Programmazione per il Web
Riassunto della lezione del 29/02/2016

Igor Melatti

Esercizi: HTML Statico

- `pag036_form_text_buttons.html`:
  - come mai non si vede nulla dopo il titolo?
  - prima di provare a cliccare: i due pulsanti funzioneranno?
- `pag044_complex_form.html`:
  - scrivere "Enter your comments here" dapprima all'inizio, poi in mezzo alla textarea
  - mettere le 2 scelte multiple una al fianco dell'altra, ciascuna in verticale anziché in orizzontale
  - mettere uno al fianco dell'altro anche commenti ed email
  - mettere l'email visibile
  - mettere i due pulsanti all'estremità destra, uno sotto l'altro

Prime specifiche del progetto

- Si vuole costruire un sito di commercio elettronico
- Funzionalmente, ci devono essere un certo numero di pagine; per ora ci si limiterà alle seguenti:
  - pagina iniziale che mostra le sezioni di prodotti in vendita (per adesso si supponga che siano: libri, oggetti per la casa, oggetti per l'igiene personale)
  - la pagina iniziale deve anche permettere di effettuare un login e di registrarsi
  - i prodotti in vendita possono essere mostrati anche senza aver fatto login

  - ma se si è fatto login, occorre mostrare, da qualche parte nella pagina, il carrello attuale (con una visione ridotta degli elementi già acquistati)
  - ogni prodotto ha un pulsante per l'acquisto (dipendentemente dal fatto che sia al momento disponibile oppure no); ma se si cerca di acquistare senza aver fatto login, si viene rimandati ad una pagina di solo login
  - cliccando su "acquista", si va in una pagina che mostra tutto il contenuto del carrello per esteso, con la somma finale da pagare, e richiede i dati di una carta di credito

- Iniziare quindi a costruire le pagine statiche, sulle quali si baserà il progetto: sono quindi le seguenti pagine
  - pagina iniziale
  - pagina di solo login
  - pagina con i libri (metterne 3-4 d'esempio) con carrello
  - pagina con i libri (metterne 3-4 d'esempio) senza carrello
  - pagina con gli oggetti per la casa (metterne 3-4 d'esempio) con carrello
  - pagina con gli oggetti per la casa (metterne 3-4 d'esempio) senza carrello
  - pagina con i prodotti per l'igiene personale (metterne 3-4 d'esempio) con carrello
  - pagina con i prodotti per l'igiene personale (metterne 3-4 d'esempio) senza carrello
  - pagina che mostra il carrello
    * assumere che il carrello contenga due oggetti per ogni categoria

- Verrà sempre controllato che le pagine siano conformi all'XHTML, tramite lo script Python `check_html.py` (richiede il package `lxml`)

Verso le Pagine Web Dinamiche

- Slides 58–68: riassunto dal corso di Reti
  - in particolare, il fatto che sia senza stato costringe ad alcune acrobazie i programmatori Web
  - infatti, memorizzare uno stato è spesso indispensabile
    * ad esempio, per vedere il proprio conto corrente occorre prima essere autenticati: quindi ci sono almeno due stati in cui la pagina web del conto viene visualizzata, autenticato e non autenticato, dipendentemente da visite a pagine Web fatte precedentemente

  - nel seguito del corso verranno presentati diversi modi di gestire lo stato, aggirando il problema

- Slide 60: un possibile metodo (un po' macchinoso) per gestire uno stato anche se l'HTTP non lo supporta
  - il Web Server risponde a delle richieste fatte tramite il metodo "GET" da un form HTML
  - come si può vedere, con il metodo "GET" l'URL (qui mostrato senza l'indirizzo del server) contiene le informazioni che vengono mandate (le informazioni cominciano dopo il punto interrogativo)
  - comunque, il trucco funzionerebbe anche con il metodo "POST"
  - assunzione: sul Web server c'è una qualche applicazione che si chiama `miaservlet` e che è capace di accorgersi di com'è scritto l'URL in arrivo, e risponde con una pagina HTML di contenuto diverso in funzione di tale URL
  - ad esempio, tra la prima e la seconda freccia blu la differenza è il `Cognome=Rossi`
  - a seconda di quali informazioni sono già state inviate (quindi, a seconda dello "stato"), il form ha apparenze diverse
  - il funzionamento di `miaservlet` è il seguente:
    1. si visita una pagina HTML che ha un form dove si inserisce solamente il nome
    2. dopo aver inserito il nome e cliccato sul pulsante di sottomissione, si manda il nome a `miaservlet`, come mostrato nella prima freccia blu in alto nella slide
    3. `miaservlet` risponde (prima freccia rossa) con una nuova pagina HTML, molto simile alla precedente, ma in cui il campo del nome è hidden e fissato a ciò che è stato immesso al passo precedente (nell'esempio, "Giovanni"); c'è inoltre un campo in più (l'unico visibile), per il cognome
    4. dopo aver inserito il cognome e cliccato sul pulsante di sottomissione, si manda il cognome a `miaservlet`, come mostrato nella seconda freccia blu
       * c'è anche il nome, perché è quello che succede coi campi hidden: non vengono visualizzati dal browser, ma il loro contenuto (fissato) viene comunque mandato insieme ai campi visibili
    5. `miaservlet` risponde (seconda freccia rossa) con una nuova pagina HTML, molto simile alla precedente, ma in cui i campi del nome e del cognome sono hidden e fissati a ciò che è stato immesso al passo precedente (nell'esempio, "Giovanni" e "Rossi"); c'è inoltre un campo in più (l'unico visibile), per l'indirizzo

6. e così via

- quindi, nonostante l'HTML non abbia uno stato, si è riuscito ad introdurne uno
- ovvero, cosa viene visualizzato in una particolare pagina HTML dipende da cosa si è visitato (o inviato) prima
- nota: `miaservlet` non può memorizzare le informazioni al suo interno man mano che gli arrivano!
- questo perché `miaservlet` deve poter essere invocata da svariati utenti contemporaneamente, anche provenienti dallo stesso host
- essenzialmente, per risolvere questo problema, le informazioni sullo stato vengono scritte nella pagina HTML
- come si vedrà, servlet e pagine JSP hanno altri modi per memorizzare le informazioni di stato (cookies e sessioni), ma la cosa interessante di questo metodo è il fatto che sia basato direttamente sull'HTML "puro"
- ultima nota: mettere il campo hidden permette di far sì che l'utente non possa più modificare quanto immesso in precedenza, e quindi che non possa modificare lo "stato" con cui è arrivato alla pagina
- in realtà, un utente smaliziato potrebbe cambiare il sorgente della pagina HTML...

- Slide 64
  - esempio per server software: Apache, IIS
  - content-type: non è detto che sia un HTML, potrebbe essere un'immagine, o una pagina HTML compressa, o testo formattato (es.: JSON)
- Slide 68: la distinzione funzionale è ritornata in auge nelle applicazioni RESTful
- Slide 69: pagine "dinamiche" nel senso che il loro contenuto varia a seconda delle informazioni ulteriori che vengono inviate con le richieste GET o POST
  - o anche a seconda dello "stato" (da introdurre con metodi vari)
- Slide 70: le tecnologie lato server elencate qui fanno tutte riferimento al Java Enterprise Edition (tipicamente indicata come J2EE)
  - si tratta di una suite di programmazione della Oracle che permette di costruire applicazioni professionali per le imprese
  - Java Transaction Management, vedere qui: `http://en.wikipedia.org/wiki/Java_Transaction_API`

  - questo corso è principalmente basato su questa suite
  - Java Server Pages fa pure parte di questa suite, e permette di mischiare HTML statico e dinamico (in questo è simile al PHP e all'ASP)

- Slide 72: CGI sta per "Common Gateway Interface"; si tratta di eseguire un programma (C/C++ o Perl), il cui output è la pagina HTML richiesta (quindi occorre che l'output del programma rispetti la sintassi dell'HTML...)

- il programma CGI può accedere opportunamente a delle variabili che descrivono l'input (ad esempio, i campi di un form inviati come richiesta)

- Slide 73: sono comunque soluzioni molto usate, soprattutto nei casi in cui non si ha la necessità di usare le API per le imprese che sono offerte da J2EE

- Slide 74

  - se non viene usato un form, allora ogni richiesta di pagina HTML è un GET
  - pagina generata dinamicamente: nelle CGI la pagina HTML è l'output di un programma, qui (semplificando, lo si vedrà meglio in seguito) è l'output di una funzione di una classe Java
  - questo vuol dire scrivere una funzione piena di `println`, in modo che il risultato sia un documento HTML completo o comunque correttamente interpretabile da un browser (nel libro di testo questa soluzione è indicata come "pure Servlet")
  - J2EE offre un'alternativa nel caso in cui si debba modificare una piccola parte di una pagina HTML per lo più statica: Java Server Pages (JSP)
  - in modo simile al PHP e all'ASP, JSP permette di scrivere tranquillamente in HTML, e poi ci sono dei tag speciali al posto dei quali il server esegue un metodo di una servlet, il cui risultato viene messo al posto dei soli tag speciali (esempio a pag. 10 del libro di testo)

- Slide 75

  - servlet container = servlet engine, non è altro che l'ambiente Java che gestisce le varie servlet che vengono definite su un determinato server Web
  - le servlet sono organizzate con un preciso ciclo di vita (definito dal fatto che le servlet, come si vedrà, ereditano tutte da una ben precisa classe del J2EE)
  - il programmatore di servlet può modificare solo alcune parti ben definite di questo ciclo di vita (ridefinendo alcuni metodi di alcune classi)

- questo ciclo di vita “esterno”, sul quale il programmatore di servlet non ha controllo, è implementato in modo tale che, alla prima GET o POST di una servlet, viene caricato in memoria un nuovo processo persistente
- creato tale processo, ogni invocazione della servlet (compresa la prima) sono solo thread di questo processo
- quando un’invocazione della servlet viene servita completamente, “muore” solo il thread, ma non il processo
- ad esempio, si può far sì che, se occorre comunicare con un database, si apra la connessione una volta sola e la si sfrutti per tutte le richieste successive
- dato che i thread sono più leggeri dei processi (non hanno la parte di memoria, che è condivisa, ma solo lo stack e il program counter), questo risulta in una modalità di esecuzione più efficiente del CGI
- c’è anche un retro della medaglia, non menzionato in queste slides e detto tra le righe del libro di testo: se ci sono tante servlet che vengono invocate, restano tutte in memoria (non muoiono mai) e la memoria stessa può finire!
- questo problema può essere risolto in due modi:
  * l’amministratore del Web server si accorge del degrado di prestazione e uccide le servlet meno importanti (per lui...)
  * J2EE si accorge che una servlet è idle (ovvero, in memoria ma senza richieste) da troppo tempo, e allora la uccide
  * garbage collection di J2EE: vengono opportunamente selezionate alcune servlet, che vengono uccise
- in tutti i casi di cui sopra, l’utente può definire un metodo da eseguire prima della rimozione di una servlet