

轻量级 J2EE 框架应用

E 1 A Simple Controller

学号: SA16225221

姓名: 欧勇

报告撰写时间: 2016/11/25

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. 搭建 Java Web 开发环境(Eclipse, JDK, Tomcat)
2. 新建 Java Web 工程 SimpleController ， 新建 HttpServlet 子类 LoginController，使用 RequestDispatcher 实现登录控制器功能如下：
 - 2.1 该控制器能够接受 Http Request 登录请求（请求中携带登录数据用户名、密码）
 - 2.2 该控制器能够将登录请求分发至模型 UserBean，UserBean 完成登录业务处理
 - 2.3 该控制器能够将 UserBean 业务处理结果定向到结果视图；如果登录成功视图定向至 login_success.jsp/html；如果失败，视图定向至 login_fail.jsp/html
 - 2.4 将 SimpleController 打包为 war，部署在 Tomcat 中，并根据设计的测试用例测试该控制器，验证结果是否正确
3. Http request 在 Web Container 中的处理流程
4. 比较 JSP Model 1 与 Model 2 架构，说明各自的优缺点及适用场景

1) MVC 概念

MVC(Model View Controller)，分别表示 M 模型(model)–V 视图(view)–C 控制器(controller)，

MVC 是一种软件框架模式（不是设计模式，两者有区别），用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC 被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。

2) MVC 中 Controller 的作用与实现

控制层在服务器端，作为连接视图层（比如用户交互的界面）和模型层（处理业务、提供服务）的纽带，对客户端 request 进行过滤和转发，决定由哪个类来处理请求，或者决定给客户端返回哪个视图，即确定服务器的 response 相对应的视图。

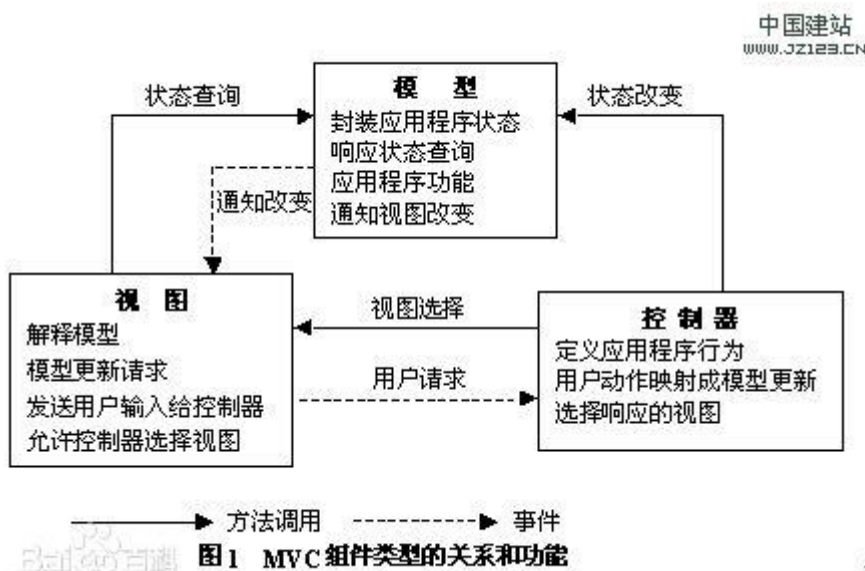
一个 Controller 其实本质上也是一个 servlet，它的实现需要符合 servlet 的标准，即继承 HttpServlet 类，覆写 doGet 或 doPost 等方法，同时可以定义过滤器。

3) MVC 中 View 的作用与实现

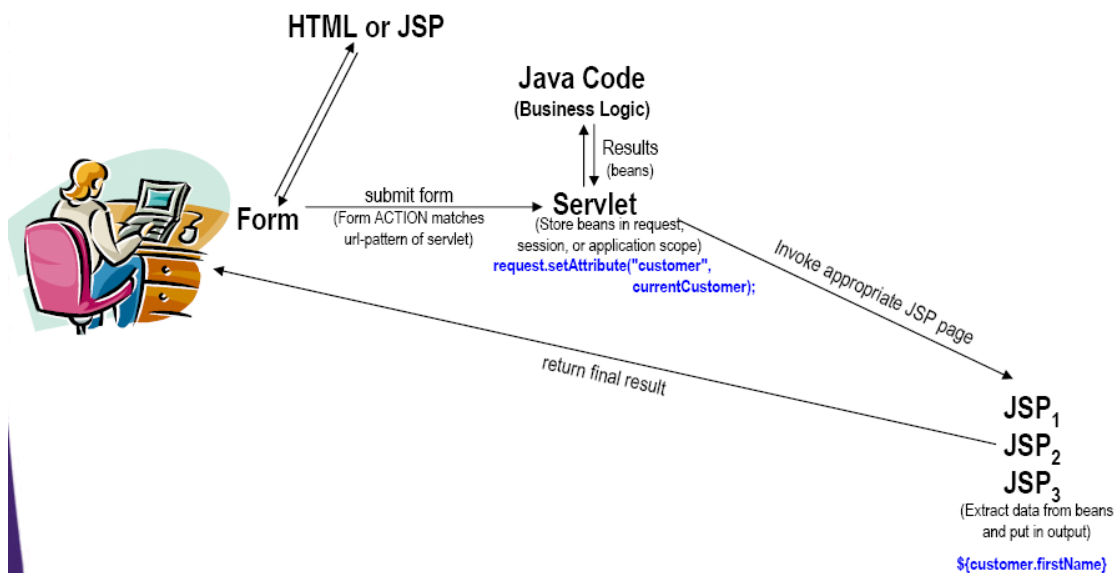
视图层其实就是真正与客户端交互的界面，一般为 jsp 动态页面或 html 静态网页，视图层的数据可以来自用户（用户输入数据通过请求发送到服务器端）和模型层（服务器返回请求的数据），但是如何展示这些数据却是由视图层自己控制和安排的，所以可以根据用户的需求来改变展示的方式和形式。

4) MVC 中 Model 的作用与实现

模型层才是真正处理用户请求的地方，所以模型层会很庞大（因为业务的种类很多，需要处理和计算的任务也会有很多），所以通常还会对模型层进行再分层。比如与数据库的连接和对数据进行处理等。



MVC 框架调用图，取自百度百科



如上图，首先假设用户已经打开一个页面，然后用户在表单中输入自己的用户名和密码，然后点击确定，这时，表单中的数据（用户名和密码）已经被浏览器包装成为一个 Http Message 请求（填入 http header 和 body）发送到服务器端了，在服务器端处理完成后，服务器将通过跳转页面（也可使用 AJAX 无页面刷新/跳转的返回）将结果页面返回。

其中，Http Message 在服务器端的处理流程（即 **Http request** 在 **Web Container** 中的处理流程）如下：

- 1) 来自客户端的 Http Message 在服务器端会被 Web Container 自动包装在一个 Http request 对象中，这个对象存放着请求的参数和信息。
- 2) Request 对象被 Web Container 接收到的时候会判断 Servlet 实例是否存在，若不存在则会先装载对应的 Servlet 类并创建对应的实例，然后调用 `init(ServletConfig)` 方法初始化 servlet 上下文，然后调用 `service(ServletRequest, ServletResponse)` 对请求进行处理，若存在则直接调用 `service(ServletRequest, ServletResponse)`。
- 3) 当调用 service 方法处理时，需要 Web Container 把 HttpRequest 对象与 HttpResponse 对象作为参数传给执行处理任务的 HttpServlet 对象。（这里有可能是一个 Controller，也有可能是一个 JavaBean）
- 4) 当 HttpServlet 对象进行处理的时候，会同时获取到 Http request 中 header 和 body 中的信息，然后就可以对这些信息进行了。
- 5) 处理完成后，HttpServlet 对象调用 HttpResponse 对象的有关方法，生成响应数据；（这里有可能会先进行 Controller 的转发，将决定响应数据的视图）
- 6) 然后 Web Container 将响应结果发给浏览器。

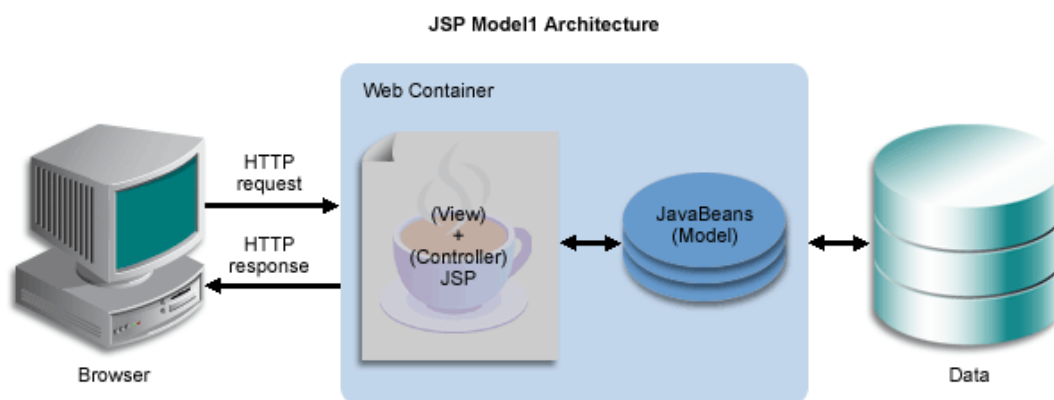
比较 JSP Model 1 与 Model 2 架构，说明各自的优缺点及适用场景

JSP 技术的优点（不分 Model 1 或 2）：

- 1) 一次编写，到处运行，基于 Java 平台的优势。
- 2) 安全性高，可以对屏蔽一些敏感信息。
- 3) 伸缩性高，通过消息处理机制，可以扩展为分布式。
- 4) 开发工具多，社区支持，活跃用户多。

JSP 技术的缺点（不分 Model 1 或 2）：

- 1) 由于基于 Java 平台也增加了复杂性。
- 2) 调试麻烦。

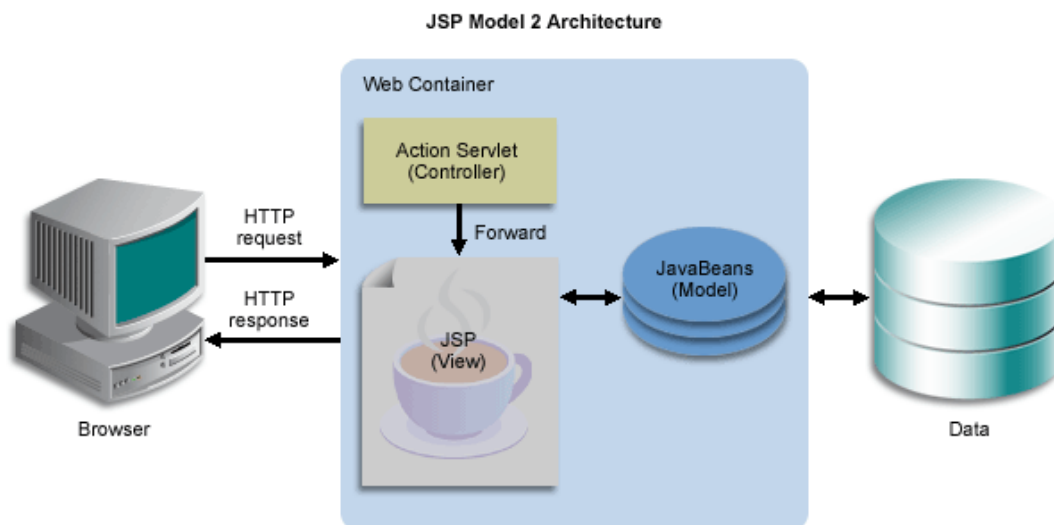


JSP Model 1 架构是以页面为中心的，将视图层和控制层都放在 jsp 页面中。

优点：开发容易迅速，逻辑简单清晰，执行效率比 Model2 要高，因为控制和展现不需要调用其他对象。

缺点：控制和视图高度耦合，维护和扩展困难。

适用于小型或者微型，对扩展需求或其他非功能性需求要求不高的项目。



JSP Model 2 是以 servlet 为中心的，将控制层和视图层分离，解决了 Model 1 的问题。

优点：相对于 Model 1 来说，控制层和视图层的分离为维护和扩展提供了便利，同时也可以采用其他的视图层技术而非 jsp 一种。

缺点：复杂度大大提升，开发效率变低了。

适用于中大型，对扩展需求或其他非功能性需求要求高的项目。

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）

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

图 1：项目目录结构，可以看出项目名称为 SimpleController，包名为 me.king，分别有两个类，一个是 LoginController 作为控制层，一个是 UserBean 作为模型层，然后还有 3 个 jsp 页面作为视图层，分别是 login_fail.jsp，login_success.jsp 和 login.jsp

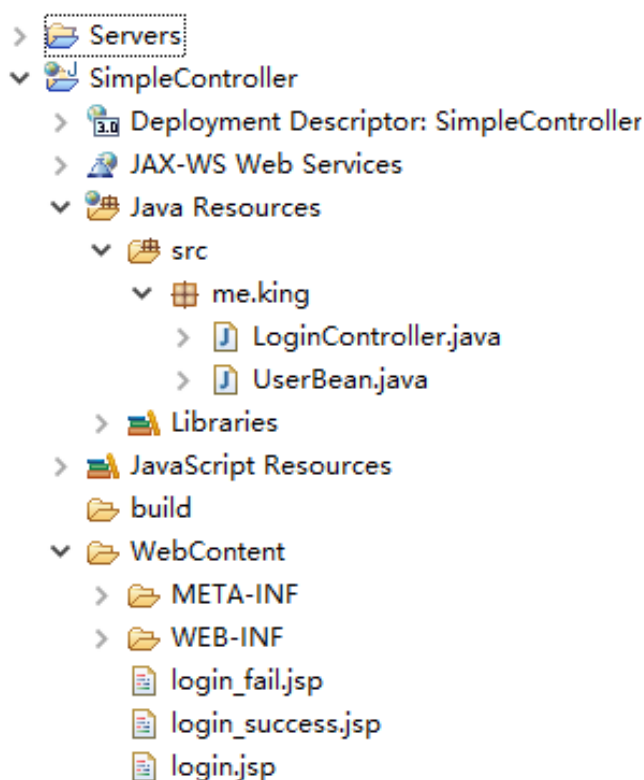


图 1：项目目录结构

图 2：LoginController 代码截图，

使用注解的方式告知容器，LoginController servlet 映射的 url 为“/LoginController”。

代码为：`@WebServlet("/LoginController")`

在 doGet 方法中，新建一个 UserBean 对象 ub，调用 ub 的 checkLogin 方法，传入 request

对象中的 `name` 和 `pwd`。若 `checkLogin` 验证用户名和密码成功返回 `true` 则设置跳转至 `login_success.jsp` 页面，否则跳转至 `login_fail.jsp` 页面。最后采用 `RequestDispatcher.forward` 将 `request` 和 `response` 两个对象返回至指定的页面。

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，也可以直接调用 `doGet()` 方法。

```

/**
 * @WebServlet("/LoginController")
 * public class LoginController extends HttpServlet {
 *     private static final long serialVersionUID = 1L;
 *
 *     /**
 *      * @see HttpServlet#HttpServlet()
 *      */
 *     public LoginController() {
 *         super();
 *         // TODO Auto-generated constructor stub
 *     }
 *
 *     /**
 *      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 *      */
 *     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
 *         // TODO Auto-generated method stub
 *         ServletContext sc = getServletContext();
 *         RequestDispatcher rd = null;
 *         UserBean ub = new UserBean();
 *
 *         if(ub.checkLogin(request.getParameter("name"), request.getParameter("pwd")))
 *             rd = sc.getRequestDispatcher("/login_success.jsp");
 *         else rd = sc.getRequestDispatcher("/login_fail.jsp");
 *
 *         rd.forward(request, response);
 *     }
 *
 *     /**
 *      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 *      */
 *     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
 *         // TODO Auto-generated method stub
 *     }
 * }

```

图 2: LoginController 代码截图

图 3: 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 `true`，否则返回 `false`。

```

package me.king;

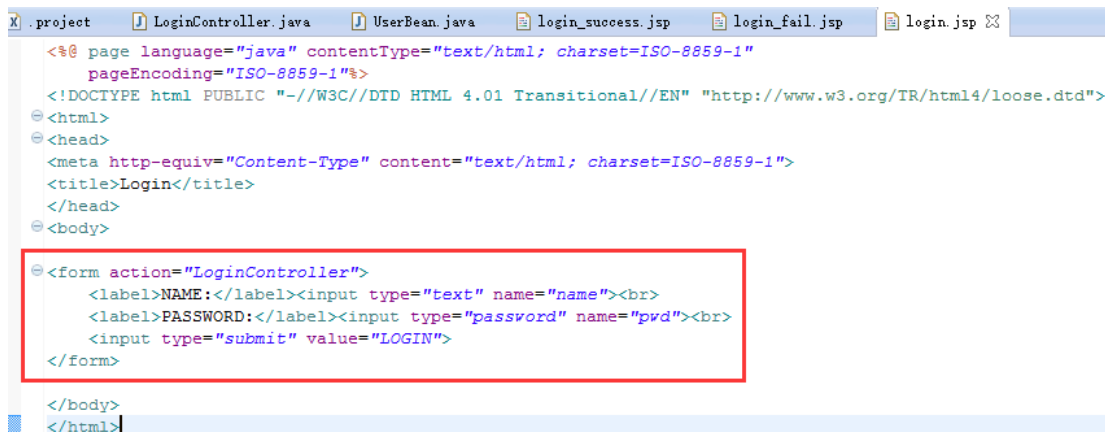
public class UserBean {
    public Boolean checkLogin(String name, String pwd){
        if(name.equals("world") && pwd.equals("hello"))
            return true;
        else return false;
    }
}

```

图 3: UserBean 代码截图

图 4.1: login.jsp 代码截图，

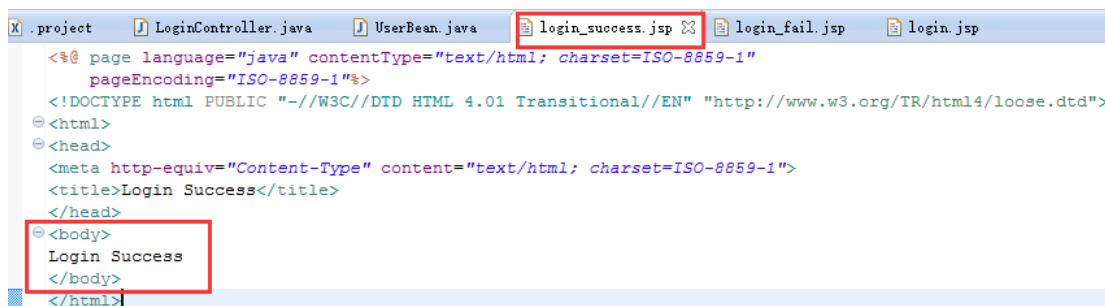
编写一个 form 表单，action 设置为 `LoginController`（就是在 `LoginController` 类中使用 `@webServlet` 注解映射的 url），设置用户名和密码的 `name` 属性为 `name` 和 `pwd`。



```
<%@ 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</title>
</head>
<body>
<form action="LoginController">
  <label>NAME:</label><input type="text" name="name"><br>
  <label>PASSWORD:</label><input type="password" name="pwd"><br>
  <input type="submit" value="LOGIN">
</form>
</body>
</html>
```

图 4.1: login.jsp 代码截图

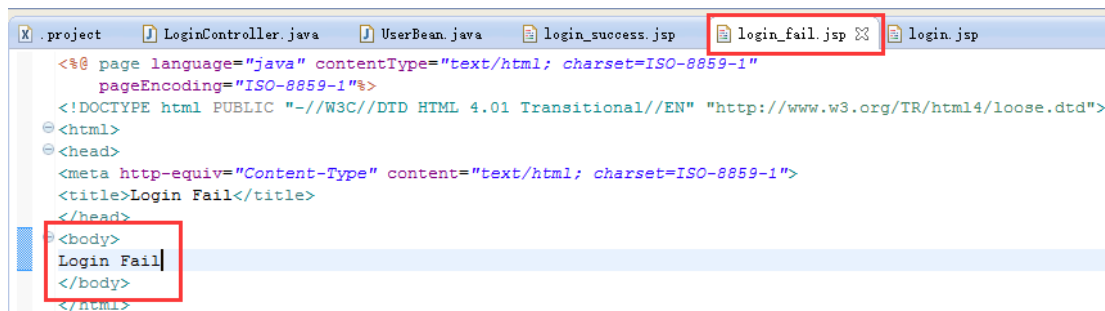
图 4.2: login_success.jsp 代码截图，
简单的在 body 中写入 Login Success。



```
<%@ 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 Success</title>
</head>
<body>
Login Success
</body>
</html>
```

图 4.2: login_success.jsp 代码截图

图 4.3: 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/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 4.3: login_fail.jsp 代码截图

图 5.1: 使用 Eclipse 内置浏览器测试截图，
可以看到 url 为: <http://localhost:8080/SimpleController/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。

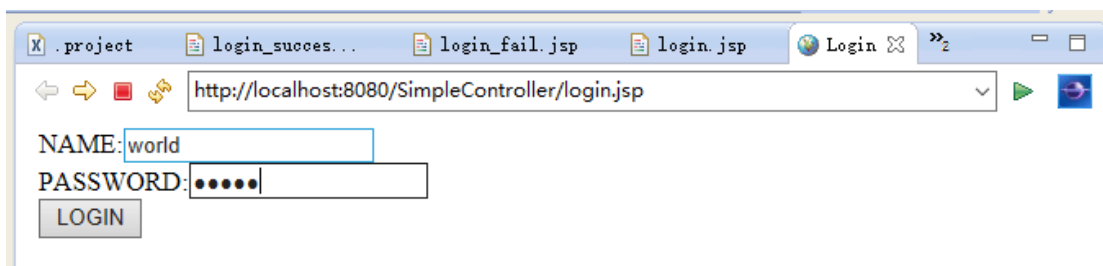


图 5.1: 使用 Eclipse 内置浏览器测试截图

图 5.2: 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 (虽然跳转至 login_success.jsp 页面, 但此时 url 没有改变为 login_success.jsp)

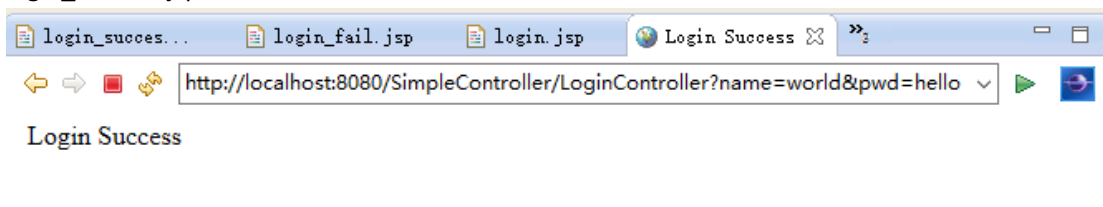


图 5.2: 登录成功截图

图 5.3: 登录失败截图,

可以通过 url 看到, 当输入的用户名 name 为 world, 密码 pwd 为 12345 时, 验证失败, 显示 Login Fail (虽然跳转至 login_fail.jsp 页面, 但此时 url 没有改变为 login_fail.jsp)

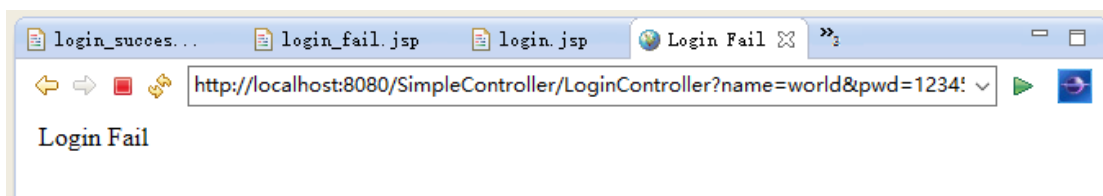


图 5.3: 登录失败截图

图 6.1: 导出为 War 包操作截图,

在项目目录中右键选择 Export --> WAR file, 然后选择导出到 Tomcat 目录下的 webapps 中即可。

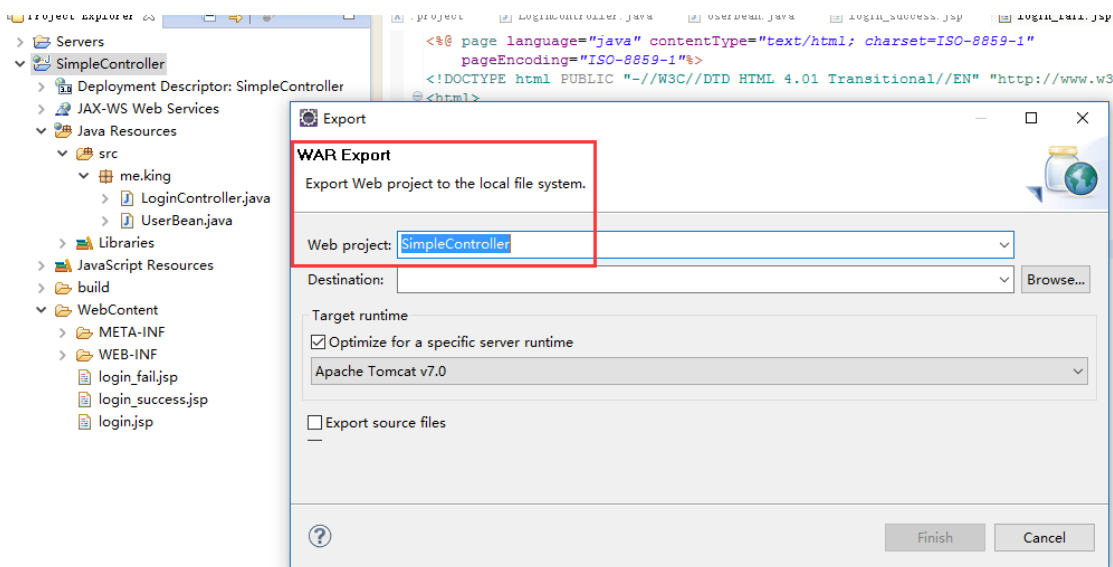


图 6.1: 导出为 War 包操作截图

图 6.2: 部署在 Tomcat 中后使用 Chrome 浏览器测试截图，
在 Tomcat 目录下 bin 中选择 startup.bat 启动 Tomcat 服务，Tomcat 会自动将 war 包解压
并将应用部署到同名文件夹下，然后在浏览器地址栏输入 SimpleController 应用的登录页
面地址：

<http://localhost:8080/SimpleController/login.jsp>

然后重复前面 图 5.* 的测试步骤即可。

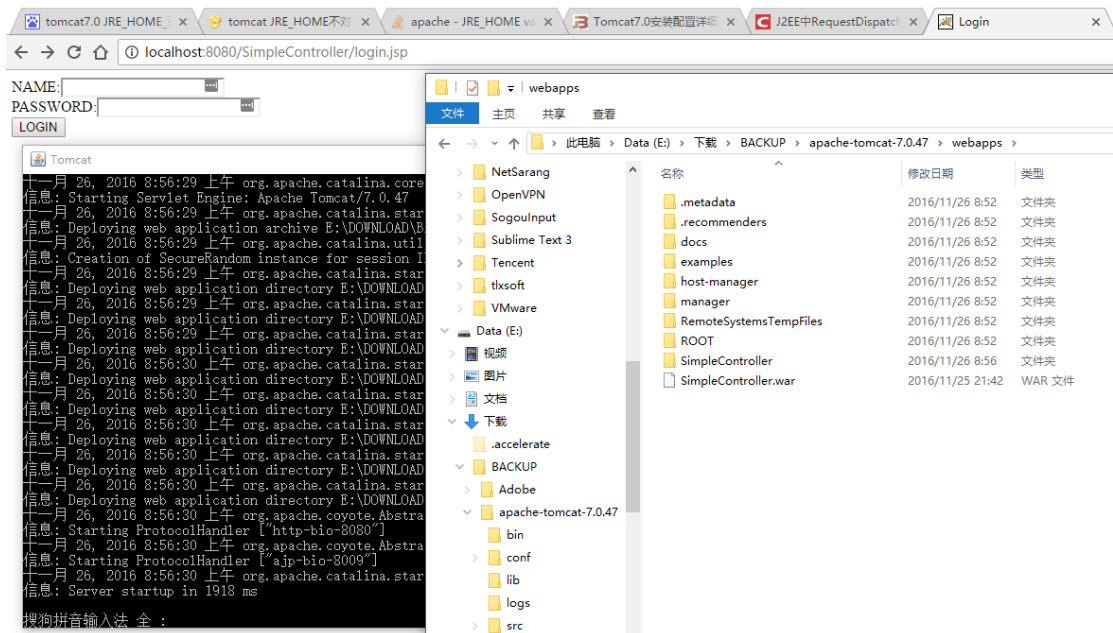


图 6.2: 部署在 Tomcat 中后使用 Chrome 浏览器测试截图

其中，图 5.2 和 5.3 浏览器地址栏 URL 没有改变的原因是：
由于在 UserBean 中采用的是 RequestDispatcher.forward() 方式，这表示在服务器端运行。

采用采用请求转发，`request` 对象始终存在，不会重新创建，前后页面共享同一个 `request` 重定向后浏览器地址栏 URL 不变。

若使用 `Response.sendRedirect()` 方式，则表示在用户的浏览器端工作。重新定向，前后页面不共享一个 `request`。重定向后在浏览器地址栏上会出现重定向页面的 URL。

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）

MVC 这样的话题其实已经是很多年前提出的概念了，但是至今依然存在而且还在继续使用，可以看出，目前虽然实现技术和业务类型日新月异，但是我们的业务本质上一直没有改变。

而对于这种“古老而经典”的技术，其实我们真的很难提出什么创造性的理论和结论了，最多能做到的就是去验证和体会，亲手编码实现这种模式，遇到一些问题，然后解决它，感受它的优点和缺点，加深印象和体会而已。

所以很多时候我们做的作业和实验就是一些基础而入门的东西，远远谈不上有什么建议和看法，我觉得踏踏实实亲手去实现，去查阅资料就已经是够了，虽然也许很多东西其实不是自己手写的，但是查阅过和直接 copy 总是会有不同的。剩下的一些东西也许得在一个大型项目才能体会了。

以下是我查到的一些资料，完成作业后，我将自己的想法写在了括号中。

MVC 相比其他设计模式有着如下的优势：

- (1) 首先，最重要的是应该有多个视图对应一个模型的能力。
- (2) 其次，由于模型返回的数据不带任何显示格式，因而这些模型也可直接应用于接口的使用。（我觉得是指数据的格式可以升级扩展，比如现在流行的 JSON，就是一种新的格式，现在很多前后端都在使用，这在以前就是 xml，但依然能自然过渡而无需对模型有什么大改变。）
- (3) 再次，由于一个应用被分离为三层，层与层之间解耦，因此有时改变其中的一层就能满足应用的改变。（就像网络 OSI 的分层结构一样，只要下层对上层提供服务不会变，上层完全不用理会下层的实现方式）
- (4) 最后，它还有利于软件工程化管理。（尤其是多人协作，分工明确可以极大的加快开发效率）

MVC 的不足体现在以下几个方面：

- (1) 增加了系统结构和实现的复杂性。（尤其是对微小型的项目，比如我们现在的 SimpleController 的项目，原本可以一个 jsp 就完成的事情，现在分为了好几个类了。但是对于中大型的项目，为了维护和扩展，分层，解耦，使用配置文件是必须的）
- (2) 视图与控制器间的过于紧密的连接。（这一点是必须的，无论任何框架都无法避免）
- (3) 视图对模型数据的低效率访问。（毕竟视图与数据之间有着很多层，即使每层的工作很简单，但相互的函数调用也会降低性能，当然不如直接对数据操作来得快。）
- (4) 一般高级的界面工具或构造器不支持 MVC 架构（这一点我觉得应该有所改变了）

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] J2EE 中 RequestDispatcher.forward()和 response.sendRedirect()的区别详谈
<http://blog.csdn.net/woshixuye/article/details/8843825>
- [2] MVC 框架
http://baike.baidu.com/link?url=2rwb4M4CYtCjFV5dIB3FGokR-YG32q92w0EE6Z0jluLu8Eit94t41BdnfRtuezMmVfmy2pnujLIKq78_BiPYBvA4HquF7vJToHsxZgelbYvama2AR1L9joPwJhak5kVY
- [3] MVC 框架的优缺点
<http://www.cnblogs.com/lsNull-Soft/articles/3266772.html>
- [4] Java HttpServlet:
https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/http/HttpServlet.html
- [5] Java RequestDispatcher
<https://docs.oracle.com/javaee/6/api/javax/servlet/RequestDispatcher.html>
- [6] JSP
<http://www.oracle.com/technetwork/java/javaee/jsp/index.html>
- [7] Html
http://www.w3schools.com/html/html_intro.asp
- [8] JSP 技术的优缺点介绍
http://blog.sina.com.cn/s/blog_aec6e4e30101dr6l.html
- [9] JavaWeb 学习篇之----容器 Request 详解
<http://blog.csdn.net/jiangwei0910410003/article/details/22925915>
- [10] WEB 请求处理三：Servlet 容器请求处理
<http://codecloud.net/12674.html>
- [11] 深入理解 ServletRequest 与 ServletResponse
<http://lavasoft.blog.51cto.com/62575/275586/>