优极限

“极限教育，挑战极限”

www.yjxxt.com

极限教育，挑战极限。优极限是一个让 95%的学生年薪过 18 万的岗前培训公司，让我们的学员具备优秀的互联网技术和职业素养，勇攀高薪，挑战极限。公司位于上海浦东，拥有两大校区，共万余平。累计培训学员超 3 万名。我们的训练营就业平均月薪 19000，最高年薪 50 万。

核心理念：让学员学会学习，拥有解决问题的能力，拿到高薪职场的钥匙。

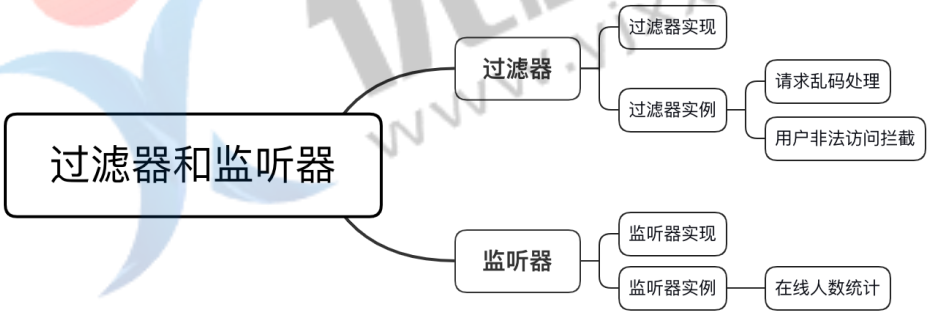
项目驱动式团队协作、一对一服务、前瞻性思维、教练式培养模型-培养你成为就业明星。首创的老学员项目联盟给学员充分的项目、技术支撑，利用优极限平台这根杠杆，不断挑战极限，勇攀高薪，开挂人生。

扫码关注优极限微信公众号：

（获取最新技术相关资讯及更多源码笔记）

过滤器和监听器

主要内容

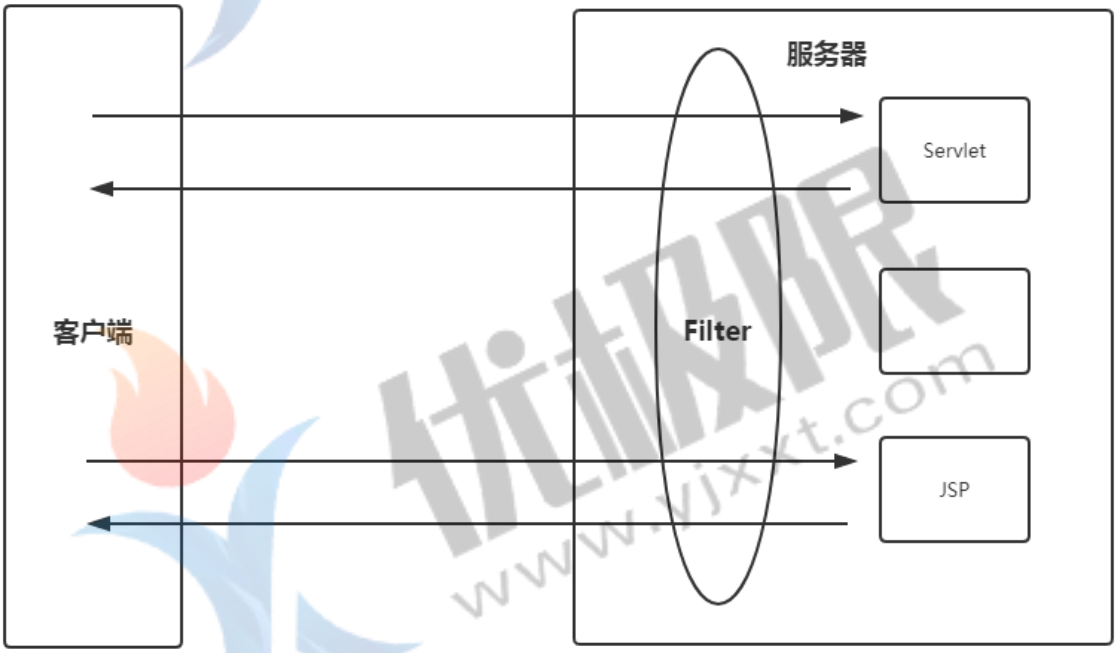


过滤器

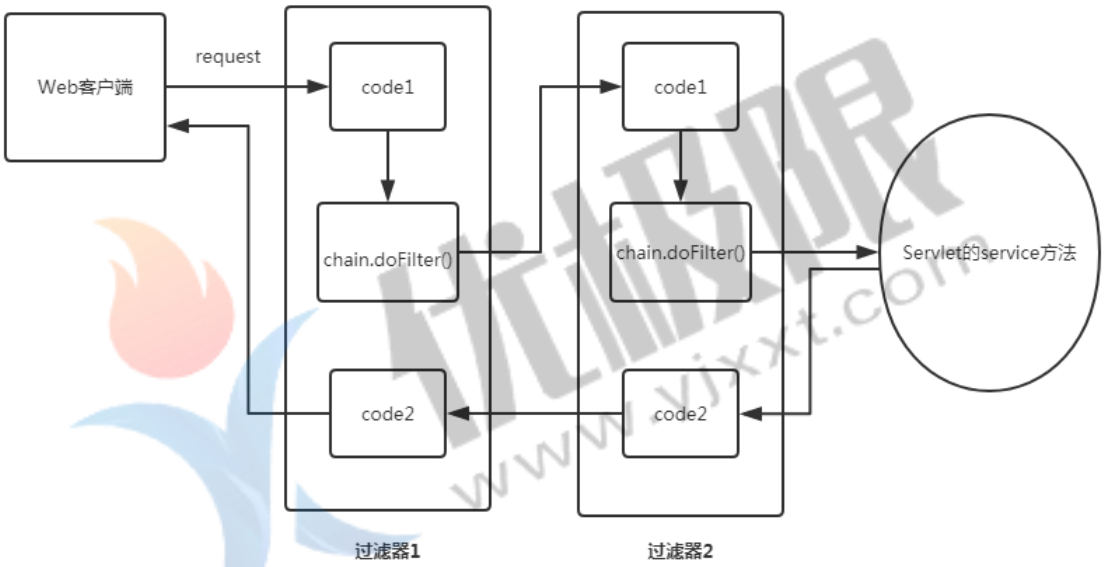
介绍

Filter 即为过滤，用于在 Servlet 之外对 Request 或者 Response 进行修改。它主要用于对用户请求进行预处理，也可以对 HttpServletResponse 进行后处理。使用 Filter 的完整流程： Filter 对用户请求进行预处理，接着将请求交给 Servlet 进行处理并生成响应，最后 Filter 再 对服务器响应进行后处理。在一个 web 应用中，可以开发编写多个 Filter，这些 Filter 组合 起来称之为一个 Filter 链。

单个过滤器



多个过滤器



若是一个过滤器链：先配置先执行(请求时的执行顺序)；响应时: 以相反的顺序执行。

在 HttpServletRequest 到达 Servlet 之前，拦截客户的 HttpServletRequest 。根据需要检查 HttpServletRequest，也可以修改 HttpServletRequest 头和数据。

在HttpServletResponse 到达客户端之前，拦截 HttpServletResponse。根据需要检查 HttpServletResponse，也可以修改 HttpServletResponse头和数据。

实现

可以通过实现一个叫做javax.servlet.Fileter的接口来实现一个过滤器，其中定义了 三个方法，init(), doFilter(), destroy()分别在相应的时机执行。后期观察生命周期。

Filter 的实现只需要两步：

Step1: 编写 java 类实现 Filter 接口，并实现其 doFilter 方法。

Step2: 通过@WebFilter注解设置它所能拦截的资源。

```
@WebFilter("/*")
public class Filter01 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {

    }

    @Override
    public void destroy() {

    }
```

```
}
```

Filter 接口中有一个 doFilter 方法，当开发人员编写好 Filter，并配置对哪个 web 资源进行拦截后，Web 服务器每次在调用 web 资源的 service 方法之前，都会先调用一下 filter 的 doFilter 方法。因此可以达到如下效果：

调用目标资源之前，让一段代码执行。

是否调用目标资源（即是否让用户访问 web 资源）。

web 服务器在调用 doFilter 方法时，会传递一个 filterChain 对象进来，filterChain 对象是 filter 接口中最重要的一个对象，它提供了一个 **doFilter** 方法，开发人员可以根据需求决定 是否调用此方法，调用该方法，则 web 服务器就会调用 web 资源的 service 方法，即 web 资源就会被访问，否则 web 资源不会被访问。（本质是放行，调用doFilter方法后，即请求可以到达资源）

实例

请求乱码处理

```
/**
 * 字符乱码处理
 *  乱码情况：
                    Tomcat8及以上版本                                   Tomcat7及以下版本

        POST请求      乱码，需要处理                                    乱码，需要处理
                    request.setCharacterEncoding("UTF-8");

        GET请求
                    不会乱码，不需要处理                                  乱码，需要处理
            new String(request.getParameter("参数名").getBytes("ISO-8859-1"),"UTF-
8");


        如何处理：
            1、处理POST请求
                request.setCharacterEncoding("UTF-8");
            2、处理GET请求且服务器版本在Tomcat8以下的
                1> 得到请求类型 （GET请求）
                2> 得到服务器的版本的信息
                3> 判断是GET请求且Tomcat版本小于8
                4> 处理乱码
                  new String(request.getParameter("参数名").getBytes("ISO-8859-
1"),"UTF-8");

 */
@WebFilter("/*")
public class AEncodingFilter implements Filter {


    public AEncodingFilter() {
    }

    public void destroy() {
    }
```

```
    public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain
chain) throws IOException, ServletException {
        // 基于HTTP
        HttpServletRequest request = (HttpServletRequest) arg0;
        HttpServletResponse response = (HttpServletResponse) arg1;

        // 处理请求乱码乱码 （处理POST请求）
        request.setCharacterEncoding("UTF-8");

        // 处理GET请求且服务器版本在Tomcat8以下的
        String method = request.getMethod();
        // 如果是GET请求
        if ("GET".equalsIgnoreCase(method)) {
            // 服务器版本在Tomcat8以下的 Apache Tomcat/8.0.45
            String serverInfo = request.getServletContext().getServerInfo();
            // 得到具体的版本号
            String versionStr = serverInfo.substring(serverInfo.indexOf("/")+1,
serverInfo.indexOf("."));
            // 判断服务器版本是否小于8
            if (Integer.parseInt(versionStr) < 8) {
                // 得到自定义内部类 （MyWapper继承了HttpServletRequestWapper对象，而
HttpServletRequestWapper对象实现了HttpServletRequest接口，所以MyWapper的本质也是
request对象）
                HttpServletRequest myRequest = new MyWapper(request);
                // 放行资源
                chain.doFilter(myRequest, response);
                return;
            }
        }

        // 放行资源
        chain.doFilter(request, response);
    }

    public void init(FilterConfig fConfig) throws ServletException {

    }


    /**
     * 定义内部类，继承HttpServletRequestWrapper包装类对象，重写getParameter()方法
     */
    class MyWapper extends HttpServletRequestWrapper {

        // 定义成员变量，提升构造器 中的request对象的范围
        private HttpServletRequest request;

        public MyWapper(HttpServletRequest request) {
            super(request);
            this.request = request;
        }

        /**
         * 重写getParameter()方法
         */
        @Override
        public String getParameter(String name) {
```

```
            String value = request.getParameter(name);

            if (value != null && !"".equals(value.trim())) {
                try {
                    // 将默认ISO-8859-1编码的字符转换成UTF-8
                    value = new String(value.getBytes("ISO-8859-1"),"UTF-8");
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
            }
            return value;
        }
    }
}
```

用户非法访问拦截

```
/**
 * 非法访问拦截（当用户未登录时，拦截请求到登录页面）
 *      拦截的资源：
 *          拦截所有资源   /*
 *      需要被放行的资源：
 *          不需要登录即可访问的资源
 *          1、放行指定页面，不需要登录可以访问的页面 （例如：登录页面、注册页面等）
 *          2、放行静态资源（例如：css、js、image等资源）
 *          3、放行指定操作，不需要登录即可执行的操作（例如：登录操作、注册操作等）
 *          4、登录状态放行 （如果存在指定sessuin对象，则为登录状态）
 */
@WebFilter("/*")
public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain
chain) throws IOException, ServletException {

        // 基于HTTP
        HttpServletRequest request = (HttpServletRequest) arg0;
        HttpServletResponse response = (HttpServletResponse) arg1;
        // 得到请求的路径
        String path = request.getRequestURI(); //    站点名/资源路径
        // 1、放行指定页面，不需要登录可以访问的页面 （例如：登录页面、注册页面等）
        if (path.contains("/login.jsp") || path.contains("/register.jsp")) {
             chain.doFilter(request, response);
             return;
        }
        // 2、放行静态资源（例如：css、js、image等资源）
        if (path.contains("/js")) {
             chain.doFilter(request, response);
             return;
        }
        // 3、放行指定操作，不需要登录即可执行的操作（例如：登录操作、注册操作等）
        if (path.contains("/loginServlet")) {
             chain.doFilter(request, response);
             return;
        }
        // 4、登录状态放行 （如果存在指定sessuin对象，则为登录状态）
        // 得到session域对象
        String uname = (String) request.getSession().getAttribute("user");
```

```
        // 如果session域对象不为空，则为登录状态，放行资源
        if (uname != null && !"".equals(uname.trim())) {
            chain.doFilter(request, response);
             return;
        }

        // 若以上条件均不满足，拦截跳转到登录页面
        response.sendRedirect("login.jsp");
        return;
    }
```

监听器

介绍

web 监听器是Servlet 中一种的特殊的类，能帮助开发者监听 web 中的特定事件， 比如 ServletContext，HttpSession，ServletRequest 的创建和销毁；变量的创建、销毁和修改等。 可以在某些动作前后增加处理，实现监控。例如可以用来统计在线人数等。

实现

监听器有三类8种：

(1) 监听生命周期：

ServletRequestListener

HttpSessionListener

ServletContextListener

(2) 监听值的变化：

ServletRequestAttributeListener

HttpSessionAttributeListener

ServletContextAttributeListener

(3) 针对 session 中的对象:

监听 session 中的 java 对象(javaBean) ，是 javaBean 直接实现监听器 的接口。

示例

做一个对在线人数的监控。

实现步骤：

Step1：创建一个监听器，需要实现某种接口，根据需求选取 HttpSessionListener

Step2：通过@WebListener注解配置该监听器

创建一个类，并实现 HttpSessionListener 接口，用来检测 Session 的创建和销毁。

1.在类中定义一个成员变量用来存储当前的 session 个数。（OnlineListener.java）

```
/**
 * 在线人数统计
 *  当有新的session对象被创建，则在线人数+1；
 *  有session对象被销毁，在线人数-1；
 * @author Lisa Li
 *
 */
@WebListener
public class OnlineListener implements HttpSessionListener {

    // 默认在线人数
    private Integer onlineNumber = 0;

    /**
     * 当有新的session对象被创建，则在线人数+1；
     */
    @Override
    public void sessionCreated(HttpSessionEvent se) {
        // 人数+1
        onlineNumber++;
        // 将人数存到session作用域中
        // se.getSession().setAttribute("onlineNumber", onlineNumber);
        // 将人数存到application作用域中
        se.getSession().getServletContext().setAttribute("onlineNumber",
onlineNumber);
    }

    /**
     * 有session对象被销毁，在线人数-1；
     */
    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        // 人数-1
        onlineNumber--;
        // 将人数存到session作用域中
        // se.getSession().setAttribute("onlineNumber", onlineNumber);
        // 将人数存到application作用域中
        se.getSession().getServletContext().setAttribute("onlineNumber",
onlineNumber);
    }

}
```

2.做一个测试的 Servlet 用来登录，和显示当前在线人数。（OnlineServlet.java）

```
/**
 * 在线人数统计
 */
public class OnlineServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

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

        // 得到参数
```

```
        String key = request.getParameter("key");

        // 判断是否为空 (不为空，且值为logout则为退出操作)
        if (key != null && "logout".equals(key)) {
            // 传递了参数，表示要做用户退出操作
            request.getSession().invalidate();
            return;
        }

        // 创建session对象
        HttpSession session = request.getSession();
        // 获取sessio作用域中的在线人数
        Integer onlineNumber = (Integer)
session.getServletContext().getAttribute("onlineNumber");

        // 输出
        response.setContentType("text/html;charset=UTF-8");
        response.getWriter().write("<h2>在线人数："+onlineNumber+"</h2><h4><a
href='online?key=logout'>退出</a><h4>");

    }
}
```