

Introduzione alle Servlet 2

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*

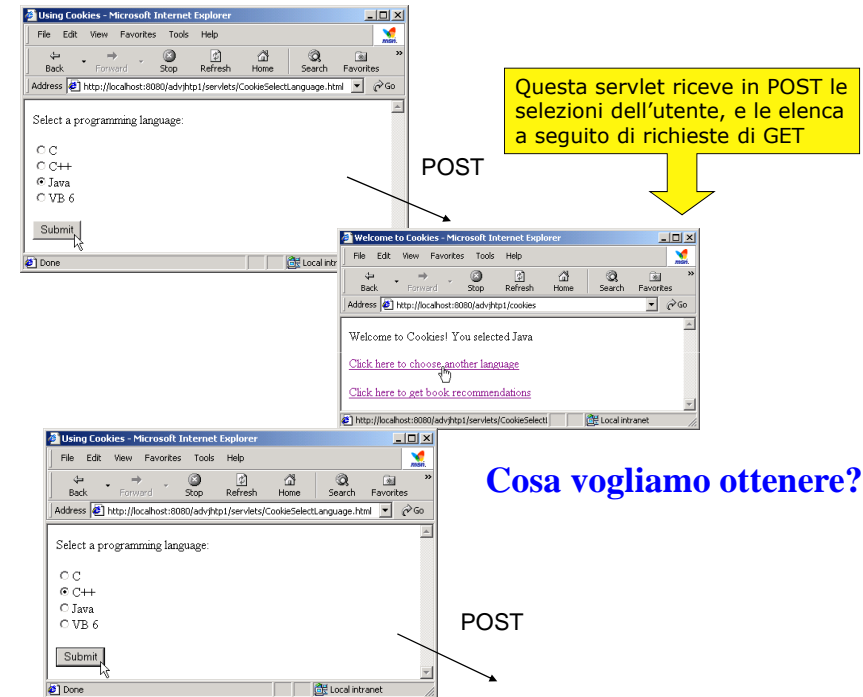
- Tecniche di gestione di informazioni persistenti durante una sessione di navigazione:
 - Cookie
 - Session tracking
 - hidden type input
 - URL rewriting (parametri get)

I cookie

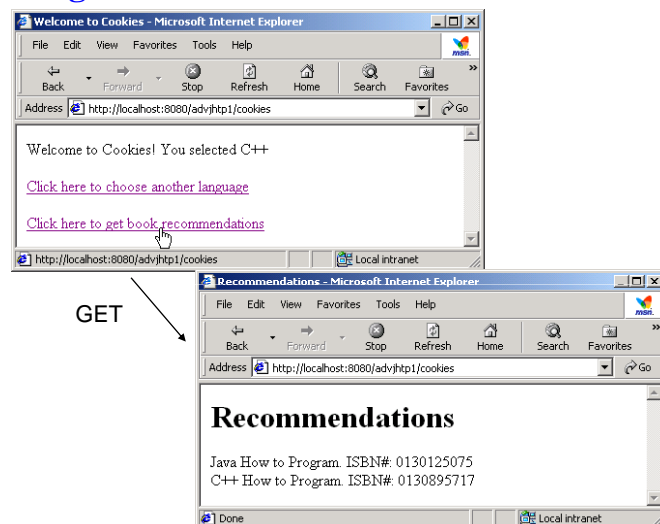
- Stringhe contenenti dati testuali nella forma di coppie (*nome, valore*)
- Vengono spediti dal server e memorizzati sul computer dell'utente (client) per utilizzi successivi
- Contengono le informazioni che si vogliono far persistere per richieste successive
- Un parametro ne caratterizza l'età massima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

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.



Cosa vogliamo ottenere?



Inviemo 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 devono 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
<code>getComment()</code>	Returns a String describing the purpose of the cookie (null if no comment has been set with <code>setComment</code>).
<code>getDomain()</code>	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.
<code>getMaxAge()</code>	Returns an int representing the maximum age of the cookie in seconds.
<code>getName()</code>	Returns a String containing the name of the cookie as set by the constructor.
<code>getPath()</code>	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.
<code>getSecure()</code>	Returns a boolean value indicating if the cookie should be transmitted using a secure protocol (true).
<code>getValue()</code>	Returns a String containing the value of the cookie as set with <code>setValue</code> or the constructor.
<code>getVersion()</code>	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 <i>Request for Comments (RFC) 2109</i> .

(Part 1 of 2)

Metodi della classe Cookie (2/2)

<code>setComment(String)</code>	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.)
<code>setDomain(String)</code>	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.
<code>setMaxAge(int)</code>	Sets the maximum age of the cookie in seconds.
<code>setPath(String)</code>	Sets the "target" URL prefix indicating the directories on the server that lead to the services that can receive this cookie.
<code>setSecure(boolean)</code>	A true value indicates that the cookie should only be sent using a secure protocol.
<code>setValue(String)</code>	Sets the value of a cookie.
<code>setVersion(int)</code>	Sets the cookie protocol for this cookie.

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`

```
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>
```

```

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

```

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**

```

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 );
33     response.setContentType( "text/html" );
34     PrintWriter out = response.getWriter();

```

```

36 // send XHTML page to client
37
38 // start XHTML document
39 out.println( "<?xml version = \"1.0\"?>" );
40
41 out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD \" +
42     \"XHTML 1.0 Strict//EN\" \"http://www.w3.org\" +
43     \"/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
44
45 out.println(
46     "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
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 }

```

```

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**

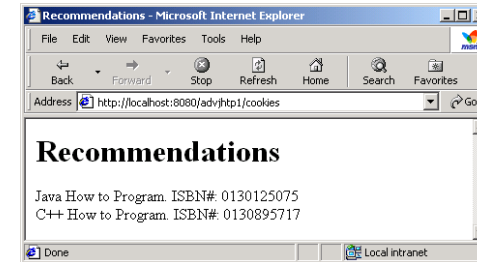
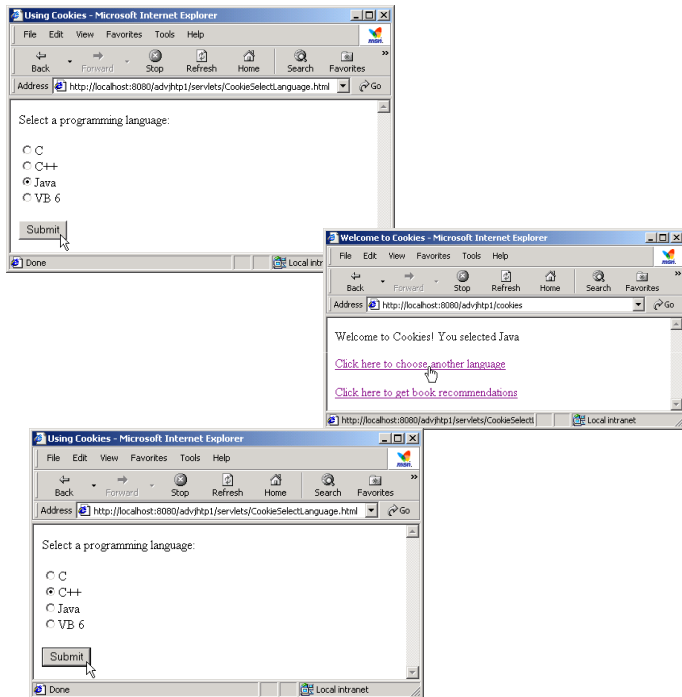
Nota:

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 }

```



web.xml (CookieServlet)

Descriptor element	Value
<i>servlet element</i>	
servlet-name	cookies
description	Using cookies to maintain state information.
servlet-class	DirectoryDiSaluto.CookieServlet
<i>servlet-mapping element</i>	
servlet-name	cookie
url-pattern	/cookies

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>
- Identifica un utente attraverso diverse richieste di pagina e permette di memorizzare informazioni su di lui e sulle azioni compiute durante la navigazione.
- Il servlet container usa questa interfaccia per creare una **sessione** tra un client HTTP e un server HTTP.
 - La sessione persiste per uno specifico periodo di tempo.
 - Corrisponde ad un solo utente, che può visitare il sito anche più volte.

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'id unico della sessione
 - boolean `isNew()`
Restituisce true se la sessione con il client è stata creata in occorrenza della presente richiesta
 - 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)

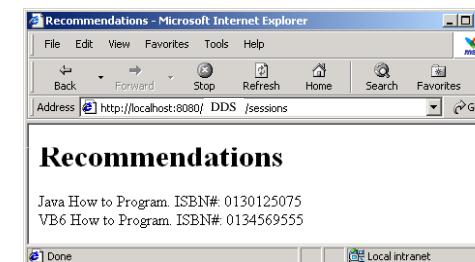
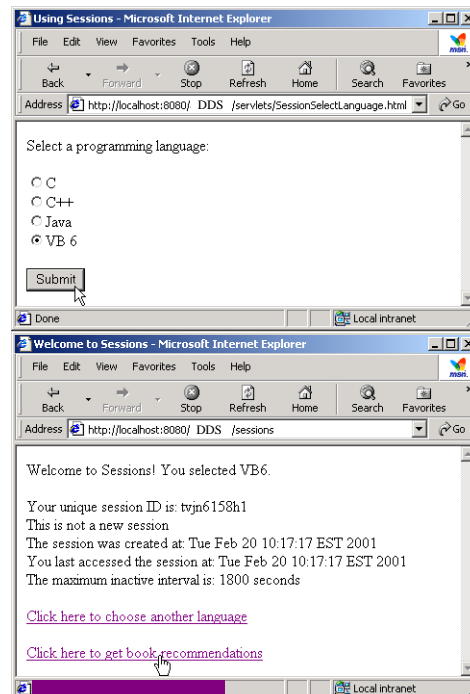
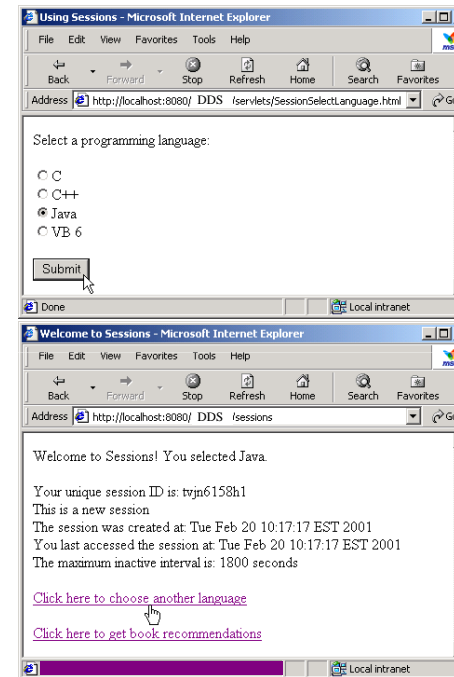
- Metodi dell'oggetto `HttpSession`
 - int `getMaxInactiveInterval()`
Restituisce il massimo intervallo di tempo, in secondi, in cui il container può mantenere la sessione aperta, tra due accessi consecutivi da parte del client.
 - java.util.Enumeration `getAttributeNames()`
Restituisce un oggetto Enumeration (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! You selected " +
55     language + ".</p>" );
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;

```

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

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.

Session Tracking with HttpSession (Cont.)

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

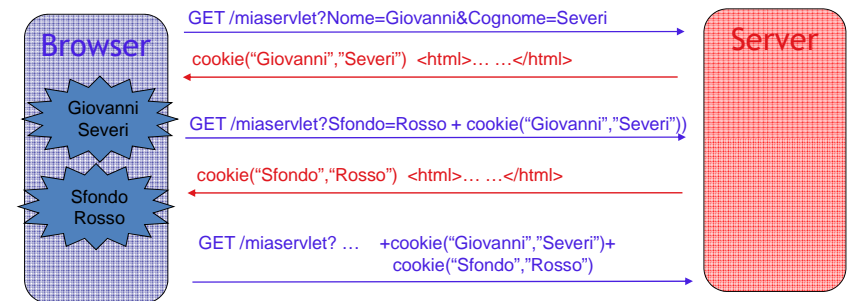
Chiusura della sessione di navigazione

- 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 (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?
- R: Viene usato un cookie!!!

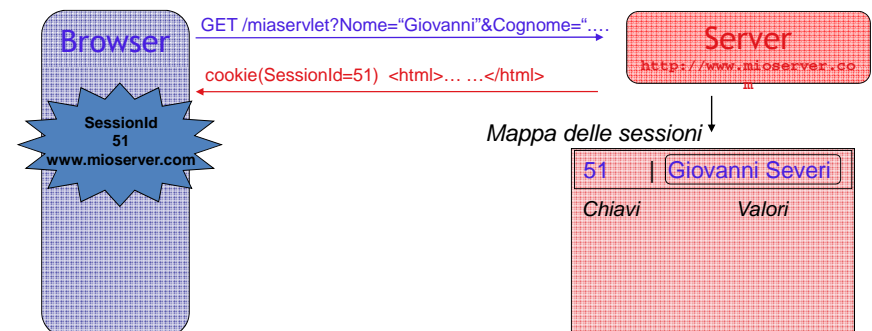
Gestione della sessione tramite cookie



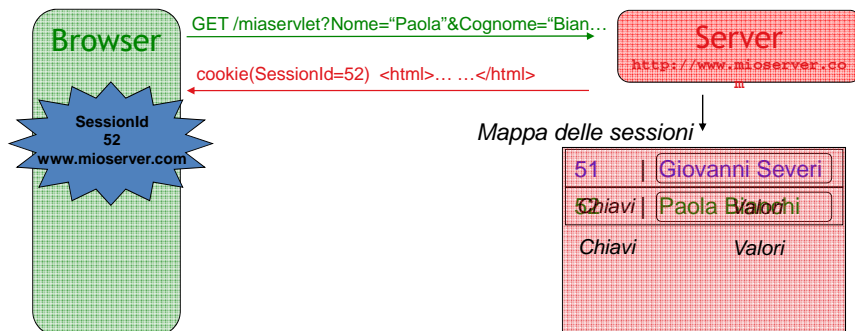
Gestione della sessione di navigazione tramite oggetti persistenti sul server

- Uso di oggetti che implementano l'interfaccia **HttpSession**

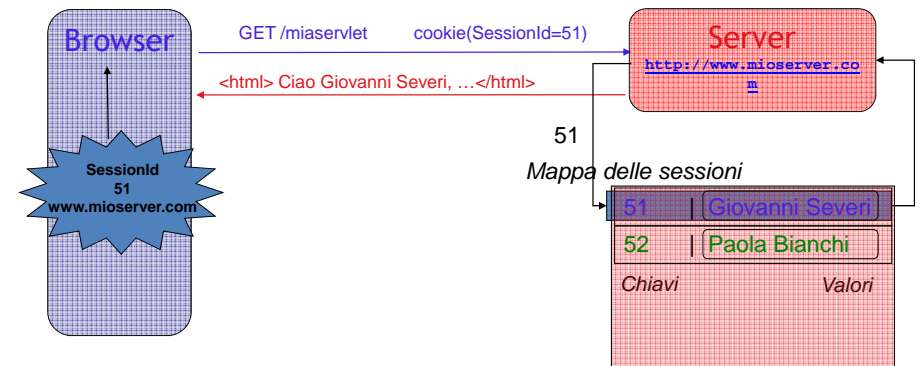
Gestione della sessione tramite (1/3)
oggetto persistente sul server (interfaccia **HttpSession**)



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



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



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)

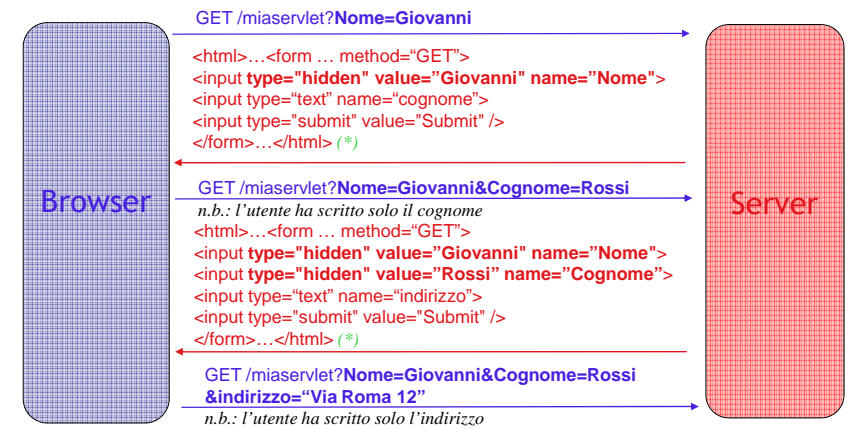
Tomcat (così come gli altri servlet container) mette a disposizione del programmatore una particolare infrastruttura e le API per una **gestione trasparente** di questo meccanismo:

- 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)

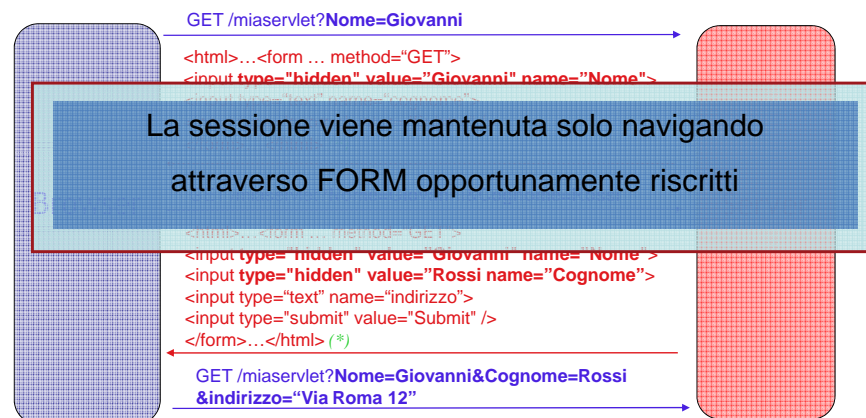
- Se nella richiesta non viene trovato nessun identificatore allora, ove necessario (argomento true nel metodo getSession()), il container procede alla creazione della nuova sessione.
 - Vengono creati:
 - un nuovo identificatore,
 - un nuovo oggetto sessione,
 - una relativa entry nella mappa delle sessioni,
 - un cookie di sessione che viene agganciato alla risposta.

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



(*) 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

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**).

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:

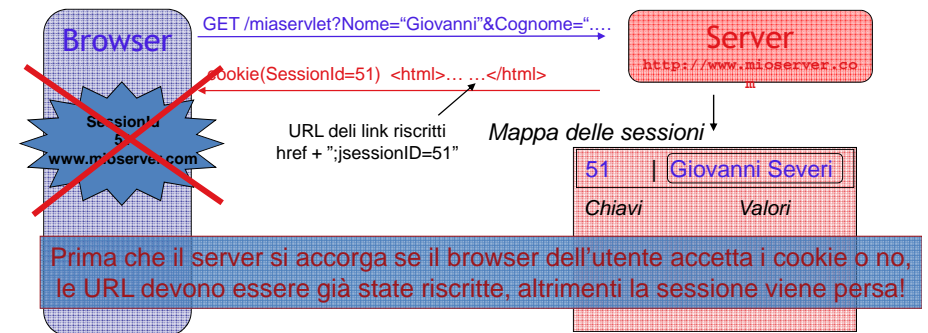
Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged

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()



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 accetta 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 tramite un cookie oppure tramite la URL della richiesta.
 - 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 HttpServletRequest 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.