Laboratorio di Programmazione di Rete

Lezione del 12 Aprile 2010

Docente: Novella Bartolini
Ricevimento: Mercoledì ore 12:30-14:00
Via Salaria 113, terzo piano, stanza 309
Email: bartolini@di.uniroma1.it

Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

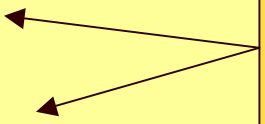
Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

Inoltro delle richieste ad altre risorse

- Servlet di esempio **RedirectServlet**
  - Inoltra la richiesta ad un nuovo URL
  - Fa uso del metodo sendRedirect() dell'interfaccia 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 una servlet di redirezione, cui abbiamo assegnato l'alias "redirect", realizzata attraverso la classe **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.

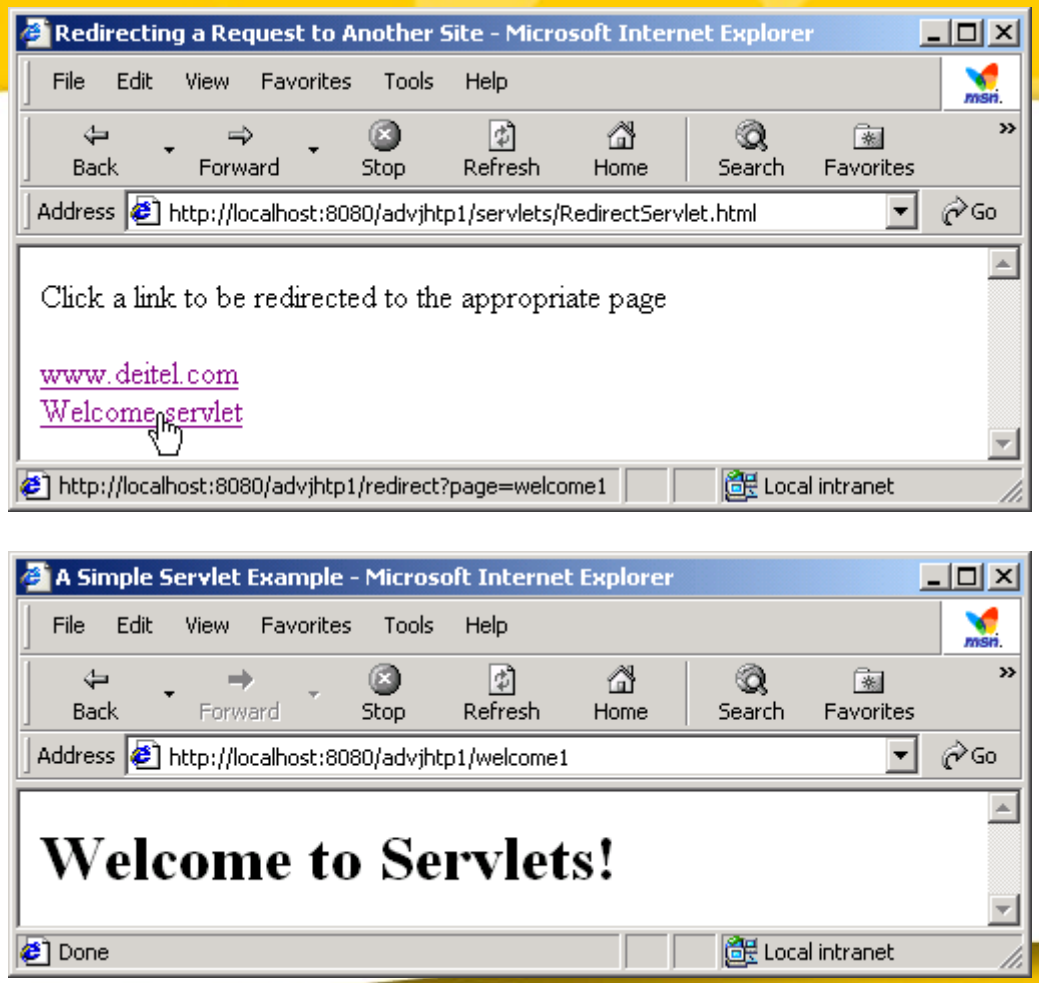
Si inoltra la richiesta alla servlet **WelcomeServlet1**.

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 utilizzato dal web server per costruire la risposta HTTP.
2. La risposta HTTP contiene un header di redirezione verso la nuova locazione
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

Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

Concetto di *sessione di navigazione*

- Una *sessione di navigazione* è una sequenza di richieste HTTP logicamente correlate, provenienti da uno stesso client e dirette verso uno stesso server

Esempi:

- Personalizzazione delle informazioni presenti nelle pagine di un'applicazione web
- Gestione di informazioni private

Gestione della sessione di navigazione

PROBLEMA

HTTP – è un protocollo stateless

Non supporta la persistenza delle informazioni

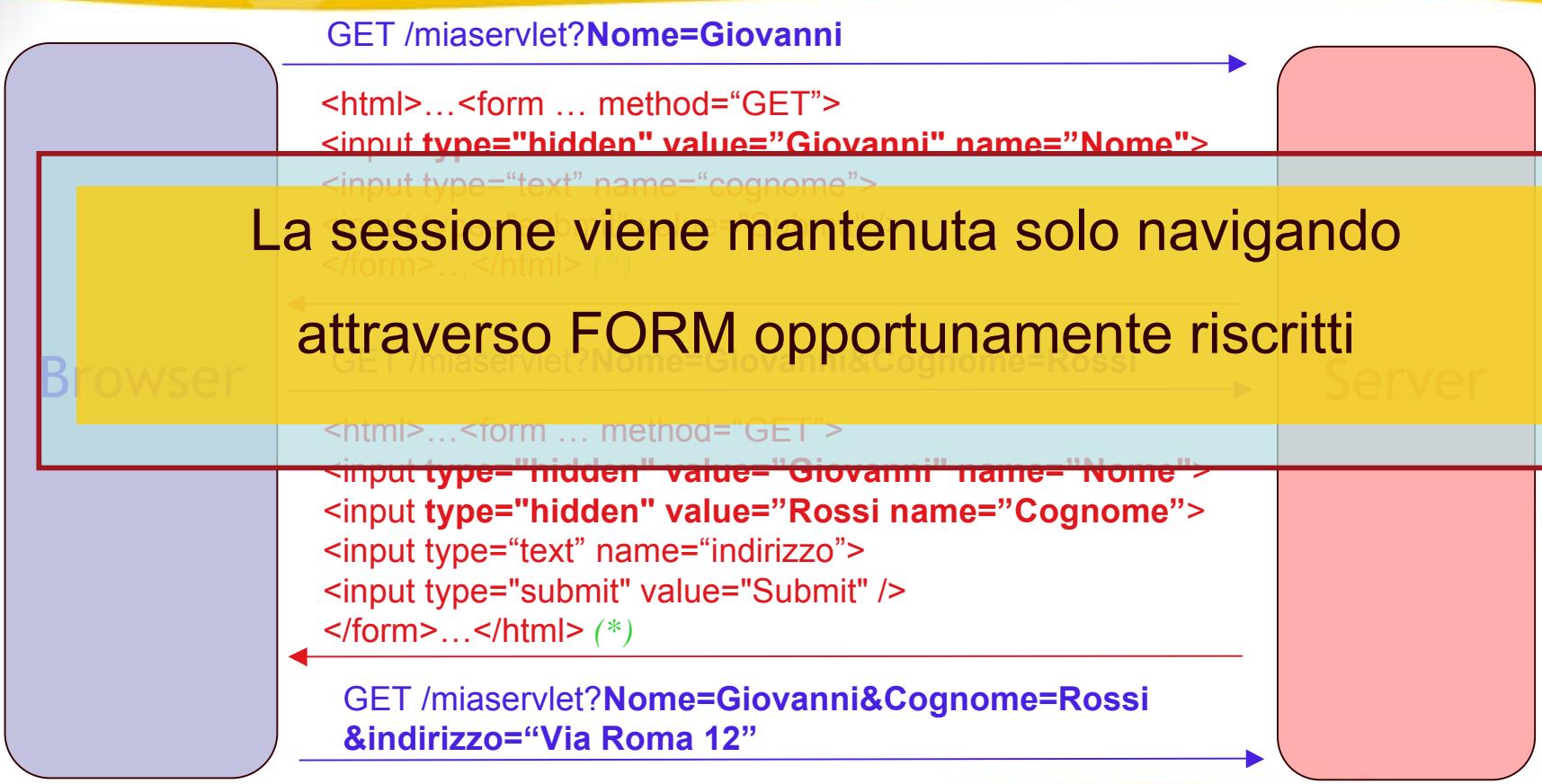
- Tecniche di gestione di informazioni persistenti durante una sessione di navigazione:
  - Hidden type input
  - Cookie
  - Session tracking
  - URL rewriting (parametri get)

Gestione della sessione tramite parametri hidden

Gestione della sessione tramite parametri hidden

(*) *Il testo della pagina html viene generato dinamicamente dalla servlet in funzione dei parametri ricevuti nella richiesta, sia che fossero hidden sia che fossero visibili nel form*

Gestione della sessione tramite parametri hidden



La sessione viene mantenuta solo navigando

attraverso FORM opportunamente riscritti

(*) Il testo della pagina html viene generato dinamicamente dalla servlet in funzione dei parametri ricevuti nella richiesta, sia che fossero hidden sia che fossero visibili nel form

Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

I cookie

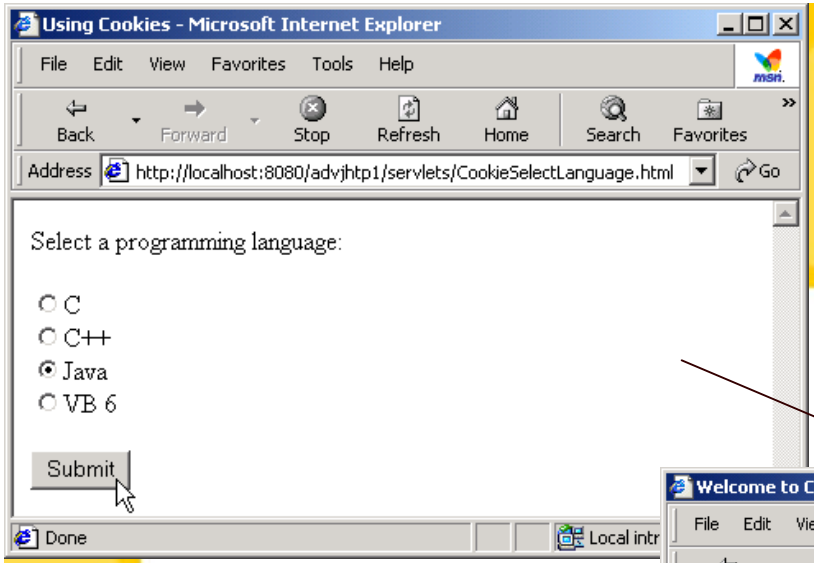
- File contenenti dati testuali nella forma di coppie (nome, valore)
- Vengono spediti dal server e memorizzati sul computer dell'utente (client) per utilizzi successivi
- Contengono informazioni necessarie nelle richieste successive
- Un parametro ne caratterizza l'età massima (vengono cancellati automaticamente quando scadono)

- Esempio: servlet **`CookieServlet`**
  - Gestisce sia richieste di **`get`** che di **`post`**

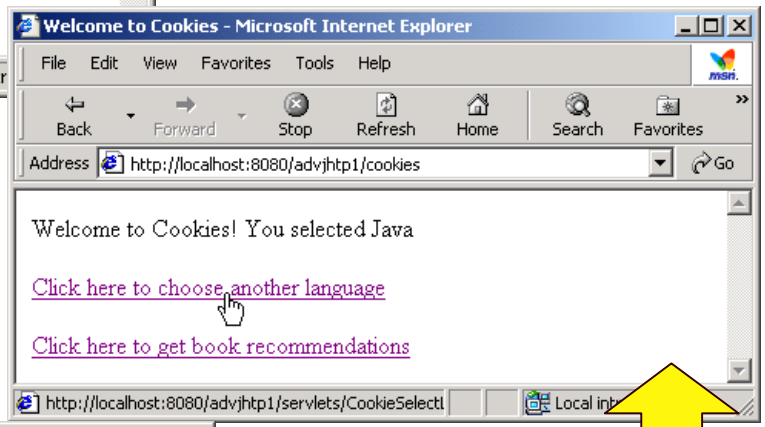
Esempio di servlet che gestisce informazioni di sessione attraverso cookie

- Un'applicazione web consente all'utente di selezionare, attraverso richieste successive, alcuni linguaggi di programmazione di suo interesse.
- Al termine di una serie di selezioni, viene proposto all'utente un elenco di libri aventi per argomento i linguaggi selezionati.

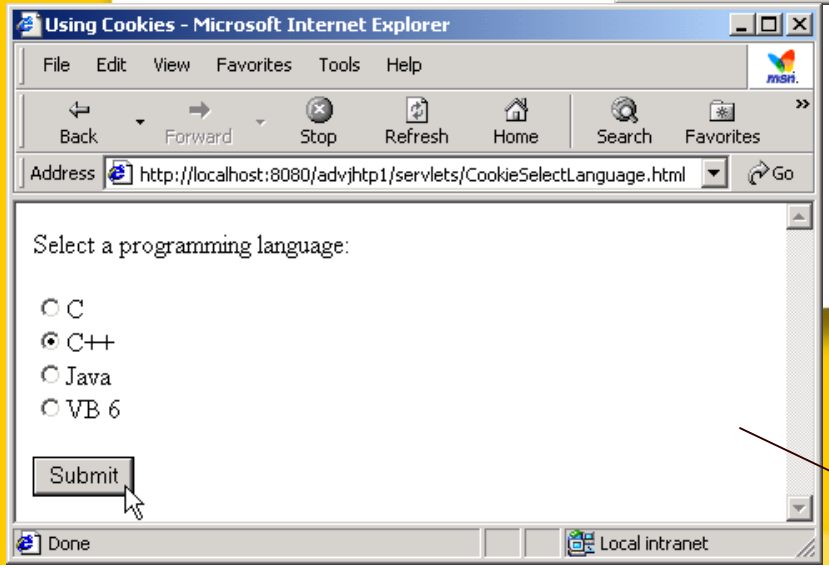
Funzionamento della servlet



POST

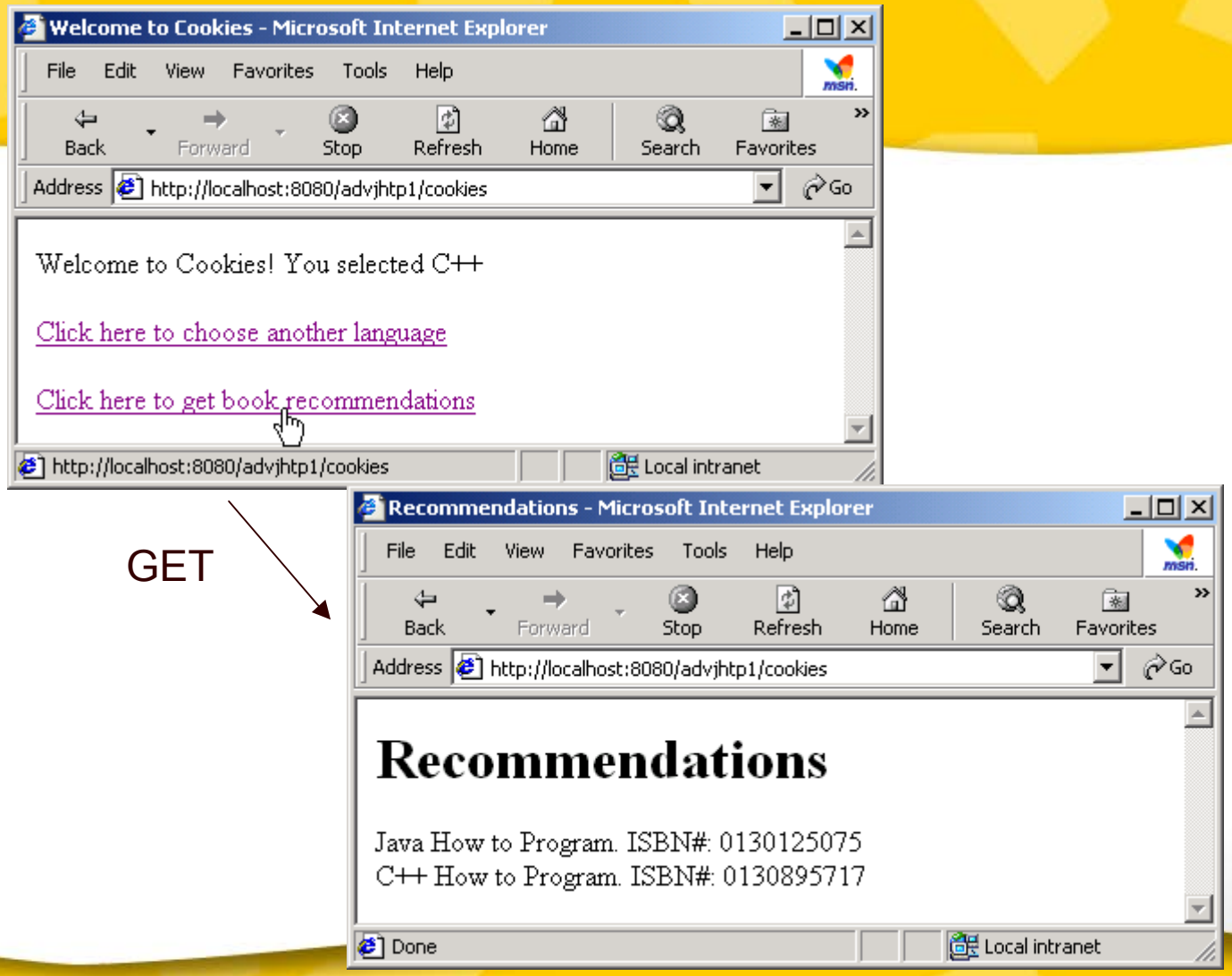


Questa servlet riceve in POST le selezioni dell'utente, e le elenca a seguito di richieste di GET



POST

Cosa vogliamo ottenere?



Inviamo le selezioni con una richiesta di POST e le preleviamo con richieste di GET

Classe `Cookie`: costruttore

- Un cookie viene creato invocando il costruttore.
- Il metodo costruttore di cookie prende due oggetti stringa in ingresso: nome e valore.
  - Né il nome, né il valore dovono contenere spazi o uno dei seguenti caratteri: [ ] ( ) = , " / ? @ : ;

- `Cookie(java.lang.String name, java.lang.String value)`

costruisce un cookie con il nome e il valore specificati

Metodi della classe Cookie (1/2)

Method	Description
`getComment()`	Returns a `String` describing the purpose of the cookie (`null` if no comment has been set with `setComment`).
`getDomain()`	Returns a `String` containing the cookie's domain. This determines which servers can receive the cookie. By default, cookies are sent to the server that originally sent the cookie to the client.
`getMaxAge()`	Returns an `int` representing the maximum age of the cookie in seconds.
`getName()`	Returns a `String` containing the name of the cookie as set by the constructor.
`getPath()`	Returns a `String` containing the URL prefix for the cookie. Cookies can be "targeted" to specific URLs that include directories on the Web server. By default, a cookie is returned to services operating in the same directory as the service that sent the cookie or a subdirectory of that directory.
`getSecure()`	Returns a `boolean` value indicating if the cookie should be transmitted using a secure protocol (`true`).
`getValue()`	Returns a `String` containing the value of the cookie as set with `setValue` or the constructor.
`getVersion()`	Returns an `int` containing the version of the cookie protocol used to create the cookie. A value of 0 (the default) indicates the original cookie protocol as defined by Netscape. A value of 1 indicates the current version, which is based on *Request for Comments (RFC) 2109*.

(Part 1 of 2)

Metodi della classe Cookie (2/2)

`setComment(String)`	The comment describing the purpose of the cookie that is presented by the browser to the user. (Some browsers allow the user to accept cookies on a per-cookie basis.)
`setDomain(String)`	This determines which servers can receive the cookie. By default, cookies are sent to the server that originally sent the cookie to the client. The domain is specified in the form `".deitel.com"`, indicating that all servers ending with `.deitel.com` can receive this cookie.
`setMaxAge(int)`	Sets the maximum age of the cookie in seconds.
`setPath(String)`	Sets the "target" URL prefix indicating the directories on the server that lead to the services that can receive this cookie.
`setSecure(boolean)`	A `true` value indicates that the cookie should only be sent using a secure protocol.
`setValue(String)`	Sets the value of a cookie.
`setVersion(int)`	Sets the cookie protocol for this cookie.

n.b.: non esiste il metodo setName() il nome di un cookie non può essere cambiato dopo la sua creazione

Come trasmettere i cookie dal client al server e viceversa?

- Un cookie viene trasmesso dal client al server come oggetto di input, ovvero viene associato all'oggetto richiesta
  - INPUT:

    metodo getCookies() dell'interfaccia HttpServletRequest
    - Il metodo restituisce un array di oggetti Cookie

- Un cookie viene trasmesso dal server al client come oggetto di output, ovvero viene associato all'oggetto risposta
  - OUTPUT:

    metodo addCookies() dell'interfaccia HttpServletResponse
    - Il metodo prende in ingresso un oggetto Cookie

Cancellazione di cookie

- Non è previsto un metodo esplicito per la cancellazione dei cookie, quindi per poter rimuovere i cookie si deve:
  - Prelevare il cookie
  - Configurare il suo tempo di vita a zero
    (attraverso il metodo setMaxAge(int) della classe Cookie)
  - Inviare di nuovo il cookie al client

- Attenzione: se creo due cookie con lo stesso nome, anche da pagine diverse, il secondo cookie sovrascrive il primo.

```
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  <!-- CookieSelectLanguage.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9    <title>Using Cookies</title>
10 </head>
11
12 <body>
13   <form action = "/DirectoryDiSaluto/cookies" method = "post">
14
15     <p>Select a programming language:</p>
16     <p>
17       <input type = "radio" name = "language"
18         value = "C" />C <br />
19
20       <input type = "radio" name = "language"
21         value = "C++" />C++ <br />
22
23       <!-- this radio button checked by default -->
24       <input type = "radio" name = "language"
25         value = "Java" checked = "checked" />Java<br />
26
27       <input type = "radio" name = "language"
28         value = "VB6" />VB 6
29     </p>
30
31     <p><input type = "submit" value = "Submit" /></p>
32
33   </form>
34 </body>
35 </html>
```

```
1  CookieServlet.java
2  // Using cookies to store data on the client computer.
3  package com.deitel.advjhtp1.servlets;
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8  import java.util.*;
9
10 public class CookieServlet extends HttpServlet {
11    private final Map books = new HashMap();
12
13    // initialize Map books
14    public void init()
15    {
16       books.put( "C", "0130895725" );
17       books.put( "C++", "0130895717" );
18       books.put( "Java", "0130125075" );
19       books.put( "VB6", "0134569555" );
20    }
21
22    // receive language selection and send cookie containing
23    // recommended book to the client
24    protected void doPost( HttpServletRequest request,
25       HttpServletResponse response )
26          throws ServletException, IOException
27    {
28       String language = request.getParameter( "language" );
29       String isbn = books.get( language ).toString();
30       Cookie cookie = new Cookie( language, isbn );
31
32       response.addCookie( cookie );  // must precede getWriter
33       response.setContentType( "text/html" );
34       PrintWriter out = response.getWriter();
35
```

Chiave

Valore

Il metodo **init** popola la collezione books con 4 coppie chiave/valore.

Uso il metodo **getParameter** per ottenere il valore selezionato dall'utente

Ricerco nella collezione books l'elemento corrispondente al linguaggio scelto, per ottenere l'ISBN

Creo un nuovo oggetto **Cookie**, usando le due stringhe di testo **language** e **isbn** come nome e valore rispettivamente

Invio il cookie al client attraverso l'oggetto **response** con il metodo **addCookie**

```
44
45       out.println(
46          "<html>" );
47
48       // head section of document
49       out.println( "<head>" );
50       out.println( "<title>Welcome to Cookies</title>" );
51       out.println( "</head>" );
52
53       // body section of document
54       out.println( "<body>" );
55       out.println( "<p>Welcome to Cookies! You selected " +
56          language + "</p>" );
57
58       out.println( "<p><a href = " +
59          "\"/DirectoryDiSaluto/CookieSelectLanguage.html\">" +
60          "Click here to choose another language</a></p>" );
61
62       out.println( "<p><a href = \"/DirectoryDiSaluto/cookies\">" +
63          "Click here to get book recommendations</a></p>" );
64       out.println( "</body>" );
65
66       // end XHTML document
67       out.println( "</html>" );
68       out.close();    // close stream
69    }
70
```

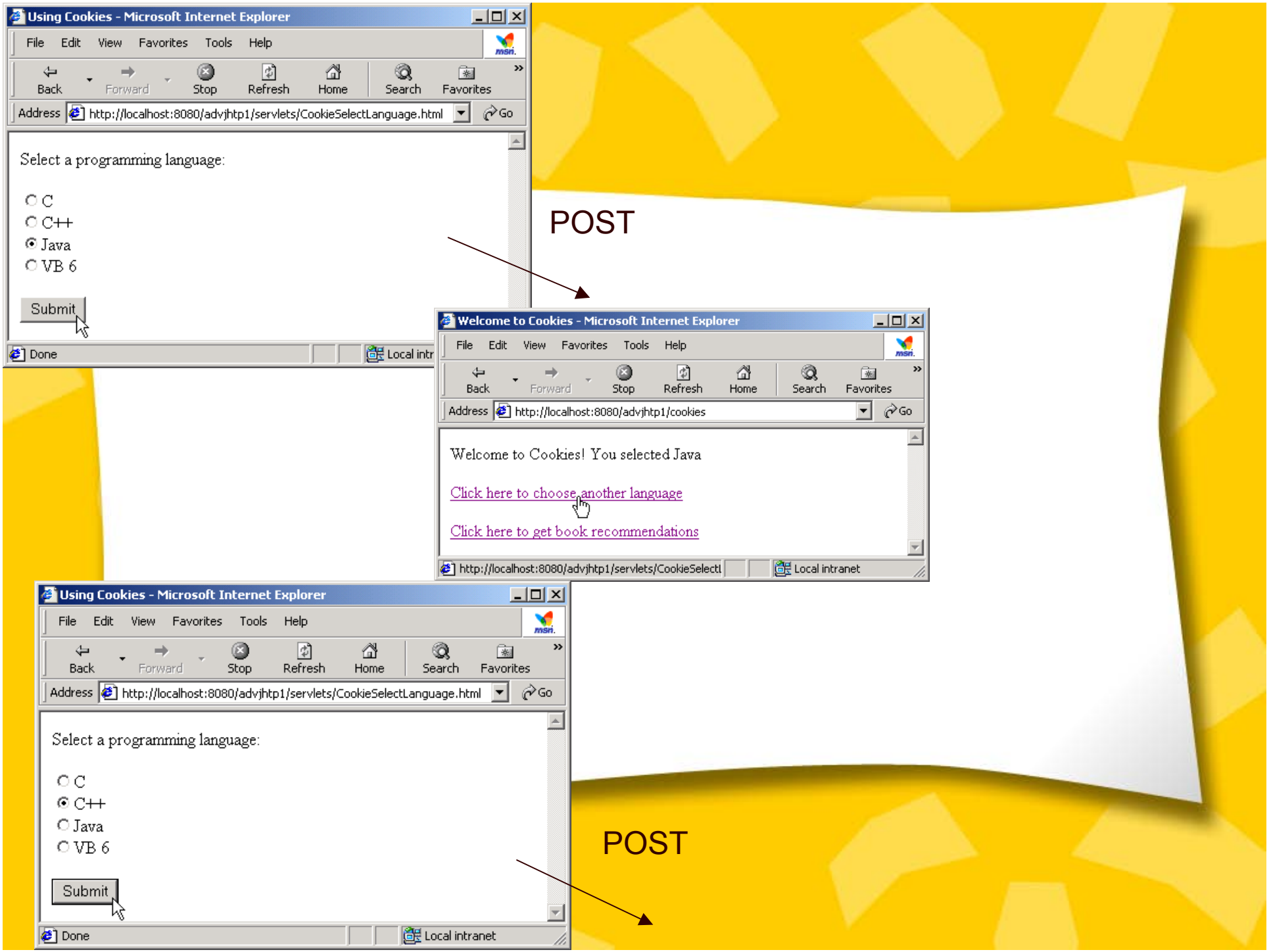
```
71  // read cookies from client and create XHTML document
72  // containing recommended books
73  protected void doGet( HttpServletRequest request,
74     HttpServletResponse response )
75        throws ServletException, IOException
76  {
77     Cookie cookies[] = request.getCookies();  // get cookies
78
79     response.setContentType( "text/html" );
80     PrintWriter out = response.getWriter();
81
82     // start XHTML document
83     out.println( "<?xml version = \"1.0\"?>" );
84
85     out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
86        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
87        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
88
89     out.println(
90        "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
91
92     // head section of document
93     out.println( "<head>" );
94     out.println( "<title>Recommendations</title>" );
95     out.println( "</head>" );
96
97     // body section of document
98     out.println( "<body>" );
99
100    // if there are any cookies, recommend a book for each ISBN
101    if ( cookies != null && cookies.length != 0 ) {
102       out.println( "<h1>Recommendations</h1>" );
103       out.println( "<p>" );
104
```

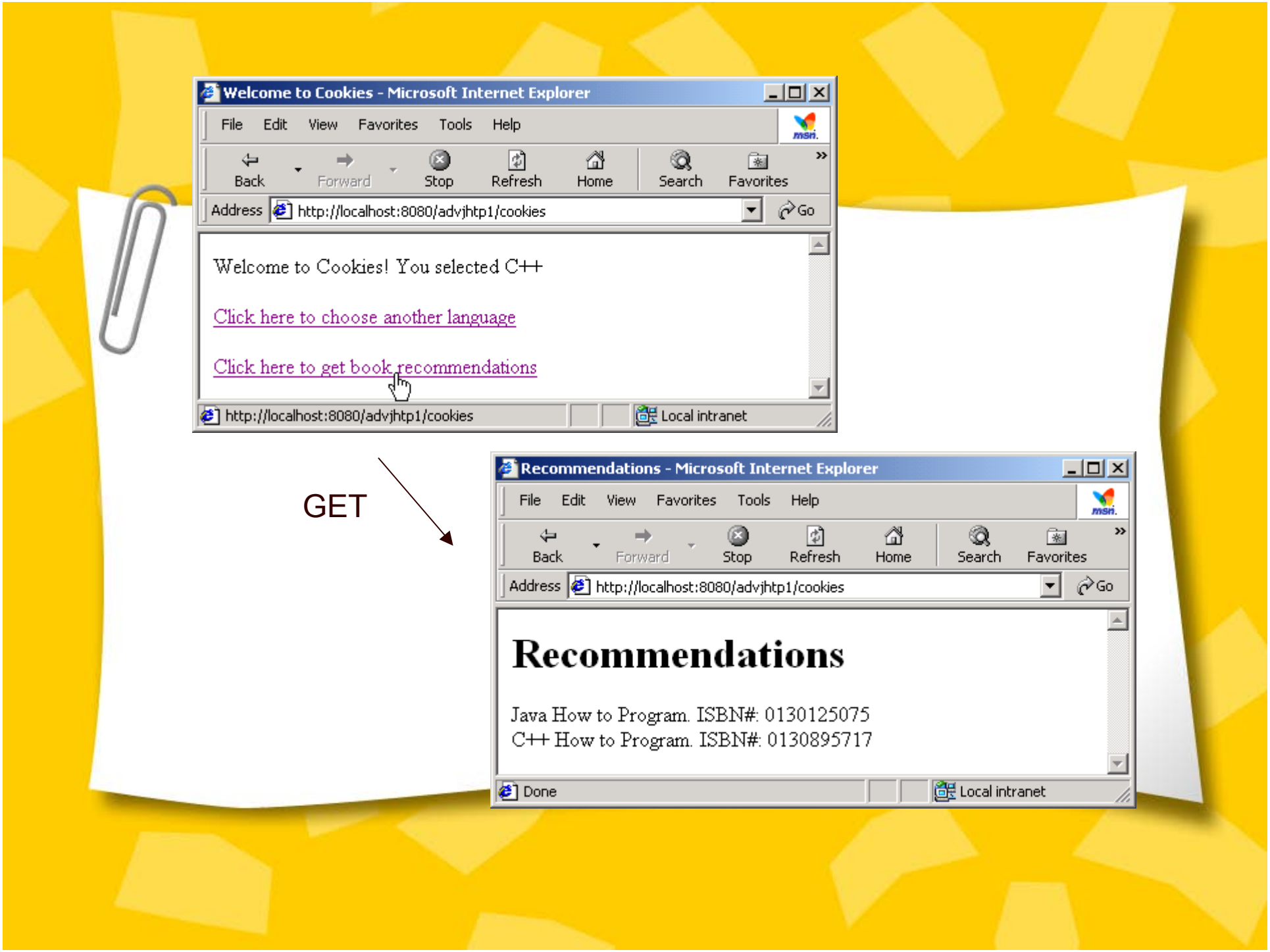
Legge i cookie presenti sul client utilizzando il metodo **getCookies** dell'oggetto **Request**, che restituisce un array di oggetti **Cookie**

Nota:

Aggiungiamo i cookie all'oggetto risposta e li preleviamo dall'oggetto richiesta

```
105         // get the name of each cookie
106         for ( int i = 0; i < cookies.length; i++ )
107            out.println( cookies[ i ].getName() +
108               " How to Program. ISBN#: " +
109               cookies[ i ].getValue() + "<br />" );
110
111         out.println( "</p>" );
112      }
113      else {   // there were no cookies
114         out.println( "<h1>No Recommendations</h1>" );
115         out.println( "<p>You did not select a language.</p>" );
116      }
117
118      out.println( "</body>" );
119
120      // end XHTML document
121      out.println( "</html>" );
122      out.close();    // close stream
123   }
124 }
```

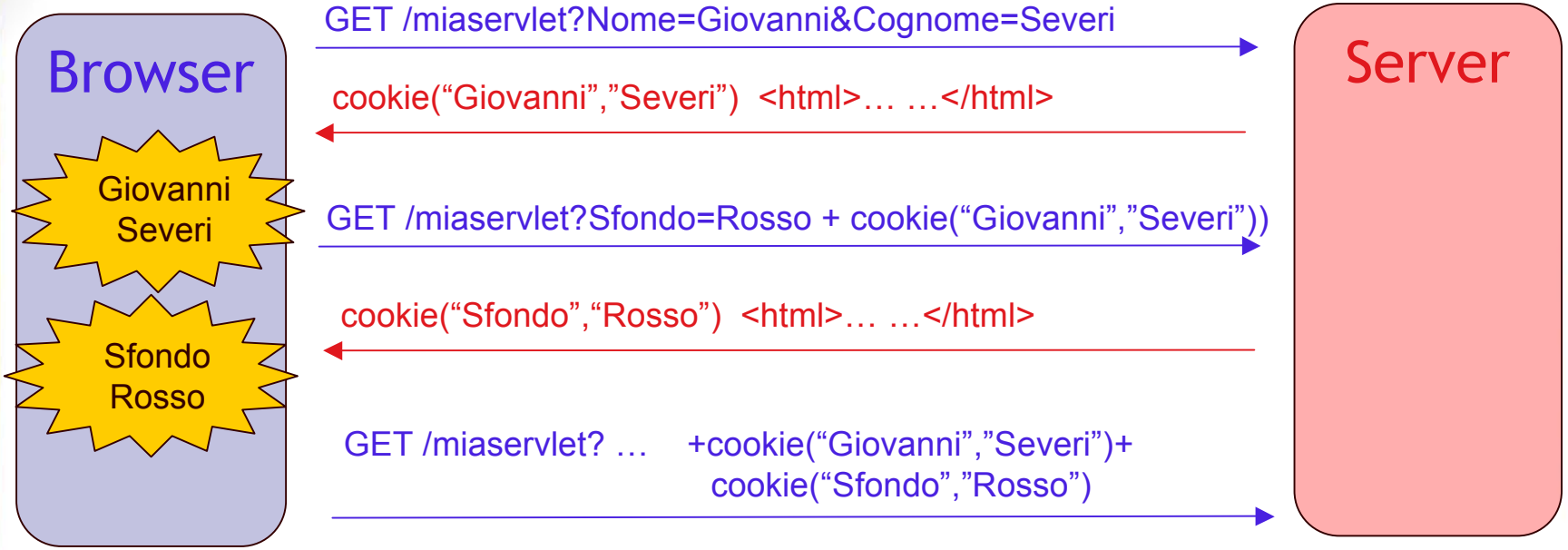

Using Cookies - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/servlets/CookieSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intr
POST
Welcome to Cookies - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/cookies
Go
Welcome to Cookies! You selected Java
Click here to choose another language
Click here to get book recommendations
http://localhost:8080/advjhtp1/servlets/CookieSelectl
Local intranet
Using Cookies - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/servlets/CookieSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intranet
POST


Welcome to Cookies - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/cookies
Go
Welcome to Cookies! You selected C++
Click here to choose another language
Click here to get book recommendations
http://localhost:8080/advjhtp1/cookies
Local intranet
GET
Recommendations - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/advjhtp1/cookies
Go
Recommendations
Java How to Program. ISBN#: 0130125075
C++ How to Program. ISBN#: 0130895717
Done
Local intranet

web.xml (CookieServlet)

Descriptor element	Value
servlet *element*	
servlet-name	**cookies** (ma va bene anche **"Paperino"**)
description	Using cookies to maintain state information.
servlet-class	**DirectoryDiSaluto.CookieServlet**
servlet-mapping *element*	
servlet-name	**cookies** (ma va bene anche **"Paperino"**)
url-pattern	**/cookies**

Gestione della sessione tramite cookie



Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

Gestire la sessione con oggetti persistenti sul server (`HttpSession`)

- Interfaccia **`HttpSession`**
- Trovate informazioni all'indirizzo:

  http://127.0.0.1:8080/tomcat-docs/servletapi/index.html

- Un oggetto che implementi l'interfaccia HttpSession identifica un utente/browser attraverso diverse richieste e permette di memorizzare informazioni acquisite durante la sessione di navigazione.
- Tale oggetto rappresenta la sessione:
  - Persiste per uno specifico periodo di tempo.
  - Corrisponde ad un solo utente, che può visitare il sito anche più volte.

Perché usare HttpSession

- Non potremmo memorizzare le informazioni che ci servono direttamente attraverso gli attributi della servlet?

- Più client possono accedere alla stessa servlet in parallelo!

Metodi da conoscere per usare le sessioni (1/3)

- Metodi dell'oggetto HttpServletRequest
  - HttpSession **getSession** (boolean create)
    Restituisce l'oggetto HttpSession associato con la richiesta. Se non esiste ancora nessun oggetto HttpSession, ne viene creato uno nel caso create valga true.
- Metodi dell'oggetto HttpSession
  - public void **setAttribute** (java.lang.String name, **java.lang.Object value**)
    - Associa un oggetto alla presente sessione, utilizzando il nome specificato. Se esiste un collegamento con un oggetto con lo stesso nome, questo viene rimpiazzato.
  - java.lang.Object **getAttribute**(java.lang.String name)
    - Restituisce l'oggetto associato con il nome dato in input, oppure null se non esiste nessun oggetto con quel nome

Metodi da conoscere per usare le sessioni (2/3)

- Metodi dell'oggetto HttpSession
  - java.lang.String **getId**()
    Restituisce una stringa contenente l'identificativo unico della sessione
  - boolean **isNew**()
    Restituisce true se la sessione con il client è stata creata in seguito alla richiesta corrente
  - long **getCreationTime**()
    Restituisce l'istante di creazione della sessione misurato in millisecondi a partire dalle 00:00 del 1 Gennaio 1970, GMT
  - long **getLastAccessedTime**()
    Restituisce l'istante dell'ultima richiesta associata alla sessione misurato in millisecondi a partire dalle 00:00 del 1 Gennaio 1970, GMT

Metodi da conoscere per usare le sessioni (3/3)

- … metodi dell'oggetto HttpSession
  - int **getMaxInactiveInterval**()
    Restituisce il massimo intervallo di tempo, in secondi, in cui il container può mantenere la sessione aperta, tra due accessi consecutivi da parte del client.
  - java.util.Enumeration **getAttributeNames**()
    Restituisce un oggetto Enumeration (un elenco) di stringhe contenenti i nomi di tutti gli oggetti associati alla sessione

Esercizio: uso di HttpSession per memorizzare le scelte dell'utente

- Memorizziamo in un oggetto che implementa l'interfaccia **`HttpSession`** le informazioni che nell'esercizio precedente mettevamo nei cookie.
  - Possiamo memorizzare coppie nome – valore

    dove il valore può essere un oggetto (non solo una stringa come nei cookie)
- Servlet **`SessionServlet`**
  - Usa l'oggetto **`HttpSession`**
  - Gestisce sia richieste di **`GET`** che richieste di **`POST`**

E’ utile sapere per svolgere questo esercizio… (cont.)

- Interface Enumeration
  - Consente l’elencazione di una serie di elementi, uno alla volta
  - Due metodi:
    - java.lang.Object nextElement()
      restituisce il prossimo elemento dell’elenco
    - boolean hasMoreElements()
      restituisce true se l’elenco contiene ancora elementi

```
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  <!-- SessionSelectLanguage.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9    <title>Using Sessions</title>
10 </head>
11
12 <body>
13   <form action = "/DDS/sessions" method = "post">
14
15     <p>Select a programming language:</p>
16     <p>
17       <input type = "radio" name = "language"
18         value = "C" />C <br />
19
20       <input type = "radio" name = "language"
21         value = "C++" />C++ <br />
22
23       <!-- this radio button checked by default -->
24       <input type = "radio" name = "language"
25         value = "Java" checked = "checked" />Java<br />
26
27       <input type = "radio" name = "language"
28         value = "VB6" />VB 6
29     </p>
30
31     <p><input type = "submit" value = "Submit" /></p>
32
33   </form>
34 </body>
35 </html>
```

Stesso form dell'esercizio sui cookie

```
1  // SessionServlet.java
2  // Using HttpSession to maintain client state information.
3
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8  import java.util.*;
9
10 public class SessionServlet extends HttpServlet {
11    private final Map books = new HashMap();
12
13    // initialize Map books
14    public void init()
15    {
16       books.put( "C", "0130895725" );
17       books.put( "C++", "0130895717" );
18       books.put( "Java", "0130125075" );
19       books.put( "VB6", "0134569555" );
20    }
21
22    // receive language selection and create HttpSession object
23    // containing recommended book for the client
24    protected void doPost( HttpServletRequest request,
25       HttpServletResponse response )
26          throws ServletException, IOException
27    {
28       String language = request.getParameter( "language" );
29
30       // Get the user's session object.
31       // Create a session (true) if one does not exist.
32       HttpSession session = request.getSession( true );
33
34       // add a value for user's choice to session
35       session.setAttribute( language, books.get( language ) );
```

Chiave

Valore

Usa il metodo **getSession** dell'interfaccia **HttpServletRequest** per ottenere l'oggetto **HttpSession** o crearlo (true)

Usa **setAttribute** per inserire il nome del linguaggio **language** e il corrispondente numero ISBN nell'oggetto **HttpSession** object.

```
36
37      response.setContentType( "text/html" );
38      PrintWriter out = response.getWriter();
39
40      // send XHTML page to client
41
42      // start XHTML document
43      out.println( "<?xml version = \"1.0\"?>" );
44
45      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
46        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
47        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
48
49      out.println(
50        "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
51
52      // head section of document
53      out.println( "<head>" );
54      out.println( "<title>Welcome to Sessions</title>" );
55      out.println( "</head>" );
56
57      // body section of document
58      out.println( "<body>" );
59      out.println( "<p>Welcome to Sessions! You selected " +
60        language + ".</p>" );
61
62      // display information about the session
63      out.println( "<p>Your unique session ID is: " +
64        session.getId() + "<br />" );
65
66      out.println(
67        "This " + ( session.isNew() ? "is" : "is not" ) +
68        " a new session<br />" );
69
```

Usa il metodo **getID** di **HttpSession** per conoscere l'ID di sessione.

Decide se si tratta di una nuova sessione usando il metodo **isNew**.

```
70      out.println( "The session was created at: " +
71        new Date( session.getCreationTime() ) + "<br />" );
72
73      out.println( "You last accessed the session at: " +
74        new Date( session.getLastAccessedTime() ) + "<br />" );
75
76      out.println( "The maximum inactive interval is: " +
77        session.getMaxInactiveInterval() + " seconds</p>" );
78
79      out.println( "<p><a href = " +
80        "\"servlets/SessionSelectLanguage.html\">" +
81        "Click here to choose another language</a></p>" );
82
83      out.println( "<p><a href = \"sessions\">" +
84        "Click here to get book recommendations</a></p>" );
85      out.println( "</body>" );
86
87      // end XHTML document
88      out.println( "</html>" );
89      out.close();    // close stream
90   }
91
92   // read session attributes and create XHTML document
93   // containing recommended books
94   protected void doGet( HttpServletRequest request,
95      HttpServletResponse response )
96         throws ServletException, IOException
97   {
98      // Get the user's session object.
99      // Do not create a session (false) if one does not exist.
100     HttpSession session = request.getSession( false );
101
102     // get names of session object's values
103     Enumeration valueNames;
104
```

Usa il metodo **getMaxInactiveInterval** per conoscere il massimo intervallo di tempo in cui la sessione può restare inattiva prima che il container la scarti.

ATTENZIONE all'uso dei percorsi relativi!!!

Il percorso di partenza è quello definito nell'url pattern e usato per invocare la servlet.

Provare a cambiare l'url pattern, dovrete ridefinire anche questi link

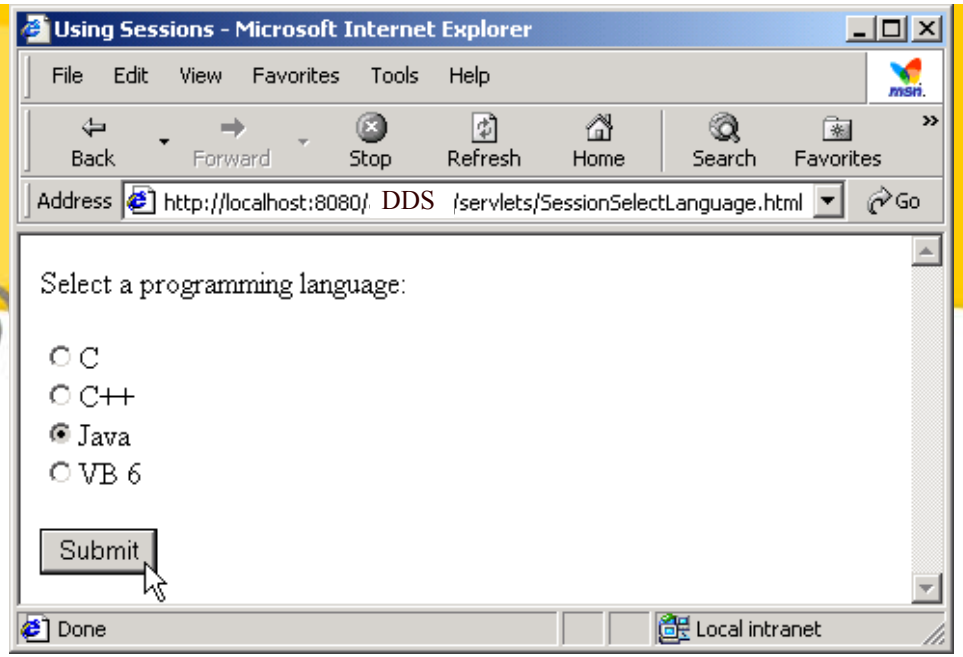
Ottiene l'oggetto **HttpSession** correlato alla sessione corrente, attraverso il metodo **getSession**.

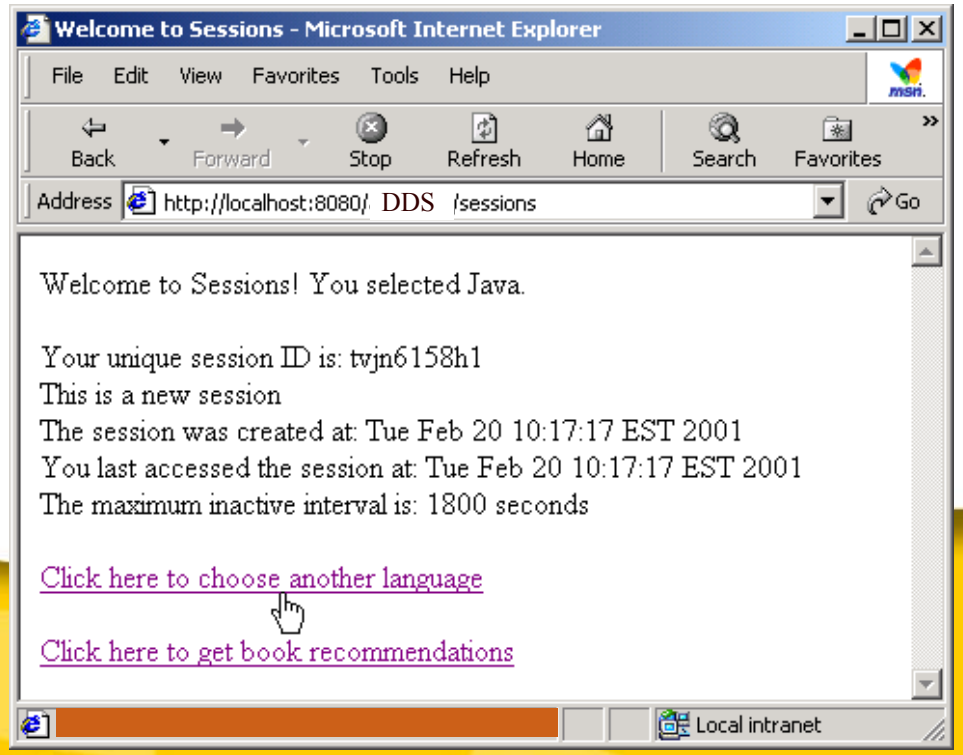
```
105    if ( session != null )
106       valueNames = session.getAttributeNames();
107    else
108       valueNames = null;
109
110    PrintWriter out = response.getWriter();
111    response.setContentType( "text/html" );
112
113    // start XHTML document
114    out.println( "<?xml version = \"1.0\"?>" );
115
116    out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
117       "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
118       "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
119
120    out.println(
121       "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
122
123    // head section of document
124    out.println( "<head>" );
125    out.println( "<title>Recommendations</title>" );
126    out.println( "</head>" );
127
128    // body section of document
129    out.println( "<body>" );
130
131    if ( valueNames != null &&
132         valueNames.hasMoreElements() ) {
133       out.println( "<h1>Recommendations</h1>" );
134       out.println( "<p>" );
135
136       String name, value;
137
```

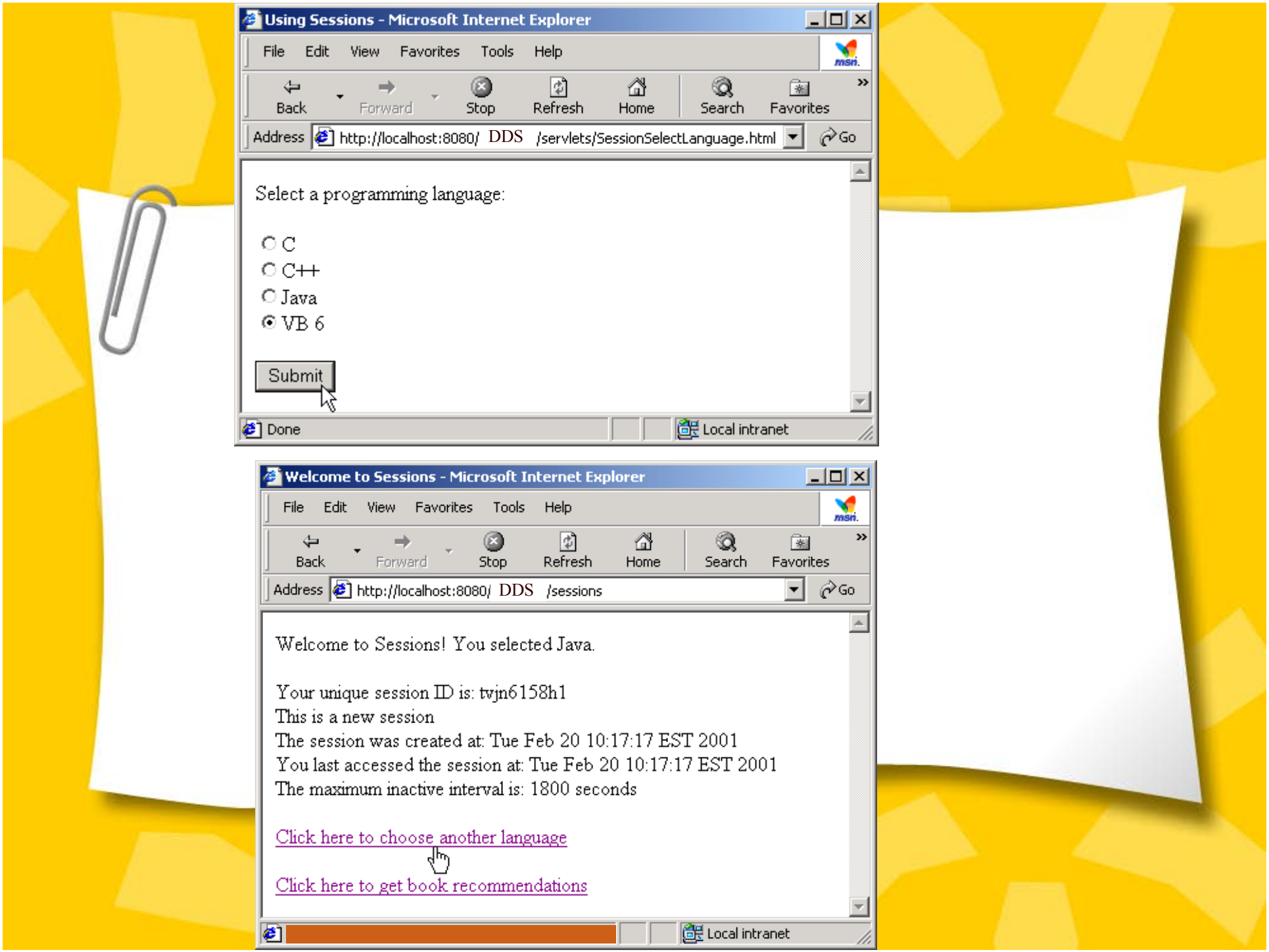
Uses **HttpSession** method **getAttributeNames** to retrieve an **Enumeration** of the attribute names.

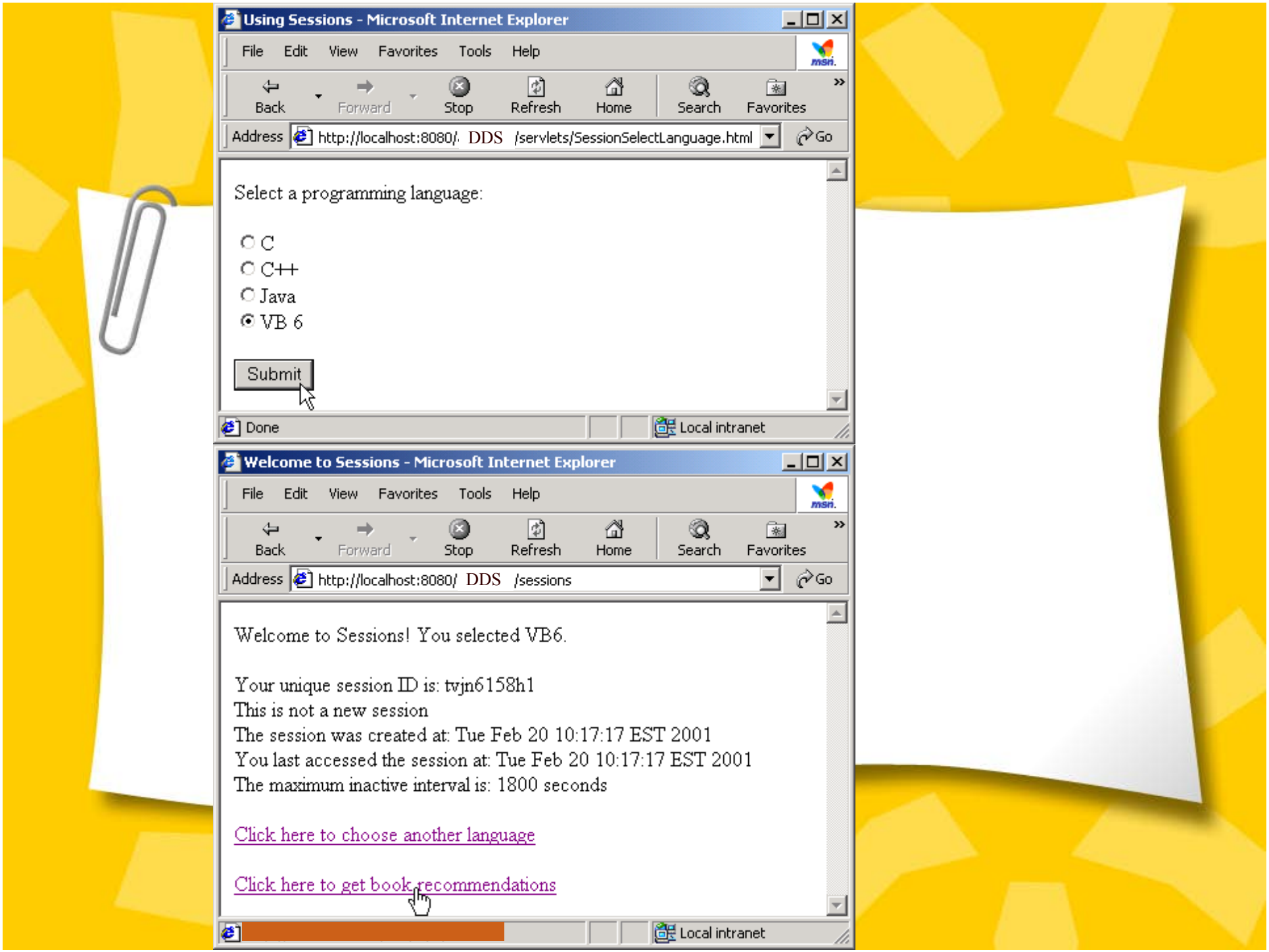
```
138         // get value for each name in valueNames
139         while ( valueNames.hasMoreElements() ) {
140            name = valueNames.nextElement().toString();
141            value = session.getAttribute( name ).toString();
142
143            out.println( name + " How to Program. " +
144               "ISBN#: " + value + "<br />" );
145         }
146
147         out.println( "</p>" );
148      }
149      else {
150         out.println( "<h1>No Recommendations</h1>" );
151         out.println( "<p>You did not select a language.</p>" );
152      }
153
154      out.println( "</body>" );
155
156      // end XHTML document
157      out.println( "</html>" );
158      out.close();    // close stream
159   }
160 }
```

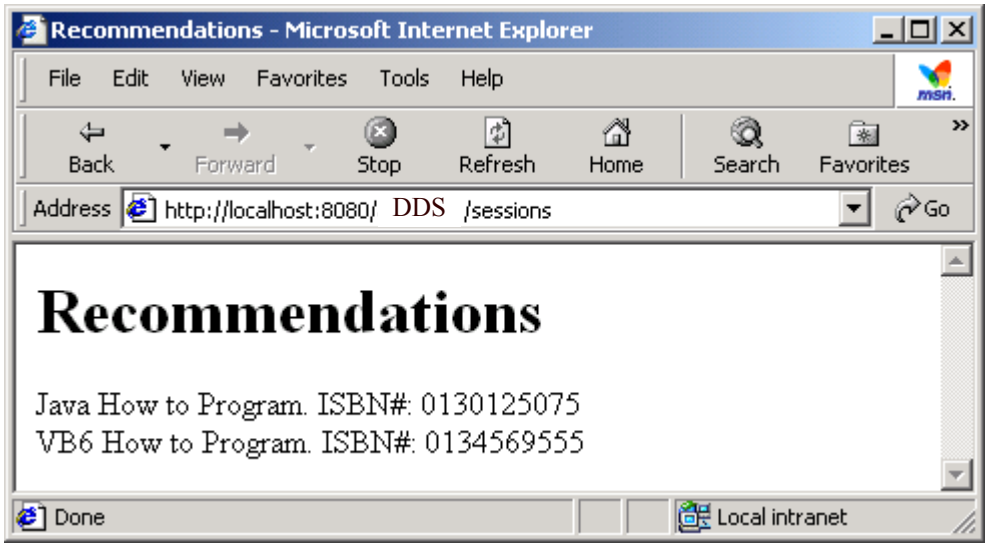
Invokes method **getAttribute** of **HttpSession** to retrieve the ISBN of a book from the **HttpSession** object.


Using Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /servlets/SessionSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intranet


Welcome to Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /sessions
Go
Welcome to Sessions! You selected Java.
Your unique session ID is: tvjn6158h1
This is a new session
The session was created at: Tue Feb 20 10:17:17 EST 2001
You last accessed the session at: Tue Feb 20 10:17:17 EST 2001
The maximum inactive interval is: 1800 seconds
Click here to choose another language
Click here to get book recommendations
Local intranet


Using Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /servlets/SessionSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intranet
Welcome to Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /sessions
Go
Welcome to Sessions! You selected Java.
Your unique session ID is: tvjn6158h1
This is a new session
The session was created at: Tue Feb 20 10:17:17 EST 2001
You last accessed the session at: Tue Feb 20 10:17:17 EST 2001
The maximum inactive interval is: 1800 seconds
Click here to choose another language
Click here to get book recommendations
Local intranet


Using Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /servlets/SessionSelectLanguage.html
Go
Select a programming language:
C
C++
Java
VB 6
Submit
Done
Local intranet
Welcome to Sessions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /sessions
Go
Welcome to Sessions! You selected VB6.
Your unique session ID is: twjn6158h1
This is not a new session
The session was created at: Tue Feb 20 10:17:17 EST 2001
You last accessed the session at: Tue Feb 20 10:17:17 EST 2001
The maximum inactive interval is: 1800 seconds
Click here to choose another language
Click here to get book recommendations
Local intranet


Recommendations - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address http://localhost:8080/ DDS /sessions
Go
Recommendations
Java How to Program. ISBN#: 0130125075
VB6 How to Program. ISBN#: 0134569555
Done
Local intranet

Session Tracking with `HttpSession` (Cont.)

Descriptor element	Value
servlet *element*	
servlet-name	**sessions**
description	Using sessions to maintain state information.
servlet-class	**SessionServlet**
servlet-mapping *element*	
servlet-name	**sessions**
url-pattern	**/sessions**

Chiusura della sessione di navigazione

- Mentre per i cookie non è previsto un metodo esplicito per la cancellazione dei cookie,
  - Prelevare il cookie
  - Configurare il suo tempo di vita a zero (metodo setMaxAge(int) della classe Cookie)
  - Inviare di nuovo il cookie al client

  la cancellazione della sessione deve fare uso del metodo invalidate() dell'interfaccia HttpSession

Riflessioni…

- Q: Come può il server riconoscere le richieste di una stessa sessione e associarle correttamente all'oggetto persistente con il contesto di visibilità voluto?

Riflessioni…

- Q: Come può il server riconoscere le richieste di una stessa sessione e associarle correttamente all'oggetto persistente con il contesto di visibilità voluto?
- R: Viene usato un cookie!!!

Un COOKIEEEEE ????

- Q1: che senso ha usare la sessione se poi questa dipende dal cookie?
- R1: i dati sensibili restano sul server, il cookie creato non è trasferibile da un computer ad un altro in quanto contiene un identificativo generato dinamicamente.

Riflessioni…

- Q2: se il browser ha i cookie disabilitati come faccio a gestire la sessione sul server?
- R2: Si deve fare in modo che tutti i percorsi utilizzati dal browser appendano all'url l'identificativo di sessione (sarà sempre il server a scrivere dinamicamente questi url con una tecnica detta di URL rewriting). Questo si ottiene con il metodo dell'interfaccia HttpServletResponse:

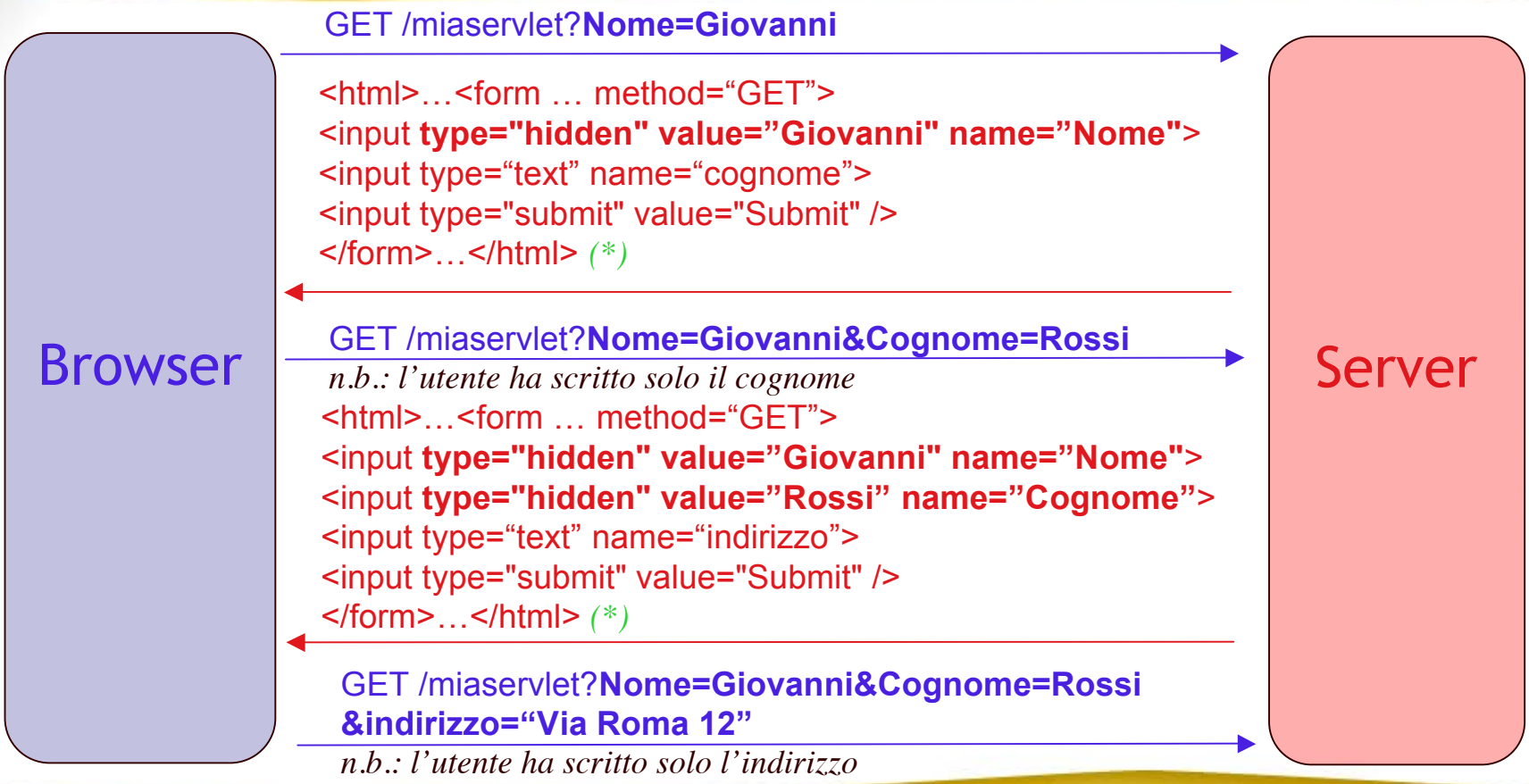
java.lang.String **encodeURL**(java.lang.String url)

Dalla documentazione:

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

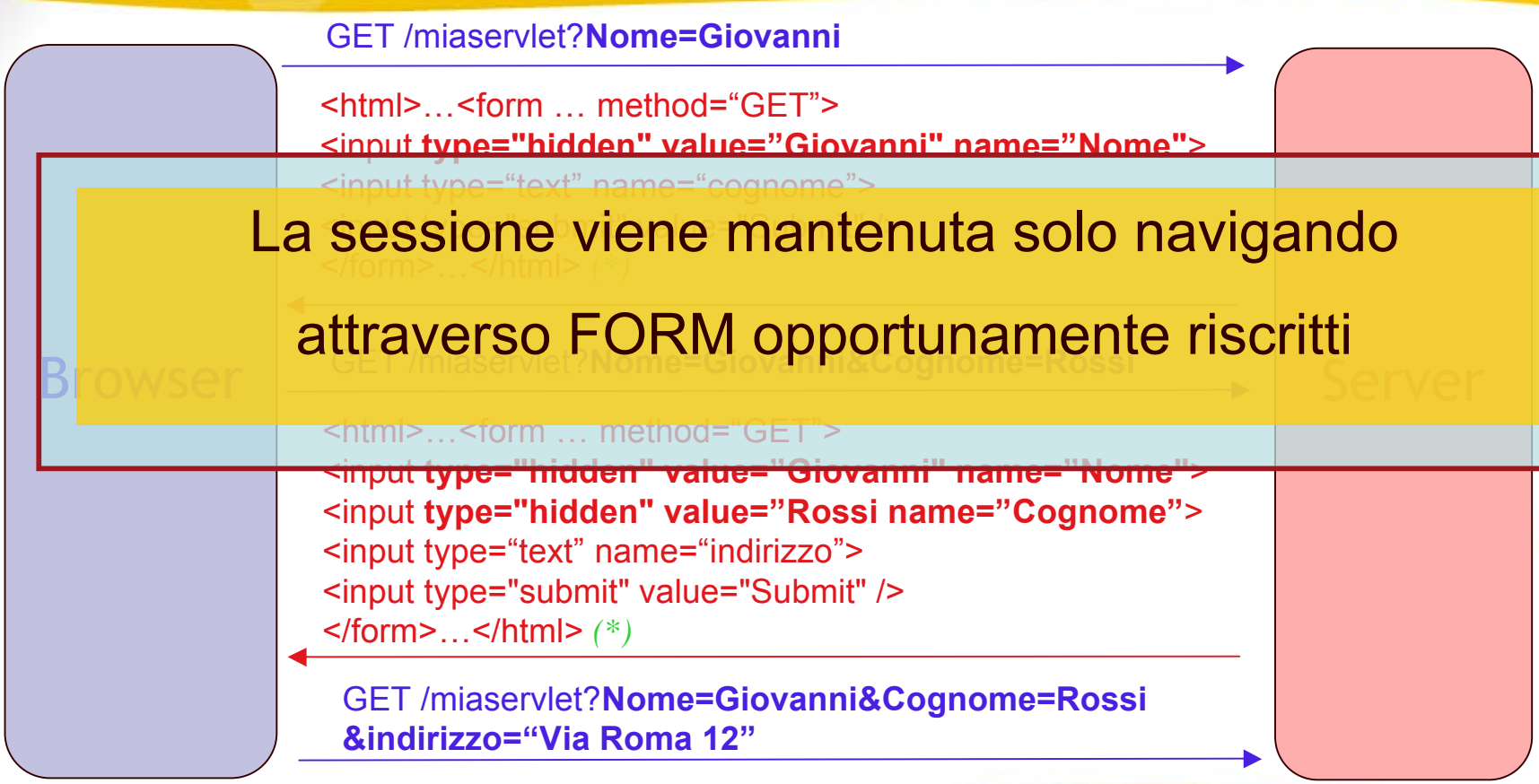
Riassunto: gestione della sessione tramite parametri hidden

Riassunto: gestione della sessione tramite parametri hidden



(*) *Il testo della pagina html viene generato dinamicamente dalla servlet in funzione dei parametri ricevuti nella richiesta, sia che fossero hidden sia che fossero visibili nel form*

Riassunto: gestione della sessione tramite parametri hidden



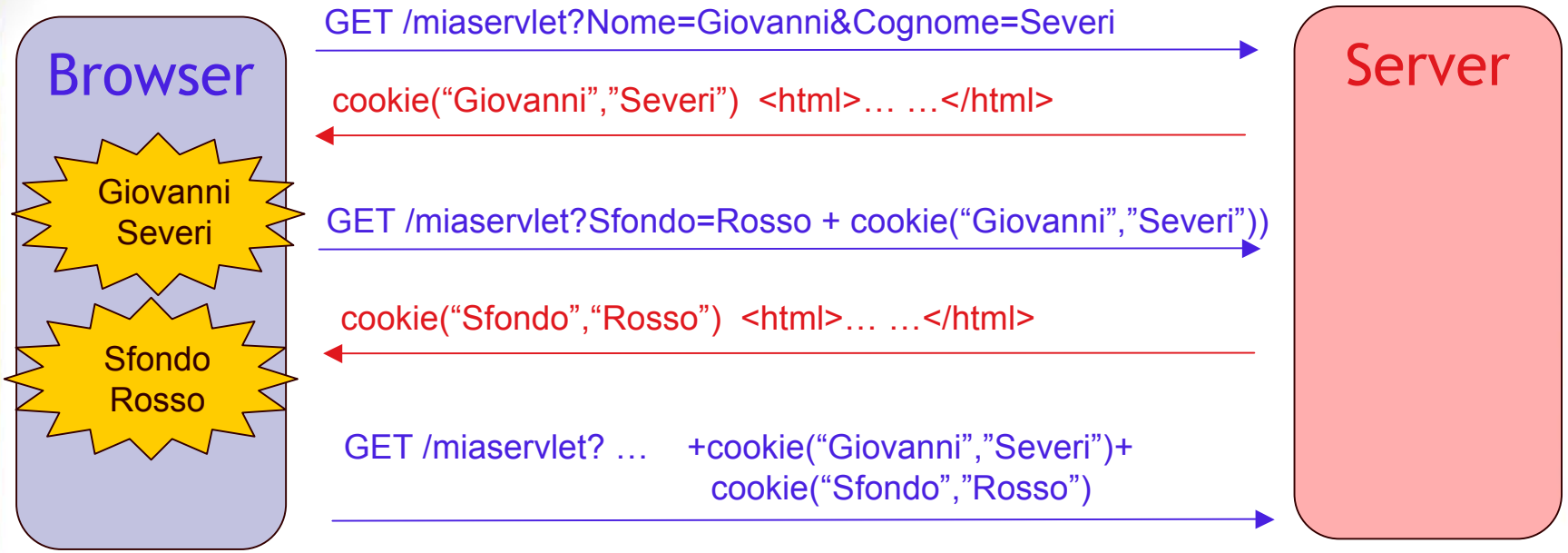
La sessione viene mantenuta solo navigando

attraverso FORM opportunamente riscritti

(*) Il testo della pagina html viene generato dinamicamente dalla servlet in funzione dei parametri ricevuti nella richiesta, sia che fossero hidden sia che fossero visibili nel form

Riassunto: gestione della sessione tramite cookie

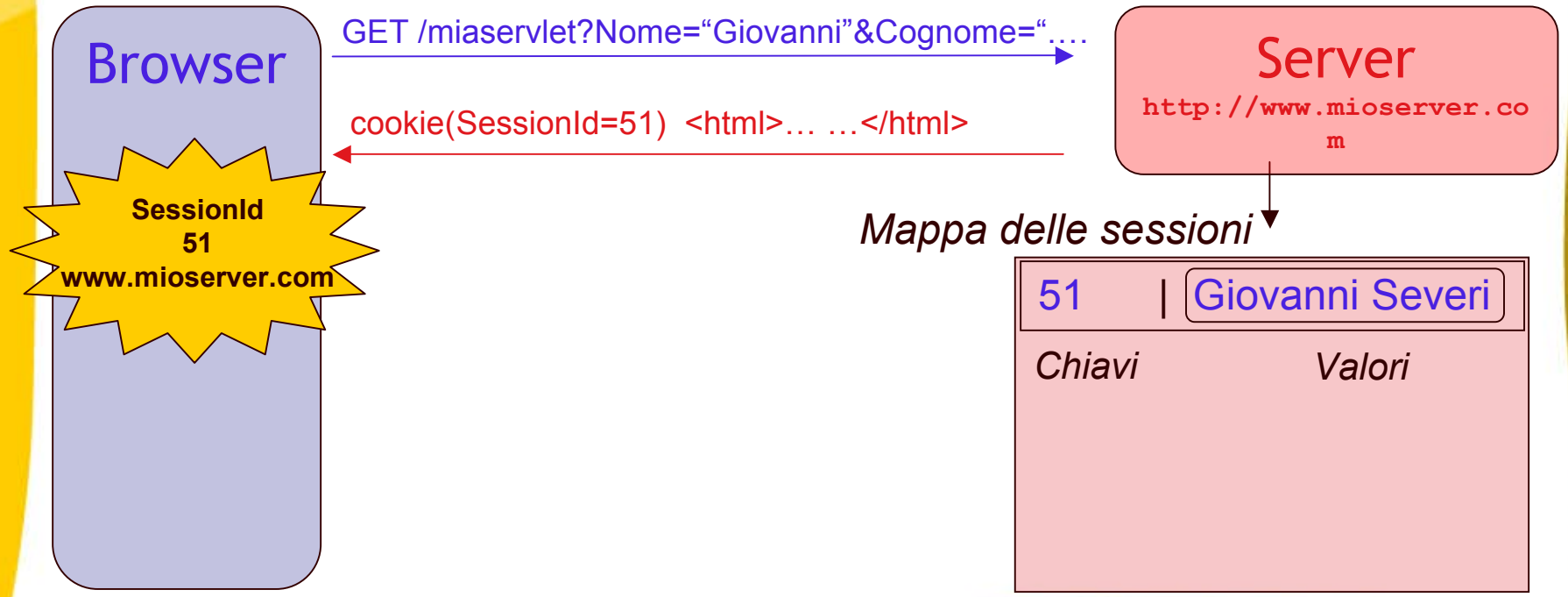
Riassunto: gestione della sessione tramite cookie



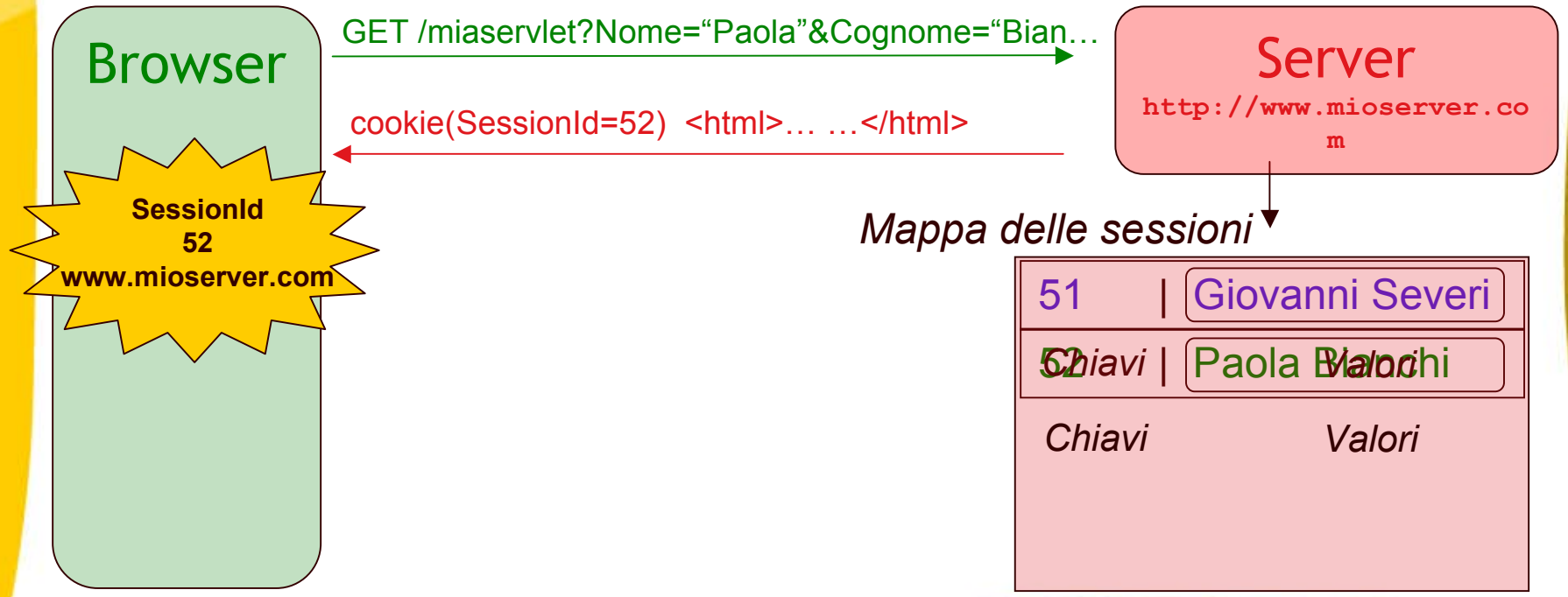
Gestione della sessione di navigazione tramite oggetti persistenti sul server

- Uso di oggetti che implementano l'interfaccia HttpSession

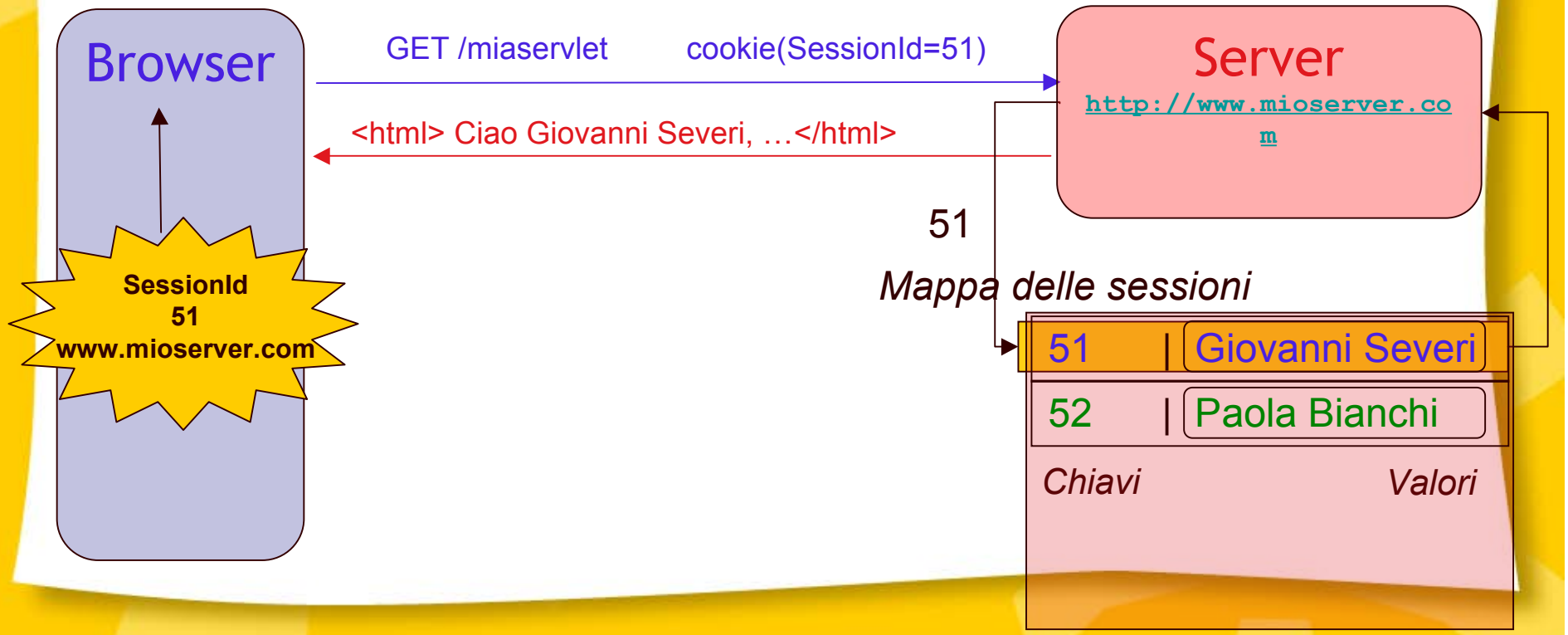
Gestione della sessione tramite oggetto persistente sul server (interfaccia **HttpSession**) (1/3)



Gestione della sessione tramite oggetto persistente sul server (interfaccia **HttpSession**) (2/3)



Gestione della sessione tramite oggetto persistente sul server (interfaccia **HttpSession**) (3/3)



Chiusura della sessione di navigazione

- Come detto, la cancellazione della sessione deve fare uso del metodo invalidate() dell'interfaccia HttpSession
- I cookie usati per la gestione della sessione sono cookie di sessione: scadono alla chiusura del browser

Uso dei cookie di sessione

- In pratica il server gestisce una **mappa**: ogni entry rappresenta una sessione di lavoro distinta
  - La chiave della mappa è un identificatore per la sessione
  - Il valore della mappa è un oggetto che contiene informazioni associate a quella specifica sessione
- In tutte le risposte al client il server aggiunge automaticamente e in maniera trasparente un cookie che contiene l'identificatore di sessione
- Attraverso questo identificatore, ad ogni richiesta il server è in grado di recuperare l'oggetto con le informazioni associate alla sessione

Supporto del container alla gestione della sessione di navigazione (1/2)

Tomcat (così come gli altri servlet container) mette a disposizione del programmatore una particolare infrastruttura e le API per una **gestione trasparente** di questo meccanismo:

- Nel processare una richiesta, il programmatore deve semplicemente chiedere all'oggetto HttpRequest l'oggetto associato alla sessione
- Il contenitore, in maniera trasparente, estrae dalla richiesta l'identificatore di sessione e lo usa per recuperare dalla mappa delle sessioni l'oggetto associato a quell'identificatore

Supporto del container alla gestione della sessione di navigazione (2/2)

- Se nella richiesta non viene trovato nessun identificatore allora, ove necessario (argomento true nel metodo getSession()), il container procede alla creazione della nuova sessione.
  - Vengono creati:
    - un nuovo identificatore,
    - un nuovo oggetto sessione,
    - una relativa entry nella mappa delle sessioni,
    - un cookie di sessione che viene agganciato alla risposta.
- Se il browser ha i cookie disabilitati come faccio a gestire la sessione sul server?

Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

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

Domande:

- Appurato che il meccanismo di gestione della sessione sul server basato su trasmissione di cookie potrebbe non funzionare, allora:
- 1. Perché non adottare sempre e solo l'url rewriting?
- 2. Perché non adottare entrambi i metodi contemporaneamente e non alternativamente?

Domanda 1: solo URL rewriting?

- Perché usare i cookie se a volte i browser li rifiutano? Non sarebbe stato meglio progettare container che utilizzassero sempre e solo l'URL rewriting?

  L'URL rewriting consente la gestione della sessione solo se l'utente naviga attraverso i link della pagina di risposta.

  La sessione viene persa se:

  - l'utente usa dei bookmark
  - l'utente usa il tasto backward raggiungendo una pagina richiesta precedentemente senza identificativo riscritto
  - l'utente scrive l'url sulla barra degli indirizzi del browser

Domanda 2: URL rewriting AND cookie?

- Perché il metodo encodeURL() non funziona semplicemente aggiungendo sempre l'id di sessione invece che farlo solo se il browser dell'utente non accetta i cookie?
- L'encoding dell'URL costituisce un carico sul server per riscrivere le stringhe
- Carico sulla rete perché ciascuna risposta conterrà diversi url riscritti (notare che nel caso di gestione tramite cookie il server invia il cookie di sessione solo al momento della sua creazione).

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.

Argomenti prossime lezioni

1. Ricapitolazione sulla redirezione tramite metodo sendRedirect dell'interfaccia HttpServletResponse
2. Concetto di sessione di navigazione
3. Uso di cookie per la gestione di una sessione
4. Uso dell'interfaccia HttpSession
5. URL rewriting nella gestione della sessione
6. Redirezione tramite il metodo forward dell'interfaccia RequestDispatcher

Redirezione di richieste

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");`**