Programmazione Web

Lezione del 25 Marzo 2019

Docente: Novella Bartolini
Ricevimento: Mercoledi` 12.30-13.30
Via Salaria 113, terzo piano, stanza 309
Email: bartolini@di.uniroma1.it

URL REWRITING nella gestione delle sessioni

- Se il browser ha i cookie disabilitati come faccio a gestire la sessione sul server?
- Si deve fare in modo che **tutti i percorsi utilizzati dal browser appendano all'url l'identificativo di sessione** (tecnica detta di **URL rewriting**).
  - N.B. il programmatore non conosce l'ID di sessione che verrà usato

Per una gestione semplice e trasparente delle operazioni di URL rewriting si ricorre al metodo dell'interfaccia HttpServletResponse:

java.lang.String **encodeURL**(java.lang.String url)

Dalla documentazione:

Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.

URL rewriting

Si deve poter rispondere a due questioni fondamentali:

1. Quali sono le URL che l'utente utilizzerà in futuro?
2. Come possiamo costringere l'utente ad appendere a queste URL l'identificativo di sessione?

Non possiamo sapere con certezza che richieste verranno effettuate dall'utente, ma possiamo controllare quelle effettuate attraverso gli hyperlink presenti nella pagina.

URL rewriting

- Se il client rifiuta i cookie, è possibile chiedere al contenitore di appendere l'identificatore della sessione agli URL degli hyperlink presenti nel codice HTML della risposta
  - Pagine di risposta generate da sessioni diverse conterranno URL diversi (con diversi identificativi di sessione appesi all'url)
  - Se la navigazione dell'utente procederà attraverso gli hyperlink la sessione verrà mantenuta: il client riproporrà al server l'identificatore della sessione nelle successive richieste HTTP
  - Identificazione trasparente della sessione

URL rewriting tramite il metodo encodeURL(...)

- Attenzione: il contenitore **riscrive gli URL solo se lo sviluppatore lo richiede esplicitamente**.
- Per realizzare questa operazione si usa il metodo **`String encodeURL(String url)`** di **`HttpServletResponse`**
  - il parametro **`url`** rappresenta l'URL non riscritto
  - il risultato del metodo è l'URL riscritto dal contenitore con appeso l'ID di sessione
- Se il programmatore richiede la riscrittura degli URL il comportamento del container è quello di commutare automaticamente tra le due modalità:
  - Se il browser del client accetta i cookie la sessione viene gestita solo con i cookie
  - Se i cookie vengono rifiutati, viene attivata la riscrittura degli URL

URL rewriting

- Poiché la riscrittura dell'URL avviene in modo trasparente da parte del contenitore
  - Lo sviluppatore deve solo usare **`encodeURL()`**
- Il contenitore si preoccupa di adottare la riscrittura degli url **solo** nel caso in cui i cookie vengono rifiutati
- E' quindi conveniente utilizzare sempre **`encodeURL()`** per rendere più robusta la gestione delle sessioni

Logica di funzionamento del metodo encodeURL()

- L'interfaccia HttpServletResponse fornisce questo metodo:

  java.lang.String **encodeURL**(java.lang.String url)

Dalla documentazione:

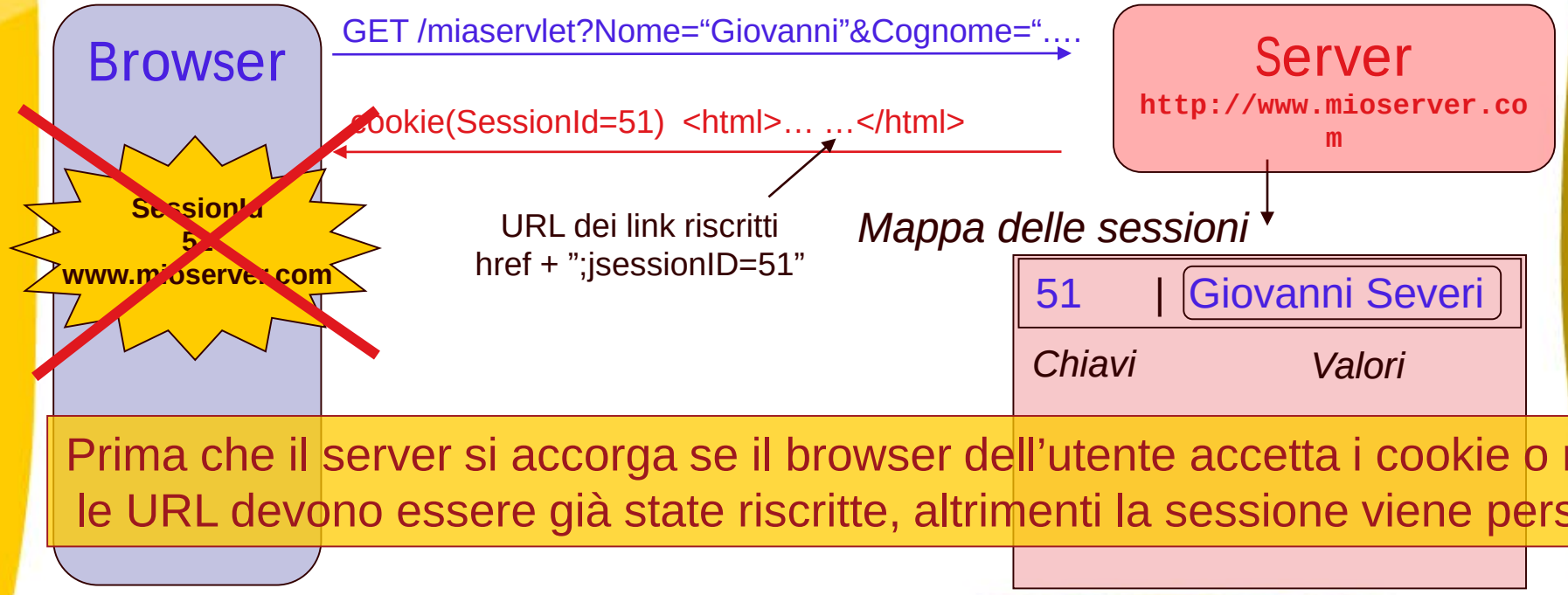
Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged

Logica di funzionamento del metodo encodeURL()

- Se il browser dell'utente accetta i cookie, il metodo lascia le URL inalterate
- Se il browser dell'utente NON accetta i cookie, il metodo effettua la riscrittura dell'URL passato come argomento
- COME FA QUESTO METODO A CAPIRE SE IL BROWSER DELL'UTENTE ACCETTA I COOKIE???

  Se la richiesta non contiene già dei cookie non ha modo di saperlo. Torniamo allora al primo esempio …

Logica di funzionamento del metodo encodeURL()



Prima che il server si accorga se il browser dell'utente accetta i cookie o no, le URL devono essere già state riscritte, altrimenti la sessione viene persa!

Logica di funzionamento del metodo encodeURL(): prima richiesta di una sessione

- All'atto della **creazione** dell'oggetto sessione, il **cookie** di sessione viene **sempre automaticamente aggiunto** all'oggetto rappresentativo della risposta
- La prima volta che viene utilizzato il metodo encodeURL nel corso di una sessione, il container può non sapere se il browser accetti i cookie o no.
- Al primo utilizzo del metodo encodeURL vengono inviati sia il cookie di sessione che le URL riscritte (solo quelle per cui lo sviluppatore avrà richiesto la riscrittura).

Logica di funzionamento del metodo encodeURL(): richieste successive alla prima di una sessione

- All'arrivo di richieste successive a quella che ha generato la sessione corrente, viene controllato se l'ID di sessione è stato ottenuto 1) anche (o solo) tramite un cookie o 2) non è stato ottenuto da un cookie.
  - Nel primo caso non viene effettuato encoding,
  - Nel secondo caso l'URL viene riscritta con appeso l'ID di sessione.

Supporto dell'interfaccia HttpServletRequest alle operazioni di encoding dell'URL

- L'interfaccia HttpSerlvetRequest fornisce i seguenti metodi:
  - boolean **isRequestedSessionIdFromCookie()**

    Checks whether the requested session ID came in as a cookie.

  - boolean **isRequestedSessionIdFromURL()**

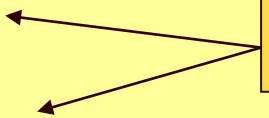
    Checks whether the requested session ID came in as part of the request URL.

REDIREZIONE DI RICHIESTE (ripasso)

Inoltro delle richieste ad altre risorse

- Servlet di esempio **RedirectServlet**
  - Inoltra la richiesta ad una risorsa diversa
  - Fa uso del metodo sendRedirect() dell'oggetto HttpServletResponse
  - Il parametro di ingresso è una stringa: il percorso di destinazione della ridirezione
    - Il metodo accetta percorsi assoluti e relativi.
    - Un percorso relativo senza "/" iniziale viene interpretato rispetto alla URL corrente
    - Un percorso relativo con la "/" iniziale viene interpretato rispetto alla dir. radice del container

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!--  RedirectServlet.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9    <title>Redirecting a Request to Another Site</title>
10 </head>
11
12 <body>
13   <p>Click a link to be redirected to the appropriate page</p>
14   <p>
15   <a href = "/DirectoryDiSaluto/redirect?page=deitel">
16     www.deitel.com</a><br />
17   <a href = "/DirectoryDiSaluto/redirect?page=welcome1">
18     Welcome servlet</a>
19   </p>
20 </body>
21 </html>
```



Fornisce hyperlink che invocano **RedirectServlet**

```
1
2  // Redirecting a user to a different Web page.
3
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class RedirectServlet extends HttpServlet {
10
11    // process "get" request from client
12    protected void doGet( HttpServletRequest request,
13      HttpServletResponse response )
14        throws ServletException, IOException
15    {
16      String location = request.getParameter( "page" );
17
18      if ( location != null )
19
20        if ( location.equals( "deitel" ) )
21          response.sendRedirect( "http://www.deitel.com" );
22        else
23          if ( location.equals( "welcome1" ) )
24            response.sendRedirect( "welcome1" );
25
26        // codice che viene eseguito solo se questa servlet non riesce a redirigere
27        // la richiesta come voluto
28
29      response.setContentType( "text/html" );
30      PrintWriter out = response.getWriter();
31
```

Attenzione all'uso di percorsi relativi all'interno di una servlet:

per non sbagliare pensate sempre attentamente a come viene formulato l'url per intero una volta che viene passato al browser

Il percorso relativo welcome1 viene aggiunto al percorso della servlet chiamante il metodo sendRedirect

Viene ottenuto il parametro **page** dalla richiesta.

Si valuta se il valore di **page** sia "deitel" o "welcome1"

Si inoltra la richiesta all'url: www.deitel.com.

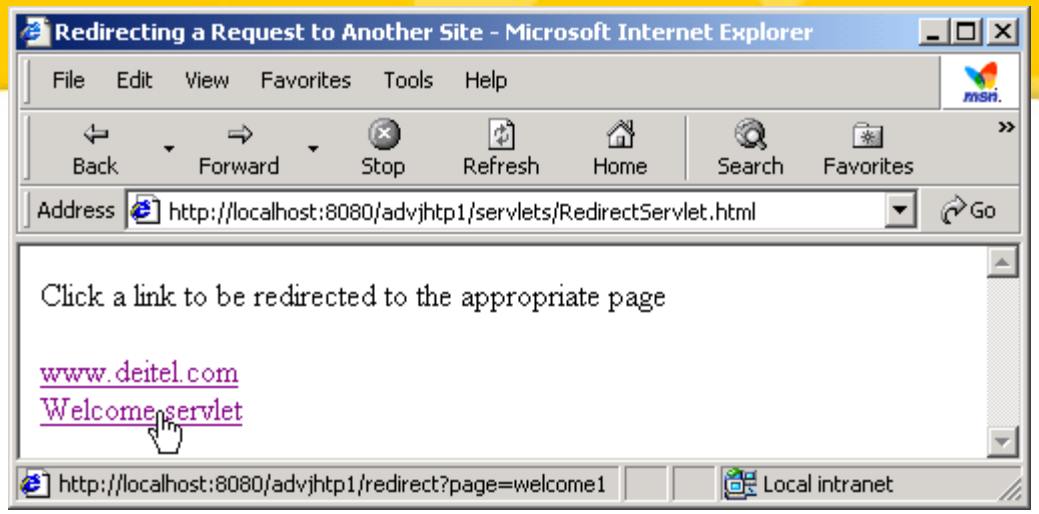
Si inoltra la richiesta al servlet **WelcomeServlet1**.

Pagina web di output che viene visualizzata nel caso in cui si sia ricevuta una richieta non valida (non viene invocato il metodo **sendRedirect**).

```
32      // start XHTML document
33      out.println( "<?xml version = \"1.0\"?>" );
34
35      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
36        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
37        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
38
39      out.println(
40        "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
41
42      // head section of document
43      out.println( "<head>" );
44      out.println( "<title>Invalid page</title>" );
45      out.println( "</head>" );
46
47      // body section of document
48      out.println( "<body>" );
49      out.println( "<h1>Invalid page requested</h1>" );
50      out.println( "<p><a href = " +
51        "\"servlets/RedirectServlet.html\">" );
52      out.println( "Click here to choose again</a></p>" );
53      out.println( "</body>" );
54
55      // end XHTML document
56      out.println( "</html>" );
57      out.close();  // close stream to complete the page
58    }
59  }
```

Si noti come l'invocazione di altre risorse facenti parte della stessa web application non richieda di specificare esplicitamente la context root.

Si assume che la servlet invocata si trovi nella stessa context root a meno che non venga specificata una URL completa.


Redirecting a Request to Another Site - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/servlets/RedirectServlet.html
Go
Click a link to be redirected to the appropriate page
www.deitel.com
Welcome servlet
http://localhost:8080/advjhtp1/redirect?page=welcome1
Local intranet


A Simple Servlet Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/welcome1
Go
Welcome to Servlets!
Done
Local intranet

Inoltro delle richieste ad altre risorse: modifiche al file web.xml

Descriptor element	Value
servlet *element*	
servlet-name	**redirect**
description	**Redirecting to static Web pages and other servlets.**
servlet-class	**RedirectServlet**
servlet-mapping *element*	
servlet-name	**redirect**
url-pattern	**/redirect**

??????

Come mai per redirigere una richiesta faccio uso di un metodo dell'oggetto HttpServletResponse response**?**

Alcune precisazioni...

1. L'oggetto della classe **`HttpServletResponse`** su cui viene invocato il metodo **`sendRedirect(String location)`** viene comunque utilizzato dal web server per costruire la risposta HTTP.
2. La risposta HTTP contiene un header di redirezione verso la nuova locazione
   <meta http-equiv="refresh" content="0" url=http://example.com/" />
3. L'header di redirezione viene interpretato dal browser del client
4. Il client spedisce automaticamente una nuova richiesta verso la nuova locazione.

Alcune precisazioni (continua)

- La locazione può anche essere esterna alla applicazione web da cui viene invocato il metodo (il metodo sendRedirect accetta sia percorsi relativi che assoluti)
- La redirezione coinvolge il client (che può decidere di non accettare redirezioni)
- Può essere utilizzata esclusivamente per richieste di GET (la specifica HTTP richiede che tutte le richieste di redirezione debbano essere inoltrate attraverso richieste di GET) – non si può usare sendRedirect per inviare richieste di POST
- I parametri della richiesta sono visibili al client (appesi all'URL nella richiesta di GET)

Documentazione dell'interfaccia HttpServletResponse, metodo sendRedirect

- **sendRedirect**
- public void **sendRedirect**(java.lang.String location) throws java.io.IOException
  - Sends a temporary redirect response to the client using the specified redirect location URL. This method can accept relative URLs; the servlet container must convert the relative URL to an absolute URL before sending the response to the client. If the location is relative without a leading '/' the container interprets it as relative to the current request URI. If the location is relative with a leading '/' the container interprets it as relative to the servlet container root. If the response has already been committed, this method throws an IllegalStateException. After using this method, the response should be considered to be committed and should not be written to.
  - **Parameters:**
    - location - the redirect location URL
  - **Throws:**
    - java.io.IOException - If an input or output exception occurs
    - IllegalStateException - If the response was committed or if a partial URL is given and cannot be converted into a valid URL

Redirezione di richieste (new)

Configurazione di una servlet

- Il container utilizza un oggetto corrispondente all'interfaccia ServletConfig per passare informazioni alla servlet nel momento della sua creazione
- Si può ottenere tale oggetto per una specifica servlet con il metodo Servlet.getServletConfig()
- **Metodi:**
  - java.lang.String **getInitParameter**(java.lang.String name)
    Fornisce una stringa corrispondente al valore del parametro indicato.
  - java.util.Enumeration **getInitParameterNames**()
    Fornisce un elenco di nomi di parametri di inizializzazione
  - java.lang.String **getServletName**()
    Fornisce il nome dell'istanza corrente della servlet
  - ServletContext **getServletContext**()
    Fornisce il riferimento al contesto operativo in cui viene eseguita la servlet chiamante (segue).

Contesto di una servlet (1)

- L'oggetto ServletContext è contenuto nell'oggetto ServletConfig creato e associato ad una servlet al momento della sua creazione.
- Viene definito attraverso l'interfaccia

  Interface ServletContext
- Definisce un insieme di metodi che una servlet usa per comunicare con il proprio container (il motore servlet)
- Esiste un solo contesto per ciascuna web application
- Nel servlet container esistono uno o più contesti servlet e ogni servlet deve essere contenuta in un contesto

Contesto di una servlet (2)

- Il contesto delle servlet è un contenitore di oggetti condivisi e può essere usato per comunicazioni tra servlet di una stessa web application

- Esempio: per trasferire una richiesta da una servlet ad un'altra dello stesso contesto si può usare il metodo di ServletContext:
  getRequestDispatcher(java.lang.String path)
  che fornisce un oggetto RequestDispatcher associato al percorso path

Interface RequestDispatcher (1)

- Si tratta di un oggetto che riceve richieste da un client e le inoltra a qualsiasi risorsa (servlet, pagine HTML o JSP) sul server.
- Un oggetto che implementa l'interfaccia RequestDispatcher viene richiamato attraverso il metodo
  - ServletContext.getRequestDispatcher("<URL>");

Interface RequestDispatcher: forward()

- public void **forward**(ServletRequest request, ServletResponse response)
- Inoltra una richiesta da una servlet ad un'altra risorsa (servlet o pagina JSP o HTML) sullo stesso server.
- Se l'oggetto RequestDispatcher è stato ottenuto attraverso una chiamata del metodo getRequestDispatcher(<URL>), il percorso di destinazione dell'oggetto ServletRequest è stato riconfigurato con l'<URL> specificato.
  - Il metodo forward deve essere chiamato prima che l'oggetto risposta venga inviato altrimenti si genera una IllegalStateException.
  - **Parametri:**
    - richiesta – un oggetto ServletRequest: la richiesta ricevuta dal client
    - risposta – un oggetto ServletResponse: la risposta che deve pervenire al client
- Questo metodo consente di effettuare un'elaborazione preliminare della richiesta da parte di una risorsa e demandare l'elaborazione definitiva della risposta ad un'altra risorsa.

Interface RequestDispatcher: include()

- public void **include**(ServletRequest request, ServletResponse response)
  - Include il contenuto di una risorsa (servlet o pagina JSP o HTML) nella risposta.
  - L'oggetto ServletResponse è lo stesso utilizzato dalla risorsa chiamante e i percorsi non vengono riconfigurati perché il client riceve la risposta dalla servlet che ha eseguito il metodo include.
  - **Parameters:**
    - richiesta – un oggetto ServletRequest: la richiesta ricevuta dal client
    - risposta – un oggetto ServletResponse: la risposta che deve pervenire al client

Inoltrare una richiesta da una servlet ad un'altra risorsa (servlet, jsp, html)

- L'oggetto RequestDispatcher fornisce un metodo alternativo al metodo ServletResponse.sendRedirect(<URL>)

- 1. Si invoca il metodo **getRequestDispatcher** di **ServletContext**
  - Si fornisce la URL relativa al server o alla applicazione web (NO PERCORSI ASSOLUTI!)

```
String url = "/welcome1";
RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher(url);
```

Inoltrare una richiesta... metodo alternativo

- 2. Si invoca il metodo **`forward`** per trasferire il controllo completo alla pagina di destinazione
  - non è prevista nessuna comunicazione con il client, al contrario di quanto avviene con **`response.sendRedirect`**)

- 2 bis. Si invoca il metodo **`include`** per inserire l'output della pagina di destinazione e continuare l'elaborazione

Inoltrare una richiesta: esempio

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
  String operation = request.getParameter(“destinazione");
  if (operation = = null) {
    operation = "unknown";
  }
  if (operation.equals(“destinazione1")) {
    gotoPage("/operations/paginadidestinazione.jsp",
             request, response);
  } else if (operation.equals(“destinazione2")) {
    gotoPage("/operations/servletdidestinazione",
             request, response);
  } else {
    gotoPage("/operations/servletdierrore",
             request, response);
  }
}
```

Inoltrare una richiesta: esempio (cont.)

```
private void gotoPage(String address,
                      HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
  RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(address);
  dispatcher.forward(request, response);
}
```

Inoltrare una richiesta… (continua)

- Il metodo **`forward(req, resp)`** della classe **`RequestDispatcher`**
  - La locazione deve essere all'interno della applicazione web, non può essere un URL esterno
  - La redirezione non coinvolge il client. Avviene in modo trasparente al client
  - Può essere utilizzata sia per richieste di GET che per richieste di POST
  - E' più efficiente del metodo **`sendRedirect()`** e va preferito a quest'ultimo ove possibile

Attenzione alle richieste di POST…

- Al contrario delle richieste di GET, non possono essere inoltrate a normali pagine HTML.
- Se avete l'esigenza di inoltrare una richiesta di POST ad una pagina HTML statica rinominatela con estensione **`*.jsp`**
  - nomefile.html non può gestire richieste di POST
  - nomefile.jsp restituisce la stessa risposta sia alla richiesta di GET che alla richiesta di POST

Come fornire dati alla pagina/servlet di destinazione

- **Se** :
  - la richiesta può essere inoltrata a più pagine di destinazione
  - richiede l'elaborazione di dati contenuti nell'oggetto **`HttpServletRequest`**
- conviene spostare l'elaborazione dei dati nella servlet da cui la richiesta ha origine e passare alla pagina/servlet di destinazione i dati già elaborati

Come fornire dati alla pagina di destinazione

- i dati preventivamente elaborati dalla servlet di origine possono essere inclusi come attributi dell'oggetto **`HttpServletRequest`**
- **`request.setAttribute("key1", value1);`**
- La pagina di destinazione può prelevare questi dati
- **`Type1 value1 = (Type1)request.getAttribute("key1");`**

Esercizi di Programmazione web

Esercizio 1

Si scriva un’applicazione basata su servlet che tenga traccia delle sessioni utente attraverso oggetti persistenti sul server. Si mostri un esempio di pagina di login e di servlet di autenticazione che permettano la creazione di una sessione di navigazione in caso di autenticazione corretta.

Esercizio 2

Si scrivano due servlet (A e B) che utilizzino un contatore condiviso di accessi all'applicazione. Il contatore deve essere condiviso, e deve essere incrementato ad ogni accesso, indipendentemente da quale sia la servlet a cui si accede. Ciascuna servlet deve produrre come risposta una pagina html che visualizzi il valore del suddetto contatore. Descrivere tutti i file necessari al funzionamento dell'applicazione e descrivere la struttura della context root.

Esercizio 3

Scrivere una servlet (servlet_uno) che legga da un form nome e password di un utente. Una volta autenticato l'utente, l'esercizio richiede di associare alla sessione un oggetto che lo rappresenti (che contenga almeno il nome), nel quale sia presente un parametro che dica se l'utente è amministratore o no. La servlet produca una pagina di risposta di avvenuta o errata autenticazione.

Esercizio 4

Scrivere una servlet che realizzi una bacheca di messaggi.
Quando questa servlet viene interrogata senza parametri nella richiesta deve produrre un form attraverso il quale un utente possa inserire un messaggio testuale e in fondo alla pagina deve visualizzare tutti i messaggi ricevuti precedentemente, se ve ne sono.
Questa stessa pagina costituirà anche la destinazione del form citato sopra e acquisirà dunque nuovi messaggi, producendo anche in questo caso il form di inserimento e visualizzando in fondo alla pagina tutti i messaggi ricevuti, compreso quello della richiesta corrente a cui la pagina sta rispondendo.

Esercizio 5

Scrivere un'applicazione web che realizzi una pagina di sondaggio elettronico per la scelta di uno tra cinque candidati sindaci.
L'applicazione deve prevedere l'impostazione di un cookie di registrazione del voto sulla macchina dei client che hanno già espresso la loro preferenza.
Se la macchina da cui si connette il client non contiene il cookie di registrazione del voto, l'utente è ammesso alla pagina di voto, altrimenti l'utente deve essere automaticamente ridiretto su una pagina di statistiche che visualizza le percentuali di voti acquisiti dai cinque candidati.
Qualora l'utente sia ammesso al voto, dopo l'espressione della propria preferenza, oltre all'impostazione del cookie di registrazione del voto, deve essere effettuata la ridirezione interna del client sulla pagina di statistiche menzionata in precedenza.

Esercizio 6

Scrivere una pagina servlet che legga da un form alcuni dati anagrafici di un utente e li memorizzi in un apposito oggetto con visibilità di sessione.
La servlet dovrà poi aggiungere alla sessione un parametro booleano (confronto) corrispondente al fatto che l'utente sia nato prima o dopo il 2000 (vero se prima del 2000, falso altrimenti) e redirigere l'utente verso una servlet (servlet_destinazione) con una redirezione esterna/interna.
La servlet di destinazione produrrà una pagina personalizzata a seconda del valore del parametro confronto.

Esercizio 7

Si mostri un esempio di pagina di login e di servlet di autenticazione che permettano la creazione di una sessione di navigazione solo in caso di autenticazione corretta, inviando all'utente una serie di tre domande in sequenza, a cui l'utente dovra` rispondere singolarmente in tre richieste http consecutive, che permettano alla servlet di acquisire tre valori nome, eta`, nazionalita`. Al termine dei tre inserimenti la servlet produrra` un messaggio riassuntivo dei dati acquisiti.

Esercizio 8

Si scriva una servlet che, in risposta a un form in cui l'utente possa inserire il proprio nome, fornisca un messaggio di saluto.
La suddetta servlet dovra` creare una sessione per ciascun nuovo utente e visualizzare nel relativo messaggio di saluto un contatore del numero di sessioni istanziate fino al momento dell'interrogazione. Se la servlet viene interrogata piu` volte consecutive dallo stesso utente, il contatore di sessioni NON deve essere aggiornato, ma solo visualizzato insieme al messaggio di benvenuto.

Pagine JSP

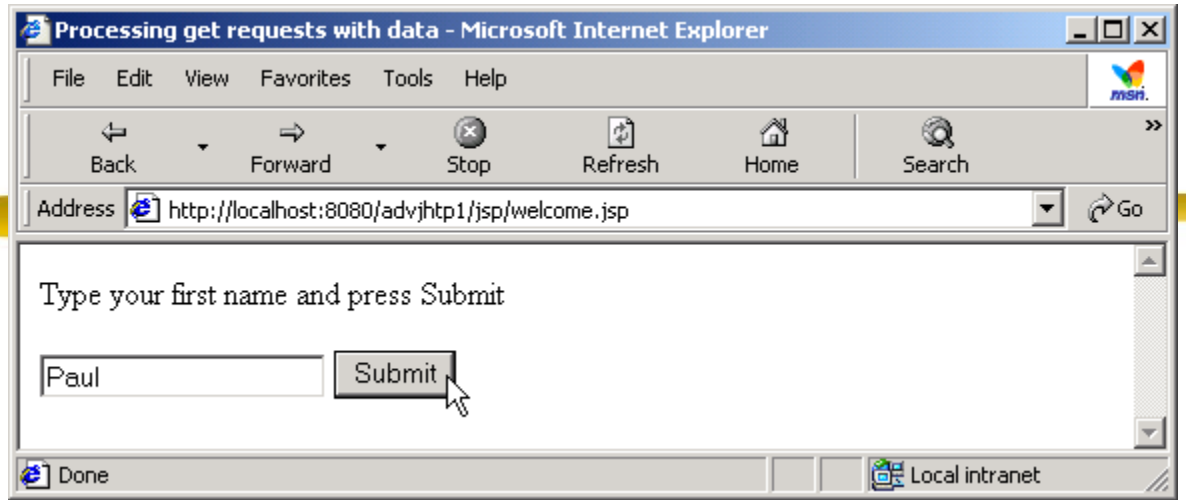
Un esempio, prima di cominciare la trattazione teorica

```
4
5  <!-- welcome.jsp -->
6  <!-- JSP that processes a "get" request containing data. -->
7
8  <html>
9
10    <!-- head section of document -->
11    <head>
12       <title>Processing "get" requests with data</title>
13    </head>
14
15    <!-- body section of document -->
16    <body>
17       <% // begin scriptlet
18
19          String name = request.getParameter( "firstName" );
20
21          if ( name != null ) {
22
23       %> <%-- end scriptlet to insert fixed template data --%>
24
25             <h1>
26                Hello <%= name %>, <br />
27                Welcome to JavaServer Pages!
28             </h1>
29
30       <% // continue scriptlet
31
32          }  // end if
```

Uso di scriptlet per inserire codice java

Uso dell'oggetto implicito **`request`** per ottenere il valore di un parametro

```
33        else {
34
35     %> <%-- end scriptlet to insert fixed template data --%>
36
37           <form action = "welcome.jsp" method = "get">
38              <p>Type your first name and press Submit</p>
39
40              <p><input type = "text" name = "firstName" />
41                 <input type = "submit" value = "Submit" />
42              </p>
43           </form>
44
45     <% // continue scriptlet
46
47        }  // end else
48
49     %> <%-- end scriptlet --%>
50   </body>
51
52 </html>  <!-- end XHTML document -->
```


Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/advjhtp1/jsp/welcome.jsp
Go
Type your first name and press Submit
Paul
Submit
Done
Local intranet


Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/advjhtp1/jsp/welcome.jsp?firstName=Paul
Go
Hello Paul,
Welcome to JavaServer Pages!
Done
Local intranet