

Jakarta EE Web Total

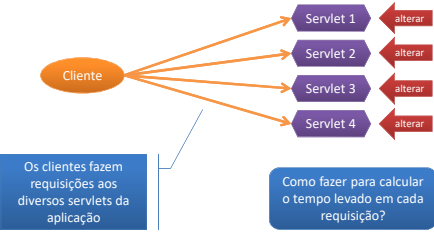
Conhecendo os Filters

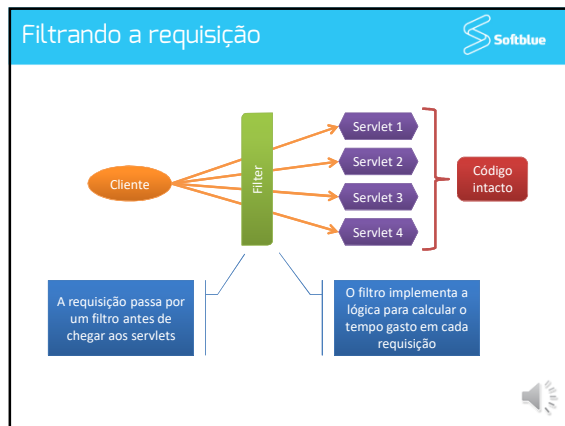


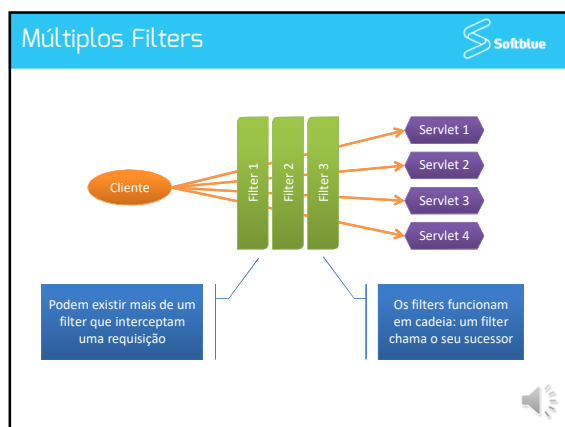
Tópicos Abordados

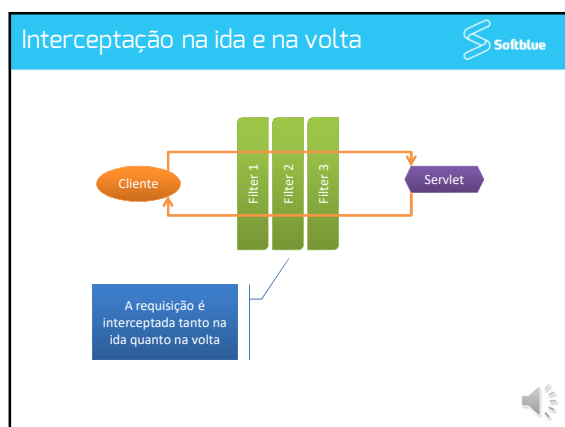
- Filtrando a requisição
- Múltiplos filters
- Intercepção na requisição e na resposta
- Exemplos de filters
- Criando um filter
- Configurando um filter
- Ordem de carregamento
- AOP (Aspect Oriented Programming)

A requisição direta aos Servlets





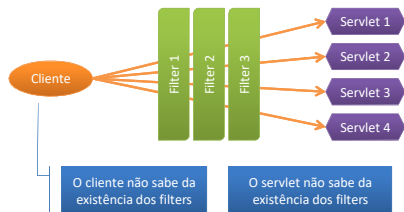




Os Filters são transparentes



O filter não sabe da existência de outros filters



O cliente não sabe da existência dos filters

O servlet não sabe da existência dos filters



Os Filters são transparentes



- Ninguém sabe que os filters existem
 - A não ser o container, que gerencia a chamada dos filters
- Isto permite um desenvolvimento totalmente desacoplado
 - Filters podem ser colocados e retirados da cadeia de filters sem que seja necessário alterar código



Exemplos de Filters



- Alguns exemplos do que pode ser feito com o uso de filters
 - Log
 - Auditoria
 - Verificação de segurança
 - Alterações na request e na response



Criando um Filter



- A implementação de um filter é feita através da implementação da interface `javax.servlet.Filter`

Método	Quando é chamado...
<code>init()</code>	Quando o filter é iniciado
<code>destroy()</code>	Quando o filter é removido
<code>doFilter()</code>	Quando o filter intercepta a requisição

- Assim como servlets, apenas uma instância de um filter existe em uma aplicação



Criando um Filter



```
public class MyFilter implements Filter {  
    private FilterConfig config;  
  
    public void init(FilterConfig config) throws ServletException {  
        //armazena o objeto FilterConfig  
        this.config = config;  
    }  
  
    public void destroy() {  
        //limpeza de recursos  
    }  
  
    public void doFilter(ServletRequest request,  
        ServletResponse response,  
        FilterChain chain) throws IOException, ServletException {  
        //faz algo antes de repassar a request  
        chain.doFilter(request, response);  
        //faz algo depois de receber a response  
    }  
}
```



Configurando um Filter



- O filter deve ser registrado no `web.xml` para que o container possa carregá-lo

```
<web-app>  
    <filter>  
        <filter-name>LogFilter</filter-name>  
        <filter-class>filter.LogFilter</filter-class>  
    </filter>  
  
    <filter>  
        <filter-name>SecurityFilter</filter-name>  
        <filter-class>filter.SecurityFilter</filter-class>  
    </filter>  
</web-app>
```



Configurando um Filter



- Como o container sabe quando invocar determinado filter?
 - Através de mapeamentos de URL
 - Da mesma forma como acontece com servlets

```
<web-app>
  <filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>SecurityFilter</filter-name>
    <url-pattern>/admin/*</url-pattern>
  </filter-mapping>
</web-app>
```

Intercepta todas as requisições

Intercepta requisições em /admin



Ordem de Carregamento



- Se mais de um filter for mapeado para a mesma URL, o container usa a ordem da declaração dos filters no *web.xml* para definir a cadeia

```
<web-app>
  <filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>AuditFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

A ordem de carregamento é LogFilter e AuditFilter



Filters e Annotations



- Filters também podem ser configurados através de annotations

```
LogFilter
@WebFilter("/*")
public class LogFilter implements Filter {
    public void doFilter(ServletRequest request,
        ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        //código do filter
    }

    public void init(FilterConfig fConfig) throws
        ServletException { }

    public void destroy() { }
}
```

@WebFilter define um filter e o mapeamento

A definição de ordem só é possível através do web.xml



Aspect Oriented Programming



- Programação orientada a aspectos
 - AOP
- Permite plugar funcionalidades no código
 - Estas funcionalidades são chamadas de aspectos
- O código não possui conhecimento a respeito dos aspectos, o que permite habilitá-los e desabilitá-los sem que seja necessária qualquer alteração no código





Softblue