

Gestione delle sessioni

Qualche lezione fa abbiamo detto che HTTP è stateless, quindi non memorizziamo informazioni riguardo il client sul server; di conseguenza ipotizziamo che tutte le informazioni necessarie sono presenti all'interno della richiesta, visto che **il server non conosce nulla di quell'interazione**.

Questo non significa che **non è possibile gestire una memoria di dialogo** in maniera diversa: usando la memoria del browser.

Di conseguenza il browser dovrà inviare tutto ciò che serve ogni volta, ma il client può memorizzare all'interno del browser stesso, delle informazioni, in modo che l'utente non debba inserirle ogni volta.

Cookie - gestione della sessione

Vediamo 3 componenti in gioco:

- Client
- Web Server
- Application - Servlet engine (nel caso in esecuzione all'interno di un engine Tomcat)

Il messaggio di richiesta arriva al server e viene inoltrato al servlet engine. Nel nostro caso il web server e servlet sono la stessa cosa. All'interno della componente applicativa un **cookie sarà un oggetto** e quando sarà trasferito verso il client sarà **una stringa**, ovvero il valore contenuto all'interno di **set-Cookie**.

Le richieste successive saranno collegate al messaggio di risposta precedente, quindi potranno usare le informazioni precedentemente recuperate dal server grazie al campo **Cookie**. In questo caso la stringa precedentemente ricevuta dal server, viene ritrasmessa a quest'ultimo. Insieme al cookie viene ovviamente inviata anche la richiesta.

Esempio cookie

Abbiamo una classe Cookie che possiamo usare per creare oggetti che il server invia al browser. Il browser li memorizza come stringhe (e non più oggetti).

L'applicazione a cui facciamo riferimento è un'applicazione di Ecommerce che ci permette di realizzare un'applicazione che riceve delle preferenze (nome esplicito inserito dall'utente in qualche momento del dialogo con il server), il server è in grado di fornire dei suggerimenti sulla base delle informazioni che l'utente ha precedentemente trasferito.

Viene quindi mantenuta memoria delle informazioni scambiate, e questo fa sì che l'utente non debba reinserire queste informazioni ogni volta.

Per realizzare l'applicazione usiamo una servlet ed impieghiamo due form:

1. fornisce al server delle preferenze
2. richiede al server dei suggerimenti

I due form operano con la stessa servlet perché ipotizzeremo che il primo form (esprimere preferenze) attivi l'esecuzione del metodo di servizio POST, mentre il secondo form è associato al metodo di servizio GET.

```
1
2 <HTML>
3 <HEAD>
4   <TITLE>Cookies</TITLE>
5 </HEAD>
6 <BODY>
7   <FORM ACTION=http://localhost:8080/servlet/CookieExample METHOD="POST">
8
9     <STRONG>Select a programming language: </STRONG>
10    <BR>
11
12    <INPUT TYPE="radio" NAME="lang" VALUE="C">C<BR>
13    <INPUT TYPE="radio" NAME="lang" VALUE="C++">C++<BR>
14    <INPUT TYPE="radio" NAME="lang" VALUE="Java"
15      CHECKED>Java<BR>
16    <INPUT TYPE="radio" NAME="lang"
17      VALUE="Visual Basic"><BR>
18
19    <INPUT TYPE="submit" VALUE="Submit">
20    <INPUT TYPE="reset">
21  </FORM>
22 </BODY>
23 </HTML>
```

Abbiamo dei campi come quelli visti in precedenza negli altri form, ovvero dei campi di input di tipo **radio (cerchi selezionabili)** dove possiamo esprimere una preferenza per un linguaggio di programmazione (e-commerce di libri). Di conseguenza se esprimo un interesse per il linguaggio C, successivamente mi saranno raccomandati libri sul linguaggio C.

```

1
2 <HTML>
3 <HEAD>
4     <TITLE>Cookies</TITLE>
5 </HEAD>
6 <BODY>
7     <FORM ACTION=http://localhost:8080/servlet/CookieExample METHOD="GET">
8
9         Press "Recommend books" for a list of books.
10        <INPUT TYPE=submit VALUE="Recommend books">
11    </FORM>
12 </BODY>
13 </HTML>

```

Il secondo form fa riferimento alla stessa servlet, ma è associato al metodo POST. Non ci sono campi di input, perchè questo form è usato per **visualizzare delle preferenze**.

Invio del cookie

Nel metodo `doPost` abbiamo il comportamento che il server prevede per creare un cookie contenente delle informazioni che il server vuole che vengano mantenute dal client, che verranno poi inviate al browser.

Viene creato un oggetto di tipo `Cookie`, che è composto dalla coppia **chiave-valore**; in questo caso il valore è un ISBN.

Infine, questo cookie viene aggiunto alla **risposta** da inviare al client (che lo memorizza). Quando il client riceve il messaggio di risposta, riceverà anche il cookie, anche se non se ne accorgerà.

```

public void doPost( HttpServletRequest request,
                    HttpServletResponse response )
    throws ServletException, IOException
{
    PrintWriter output;
    String language = request.getParameter( "lang" );

    Cookie c = new Cookie( language, getISBN( language ) );
    c.setMaxAge( 120 );
    response.addCookie( c );

    response.setContentType( "text/html" );
    output = response.getWriter();
}

```

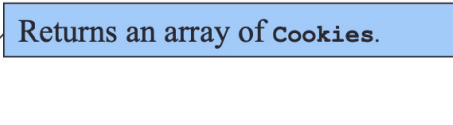
Creates a new **Cookie**,
containing the **language**.

Sets the maximum age of
the cookie and adds it to
the response

Utilizzo del cookie

Quando il client utilizzerà il form per la richiesta di un suggerimento, siccome il form è associato al metodo get, viene eseguito il metodo di servizio:

```
40
41 public void doGet( HttpServletRequest request,
42                   HttpServletResponse response )
43                   throws ServletException, IOException
44 {
45     PrintWriter output;
46     Cookie cookies[];
47
48     cookies = request.getCookies(); // get client's cookies
49
50     response.setContentType( "text/html" );
51     output = response.getWriter();
52
53     output.println( "<HTML><HEAD><TITLE>" );
54     output.println( "Cookies II" );
55     output.println( "</TITLE></HEAD><BODY>" );
56 }
```



Si prova a recuperare i cookie che il browser dovrebbe trasferire, quindi il server si aspetta che il browser gli invii i cookies che il server gli ha inviato in precedenza; `getCookies()` restituisce un array di cookie.

Se l'array è diverso da null (esistono), vengono analizzati tutti i cookie.

HttpSession - gestione della sessione

Questa soluzione cerca di evitare un potenziale problema che possiamo osservare con l'impiego dei cookie: se vogliamo usare come dato di sessione un dato di grandi dimensioni, può essere poco efficiente trasferire questo dato in tutte le interazioni.

La soluzione è quella di memorizzare l'oggetto di sessione sul server, e trasferire al client solo l'informazione di sintesi, sempre trasferita attraverso un cookie, che fa riferimento alla possibilità che si sta operando in una specifica sessione; grazie a questa informazione è possibile recuperare la sessione sul server, e non sul client (sul client è presente solo il riferimento alla sessione che però è presente sul server).

Gli oggetti che sono mantenuti sul server sono ospitati all'interno di `HttpSession`, e ci sarà un'istanza dell'oggetto per ogni client che dialoga con il server.

Come programmiamo la memoria di sessione?

L'oggetto contenitore viene recuperato da un messaggio di richiesta attraverso l'invocazione di `getSession()`, che restituisce un oggetto di tipo **HttpSession**. In alternativa possiamo avere l'invocazione di `getSession` con due parametri bool:

- `getSession(true)` crea l'oggetto contenitore dei dati di sessione se non è stato creato in precedenza, lo restituisce se è stato creato in precedenza

- `getSession(false)` recupera l'oggetto contenitore se questo è stato creato in precedenza, altrimenti restituisce null, quindi non è in grado di crearlo.

Applicazione di ecommerce - stessa app precedente

Nel primo metodo invocato, usato per memorizzare le preferenze dell'utente viene invocato il metodo **`getSession(true)`**, quindi se non è presente una sessione, questa viene creata. Successivamente invocando **`setAttribute()`** memorizziamo nel contenitore la coppia linguaggiopreferito-codiceISBN.

```
public void doPost( HttpServletRequest request,
                    HttpServletResponse response )
    throws ServletException, IOException
{
    PrintWriter output;
    String language = request.getParameter( "lang" );

    // Get the user's session object.
    // Create a session (true) if one does not exist.
    HttpSession session = request.getSession( true );

    // add a value for user's choice to session
    session.setAttribute ( language, getISBN( language ) );
}
```

Loads **`HttpSession`** if it exists, otherwise it creates it

Stores the pair (name, value)

Per quanto riguarda il metodo `get` (da eseguire quando viene ricevuta la richiesta per avere un suggerimento) viene invocato **`getSession(false)`**, perchè non ci interessa crearne una nuova (a questo punto) se questa non esiste.

```

public void doGet( HttpServletRequest request,
                  HttpServletResponse response )
    throws ServletException, IOException
{
    PrintWriter output;

    HttpSession session = request.getSession( false );

    // get names of session object's values
    String valueNames[];

    if ( session != null )
        valueNames = session.getAttributeNames();
    else
        valueNames = null;

    response.setContentType( "text/html" );
    output = response.getWriter();

    output.println( "<HTML><HEAD><TITLE>" );
    output.println( "Sessions II" );
    output.println( "</TITLE></HEAD><BODY>" );

    if ( valueNames != null && valueNames.length != 0 ) {
        output.println( "<H1>Recommendations</H1>" );

        // get value for each name in valueNames
        for ( int i = 0; i < valueNames.length; i++ ) {
            String value =
                (String) session.getAttribute( valueNames[ i ] );

            output.println(
                valueNames[ i ] + " How to Program. " +
                "ISBN#: " + value + "<BR>" );
        }
    }
    else {
        output.println( "<H1>No Recommendations</H1>" );
        output.println( "You did not select a language or" );
        output.println( "the session has expired." );
    }
}

```

It does not create the session object if it does not exist. **session** is set to null.

Gets the values associated to the names.

I dati di sessione (che possono essere anche molto grandi) sono memorizzati sul server, ed al client viene inviata un'informazione sintetica che fa riferimento alla sessione reale presente sul server.

🚩 1:34

Esercizi 9.x

In questo esercizio dobbiamo scrivere una servlet che quando invocata ritorni la stringa: "hai contattato questo componente <n> volte"

Di conseguenza non possiamo avere una semplice variabile di istanza, visto che conterebbe le volte che è stata contattata da **tutti** i client.

Potremmo usare i cookies o HttpSession; usiamo la seconda perchè più difficile da implementare. Possiamo inizializzare la sessione a partire da request.getSession();

Se però è la prima volta che ci si connette al server, dobbiamo inizializzare la sessione con `session.setAttribute()`

In realtà il codice complessivo è abbastanza semplice:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    int myContacts = 0;
    HttpSession session = request.getSession();

    // se la sessione non è nuova
    if(!session.isNew()) {
        myContacts = (Integer) session.getAttribute("myContacts");
    }
    // se la sessione è nuova
    session.setAttribute("myContacts", ++myContacts);

    response.setContentType("text/plain");
    PrintWriter pw = response.getWriter();
    pw.println("Hai contattato il server " + myContacts + " volte.");

}
```

Fine lezione 32