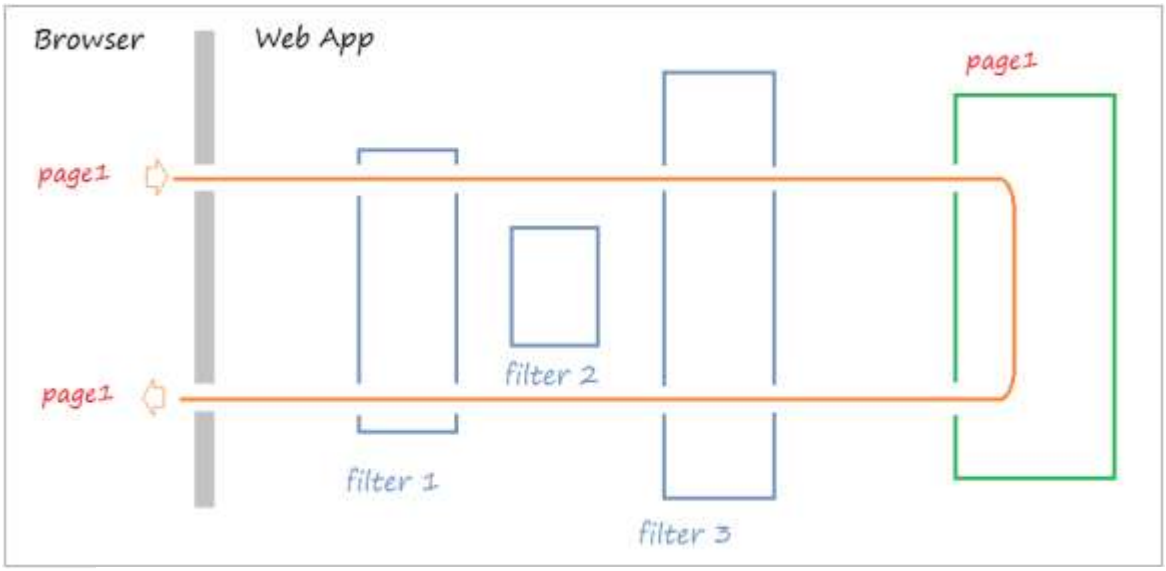
Filters

Назначение фильтров

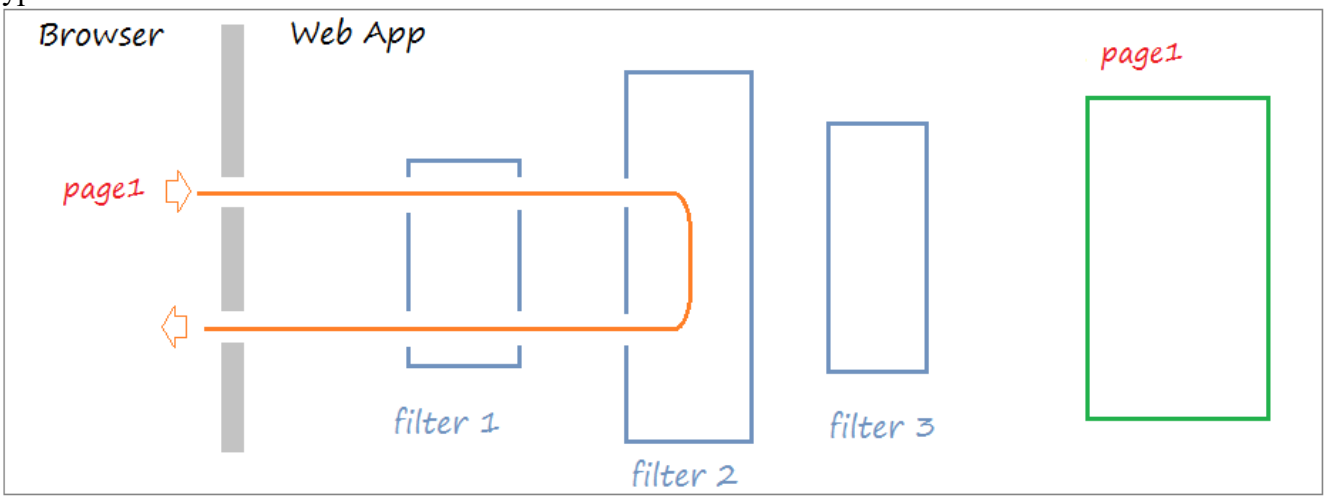
Мы можем использовать фильтры для следующих задач:

- Работа с ответами сервера перед тем, как они будут переданы клиенту.
- Перехватывание запросов от клиента перед тем, как они будут отправлены на сервер.

Обычно, когда пользователь запрашивает веб-страницу, запрос будет отправлен на server, он должен проходить через фильтры (Filter) до того как дойти до запрошенной страницы, как в изображении ниже.



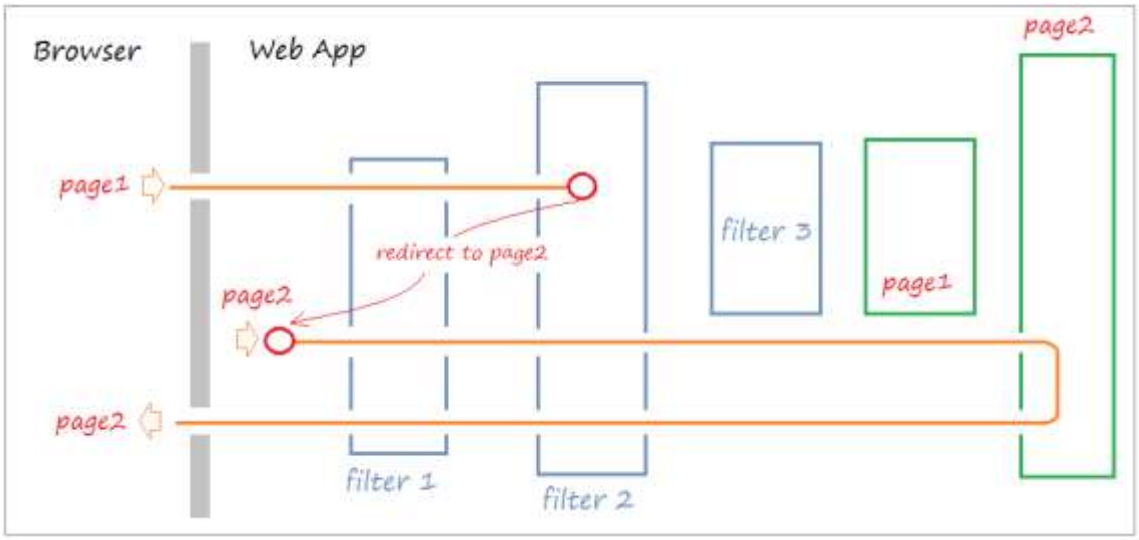
Однако существуют ситуации запроса пользователя, которые не проходят все уровни **Filter**.



Ситуация когда пользователь отправляет запрос одной страницы (page1), этот запрос проходит через Filter, у определенного filter запрос будет перенаправлен на другую страницу (page2).

Пример ситуации:

1) Пользователь посылает запрос, чтобы посмотреть страницу с личной информацией.
2) Запрос будет отправлен на Server.
3) Она проходит Filter, который записывает информацию log.
4) Идет к Filter и проверяет вошел ли пользователь в систему, filter проверил, и увидел, что пользователь не вошел в систему, то перенаправляет запрос на страницу входа в систему пользователя.



На самом деле **Filter** может быть использован для кодирования вебсайта (encoding). Например, установить кодировку **UTF-8** для страницы. Открыть и закрыть соединение к Database и подготовить транзакцию **JDBC** (JDBC Transaction).

Сервлетные фильтры

Сервлетный фильтр, в соответствии со спецификацией, это Java-код, пригодный для повторного использования и позволяющий преобразовать содержание HTTP-запросов, HTTP-ответов и информацию, содержащуюся в заголовках HTML. *Сервлетный фильтр занимается предварительной обработкой запроса, прежде чем тот попадает в сервлет, и/или последующей обработкой ответа, исходящего из сервлета.*

Сервлетные фильтры могут:

- перехватывать инициацию сервлета прежде, чем сервлет будет инициирован;
- определить содержание запроса прежде, чем сервлет будет инициирован;
- модифицировать заголовки и данные запроса, в которые упаковывается поступающий запрос;
- модифицировать заголовки и данные ответа, в которые упаковывается получаемый ответ;
- перехватывать инициацию сервлета после обращения к сервлету.

Сервлетный фильтр может быть конфигурирован так, что он будет работать с одним сервлетом или группой сервлетов. Основой для формирования фильтров служит интерфейс *javax.servlet.Filter*, который реализует три метода:

- void *init* (FilterConfig config) throws ServletException;
- void *destroy ()*;
- void *doFilter* (ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException;

Метод *ini* вызывается прежде, чем фильтр начинает работать,и настраивает конфигурационный объект фильтра. Метод *doFilter* выполняет непосредственно работу фильтра. Таким образом, сервер вызывает *init* один раз, чтобы запустить фильтр в работу, а затем вызывает *doFilter* столько раз, сколько запросов будет сделано непосредственно к данному фильтру. После того, как фильтр заканчивает свою работу, вызывается метод *destroy*.

Листинг простого фильтра

```
package common;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FilterConnect implements Filter
{
    private FilterConfig config = null;
    private boolean active = false;
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    public void init (FilterConfig config) throws ServletException
    {
        this.config = config;
        String act = config.getInitParameter("active");
        if (act != null)
           active = (act.toUpperCase().equals("TRUE"));
    }
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    public void doFilter (ServletRequest request, ServletResponse response,
                    FilterChain chain) throws IOException,
                                        ServletException
    {
        if (active)
        {
            // Здесь можно вставить код для обработки
        }
        chain.doFilter(request, response);
    }
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    public void destroy()
    {
        config = null;
    }
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
}
```

*В примере представленного шаблона фильтра инициализируется значение параметра **active**. При вызове фильтра (**doFilter**) проверяется значение параметра **active**, и если **active = true**, могут быть выполнены действия, определенные разработчиком.*

Интерфейс *FilterConfig* содержит метод для получения имени фильтра, его параметров инициации и контекста активного в данный момент сервлета. С помощью своего метода *doFilter* каждый фильтр получает текущий запрос *request* и ответ *response*, а также *FilterChain*, содержащий список фильтров, предназначенных для обработки.

В *doFilter* фильтр может делать с запросом и ответом всё, что ему захочется - собирать данные или упаковывать объекты для придания им нового поведения. Затем фильтр вызывает *chain.doFilter*, чтобы передать управление следующему фильтру. После возвращения этого вызова фильтр может по окончании работы своего метода *doFilter* выполнить дополнительную работу над полученным ответом. К примеру, сохранить регистрационную информацию об этом ответе.

После того, как класс-фильтр откомпилирован, его необходимо установить в контейнер и "приписать"(map) к одному или нескольким сервлетам. Объявление и подключение фильтра отмечается в дескрипторе поставки *web.xml* внутри элементов <*filter*> и <*filter-mapping*>. Для подключение фильтра к сервлету необходимо использовать вложенные элементы <*filter-name*> и <*servlet-name*>. Например:

Объявление фильтра

```
<filter>
    <filter-name>FilterName</filter-name>
    <filter-class>FilterConnect</filter-class>
    <init-param>
        <param-name>active</param-name>
        <param-value>true</param-true>
    </init-param>
</filter>
```

Подключение фильтра к сервлету

```
<filter-mapping>
    <filter-name>FilterName</filter-name>
    >servlet-name>ServletName</servlet-name>
</filter-mapping>
```

В представленном коде дескриптора поставки *web.xml* объявлен класс-фильтр *FilterConnect* с именем *FilterName*. Фильтр имеет параметр инициализации *active*, которому присвивается значение *true*. Фильтр *FilterName* в разделе <filter-mapping> подключен к сервлету *ServletName*.

Порядок, в котором контейнер строит цепочку фильтров для запроса определяется следующими правилами:

- цепочка, определяемая *url-pattern*, выстраивается в том порядке, в котором встречаются соответствующие описания фильтров в *web.xml*;
- последовательность сервлетов, определенных с помощью *servlet-name*, также выполняется в той последовательности, в какой эти элементы встречаются в дескрипторе поставки *web.xml*.

Для связи фильтра со страницами HTML или группой сервлетов необходимо использовать тег <*url-pattern*>. Например, после следующего кода

Подключение фильтра к HTML-страницам

```
<filter-mapping>
    <filter-name>FilterName</filter-name>
    <url-pattern>*.html</url-pattern>
```

```
</filter-mapping>
```

фильтр будет применен ко всем вызовам страниц HTML.

Использование дополнительных ресурсов

В отдельных случаях недостаточно вставить в сервлет фильтр или даже цепочку фильтров, а необходимо обратиться к другому сервлету, странице JSP, документу HTML, XML или другому ресурсу. Если требуемый ресурс находится в том же контексте, что и сервлет, который его вызывает, то для получения ресурса необходимо использовать метод

```
public RequestDispatcher getRequestDispatcher(String path);
```

представленному в интерфейсе *ServletRequest*. Здесь *path* - это путь к ресурсу относительно контекста. Например, необходимо обратиться к сервлету *Connect*:

```
RequestDispatcher rd = request.getRequestDispatcher("Connect");
```

Если ресурс находится в другом контексте, то необходимо предварительно получить контекст методом

```
public ServletContext getContext(String uripath);
```

интерфейса *ServletContext*, а потом использовать метод

```
public RequestDispatcher getRequestDispatcher (String uripath);
```

интерфейса *ServletContext*. Здесь путь *uripath* должен быть абсолютным, т.е. начинаться с наклонной черты /. Например:

```
RequestDispatcher rd = config.getServletContext().getContext("/prod").getRequestDispatcher("/prod/Customer");
```

Если требуемый ресурс - сервлет, помещенный в контекст под своим именем, то для его получения можно обратиться к методу

```
RequestDispatcher getNamedDispatcher (String name);
```

интерфейса *ServletContext*. Все три метода возвращают *null*, если ресурс недоступен или сервер не реализует интерфейс *RequestDispatcher*.
Как видно из описания методов, к другим ресурсам можно обратиться только через объект типа *RequestDispatcher*, который предлагает два метода обращения к ресурсу. Первый метод

```
public void forward (ServletRequest request, ServletResponse response);
```

просто передает управление другому ресурсу, предоставив ему свои аргументы *request* и *response*. Вызывающий сервлет выполняет предварительную обработку объектов *request* и *response* и передает их вызванному сервлету или другому ресурсу, который окончательно формирует ответ *response* и отправляет его клиенту или, опять-таки, вызывает другой ресурс. Например:

```
if (rd != null)
   rd.forward (request, response);
else
   response.sendError (HttpServletResponse.SC_NO_CONTENT);
```

Вызывающий сервлет не должен выполнять какую-либо отправку клиенту до обращения к методу *forward,* иначе будет выброшено исключение класса *IllegalStateException*. Если же вызывающий сервлет уже что-то отправлял клиенту, то следует обратиться ко второму методу

public void include (ServletRequest request, ServletResponse response);

Этот метод вызывает ресурс, который на основании объекта request может изменить тело объекта response. Но вызванный ресурс не может изменить заголовки и код ответа объекта response. Это естественное ограничение, поскольку вызывающий сервлет мог уже отправить заголовки клиенту. Попытка вызванного ресурса изменить заголовок будет просто проигнорирована. Можно сказать, что метод include выполняет такую же работу, как вставки на стороне сервера SSI(Server Side Include).