Tecnologie di Sviluppo per il Web

Applicazioni Web J2EE:
Java Servlet
Parte a

versione 3.1

G. Mecca – mecca@unibas.it – Università della Basilicata

Sommario

- Preliminari
- Ciclo di Vita
  - ⇨Un Esempio

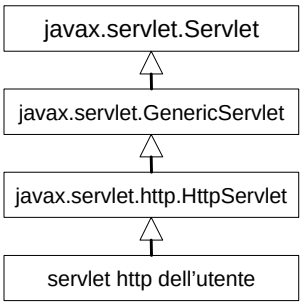
Preliminari

- Servlet
  - ⇨classe Java orientata alla comunicazione client-server
  - ⇨riceve messaggi di richiesta, produce messaggi di risposta
  - ⇨viene eseguita dal contenitore secondo un opportuno “ciclo di vita”
  - ⇨le API relative sono definite nell’ambito del JCP (attualmente alle versione 2.4)

Preliminari

- Due tipologie di servlet
  - ⇨servlet generici (es: servlet per implementare un servizio di posta elettronica) e servlet HTTP
- Due package fondamentali
  - ⇨javax.servlet.* per i servlet generici
  - ⇨javax.servlet.http.* per i servlet HTTP
- Nota: nel seguito
  - ⇨ci concentreremo solo sui servlet HTTP

Preliminari



- Gerarchia
  - ⇨interfaccia javax.servlet.Servlet
  - ⇨servlet generici: estendono la classe astratta javax.servlet.GenericServlet
  - ⇨servlet http: estendono la classe astratta javax.servlet.http.HttpServlet

Preliminari

- Struttura di base di un servlet
  - ⇨la parte fondamentale del servlet sono i metodi di servizio
  - ⇨metodi invocati dal contenitore per gestire le richieste HTTP indirizzate al servlet
  - ⇨sono pensati in modo da gestire in modo differenziato richieste effettuate con metodi diversi (POST e GET)

Preliminari

- Metodo doGet()
  - ⇨operazioni da effettuare per rispondere a richieste di tipo GET
- Metodo doPost()
  - ⇨operazioni da effettuare per rispondere a richieste di tipo POST
- Le implementazioni ereditate
  - ⇨sono entrambe vuote
  - ⇨è possibile sovrascriverli entrambi o solo uno

Struttura di Base di Un Servlet Http

```
package nomepackage;

public class NomeServlet extends javax.servlet.http.HttpServlet {

  public void doGet(javax.servlet.http.HttpServletRequest request,
                    javax.servlet.http.HttpServletResponse response)
              throws java.io.IOException,
                    javax.servlet.ServletException {
   // codice del metodo doGet
   // altri servlet implementano doPost e non doGet
   // oppure entrambi
   ...
  }
}
```

Preliminari

- Attenzione
  - per la compilazione del servlet i due package javax.servlet e javax.servlet.http devono essere disponibili nel classpath
  - forniti da Tomcat come servlet-api.jar nella cartella %TOMCAT_HOME%\common\lib
  - per poter compilare i servlet è necessario aggiungere servlet-api.jar al classpath

Ciclo di Vita

- Istanze del servlet
  - non vengono create direttamente dal programmatore ma dal contenitore
  - secondo un opportuno ciclo di vita
- Attenzione
  - il programmatore non dispone di riferimenti ai servlet e non ne può chiamare i metodi
  - si tratta quindi di metodi per la gestione di eventi, dove gli eventi sono le richieste http

Ciclo di Vita

- Il ciclo di vita di un oggetto servlet
  - ⇨prevede tre fasi
  - ⇨inizializzazione: il contenitore crea una o più istanze del servlet
  - ⇨servizio: le istanze vengono utilizzate per gestire le richieste
  - ⇨distruzione: le istanze vengono rimosse dal contenitore

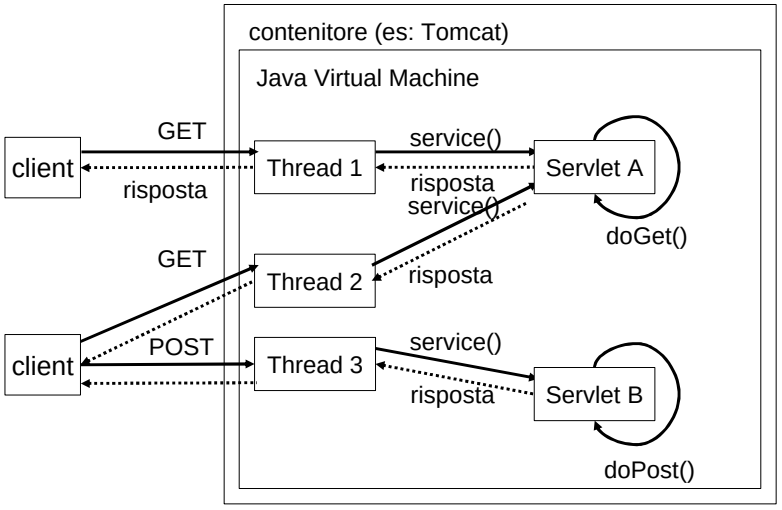
Ciclo di Vita

- Inizializzazione
  - ⇨il contenitore tipicamente crea un'unica istanza del servlet all'avvio dell'applicazione
  - ⇨può essere utilizzato un "pool" di istanze
  - ⇨viene eseguito il metodo void init() ereditato da javax.servlet.GenericServlet
  - ⇨è possibile sovrascrivere il metodo init() per effettuare operazioni all'inizializzazione (es: inizializzare contatori)

Ciclo di Vita

- Servizio
  - ⇨da quel momento in poi il servlet è disponibile per le richieste
  - ⇨per ogni richiesta all'URI del servlet il contenitore crea un nuovo thread
  - ⇨il thread chiama il metodo service() ereditato da java.servlet.GenericServlet
  - ⇨il metodo service() chiama doGet() o doPost() passando richiesta e risposta

Ciclo di Vita



Ciclo di Vita

- Servizio
  - ⇨lo sviluppatore sovrascrive doGet() e/o doPost() (non service())
  - ⇨possono essere sovrascritti indipendentemente
  - ⇨oppure è possibile fare in modo che effettuino le stesse operazioni chiamando l'uno dall'altro (utile per consentire di chiamare metodo nelle form)

Ciclo di Vita

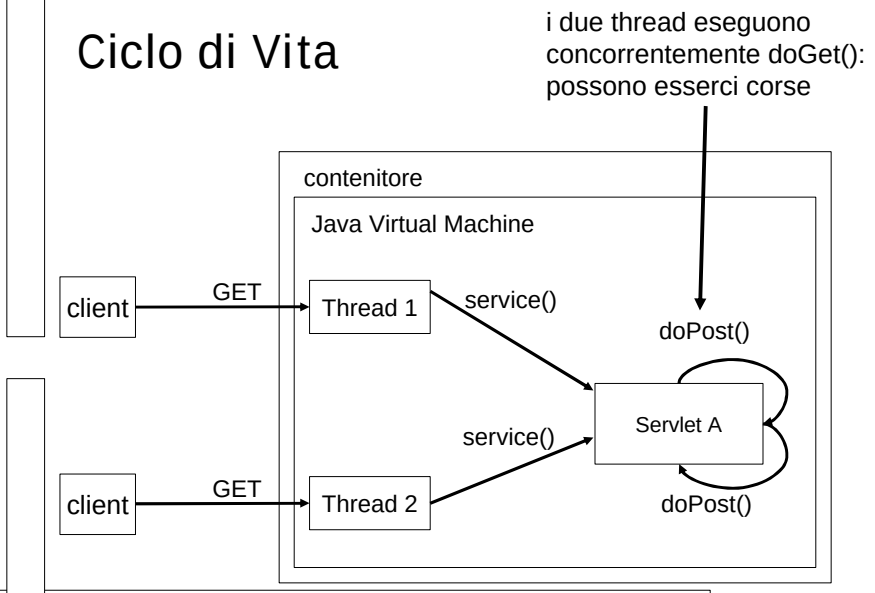
```
public class NomeServlet extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
              throws IOException, ServletException {
   // codice del metodo doGet
   ...
  }
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
              throws IOException, ServletException {
    doGet(request, response);
  }
}
```

Ciclo di Vita

- Nota
  - ⇨l'istanza del servlet è condivisa tra i thread, che possono chiamare concorrentemente i metodi di servizio
  - ⇨è necessario gestire la sincronizzazione
- Dati a cui sincronizzare l'accesso
  - ⇨oggetti dell'applicazione condivisi tra i servlet
  - ⇨proprietà statiche del servlet (condivisi tra le istanze del servlet)

Ciclo di Vita



Ciclo di Vita

- Distruzione
  - ⇨quando è necessario, il contenitore rimuove l'istanza del servlet; esempio: shutdown oppure richiesta di ricaricare o rimuovere l'applicazione
  - ⇨viene chiamato il metodo void destroy() ereditato da javax.servlet.GenericServlet
  - ⇨è possibile sovrascriverlo per per deallocare risorse allocate con init() oppure per salvare dati nello strato di persistenza (es: contatori)

Ciclo di Vita

- Vantaggi collegati al ciclo di vita
  - ⇨viene usata un'unica macchina virtuale: i servlet possono comunicare tra di loro
  - ⇨è possibile creare pool di servlet: minore sovraccarico
  - ⇨gestione delle richiesta con thread: efficienti perché richiedono meno risorse di un processo per l'attivazione
  - ⇨però si deve gestire la sincronizzazione

Ciclo di Vita

- Attenzione
  - ⇨è necessario fare molta attenzione nella programmazione
- Infatti
  - ⇨i metodi dei servlet vengono eseguiti in un ambiente multithread
  - ⇨il programmatore non ha completo controllo sul ciclo di vita

Ciclo di Vita

- Manca il controllo sul ciclo di vita
  - ⇨le procedure di inizializzazione possono variare e non sono standard (es: il contenitore può creare un istanza unica oppure un pool; può farlo in modo pigro o avido)
  - ⇨non è garantito che le procedure di distruzione siano sempre eseguite (es: crash dell'applicazione o del contenitore)

Un Esempio

- Un esempio
  - la gestione del numero di giocatori di Indovina il Numero
  - devo utilizzare un dato visibile in tutti i servlet, indipendentemente dalla sessione
- Idea
  - posso utilizzare il “contesto dell’applicazione”, il contenitore di riferimenti visibili a tutti i servlet

Un Esempio

- Una possibile strategia
  - all’avvio dell’applicazione inizializzo il valore a 0 e lo salvo (sotto forma di Integer) nel contesto dell’applicazione
  - successivamente lo incremento ogni volta che comincia una partita
  - e lo decremento ogni volta che una partita finisce

Un Esempio

- Inizializzazione del valore
  - ⇨è necessario farlo una volta per tutte al deployment dell'applicazione
  - ⇨posso utilizzare il metodo init() di ServletTentativo, che viene eseguito solo all'inizializzazione del servlet
- Esempio: in init() di ServletTentativo
  - ⇨ServletContext application = getServletContext();
    application.setAttribute("giocatori", new Integer(0));

Un Esempio

- Aggiornamento del valore
  - ⇨nel metodo doPost() di ServletTentativo, se comincia una nuova partita
  - ⇨ Integer giocatori = (Integer)application.getAttribute("giocatori");
  - ⇨ application.setAttribute("giocatori",
    new Integer(giocatori.intValue() + 1));
  - ⇨quando finisce una partita
  - ⇨ Integer giocatori = (Integer)application.getAttribute("giocatori");
  - ⇨ application.setAttribute("giocatori",
    new Integer(giocatori.intValue() - 1));

Un Esempio

- I Problema: sincronizzazione
  - ⇨giocatori diversi possono cominciare a giocare contemporaneamente
  - ⇨oppure un giocatore può cominciare a giocare mentre un altro interrompe
  - ⇨in entrambi questi casi thread diversi eseguono doPost() di ServletTentativo e tentano di aggiornare l'oggetto Integer

Un Esempio

- Di conseguenza
  - ⇨è necessario immergere le operazioni in una sezione critica
  - ⇨sincronizzando i servlet utilizzando il contesto dell'applicazione

```
⇨ synchronized(application) {
      Integer giocatori =
            (Integer)application.getAttribute("giocatori");
      application.setAttribute("giocatori",
                        new Integer(giocatori.intValue() + 1));
  }
```

Un Esempio

- Il Problema: inizializzazione
  - ⇨è possibile che il contenitore crei un pool di istanze di ServletTentativo
- In questo caso è possibile che
  - ⇨il contenitore crei inizialmente l'istanza n. 1
  - ⇨l'istanze n.1 inizializzi giocatori a 0
  - ⇨comincino delle partite e giocatori aumenti
  - ⇨il contenitore crei un'istanza n. 2
  - ⇨l'istanza n. 2 re-inizializzi giocatori a 0

Un Esempio

>> ServletTentativo.java

- Di conseguenza
  - ⇨nel metodo init(), prima di inizializzare l'attributo "giocatori" devo verificare che non sia già stato inizializzato da un'altra istanza dello stesso servlet

Riassumendo

- Preliminari
- Ciclo di Vita
  - ⇨Un Esempio

Termini della Licenza