Esercitazione di Sistemi Distribuiti e Pervasivi

Suggerimenti di Sviluppo e Progettazione I

Gabriele Civitarese
gabriele.civitarese@unimi.it

EveryWare Lab
Università degli Studi di Milano
-
Docente: Claudio Bettini

Slide tratte da versioni precedenti di Letizia Bertolaja e Dario Freni

Laboratorio

- 4 lezioni dedicate all'assistenza allo svolgimento del progetto
  - Voi potete sfruttare le ore di lezione per sviluppare il progetto
  - Potete nel frattempo chiarire dubbi o chiedere consigli
  - Nel caso in cui abbiate concluso il progetto in anticipo, potete sfruttare l'ultima lezione per discuterlo
- Le lezioni partiranno ufficialmente alle 09:00 (puntuali!), con la possibilità di rimanere fino alle 11:30 (o leggermente oltre)
- In realtà io saró in aula dalle 08:30 circa ed è possibile entrare
- Prima parte di "lezione frontale"
- È possibile lavorare sul proprio portatile
- L'importante è che lavoriate individualmente
  - Questo non vuol dire che non possiate parlare tra di voi...
  - Due progetti copiati SONO EVIDENTI

Statistiche

Giusto un po' di dati degli anni scorsi...

- Circa il 67% degli studenti frequentanti ha passato il progetto entro l'appello di febbraio
- Di questi, circa il 70% ha verbalizzato entro luglio
- Circa l'80% dei voti assegnati ai progetti è stato $>= 27$
- Solo il 5% circa degli studenti ha preso un voto < 23
- Spero che queste statistiche vi abbiano rincuorato sulla fattibilità dell'esame :)

Disclaimer

- Il contenuto di questi lucidi è da considerarsi puramente indicativo
  - Verranno forniti dei suggerimenti di sviluppo e di progettazione
  - Ogni studente può effettuare scelte differenti
- Le direttive su come svolgere il progetto sono riportate nel testo del progetto presente sul sito del corso.

Identificazione delle applicazioni

Il sistema richiede lo sviluppo di quattro applicazioni diverse:

- *Simulatore di sensori*
- *Nodo edge*
- *Server cloud*
- *Analista*

Un consiglio..

Flusso di sviluppo consigliato

- Primo step (lezione di oggi): sviluppo del server
  - Progettazione server (risorse e metodi)
  - Sviluppo servizi REST
  - Analisi e risoluzione di problemi di sincronizzazione
  - Testing con tool dedicati
  - Sviluppo client *Analista*
- Secondo step (prossima lezione): sviluppo della rete di *nodi edge*
  - Progettazione architettura e protocolli per rete di *nodi edge*
  - Sviluppo della rete dinamica di *nodi edge* (con elezione coordinatore)
  - Sviluppo di *Simulatore di sensori*
  - Estensione della rete di *nodi edge* per l'analisi dei dati di sensori
- È assolutamente necessario considerare attentamente tutti i problemi di sincronizzazione *interna* e sincronizzazione *distribuita*

Server cloud

- Il *server cloud* è un'unica applicazione che ha il compito di:
  - Gestire la rappresentazione della città con inserimento e rimozione di *nodi edge*
  - Ricevere le statistiche sulla città dai *nodi edge*
  - Permettere agli analisti di analizzare le statistiche
  - Fornire ad un sensore che fa richiesta il *nodo edge* più vicino (se disponibile)
- Questi servizi devono esere erogati tramite architettura REST
- Sono quindi necessari meccanismi di sincronizzazione per gestire l'accesso alle risorse condivise

Risorse e operazioni

Quali risorse?

- Il primo passo è sicuramente quello di individuare:
  - Le risorse da modellare
  - Le operazioni CRUD effettuabili sulle risorse e il mapping con HTTP
- La griglia della città può essere considerata una risorsa?
- Come modellare le statistiche?

Esempio mapping CRUD - HTTP

- GET $\rightarrow$ ottenere i *nodi edge* e le loro posizioni sulla città
- POST $\rightarrow$ inserire un nuovo *nodo edge* nella griglia
- PUT $\rightarrow$ modificare le informazioni relative ad un *nodo edge* (ci serve?)
- DELETE $\rightarrow$ eliminare un *nodo edge* dalla griglia

Gestione *nodi edge*

Inserimento/uscita nodi edge

- Quando un *nodo edge* fa richiesta di entrare, il *server cloud*:
  - Cerca di inserirlo nella griglia della città
  - Se esiste già un *nodo edge* con lo stesso identificatore viene restituito un messaggio di errore
  - Anche se esiste un *nodo edge* troppo vicino rispetto alle coordinate ricevute viene restituito un messaggio di errore
  - Se invece è tutto regolare viene aggiunto nella griglia e viene restituita la lista di *nodi edge* presenti nella città
- L'uscita di un *nodo edge* avviene semplicemente rimuovendolo dalla griglia
- Che tipo di sincronizzazione è necessaria?

Gestione statistiche

- Lato *nodo edge*:
  - Il coordinatore invia periodicamente le statistiche globali della città insieme alle statistiche locali più recenti di ogni altro *nodo edge*.
  - Queste misurazioni vanno salvate internamente in una struttura dati per permettere successivamente l'analisi
- Lato analista:
  - Sono necessarie delle interfacce per analizzare i dati come da specifiche
- Cercare di realizzare la sincronizzazione più a grana fine possibile:
  - Ha senso bloccare tutto il server mentre vengono calcolate le statistiche?
  - Ha senso bloccare tutta la struttura dati se vogliamo analizzare le statistiche di uno specifico *nodo edge*?

Interfaccia per i sensori

- Un sensore può interrogare il *server cloud* per capire quale sia il *nodo edge* più vicino
- Per questo scopo, passa come parametro le proprie coordinate
- Che problemi di sincronizzazione abbiamo?

Reminder

Ricordatevi che...

- Ogni classe che gestisce una risorsa (annotata con *@Path*) viene istanziata ogni volta che viene effettuata una singola richiesta HTTP
  - È quindi necessario gestire adeguatamente la memoria condivisa
- Viene automaticamente gestito il multi-threading: chiamate concorrenti eseguono in concorrenza il codice di diverse istanze della classe che gestisce la risorsa
  - Bisogna fare attenzione ai problemi di concorrenza

Possibili problemi di sincronizzazione

- All'interno dello sviluppo del server sono presenti svariati problemi di sincronizzazione:
  - La griglia della città viene modificata e letta in maniera concorrente
  - Le statistiche della città vengono aggiunte e lette in maniera concorrente
- Gestite la sincronizzazione in maniera snella
- Utilizzare *synchronized* alla rinfusa non è una best practice

Come testare il server cloud?

- Il primo passo è di testare ogni singolo metodo con tool comodi
  - Ad esempio *Advanced REST Client*
- Testate i problemi di concorrenza con le *sleep()*
- Per automatizzare test più complicati, vi conviene scrivere codice Java
- Il consiglio è quello di utilizzare le API Client di Jersey
  - Il vantaggio è anche quello di sfruttare nuovamente marshalling e unmarshalling offerto da JAXB

Nella prossima puntata...

Spoiler

- Nella prossima lezione vedremo invece lo sviluppo della rete di *nodi edge*
  - Progettazione dell'architettura
  - Elezione del coordinatore
  - La comunicazione tra processi
  - ...

Buon lavoro!