Pi-Injector

Manuel développeur

Architecture du code

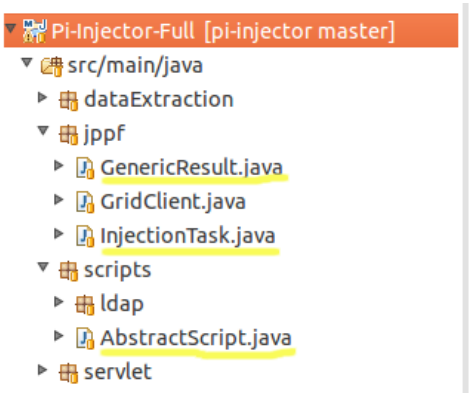
L'architecture du code a été pensée de manière à ce qu'il n'y ait qu'une seule classe (sans compter les vues) à ajouter au projet pour chaque protocole implémenté.

Cette architecture s'articule principalement autour de trois éléments :

- AbstractScript : Un AbstractScript a pour objectif de proposer un mécanisme permettant de stocker des méthodes dans un certain ordre avec leurs paramètres afin de pouvoir les invoquer ultérieurement. Dans le cadre d'un « scénario » (un ensemble de requêtes d'un protocole à exécuter l'une après l'autre), cela nous permet d'enregistrer le code à appeler pour chacune des requêtes. Chaque « Script » (qui hérite de AbstractScript) implémentant un protocole doit alors créer une méthode par type de requête du protocole en question. L'ajout d'un nouveau protocole sera vu plus en détails dans la suite.

- InjectionTask : Il s'agit d'une classe qui est envoyée par sérialisation sur un nœud de la grille distribuée (JPPF-node) afin d'y exécuter le contenu de sa méthode « run ».
  L'InjectionTask prend un AbstractScript dans les paramètres de son constructeur. Ainsi, dans sa méthode « run », il va pouvoir invoquer les méthodes stockées dans la liste des méthodes du AbstractScript tout en calculant leurs temps d'exécution.

- GenericResult : Il s'agit d'un objet qui permet de stocker le temps d'exécution des méthodes enregistrées dans le AbstractScript. Il permet aussi de stocker le temps d'exécution du « scénario » en entier, c'est à dire de l'ensemble de ces méthodes. Les objets GenericResult sont donc crées par l'InjectionTask (qui calcule les temps d'exécution) qui va ensuite les envoyer à intervalles réguliers au « programme client » afin d'y être agrégés et analysés.

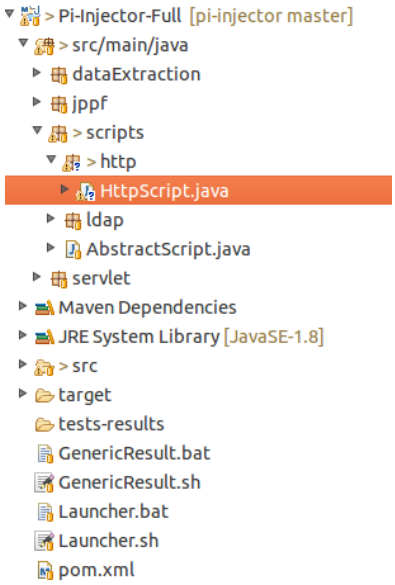


Les 3 fichiers principaux de l'architecture.

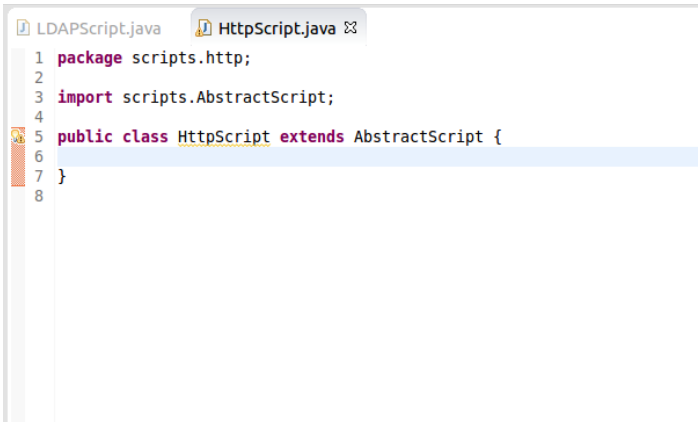
Grâce à cette architecture, il nous sera possible de proposer de nouveaux protocoles très simplement.

Ajout d'un nouveau protocole

Pour ajouter un protocole à l'application Pi-Injector, il faut créer un nouveau script héritant de la classe AbstractScript.



Fichier HttpScript ajouté dans le package scripts.



Classe HttpScript héritant de AbstractScript.

On peut alors ajouter un constructeur permettant de récupérer l'adresse IP et le port du serveur sur lequel il va falloir envoyer les requêtes de notre protocole.

```
1  package scripts.http;
2
3  import scripts.AbstractScript;
4
5  public class HttpScript extends AbstractScript {
6
7      // Needed by AbstractScript which is Serializable
8      private static final long       serialVersionUID = 1L;
9
10     // Target server configuration
11     private String                  hostname;
12     private int                     port;
13
14     public HttpScript(String _hostname, int _port) {
15         hostname = _hostname;
16         port = _port;
17     }
18
```

Constructeur permettant de récupérer l'IP et le port.

Ensuite, si besoin, l'AbstractScript nous offre la possibilité d'allouer et de relâcher des ressources avant et après l'exécution du test de charge qu'on cherche à réaliser (donc avant et après que le scénario soit exécuté un certain nombre de fois).

```
20
21      /*** Script settings ***/
22
23      @Override
24      public void beforeRun() {
25          // Initialize some ressources
26      }
27
28      @Override
29      public void afterRun() {
30          // Release your ressources
31      }
32
33
```

Si besoin, les méthodes beforeRun et afterRun peuvent être redéfinies.

On va maintenant pouvoir écrire les méthodes contenant nos requêtes pour le protocole que l'on cherche à implémenter. Il faut simplement respecter le format « 1 requête = 1 méthode ».

```
34
35      /*** Adding some methods with HTTP requests ***/
36
37      /* GET */
38      public void executeGetRequest(String URL) {
39          // Write the code to perform a GET request
40          // using an API or not...
41      }
42
43      /* POST */
44      public void executePostRequest(String URL) {
45          // Write the code to perform a GET request
46          // using an API or not...
47      }
```

Méthodes pour exécuter les requêtes du protocole.

Une fois que les requêtes du protocole sont implémentées dans ces méthodes, il va nous falloir un moyen de les enregistrer dans un certain ordre afin de correspondre au scénario paramétré par l'utilisateur. Pour ce faire, on va utiliser la méthode « addNewMethod » de AbstractScript.

```
53
54      /*** Offering some methods to add the requests previously written to the script ***/
55
56      /* Add a GET request */
57      public void addGetRequest(String url) {
58          List<Object> params = new ArrayList<Object>();
59          params.add(url);
60
61          addNewMethod("executeGetRequest",
62                       new Class[] {String.class},
63                       params);
64      }
65
66      /* Add a POST request */
67      public void addPostRequest(String url) {
68          List<Object> params = new ArrayList<Object>();
69          params.add(url);
70
71          addNewMethod("executePostRequest",
72                       new Class[] {String.class},
73                       params);
74      }
```

Méthodes proposant d'enregistrer les méthodes exécutant les requêtes du protocole.

« addNewMethod » prend en paramètres :

- le nom de la méthode à enregistrer
- le tableau des types des paramètres attendus par cette méthode
- la liste des paramètres à transmettre.

Ainsi les méthodes qui seront enregistrées pourront être invoquées ultérieurement grâce à la réflexivité de Java, ce qui nous permettra d'exécuter les requêtes sur chacun des nœuds lorsque le test de charge sera démarré.

Lorsqu'un nouveau test est soumis, le formulaire est envoyé sur la servlet TestNew. Celle-ci va lire les premiers paramètres du test comme le nombre d'injecteurs alloués au test, le nombre d'itération à faire sur le scénario et surtout le protocole utilisé.

Si par exemple le protocole est LDAP, la servlet va appeler la classe LDAPScriptBuilder pour lui déléguer la suite du traitement du formulaire. Le rôle de cette classe, située dans le package « servlet.protocol », est de créer un LDAPScript représentant le scénario en y ajoutant les requêtes LDAP à exécuter. Le script est ensuite retourné à la servlet (qui attend simplement un AbstractScript) et qui va lancer l'exécution de ce script.

Pour ajouter un nouveau protocole, il faut donc également créer un « [protocol]ScriptBuilder » et modifier TestNew afin que le nouveau constructeur de script soit utilisé à la soumission d'un nouveau test pour ce protocole.

Enfin, il reste à développer une vue (formulaire) correspondant aux différents types de requêtes du nouveau protocole et à appeler les méthodes « add[protocol]Request » (voir le bout de code précédent) pour permettre d'enregistrer ces requêtes dans un scénario en spécifiant les paramètres souhaités. Une modification du JavaScript est également nécessaire afin d'ajouter les nouveaux types de requêtes au plan de test.