Programmazione per il Web
Riassunto della lezione del 02/03/2016

Igor Melatti

Verso le Pagine Web Dinamiche

- Slide 77: architettura J2EE per le servlet
  - si potrebbe anche ereditare direttamente `GenericServlet`, ma `HttpServlet` è già pronta per HTTP
- Slide 79: javax è il prefisso dei package di J2EE
- Slide 80:
  - in realtà, è sempre il servlet container a chiamare i metodi scritti qui (o direttamente, con `init` e `service`, o indirettamente con `destroy`, che viene invocato automaticamente quando il container distrugge una servlet)
  - una volta sola = non per ogni invocazione della servlet (dopo `destroy` ci sarà nuovamente la `init`)
  - `init` e `destroy` le scrive il programmatore web (ovviamente, la signature è fissata nell'interfaccia: si può scrivere solo il corpo delle funzioni)
  - `service` va scritta solo se si eredita da `GenericServlet`, altrimenti, se si eredita da `HttpServlet`, viene lasciata così com'è, e si scrive almeno uno dei due metodi chiamati da `service`: `doGet` e `doPost`
- Slide 83: in realtà il servlet engine non chiama la `doXxx`, bensì la `service`, che poi smista al corretto metodo (a meno che non venga ridefinita...)
- Slide 88
  - `getParameterNames` va bene anche per le GET, non solo per le POST
  - `getParameterValues`: ad esempio per i checkbox, o per i casi in cui più campi di un form vengono chiamati con lo stesso nome (vedere es. pagina 73 del libro)
  - `getCookies` e `getSession`: diversi modi di avere lo "stato" sulle pagine HTML; questo argomento verrà ripreso più avanti

- Slide 90
  - `getOutputStream`: ad esempio, la pagina HTML potrebbe essere compressa con gzip se è troppo grande; oppure potrebbe consistere in una singola immagine
  - non c'è la gestione delle sessioni, perché si fa tutta agendo sulla classe ritornata da `getSession`
  - `getWriter`: per mandare l'HTML "normale"
- Slide 93-94: in questo corso verrà usato TomCat (letteralm.: gatto maschio)
  - inizialmente era stato sviluppato all'interno del progetto Jakarta
  - Jakarta è stato sospeso nel 2011, e TomCat è ora distribuito da Apache, che era il promotore di Jakarta
  - esistono molte altre soluzioni per avere servlet e JSP, addirittura Apache ne fornisce due
- Slides 94-95: vale per Windows; per Linux Ubuntu dovrebbero bastare i seguenti comandi:

  ```
  sudo aptitude install apache2
  sudo aptitude install default-jdk
  sudo aptitude install tomcat7
  ```

  - il primo installa il Web server "normale" (HTTP con pagine statiche)
  - il secondo installa Java con il compilatore (se non già presente)
  - il terzo aggiunge ad Apache la gestione di servlet e JSP
  - è possibile soffrire e compilare tutto da sorgenti, vedere ad esempio qui: `http://www.sitepoint.com/jsp-quick-start-guide-linux/`
  - alla fine, la pagina di testing è un po' diversa: `http://localhost:8080`
  - la directory dove mettere i files che definiscono una servlet è la seguente: `/var/lib/tomcat7/webapps/`
  - si potrebbe anche configurare per metterla altrove, ma è più comodo lasciarla lì e cambiargli l'owner:

    ```
    cd /var/lib/tomcat7/
    sudo chown -R vostronomeutente webapps
    ```

  - da qui si può creare una struttura di directory del tipo quella delineata nelle slides 101 e 108

– la “directory radice” di cui si parla alla slide 101 è una sottodirectory di `webapps`, da creare
– se ad esempio si crea una directory `vediamo` con dentro un file `esempio.html`, allora basterà aprire un browser su `localhost:8080/vediamo/esempio.html` per vedere il risultato
– se il file si chiama `index.html`, basta andare su `localhost:8080/vediamo/`
– se la directory si chiama `ROOT`, basta scrivere `localhost:8080/` (infatti, la pagina di default di TomCat è proprio lì...)
– la directory `ROOT` esiste già