Listeners e Filters — Notas de Aula

Flávio Velloso Laper

7 de maio de 2009

Listeners

Eventos do ciclo de vida

- Pode-se monitorar o ciclo de vida de um servlet definindo objetos ouvintes (*listeners*) cujos métodos são chamados quando os eventos ocorrem.
- Interfaces e classes para controle do ciclo de vida de um servlet:
  - ServletContextListener (e ServletContextEvent): inicialização e destruição de um servlet.
  - ServletContextAttributeListener (e ServletContextAttributeEvent): criação, remoção e alteração de atributos do contexto.
- Interfaces e classes para controle do ciclo de vida de uma sessão:
  - HttpSessionListener (e HttpSessionEvent): criação, invalidação e timeout da sessão do cliente.
  - HttpSessionAttributeListener(e HttpSessionBindingEvent): criação, remoção e alteração de atributos de sessão.

Observação:

- Atributos podem ser gravados em quatro tipos de objetos de escopo que oferecem diferentes níveis de persistência (em ordem decrescente de duração):
  - Contexto da aplicação: dura enquanto a aplicação estiver ativa (ServletContext).
  - Sessão: dura enquanto a sessão do cliente for válida (HttpSession).
  - Requisição: dura enquanto uma requisição estiver sendo processada (ServletRequest).
  - Página JSP: dura enquanto uma página JSP estiver sendo procesada (PageContext).
- Para gravar dados em um objeto de escopo:

  escopo.setAttribute ("nome", objeto;
- Para recuperar os dados:

  Object dado = escopo.getAttribute("nome");

Exemplo de Listener

- O listener abaixo escuta eventos de contexto, apenas exibindo uma mensagem da saída padrão (declarações de importação omitidas).

```
public class HelloListener implements ServletContextListener {
  public void contextInitialized (ServletContextEvent e) {
    System.out.println("Servlet criado");
  }

  public void contextDestroyed (ServletContextEvent e) {
    System.out.println("Servlet destruído");
  }
}
```

- Cadastramento no web.xml:

```
<web-app>
  . . .
  <listener>
    <listener-class>HelloListener</listener-class>
  </listener>
  . . .
</web-app>
```

- O listener abaixo escuta eventos de contexto e de sessão, fazendo uma contagem do número de sessões da aplicação com a utilização de um atributo de contexto.

```
public class ContadorListener implements ServletContextListener,
    HttpSessionListener {
  private ServletContext context;
  public void contextInitialized (ServletContextEvent e) {
    context = e.getServletContext();
    context.setAttribute ("contador", 0);
  }

  public void contextDestroyed (ServletContextEvent e) {}

  public void sessionCreated (HttpSessionEvent e) {
    int cont = (Integer)context.getAttribute("contador");
    context.setAttribute("contador", ++cont);
  }

  public void sessionDestroyed (HttpSessionEvent e) {
    int cont = (Integer)context.getAttribute("contador");
    context.setAttribute("contador", --cont);
  }
}
```

- O servlet abaixo exibe o atributo mantido pelo listener:

```
1  public class ContadorServlet extends HttpServlet {
2    public void doGet (HttpServletRequest req, HttpServletResponse resp )
3        throws ServletException, IOException {
4      ServletContext context = getServletContext();
5      HttpSession session = req.getSession();
6      int cont =(Integer)context.getAttribute("contador");
7      resp.setContextType("text/html");
8      PrintWriter out = resp.getWriter();
9      out.println("<html><head><title>");
10     out.println("Contador");
11     out.println ("</title></head><body>");
12     out.println ("Número de usuários = " + cont);
13     out.println ("</body></html>");
14     out.close();
15   }
16 }
```

Filters

- Um filtro é um objeto que pode transformar o cabeçalho, o conteúdo, ou ambos, de uma requisição ou de uma resposta.
- Filtros interceptam requisições e/ou respostas.
- Filtros podem ser encadeados.
- Principais tarefas de um filtro:
  - Interromper o fluxo da requisição e/ou resposta.
  - Modificar os dados da requisição e/ou resposta.
  - Interagir com recursos externos.
- Aplicações típicas:
  - Autenticação (Ex: spring security).
  - Autorização (Ex: spring security).
  - Decoração (Ex: sitemesh).
  - Auditoria.
  - Transformações (criptografia, compressão, etc).
- Para utilizar um filtro é preciso:
  - Programar o filtro.
  - Programar as requisições e respostas filtradas.
  - Especificar a ordem de chamada para cada recurso (web.xml).
- Filtros podem ser usados para interceptar tanto a requisição quanto a resposta, ou ambos.
- Um mesmo filtro pode interceptar requisições/respostas para diversos servlets (ou JSP's).
- Diversos filtros pode interceptar requisições/respostas para o mesmo servlet (ou JSP). Neste caso os diferentes filtros formam uma cadeia (*chain*) entre o cliente e o servlet.

API dos filtros

- Um filtro deve implementar a interface Filter; outras interfaces importantes são FilterConfig e FilterChain .
- Filter:
  - **void** init(FilterConfig)
    - init() é executado uma única vez quando o filtro é criado. Deve executar operações de inicialização.
  - **void** doFilter(ServletRequest, ServletResponse, FilterChain)
    - doFilter() é chamado a cada requisição interceptada.
  - **void** destroy()
    - destroy() é executado uma única vez quando o filtro é destruído. Deve executar operações de finalização.
- FilterConfig dá acesso à configuração do filtro e ao container.
  - String getFilterName()
    - Retorna o nome cadastrado para o filtro.
  - String getInitParameter(String parameterName)
    - Recupera um parâmetro de inicialização.
  - Enumeration getInitParameterNames()
    - Recupera os nomes dos parâmetros de inicialização.
  - ServletContext getServletContext()
    - Recupera o contexto da aplicação.
- FilterChain: contém informações sobre a cadeia de filtros para um determinado servlet.
  - **void** doFilter(ServletRequest, ServletResponse)
    - Repassa a requisição/resposta para o próximo elo ou elemento encadeado.

Exemplo

```
1  public class HelloFilter implements Filter {
2    private FilterConfig config;
3
4    public void init(FilterConfig config) throws ServletException {
5      System.out.println("Filtro Iniciado");
6      this.config = config;
7    }
8
9    public void destroy() {
10     System.out.println("Filtro destruído");
11   }
12
13   public void doFilter(ServletRequest req, ServletResponse resp, FilterChain
       chain)
14       throws IOException, ServletException {
15     System.out.println("Requisição interceptada");
```

```
16        chain.doFilter(req, resp); // requisição prossegue para o elemento
              seguinte
17        System.out.println("Resposta interceptada");
18    }
19 }
```

- A chamada chain.doFilter() permite que a requisição siga para o próximo elemento da cadeia (pode ser o servlet ou um outro filtro).
  - Tudo o que for programado antes dessa chamada corresponde à interceptação da requisição.
  - Tudo o que for programado após essa chamada corresponde à interceptação da resposta.
  - Se a chamada não for executada, a requisição não prossegue.

Cadastramento no web.xml

```
1  <web-app>
2     . . .
3     <filter>
4        <filter-name>Hello</filter-name>
5        <filter-class>HelloFilter</filter-class>
6     </filter>
7
8     <filter-mapping>
9        <filter-name>Hello</filter-name>
10       <servlet-name>HelloServlet</servlet-name>
11    </filter-mapping>
12 </web-app>
```

- Observações:
  - O mapeamento especifica qual servlet será interceptado pelo filtro.
  - É possível também utilizar a tag <url-pattern> no lugar de <servlet-name> no mapeamento. Neste caso, serão interceptadas todas as requisições cujas URL's coincidirem com o padrão indicado.
  - Pode haver vários mapeamentos para o mesmo filtro.
  - Diferentes filtros podem ser mapeados para o mesmo filtro. A ordem do cadastramento indica a sequência de encadeamento.