
Sun
microsystems


ВЛАДИМИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
1958

Фильтры

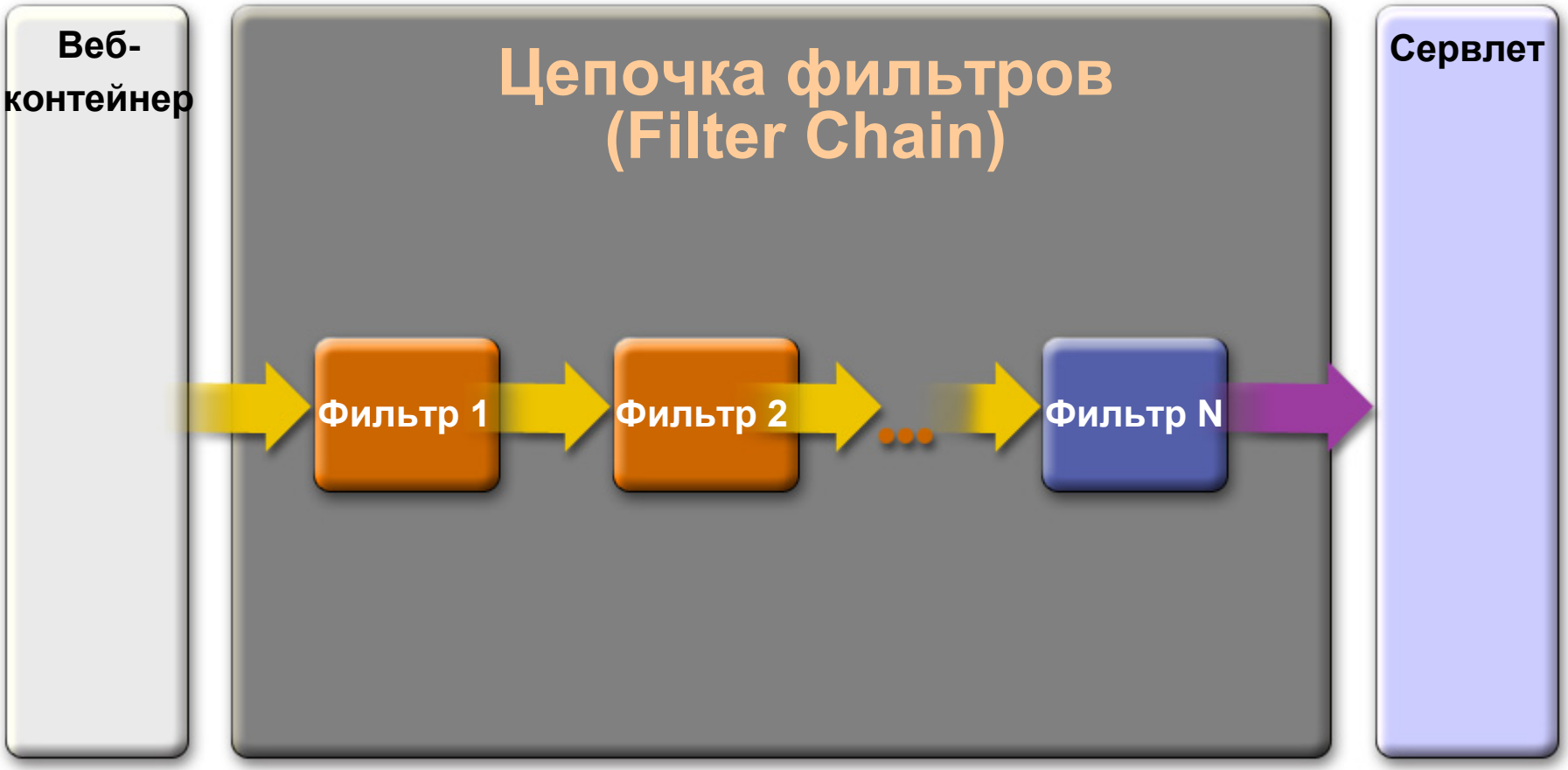
Java Servlet Filter

- Фильтр — компонент для перехвата и изменения запросов к веб-ресурсам и ответов от них
  - При установке веб-приложения из фильтров можно выстроить «конвейер»
- Функции фильтров
  - Как правило, не имеют прямого отношения к бизнес-логике приложения
  - Контроль доступа (аутентификация)
  - Журналирование и аудит
  - Кодирование/декодирование информации (сжатие, шифрование, преобразование формата и прочее)
  - Кэширование

Что может сделать фильтр?

- Проверить заголовки запроса
- Создать оболочку для запроса/ответа, чтобы изменять заголовки или данные
- Вызвать следующий фильтр в цепочке
- Проверить заголовки ответа после вызова следующего фильтра
- Выбросить исключение, чтобы указать на ошибку при обработке запроса

Как работает цепочка фильтров?



Как работает цепочка фильтров?

- Перед одним веб-ресурсом может быть выстроена цепочка из нескольких фильтров
  - Последовательность фильтров в цепочке определяется порядком описания фильтров в web.xml
- Веб-контейнер вызывает первый фильтр в цепочке
  - doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
  - Фильтр выполняет нужную обработку и вызывает следующий фильтр с помощью метода chain.doFilter(..)
- Вызов chain.doFilter() из последнего фильтра в цепочке приводит к вызову на сервлете метода service()

Интерфейс javax.servlet.Filter

- init(FilterConfig)
  - вызывается веб-контейнером один раз при инициализации фильтра
  - объект FilterConfig можно сохранить в поле для последующего использования в методе doFilter()
  - для чтения параметров инициализации предназначен метод FilterConfig.getInitParameter()
- destroy()
  - вызывается веб-контейнером один раз при удалении объекта фильтра
  - позволяет освободить ресурсы (закрыть файлы, соединения с БД)

Интерфейс javax.servlet.Filter

- doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
  - вызывается при каждом обращении к фильтру
  - содержит большую часть логики фильтрации
  - параметр ServletRequest можно преобразовать к типу HttpServletRequest, если используется протокол HTTP
  - можно создать оболочку для запроса и/или ответа,
  - вызвать следующий фильтр методом chain.doFilter(..)
  - или сформировать ответ самостоятельно
    - в т.ч. перенаправить запрос,
  - а также установить дополнительные заголовки в ответе, сформированном следующим фильтром

Конфигурирование фильтров в web.xml

- <filter>
  - <filter-name>: название фильтра
  - <filter-class>: класс фильтра
- </filter>
- <filter-mapping>
  - <filter-name>: название фильтра
  - <url-pattern>: шаблоны URL веб-ресурсов или
  - <servlet-name>: имя сервлета, запросы к которым обрабатывает фильтр
  - <dispatcher>: виды обрабатываемых запросов
    - REQUEST (по умолчанию), FORWARD, INCLUDE, ERROR
- </filter-mapping>

Пример фильтра

- Сохранение в контексте сессии объекта пользователя после успешной аутентификации

```
public class AuthFilter implements Filter {
    private FilterConfig filterConfig = null;

    public AuthFilter() {
    }

    public void destroy() {
    }

    public void init(FilterConfig filterConfig) {
        this.filterConfig = filterConfig;
    }
```

Пример фильтра (продолжение)

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
        throws IOException, ServletException {

    if (((HttpServletRequest) request).getSession()
            .getAttribute("current_user") == null) {
        String username =
            ((HttpServletRequest) request).getRemoteUser();
        if (username != null) {
            User user = new UserDAO().findUser(username);
            ((HttpServletRequest) request).getSession()
                .setAttribute("current_user", user);
        }
    }

    chain.doFilter(request, response);
}
}
```

Пример фильтра (продолжение)

- Конфигурирование фильтра в web.xml

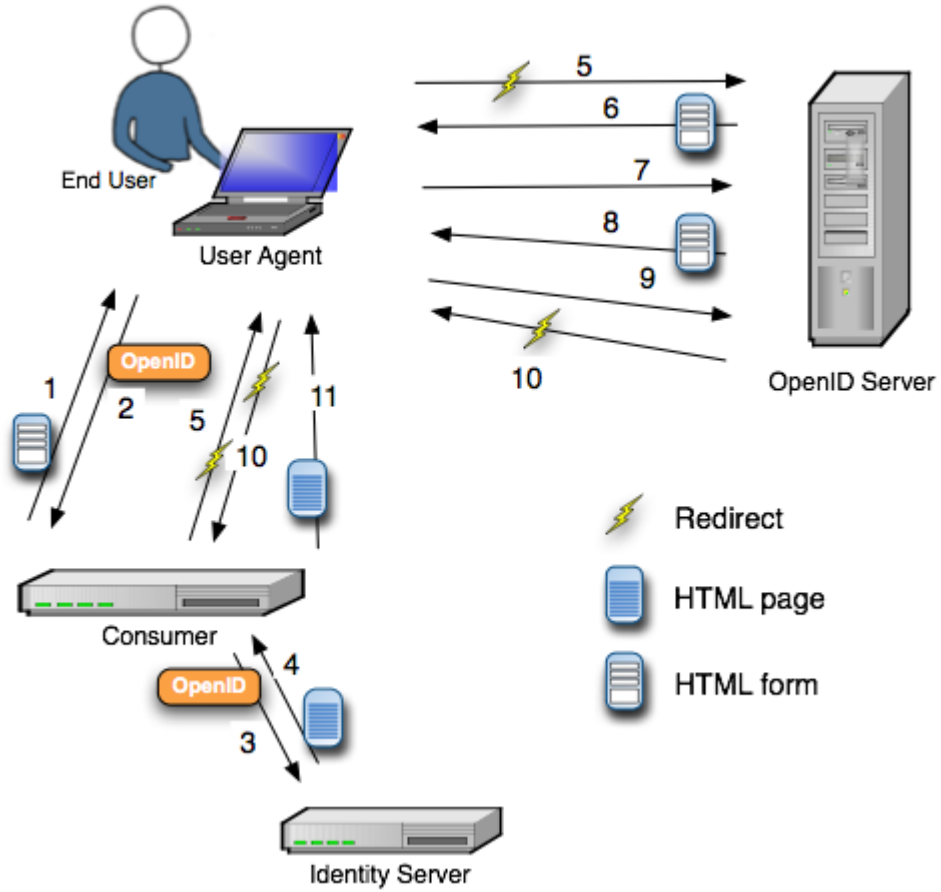
```
<web-app>
  <display-name>MySample</display-name>
  <filter>
    <filter-name>AuthFilter</filter-name>
    <filter-class>ru.ivgratchev.filters.AuthFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>AuthFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

Аутентификация с использованием OpenID

OpenID

- Открытая децентрализованная система единого входа (single sign-on = SSO)
  - Пользователь может использовать единый логин для входа на любой сайт, поддерживающий OpenID
- Участники
  - Конечный пользователь — лицо, которое хочет идентифицировать себя на сайте *Зависимой стороны* с помощью предоставленного *провайдером* Идентификатора (URI или XRI)
    - http://vlsu-sbd-test.myopenid.com/
  - Зависимая сторона — лицо, желающее проверить подлинность Идентификатора
  - OpenID-провайдер — лицо, предоставляющее *Пользователям* сервис регистрации и *Зависимой стороне* сервис проверки подлинности Идентификаторов

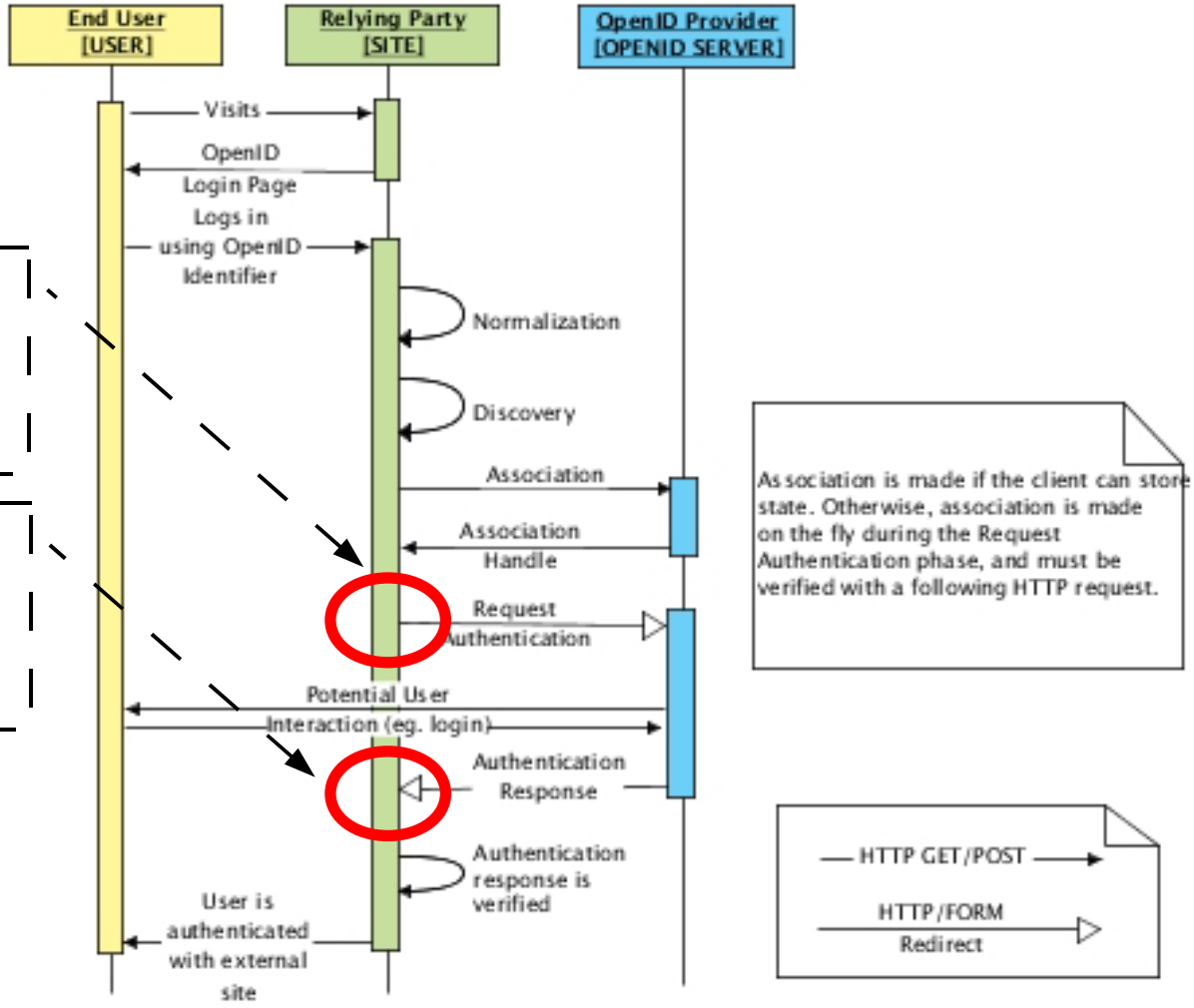
Вход в систему с использованием OpenID



Вход в систему с использованием OpenID

Страница входа в систему

Фильтр аутентификации



Страница входа в систему по OpenID

```
<%@ page import="org.verisign.joid.consumer.OpenIdFilter" %>
<%@ page import="org.verisign.joid.util.UrlUtils" %>
<% String returnTo = UrlUtils.getBaseUrl(request);
   if (request.getParameter("signin") != null) {
            String id = request.getParameter("openid_url");
            if (!id.startsWith("http"))
                  id = "http://" + id;
            String trustRoot = returnTo;
            String s = OpenIdFilter.joid().getAuthUrl(id, returnTo, trustRoot);
            response.sendRedirect(s);
            return;
      }
%>
<form action="index.jsp" method="post" id="openid_form">
      <input type="hidden" name="signin" value="true"/>
      <b>Login with your OpenID URL:</b>
      <input type="text" size="30" name="openid_url" />
      <input type="submit" value="Login"/>
</form>
```

Фильтр аутентификации по OpenID

```
public class OpenIdFilter implements Filter {
        private static Log log = LogFactory.getLog(OpenIdFilter.class);
        private static JoidConsumer joid = new JoidConsumer();
        public static final String OPENID_ATTRIBUTE = "openid.identity";
        public static final String OPENID_IDENTITY = OPENID_ATTRIBUTE;
        boolean saveIdentityUrlAsCookie = false;
    private String cookieDomain;
    private List ignorePaths = new ArrayList();
    private static boolean configuredProperly = false;
    private Integer cookieMaxAge;

    public void init(FilterConfig filterConfig) throws ServletException {
                log.info("init OpenIdFilter");
                String saveInCookie = filterConfig.getInitParameter("saveInCookie");
                if(saveInCookie != null){
                        saveIdentityUrlAsCookie = org.verisign.joid.util.Boolean.parseBoolean(saveInCookie);
                        //saveIdentityUrlAsCookie = Boolean.parseBoolean(saveInCookie);
                        log.debug("saving identities in cookie: " + saveIdentityUrlAsCookie);
                }
                cookieDomain = filterConfig.getInitParameter("cookieDomain");
        String cookieMaxAgeString = filterConfig.getInitParameter("cookieMaxAge");
        if(cookieMaxAgeString != null){
            cookieMaxAge = Integer.valueOf(cookieMaxAgeString);
        }
        String ignorePaths = filterConfig.getInitParameter("ignorePaths");
        if(ignorePaths != null){
            String paths[] = ignorePaths.split(",");
            for (int i = 0; i < paths.length; i++)
            {
                String path = paths[i].trim();
                this.ignorePaths.add(path);
            }
        }
        configuredProperly = true;
        log.debug("end init OpenIdFilter");
        }
```

Фильтр аутентификации по OpenID

```
public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
                     FilterChain filterChain) throws IOException, ServletException {
            // basically just check for openId parameters
    HttpServletRequest request = (HttpServletRequest) servletRequest;
    if (servletRequest.getParameter(OPENID_IDENTITY) != null && !ignored(request)) {
        try {
            AuthenticationResult result = joid.authenticate(servletRequest.getParameterMap());
            String identity = result.getIdentity();
            if(identity != null){
                HttpServletRequest req = (HttpServletRequest) servletRequest;
                req.getSession(true).setAttribute(OpenIdFilter.OPENID_ATTRIBUTE, identity);
                HttpServletResponse resp = (HttpServletResponse) servletResponse;
                Cookie cookie = new Cookie(OPENID_IDENTITY, identity);
                if(cookieDomain != null){
                        cookie.setDomain(cookieDomain);
                }
                if(cookieMaxAge != null){
                    cookie.setMaxAge(cookieMaxAge);
                }
                resp.addCookie(cookie);
                // redirect to get rid of the long url
                resp.sendRedirect(result.getResponse().getReturnTo());
                return;
            }
                    } catch(AuthenticationException e){
            e.printStackTrace();
            log.info("auth failed: " + e.getMessage());
            // should this be handled differently?
        } catch(Exception e) {
                            e.printStackTrace();
        }
    }
    filterChain.doFilter(servletRequest, servletResponse);
}
```

Фильтр аутентификации по OpenID — конфигурирование в web.xml

```
<filter>
        <filter-name>OpenIdFilter</filter-name>
        <filter-class>org.verisign.joid.consumer.OpenIdFilter</filter-class>
         <init-param>
                <param-name>saveInCookie</param-name>
                <param-value>true</param-value>
         </init-param>
        <init-param>
                <param-name>ignorePaths</param-name>
                <param-value>/login,/server,/echo</param-value>
         </init-param>
</filter>
<filter-mapping>
        <filter-name>OpenIdFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

Обработчики событий

События жизненного цикла сервлета

- Уведомления об изменении состояния
  - Веб-приложения
    - Запуск/останов
    - Изменение атрибутов
  - HTTP-сессии (HttpSession)
    - Создание и закрытие
    - Изменение атрибутов

Реализация обработчика событий

1. Определить, какие события и для какого контекста нужно обрабатывать
2. Реализовать соответствующий интерфейс
3. Переопределить методы, соответствующие интересующим событиям
4. Сконфигурировать обработчик в web.xml
5. Указать необходимые параметры инициализации

Регистрация обработчиков событий

- Веб-контейнер
  - Создает один объект указанного класса обработчика
  - Регистрирует его до обработки первого запроса к веб-приложению
  - Регистрация обработчиков выполняется согласно
    - Реализуемым интерфейсам
    - Их порядку в web.xml
- Во время выполнения обработчики вызываются в порядке регистрации

Интерфейсы обработки событий

- ServletContextListener
  - contextInitialized/Destroyed(ServletContextEvent)
- ServletContextAttributeListener
  - attributeAdded/Removed/Replaced(ServletContextAttributeEvent)
- HttpSessionListener
  - sessionCreated/Destroyed(HttpSessionEvent)
- HttpSessionAttributeListener
  - attributedAdded/Removed/Replaced(HttpSessionBindingEvent)
- HttpSessionBindingListener
  - valueBound/Unbound(HttpSessionBindingEvent)
- HttpSessionActivationListener
  - sessionWillPassivate(HttpSessionEvent)
  - sessionDidActivate(HttpSessionEvent)

Пример обработчика событий

```
public final class ContextListener
     implements ServletContextListener {
     private ServletContext context = null;

     public void contextInitialized(ServletContextEvent event) {
          context = event.getServletContext();

          try {
             BookDB bookDB = new BookDB();
             context.setAttribute("bookDB", bookDB);
          } catch (Exception ex) {
             context.log("Couldn't create bookstore
                          database bean: " + ex.getMessage());
          }

          Counter counter = new Counter();
          context.setAttribute("hitCounter", counter);
     }

     public void contextDestroyed(ServletContextEvent event) {
          context = event.getServletContext();
          BookDB bookDB = (BookDB)context.getAttribute("bookDB");
          bookDB.remove();
          context.removeAttribute("bookDB");
          context.removeAttribute("hitCounter");
     }

 }
```

Конфигурирование обработчика в web.xml

```
<web-app>
  <display-name>Bookstore1</display-name>
  <description>no description</description>

  <filter>..</filter>
  <filter-mapping>..</filter-mapping>
  <listener>
    <listener-class>listeners.ContextListener</listener-class>
  </listener>
  <servlet>..</servlet>
  <servlet-mapping>..</servlet-mapping>
  <session-config>..</session-config>
  <error-page>..</error-page>
  ...
</web-app>
```