Programmazione per il Web
Note per la lezione del 20/04/2016

Igor Melatti

Pagine JSP e JavaBeans: Esempi

- Qui si parla delle slides 299–324
- `WEB-INF/classes/example/esempio1.java`: non è un Java Bean valido, la classe dev'essere `public`
- `beans.jsp`: non va bene, all'attributo class e' possibile passare solamente stringhe costanti
- `beans2.jsp`: adesso sì, ma non visualizza nulla, viene solo caricato il bean
  - vedere il codice della servlet generata: gli attributi `class` e `id` di `jsp:useBean` sono usati per dichiarare un'istanza del Bean, e poi c'è un pezzo di codice che serve a capire se l'oggetto esiste o no nello `scope` richiesto (non essendo precisato, è quello di default, ovvero di pagina)

    ```
    example.esempio2 bean2 = null;
    bean2 = (example.esempio2) _jspx_page_context.getAttribute("bean2",
            javax.servlet.jsp.PageContext.PAGE_SCOPE);
    if (bean2 == null){
      bean2 = new example.esempio2();
      _jspx_page_context.setAttribute("bean2", bean2,
            javax.servlet.jsp.PageContext.PAGE_SCOPE);
    }
    ```

  - se non esiste, lo crea e poi lo mette nello scope richiesto
  - notare che lo scope è, come al solito, un'hash table, nella quale l'id viene usato come chiave, mentre il valore è proprio l'oggetto bean
- `beans3.jsp`: sintatticamente, va bene anche così, ma è sconsigliato mettere con attributi `public`
- `beans4.jsp`: non visualizza nulla, ma è un bean completo, con tanto di getter e setter, anche se non in modalità cammello (c'è l'_ nel nome della variabile...)

- `beans4_ok_appl.jsp`: chiama anche il getter, visualizzando così il valore della proprietà
  - vedere il codice della servlet generata: lo `scope` richiesto stavolta è `application`, e la fase di test e creazione dell'istanza locale del bean viene protetto da un blocco `synchronized`:

```
example.esempio4 applicazione = null;
synchronized (application) {
  applicazione=(example.esempio4)_jspx_page_context.getAttribute("applicazione",
               javax.servlet.jsp.PageContext.APPLICATION_SCOPE);
  if (applicazione == null){
    applicazione = new example.esempio4();
    _jspx_page_context.setAttribute("applicazione", applicazione,
               javax.servlet.jsp.PageContext.APPLICATION_SCOPE);
  }
}
```

  - infatti, nello scope di applicazione si vuole che il bean sia lo stesso per tutte le risorse (JSP e/o servlet) che vi accedono all'interno della stessa web application
  - quindi, non si vuole che ad esempio 2 diversi thread contemporanei cerchino di creare ognuno la sua copia del bean (come potrebbe succedere, come corsa critica, senza il `synchronized`)
  - se la stessa cosa succede con lo scope di pagina o di richiesta non c'è problema: è giusto che pagine diverse vedano bean diversi

- `beans4_ok_appl_bis.jsp`: chiama anche il setter
  - sembra inutile: lo setta a 2, ma poi non lo visualizza? questo set viene perso
  - e invece no! lo scope è application!
  - si provi ora a ricaricare `beans4_ok_appl.jsp` oppure `beans4_ok_appl_bis.jsp`
  - il valore non è più il 10 di default, ma il 2 settato da `beans4_ok_appl_bis.jsp`
  - qualsiasi altra pagina JSP che usi il bean con id `applicazione` e scope `application` si troverà con 2 come valore di `proprieta_privata` (ovviamente, la classe dev'essere quella giusta)

- `beans4_ok_sess.jsp` e `beans4_ok_sess_bis.jsp`: stessa cosa, ma a livello di sessione, non di applicazione
  - stavolta, ogni utente diverso (o anche apparentemente lo stesso utente, se usa diversi browser o se cancella i cookie) che invochi la pagina JSP, avrà il valore di default 10

  - se lo stesso utente apre in sequenza `beans4_ok_sess_bis.jsp` e `beans4_ok_sess.jsp`, vedrà il valore cambiare da 10 a 2
  - per un altro utente che apra `beans4_ok_sess.jsp`, il valore sarà di nuovo 10
  - questa è appunto la differenza tra session e application: l'una distingue tra utenti diversi, l'altra li tratta tutti allo stesso modo
  - messa così, qui non c'è però modo di "difendersi" dalla disabilitazione dei cookie: niente `encodeURL`, se i cookie vengono disabilitati il meccanismo semplicemente non funziona
  - bisognerebbe invece fare dei link da una pagina all'altra, passandoli ad `encodeUrl` (se invece l'utente scrive lui stesso l'URL come nell'esempio di sopra, non c'è salvezza)
  - vedere più sotto per l'interazione con la servlet
  - nella servlet generata, c'è di nuovo un blocco `synchronized` (ma questa volta sull'oggetto `session`) nella creazione dell'oggetto bean, dato che comunque si tratta di un oggetto che può essere condiviso (non per tutti come per lo scope application, ma solo per richieste che vengono da uno stesso client)

- `beans5.jsp`: ok avere, oltre al costruttore con 0 argomenti, anche altri costruttori
- `beans6.jsp`: ok anche se gli attributi cominciano con la lettera maiuscola: quello che veramente importa è come sono scritti i metodi getter e setter
- `beans7.jsp`: ovviamente, se non viene definito il metodo setter ma si usa comunque l'azione `jsp:setProperty`, il JSP container solleva un'eccezione
- `beans8.jsp`, ok anche se si usano vettori
  - lo scope è request (così come anche per `beans4_ok.jsp`), perché si vuole usare queste pagine JSP come destinazione di un forward
  - differentemente dallo scope di pagina, nello scope di richiesta l'oggetto permane durante i forward (proprio perché `useBean` si va a prendere l'oggetto dalla richiesta...)
- servlet `passBean`: comunicazione tra servlet e pagina JSP
  - è il metodo più comune: alle servlet viene demandata la computazione (sono classi Java, e ne possono facilmente usare altre...)
  - il risultato di tale computazione viene poi mandato ad un'opportuna pagina JSP per la visualizzazione in HTML (più facile farla con le JSP che con le servlet)

- la servlet `passBean` (invocabile sia come `/20150422/passBean.html` che come come `/20150422/altro/passBean.html`, vedere `web.xml`) può comunicare con alcune delle pagine JSP viste sopra, a seconda di quale parametro viene specificato nella query string (non serve passare anche un valore)
  * `/20150422/passBean.html?request` crea un bean in cui la proprietà ha valore 5, usando un costruttore con 1 argomento (possibile solo nella servlet, nella pagina JSP è per forza con 0 argomenti...), e lo passa a `beans4_ok.jsp`. Notare che il passaggio avviene tramite forward sulla request, e infatti la pagina JSP ha lo scope `request`
  * `/20150422/passBean.html?application` lavora a livello di applicazione, quindi il bean (corrispondente alla chiave "applicazione") potrebbe anche già essere stato creato (se si è già visualizzato `beans4_ok_appl.jsp` o `beans4_ok_appl.jsp`). Se già c'è, la servlet usa getter e setter per incrementare la proprietà di 1, altrimenti lo crea con valore 50. Non è necessario fare una forward: se si prova adesso a visualizzare `beans4_ok_appl.jsp` o `beans4_ok_appl.jsp` (che prendono il bean con scope application e vhiace "applicazione"), il valore sarà quello settato in questa servlet
  * `/20150422/passBean.html?session` funziona come sopra, ma a livello di sessione: la comunicazione è quindi con `beans4_ok_sess.jsp` o `beans4_ok_sess.jsp`, e in caso di creazione la proprietà viene settata a 30. Trattandosi di sessione, utenti diversi vedranno valori diversi, mentre nel caso di cui sopra (applicazione) utenti diversi vedono la stessa cosa
  * `/20150422/passBean.html` (o anche come?) passa un intero vettore di informazioni a `beans8.jsp`. In questo modo, una servlet può inviare ad una pagina JSP informazioni complicate a piacimento: basta metterle in un bean