MVC模式

本章内容：共2小节，7个知识点

- 第1节：实现MVC模式
- 第2节：MVC思考

本章目标

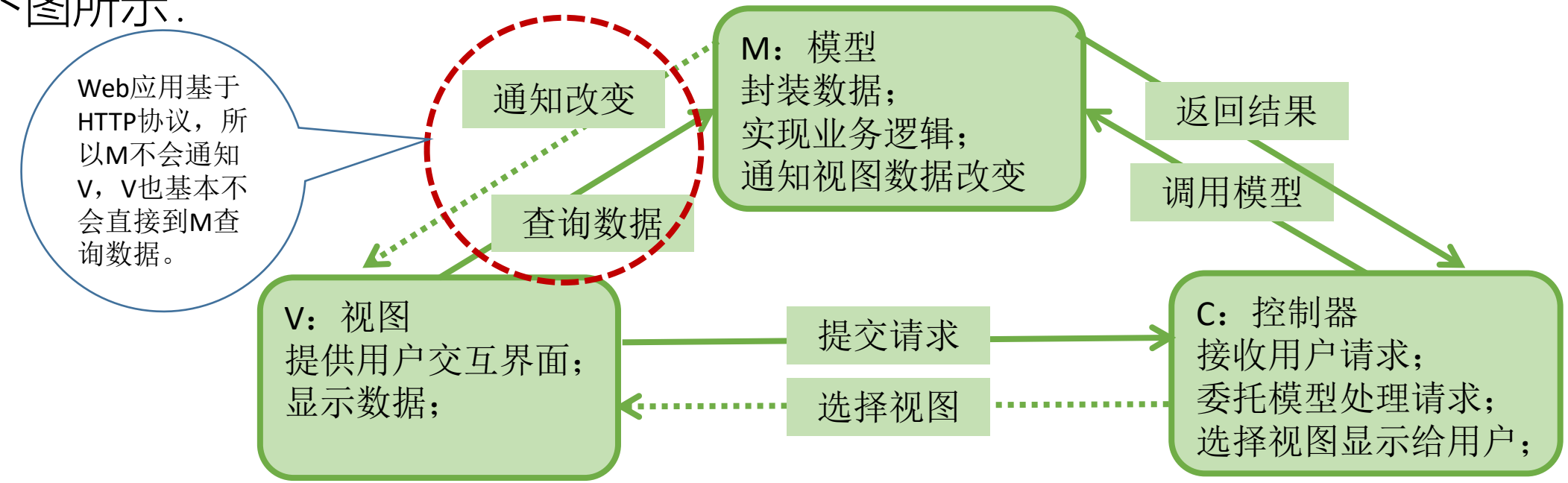
- 理解MVC模式的含义、作用；
- 能够使用JavaEE 技术构建MVC模式的Web应用；
- 掌握避免重复提交常用功能；
- 深刻理解单一外部控制器

第1节【实现MVC模式】

- 知识点1：MVC基本概念、作用、优势
- 知识点2： MVC模式中的不同角色
- 知识点3：在控制器和视图之间共享数据
- 知识点4： redirect\forward\include几种跳转方式的功能与差异

知识点1：MVC基本概念、作用、优势-1

- MVC（Model-View-Controller）是一种软件架构设计模式，最初应用在桌面应用程序；
- MVC模式将软件的代码按照模型（M）、视图（V）、控制器（C）三部分组织，如下图所示：

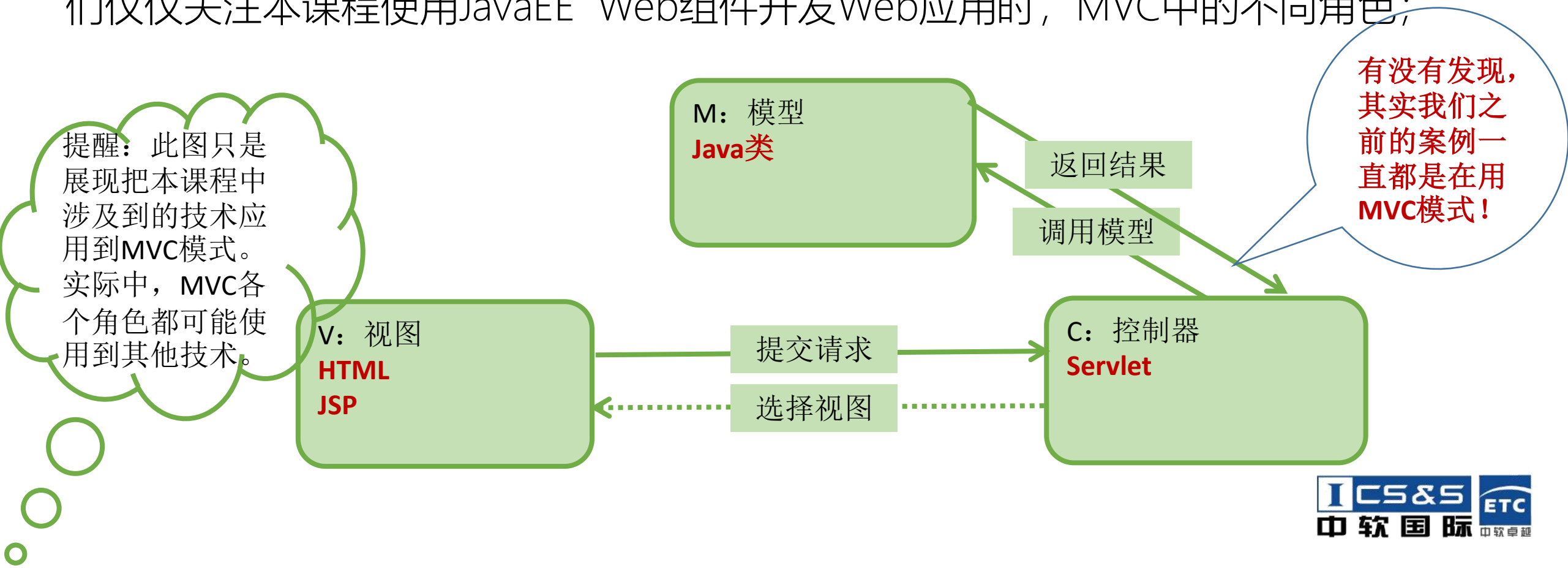


知识点1：MVC基本概念、作用、优势-2

- 使用MVC模式构建应用，具有以下优势：
  - 耦合性低：视图层和业务层分离，耦合性降低，可以独立修改；
  - 重用性高：可以用不同的视图访问模型部分，实现在不同终端上访问应用；
  - 可维护性高：视图与业务分离，降低了维护成本；

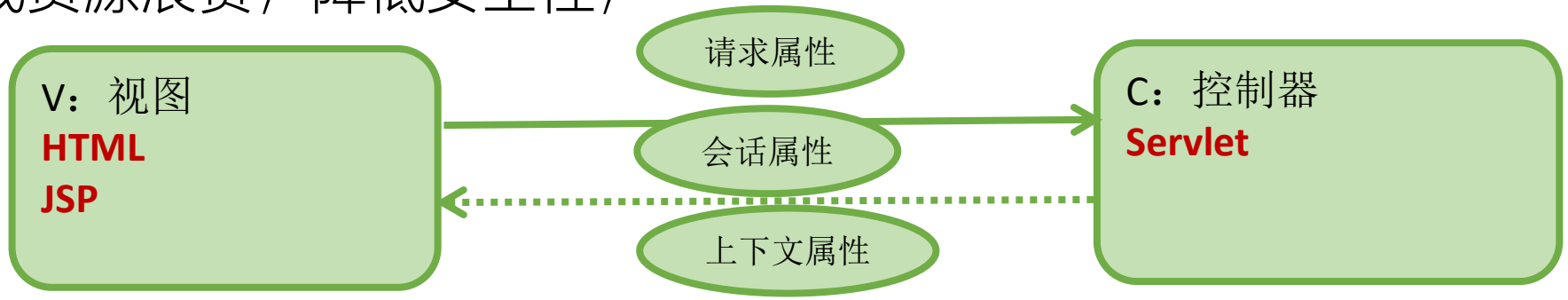
知识点2：MVC模式中的不同角色

- MVC模式中有三个角色，分别是M模型，V视图，C控制器；
- 使用不同的技术都可以实现MVC模式，我们本课程是学习JavaEE Web组件，因此我们仅仅关注本课程使用JavaEE Web组件开发Web应用时，MVC中的不同角色；



知识点3：在控制器和视图之间共享数据

- 在控制器和视图之间，常常需要共享数据；例如从数据查出来的商品列表信息，需要从控制器发送到视图；
- Servlet和JSP之间共享数据一般使用请求、会话、上下文范围的属性进行；
- HttpServletRequest/HttpSession/ServletContext接口中都定义了存取、查询、删除属性的方法【前面已经学习过】；
- 使用原则：尽量用范围小的属性，即，请求范围内共享即可就用请求，以此类推；否则会造成资源浪费，降低安全性；



知识点4：redirect\forward\include几种跳转方式的功能与差异

- MVC模式中，控制器和视图之间需要进行跳转，Servlet规范中，有三种跳转方式：
  - redirect：调用响应接口的sendRedirect方法，响应重定向，相当于重新请求新的资源，当前请求对象不会到目标资源；
  - forward: 调用请求转发器接口的forward方法，请求转发，将当前的请求、响应对象转发到目标资源；
  - include：调用请求转发器接口的include方法，动态包含，将目标资源的请求、响应对象包含到当前资源；

本节总结提问【实现MVC模式】

- MVC模式有什么作用和优势?
- 使用本课程的技术，如何实现MVC模式?
- 视图和控制器之间如何共享数据?
- 视图和控制器之间如何跳转?

本节总结【实现MVC模式】

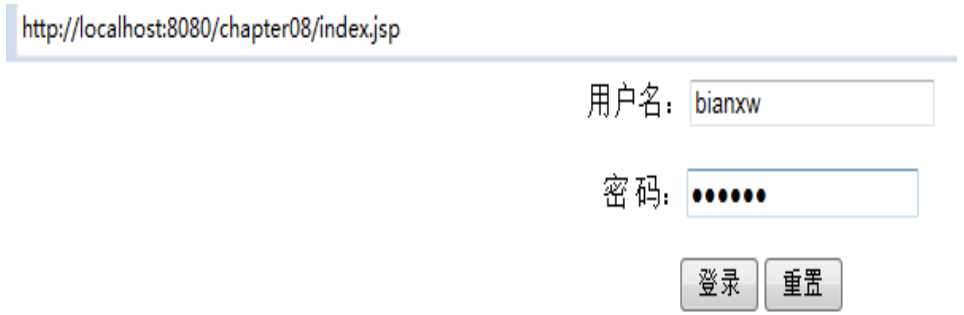
- MVC是一种软件架构设计模式，把软件的代码组织分为三个部分，分别是实现业务逻辑和封装数据的模型M，实现视图的V，用来接收请求的控制器C；
- 本课程中的技术，JSP可以用来做View，Servlet用来做Controller，Java类用来做Model；
- 控制器和视图之间可以通过请求、会话、上下文的属性来共享数据；尽量用小的范围共享数据；
- 控制器和视图之间的跳转有响应重定向、请求转发、动态包含，其中最常用的是请求转发

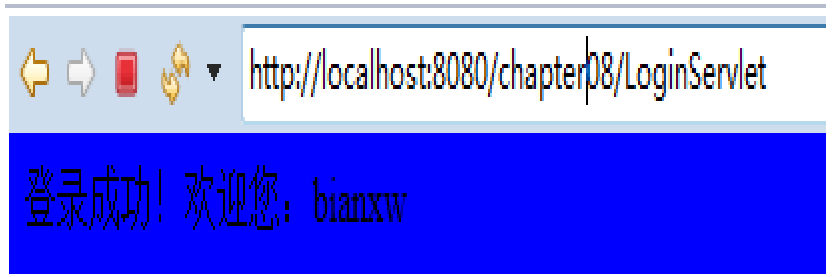
第2节【MVC思考】

- 知识点1：forward带来的重复提交问题
- 知识点2：解决重复提交
- 知识点3：单一外部控制器模式的实现

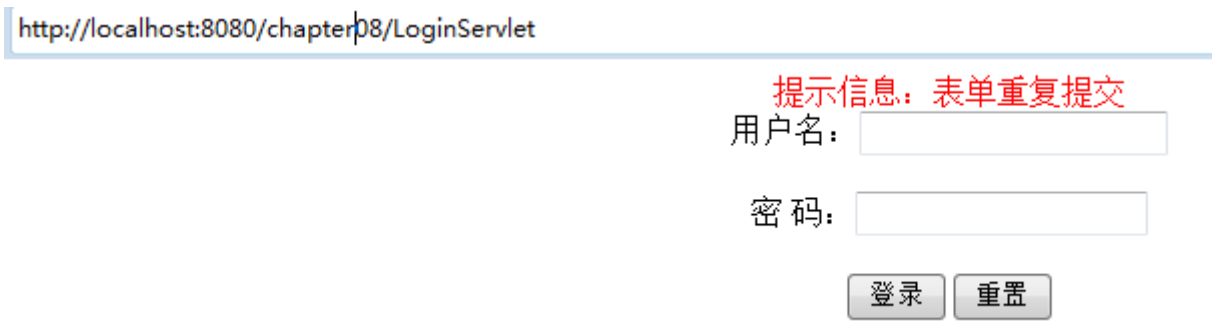
知识点1：forward带来的重复提交问题

- 使用forward转发请求后，再次刷新当前页面，会进行重复提交；
- 例如：使用LoginServlet进行登录，成功后跳转到loginsuccess.jsp页面：





- 刷新当前页面，再次直接进行了登录，出现提示信息：



如果这是支付、买票等页面，重复提交将导致严重后果。因此要解决重复提交的问题。

知识点2：解决重复提交-1

课堂案例：
index.jsp

- 为了能够解决重复提交问题，关键在于：能够标志一次提交，从而识别出该提交已经处理；
- 步骤一：在JSP中记录一个随机数，称为令牌（token），存储在session中

```
<%
        session.setAttribute("token", System.nanoTime()+"");
%>
```

- 步骤二：将token值作为表单的一个隐藏域

```
<input type="hidden" name="token" value="<%=session.getAttribute("token")%>">
```

课堂案例：
index.jsp

```
<%
		session.setAttribute("token", System.nanoTime()+"");
 %>
 <center>
		<form name="form1" action="LoginServlet" method="post">
			<%
			String msg=(String)request.getAttribute("msg");
	if(msg!=null&&!msg.equals("")) {%>
			<font color='red'>提示信息：<%=msg %></font><br>
			<%} %>
			 <input type="hidden" name="token"
value="<%=session.getAttribute("token")%>">
		用户名：<input type="text" name="username"><br>
		密 码：<input type="password" name="pwd"><br>

			<input type="submit" value="登录" ><input type="reset" value="重置" >
		</form>
</center>
```

知识点2：解决重复提交-2

课堂案例：
LoginServlet.java

- 步骤三：在LoginServlet中获取token值，并进行判断

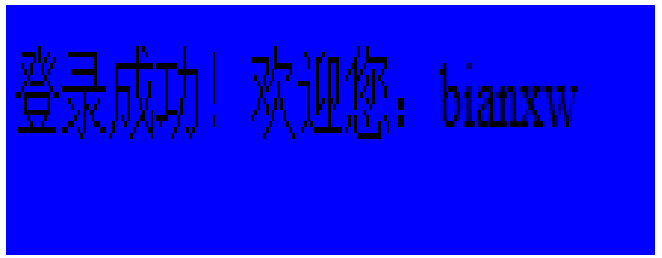
```
//取出存储在请求参数中的token
String requestToken = request.getParameter("token");
//取出存储在Session中的token
String sessionToken = (String) request.getSession().getAttribute("token");
//表单数据中没有token，表单重复提交
//Session中不存在Token，表单重复提交
//Session中的Token与表单提交的Token不同，表单重复提交
if(requestToken==null||sessionToken==null||!requestToken.equals(sessionToken)){
request.setAttribute("msg", "表单重复提交");
request.getRequestDispatcher("index.jsp").forward(request, response);
}
```

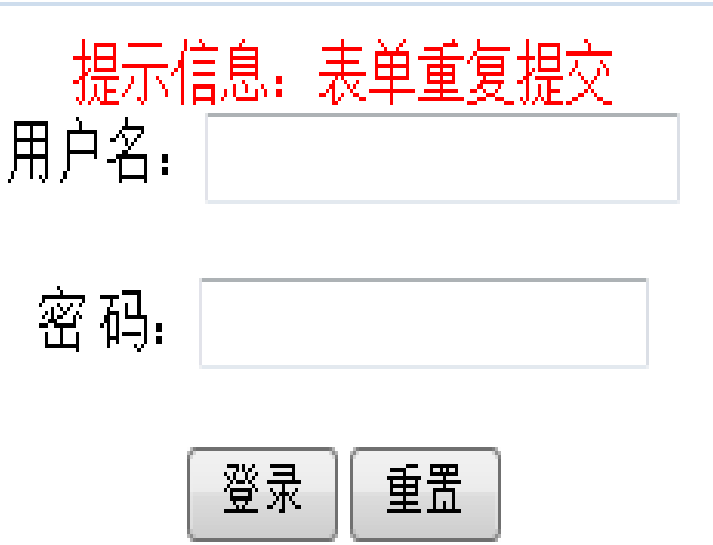
再次刷新LoginServlet所在URL，则跳转到首页，并显示“表单重复提交”。

- 步骤四：将token值从会话中删除

```
else{
        request.getSession().removeAttribute("token");//移除session中的token
        request.getRequestDispatcher("loginsuccess.jsp").forward(request, response);}
```

启动服务器访问http://localhost:8080/chapter08/index.jsp






```
<body bgcolor= "blue">
 登录成功！欢迎您：<%=request.getParameter("username") %><br>
</body>
```

loginSuccess.jsp

知识点3：单一外部控制器模式的实现-1

课堂案例：
MainServlet.java

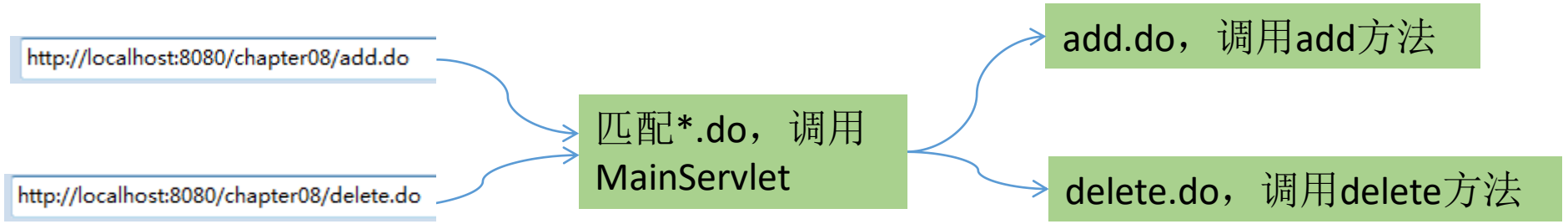
- 前面实现的示例中，都是一个功能对应一个Servlet进行处理；
- 实际应用中，可以使用一个核心Servlet接收多个请求，通过判断请求路径去处理不同的逻辑；
- 步骤一：创建MainServlet，doXXX方法中判断请求路径不同，调用不同业务逻辑方法；

```
                MainService service=new MainService();
//              获取请求路径并解析
                String url=request.getRequestURI();
                String[] strs=url.split("/");
//              请求add.do，调用add方法，请求delete.do，调用delete方法
                if(strs[strs.length-1]!=null&&strs[strs.length-1].equals("add.do")){
                        service.add();
                }
                if(strs[strs.length-1]!=null&&strs[strs.length-1].equals("delete.do")){
                        service.delete();
                }
```

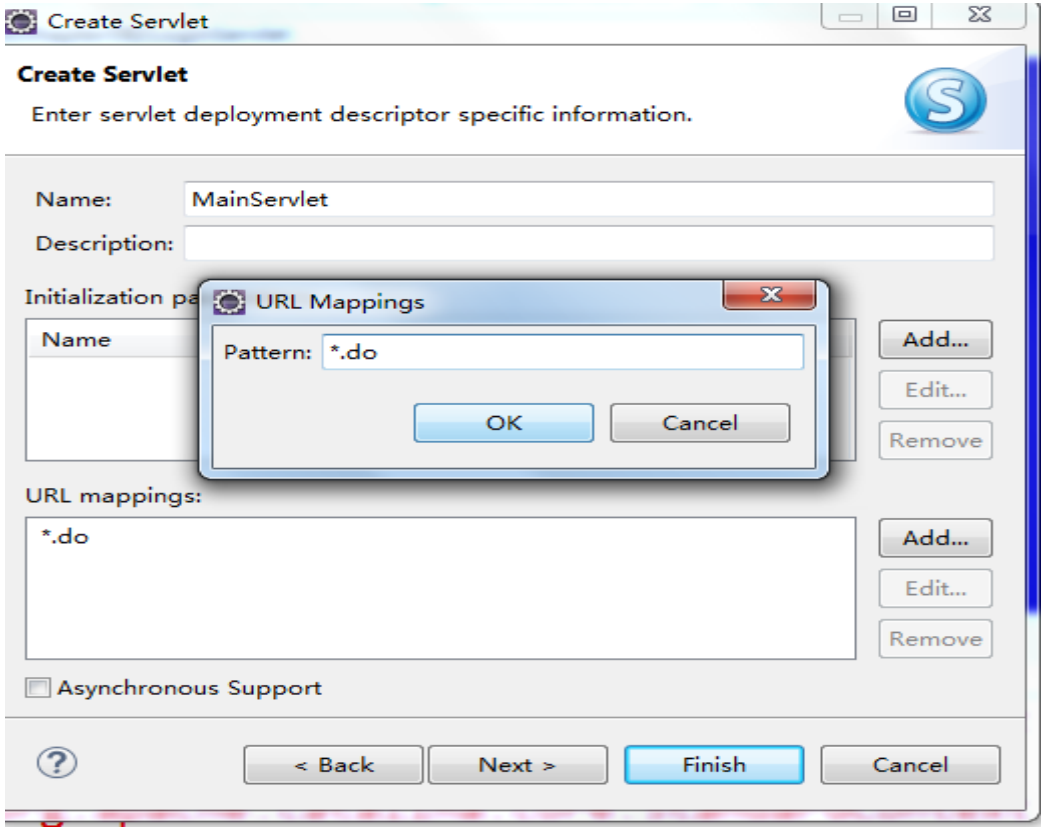
知识点3：单一外部控制器模式的实现-2

- 步骤二：web.xml中配置MainServlet的url-pattern为*.do，可以匹配所有*.do格式请求；

```
<servlet>
  <description></description>
  <display-name>MainServlet</display-name>
  <servlet-name>MainServlet</servlet-name>
  <servlet-class>com.chinasofti.chapter08.section02.MainServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>MainServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
 </servlet-mapping>
```



知识点3：单一外部控制器模式的实现-3




```
public class MainService {
    public void add(){
        System.out.println("调用添加方法");
    }

    public void delete(){
        System.out.println("调用删除方法");
    }
}
```

本节总结提问【MVC思考】

- 重复提交会带来什么问题?
- 如何解决forward带来的重复提交?
- 如何实现单一核心控制器?

本节总结【 MVC思考 】

- 重复提交会导致系统重复记录数据，比如对一张订单多次支付、一个用户多次注册等；
- forward重复提交的问题可以使用令牌解决，在请求和会话范围存令牌信息，通过判断令牌，阻止重复提交；
- 把Servlet用通配符配置url-pattern，根据判断请求路径调用不同的业务逻辑，实现业务处理；

本章总结

- MVC是Web应用中广泛使用的软件设计模式；
- M指的是模型，实现业务逻辑；V指的是视图，实现显示页面；C指的是控制器，实现控制逻辑，接收视图的请求，调用业务逻辑处理，并跳转到不同的视图显示处理结果；
- Servlet可以用来实现控制器，JSP可以用来实现视图；
- 控制器和视图之间可以通过请求、会话、上下文共享数据；
- 控制器和视图之间的跳转方式包括：重定向、请求转发、动态包含；
- 之前章节的示例都是使用MVC模式实现；

本章作业

- 作业1:
- 题目：模拟登录系统，对表单提交避免重复提交问题;
- 难度：中
- 作业2:
- 题目：分别用配置文件和注解方式实现一个单一外部控制器;
- 难度：中