Mise en place de Redis

Introduction

Dans le chapitre précédent, nous avons exploré un aperçu de la base de données Redis. Maintenant, passons à la mise en place de notre environnement et à l'installation de Redis, que ce soit sur notre machine locale ou dans le cloud. Bien que le processus d'installation de Redis soit bien documenté, ce chapitre vise à servir de guide définitif couvrant les exigences environnementales et les ajustements potentiels pour les besoins futurs.

Structure

Ce chapitre aborde les sujets suivants :

- Configuration de l'environnement Redis
- Configuration de Redis
- Scripting avec Redis
    - Avantages du scripting Redis
    - Fonctionnement du scripting Redis
    - Clés et arguments dans le scripting Redis
    - Comment appeler des scripts Redis en utilisant l'interpréteur Lua

Objectifs

À la fin de ce chapitre, vous serez en mesure de :

- Installer Redis sur votre système Debian/Ubuntu
- Comprendre les paramètres de configuration de base de Redis dans le fichier de configuration Redis
- Vous familiariser avec l'environnement de scripting Redis

Environnement Redis

Redis est reconnu comme la base de données NoSQL la plus populaire. Selon le site officiel de Redis, Redis est écrit en ANSI C et fonctionne sans problème sur la plupart des systèmes POSIX tels que Linux, *BSD et OS X, sans nécessiter de dépendances externes. Bien que Redis soit principalement développé et testé sur Linux et OS X, il peut également fonctionner sur des systèmes dérivés de Solaris comme SmartOS, bien que le support soit limité. Il est important de noter qu'il n'existe pas de support officiel pour les versions de Redis sur Windows.

Pour obtenir la version la plus récente de Redis et les plateformes prises en charge, il est recommandé de consulter le site Redis (https://redis.io/topics/introduction).

Installation de Redis sur wsl2 Ubuntu 20.04

Prérequis

- Un système d'exploitation basé sur Linux (Ubuntu 20.04)
- Un terminal pour exécuter les commandes
- Un accès à Internet pour télécharger les packages

Étapes d'installation

1. Ouvrez le terminal et mettez à jour les packages existants en exécutant la commande suivante :

```
sudo apt update
```

2. Installez les packages de base nécessaires pour compiler Redis en exécutant la commande suivante :

```
sudo apt install redis-server
```

Redis configuration

1. Ouvrez le fichier de configuration Redis en exécutant la commande suivante :

```
sudo nano /etc/redis/redis.conf
```

2. Exemple de configuration pour set un password pour accéder à Redis

```
requirepass "yourpassword"
```

3. Pour "daemonize" Redis, changer la ligne suivante dans le fichier de configuration Redis :

```
supervised auto

en

supervised systemd
```

4. Redémarrez le service Redis pour appliquer les modifications en exécutant la commande suivante :

```
sudo service redis-server restart
```

```
sudo service redis-server status
```

Tester le serveur redis

1. Ouvrez le client Redis en exécutant la commande suivante :

```
redis-cli -a yourpassword
```

2. Pour tester si le serveur Redis est en cours d'exécution, exécutez la commande suivante :

```
ping

OUTPUT : PONG
```

3. Performer une simple key-set operation

```
set keycheck "It's working!"

OUTPUT : OK
```

4. Pour récupérer la valeur de la clé, exécutez la commande suivante :

```
get keycheck

OUTPUT : "It's working!"
```

5. Pour vérifier que les données sont toujours présentes après un redémarrage du serveur Redis, redémarrez le serveur Redis en exécutant la commande suivante :

```
sudo service redis-server restart
```

6. Ouvrez le client Redis et récupérez la valeur de la clé en exécutant les commandes suivantes :

```
redis-cli -a yourpassword
get keycheck

OUTPUT : "It's working!"
```

> *STRUCTURES DE DONNÉES REDIS EN DÉTAIL*

Introduction

Maintenant que nous avons configuré l'environnement Redis, nous sommes prêts à approfondir notre compréhension de Redis. Lorsqu'on étudie une base de données, il est crucial de comprendre comment elle stocke les données, quelles opérations elle prend en charge, et quelles sont ses capacités.

Redis est une base de données clé-valeur. Le choix du type de données pour stocker les données au départ est très important car il détermine les fonctionnalités que nous pouvons mettre en œuvre. Ce chapitre est donc dédié à une compréhension approfondie des structures de données Redis et de leurs cas d'utilisation.

Structure

Dans ce chapitre, nous aborderons les sujets suivants :

- Les chaînes Redis
- Les hachages Redis

Chaque structure de données sera explorée en détail, avec des explications sur leur utilisation, leurs capacités et leurs meilleures pratiques.

En comprenant ces structures de données, vous serez en mesure de choisir la meilleure approche pour modéliser vos données dans Redis en fonction des exigences spécifiques de votre application. Cela vous permettra d'exploiter pleinement les fonctionnalités et les performances de Redis pour votre cas d'utilisation particulier.

Strings Redis

Les chaînes Redis sont le type de données le plus fondamental dans Redis, similaire aux types de chaînes disponibles dans d'autres langages, mais elles offrent également des fonctionnalités uniques. Voici une analyse approfondie des caractéristiques des chaînes Redis :

1. **Commandes universelles :** Certaines commandes Redis comme EXISTS et DEL peuvent être utilisées avec n'importe quel type de clé, même si elles ne sont pas spécifiquement définies pour ce type.
2. **Accès aléatoire :** Les chaînes Redis peuvent fonctionner comme des vecteurs d'accès aléatoire.
3. **Encodage efficace :** De grandes quantités de données peuvent être encodées de manière efficace dans des espaces réduits.

Cas d'utilisation :

Les chaînes Redis sont extrêmement polyvalentes et peuvent être utilisées dans divers cas d'utilisation où une réponse rapide est nécessaire :

1. **Stockage de chaînes simples :** Utilisation pour stocker et servir des pages web statiques de manière efficace.
2. **Mise en cache :** Très couramment utilisé pour mettre en cache les données les plus fréquemment consultées, ce qui accélère les temps de réponse en récupérant les résultats à partir de sources de données telles que des bases de données relationnelles, des systèmes de fichiers, etc.
3. **Compteurs :** Idéal pour stocker des statistiques simples telles que les visites quotidiennes d'utilisateurs sur un site web ou sur des pages spécifiques.
4. **Objets temporairement traités :** Utilisé dans les microservices pour stocker temporairement les données traitées avant de les transmettre à l'étape suivante du processus.
5. **Catalogues maîtres, configurations :** Stockage des configurations d'application et des catalogues maîtres comme les noms de villes, les codes de pays, etc., fréquemment utilisés dans l'application.

Excercice

Dans Redis, les chaînes sont utilisées pour stocker des valeurs simples comme des OTP (One Time Passwords), et elles peuvent être associées à des expirations pour contrôler leur durée de validité. Voici comment cela fonctionne :

1. **Création d'un OTP avec expiration :** Pour générer un OTP et le stocker dans Redis avec une expiration de 60 secondes, vous utilisez la commande `SET` avec l'option `EX` suivie du nombre de secondes avant expiration :

   ```
   SET otp:use_10001 4532 EX 60
   ```

   Cette commande crée une clé `otp:use_10001` avec la valeur `4532` et la règle d'expiration de 60 secondes.

2. **Vérification du TTL (Time To Live) :** Pour vérifier combien de temps il reste avant que la clé expire, utilisez la commande `TTL` suivie du nom de la clé :

   ```
   TTL otp:use_10001
   ```

   Cette commande retournera le nombre de secondes restantes avant que la clé `otp:use_10001` n'expire. Par exemple, si elle retourne `50`, cela signifie qu'il reste 50 secondes avant expiration.

3. **Récupération de la valeur de la clé :** Pour récupérer la valeur associée à la clé `otp:use_10001`, utilisez la commande `GET` :

   ```
   GET otp:use_10001
   ```

   Cette commande retournera la valeur actuelle de la clé `otp:use_10001`, par exemple "`4532`".

4. **Expiration de la clé :** Une fois que la durée d'expiration définie (60 secondes dans cet exemple) est écoulée, la clé expire automatiquement. Si vous utilisez `TTL` après l'expiration, cela retournera `-2`, indiquant que la clé n'existe plus. De même, si vous utilisez `GET`, cela retournera `nil` pour indiquer que la clé n'a plus de valeur associée.

Cette méthode est efficace pour générer des OTP temporaires et les gérer automatiquement en utilisant les fonctionnalités d'expiration intégrées à Redis, assurant ainsi une gestion propre et automatisée des données temporaires dans votre application.

Hashes Redis

Les hachages Redis sont des structures de données qui stockent des paires clé-valeur, où chaque clé est unique dans le hachage. Les hachages Redis sont similaires aux tableaux associatifs dans d'autres langages de programmation, mais ils offrent des fonctionnalités supplémentaires. Voici une analyse approfondie des caractéristiques des hachages Redis :

Cas d'utilisation

Exemple d'un profile utilisateur représenté par différents langagues

Les hashes dans Redis sont des structures de données permettant de stocker plusieurs champs associés à leurs valeurs. Cela les rend idéales pour représenter des objets complexes comme des profils d'utilisateurs, des configurations, etc. Voici comment vous pouvez travailler avec les hashes dans Redis :

1. **Création d'un hash avec plusieurs champs :** Pour créer un hash et définir plusieurs champs avec leurs valeurs associées, utilisez la commande `HSET`. Par exemple, pour créer ou mettre à jour un profil de blogueur avec l'ID `blogger:10001` :

   ```
   HSET blogger:10001 lastvisit "April 05, 2020 10:47:00" name "Terence Laurent" points 54210 rank 1 blogposts 45 reviews 50 follo
   ```

   Cette commande ajoute les champs `lastvisit`, `name`, `points`, `rank`, `blogposts`, `reviews`, et `followers` avec leurs valeurs respectives au hash `blogger:10001`.

2. **Récupération de tous les champs d'un hash :** Pour récupérer tous les champs et leurs valeurs d'un hash, utilisez la commande `HGETALL`. Elle retourne une liste de tous les champs suivis de leurs valeurs dans l'ordre :

   ```
   HGETALL blogger:10001
   ```

   Ce qui retournerait :

   ```
   1) "lastvisit"
   2) "April 05, 2020 10:47:00"
   3) "name"
   4) "Terence Laurent"
   5) "points"
   6) "54210"
   7) "rank"
   8) "1"
   9) "blogposts"
   10) "45"
   11) "reviews"
   12) "50"
   13) "followers"
   14) "560"
   ```

3. **Modification d'un champ dans un hash :** Pour modifier la valeur d'un champ spécifique dans un hash, vous pouvez utiliser à nouveau la commande `HSET`. Par exemple, pour mettre à jour le nombre de `blogposts` pour `blogger:10001` :

   ```
   HSET blogger:10001 blogposts 46
   ```

   Cette commande met à jour le champ `blogposts` à `46` pour le hash `blogger:10001`.

Les hashes dans Redis sont flexibles et peuvent contenir jusqu'à 2^32 - 1 paires champ-valeur, ce qui les rend adaptés pour gérer des données complexes avec une performance élevée. Ils offrent une alternative efficace aux structures de données plus traditionnelles et sont bien adaptés aux besoins des applications modernes.

On peut aussi incrémenter un champ dans un hash avec la commande `HINCRBY` pour les valeurs numériques

```
HINCRBY blogger:10001 points 1234512
```

Lists Redis

Quickstart de la partie Listes en français

Les listes Redis

Une liste est une séquence d'éléments ordonnés, et l'ordre est basé sur leur séquence d'insertion. Comme les éléments stockés dans une liste sont des chaînes de caractères, ils suivent toutes les caractéristiques des types de données de chaîne de caractères. Voici une représentation schématique de la façon dont Redis stocke ses données en interne :

Cas d'utilisation - Planification de tâches

La planification de tâches est un cas d'utilisation très courant pour les listes Redis. Par exemple, vous pouvez avoir une application qui traite des images. Cette application peut permettre aux utilisateurs de télécharger leurs photos en haute définition, puis vous pouvez avoir des tâches qui traitent ces images pour détecter du contenu pornographique, compresser les images pour le web, convertir les formats, etc. Vous pouvez également vouloir prioriser le traitement des images si elles sont téléchargées par des profils tendance, comme des célébrités. Dans ce cas, vous pouvez simplement utiliser une ou plusieurs listes Redis pour ajouter les images à traiter, et ensuite il peut y avoir plusieurs tâches qui récupèrent une image à la fois et les traitent. En cas d'exception, vous pouvez écrire une logique pour réinsérer cette image dans la liste de traitement.

Exemple de commandes

Pour ce cas d'utilisation, nous utiliserons la commande `LPUSH` pour insérer une image en tête de liste (si elle est téléchargée par un profil de célébrité) et la commande `RPUSH` pour insérer en fin de liste.

Ajout d'images

```
LPUSH list-images image1   # la liste "list-images" contient maintenant "image1"
LPUSH list-images image2   # la liste "list-images" contient maintenant "image2", "image1"
RPUSH list-images image3   # la liste "list-images" contient maintenant "image2", "image1", "image3"
```

Vous avez peut-être remarqué que l'image3 a été ajoutée à la fin de la liste "list-images" par la commande `RPUSH`.

Récupération d'images

Vous pouvez utiliser la commande `LPOP list-images` pour récupérer un élément à la fois à partir de la gauche.

Planificateur de tâches basique

Les opérations peuvent être subdivisées comme suit :

A. **Générateur de tâches** :

1. Enqueue les tâches
2. Priorise les tâches
3. Ajoute au répartiteur

B. **Générateur de ressources** :

1. Enqueue les ressources
2. Ajoute au répartiteur

C. **Enfin**, après le traitement, la ressource du répartiteur peut effectuer les tâches allouées.

En utilisant la commande `LPUSH`, vous pouvez ajouter les tâches ainsi que les ressources à votre liste. Voici un exemple de commandes :

Ajout de tâches et ressources

```
LPUSH task-list task1   # Ajoute une tâche en tête de liste
LPUSH task-list task2   # Ajoute une autre tâche en tête de liste
RPUSH resource-list resource1   # Ajoute une ressource en fin de liste
RPUSH resource-list resource2   # Ajoute une autre ressource en fin de liste
```

Ces opérations vous permettent de gérer efficacement la planification et le traitement des tâches en utilisant les listes Redis.

url: https://redis.io/commands#list (https://redis.io/commands#list)

Sets Redis

Quickstart des Listes Redis

Redis propose diverses structures de données, dont les listes, qui sont des séquences d'éléments ordonnés. Voici comment utiliser les listes dans Redis.

Utilisation de base des listes

Les listes sont des séquences ordonnées d'éléments, et l'ordre est basé sur leur séquence d'insertion. Les éléments stockés dans une liste sont des chaînes et suivent toutes les caractéristiques des types de données chaîne.

Exemple de cas d'utilisation - Ordonnancement de tâches

L'ordonnancement de tâches est un cas d'utilisation très courant des listes Redis. Par exemple, vous pouvez avoir une application pour traiter des images. Vous pouvez prioriser certaines images en les plaçant au début de la liste si elles proviennent de profils tendance (comme des célébrités).

Voici comment utiliser les commandes `LPUSH` et `RPUSH` pour insérer des images dans une liste :

1. **Ajouter des éléments à la liste :**

```
LPUSH list-images "image1"
LPUSH list-images "image2"
RPUSH list-images "image3"
```

   - Après ces commandes, la liste `list-images` contiendra : `["image2", "image1", "image3"]`.

2. **Récupérer des éléments de la liste :**

```
LPOP list-images
```

   - Cette commande retirera et retournera le premier élément de la liste, soit `"image2"`.

Commandes utiles pour les listes

- **LPUSH** : Ajoute un ou plusieurs éléments au début de la liste.
- **RPUSH** : Ajoute un ou plusieurs éléments à la fin de la liste.
- **LPOP** : Retire et retourne le premier élément de la liste.
- **LRANGE** : Retourne les éléments d'une liste dans une plage donnée.

Exemple complet

```
# Ajouter des étudiants à la liste
LPUSH students "John"
LPUSH students "Maria"
LPUSH students "Julie"

# Vérifier le contenu de la liste
LRANGE students 0 2  # Retourne ["Julie", "Maria", "John"]

# Enlever un élément sans vider toute la liste
LPOP students  # Retire "Julie"

# Vérifier la liste après LPOP
LRANGE students 0 -1  # Retourne ["Maria", "John"]
```

Utilisation des ensembles Redis

Les ensembles Redis sont une collection non ordonnée de chaînes. Contrairement aux listes, les ensembles ne permettent pas les opérations `push` et `pop` simples, mais offrent des opérations puissantes comme l'intersection, la différence et l'union.

Commandes d'opérations sur les ensembles

- **SADD** : Ajoute un ou plusieurs éléments à un ensemble.
- **SREM** : Retire un ou plusieurs éléments d'un ensemble.
- **SINTER** : Intersecte plusieurs ensembles.
- **SUNION** : Ajoute plusieurs ensembles.
- **SDIFF** : Soustrait plusieurs ensembles.

Exemple d'utilisation des ensembles

```
# Ajouter des éléments à un ensemble
SADD names "Kevin"
SADD names "Peter"
SADD names "Moris"
SADD names "Oliver"

SADD names2 "George"
SADD names2 "Noah"
SADD names2 "Oliver"

# Intersecter deux ensembles
SINTER names names2  # Retourne ["Oliver"]

# Union de deux ensembles
SUNION names names2  # Retourne ["Kevin", "Peter", "Moris", "Oliver", "George", "Noah"]

# Différence de deux ensembles
SDIFF names names2  # Retourne ["Kevin", "Peter", "Moris"]
```

Redis propose de nombreuses autres commandes pour les opérations de base sur les ensembles et les listes. Pour plus de détails, vous pouvez consulter la documentation officielle ici (https://redis.io/commands#set).

En utilisant ces commandes, vous pouvez gérer efficacement vos données avec Redis et les adapter à divers cas d'utilisation comme les réseaux sociaux, la communication réseau, et bien d'autres.

Cas d'utilisation – Suivi des utilisateurs uniques visitant un site web

Prenons l'exemple d'une plateforme d'apprentissage en ligne où vous allez suivre les utilisateurs visitant le site web. De plus, vous pouvez utiliser la même logique pour suivre les utilisateurs visitant chaque cours et filtrer les cours les plus demandés.

Voici un flux général que vous pouvez appliquer pour implémenter cette logique :

Étapes de la mise en œuvre

1. **Sharder la base de données** :
    - Initialement, il y aura une grande base de données contenant plusieurs champs pour chaque utilisateur. Vous devez fragmenter cette base de données en petites partitions pour stocker les données dans un ensemble Redis. La fragmentation (sharding) est un processus de partitionnement de la grande base de données en petites parties. Cela rend la traversée de la base de données beaucoup plus rapide car nous considérons uniquement les champs pertinents. Nous effectuons ce processus en utilisant l'une des commandes Redis communes pour les ensembles, `SADD`.

      ```
      SADD user_ids_shard1 "user1" "user2" "user3"
      SADD user_ids_shard2 "user4" "user5" "user6"
      ```

2. **Stocker le nombre unique d'utilisateurs** :
    - Maintenant que nous avons fragmenté les données, nous pouvons stocker le nombre unique d'utilisateurs. Un nouvel ensemble sera généré chaque jour avec une date particulière et les utilisateurs visitant le site web à cette date. Lors de l'ajout de données à l'ensemble de la date du jour, nous devons d'abord vérifier si l'utilisateur est déjà présent dans l'ensemble. Si ce n'est pas le cas, nous ajoutons l'identifiant utilisateur unique à l'ensemble.

      ```
      SADD daily_visitors_2024_06_13 "user1" "user2"
      SADD daily_visitors_2024_06_13 "user3"  # Ajout de "user3"
      ```

3. **Compter les utilisateurs uniques** :

- Un autre ensemble gardera le compte des utilisateurs uniques. Dès que nous ajoutons l'utilisateur à l'ensemble des visiteurs quotidiens, si l'identifiant utilisateur est unique, nous devons incrémenter le compte dans l'ensemble de comptage.

```
SADD unique_visitors "user1" "user2" "user3"
```

4. **Suivre les variations des utilisateurs** :

   - Vous devez suivre les variations du nombre d'utilisateurs visitant le site web. Pour ce faire, vous devez écrire une fonction qui :

     1. Définit une variable avec un compte par défaut (un peu élevé).
     2. Garde une copie locale de la valeur attendue.
     3. Utilise la valeur calculée ou, si quelqu'un d'autre l'a calculée, utilise cette valeur.
     4. Récupère le compte unique des utilisateurs ou utilise la valeur par défaut initiale.
     5. Définit une limite de pourcentage où ce serait le nombre de vues minimum attendu chaque jour.
     6. Sauvegarde le nombre attendu de vues et enregistre une copie locale avant de la retourner à l'appelant.

   Voici un pseudo-code pour cette fonctionnalité :

```
def track_user_increase(expected_count, current_count, threshold_percent):
    default_count = 1000  # Valeur par défaut élevée
    expected_value = expected_count if expected_count else default_count
    unique_user_count = current_count if current_count else expected_value

    threshold = expected_value * (threshold_percent / 100)
    if unique_user_count < threshold:
        # Logique pour gérer les vues inférieures au seuil
        pass

    # Enregistrer la valeur attendue
    save_expected_value(unique_user_count)
    return unique_user_count
```

Autres cas d'utilisation des ensembles Redis

1. **Modèle de recommandation** :

   - Il est facile de mettre en œuvre une recommandation de produit ou de publicité basée sur les actions historiques des utilisateurs. Vous pouvez créer plusieurs ensembles pour les identifiants utilisateur qui ont acheté le même produit ou vu la même publicité. Ensuite, vous pouvez suggérer un produit ou une publicité à un nouvel utilisateur en fonction de ses attributs de profil comme le sexe, l'âge, la ville en correspondance avec d'autres identifiants utilisateur, puis ce qu'ils ont acheté dans le passé ou quelles publicités ils ont cliquées.

```
SADD product_views:product1 "user1" "user2"
SADD product_views:product2 "user3" "user4"
```

2. **Suivi des adresses IP** :

   - Il est très facile de stocker des adresses IP uniques dans des ensembles pour suivre les adresses IP uniques des hôtes ayant visité votre site web, blog ou publication.

```
SADD unique_ips "192.168.1.1" "192.168.1.2"
```

   En utilisant ces ensembles Redis, vous pouvez implémenter efficacement divers cas d'utilisation et améliorer les performances de votre application.

Summary of commands

Sorted Sets Redis

Redis Sorted Sets

Comme les Hashes Redis, les ensembles triés (sorted sets) de Redis stockent plusieurs champs (appelés membres) et leurs valeurs numériques (appelées scores). La principale différence ici est que tous les membres dans les ensembles triés de Redis sont uniques et ordonnés en fonction de leurs scores.

Schéma de stockage des ensembles triés Redis

(Figure 6.10: Schéma de stockage des ensembles triés de Redis)

Vous pouvez considérer les ensembles triés comme une version superset des ensembles, car ils comportent toutes les fonctionnalités des ensembles, avec l'ajout que les ensembles triés associent chaque membre à un score numérique, permettant ainsi de les ordonner.

Cas d'utilisation : Mise en œuvre de tableaux de classement

Sur la base de la définition des ensembles triés de Redis, il est évident que la construction d'un tableau de classement utilisateur est un cas d'utilisation commun. Vous devez simplement ajouter tous les identifiants utilisateur (userIds) dans des ensembles triés Redis et mettre à jour leurs scores en fonction des points qu'ils ont gagnés.

Exemple de mise en œuvre

Construisons une application de jeu et voyons comment stocker un tableau de classement dans les ensembles triés de Redis.

Ajouter des utilisateurs avec leurs scores

Vous pouvez utiliser la commande `ZADD` pour ajouter un ou plusieurs utilisateurs avec leur score dans des ensembles triés Redis.

```
redis> ZADD game1:leaderboard 100 user:101 200 user:201
(integer) 2

redis> ZADD game1:leaderboard 90 user:301
(integer) 1
```

La commande `ZADD` retourne le nombre de membres ajoutés à l'ensemble trié.

Mettre à jour les scores des utilisateurs

Si les utilisateurs gagnent de nouveaux points, vous pouvez simplement utiliser `ZINCRBY` ou `ZADD` pour mettre à jour les scores spécifiques des utilisateurs.

```
redis> ZADD game1:leaderboard 190 user:301
(integer) 0

redis> ZRANGE game1:leaderboard 0 -1 WITHSCORES
1) "user:101"
2) "100"
3) "user:301"
4) "190"
5) "user:201"
6) "200"

redis> ZINCRBY game1:leaderboard 100 user:301
"290"

redis> ZRANGE game1:leaderboard 0 -1 WITHSCORES
1) "user:101"
2) "100"
3) "user:201"
4) "200"
5) "user:301"
6) "290"
```

Vous pouvez remarquer dans les commandes précédentes que lorsque vous utilisez `ZADD` pour mettre à jour le score d'un membre existant, il met simplement à jour la valeur du score et retourne 0 car l'élément existe déjà. De plus, lorsque vous utilisez `ZINCRBY` pour incrémenter le score d'un membre spécifique, il retourne le score nouvellement mis à jour pour ce membre.

Récupérer la liste des membres

En utilisant la commande `ZRANGE`, vous pouvez retourner la liste des membres. Ici, 0 est l'index de départ et -1 est l'index du dernier membre. Vous pouvez également modifier ces index de départ et de fin si vous souhaitez implémenter une pagination ou récupérer une plage spécifique de membres ordonnés par score. `WITHSCORES` est utilisé pour récupérer les listes de membres avec leurs scores.

```
redis> ZREVRANGE game1:leaderboard 0 -1 WITHSCORES
1) "user:301"
2) "290"
3) "user:201"
4) "200"
5) "user:101"
6) "100"
```

En utilisant `ZREVRANGE`, vous pouvez simplement lister le total des membres ou les quelques meilleurs membres en fonction de leurs scores.