TP1 – Redis

Geoffrey Gaillard

October 2, 2016

---

1 Installation

Via Docker (testé également via une installation classique).

```
> docker run --name redis -p6379:6379 -d redis
```

Lance un container nommé `redis` (`--name`), basé sur l'image `redis`, écoutant sur le port `6379 (-p)` de l'hote. Ce port est mappé au port 6379 du container `(:6379)`.

```
> docker exec -it redis redis-cli
```

Lance une session interactive, exécutant le programme `redis-cli` sur le container nommé `redis`.

1.1 Test de connexion Jedis → Redis

```
Jedis conn = new Jedis("localhost", 6379);
conn.set("cle","valeur");
String val = conn.get("cle");
System.out.println(val);  // => "valeur"
```

Tout est donc OK!

2 Choix des structures de données

Les commandes données en exemple sont une illustration de la démarche uniquement. L'API Jedis fournit des outils plus efficaces pour manipuler les données, notamment par l'utilisation de hashmaps, du pipelining et des transactions. Ces outils sont utilisés dans l'implémentation.

2.1 Question a)

La structure de données me paraissant la plus appropriée pour représenter un article est un hash se présentant sous cette forme:

Pour $id = 72$:

```
> HSET articles:72 timestamp     1380886601
> HSET articles:72 lien          http://www.foo.org/articles/article/1
> HSET articles:72 utilisateur   utilisateur:23
> HSET articles:72 titre         Redis pour les nuls
```

Pour simplifier l'accès aux données, on y stocke aussi son score. Exemple avec un vote:

```
> HSET    articles:72  score     1380886601
> HINCRBY articles:72  score 457
```

Le nombre de votants pourra être obtenu soit

- par la cardinalité de l'ensemble des votants (cet ensemble sera construit plus tard),
- par la formule $\frac{score - timestamp}{457}$.

2.2 Question b)

Pour accéder aux articles, un sorted-set composé du timestamp et de la clef de l'article semble approprié (le timestamp est un entier, et donc ordonnable):

```
> ZADD timelineSet  1380886601  articles:72
```

Pour accéder aux articles en fonction de leurs scores, le score étant un ordonnable (entier), un sorted-set semble aussi approprié:

```
> ZADD    scoresSet 1380886601  articles:72
> ZINCRBY scores    457         articles:72
```

2.3 Question c)

Afin de stocker les utilisateurs ayant voté pour un article donné, on utilise un set (un ensemble non ordonné[1]). Cet ensemble est associé à la clef `articles:<id>:voters`, où *id* est l'identifiant de l'article. Il n'est pas possible de stocker une collection dans un hash, on utilise donc une clef séparée.

```
> SADD articles:72:voters  user:1
```

[1] Ne pas confondre avec la commande `SET`

2.4 Récupération des articles

2.4.1 Tous les articles

Il faut procéder en deux temps pour récupérer tous les articles. D'abord, il faut utiliser la commande `ZRANGE` sur la clef timeline. En effet, la clef timeline est associée à un sorted-set contenant toutes les clefs des articles. On préférera récupérer les articles du plus récent au plus ancien. Pour cela on utilise la commande:

```
> ZREVRANGE timeline 0 -1
```

On obtient la liste des clefs des articles, triée du plus récent au plus ancien. Il faut ensuite récupérer chaque article par sa clef, on utilise pour cela la commande:

```
> HGETALL <clef>
```

Ainsi pour un article ayant pour clef `articles:72` on utilisera:

```
> HGETALL articles:72
```

L'API Jedis nous permet d'effectuer ces opérations en pipeline. Les instructions sont envoyées à Redis mais les réponses ne sont que des promises. Une fois l'ensemble des requêtes envoyées, la méthode `sync()` permet la réalisation de toutes les promises. Cette technique nous permet de récupérer tous les articles en une seule lecture.

L'API Jedis implémente la commande `HGETALL` de façon à ce qu'elle retourne une `Map<String, String>`, nous évitant d'avoir à implémenter notre propre fonction `zipmap`[2].

2.4.2 Les mieux notés

La collection `scores` est aussi un sorted-set, triant les articles par leur nombre de votes, de manière croissante. La commande

```
> ZREVRANGE scores 0 n
```

où $n \in \mathbb{N}$, permets de récupérer les clefs correspondant aux n meilleurs articles.

Il faut ensuite procéder de la même façon qu'au point précédent pour récupérer la liste des articles, c'est-à-dire récupérer chaque article grâce à la commande `HGETALL`. Il est également judicieux d'utiliser un pipeline pour éviter les aller et retour intempestifs entre le client java et le serveur Redis.

2.5 Vote

Le vote sur un article correspond à un "like". C'est-à-dire que l'article gagne un point par vote. Pour chaque vote, son score est incrémenté de 457. Il faut également s'assurer qu'un utilisateur ne puisse pas voter deux fois.

Grâce aux étapes précédentes, nous disposons des clefs:

- `articles:<id>:voters` (set)

[2] https://clojuredocs.org/clojure.core/zipmap

- `articles:<id>:score`
- `articles:<id>:nbVotes`
- `scores` (sorted-set)

Il faut tester si l'utilisateur est déjà présent parmi les utisateurs ayant votés.

```
> SISMEMBER articles:<id>:voters user
```

où `user` est la clef de l'utilisateur votant. La valeur de retour de `SISMEMBER` est dans l'ensemble $\{0, 1\}$ correspondant à un booléen.

Si l'utilisateur n'y est pas, on peut l'ajouter en procédant ainsi:

```
> HINCRBY articles:<id>         score    457
> ZINCRBY scores                457      articles:<id>
> SADD    articles:<id>:voters  user
```

2.6 Gestion des catégories

Afin de gérer les catégories correctement, il faut que l'on puisse trouver tous les articles d'une catégorie donnée, mais également trouver toutes les catégories d'un article donné.

Redis ne permets pas le stockage d'une collection dans un hash (les commandes `HSADD` et `HZADD`, entre autres, n'existent pas) il faut donc stocker ces informations via des clefs séparées.

```
> SADD category:java article:72
```

permet d'ajouter l'article 72 à la catégorie `java`.

```
> SADD articles:72:categories category:java
```

permet de garder une trace des catégories affectées à un article.

Pour récupérer le score des articles d'une catégorie donnée, le plus efficace est d'utiliser la commande

```
> ZINTERSTORE temp 2 category:<group> scores
```

où

- `temp` est le nom d'une collection temporaire, dont le nom est possiblement généré et unique,
- `group` est le nom de la catégorie d'articles (`java`, `python`, `R`), formant une clef désignant un ensemble de clefs d'articles,
- `scores` est le sorted-set des scores,
- 2 est le nombre d'ensembles participants à l'intersection.

On obtient dans `temp` l'ensemble des articles avec leurs scores faisant partie de la catégorie donnée.

On peut alors obtenir les scores via la commande

```
> ZRANGE temp 0 -1 WITHSCORES
```

ou en récupérant directement, via un pipeline et la commande `HGETALL`, l'ensemble des données des articles.

2.7 Implémentation

L'implémentation est en Java 8 et disponible à l'adresse https://github.com/ggeoffrey/tp1-redis. L'hôte et le port sur lesquels se connecter peuvent êtres spécifiés dans `config/redis.properties`.

Assurez-vous de disposer d'Apache Maven, puis procédez comme ceci:

```
> git clone https://github.com/ggeoffrey/tp1-redis.git
  # éditez éventuellement config/redis.properties
> mvn clean test
```

Les sources sont composées d'une classe ArticleStore servant à manipuler les articles (création, sauvegarde, récupération), d'une classe ArticleStoreTest permettant d'effectuer les tests unitaires et d'autres classes utilitaires. Il n'y a pas de classe principale (Main).

Pour obtenir la documentation, exécutez

```
> mvn javadoc:javadoc
```

et ouvrez le fichier `target/site/apidocs/index.html`.