Sujet de TP 1

Servlets asynchrones

Chouki Tibermacine - `chouki.tibermacine@umontpellier.fr`

Mise en place de l'environnement de travail :

- Installer IntelliJ Ultimate edition. Vous avez la possibilité d'avoir une licence gratuite (pas uniquement la licence d'essai d'un mois) si vous entrez votre adresse institutionnelle (@etu.umontpellier.fr)
- Créer un projet Gradle pour une app Java et Web
- Ajouter dans le build le plugin suivant :
  `id "org.gretty" version "3.0.3"` (prendre la dernière version, si c'est différent)
  Celui-ci nous permettra de démarrer un serveur Web Tomcat ou Jetty et déployer nos applications Web dedans
  * Pour ce TP, il nous faudra Tomcat, qui est l'implémentation de référence (implém. complète) de l'API `javax.servlet`. Du coup, le support des servlets asynchrones y est.
- Ajouter `jcenter()` dans les repositories si ce n'est pas déjà présent

Exercice.

Énoncé

Écrire une application Web avec une servlet qui exécute une tâche très complexe et trop longue : `Thread.sleep(10000); // ;-)`
Après l'exécution de la tâche, la servlet renvoie un message de bienvenue. Cette application fournit une page Web sur laquelle un bouton est affiché. Lorsqu'on clique sur le bouton, la tâche est exécutée (par envoi de requête AJAX).

Étapes à suivre

Pour commencer, écrire une servlet synchrone qui renvoie le message de bienvenue et tester en cliquant plusieurs fois sur le bouton.

Installer et démarrer VisualVM pour voir le nombre de threads qui sont créés par Tomcat.

Ensuite, écrire une servlet asynchrone et tester aussi.

Enfin, étendre la version avec une servlet asynchrone en vous appuyant sur un pool de 10 threads, créés à l'initialisation du contexte de la servlet dans un listener :

```
public class ClientTransfer implements ServletContextListener {
  private static final int CLIENT_THREAD_COUNT = 10;
  private final Executor executor =
   Executors.newFixedThreadPool(CLIENT_THREAD_COUNT);
  // ...

  public void contextInitialized(ServletContextEvent sce) {
    int count = 0;
    while(count < CLIENT_THREAD_COUNT) {
      executor.execute(()-> {
       System.out.println("Thread Name-ID: "+
       Thread.currentThread().getName()+"-"+
       Thread.currentThread().getId());
       // ...
      });
      count++;
        }
  }
}
```

Cette servlet utilise une file d'attente de type `BlockingQueue` pour débloquer les threads qui traitent la tâche :

```
private static final BlockingQueue<Client> clients = new LinkedBlockingDeque<>();
```

La méthode `take()` permet d'obtenir un objet stocké dans la file d'attente. (Ceci est un appel bloquant, si jamais il n'y a pas d'objet dans la file.)

Il nous faudrait ici définir une classe simulant un client. Chaque client doit disposer d'une méthode (`doWork()`) qui permet d'exécuter la tâche longue.

Pour déclencher l'exécution des tâches, la servlet ajoute un objet Client dans la file de messages (méthode `add(...)`).