Фильтры веб-сервера

JSP & Servlets
12 уровень, 5 лекция

ОТКРЫТА

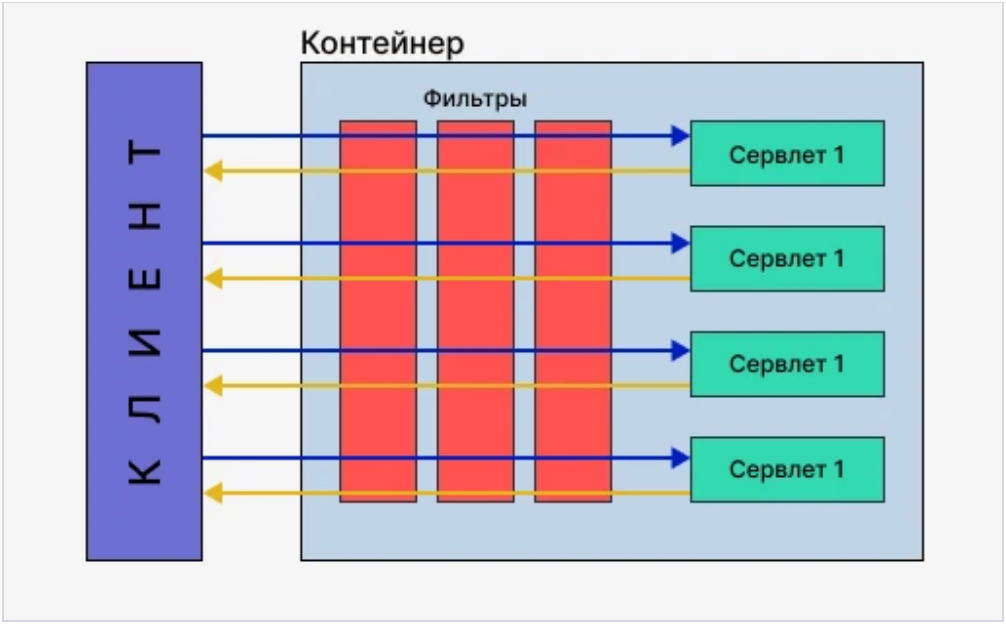
Знакомство с фильтрами

Но и это еще не все. Правда, ты же не думаешь, что сервлеты устроены так просто?

Кроме сервлетов, которые мы уже разобрали, есть еще так называемые “служебные сервлеты” — **фильтры**. Они очень похожи на сервлеты, но их основная задача — помогать сервлетам обрабатывать запросы.

Фильтр — это как секретарь, а сервлет – директор. Прежде чем документ попадет на стол директору, он пройдет через руки секретаря. И после того, как директор его подпишет, он снова попадет секретарю, уже как исходящая корреспонденция, например. Такой секретарь может отбраковывать часть запросов к директору (например, спам). Или давать стандартные ответы на известные ему вопросы (“директора нет на месте”). И так далее. Более того, таких секретарей может быть несколько: один может фильтровать спам сразу для всех директоров, другой перекидывать запросы между разными директорами и тому подобное.

Так же работают и фильтры:



Классы Filter, FilterChain, FilterConfig

Фильтры очень похожи на сервлеты, но с парой небольших отличий. Чтобы написать свой фильтр, нужно наследоваться от интерфейса `javax.servlet.Filter`.

У фильтра так же есть методы `init()` и `destroy()`. Вместо метода `service()` у фильтра есть метод `doFilter()`. И даже есть свой класс FilterConfig. Фильтр также добавляется в сервлет в файле web.xml или же с помощью аннотации @WebFilter.

Список методов:

	Методы	**Описание**
1	`init(FilterConfig config)`	инициализация фильтра
2	`destroy()`	выгрузка фильтра
3	`doFilter(ServletRequest , ServletResponse, FilterChain)`	обработка (фильтрация) запроса

В чем же отличие сервлета и фильтра?

Фильтров может быть несколько, и они последовательно обрабатывают запрос (и ответ). Они объединены в так называемую цепочку — и для них даже есть специальный класс `FilterChain`.

После обработка запроса в методе `doFilter()` нужно вызвать метод `doFilter()` следующего фильтра в цепочке. Пример:

```
1  public class MyFilter implements Filter {
2
3    public void init(FilterConfig arg0) throws ServletException {
4    }
5
6    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws Exception {
7
8        PrintWriter out = resp.getWriter();
9        out.print("Дописываем что-то перед телом ответа");
10
11       chain.doFilter(req, resp); //вызываем следующий фильтр в цепочке
12
13       out.print("Дописываем что-то после тела ответа");
14     }
15
16   public void destroy() {
17   }
18 }
```

Вообще-то так дописывать тело ответа нельзя. Формально фильтры и сервлеты независимы друг от друга и могут изменяться независимо. Их могут писать разные разработчики в разное время. Функция фильтров именно служебная, например:

- Логирование всех входящих запросов (и ответов)
- Сжатие данных
- Шифрование (и расшифровка) данных
- Валидация данных запроса
- Добавление/удаление нужных заголовков
- Перенаправление запросов
- Контроль доступа (проверка, залогинен ли пользователь)

Класс RequestDispatcher

Иногда в процессе работы фильтра внутри метода `doFilter()` может возникнуть необходимость вызвать другой сервлет. Для этого у контейнера есть специальный объект `RequestDispatcher`.

Получить его можно двумя способами:

- У объекта `HttpServletRequest`
- У объекта `ServletContext`

Этот объект можно использовать для того, чтобы перенаправить существующий запрос на другой сервлет. Например, выяснилось, что пользователь не авторизован и мы хотим показать ему страницу с авторизацией. Ну или произошла ошибка на сервере и мы хотим отобразить пользователю error-страницу :)

```
1  public class HelloServlet extends HttpServlet {
2      protected void doGet(HttpServletRequest request, HttpServletResponse response)  throws Exception {
3              String path = "/error.html";
4              ServletContext servletContext = this.getServletContext();
5              RequestDispatcher requestDispatcher = servletContext.getRequestDispatcher(path);
6              requestDispatcher.forward(request, response);
7      }
8  }
```

Также ты можешь вызвать `RequestDispatcher` из фильтра.

```java
public class MyFilter implements Filter {

  public void init(FilterConfig arg0) throws ServletException {
  }

  public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws Exception {
           String path = "/error.html";
           ServletContext servletContext = req.getServletContext();
           RequestDispatcher requestDispatcher = servletContext.getRequestDispatcher(path);
           requestDispatcher.forward(request, response);
    }

  public void destroy() {
  }
}
```

Обрати внимание, что запрос будет обработан в методе `forward()`, и вызывать `doFilter()` после использования `RequestDispatcher` не нужно.

Сравнение редиректа и форварда

И еще один важный момент. Если ты хочешь в своем сервлете перенаправить пользователя на другой URI, то сделать это можно двумя способами:

- `redirect`
- `forward`

Мы их уже разбирали, но для удобства проговорю это еще раз.

Когда ты выполняешь **redirect** через вызов `response.sendRedirect("ссылка")`, то сервер отсылает браузеру (клиенту) ответ `302` и указанную тобой ссылку. А браузер, проанализировав ответ сервера, загружает переданную тобой ссылку. То есть ссылка в браузере меняется на новую.

Если ты выполняешь **forward** через вызов `requestDispatcher.forward()`, то новый запрос выполняется внутри контейнера, и его ответ твой сервлет отсылает браузеру (клиенту) как ответ твоего сервлета. При этом браузер получает ответ от нового сервлета, но ссылка в браузере не меняется.

‹ Предыдущая лекция

Следующая лекция ›