Master DCISS – Université Grenoble-Alpes
Jérôme David

TP Mini Serveur Web basique

Le but de ce TP est de développer un petit serveur web qui est capable de répondre à des requêtes HTTP GET. Le serveur permettra de servir tous les fichiers/dossiers contenus dans un répertoire « racine » donné. Si la requête HTTP GET demande

- un fichier, alors une réponse avec comme corps le contenu du fichier sera retournée.
- un répertoire, alors une réponse HTTP avec comme corps 'une page HTML listant le contenu du répertoire sera retournée.
- une ressource qui n'existe pas, une réponse d'erreur 404 sera retournée.

Vous avez vu le protocole HTTP dans le cours « Introduction à a programmation Web ». Si vous ne vous rappelez plus de son fonctionnement, allez voir votre cours ou le lien suivant https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

Le TP fait appelle non seulement aux flux mais également à la manipulation des chemins (classes et interfaces : `Path`, `Paths`, `Files`).

L'algorithme du serveur sera le suivant :

```
écouter sur port 8080
TANT QUE (vrai) FAIRE
	accepter connexion
	récupérer flux d'entrée et flux de sortie
	lire la première ligne sur flux d'entrée
	SI (méthode get) ALORS
		récupérer le chemin dans la première ligne
		SI (CHEMIN est un répertoire) ALORS
			construire réponse affichant le contenu du répertoire
		SINON SI (CHEMIN est un fichier)
			construire réponse avec le contenu du fichier
		SINON
			envoyer erreur 404
	FIN SI
	fermer la connexion
FIN TANT QUE
```

Si la racine de votre serveur web est le dossier « `/truc/racine/web` » et que la première ligne de requête est « `GET /chose/fichier.html HTTP/1.1` » alors le fichier à servir sera `/truc/racine/web/chose/fichier.html`

Voici un peu d'aide au travers d'un programme qui affiche les requêtes HTTP qui arrivent sur le port 8080 du serveur localhost :

```java
import java.io.*;
import java.net.*;

public class AfficheRequetesHttp {

	public static void main(String[] args) throws IOException {
		ServerSocket s = new ServerSocket(8080);
		while (true) {
			Socket c = s.accept();
			BufferedReader r = new BufferedReader(
						new InputStreamReader(c.getInputStream()));

			String line = null;
			System.out.println("----DEBUT REQUETE----");
			// On arrête à la fin des entête qui est définie comme une ligne vide
			line=r.readLine()
			while (!"".equals(line)) {
```

```
                        System.out.println(line);
                        line=r.readLine()
                    }
                    System.out.println("----FIN REQUETE----");
                    System.out.println();
                    r.close();
                    c.close();
                }
        }
}
```

Pour utiliser ce code, il faut l'exécuter, puis mettre dans votre navigateur des URLs de ce type : http://localhost:8080/une/chemin/de/votrechoix . Comme vous pouvez le voir, le programme contient une boucle infinie. Pour l'arrêter, il faut utiliser la petite icône « carré rouge » en haut à droite de la console d'eclipse. Si vous oubliez cela, vous aurez l'erreur : « `Exception in thread "main" java.net.BindException: Address already in use` » à la prochaine exécution de votre programme.

Ecouter sur le port 8080 : `ServerSocket s = new ServerSocket(8080);`
Accepter une connexion : `Socket c = s.accept();`
Récupérer le flux en lecture `c.getInputStream();`
Récupérer le flux en écriture : `c.getOutputStream();`
Copier le contenu d'un ficher vers un flux en sortie : `Files.copy(path, outputStream);`

Une documentation utile sur l'utilisation des sockets :
http://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

Question subsidiaire : multithreader le serveur pour que chaque requête soit traitée dans un thread différent