Interazione Uomo-Macchina e tecnologie web

Prof.ssa Liliana Ardissono
Dipartimento di Informatica
Università di Torino

Programmazione server-side: Java Servlets – parte 2

Protocolli di interazione stateless e stateful

- **Protocollo stateless**
  - Ogni richiesta del client è gestita in una nuova sessione ⇒ il server non ha memoria delle precedenti richieste fatte dal client. Es: **HTTP 1.0**
- **Protocollo stateful**
  - Un protocollo stateful gestisce un canale virtuale di comunicazione. Il canale permette di vedere richieste e risposte multiple come parte di una stessa connessione tra client e server
  - ⇒ il protocollo permette di gestire risposte a richieste che dipendono dal contenuto della richiesta e dai risultati delle precedenti richieste
  - es: FTP (se io eseguo “dir” dopo aver fatto “cd miaCartella”, io vedo la lista di file che stanno sotto “miaCartella”)

Gestione della sessione utente - motivazioni

Nello sviluppo di applicazioni web serve un meccanismo che permetta di

1. **riconoscere la sequenza di azioni effettuate dallo stesso browser durante interazione** - se no non si può capire che le richieste appartengono alla stessa interazione ⇒ **gestione della sessione utente (session tracking)**

2. **mantenere una memoria associata alla particolare interazione**, per memorizzare i dati l'esecuzione delle operazioni ⇒ **gestione dello stato della sessione utente**

Gestione della sessione utente - motivazioni - I

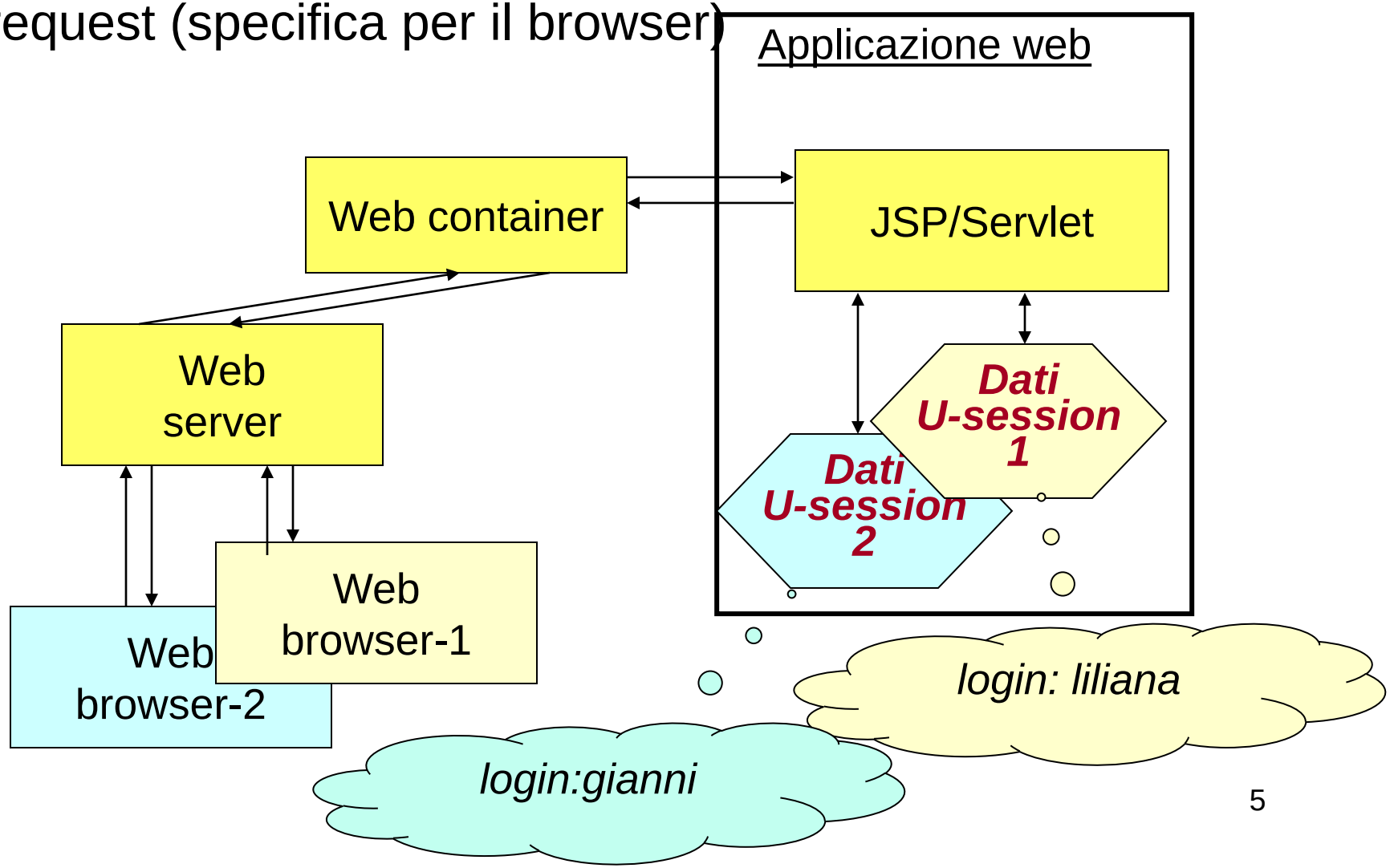
Per esempio, molte applicazioni web devono mantenere dei dati durante l'interazione con l'utente

- **autenticazione dell'utente** ($\Rightarrow$ bisogna mantenere i suoi dati per tutta l'interazione, per es., per adattare l'interazione a ruoli utente diversi)

- **salvataggio di scelte fatte dall'utente.** Es:
  - inserimento di prodotti nel carrello della spesa
  - impostazione di criteri per la ricerca di prodotti (prezzo max di camera d'albergo, data di arrivo e partenza desiderate, etc.)

Interazioni parallele con l'applicazione

Per abbinare i dati all'interazione a cui si riferiscono, lo stato della sessione utente viene associato alla HTTP request (specifica per il browser)



Sessione utente e stato dell'interazione

- **Riconoscimento della sessione utente (session tracking)**
  - Il server **associa le richieste successive di uno stesso browser ad una sola interazione (traccia l'identità del client)**
- **Mantenimento dello stato della sessione utente**
  - Il server **ricorda i dati relativi alle precedenti richieste** effettuate durante la sessione. *Può rispondere alle richieste tenendo conto di tali dati*

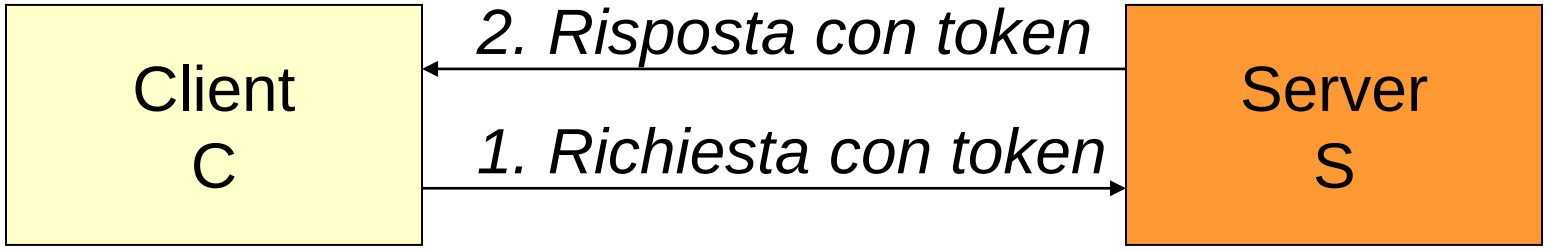
NB: la gestione della sessione utente è necessaria per gestirne lo stato, ma non sufficiente

Es: HTTP 1.1 supporta le sessioni utente ma non lo stato delle interazioni

Tecniche di session tracking – modello generale

Le tecniche di session tracking sono basate sullo scambio di **token identificativi** tra client (C) e server (S):

- C invia richiesta HTTP a S
- S risponde associando un token univoco T alla risposta
- C invia ulteriori richieste associando il token T. T permette a S di riconoscere C



Tecniche di session tracking – Cookies - I

- **Cookie**
  - informazione testuale inviata dal server al client. Il cookie viene memorizzato sul client e restituito dal client al server ad ogni richiesta HTTP
  - contiene una lista di coppie <nome, valore>, con attributi addizionali (come campo commento, massimo tempo di vita del cookie, …)
  - **Client e server scambiano il cookie nell'header di ciascuna HTTP request e response**
- Il Web server invia un cookie come segue (esempio)
  - Set-cookie: uid=pippo; age=3600; Path="/"

nome del cookie: uid
valore del cookie: pippo
attributi: domain, ...

Tecniche di session tracking – Cookies - II

- Normalmente il session tracking viene effettuato usando i **cookies**

- Però, **se il browser rifiuta i cookies,** il session tracking non funziona bene perché **ad ogni richiesta si crea una nuova sessione utente**

- URL rewriting sopperisce a questo problema

Tecniche di session tracking - URL rewriting

- **Data l'HTTP request del client, il server**
  - **genera il token T** identificativo del client
    - Es: T= 2F392C7084114FADC082D9BA22D81957
  - **inserisce il token come parametro delle richieste** per ciascun link (ancora) delle pagine di risposta che invia a client: **sessionid=T**
    - Es: link

- **Per ogni link che l'utente può seguire, l'HTTP request contiene anche il parametro *sessionid***, usato dal server per riconoscere il client

Altre tecniche di session tracking

- **Hidden form fields**
  - Il token è inserito in campi nascosti delle form
- **Sessioni che usano Secure Socket Layer (SSL)**
  - SSL usa il protocollo di scambio di messaggi crittografati HTTPS. Per stabilire le connessioni, client e server generano session keys (chiavi simmetriche per crittografare e decrittare messaggi). Tali chiavi possono essere usate per identificare il client e la sessione utente

Gestione di sessioni utente con le Servlet

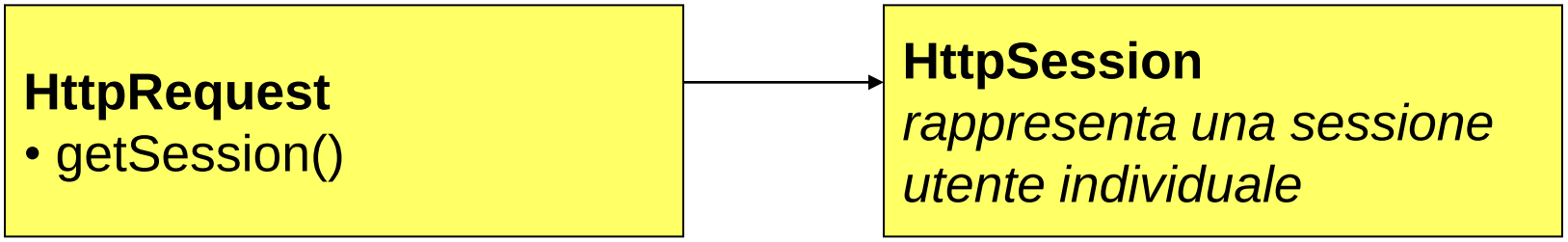
- Le Servlet permettono il session tracking
  - Esse utilizzano i cookies e/o l'URL rewriting come strumenti di session tracking
- e il mantenimento dello stato della sessione utente
  - Esse offrono un oggetto di tipo “**HttpSession**” per salvare i dati da mantenere nell’ambito di una sessione di interazione con l’utente
  - L'Interface **javax.servlet.HttpSession** incapsula la nozione di sessione utente *(il Web Container implementa HttpSession)*

HttpRequest e sessioni utente

L'Interface **HttpServletRequest offre il metodo getSession()** per accedere all'oggetto HttpSession associato alla **richiesta utente**

Il Web container gestisce

- gli scambi di token tra client e server
- le associazioni tra token e oggetti HttpSession



HTTPServletRequest - getSession() - I

- se l'HTTP request non ha un oggetto HTTPSession associato, getSession() crea nuovo oggetto sessione e lo restituisce al chiamante. Questo accade alla prima richiesta fatta da un browser, o se il browser non accetta i cookies.
- altrimenti getSession() restituisce l'oggetto sessione esistente. L'oggetto sessione è stato creato da una precedente richiesta effettuata dallo stesso browser nell'ambito dell'interazione con l'utente.

HTTPServletRequest - getSession() - II

I metodi doGet() e doPost() hanno come primo parametro l'oggetto HttpServletRequest che tiene i dati della sessione utente a cui appartiene la richiesta HTTP.

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response)  throws
    ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        String nm = request.getParameter("nome");

        HttpSession s = request.getSession();
…
}
```

javax.servlet.http - Interface HttpSession - I

- **String getId():**
  - restituisce l'identificativo di sessione (token, es: 2F392C7084114FADC082D9BA22D81957)
- **long getLastAccessedTime():**
  - restituisce il numero di millisecondi trascorsi dall'ultima richiesta del client
- **int getMaxInactiveInterval():**
  - restituisce il numero massimo di secondi di attività della sessione tra due richieste utente, dopodiché la sessione scade
- **long getCreationTime()**
  - restituisce la data di creazione della sessione utente

javax.servlet.http - Interface HttpSession - II

- **void invalidate():**
  - fa terminare la sessione utente (per esempio, per implementare operazione di logout) – distrugge l'oggetto HttpSession
- **Object getAttribute(String name)**
  - restituisce valore dell'attributo “name” memorizzato nella sessione utente (oggetto HttpSession), se l’attributo esiste. Null altrimenti
- **Enumeration getAttributeNames()**
  - restituisce l’enumerazione dei nomi degli attributi memorizzati nella sessione utente
- **void setAttribute(String name, Object o)**
  - salva in sessione l'attributo name con valore o (o è un riferimento a un oggetto)

NOTE: Attributi di una sessione utente

NB: setAttribute() salva il riferimento all'oggetto java ⇒
  - se il riferimento cambia bisogna rieseguire il metodo
  - se invece cambiano solo i valori dei campi dell'oggetto non è necessario

- **removeAttribute()** rimuove un attributo dalla sessione utente

- **getAttribute(), setAttribute() e removeAttribute() non sono synchronized** ⇒ se thread diversi modificano una stessa sessione utente si possono avere problemi ⇒ gestite la mutua esclusione!

Sessione utente – esempio

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response)  throws ServletException,
    IOException {
        String nm = request.getParameter(“nome");

        HttpSession s = request.getSession();
        s.setAttribute("userName", nm);

…
}
```

Recupera l'oggetto sessione dall'HTTPRequest, se esiste. Se non c'è, viene creato sul momento

Salva il nome dell’utente in sessione utente, come attributo di nome userName, per renderlo disponibile a tutte le componenti dell'applicazione durante l’interazione con l’utente

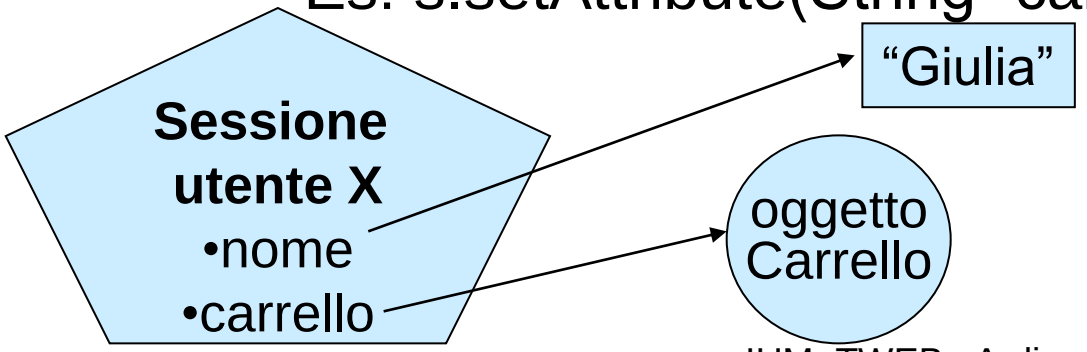
Interfaccia HttpSession - stato della sessione utente – cioè, mantenimento di dati nella memoria del web container

Un oggetto HttpSession s può contenere attributi che descrivono lo stato della sessione utente

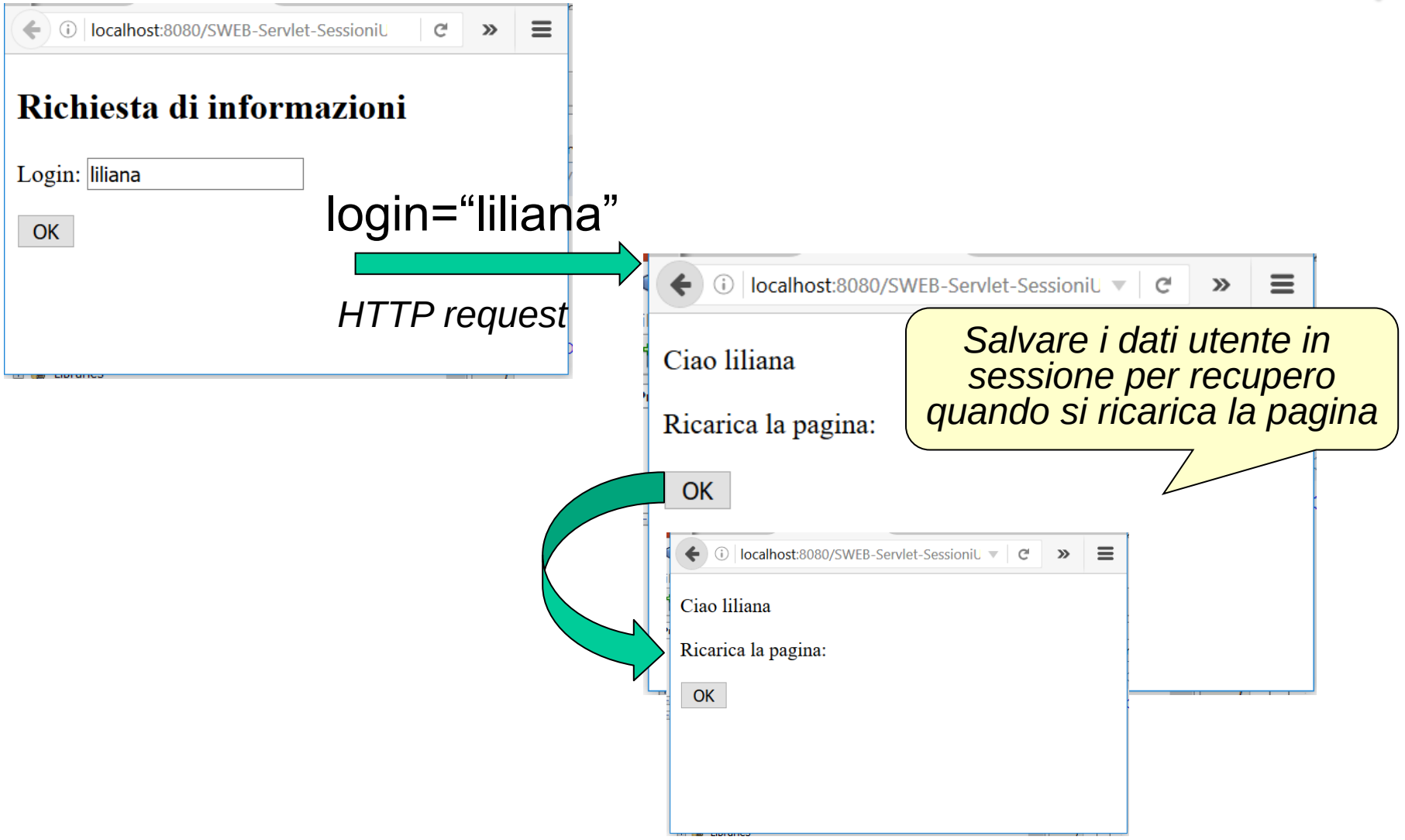
- Es: s.setAttribute(String “nome”, “Giulia”);

NB: potrei salvare un riferimento a un oggetto complesso (es. Carrello contiene i prodotti scelti dall’utente in un catalogo)

- Es: s.setAttribute(String “carrello”, new Carrello());



Es: gestione di dati in sessione utente – ServletSessioniUtente0Mav



GestioneSessioneUtente0– codice - I

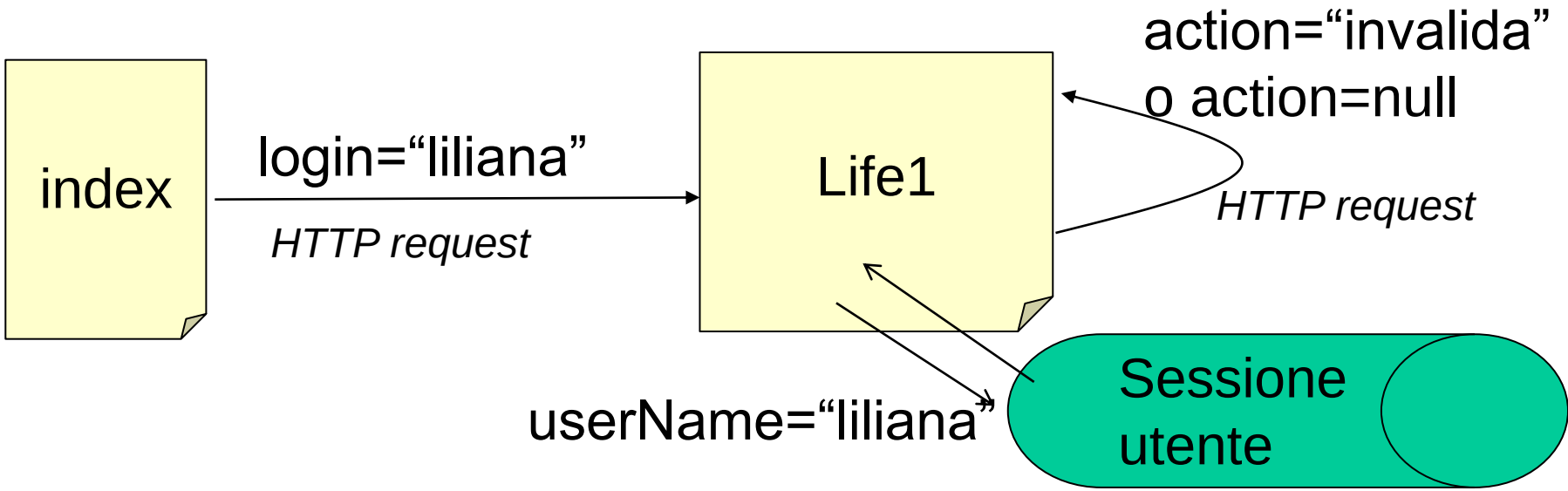
```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response)  throws ServletException, IOException {
      String account = request.getParameter("login"); // prendo login da richiesta HTTP
      HttpSession s = request.getSession(); // prendo il riferimento alla sessione utente
      if (account!=null)
          s.setAttribute("account", account);
      try (PrintWriter out = response.getWriter()) {
          out.println("<!DOCTYPE html>");
          out.println("<html><head>("<title>Servlet GestisciSessione</title></head>");
          out.println("<body>");
          out.println("<p>Ciao " + s.getAttribute("account") + "</p>");

          out.println("<p>Ricarica la pagina: ");
          out.println("<form action=\"hello-servlet\" method=\"post\">"
                  + "   <input type=\"submit\" name=\"submit\" value=\"OK\"/></form>
    </p>");
          out.println("</body>");
          out.println("</html>");
      }
  }
```

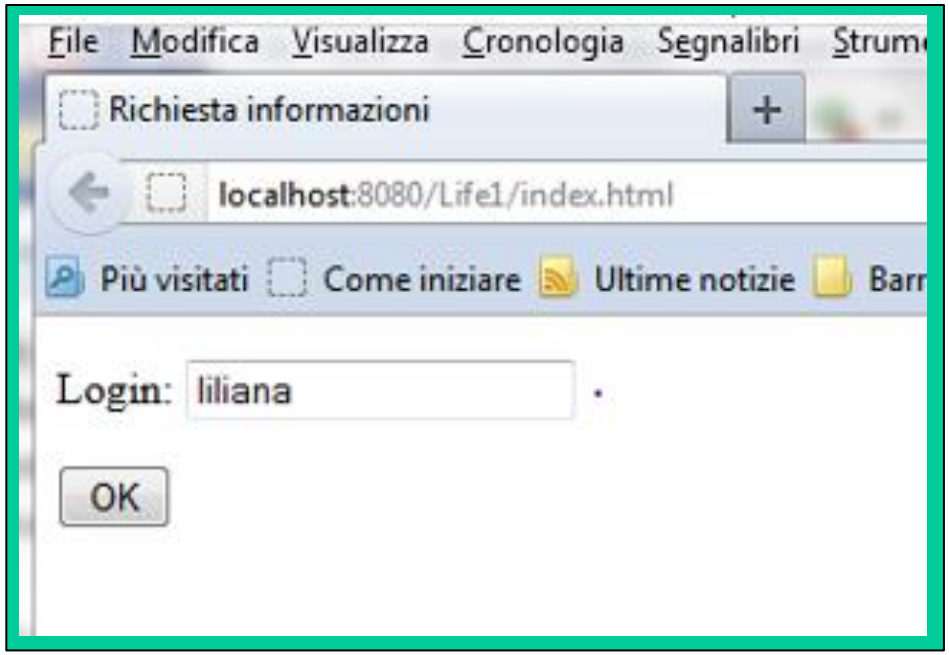
Se HTTPRequest contiene parametro login, viene salvato in sessione utente

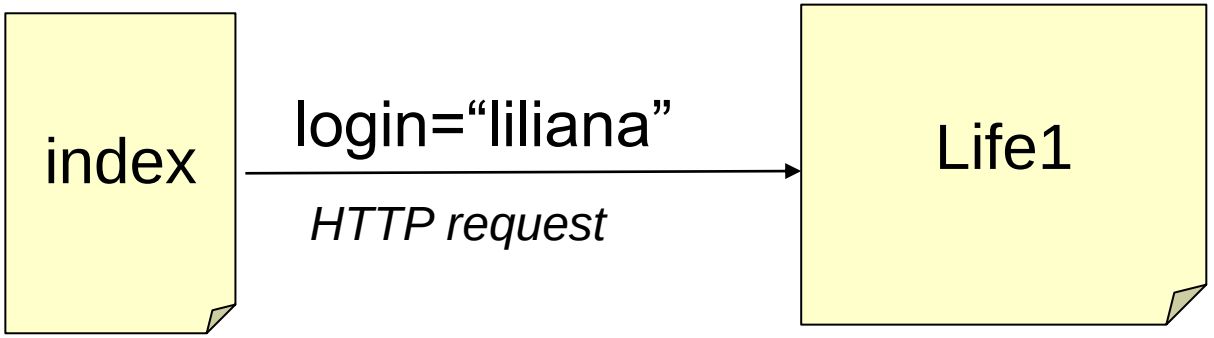
Prende account da sessione utente

Life1: Flusso di informazioni tra pagine

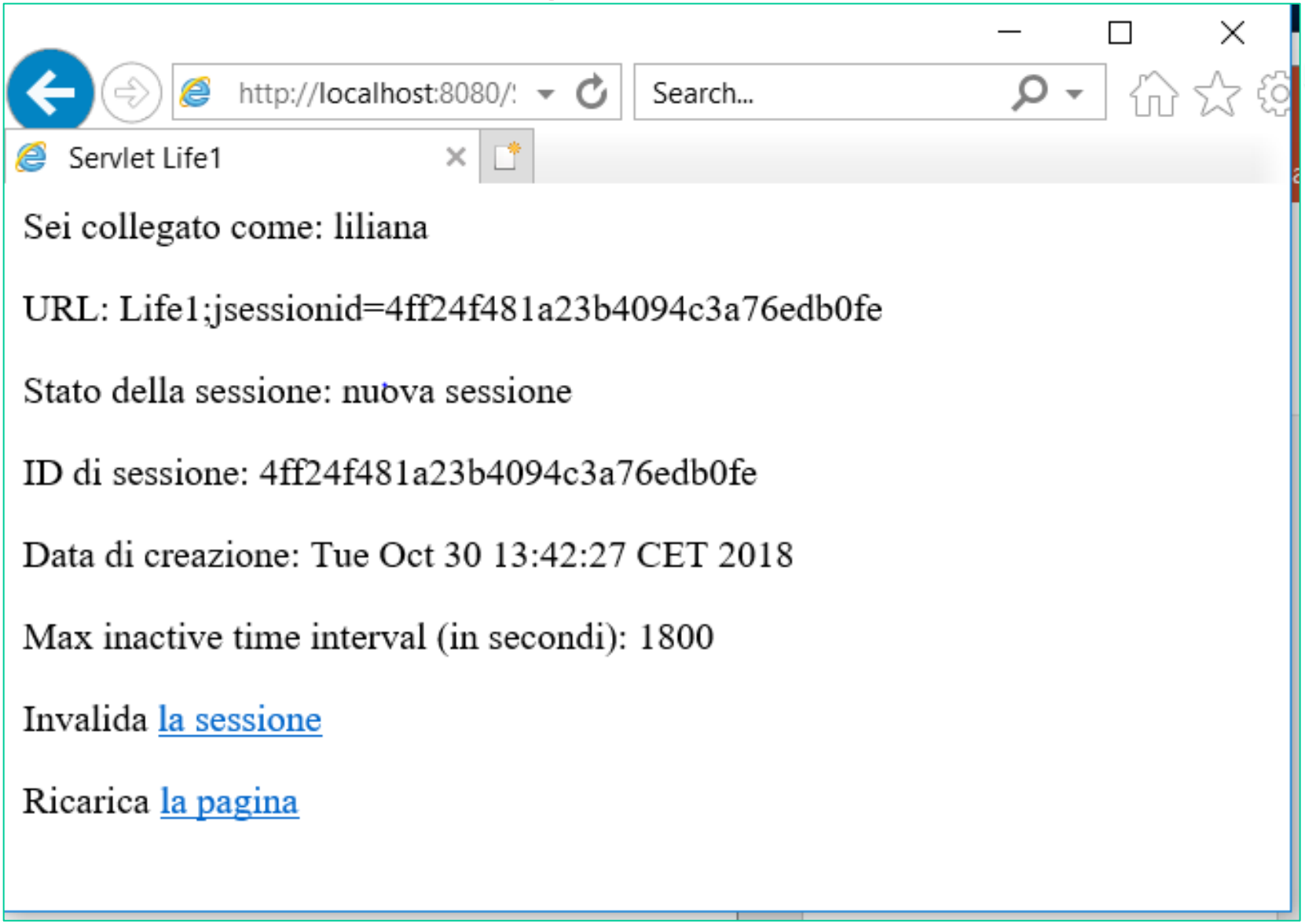


Life1 – pagina 1 (index.html)

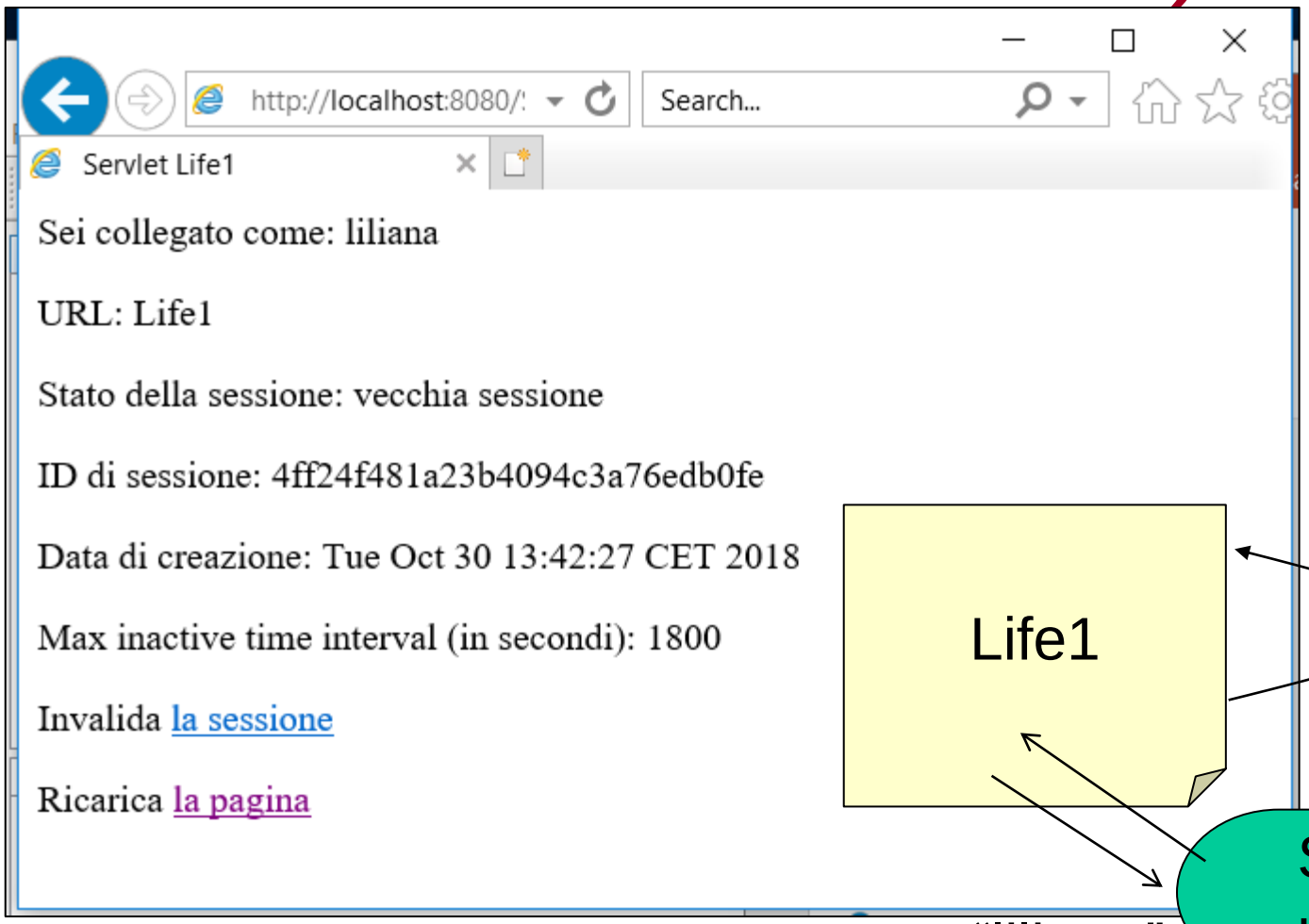




Life1 – seconda pagina– *sessione utente nuova*



Life1 – ricarico la pagina *(entro il timeout – sessione vecchia)*



Life1

action="invalida"
o action=null

HTTP request

Sessione utente

userName="liliana"

Life1 – codice - I

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response)  throws ServletException,
    IOException {
      response.setContentType("text/html;charset=UTF-8");
      String userName = request.getParameter("login");
      HttpSession s = request.getSession();

if (userName!=null)
        s.setAttribute("userName", userName);
      String url = response.encodeURL("life1-servlet");
      PrintWriter out = response.getWriter();
      try {
        out.println("<html><head>");
        out.println("<title>Servlet Life1</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<p>Sei collegato come: " +
                                  s.getAttribute("userName") + "<p>");
//... continua...
```

Se HTTPRequest contiene il parametro login, salvarlo in sessione utente

*Faccio computare l'url per le invocazioni successive della Servlet, Con URL encoding – **vd. poi***

Prendo userName dalla sessione utente

Life1 – codice - II

```
//… continua…
        String azione = request.getParameter("action");
         out.println("<p>" + url + "</p>");
         if (azione!=null && azione.equals("invalida")) {
            s.invalidate();
            out.println("<p>Sessione invalidata!</p>");
            out.println("<p>Ricarica <a href=\"" + url + "\"> la pagina</a></p>");
            }
         else {
            out.print("<p>Stato della sessione: ");
            if (s.isNew()) out.println(" nuova sessione</p>");
            else out.println(" vecchia sessione</p>");

            out.println("<p>ID di sessione: "+s.getId());
            out.println("<br>Data di creazione: " +
                                    new Date(s.getCreationTime()));
            out.println("<br>Max inactive time interval (in secondi): “ +
                                          s.getMaxInactiveInterval() +
    "</p>");
// … continua …
```

Distrugge la sessione utente

URL per ricaricare pagina generato con URL encoding

Recupera informazioni sulla sessione (id, etc.)

Life1 – codice - III

```
//… continua...
            out.println("<p>Invalida <a href=\"" + url +
    "?action=invalida\"> la sessione</a></p>");
            out.println("<p>Ricarica <a href=\"" + url + "\"> la
    pagina</a></p>");
        }
        out.println("</body>");
        out.println("</html>");
      } finally {
        out.close();
      }
   }
```

NB: Life1 gestisce uno stato che include solo il nome dell'utente. Però lo stato potrebbe contenere oggetti java complessi, come un oggetto Utente che contiene i dati dell'utente di uso frequente nell'applicazione

NOTE: memorizzazione di dati in sessione utente – siate parchi!

- **Troppi dati nelle sessioni utente appesantiscono la gestione dell'applicazione** perché occupano memoria del Web Container
- Quindi le applicazioni devono mettere in sessione utente il minor numero di dati possibile. Per es:
  - **usate le richieste HTTP, *non* la sessione utente, per passare parametri da una pagina di applicazione all'altra** (request.setAttribute()/getAttribute())

Session tracking con URL rewriting - I

Gli url da inserire nelle pagine devono essere computati dinamicamente, sulla base dell'id di sessione utente, anziché essere fissi:

url = response.encodeURL("life1-servlet");

- L'url generato contiene il token di sessione, memorizzato nel parametro jsessionid. Es:

\Life1\Life1;jsessionid=54C84D2221E28E60B1D4C8C8521BA89E

- *NB: response.encodeURL() deve essere invocato DOPO aver fatto generare la sessione utente, quindi dopo request.getSession()*

Session tracking con URL rewriting - II

- Siccome il token viene inserito negli URL, funziona anche se il browser non accetta i cookies
- Per impostare URL rewriting bisogna scrivere del codice specifico nell'applicazione Web – vd. Life1

```
...
HttpSession s = request.getSession();
... // encodeURL va invocato DOPO getSession();
    // fa generare un url che contiene il token di sessione
String url = response.encodeURL("life1-servlet");
out.println("<p>Ricarica <a href=\" + url + "\>
                                    la pagina</a></p>");
// uso l'url computato da encodeURL per inserire il link alla pagina
// di applicazione successiva
```

Durata delle sessioni utente

- NB: le sessioni utente scadono per timeout per non caricare il server quando gli utenti abbandonano l'interazione
- Il tempo di **timeout** è specificato **in minuti** nel deployment descriptor **web.xml**. Per esempio, 30 minuti:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    …">
...
   <session-config>
        <session-timeout> 30 </session-timeout>
   </session-config>
<web-app>
```