorry, there is no resources you request.</div>
11 </body>
12 </html>
```

图 7.5: error_result.jsp 代码截图

图 7.5: error_result.jsp 代码截图

简单的在 body 中写入 `Sorry, there is no resources you request.`

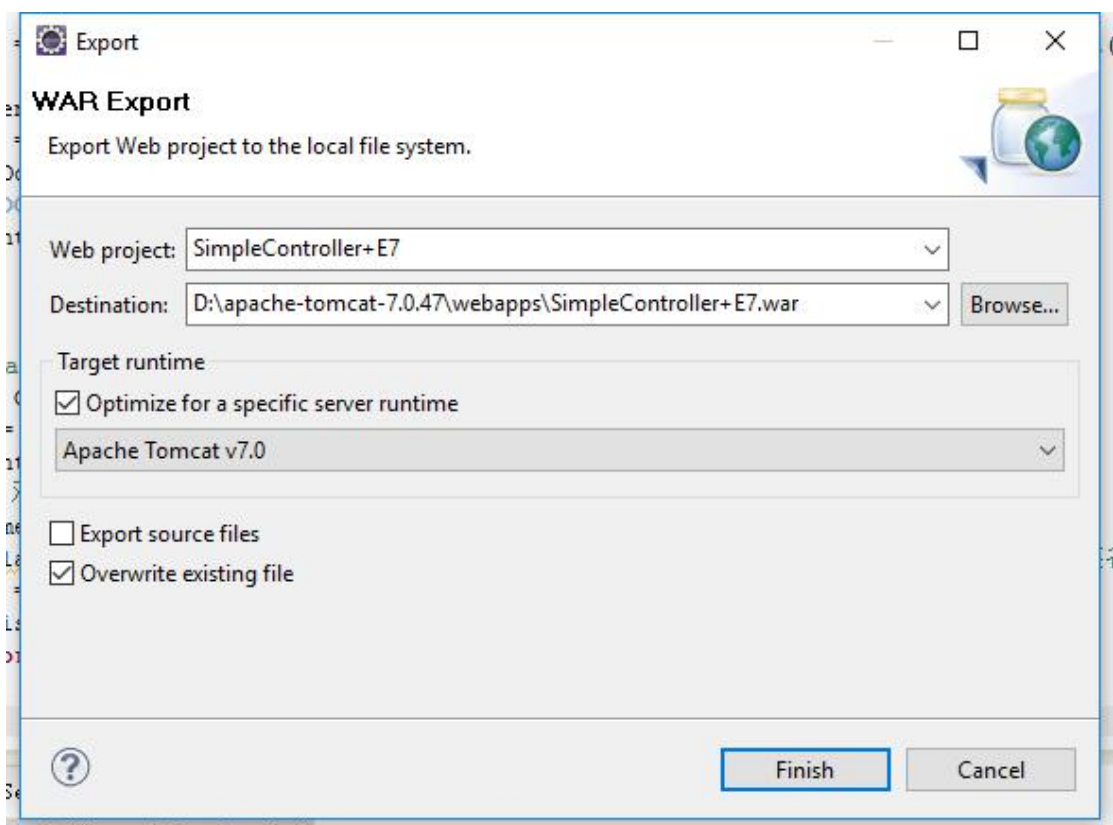


图 8: 将项目导出为 war 包

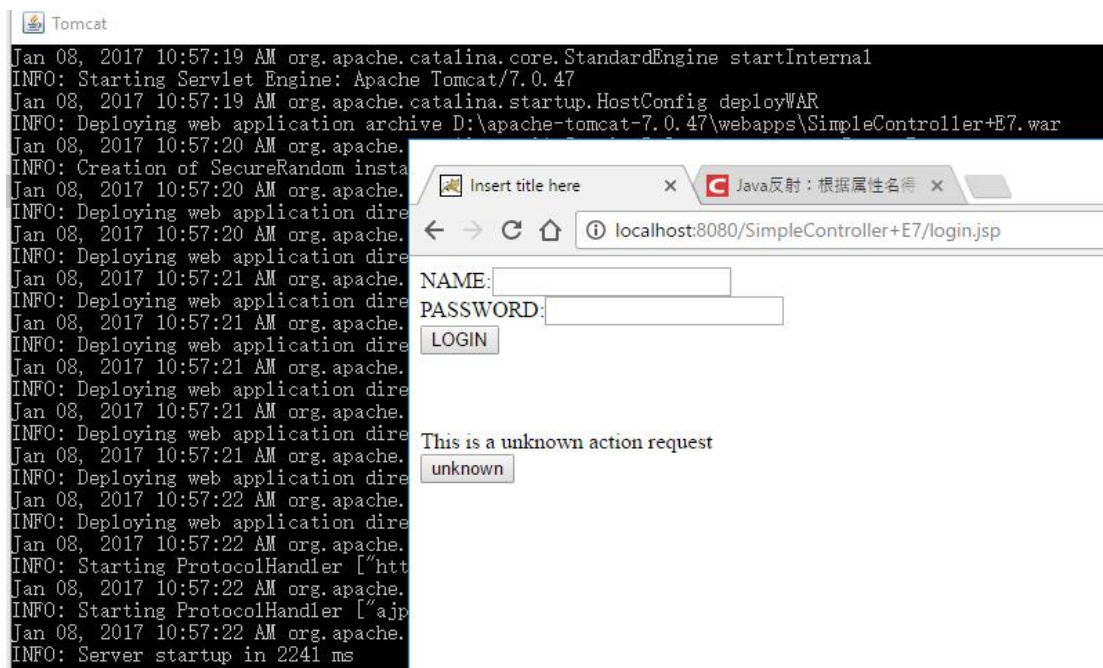


图 8.1: 使用 Chrome 浏览器测试登录页面截图

图 8.1: 使用 Chrome 浏览器测试登录页面截图，可以看到 url 为：<http://localhost:8080/SimpleController+E7/login.jsp>，可以在表单中分别填入 NAME 和 PASSWORD，点击 LOGIN 按钮提交至后台服务器。



图 8.2: 登录成功截图

图 8.2: 登录成功截图，可以通过 url 看到，当输入的用户名 name 为 world，密码 pwd 为 hello 时验证通过时显示 Login Success（虽然页面内容已经变为 login_success.jsp 页面，但此时 url 没有改变为 login_success.jsp 而是在 login.scaction 中）



图 8.3: 登录失败截图

图 8.3: 登录失败截图,

可以通过 url 看到, 当输入的用户名 name 或密码 pwd 错误时, 验证失败, 跳转至 login_fail.jsp 页面, 显示 Login Fail, 而由于使用的是 `sendRedirect` 重定向, 所以会对 url 进行转码, 将+号转为%2bE7



图 8.4: 不可识别的 action 请求截图

图 8.4: 不可识别的 action 请求截图

当点击图 8.1 中的 unknown 按钮的时候, 页面并没有跳转至 error_action.jsp 页面, 而是直接显示 Sorry, this is a unrecognized action request. 因为在 LoginController 类 doGet 方法中使用 write 直接响应客户端:

```
if(!isFindAction) //当找不到对应action请求的时候直接响应客户端“无法识别该请求”
    response.getWriter().write("Sorry, this is a unrecognized action request.");
//response.sendRedirect("../pages/error_action.jsp");
```



图 8.5: 没有请求的资源截图

图 8.5: 没有请求的资源截图

当在登陆表单中输入的用户名 name 为 unknown, 密码 pwd 为 result 时, 跳转至 error_result.jsp 页面。显示 Sorry, there is no resources you request.

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）

8. 基于以上内容，修改 E3-E6 中关于对象依赖的代码，将对象依赖关系通过 di.xml 进行管理。重新打包工程，测试是否能够正确运行。如修改 DAO 的依赖代码，修改 Model 的依赖代码等。

关于第 8 步，由于时间关系（主要准备复习考试），且在已完成前 7 个任务的基础上对其他几个作业进行修改难度不大，所以并没有对 E3-E6 中关于对象依赖的代码进行修改和测试。

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

- [1] Java Reflection : <https://docs.oracle.com/javase/tutorial/reflect/>
- [2] XML Parser SAX : <http://www.saxproject.org/quickstart.html>
- [3] XML Parser DOM : http://www.w3schools.com/dom/dom_parser.asp
- [4] 在 java 中使用 dom4j 解析 xml : <http://www.jb51.net/article/42323.htm>
- [5] 通过 Java 反射调用方法 : <http://blog.csdn.net/ichsonx/article/details/9108173>