Filters

Intercepting all requests

EHSDI

eBuzima

Introduction

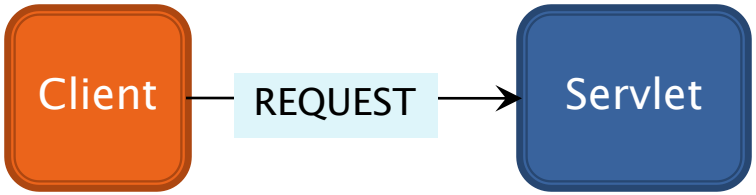
- Suppose you've developed a large web app with lots of servlets
- You are then asked to add request tracking to the site – i.e. record some data about every single request to all of the servlets

- How can we do this without modifying every servlet's code?

Filters

- One filter can intercept all requests to all servlets
- More than one filter can be chained
- A ***request filter*** can process the request before it gets to the servlet
- A ***response filter*** can process the response from the servlet before it's sent to the client

Request filters



- Can be used to...

- Check a client's privileges to ensure they are allowed to access the content they are requesting
- Log requests – collect statistics about who is accessing the web site
- Can modify the request headers

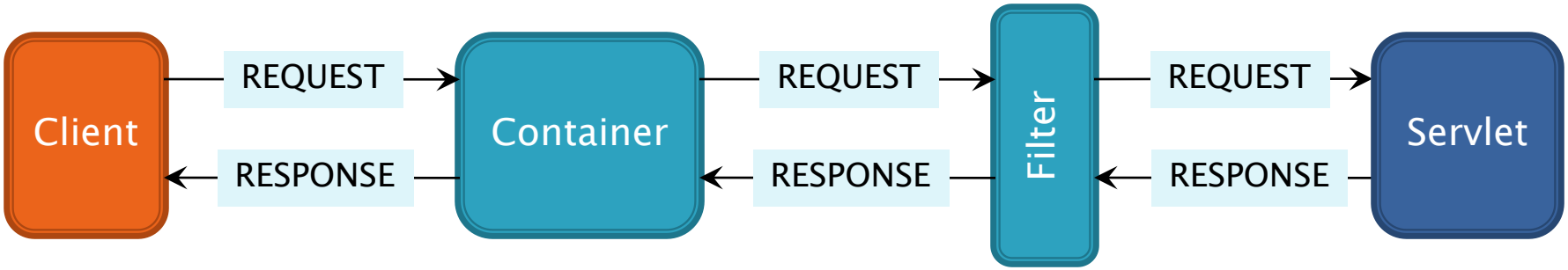
Response filters

- Can be used to...

- Compress the output stream
  - OpenMRS can do this using gzip compression
- Modify the output stream, e.g.
  - Strip out bad words
  - Encode email addresses
  - Add extra functionality or styling

Filters

- Every filter intercepts the request and the response, regardless of what it does



- So its up to the filter to choose what to process

Example: an empty filter

- All filters implement `javax.servlet.Filter` which defines `init`, `doFilter` and `destroy`

```
public class TestFilter implements Filter {
  public void init(FilterConfig fConfig)
    throws ServletException {
  }

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

    chain.doFilter(request, response);
  }

  public void destroy() {
  }
}
```

Example: the DD

- Filters are declared in the DD and mapped to URLs, just like servlets

```
<filter>
  <filter-name>TestFilter</filter-name>
  <filter-class>test.TestFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>TestFilter</filter-name>
  <url-pattern>*.htm</url-pattern>
</filter-mapping>
```

This filter will process all requests for files with the **htm** extension

Filter mappings

- Can map to a URL pattern using `<url-pattern>`

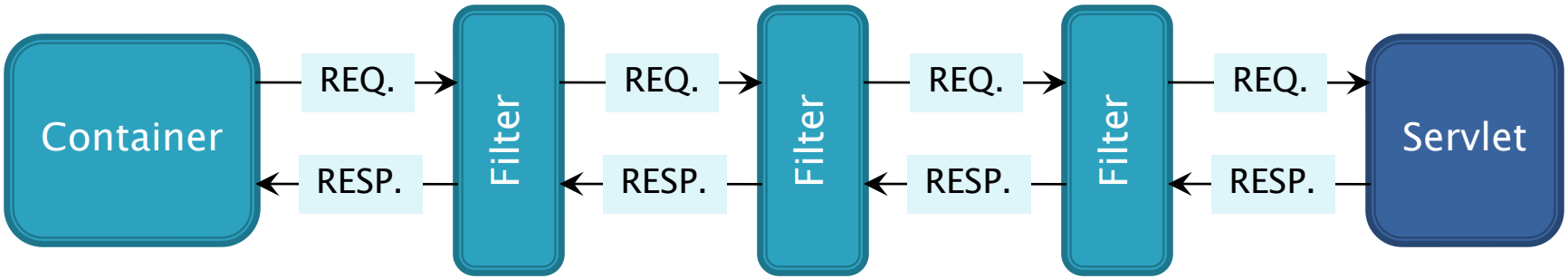
Pattern	Matches
`*.htm`	All files with 'htm' extension
`/test/*`	All items in the 'test' folder
`/*`	All requests

- ...or instead can map to a specific servlet using `<servlet-name>`, e.g.

```
<filter-mapping>
  <filter-name>TestFilter</filter-name>
  <servlet-name>TestServlet</servlet-name>
</filter-mapping>
```

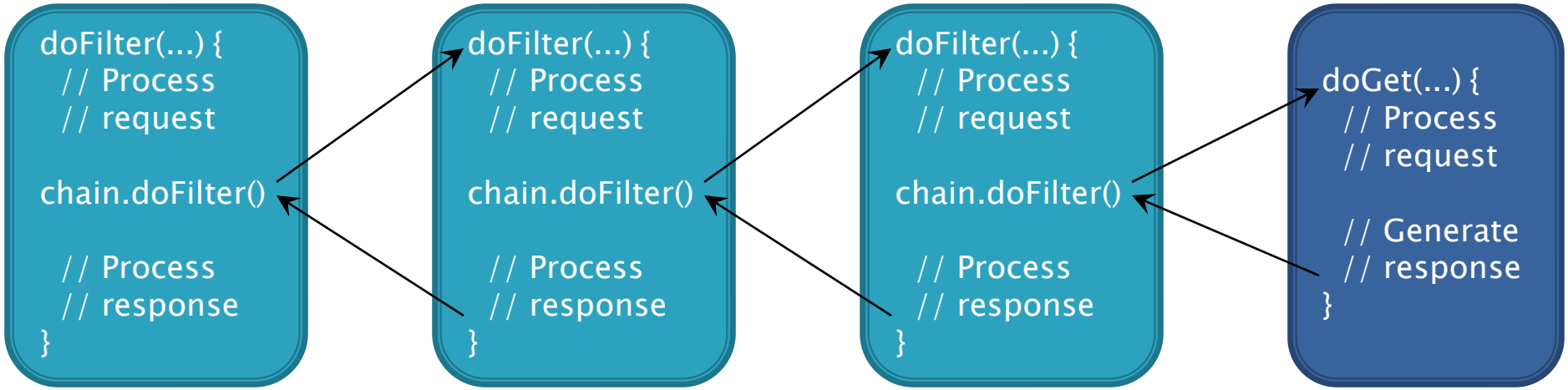
Chaining

- More than one filter's URL mapping may match the request URL
- This is called filter chaining
- The order in which the **filter mappings** appear in the DD is the order in which the filters are chained



Chaining

- The servlet itself is part of the chain
- Filters act independently – they don't know what other filters are in the chain



Chaining

- Chaining creates a kind of filter stack..
  1. Filter #1 processes the request
  2. Filter #2 processes the request
  3. Filter #3 processes the request
  4. Servlet processes the request
  5. Servlet generates the response
  6. Filter #3 processes the response
  7. Filter #2 processes the response
  8. Filter #1 processes the response

Example: request filter

- Relatively simple: process the request and then pass it to the chain...

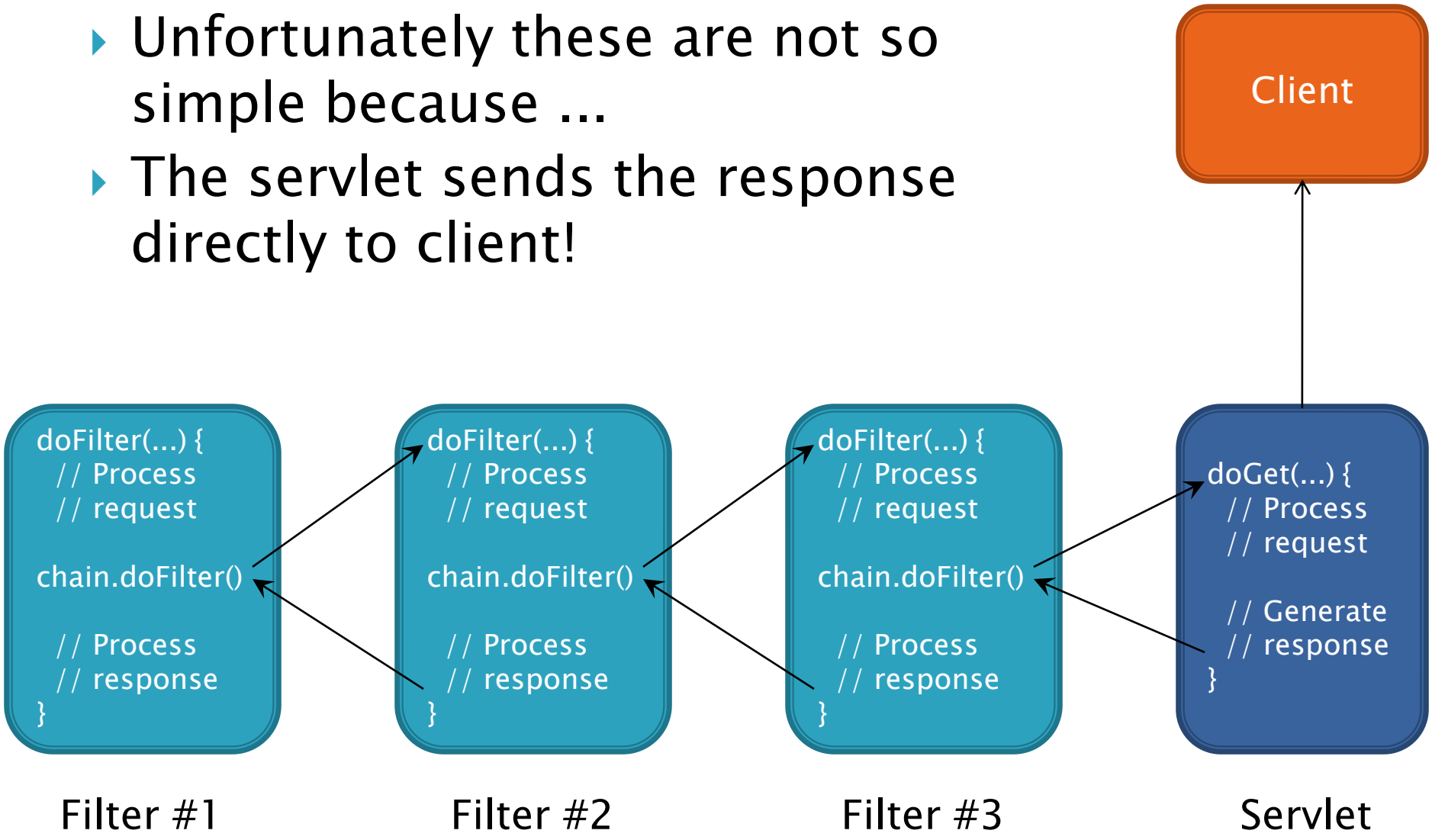
```
public class TestFilter implements Filter {
  public void doFilter(ServletRequest request,
                       ServletResponse response,
                       FilterChain chain)
    throws IOException, ServletException {

    SuperWebLogger.recordRequest(request);

    chain.doFilter(request, response);
  }

  ...
}
```

Does something with the request, the response hasn't yet been generated

This filter doesn't know or care if they are more filters in the chain

Response filters... not so simple

- Unfortunately these are not so simple because ...
- The servlet sends the response directly to client!

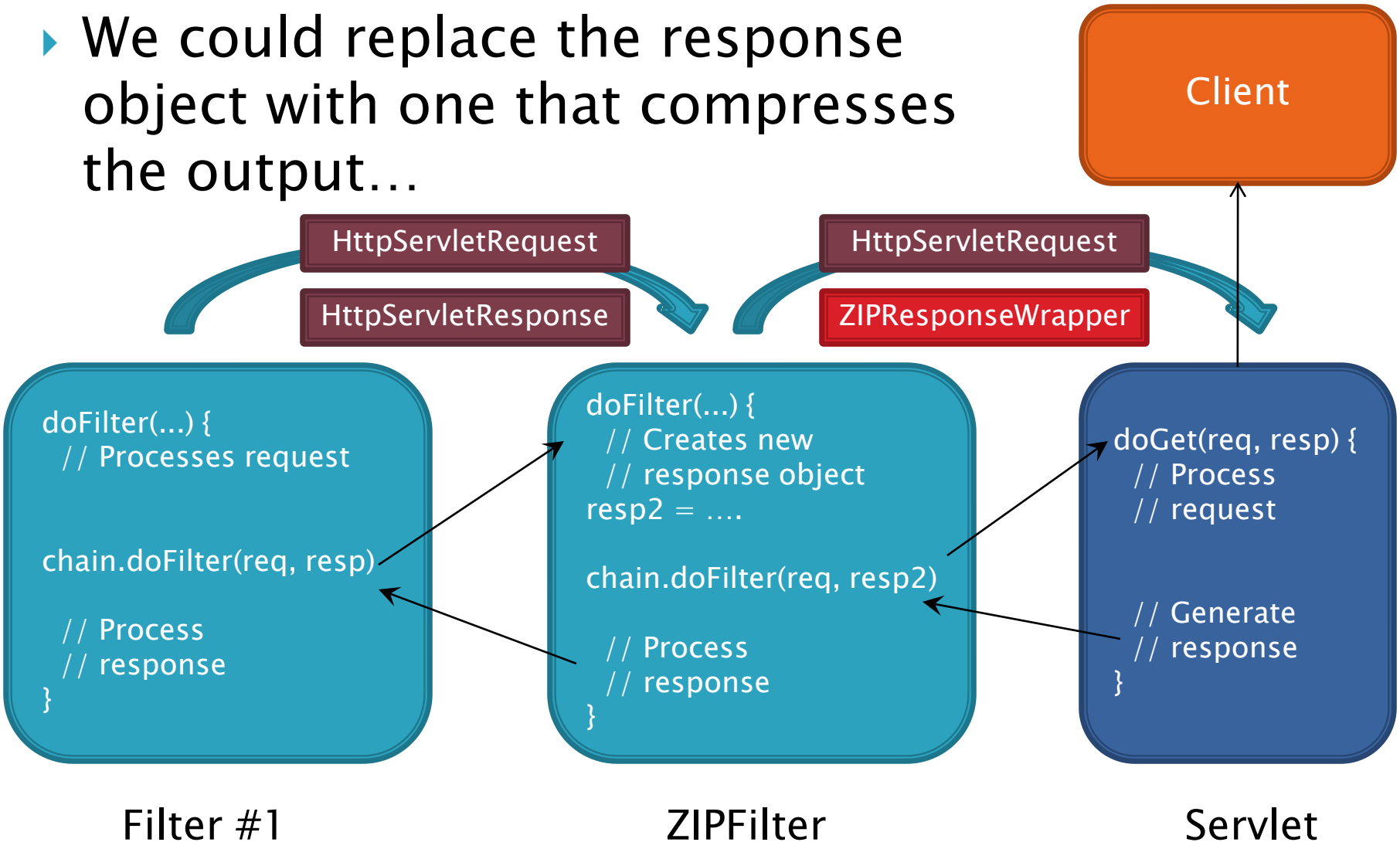


Response filters

- So the filters can access the response, but it **has already been sent to the client**, so it's too late to modify it
- What a filter can do though, is **replace the response object** that gets passed to the servlet
  - For example: replace the usual output stream in the response object with a output stream that compresses the output

Response filter example

- We could replace the response object with one that compresses the output…



References

- Books
  - Head First Servlets and JSP (O'Reilly)
- Websites
  - http://java.sun.com/products/servlet/Filters.html
  - http://java.sun.com/javaee/reference/tutorials/