

Redis



P.Mathieu

IUT de Lille

<http://www.iut-a.univ-lille.fr>

prenom.nom@univ-lille.fr

Redis



Caractéristiques

- ▶ REDIS : *REmote Dictionary Server*
- ▶ Base NoSQL de type Clé-Valeur
(comme DynamoDB, BerkeleyDB ou InfinityDB)
- ▶ in-memory key-value store
(ni schema, ni noms de colonnes, ni documents complexes)
- ▶ Système Client-serveur pour partager des données à durée de vie courte entre différents clients
- ▶ Extrêmement rapide
- ▶ Types de données variés et puissants
- ▶ Système de réplication, de monitoring et de sentinelles intégré
- ▶ <https://redis.io/>

Une database key-value n'est ni plus ni moins ...

Qu'une hashmap ...

- ▶ évoluée
- ▶ persistante
- ▶ client-serveur

Installation et Lancement

- ▶ Download : <https://redis.io/> puis `make`
- ▶ Serveur : `redis-server`
- ▶ Port utilisé par défaut : 6379
- ▶ Fichiers :
 - Configuration : `redis.conf`
 - Database : `dump.rdb`
- ▶ Client interactif :
 - `redis-cli` (quit **pour quitter**)
 - `redis-cli -h localhost -p 6379 -n 2`

Tutoriel interactif : <https://try.redis.io/>

- ▶ Redis gère 16 databases par défaut
- ▶ Elles sont identifiées par un numéro (première 0)
- ▶ `redis-cli -n 5` permet de se connecter à la 5è
- ▶ `select n` permet de changer de base en interactif
- ▶ Le numéro est indiqué dans le prompt à partir de la 1
- ▶ Jusqu'à 2^{32} clés dans une database
- ▶ Le nombre de databases est paramétrable dans les fichiers de config

Un exemple basique

```
> redis-cli
127.0.0.1:6379> set paul.bdd 12
OK
127.0.0.1:6379> set paul.math 15
OK
127.0.0.1:6379> set jean.bdd 17
OK
127.0.0.1:6379> set jean.math 10
OK
127.0.0.1:6379> get paul.bdd
"12"
127.0.0.1:6379> keys jean*
1) "jean.math"
2) "jean.bdd"
127.0.0.1:6379> quit
>
```


Chaque donnée est associée à une clé (printable ascii)

- ▶ EXISTS key
- ▶ DEL key (ou UNLINK, FLUSHDB, FLUSHALL)
- ▶ RENAME oldkey newkey
- ▶ TYPE key

- ▶ KEYS pattern
- ▶ DBSIZE

Un usage “normalisé” des clés permet de les retrouver facilement
(par ex, toutes les clés préfixées par `user:` ou `user/`)

Huit types de données différents

- ▶ Strings
- ▶ Conteneurs
 - ▶ Lists
 - ▶ Sets
 - ▶ SortedSets
 - ▶ Hashes
- ▶ Bitmaps
- ▶ Hyperlogs
- ▶ index géoSpatial

Chaque type de données utilise des ordres spécifiques (L^* , S^* , Z^* , H^*)

Entiers et réels sont codés dans le type String

- ▶ SET key val
- ▶ GET key
- ▶ INCR INCRBY DECR DECRBY
- ▶ APPEND key val
- ▶ STRLEN key
- ▶ GETRANGE key start end
- ▶ SETRANGE key offset value

```
set nom paul
```

Voir <https://redis.io/commands/string>

Capacités de calcul inexistantes :

- ▶ pas de `count (*)`, de `SUM` ou de `AVG`
- ▶ Pas d'arithmétique
- ▶ Pas de `GROUP BY`, `ORDER BY` etc ...

Uniquement du stockage clé/valeurs et incrément/décément

Les listes peuvent contenir plusieurs strings (éventuellement des doublons)

- ▶ LPUSH key val1 ... valn `LPUSH chats siamois persan`
- ▶ RPUSH key val1 ... valn
- ▶ LRANGE key start stop `lrange macle 0 -1 pour tout lister`
- ▶ LLEN key
- ▶ LPOP key
- ▶ RPOP key
- ▶ LREM count key
- ▶ LTRIM key start end `key peut être négatif`

Voir <https://redis.io/commands/#list>

Un `set` ne contient pas de doublons

- ▶ `SADD key val1 ... valn`
- ▶ `SMEMBERS key`
- ▶ `SCARD key`
- ▶ `SISMEMBER key val`
- ▶ `SDIFF key1 key2 ...`
- ▶ `SINTER key1 key2 ...`
- ▶ `SUNION key1 key2 ...`
- ▶ `SRANDMEMBER key nb`

```
SADD lettres a a b b c
```

```
ou SDIFFSTORE res key1 key2 ...  
ou SINTERSTORE res key1 key2 ...  
ou SUNIONSTORE res key1 key2 ...
```

Un `SortedSet` range ses éléments sans doublon, selon un score, par ordre croissant (mélange entre SET et HASH)

- ▶ `ZADD key score1 val1 ... scoren valn`
- ▶ `ZRANGE key start stop`
- ▶ `ZCARD key`
- ▶ `ZCOUNT key min max`
- ▶ `ZREM key val`
- ▶ `ZDIFF` , `ZINTER`, `ZUNION`,

Les Hashes sont des collections de paires clés/valeurs
Une sorte d'enregistrement structuré

- ▶ HSET key field1 value1 ... fieldn valuen
- ▶ HLEN key
- ▶ HKEYS key
- ▶ HGET key field
- ▶ HGETALL key

Avec les Hashes il devient très facile de coder une forme de table traditionnelle

Coder une table `personne(id, nom, prenom, age)`

```
hset personne:1 nom durand prenom paul age 10
hset personne:3 nom lefebvre prenom jean age 20
hset personne:7 nom dubois
hset personne:7 prenom paul age 15
hset personne:5 age 8 prenom julie nom leroy
```

```
keys personne:*
hgetall personne:3
```

Données géospaciales (extension de SortedSET)

Chaque donnée est fournie avec un nom, une longitude et une latitude

- ▶ GEOADD longitude latitude member
- ▶ GEODIST
- ▶ GEOSEARCH
- ▶ GEORADIUS

```
GEOADD villes 13.404954 52.520008 Berlin 2.352222 48.856613 Paris
```

```
GEOADD villes -0.127758 51.507351 Londres
```

```
GEODIST villes Paris Berlin KM
```

```
"877.7065"
```

```
ZRANGE villes 0 -1
```

Le “Time To Live (TTL)”

Un dispositif de durée de vie des clés est possible pour chaque clé

- ▶ `EXPIRE key durée`
Affecte la durée de vie
La durée est exprimée en secondes
- ▶ `TTL key`
fournit la durée de vie
-1 signifie sans expiration, -2 clé inexistante
- ▶ La durée est réinitialisée à chaque affectation

- ▶ `set key val EX durée (en sec)`
- ▶ `set key val PX durée (en millisec)`
- ▶ `set key val keepttl (conserve l'actuel ttl)`

Publish-subscribe est un mécanisme de publication de messages et d'abonnement.

- ▶ SUBSCRIBE canal1 .. canalN
 - ▶ PSUBSCRIBE pattern
 - ▶ PUBLISH canal message
 - ▶ UNSUBSCRIBE
-
- ▶ En mode SUBSCRIBE le client est bloqué
 - ▶ Les messages ne sont pas persistants
 - ▶ Si on s'abonne "après" l'envoi, il est trop tard

Librairie jedis

```
import redis.clients.jedis.Jedis;

public class TestRedis {
    public static void main(String args[]) {
        try (Jedis jedis = new Jedis("localhost", 6379))
        {
            jedis.select(4); // database 4
            jedis.set("score", "0");
            jedis.incrBy("score", 18);
            if (jedis.exists("score"))
                System.out.println(jedis.get("score"));
            jedis.del("score");
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

Librairie redis-py

```
import redis

def test():
    r = redis.StrictRedis(host='localhost', port=6379, db=4)
    r.set("score", "0")
    r.incrby("score", 18)
    if (r.exists("score")):
        print( r.get("score") )

    r.delete("score")
    r.close()

test()
```

- ▶ REDIS est très rapide car il fonctionne en mémoire
- ▶ Nombreux types de données
- ▶ Nombreuses méthodes très optimisées pour chaque type
- ▶ Calcul de données géospaciales
- ▶ Gestion des TTL
- ▶ Pub/Sub immédiat

- ▶ Mise en cache
stocker en base les données fréquemment consultées
- ▶ Gestion des sessions web
Partager les sessions entre plusieurs instances du serveur
- ▶ Messagerie Pub/Sub
architectures pilotées par les évènements
- ▶ Tableaux de bord et compteurs
scores et des classements pour diverses entités
- ▶ Calculs de données spaciales (bornes wifi, bornes recharge etc ..)
- ▶