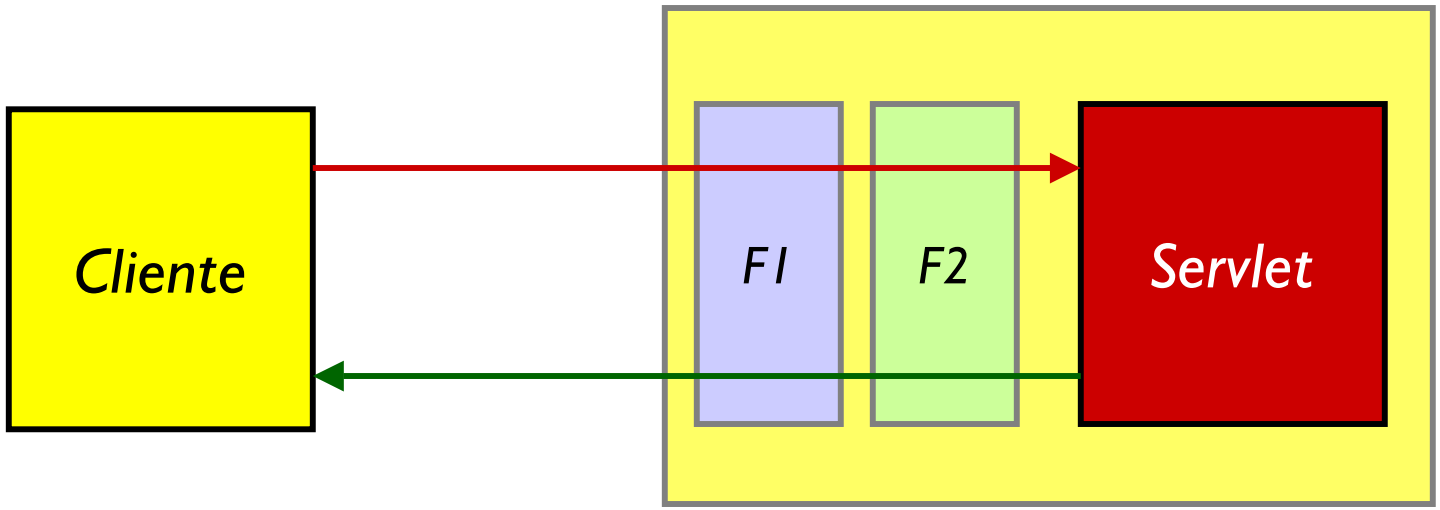
Helder da Rocha (helder@acm.org)
www.argonavis.com.br

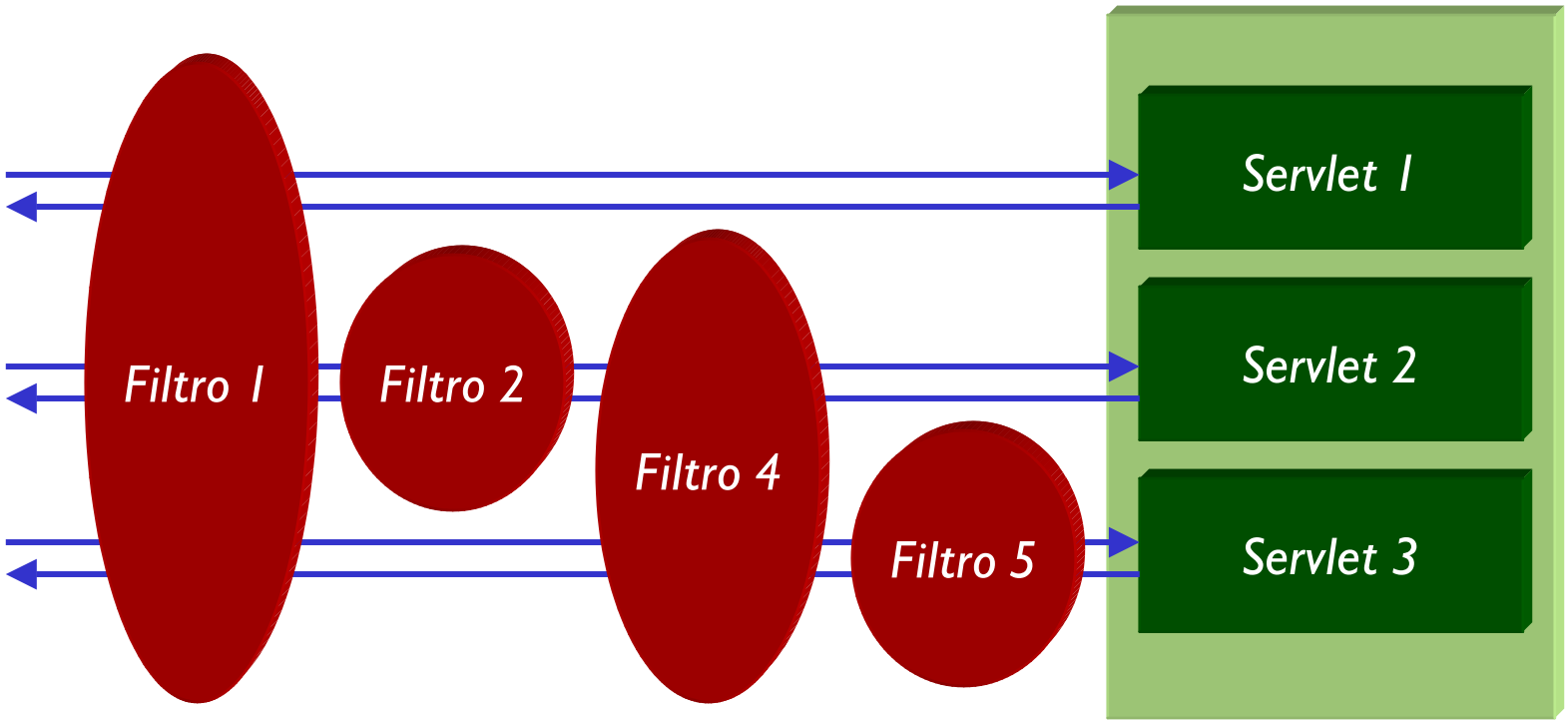
O que são Filtros

- Um *filtro* é um *componente Web* que reside no servidor
  - *Intercepta as requisições e respostas no seu caminho até o servlet e de volta ao cliente*
  - *Sua existência é ignorada por ambos. É totalmente transparente tanto para o cliente quanto para o servlet*
  - *Suportado desde a versão 2.3 da especificação de Servlets*
- *Filtros podem ser concatenados em uma corrente*
  - *Neste cenário, as requisições são interceptadas em uma ordem e as respostas em ordem inversa*



Filtros

- *Um filtro pode realizar diversas transformações, tanto na resposta como na requisição antes de passar esses objetos adiante (se o fizer)*
- *Filtros podem ser reutilizados em vários servlets*



Para que servem?

- Filtros permitem
  - Tomada de decisões: podem decidir se repassam uma requisição adiante, se redirecionam ou se enviam uma resposta interrompendo o caminho normal da requisição
  - Tratamento de requisições e respostas: podem empacotar uma requisição (ou resposta) em outra, alterando os dados e o conteúdo dos cabeçalhos
- Aplicaçoes típicas
  - Autenticação
  - Conversão de caracteres, MIME types, tokenizing
  - Conversão de imagens, compressão e decompressão
  - Criptografia
  - Transformação XSLT

Como funcionam?

- Quando o container recebe uma requisição, ele verifica se há um filtro associado ao recurso solicitado. Se houver, a requisição é roteada ao filtro
- O filtro, então, pode
  1. Gerar sua própria resposta para o cliente
  2. Repassar a requisição, modificada ou não, ao próximo filtro da corrente, se houver, ou ao recurso final, se ele for o último filtro
     - Rotear a requisição para outro recurso
- Na volta para o cliente, a resposta passa pelo mesmo conjunto de filtros em ordem inversa

API: Interfaces Filter, FilterConfig, FilterChain

- javax.servlet.Filter
  - void init(FilterConfig),
  - void doFilter(ServletRequest, ServletResponse, FilterChain)
  - void destroy()
- javax.servlet.FilterConfig
  - String getFilterName()
    String getInitParameter(String name)
  - Enumeration getInitParameterNames()
  - ServletContext getServletContext()
- javax.servlet.FilterChain
  - void doFilter(ServletRequest, ServletResponse)

API: Classes empacotadoras

- Úteis para que filtros possam trocar uma requisição por outra
  - Uma subclasse dessas classes empacotadoras pode ser passada em uma corrente de filtros no lugar da requisição ou resposta original
  - Métodos como *getParameter()* e *getHeader()* podem ser sobrepostos para alterar parâmetros e cabeçalhos
- No pacote *javax.servlet*
  - *ServletRequestWrapper implements ServletRequest*: implementa todos os métodos de ServletRequest e pode ser sobreposta para alterar o request em um filtro
  - *ServletResponseWrapper implements ServletResponse*: implementa todos os métodos de ServletResponse
- No pacote *javax.servlet.http*
  - *HttpServletRequestWrapper* e *HttpServletResponseWrapper*: implementam todos os métodos das interfaces correspondentes, facilitando a sobreposição para alteração de cabeçalhos, etc.

Como escrever um filtro simples

1. Escreva uma classe implementando a interface Filter e todos os seus métodos
   - init(FilterConfig)
   - doFilter(ServletRequest, ServletResponse, FilterChain)
   - destroy()
2. Compile usando o JAR da Servlet API
3. Configure o filtro no deployment descriptor (web.xml) usando os elementos <filter> e <filter-mapping>
   - Podem ser mapeados a URLs, como servlets
   - Podem ser mapeados a servlets, para interceptá-los
   - A ordem dos mapeamentos é significativa
4. Implante o filtro da maneira usual no servidor

Filtro simples que substitui servlet

```
package j550.filtros;
import java.io.*;
import javax.servlet.*;

public class HelloFilter implements Filter {
  private String texto;
  public void init(FilterConfig config) {
    texto = config.getInitParameter("texto");
  }
  public void doFilter(ServletRequest request,
                       ServletResponse response,
                       FilterChain filterChain)
                        throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      out.println("<HTML><HEAD><TITLE>Filter Response");
      out.println("</TITLE></HEAD><BODY>");
      out.println("<H1>Filter Response</H1>");
      out.println("<P>" + texto);
      out.println("</BODY></HTML>");
      out.close();
   }
   public void destroy() {}
}
```

Configuração

- *Os elementos <filter> e <filter-mapping> são quase idênticos aos equivalentes para <servlet>*
  - *A diferença é que <filter-mapping> é usado também para associar filtros a servlets, na ordem em que aparecem*
- *Filtro simples, que substitui um servlet*

```
<filter>
    <filter-name>umFiltro</filter-name>
    <filter-class>j550.filtros.HelloFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>umFiltro</filter-name>
    <url-pattern>/filtro</url-pattern>
</filter-mapping>
```

- *Filtro que intercepta um servlet*

```
<filter-mapping>
    <filter-name>umFiltro</filter-name>
    <servlet-name>umServlet</servlet-name>
</filter-mapping>
```

Filtros "de verdade"

- *Filtros úteis podem ser encadeados em uma corrente. Para que isto seja possível, devem chamar doFilter() no objeto FilterChain - parâmetro no seu próprio doFilter()*

```
public void doFilter(...req,...res, FilterChain chain) {
    ...
    chain.doFilter(req, res);
    ...
}
```

- *Antes da chamada ao doFilter(), o filtro pode processar a requisição e alterar ou substituir os objetos ServletRequest e ServletResponse ao passá-los adiante*

```
ServletRequest newReq = new ModifiedRequest(...);
chain.doFilter(newReq, res);
```

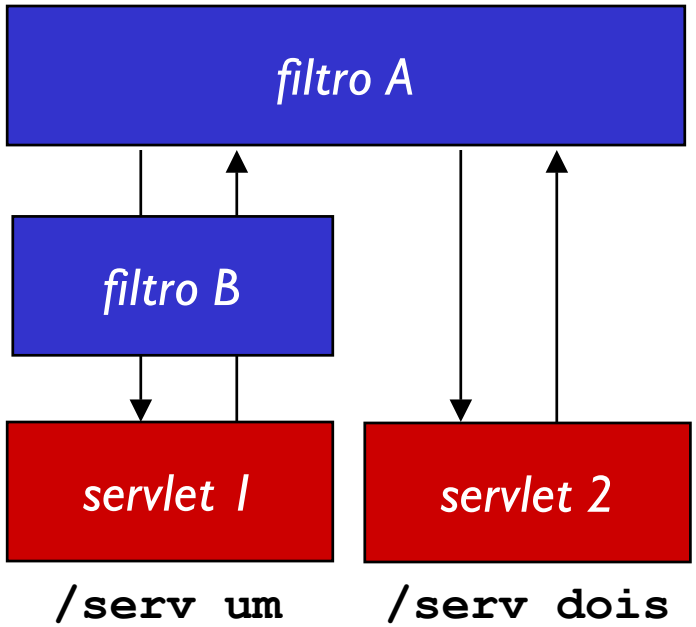
Estende ServletRequestWrapper

- *Na volta, opera sobre a resposta e pode alterá-la*

Configuração da corrente

- *A corrente pode ser configurada com definição das instâncias de filtros e mapeamentos em ordem*

```
<filter>
    <filter-name>filtroA</filter-name>
    <filter-class>j550.filtros.FilterA</filter-class>
</filter>
<filter>
    <filter-name>filtroB</filter-name>
    <filter-class>j550.filtros.FilterB</filter-class>
</filter>
<filter-mapping>
    <filter-name>filtroA</filter-name>
    <url-pattern>/serv_um</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>filtroA</filter-name>
    <servlet-name>servlet2</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>filtroB</filter-name>
    <servlet-name>servlet1</servlet-name>
</filter-mapping>
```



Filtros que tomam decisões

- *Um filtro pode ler a requisição e tomar decisões como transformá-la, passá-la adiante ou retorná-la*

```
public void doFilter(.. request, ... response, ... chain) {
   String param = request.getParameter("saudacao");
   if (param == null) {
      PrintWriter out = response.getWriter();
      response.setContentType("text/html");
      out.println("<h1>Erro!</h1>");
   } else if (param.equals("Bonjour!")) {
      class MyWrapper extends ServletRequestWrapper { // ...
        public String getParameter(String name) {
          if (name.equals("saudacao")) {
             return "Bom Dia";
          }
        }
      }
      ServletRequest myRequest = new MyWrapper(request);
      chain.doFilter(myRequest, response);
   } else { chain.doFilter(request, response); }
}
```

Classe interna (o construtor (abaixo) foi omitido da declaração por falta de espaço)

Wrappers

- *Sobrepondo um HttpServletRequest*

```
public class MyServletWrapper
                              extends HttpServletRequestWrapper {
   public MyServletWrapper(HttpServletRequest req) {
      super(req);
   }
   public String getParameter(String name) {
      return super.getParameter(name).toUpperCase();
   }
}
```

- *Usando Wrappers em servlets HTTP*

```
HttpServletRequest req = (HttpServletRequest)request;
HttpServletResponse res = (HttpServletResponse)response;
HttpServletRequest fakeReq = new MyServletWrapper(req);
HttpServletResponse fakeRes = new TestResponse(res);

chain.doFilter(fakeReq, fakeRes);
```

Observações importantes

- Para filtros usados com servlets HTTP, o request e response passados são HttpServletRequest e HttpServletResponse
  - Wrappers devem estender as classes que implementam essas interfaces
- Filtros não são chamados quando o recurso que interceptam for chamado através de um RequestDispatcher
  - O recurso é acessado diretamente sem filtragem
  - Isto ocorre para evitar loops infinitos
- Filtros associados a páginas de erro também não são chamados

Exercícios

1. Escreva um filtro simples que leia a requisição e verifique se ela contém os parâmetros usuario e senha
   - Se não tiver, repasse a requisição para a página erro.html
   - Se tiver, abra o arquivo usuarios.txt usando a classe Properties. Ele possui uma lista de nome=senha, um por linha. Veja se o usuário coincide com a senha. Se sim, chame o próximo filtro. Se não, redirecione para acessoNegado.html
   - Associe o filtro a um servlet qualquer (o SimpleServlet, por exemplo)
   - Acesse o servlet e verifique que ele passa pelo filtro
- 2. Escreva dois RequestWrappers que encapsulam HttpServletRequest e sobrepõem o método getParameter()
   - Use o primeiro em um filtro chamado UpperCaseFilter, que coloque os valores de todos os parâmetros em caixa-alta.
   - O segundo, ReverseFilter, deve inverter o texto dos parâmetros
   - Coloque os dois em cascata apontando para um servlet simples.

Exercícios (2)

- 3. Escreva um filtro chamado StyleFilter para alterar o estilo do texto passado como parâmetro
  - a) O `estilo` será definido como parâmetro inicial (<init-param>) do filtro em linguagem CSS. Se for recebido um texto original
    `texto`
    o filtro deve devolver
    `<span style='estilo'>texto</span>`
- 4. Configure duas instâncias do filtro (boldFilter e blueFilter)
  - a) Uma passando o estilo: "`font-weight: bold`"
  - b) Outra passando o estilo: "`color: blue`"
- 5. Crie duas instâncias e mapeamentos no web.xml para SimpleServlet e associe-os às instâncias de filtros
  - a) Mapeie uma instância à URL /bluebold e outra a /blue
  - b) Associe boldFilter e blueFilter, nesta ordem, a /bluebold
  - c) Associe blueFilter e UpperCaseFilter (exercício 2) a /blue
  - d) Verifique a ordem de chamada no código-fonte gerado no browser

helder@acm.org

argonavis.com.br