Projet JarRet : serveur et client de calcul

Le but du projet `JarRet` est de distribuer de gros calculs sur plusieurs machines (i.e. les clients) et collecter toutes les réponses (i.e avec un serveur). Le client `ClientJarRet` sera exécuté sur de nombreuses machines. Un gros calcul (appelé `job` dans la suite) sera décomposé en plusieurs tâche (appelé `task`. Chaque client se connectera au serveur `ServerJarRet` pour récupérer une tâche, effectuera la tâche et renverra le résultat de la tâche au serveur puis il recommencera.

Un `job` est identifié par son `jobId` qui est un `long`. Il est composé de `nbTasks` tâches qui sont numérotées de 0 à `nbTasks-1`. Les utilisateurs de ce service vont fournir l'URL d'un `jar` et le nom d'une des classes de ce `jar` qui implémente l'interface `upem.jarret.worker.Worker` donnée ci-dessous:

Dans cette présentation du projet, nous allons utiliser le job réalisé par le `jar` qui se trouve à l'url `http://igm.univ-mlv.fr/~carayol/WorkerPrimeV1.jar`. Dans ce jar, la classe `upem.workerprime.WorkerPrime` implémente l'interface `upem.jarret.worker.Worker`. Le but de job est de trouver les nombres premiers entre 0 et 1 000 000 000 -1. Le `jobId` est 23571113 et ce job comporte donc 1000000000 tasks.

Le protocole de communication

Les communications entre le client et le serveur se feront dans le cadre du protocole HTTP.

Lorsque le client demande une tâche au serveur, il fait une requête `GET` de la forme:

```
GET Task HTTP/1.1\r\n
Host: ***server address***\r\n
\r\n
```

La réponse du serveur sera au format du protocole HTTP. Elle contiendra une chaîne de caractères encodée en UTF-8 respectant le format JSON. Le format JSON est décrit **ici**. Pour le manipuler en Java, vous avez de nombreuses librairies comme par exemple **Jackson**. Par exemple, on pourra recevoir:

```
HTTP/1.1 200 OK\r\n
Content-type: application/json; charset=utf-8\r\n
Content-length: 199\r\n
\r\n
{
   "JobId": "23571113",
   "WorkerVersion": "1.0",
   "WorkerURL": "http://igm.univ-mlv.fr/~carayol/WorkerPrimeV1.jar",
   "WorkerClassName": "upem.workerprime.WorkerPrime",
   "Task":        "100"
}
```

Cette réponse donne le `jobId`, l'URL où se trouve le `jar`, le nom de la classe à utiliser dans le `jar` et le numéro de version du `jar` et enfin le numéro de la tâche à accomplir.

Si le serveur n'a plus de tâches à fournir pour l'instant, il renverra:

```
HTTP/1.1 200 OK\r\n
Content-type: application/json; charset=utf-8\r\n
Content-length: 199\r\n
\r\n
{
   "ComeBackInSeconds" : 300
}
```

Il demande au client de se reconnecter dans 300 secondes.

Le client va créer une instance de la classe `upem.workerprime.WorkerPrime` à partir du jar trouvé à l'url `http://igm.univ-mlv.fr/~carayol/WorkerPrimeV1.jar`. Il appelera ensuite la méthode `compute(100)` sur cette instance. Cette méthode

renvoie la chaîne au format JSON suivante:

```
{ "Prime" : false, "Facteur" : 2}
```

Le client va faire une requête POST au serveur au format suivant:

```
POST Answer HTTP/1.1\r\n
Host: *** server address ***
Content-Type: application/json\r\n
Content-Length: ...\r\n
\r\n
...
```

Le contenu de cette requête suit le format suivant:

1. un Long en BigEndian donnant le jobId
2. un Int en BigEndian donnant le numéro de la task
3. une chaîne au format JSON encodée en UTF-8

La chaîne doit avoir le format suivant:

```
 {
  "JobId": "23571113",
  "WorkerVersion": "1.0",
  "WorkerURL": "http://igm.univ-mlv.fr/~carayol/WorkerPrimeV1.jar",
  "WorkerClassName": "upem.workerprime.WorkerPrime",
  "Task":         "100",
  "ClientId": "Maurice",
  "Answer" : { "Prime" : false, "Facteur" : 2}
 }
```

Le champ `Answer` reprend le JSON renvoyé par la méthode `compute`. Le champ `ClientId` est un identifiant choisi au lancement du client qui permettra de savoir qui a fait le calcul des données. Les autres champs sont repris de la réponse du serveur.

Le protocole limite la taille des requêtes du client à 4096 octets (header inclus). Si la réponse devait dépasser cette taille, le client enverra:

```
 {
  "JobId": "23571113",
  "WorkerVersion": "1.0",
  "WorkerURL": "http://igm.univ-mlv.fr/~carayol/WorkerPrimeV1.jar",
  "WorkerClassName": "upem.workerprime.WorkerPrime",
  "Task":         "100",
  "ClientId": "Maurice",
  "Error" : "Too Long"
 }
```

Si l'appel à `compute` lève une exception ou s'il renvoie `null`, le client enverra:

```
 {
 ...
 "Error" : "Computation error"
 }
```

Si la chaîne renvoyée par `compute` n'est pas au format JSON, le enverra:

```
 {
 ...
 "Error" : "Answer is not valid JSON"
 }
```

Si la chaîne renvoyée par `compute` contient un champ de type OBJECT, le enverra:

```
 {
 ...
 "Error" : "Answer is nested"
```

```
}
```

Le serveur renverra le code `200 OK` (toujours dans le protocole HTTP) si tout c'est bien passé et `400 Bad Request` sinon.

Le client

Votre client `upem.jarret.ClientJarRet` prendra en ligne de commande le clientID, l'adresse du serveur et le numéro de port. Pour vous aider avec la création d'une instance implémentant l'interface `upem.jarret.worker.Worker` à partir de l'URL d'un `jar` et du nom de la classe. Nous vous fournissons la classe **WorkerFactory.java** Dans votre client, vous veillerez tout particulièrement:

- à ne pas créer plusieur instance d'un même Worker si cela n'est pas nécessaire. Si vous recevez deux task pour le même `jobId` et pour la même version du Worker, vous réutiliserez le même objet Worker pour les deux.
- vérifier que la sortie du Worker est bien valide (taille total du POST), JSON valide, JSON sans OBJECT,...)
- respecter bien le temps demander par le serveur dans le cas où il répond:

```
{
   "ComeBackInSeconds" : ...
}
```

- quoi qu'il arrive votre client doit continuer à fonctionner !!

Le serveur

Le serveur `JarRetServer` doit être implémenté en mode non-bloquant. Le serveur doit pouvoir traiter plusieurs `jobs` simultanément. Les `jobs` que le serveur doit traiter sont donnés dans un fichier `JarRetJobs.json`. Le format est donné dans l'exemple ci-dessous. Les `jobs` ont une priorité associée. Si la priorité est 0, le job doit être ignoré. Dans l'exemple ci-dessous, le serveur ne traite que deux `jobs` de `jobID` 1 et 3 respectivement. Comme le `job` 1 a la priorité 3 et le `job` 3 a la priorité 2, on attends que le serveur pour 5 tasks distribuées, en distribue 3 pour le `job` 1 et 2 pour le `job` 3.

Les réponses des clients seront rajoutées dans un fichier par `job` après avoir vérifié que la réponse est bien un JSON valide. Vous êtes libres sur le format des fichiers.

Votre serveur devra avoir une console qui supporte au moins les commandes:

1. `SHUTDOWN` qui fait en sorte que l'on accepte plus de nouvelles connections
2. `SHUTDOWN NOW` qui arrête le serveur
3. `INFO` qui donne les informations sur les serveurs : nombres de clients connectés, état des jobs, ...

Le serveur est paramatré par un fichier de configuration appelé `JarRetConfig.json`. Ce donne de nombreuses informations comme le port sur lequel le serveur est attaché, le répertoire des fichiers de logs, le répertoire des fichiers stockant les réponses, la taille maximal autorisée pour un fichier de réponse, le temps en seconde pour la réponse dans le cas où le serveur n'a plus de tâches à donner.

Pour tester

Vous pouvez utiliser le client **ClientJarRet** pour tester vos serveurs.

Vous pouvez utiliser le fichier de configurations suivant pour votre serveur.

Vous pouvez tester vos client sur le serveur `ns364759.ip-91-121-196.eu` sur le port 8080.
