Writing Filters

Kevin Jones

@kevinrjones www.rocksolidknowledge.com

Overview

Understand why filters are important

Learn how to write a filter

Understand why wrapper classes are necessary

Learn how to write and use wrappers

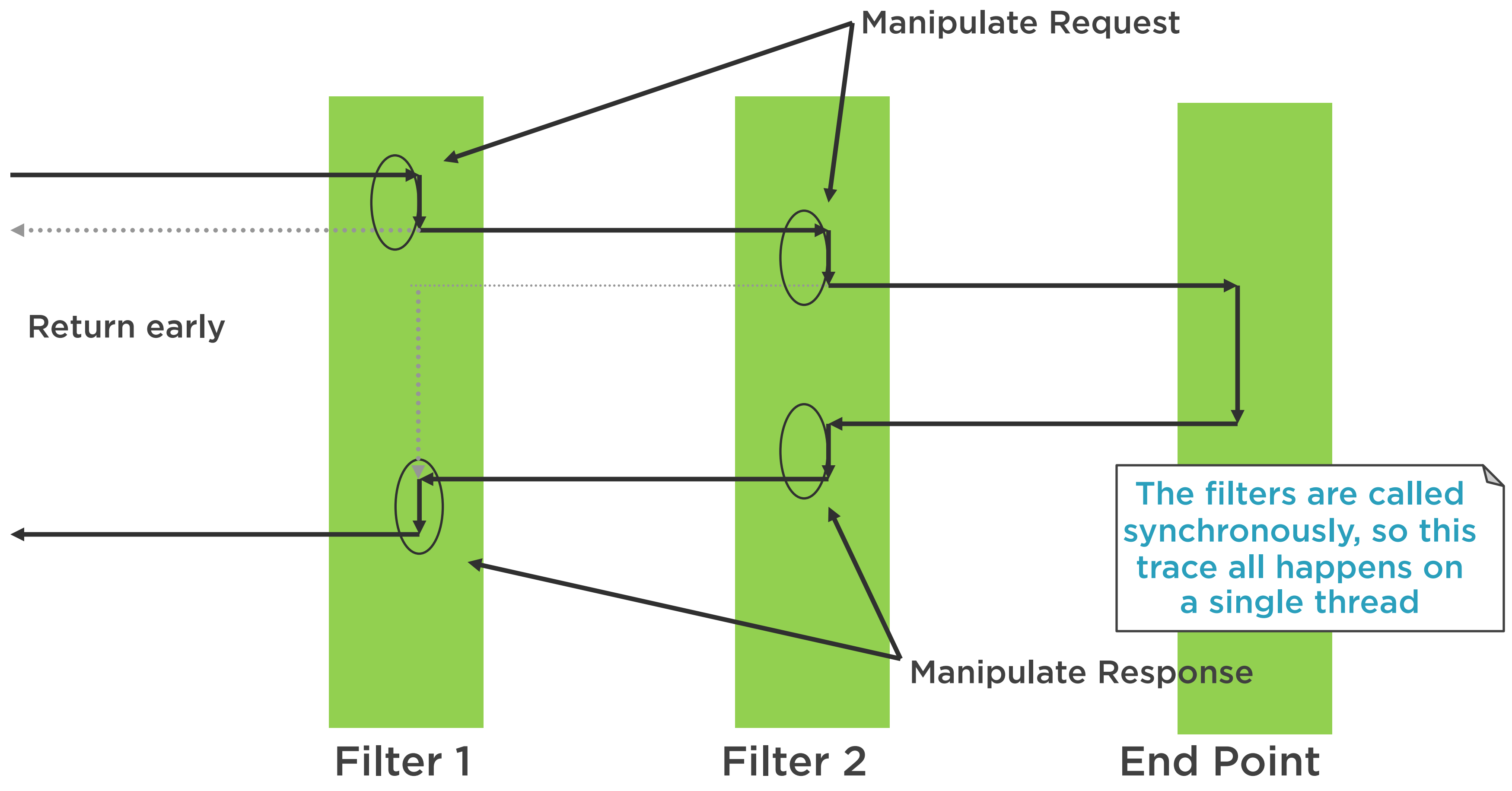
Learn how to configure filters

What Is a Filter?

Code that intercepts a request and does extra processing

- They are executed before/after the request is executed
- The request could be to a servlet/JSP/HTML etc.
- Request/Response could be modified in the filter
- Filters may be executed as part of a chain
- Filters can intercept original request
- Filters can intercept forwards and includes
- Can use filters to provide: Session management, logging, security, and XML transforms

Executing Filters



The filters are called synchronously, so this trace all happens on a single thread

Writing Filters

Filter writers implement `javax.servlet.Filter`

- **`doFilter()`** may "wrap" **`Request`** and/or **`Response`**
- Call **`chain.doFilter(...)`** to pass on the request
- doFilter can dispatch the request to a different resource
- doFilter may return to the caller without chaining the request

```
// called once at start
public void init(FilterConfig config);
// called once at end
public void destroy();
// where the work is done
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain);
```

Filter Configuration

Filter Configures in Deployment Descriptor

- May be associated with a URL
- May be associated with a named resource

```
<filter>
    <filter-name>Logging Filter</filter-name>
    <filter-class>com.mantiso.filter.LoggingFilter</filter-class>
    <init-param>
        <param-name>jbdc-url</param-name>
        <param-value>jdbc:mssql:localhost:1024</param-value>
    </init-param>
</filter>

<filter-mapping>
  <filter-name>Logging Filter</filter-name>
  <url-pattern>/*</url-pattern>
  <!--    <servlet-name>SessionTest</servlet-name> -->
</filter-mapping>
```

Initialization Method

Called once when the filter is loaded

- Passed a reference to a **`FilterConfig`**
- Use this to get a reference to the **`ServletContext`** if necessary

```
public void init(FilterConfig filterConfig)
{
    fc = filterConfig;
    if(fc != null)
    {
       ctx = fc.getServletContext();
       // other initialisation
       value = getInitParameter("param-name");
    }
}
```

parameters specified in web.xml

Doing the work

`doFilter` is passed the request, response and the filter chain

```
public void doFilter(ServletRequest req,
                     ServletResponse resp,
                     FilterChain chain)
       throws java.io.IOException, ServletException
{
  HttpServletRequest request = (HttpServletRequest)req;
  HttpServletResponse response = (HttpServletResponse)resp;

  // pre process request
  // optionally 'wrap' request and response

  chain.doFilter(req, response); // also optional

  // post process response
}
```

ServletRequest and ServletResponse

Filter could simply return or use a RequestDispatcher

Wrapping Request

Can extend `HttpServletRequestWrapper`

- Adaptor class that takes original request object as constructor parameter
- Default behaviour is to simply call the original request
- Override necessary methods
- Can: invent headers, change attributes, log calls etc…

```
class LoggingRequestWrapper extends
                      HttpServletRequestWrapper
{
    LoggingRequestWrapper(HttpServletRequest request)
    {
        super(request);
    }
}
```

Override appropriate methods

Logging Filter Example

```
public class LoggingFilter implements javax.servlet.Filter {
    Writer output;
    static Logger logger = Logger.getLogger(LoggingFilter.class);

    public void init(FilterConfig filterConfig) throws ServletException {
        // Set up a simple configuration that logs on the console.
        BasicConfigurator.configure();
    }

    public void doFilter(ServletRequest servletRequest,
        ServletResponse servletResponse,
        FilterChain chain) throws IOException, ServletException {

        LogRequestWrapper requestWrapper =
            new LogRequestWrapper((HttpServletRequest)servletRequest,  logger);
        chain.doFilter(requestWrapper, servletResponse);
    }

    public void destroy(){
    }
}
```

Using Log4j

Create Wrapper and chain request

Logging Filter Example

```
public class LogRequestWrapper extends HttpServletRequestWrapper
{
    Logger logger;
    public LogRequestWrapper(HttpServletRequest httpServletRequest,
        Logger logger) {
        super(httpServletRequest);
        this.logger = logger;
    }

    public String getHeader(String s) {
        String header =  super.getHeader(s);
        logger.info("[getHeader] Asked for :" + s + " ;got: " + header);
        return header;
    }

    /*
        Override any other methods
    */
}
```

Called whenever a 'chained' resource calls getHeader

Wrapping Response

Filter may want to change response

- Can create **`HttpServletResponseWrapper`**
- More interesting than **`HttpServletRequestWrapper`**
- Have to cope with (at least) content length, content type, getWriter and getOutputStream

Example - Compression Filter

Uses GZipOutputStream to compress content

- Servlet calls getOutputStream/getWriter
- Wrapper returns stream that compresses content

Compression Filter

Filter to use gzip compression

- Accept-Encoding: gzip

```
public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
        throws IOException, ServletException
{
    if (req instanceof HttpServletRequest){
        HttpServletRequest request = (HttpServletRequest) req;
        HttpServletResponse response = (HttpServletResponse) resp;
        String ae = request.getHeader("accept-encoding");
        if (ae != null && ae.indexOf("gzip") != -1){
            GZIPResponseWrapper wrappedResponse =
                    new GZIPResponseWrapper(response);
            chain.doFilter(req, wrappedResponse);
            wrappedResponse.finishResponse();
            return;
        }
        chain.doFilter(req, resp);
    }
}
```

Check that the caller accepts gzip encoding. If so create a ResponseWrapper that will zip up the response

Compression Wrapper i

```
public class GZIPResponseWrapper extends HttpServletResponseWrapper
{
    protected HttpServletResponse origResponse = null;
    protected ServletOutputStream stream = null;
    protected PrintWriter writer = null;

    public GZIPResponseWrapper(HttpServletResponse response)
    {
        super(response);
        origResponse = response;
    }

    public void finishResponse() throws IOException {
        if (writer != null) writer.close();
        else if (stream != null) stream.close();
    }

    public void flushBuffer() throws IOException{
        stream.flush();
    }
    // ... next slide
```

finishResponse is a helper method called from doFilter

Compression Wrapper ii

```
  // continued
  public ServletOutputStream getOutputStream() throws IOException {
        if (writer != null){
            throw new IllegalStateException("...");
        }
        if (stream == null)
            stream = createOutputStream();
        return stream;
    }

    public PrintWriter getWriter() throws IOException {
        if (writer != null){return (writer);}
        if (stream != null){
            throw new IllegalStateException("...");
        }
        stream = createOutputStream();
        writer = new PrintWriter(stream);
        return writer;
    }

    protected ServletOutputStream createOutputStream() throws IOException{
        return new GZIPResponseStream(origResponse);
    }
}
```

Shown later

GZIPResponseStream i

Stream class is responsible for compression

- extends SerlvletOutputStream

```
public class GZIPResponseStream extends ServletOutputStream
{
    protected ByteArrayOutputStream baos = null;
    protected GZIPOutputStream gzipstream = null;
    protected boolean closed = false;
    protected HttpServletResponse response = null;
    protected ServletOutputStream output = null;
    protected Logger logger;

    public GZIPResponseStream(HttpServletResponse response) throws IOException {
        super();
        this.response = response;
        this.output = response.getOutputStream();
        baos = new ByteArrayOutputStream();
        gzipstream = new GZIPOutputStream(baos);
    }

    // continues
```

Must extend ServletOutputStream

java.util.zip

GZIPResponseStream ii

```
    // continued
    public void close() throws IOException {
        if (closed) {
            throw new IOException("This output stream has already been closed");
        }
        gzipstream.finish();

        byte[] bytes = baos.toByteArray();

        response.addHeader("Content-Encoding", "gzip");
        response.addHeader("Content-Length", Integer.toString(byt

        output.write(bytes); output.flush();output.close();
        closed = true;
    }

    public void write(byte b[], int off, int len) throws IOException {
        if (closed){
            throw new IOException("Cannot write to a closed outpu
        }
        gzipstream.write(b, off, len);
    }
}
```

Make sure the correct headers are set

output is the ServletOutputStream

Also need to override the other write methods

Filters under Forward and Include

Default behaviour for filters is to not work under forward/include

Servlet 2.4 specification introduces this ability

```
<filter-mapping>
  <filter-name>Logging Filter</filter-name>
  <servlet-name>/sales/*</servlet-name>
  <dispatcher>INCLUDE</dispatcher>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

Annotations

@WebFilter

- name
- urlPatterns (required)
- dispatcherType
- servletNames
- asyncSupported
- … and others

@WebInitParam

- Used to parameterize the filter

Annotation Example

```
@WebFilter("/*")
public class ZipFilter implements Filter {
}
```

```
@WebFilter(servletName = {"SimpleServlet", "ControllerServlet"})
public class ZipFilter implements Filter {
}
```

```
@WebFilter(urlPatterns = "*.do")
public class ZipFilter implements Filter {
}
```

```
@WebFilter(
    urlPatterns = "*.do",
    initParams = @WebInitParam(name="fileTypes", value="docx;xlsx"))
public class ZipFilter implements Filter {
}
```

```
@WebFilter(
    urlPatterns = "*.do",
    dispatcherTypes = {DispatcherType.REQUEST, DispatcherType.FORWARD} )
public class ZipFilter implements Filter {
}
```

Other Uses

Controller in MVC application

- Filters can replace Servlets in this role

Security

- Can provide a security layer to an application

Summary

Filters allow us to add services to Web applications

Extremely powerful addition to the servlet specification

Request Wrappers allow us to change the request data that a resource sees

Response Wrappers allow us to filter responses before they are sent to the client

What's Next