重定向和请求分派

重定向和请求分派都是让别人而不是让自己（某servlet）来为请求处理响应

重定向=客户

请求分派=服务器

一、重定向(sendRedirect)

1.重定向的servlet只是调用了sendRedirect()，它的重定向任务就结束了，其HTTP响应中状态码是301，且还有一个Location首部（值是URL），浏览器发现了301状态码，并寻找Location的值，使用该URL建立一个新的请求。

可以看到，就是把重定向的任务交给浏览器（客户）去执行。

2.Servlet重定向让浏览器做工作

3.在sendRedirect()中使用**相对**URL

1）有斜线的相对URL

sendRedirect(“/foo/stuff.html”)

容器会相对于Web应用本身建立完整的URL，而不是原来的URL

**http://www.baidu.com** /foo/stuff.html

2）没有斜线的相对URL

sendRedirect(“foo/stuff.html”)

假设客户输入 http://www.baidu.com/myapp/cool/bar.do，请求到达bar.do这个servlet时，调用重定向且使用相对路径”foo/stuff.html”，则容器会**相对于**原先的请求URL（http://www.baidu.com/myapp/cool/bar.do）建立新的完整的URL（当然，既然是重定向，这个新的URL写在HTTP响应头部的location）

**http://www.baidu.com/myapp/cool/**  foo/stuff/html

一句话：剔除之前的servlet，然后加上这部分。

4.Servlet的任务就是处理请求，要么自己处理，要么转发给别人处理

5.sendRedirect()取一个String对象而不是URL对象

二、请求分派(getRequestDispatcher(…).forward(request,response))

1.请求分派是在服务器端做工作

2.RequestDisPatcher view = request.getRequestDispatcher(“result.jsp”);

View.forward(request, response);

把请求（可能已加工）转发给result.jsp，让result.jsp接管产生响应这个任务

3.客户不知道有人接管了工作，因为浏览器上的URL没有改变

4.请求分派机制示例

//将**业务逻辑结果**封装为容器内部属性对象styles，**相当于为请求添加一个参数和参数值**

request.setAttribute("styles", brandsResult);

//将HTTP请求转发

RequestDispatcher view = request.getRequestDispatcher("result.jsp");

view.forward(request, response);

控制器servlet调用jsp（对应另一个servlet）

/\*\*

\* Servlet implementation class BeerSelect

\*/

@WebServlet("/BeerSelect")

public class BeerSelect extends HttpServlet {

……

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("Beer Selection Advice<br>");

//这里是在HTTP请求中查找传递的参数，因此html中的name属性作为http参数名必须定义

String c = request.getParameter("color");

BeerExpert expert = new BeerExpert();

//将业务逻辑交给业务逻辑对象处理

List<String> brandsResult = expert.getBrands(c);

/\*

\* v3

\* 将业务逻辑结果封装在html响应的一个新的styles属性变量中，

\* 然后转发给jsp页面（该页面也有一个对应的servlet类进行处理，由web容器提供）

\*

\* 至此，就将页面表示和业务逻辑分离了

\*/

//将结果封装为容器内部属性对象styles

request.setAttribute("styles", brandsResult);

//将HTTP请求转发

RequestDispatcher view = request.getRequestDispatcher("result.jsp");

view.forward(request, response);

}

}

5. <!-- 要记住JSP对应的是另一个servlet，

JSP有点被动，要等待控制器Servlet来调用 -->

可以吧JSP看成一个专门负责处理页面展示的servlet

6.RequestDispatcher只有两个方法 forward()和include()

1）forward()将请求完全转发给另一个，include()转发给别人完成工作后再返回发送者。

前者比较常用。

2）RequestDispatcher的两个来源

（1）从ServletRequest得到

RequestDispatcher view = request.getRequestDispatcher(“result.jsp”);

参数是String，请求转发到资源要指定这个资源的路径。

如果路径前面有个/,容器会把它看成从这个Web应用的根开始；因为，如果路径不是以斜线开头，则认为路径相对于原来的请求位置。注意路径只能在Web应用内找，超出就无法工作。

（2）从ServletContext得到

RequestDispatcher view = getServletContext().getRequestDispatcher(“/jsp/result.jsp”)

注意使用上下文的请求分派器只能使用指定资源的路径，不能使用相对路径，也就是说路径必须从根开始。

7.在RequestDispatcher上调用forward

view.forward(request, response);

1）注意一点：在响应提交后，就不能再转发请求了！，否则报错（合理）

例如：

os.flush(); //这里已将响应提交

RequestDispatcher view = getServletContext().getRequestDispatcher(“/jsp/result.jsp”)

view.forward(request, response);

os.close();

这样会报错IIIegalStateException