

java web

张海宁¹

April 20, 2018

¹hnzhang1@gzu.edu.cn

Contents

1	servlet	1
1.1	servlet	1
1.1.1	jsp 页面	1
1.1.2	servlet 代码	1
1.2	filter	2
1.2.1	建立 filter	2
1.2.2	部署 filter	2
1.3	listener	5
1.3.1	新建 listener	6
1.3.2	部署 listener	6

Chapter 1

servlet

servlet 是运行在 web 容器中的 java 程序，很适合用来进行处理 web 应用中的业务逻辑。

通过 servlet 和 jsp 的配合使用，可以使其各负其责：

- jsp
负责显示页面
- servlet
负责处理业务逻辑

jsp 和 servlet 的使用场景就是：jsp 负责页面显示、与用户交互，jsp 页面中一些需要处理的数据就送到 servlet 来处理。

1.1 servlet

通过编写一个负责处理注册信息的 servlet 来展示 servlet 的使用方法。

在 eclipse 中，右键当前的项目，选择新建，找到 servlet，即可建立一个 servlet(这是一个 java 类)。

建立完成后，打开刚才建立的 servlet，可以看到其中会有一些 eclipse 自动填写的代码，我们关注的是其中的两个方法 doGet 和 doPost。一般来说，servlet 是用来处理 jsp 页面传递过来的数据，而传递数据的方式通常情况下是通过表单来进行的，表单中的数据提交通常使用的是 post 方法。本部分涉及到两个文件：teacher.jsp 和 Reg.java，其中 teacher.jsp 负责页面的显示以及数据的收集，Reg.java 负责处理 teacher.jsp 页面传递过来的数据。

1.1.1 jsp 页面

teacher.jsp 页面中的关键代码如列表1.1所示。从列表1.1中可以看出，当前 form 的 action 是指向的 teacher_add.jsp 页面，这是在学习 sevlet 之前的数据处理方式，从现在开始需要把 action 的值改为新建的 servlet，即：action="Reg"。

1.1.2 servlet 代码

因为 teacher.jsp 页面中 form 的数据提交方式为 post，所以我们要完成 Reg.java 中的 doPost() 方法。Reg.java 中的关键代码如列表1.2所示。

```

<form action="teacher_add.jsp" method="post">
<table>
<tr>
  <td>姓名: </td>
  <td><input type="text" name="nm"></td>
</tr>
<tr>
  <td>职工号: </td>
  <td><input type="number" name="staffid" min=20060001 ></td>
</tr>
<tr>
  <td><input type="submit"></td>
  <td><input type="reset"></td>
</tr>
</table>
</form>

```

Figure 1.1: teacher.jsp 中的关键代码

1.2 filter

filter 是 servlet 规范中定义的一种特殊类。filter 可以理解成介于客户端和目标资源之间的一个过滤器，即它会对客户端的请示进行过滤后才可以到达服务器上的目标资源，或者访问到目标资源后，对服务器端产生的响应进行处理后才送回客户端（这两个活动可以在一个过滤器中同时进行，即双向过滤）。filter 的示意图如 Figure 1.3所示。在用户注册的时候，如果想禁止某个姓名被使用，可以通过部署一个 filter 介于注册页面和业务逻辑处理模块之间。接下来的例子中，注册页面为 teacher.jsp，业务逻辑处理模块为 Reg.java（**请注意这是一个 servlet**）。

1.2.1 建立 filter

与建立一个 servlet 的方法类似，可以建立一个 filter，将其命名为 FilterReg.java。可以看到这个新建的 java 文件中，有若干方法，其中的 doFilter 方法是我们需要关注的，因为过滤功能就是在此处实现的。Filter 文件 FilterReg.java 的代码如 Figure 1.4所示。Figure 1.4中的代码会判断用户提交的姓名，如果姓名为“zq”，则会截断请求，使得用户请求不能到达 Reg 这个 servlet，从而不能完成注册，达到了禁止特定用户名注册的目的。

1.2.2 部署 filter

在上一小节中，讲述了如何建立一个 filter，其实要使 filter 发挥作用还必须进行正确的配置。从 servlet 3.0 开始就不需要在 web.xml 文件中进行配置了，只需要在 servlet 和 filter 的源代码文件中进行声明即可，如 Figure 1.5和 Figure 1.6所示。

特别要注意的是 filter 和 servlet 里声明的 urlPatterns 需要保持一致：filter 里的 @WebFilter(filterName = "/FilterReg",urlPatterns = "/Reg") 和 servlet 里的 @WebServlet(name="reg",urlPatterns= "/Reg")。

```

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // 设置request传递过来值的编码, 并获取传递值
    request.setCharacterEncoding("utf-8");

    String id = request.getParameter("staffid");
    String name = request.getParameter("nm");

    //get current date and time
    //LabDate ld = new LabDate();
    //String time = ld.getDtTm();

    //write to database
    //String[] fields= {"id","name","logDate"};
    //String[] values= new String[3];
    //values[0]=id;
    //values[1]=name;
    //values[2]=time;
    //Db db = new Db();
    //int i = db.writeDb("teachers", fields, values);
    //db.getClose();
    //String rz="";
    //if(i==1) {
    //    rz = "done! Will return to the former page in 3 seconds.";
    //} else {
    //    rz = "Something wrong! Will return in 3 seconds.";
    //}
    //response.getWriter().print(rz);
    //response.setHeader("refresh","3,URL=teacher.jsp");
}

```

Figure 1.2: Reg.java 中的关键代码

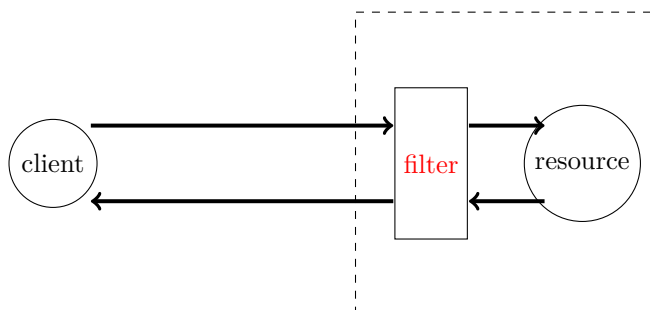


Figure 1.3: the position and function of a filter

```

public void doFilter(ServletRequest request,
    ServletResponse response, FilterChain chain)
    throws IOException, ServletException {
    // TODO Auto-generated method stub
    // place your code here
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse resp = (HttpServletResponse) response;
    String id = request.getParameter("staffid");
    String name = request.getParameter("nm");
    //System.out.println("staffid is: "+id);
    System.out.println("filter says: the name is "+name);
    if(name.equals("zq")) {
        resp.getWriter().println("The name "+name+" is forbidden!");
    }else {
        // pass the request along the filter chain
        chain.doFilter(request, response);
        resp.setHeader("refresh", "3;URL=teacher.jsp");
    }
}

```

Figure 1.4: FilterReg.java 中的关键代码

```

package cs;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Reg
 */
@WebServlet(name="reg", urlPatterns= {"/Reg"})
public class Reg extends HttpServlet {

```

Figure 1.5: servlet 中的声明 @WebServlet


```
package filter;

import java.io.IOException;
import javax.servlet.DispatcherType;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet Filter implementation class FilterReg
 */
@WebFilter(filterName = "/FilterReg",urlPatterns = {"/Reg"})
```

Figure 1.6: filter 中的声明 @WebFilter

Listener 接口	Event 类
ServletContextListener	ServletContextEvent
ServletContextAttributeListener	ServletContextAttributeEvent
HttpSessionListener	HttpSeesionEvent
HttpSessionActivationListener	
HttpSessionAttributeListener	HttpSessionBindingEvent
HttpSessionBindingListener	
ServletRequestListener	ServletRequestEvent
ServletRequestAttributeListener	ServletRequestAttributeEvent

Table 1.1: Listener 接口与 Event 类

1.3 listener

listener 是 servlet 规范中定义的一种特殊类。listener 是监听器，其作用就是用来监听 servlet 容器中一些事件 (event) 的发生。从大的分类上来说，listener 可以监听以下三个对象的事件：

- 1. ServletContex
- 2. HttpSession
- 3. ServletRequest

listener 和 event 的对应关系，如 Table 1.1所示。

监听器通常可以被用来统计在线人数。接下来以此为例展示监听器的使用方法。

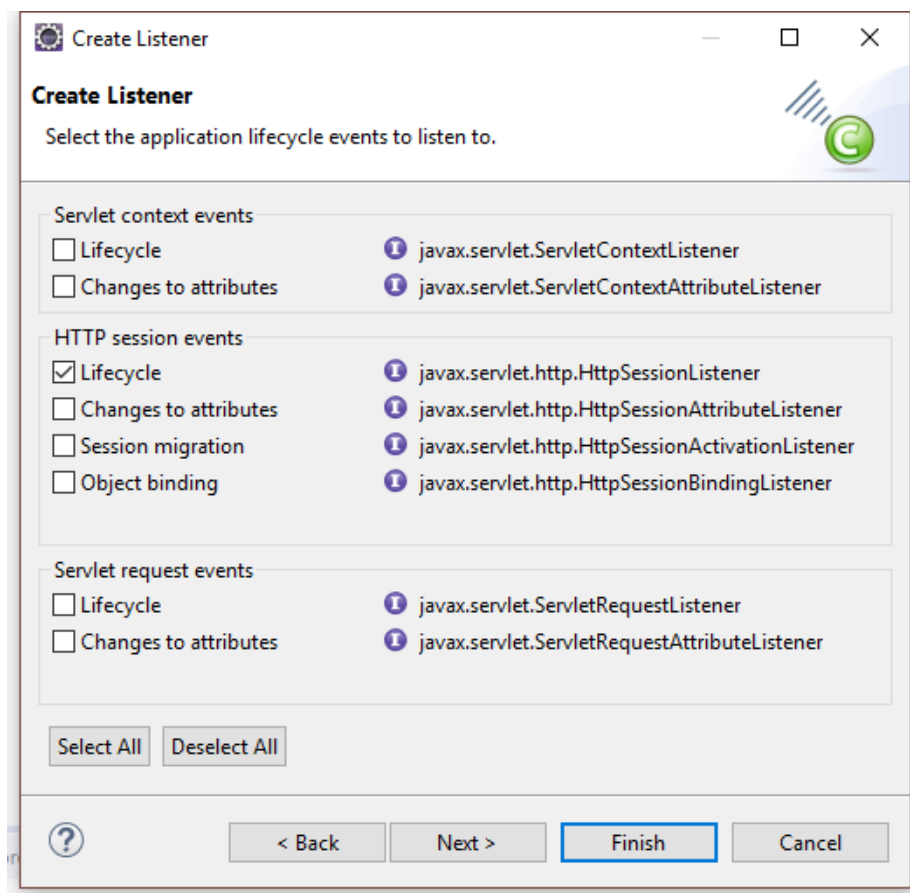


Figure 1.7: 选择 listener 监听事件

1.3.1 新建 listener

与 servlet 和 filter 的新建方式一样，新建一个 listener。这里需要注意的是，在下一步选择监听事件的时候，选择 HttpSessionEvent 里面的 lifecycle，如 Figure 1.7所示，因为这里统计在线人数是通过统计当前活动的 session 来计数的。

其代码如 Figure 1.8所示。

1.3.2 部署 listener

listener 的部署很简单，不需要另外配置，在创建 listener 的时候 eclipse 会自动帮助创建一条注释：@WebListener，这条注释就完成了 listener 的部署。

通常，每个 http 请求都会与一个 session 关联在一起。如果该请求没有与之相关联的 session，那么服务器会为其创建一个 session，此时会触发 sessionCreated 事件；该 session 会在用户不再活动的一定时间之后失效，或者用户主动注销（调用 session.invalidate() 方法），此时会触发 sessionDestroyed 事件，这两个事件会被监听器监听到。

```

package listener;

import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

import cs.LabDate;
/**
 * Application Lifecycle Listener implementation class OnlineNum
 *
 */
@WebListener
public class OnlineNum implements HttpSessionListener {
    private int count;

    public int getNum() {
        return count;
    }

    public synchronized void setCount(int c) {
        count+=c;
    }

    /**
     * Default constructor.
     */
    public OnlineNum() {
        // TODO Auto-generated constructor stub
        count=0;
        System.out.println("当前在线人数被初始化为0");
    }

    /**
     * @see HttpSessionListener#sessionCreated(HttpSessionEvent)
     */
    public void sessionCreated(HttpSessionEvent arg0) {
        // TODO Auto-generated method stub
        String dt = new LabDate().getDtTm();
        String sid = arg0.getSession().getId();
        // count++;
        setCount(1);
        System.out.println("at "+dt+", "+sid+" coming, 当前在线人数为: "+count);
    }

    /**
     * @see HttpSessionListener#sessionDestroyed(HttpSessionEvent)
     */
    public void sessionDestroyed(HttpSessionEvent arg0)
    {
        // TODO Auto-generated method stub
        // count--;
        setCount(-1);
        String dt = new LabDate().getDtTm();
        String sid = arg0.getSession().getId();
        System.out.println("at "+dt+", "+sid+" leave, 当前在线人数为: "+count);
    }
}

```

Figure 1.8: listener 代码