Programmazione per il Web
Note per la lezione del 21/03/2016

Igor Melatti

Cookie, URL rewriting e Session Tracking

- Slide 198: la prima volta che un `encodeURL` viene usato, non c'è modo di sapere se sul client i cookie sono disabilitati oppure no
  - quindi, `encodeURL`, la prima volta (dove prima volta = nell'URL non c'è `JSESSIONID e` nella richiesta non ci sono cookie) invia sia il cookie di sessione (un'unica associazione `JSESSIONID=valore`) che l'URL riscritto
  - la seconda volta, l'URL ritornerà riscritto (a meno che l'utente non abbia "barato" in uno dei modi descritti a slide 194), e nella richiesta i cookie ci saranno oppure no a seconda che siano o no abilitati sul client
  - sulla base di questo, può decidere se occorre continuare a riscrivere l'URL (se nella richiesta i cookie non ci sono) o non farlo più (altrimenti)
  - per vedere tutto ciò all'opera, si può fare come segue:
    - prendere `slide159_urlrewriting` della lezione scorsa e provare a visualizzare la seguente sequenza: HTML statico di partenza → scegliere un'opzione e cliccare su submit → cliccare sul primo link, che rimanda al form di scelta → scegliere un'opzione e cliccare su submit → cliccare sul primo link, che rimanda al form di scelta
    - farlo sia con i cookie abilitati che poi con i cookie disabilitati (dopo aver cancellato i cookie di localhost...)
    - in entrambi i casi, al terzo passaggio il link sarà riscritto, mentre all'ultimo passaggio lo sarà solo nel caso dei cookie disabilitati
- Esempi `TestCookies` e `TestCookiesSession`: servlet per capire se i cookie sono abilitati o no, usando e non usando slide 201
  - in realtà, è possibile "confondere" `TestCookies`, facendogli ritornare il fallimento anche se i cookie sono disabilitati: come si può fare?

Redirezione tra Servlet

- slide 204, in realtà:
  - non si può invocare con `ServletContext.getServletConfig` perché non è un metodo statico; è scritto così solo per ricordare che è un metodo dell'interfaccia `ServletContext`
  - c'è quindi anche `ServletContext`, che fornisce a sua volta `getInitParameter` che `getInitParameterNames` (più altro, vedere più sotto)
  - differenza: `ServletConfig` contiene inizializzazioni potenzialmente diverse per ogni servlet
    - tali inizializzazioni vanno messe dentro `web.xml`, all'interno di un tag `init-param`, che contenga una coppia di tag `param-name` e `param-value`
    - il tag `init-param` può essere specificato più volte per dare diversi valori di configurazione
    - ovviamente, i tag `init-param` vanno annidati all'internet del tag `servlet`, così da essere specifici per la servlet
  - invece, `ServletConfig` contiene inizializzazioni comuni a tutte le servlet (e pagine JSP) di una Web Application
    - come sopra, ma anziché `init-param` occorre usare `context-param`
    - e va messo annidato dentro `web-app`, quindi *fuori* dai tag `servlet` (in modo da essere comune a tutte le servet)
- slide 205: i contesti sono le varie directory dentro `webapps`
- slide 206: web application usato come sinonimo di contesto, nel senso che una web application completa (che può essere costituita da più servlet) viene messa tutta dentro lo stesso servlet context
  - proprio per permettere alle servlet di chiamarsi l'un l'altra, con i metodi descritti in queste slide
  - fa fede il `web.xml`, che per l'appunto è unico in un context...
- slide 206: il path di cui si parla è ovviamente quello definito nel `web.xml` (`url-pattern`)
- slide 210: ERRORE, non si possono mettere path relativi al server, solo relativi al context (devono *coincidere* con l'`url-pattern` di `web.xml`)
  - si possono anche usare relativi alla posizione attuale, ovvero senza slash iniziale, ma allora occorre invocare `getRequestDispatcher` su `request`

  - vedere
    `http://stackoverflow.com/questions/1411114/servletcontext-getrequestdispatcher-vs-servletrequest-getrequestdispatcher`

- da notare che `sendRedirect` fa cambiare l'indirizzo nella barra degli indirizzi del browser, queste altre redirezioni no