INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

2017

Compte-rendu de TP : Analog

3IF POO2 - 2017 – TP 2: ANALYSE DE LOGS APACHE

Apache HTTP Server

B3428 : BENOIT ZHONG - LUCAS POISSE
3IF GROUPE 4
INSA LYON
ANNEE SCOLAIRE 2016-2017

Table des matières

I. Spécifications générales

En termes de spécifications générales, l'application "Analog" doit pouvoir répondre à plusieurs besoins utilisateur :

- De manière globale, celle-ci doit pouvoir permettre de calculer et d'afficher à l'utilisateur les 10 pages Web les plus consultées à partir d'un fichier journal (*fichier log*) générée par un serveur Web Apache (plus précisément celui hébergeant les pages Web de l'intranet du département IF).

- Il doit être également possible à l'utilisateur de filtrer les données évaluées par l'application, à partir de deux critères pouvant être cumulés (à savoir : ignorer l'accès aux ressources de type image, Javascript ou CSS, ou restreindre l'analyse des logs aux ressources accédées dans un intervalle d'une heure. **Note** : le cahier des charges ne traitant que de manière vague cette fonctionnalité, on souhaite pouvoir utiliser le paramètre de filtrage en renseignant heure, minute et seconde).

- Une autre fonctionnalité principale de l'application est de générer un graphe de navigation global, recensant toutes les navigations de tous les internautes entre les différentes ressources (pages web). Ce graphe doit être construit à partir de tous les résultats évalués par l'application (et pas seulement les 10 premiers en nombre d'accès). Il se présente sous la forme d'un fichier .dot, au format GraphViz. Les noeuds de ce graphe correspondent donc aux ressources accédées (pages Web, etc…) et les arcs de celui-ci sont pondérés par le nombre d'accès effectués d'une ressource à une autre. Par ailleurs, la génération du graphe doit suivre les options de filtrage indiquées par l'utilisateur au lancement du programme.

- L'application doit se présenter comme un programme en ligne de commande. Les différentes options de filtrage et de génération de graphe sont quant à elles activées par l'utilisation d'arguments au lancement de l'application à la manière d'une commande UNIX (voir manuel de l'utilisateur). Aucune charte graphique, ni aucune conception d'IHM particulière n'est donc à prévoir.

II. Spécifications détaillées & techniques

Les spécifications générales ne donnant qu'une vision partielle du besoin utilisateur, il convient de décrire le comportement de l'application dans certains scénarios de manière plus détaillée. Les comportements associés à de tels cas d'utilisation de l'application sont indiqués ci-dessous, de même que les liens vers les tests fonctionnels correspondants.

1. Scénarios de notification

Les scénarios dits "de notification" ici listés ne correspondent pas à des cas d'erreur, mais à des utilisations de l'application entraînant l'affichage de "warnings" sur la sortie standard.

Cas d'utilisation	Résultat	Test fonctionnel associé
Filtrage sur les extensions de fichier (flag '-e'), sur les horaires (flag '-t') ou les deux simultanément	Affichage d'un warning indiquant que certains résultats n'ont pas été pris en compte.	Tests n°2, 3 et 4
Génération de graphe DOT	Affichage d'un warning indiquant qu'un fichier DOT a été généré en cas de succès	Test n°6 `../../analog -g fichier.dot ../filterFile.log`
Cumul génération graphe + filtrage	Affichage des warnings de filtrage et de génération de graphe	Test n°7 `../../analog -e -t 11:16:32 -g fichier.dot ../filterFile.log`
Argument inconnu	Affichage d'un warning 'paramètre inconnu' indiquant l'argument incorrect	Test n°8 (aussi utilisé pour un scénario d'erreur) `../../analog -h -W test.dot -t`
Répétition d'un argument plusieurs fois de suite	Affichage d'un seul warning comme si l'argument n'était passé qu'une seule fois. Pour le flag -t : on considère que le dernier argument de filtrage des horaires passé au programme est celui à utiliser, si celui-ci existe et est syntaxiquement correct.	Test n°11 `../../analog -e -e -e -e -t 10 -t 20 -t 11 ../filterFile.log`
Génération de graphe sur ensemble de données vide	Affichage d'un warning si le filtrage des entrées ne rend aucune donnée à générer dans le graphe. Le fichier .dot n'est en outre pas produit dans ce cas.	Test n°21 `../../analog -e -t 0 -g fichier.dot ../filterFile.log`
Fichier d'enregistrement du graphe déjà existant	Affichage d'un message de confirmation destiné à l'utilisateur. Celui-ci doit choisir s'il souhaite écraser ou non le fichier	Test n°22 `../../analog -g existant.dot ../filterFile.log`

2. *Scénarios de données ambigües*

Les scénarios suivants ne produisent pas d'erreur spécifique ni de notification. Ils proviennent de la présence de données du fichier journal nécessitant quelques traitements de la part de l'application. Ces traitements ont pour but de regrouper certaines URL (cibles ou *referers*) ayant des similitudes, de sorte à les faire figurer sur le même nœud lors de la génération du graphe, et à les traiter de la même façon par l'application.

Scénario	**Traitement**	**Test fonctionnel associé**
Présence de caractères délimiteurs de paramètres ('?', ';'...) à la fin d'une URL (cible ou *referer*).	Suppression des caractères de fin, de la fin du nom de la ressource à la fin de l'URL.	Test n°12 ../../analog -g fichier.dot ../filterFileParametres.log
Referer externe à https://intranet-if.insa-lyon.fr lors de l'accès à une page locale	L'application ne conserve que le nom de domaine du referer externe, et le fera figurer sur le graphe sans indiquer la ressource à partir de laquelle on aura accédé à la cible.	Tests n°6, 7 et 12 (Le referer devient simplement www.google.fr)

3. *Scénarios d'erreur*

Les scénarios suivants produisent une erreur sur le flux d'erreur standard. Note: tous les cas ne sont pas renseignés ici pour des raisons de concision du rapport.

Scénario	**Résultat**	**Tests fonctionnels associés**
Emploi du paramètre -t avec un paramètre invalide, mal formé ou inexistant	Affichage d'un message d'erreur sur le canal d'erreur standard	Tests n°5, 14, 15, 16, 17,18
Génération de graphe sans spécifier de fichier destination.		Test n°19 ../../analog -g -e ../filterFile.log
Lancement de l'application sans indiquer de fichier de lecture		Tests n°8, 13
Erreur de lecture/écriture d'un fichier		Tests n°9, 10

III. Architecture globale de l'application

1. *Conception globale et classes principales*

La phase d'analyse globale a identifié plusieurs classes composant l'application. Ces classes peuvent être décomposées en différents "packages" (c'est à dire un ensemble de modules correspondant à une "couche" spécifique de l'application).

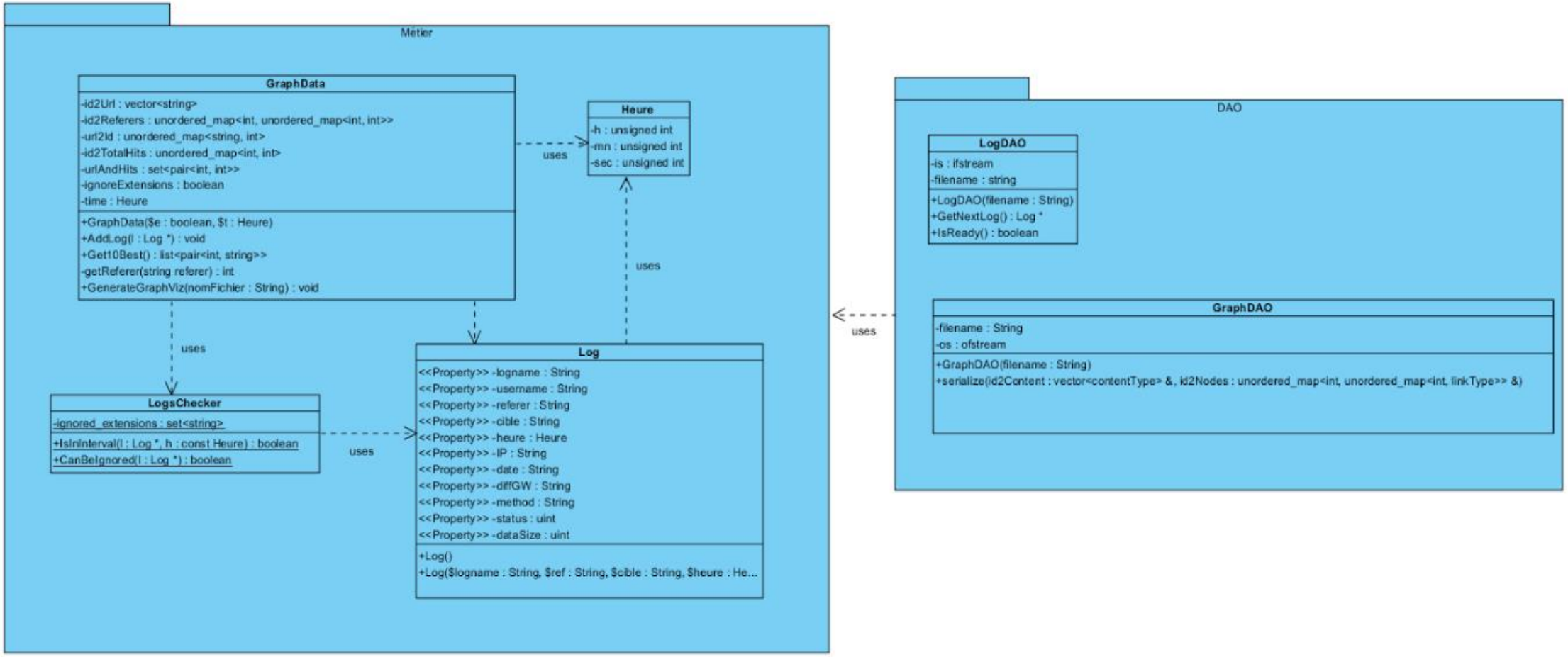
- Le package DAO (ou d'accès aux données) rassemble les classes LogDAO et GraphDAO. Ces classes ont pour but d'accéder aux fichiers de sauvegarde et de chargement utilisés par l'application. La classe LogDAO comporte une méthode renvoyant un objet métier Log (cf paragraphe suivant) construit à partir d'une ligne du fichier journal. La classe GraphDAO a pour tâche de créer un graphe GraphViz de manière générique via la méthode serialize().

- Le package métier rassemble les classes de traitement des données de l'application. Il est composé de 3 classes (et d'un module .h/.cpp définissant une structure Heure). La classe Log représente une ligne du fichier journal. Elle comporte plusieurs attributs correspondant à chaque élément compris dans une ligne de log (heure, IP, cible, referer...). Ces données sont stockées sous différentes formes (structure Heure, chaînes de caractères, entiers non signés) et accessibles via des accesseurs de type "get". La classe GraphData contient les structures de données utilisées par l'application dans le cadre de ses traitements (cf IV). Enfin, la classe LogsChecker contient deux méthodes statiques, permettant d'effectuer un filtrage des objets métier Log ne satisfaisant pas les paramètres indiqués à l'application.

- Il n'existe pas à proprement parler de package d'interface, étant donné que l'application se base essentiellement sur le principe des commandes UNIX. L'interprétation des arguments fournis par l'utilisateur est effectuée depuis le fichier principal main.cpp.

Concernant l'aspect de réutilisabilité exigé, on notera que la classe Log reste utilisable de manière indépendante vis-à-vis des classes GraphData et LogsChecker. De même, il est possible de réutiliser le composant LogDAO (couplé à la classe Log et au module Heure.cpp/Heure.h) pour extraire une ligne d'information d'un fichier journal Apache. Enfin, la classe GraphDAO est également réutilisable puisqu'elle ne requiert pas l'inclusion des classes GraphData ou Log (pas de liaison d'amitié vers une autre classe). Celle-ci peut donc en théorie rester utile pour générer n'importe quel type de graphe, étant donné que les paramètres de la méthode serialize sont génériques. Le diagramme de classes correspondant à l'analyse globale de l'application se trouve page suivante.

2. *Diagramme de classes de l'application*



3. *Présentation des structures de données*

A. Conception détaillée

La conception détaillée des classes a mené à l'élaboration des structures de données employées pour stocker les données traitées par l'application. Ces structures de données sont implémentées en tant qu'attributs de la classe GraphData. Plusieurs structures ont été choisies en conséquence, selon deux critères : performance des algorithmes associés, et économie de mémoire vive.

Il a donc été choisi dans un premier temps (de manière indépendante du langage C++), de limiter au maximum le stockage en mémoire vive des URL correspondant aux cibles/referers. Il a été décidé d'employer deux structures pour stocker ces ressources, qui peuvent être des cibles comme des referers:

- La première, url2Id, est une structure de donnée de type table de hachage, qui à chaque nom de ressource associe un identifiant de type entier. Aucune gestion des conflits ne doit être mise en place (au sens où un seul enregistrement par clé est autorisé). Cette structure est implémentée comme un `std::unordered_map<string, int>` au sein de l'application.
- La deuxième, id2Url, est un tableau comportant **une seule fois** tous les noms de ressource (URL et referers). Le principe est, à chaque nouvelle ressource rencontrée par le programme, d'insérer dans url2Id l'index de l'emplacement de la ressource dans le tableau id2Url, et d'ajouter à la fin de ce tableau l'information inverse. Ces deux opérations sont donc effectuées simultanément, et sont réalisées en temps logarithmique et constant respectivement. De plus, il devient alors très simple de stocker une fois et une seule chaque URL puisque cela revient à utiliser la valeur de `id2Url[url2Id(URL)]`. id2Url est implémenté comme un `std::vector<string>` au sein de l'application.

A partir de cette base, on peut alors définir deux autres structures de données stockant des informations relatives aux URL cibles seulement :

- id2Referers est une structure de type table de hachage. Son rôle est, pour chaque clé de cible (entier, index de la ressource dans id2Url), de stocker une autre table de hachage qui à chaque clé de *referer* (entier, également index dans id2Url) associe un nombre de hits. La structure permet donc, pour chaque cible, de connaître ses referers associés, ainsi que le nombre de hits par referer (information utile pour la génération du graphe DOT). Il a été choisi d'utiliser un conteneur de type `std::unordered_map<int, std::unordered_map<int,int>>` pour implémenter cette structure.
- urlAndHits est une structure de type arbre rouge/noir (donc triée) comportant deux informations : un entier identifiant de cible (index dans id2Url), et un nombre total de hits (soit la somme de tous les nombres partiels de hits sur la cible dans id2Referers). Cette structure ne doit pas comporter de doublons, et doit être triée seulement suivant le nombre total de hits par ordre décroissant, pour accéder facilement aux 10 premiers résultats. A chaque insertion dans id2Referers, on met donc à jour ce nombre total de hits en utilisant l'ancien total ainsi que l'identifiant de cible (opération de complexité logarithmique) pour mettre à jour l'élément "obsolète". Cette structure est implémentée via un `set <pair<int, int>>` en C++.

B. Schémas de structure

Représentation des données sur un jeu de test simple : on considère pour la suite de cette partie le jeu de test suivant (certaines informations inutiles ont été omises) :

```
192.168.0.0 - - [08/Sep/2012:11:16:02 +0200] "GET /temps/4IF16.html HTTP/1.1" 200 12106
"http://intranet-if.insa-lyon.fr/temps/4IF15.html" …

192.168.0.0 - - [08/Sep/2012:15:16:05 +0200] "GET /temps/4IF17.html HTTP/1.1" 200 5189
"http://intranet-if.insa-lyon.fr/temps/4IF16.html" …

192.168.0.0 - - [08/Sep/2012:11:16:06 +0200] "GET /temps/4IF17.html HTTP/1.1" 200 5192
"http://intranet-if.insa-lyon.fr/temps/4IF18.html" …

192.168.0.0 - - [08/Sep/2012:11:16:06 +0200] "GET /temps/4IF19.html HTTP/1.1" 200 5179
"http://intranet-if.insa-lyon.fr/temps/4IF18.html" …
```

Structure `vector` id2Url correspondante :

1	2	3	4	5
/temps/4IF16.html	/temps/4IF15.html	/temps/4IF17.html	/temps/4IF18.html	/temps/4IF19.html

Structure `unordered_map` url2Id correspondante:

/temps/4IF16.html	/temps/4IF15.html	/temps/4IF17.html	/temps/4IF18.html	/temps/4IF19.html
1	2	3	4	5

Structure `unordered_map` id2Referers :

idCible	Referers	
1	idReferer	Nombre de hits associé
	2	1
3	idReferer	Nombre de hits associé
	1	1
	4	1
5	idReferer	Nombre de hits associé
	4	1

Structure `set` urlAndHits (éléments sous la forme <nombreTotalHits, idCible>)

<**2**, 3>	<1, 5>	<1,3>

IV. Manuel d'utilisation

Manuel inspiré du format man (commande linux) :

NOM

Analog -- Extraction des logs apache et liste les 10 ressources les plus consultées, peut également générer un fichier GraphViz des logs.

SYNOPSIS

./analog [-t hour] [-e] [-g graphname] filename

DESCRIPTION

Analog analyse les logs Apache contenus dans le fichier filename et affiche les 10 ressources les plus consultées.

Les options disponibles sont les suivantes :

```
-t hour
```

Analyse les logs compris uniquement entre hour et hour+1

```
-e
```

Ignore les fichiers images, les fichiers Javascript et les fichiers CSS

```
 -g graphname
```

Génère un fichier au format GraphViz dans graphname de l'ensemble des logs analysés.

EXAMPLES

La commande :

```
./analog nomfichier.log
```

Affichera les 10 ressources les plus consultées en analysant nomfichier.log sur la sortie standard.

La commande :

```
./analog -t 12:32:21 -e nomfichier.log
```

Affichera sur la sortie standard les 10 ressources les plus consultées en analysant les logs de 12h32:21 à 13h32:21 en excluant les images et fichiers Javascript et CSS.

La commande :

```
./analog -g graph.dot nomfichier.log
```

Affichera sur la sortie standard les 10 ressources les plus consultées en analysant nomfichier.log sur la sortie standard.

La commande :

```
./analog -t 10:34 -e -g graph.dot nomfichier.log
```

Affichera sur la sortie standard les 10 ressources les plus consultées en analysant les logs de 10h34:00 à 11h34:00 en excluant les images et fichiers Javascript et CSS.