Fiche

Communications HTTP en *Java*

Cette article montre un exemple de programme Java permettant d'envoyer des données en POST à un serveur HTTP, lui-aussi écrit en Java.

Les différentes classes présentées ici correspondent à un serveur HTTP basé sur le serveur HTTP intégré au JDK. Une fois exécuté, le serveur peut être accédé directement depuis un navigateur via l'adresse `http://localhost:8080/index`. À noter que la page `index` est un contexte correspondant à l'exécution d'un code *Java*. Pour des explications sur le serveur, reportez-vous à la fiche *Exemple d'utilisation de `HttpServer`*.

1 La classe principale du serveur

Le programme serveur doit d'abord construire un objet `HttpServer` en spécifiant le port d'écoute. Ici, nous choisissons le port 8080 (le port 80 est bloqué dans certains cas).

```
HttpServer serveur = null;
try {
  serveur = HttpServer.create(new InetSocketAddress(8080), 0);
} catch(IOException e) {
  System.err.println("Erreur lors de la création du serveur " + e);
  System.exit(0);
}
```

Une fois le serveur créé, il faut ajouter des contextes associés à des *handlers*. Un contexte est un URL : lorsque le serveur reçoit une requête HTTP sur cet URL, le code du *handler* associé est exécuté. Ici, nous créons un contexte `index` auquel nous associons le *handler* nommé `AccueilSimpleHandler`. La méthode `start` démarre le serveur.

```
serveur.createContext("/index", new AccueilSimpleHandler());
serveur.setExecutor(null);
serveur.start();
```

2 La classe correspondant au *handler*

L'objectif du *handler* est de récupérer les données éventuelles envoyées dans la requête HTTP et d'y répondre en créant une ressource (HTML, image, JSON, etc.). Ici, nous nous contentons de retourner du code HTML qui contient les données qui ont été envoyées en GET et en POST. La fonction principale du *handler* est la méthode `handler` :

```
public void handle(HttpExchange t) {
  ...
}
```

Pour récupérer les données envoyées en GET, nous utilisons la méthode `getRequestURI` sur l'objet `HttpExchange`.

```
URI requestedUri = t.getRequestURI();
String query = requestedUri.getRawQuery();
```

Pour récupérer les données envoyées en POST, nous utilisons la méthode `getRequestBody` sur l'objet `HttpExchange`. Nous utilisons un flux de type `InputStreamReader` encapsulé dans un flux `BufferedReader`.

```
BufferedReader br = null;
try {
  br = new BufferedReader(new InputStreamReader(
      t.getRequestBody(),"utf-8"));
} catch(UnsupportedEncodingException e) {
  System.err.println("Erreur lors de la récupération du flux " + e);
  System.exit(0);
}

try {
  query = br.readLine();
} catch(IOException e) {
  System.err.println("Erreur lors de la lecture d'une ligne " + e);
  System.exit(0);
}
```

Le *handler* peut ensuite envoyer l'en-tête HTTP. Nous récupérons le flux de sortie avec la méthode `getResponseHeaders`. Nous pouvons spécifier le type MIME des données de retour (ici du HTML) puis nous utilisons la méthode `sendResponseHeaders` pour envoyer l'en-tête (ici code 200 et la taille des données). La variable `response` contient le code HTML généré dans le *handler*.

```
try {
  Headers h = t.getResponseHeaders();
  h.set("Content-Type", "text/html; charset=utf-8");
  t.sendResponseHeaders(200, reponse.getBytes().length);
} catch(IOException e) {
  System.err.println("Erreur lors de l'envoi de l'en-tête : " + e);
  System.exit(0);
}
```

Les données sont envoyées à l'aide du flux de sortie que l'on récupère sur l'objet `HttpExchange` à l'aide de la méthode `getResponseBody`.

```
try {
  OutputStream os = t.getResponseBody();
  os.write(reponse.getBytes());
  os.close();
} catch(IOException e) {
  System.err.println("Erreur lors de l'envoi du corps : " + e);
}
```

3 Le client HTTP *Java*

Le client HTTP demande à l'utilisateur de saisir les données à envoyer au serveur HTTP.

Les données sont sous la forme de couples "clé"/"valeur", séparés par des "&". Les données doivent être encodées à l'aide de la méthode `encode` de la classe `URLEncoder` :

```
listeDonnees += URLEncoder.encode(titre, "UTF-8") +
                "=" + URLEncoder.encode(donnees, "UTF-8");
```

Nous créons ensuite un objet `URL` en spécifiant l'URL du serveur :

```
URL url = null;
try {
  url = new URL("http://localhost:8080/index.html");
} catch(MalformedURLException e) {
  System.err.println("URL incorrect : " + e);
  System.exit(0);
}
```

Nous pouvons ensuite établir la connexion avec un objet `URLConnection`.

```
URLConnection connexion = null;
try {
  connexion = url.openConnection();
  connexion.setDoOutput(true);
} catch(IOException e) {
  System.err.println("Connexion impossible : " + e);
  System.exit(0);
}
```

Nous pouvons ensuite envoyer des données. Pour cela, nous utilisons dans cet exemple, un flux `OutputStreamWriter`. L'appel à `flush` envoie les données au serveur.

```
try {
  OutputStreamWriter writer = new OutputStreamWriter(connexion.
     getOutputStream());
  writer.write(listeDonnees);
  writer.flush();
  writer.close();
} catch(IOException e) {
  System.err.println("Erreur lors de l'envoi de la requete : " + e);
  System.exit(0);
}
```

Nous pouvons maintenant récupérer les données envoyées par le serveur. Nous utilisons un flux `InputStreamReader` encapsulé dans un flux `BufferedReader`. Puis nous lisons tous les lignes envoyées.

```
donnees = "";
try {
   BufferedReader reader = new BufferedReader(new InputStreamReader(
      connexion.getInputStream()));
  String tmp;
  while((tmp = reader.readLine()) != null)
    donnees += tmp;
  reader.close();
} catch(Exception e) {
  System.err.println("Erreur lors de la lecture de la réponse : " + e);
  System.exit(0);
}
```

4 Compilation et exécution

Dans un premier temps, compilez les trois classes. Ensuite, dans un premier terminal, exécutez le serveur qui se mettra alors en attente. Dans un second terminal, exécutez le client HTTP et saisissez des données (option 1 dans le menu) et envoyez-les au serveur (option 2 dans le menu).