Tecnologie di Sviluppo per il Web

Applicazioni Web J2EE
Altri Componenti

versione 3.2

Questo lavoro è concesso in uso secondo i termini di una licenza Creative Commons
(vedi ultima pagina)

SOME RIGHTS RESERVED creative commons

G. Mecca – mecca@unibas.it – Università della Basilicata

Sommario

- Filtri
- Costruire una Libreria di Tag

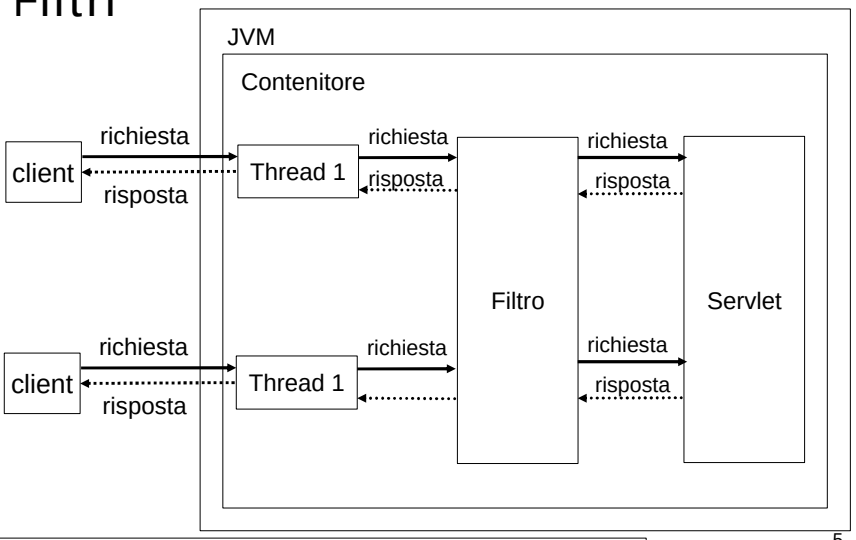
Filtri

- Riassumendo
  - un'applicazione Web J2EE è fatta essenzialmente di servlet
  - eventualmente scritti sotto forma di JSP
- Esiste una categoria speciale di servlet
  - i filtri
  - servlet pensati per modificare richieste e risposte di altri servlet

Filtri

- Filtro
  - servlet speciale che intercettano le richieste dirette ad altri servlet e le risposte prodotte da altri servlet
  - possono modificare la richiesta e la risposta
  - possono decidere se inoltrare o meno la richiesta

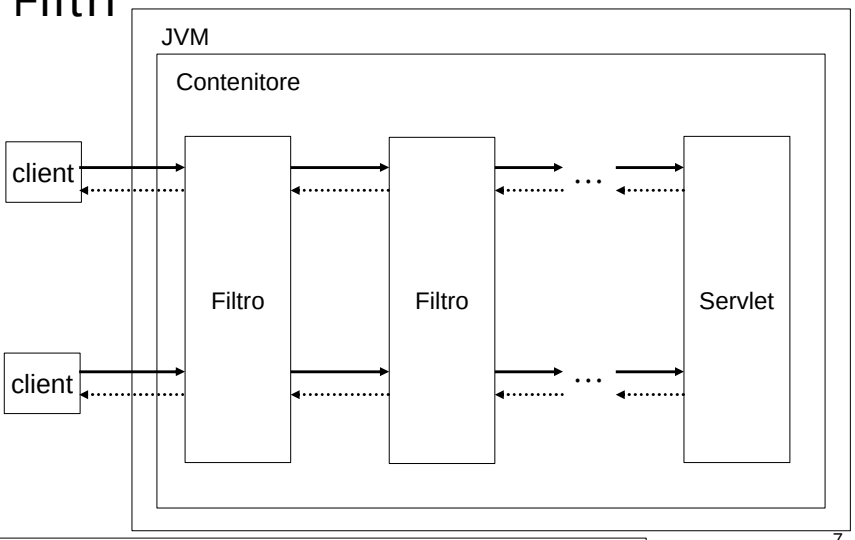
Filtri



Filtri

- Catene di filtri
  - ⇨un filtro può essere applicato ad un altro filtro
  - ⇨per creare catene di filtri, in cui ciascun filtro intercetta la richiesta rivolta al successivo
  - ⇨e poi, al contrario, la risposta che dal successivo ritorna verso il client

Filtri



Filtri

- Possibili usi dei filtri
  - ⇨in generale: modificare il comportamento di un gruppo di servlet collettivamente senza cambiare i servlet
  - ⇨impedire l'accesso ad alcuni servlet/jsp
  - ⇨tenere traccia delle richieste (es: contatori)
  - ⇨modificare la risposta di un gruppo di servlet (es: sostituzione di stringhe, sostituzione di tag HTML, compressione, cifratura, ecc.)

Filtri

- Interfaccia javax.servlet.Filter
  - ⇨tre metodi
  - ⇨void init(FilterConfig config)
    throws ServletException
  - ⇨void destroy()
  - ⇨void doFilter(ServletRequest request,
    ServletResponse response,
    FilterChain chain)
    throws ServletException, IOException

Filtri

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Filtro implements Filter {
    public void init(FilterConfig config) throws ServletException {}
    public void destroy() {}
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
                  throws IOException, ServletException {
      // operazioni da svolgere per filtrare le richieste
      // per inoltrare la richiesta
      // chain.doFilter(request, response)
    }
}
```

Filtri

- Nota
  - è necessario implementare i tre metodi
  - tipicamente init() e destroy() sono vuoti
  - doFilter() codifica il comportamento del filtro
  - può essere necessario effettuare il cast della richiesta su HttpServletResponse
  - al termine delle operazioni, per inoltrare la chiamata è necessario chiamare il metodo doFilter() sulla catena

Filtri

- Modifica della risposta
  - operazione più complessa
  - tipicamente il codice HTML stampato nella risposta viene inviato subito al client
- Strategia (cenni)
  - il filtro invia lungo la catena una “finta” risposta (ServletResponseWrapper)
  - la intercetta e invia la vera risposta al client
  - attenzione ai problemi di efficienza

Filtri

- Dichiarazione del filtro
  - ⇨descrittore dell'applicazione web.xml
- Elemento filter
  - ⇨nome e classe del filtro
- Elemento filter-mapping
  - ⇨nome del filtro
  - ⇨URL pattern a cui si applica

Filtri

- Un esempio
  - ⇨un filtro per verificare se nella sessione c'è un utente autenticato

```
<filter>
    <filter-name>FiltroAutorizzazione</filter-name>
    <filter-class>it.unibas.aci.FiltroAutorizzazione</filter-class>
</filter>
<filter-mapping>
    <filter-name>FiltroAutorizzazione</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

```
public class FiltroAutorizzazione implements Filter {

   public void init(FilterConfig config) throws ServletException {}
   public void destroy() {}

   public void doFilter(ServletRequest request, ServletResponse response,
                FilterChain chain) throws IOException, ServletException {
      Utente utente = (Utente)session.getAttribute(“utente");
      if (utente == null) {
         HttpServletRequest httpRequest = (HttpServletRequest)request;
         String path = httpRequest.getServletPath();
         if (!path.equals("/index.jsp") && !path.equals("/login.jsp")) {
            HttpServletResponse httpResponse =
                                    (HttpServletResponse)response;
            httpResponse.sendRedirect(“/acim1/index.jsp”);
            return
      }
      chain.doFilter(request,response);
   }
}
```


Costruire una Libreria di Tag

- Tag di azione predefiniti e tag standard
  - ⇨risolvono moltissime operazioni ricorrenti
- Per tutte le altre operazioni
  - ⇨è possibile definire i propri tag personalizzati
  - ⇨funzionalità supportata dalla versione 1.1
  - ⇨significativamente rivisitata nella versione 1.2

Costruire una Libreria di Tag

- Libreria di tag personalizzati
  - ⇨package di classi, una classe per ciascun tag
  - ⇨descrittore della libreria (“tag library descriptor”), file .tld
  - ⇨associa il nome del tag alla corrispondente classe
- Utilizzo
  - ⇨direttiva “taglib” nelle pagine Jsp

Costruire una Libreria di Tag

- API javax.servlet.jsp.tagext
  - ⇨classe astratta TagSupport
  - ⇨metodo int doStartTag(): operazioni corrispondenti al tag di apertura
  - ⇨metodo int doEndTag(): operazioni da eseguire al tag di chiusura
  - ⇨variabile implicita pageContext
  - ⇨varie modalità per il trattamento del corpo

Costruire una Libreria di Tag

- Trattamento del corpo
  - ⇨varie costanti
  - ⇨EVAL_BODY_INCLUDE
  - ⇨SKIP_BODY
  - ⇨EVAL_PAGE
  - ⇨SKIP_PAGE
  - ⇨EVAL_ BODY_AGAIN

Costruire una Libreria di Tag

- Descrittore della libreria
  - ⇨file xml
  - ⇨DTD fissato dalla specifica
  - ⇨estensione convenzionale .tld
  - ⇨dichiara per ciascun tag il nome, il modello di contenuto, la classe java che lo implementa, eventuali parametri

Costruire una Libreria di Tag

- Nella pagina jsp
  - direttiva taglib
  - <%@ taglib uri="*uriDescrittore*" prefix="*prefisso*" %>
  - uriDescrittore: riferiemento al tag .tld
  - prefisso: prefisso di namespace utilizzato nella pagina per distinguere i tag della libreria
  - forma dei tag: <*prefisso:nome*>

Costruire una Libreria di Tag

- Esempio
  - sviluppo di una libreria per operazioni frequenti
  - un tag per la stampa degli errori di convalida
  - un tag per disabilitare il caching
- I passi necessari
  - sviluppare la classe che implementa il tag
  - sviluppare il descrittore della libreria

```
public class TagErrori extends TagSupport {

   public int doStartTag() {
      HttpServletRequest request = (HttpServletRequest)pageContext.getRequest();
      JspWriter out = pageContext.getOut();
      java.util.List errori = (java.util.List)request.getAttribute("errori");
      if (errori != null && errori.size() != 0) {
         try {
            out.println("<h2>Errore: I dati sottomessi non sono corretti</h2>");
            java.util.Iterator it = errori.iterator();
            out.println("<ul>");
            while (it.hasNext()){
               String errore = (String)it.next();
               out.println("<li><strong>" + errore + "</strong></li>");
            }
            out.println("</ul>");
         } catch (java.io.IOException ioe) {
            System.out.println(ioe);
         }
      }
      return SKIP_BODY;
   }
}
```


```
package it.unibas.tags.esempio;

import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;

public class TagNoCache extends TagSupport {

   public int doStartTag() {
      HttpServletResponse response =
                  (HttpServletResponse)pageContext.getResponse();
      response.setHeader("Cache-Control", "no-cache");
      response.setHeader("Pragma", "no-cache");
      response.setHeader("Expires", "" + System.currentTimeMillis());
      return SKIP_BODY;
   }
}
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
      "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>0.1</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>esempio</short-name>
  <uri>http://www.db.unibas.it/users/mecca/esempio</uri>
  <description> Libreria di tag di esempio </description>

  <tag>
    <name>noCache</name>
    <tag-class>it.unibas.tags.esempio.TagNoCache</tag-class>
    <body-content>empty</body-content>
  </tag>

  <tag>
    <name>errori</name>
    <tag-class>it.unibas.tags.esempio.TagErrori</tag-class>
    <body-content>empty</body-content>
  </tag>

</taglib>
```


Costruire una Libreria di Tag

- Nelle pagine JSP
  - ⇨<%@ taglib uri="WEB-INF/esempio.tld“ prefix=“es" %>
  - ⇨<es:noCache />
  - ⇨<es:errori />
- Nota
  - ⇨un descrittore può avere un URI assoluto e non relativo

Costruire una Libreria di Tag

- URI assoluto
  - ⇨viene risolto guardando nei jar presenti nella cartella lib e cercando eventuali file .tld che corrispondono all'URI in questione contenuti nelle cartelle META-INF
- Esempio: JSTL
  - ⇨richiede semplicemente di specificare l'URI assoluto delle librerie di tag e la disponibilità dei jar

Sommario

- Filtri
- Costruire una Libreria di Tag

Termini della Licenza

- This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/1.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

- Questo lavoro viene concesso in uso secondo i termini della licenza “Attribution-ShareAlike” di Creative Commons. Per ottenere una copia della licenza, è possibile visitare http://creativecommons.org/licenses/by-sa/1.0/ oppure inviare una lettera all’indirizzo Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.