Livre choisi :

Nom : Redis Deep Dive Auteur : Kulkrani Chinmay Éditeur : BPB Publications Année de publication : 2021 ISBN : 978-81-948377-6-3

url : https://univ.scholarvox.com/catalog/book/docid/88938783?searchterm=redis%20server

History of Redis

Le chapitre 3 du livre aborde l'histoire de Redis, mettant en lumière son impact significatif dans le domaine des projets open-source. Redis, créé par Salvatore Sanfilippo, également connu sous le pseudonyme Antirez, est un exemple emblématique de la manière dont une communauté dynamique peut collaborer pour développer des solutions technologiques innovantes.

Salvatore Sanfilippo, originaire de Sicile, est le créateur et le principal contributeur de la version open-source de Redis. Ce projet est né de la nécessité d'améliorer la gestion des données pour des applications requérant des accès rapides et efficaces.

Redis a non seulement transformé la gestion des données mais a aussi encouragé les développeurs à partager leurs connaissances et à contribuer activement à son développement. C'est un exemple inspirant de la manière dont une grande communauté peut œuvrer ensemble pour créer quelque chose d'utile et d'impactant pour tous.

En résumé, Redis, sous la direction de Salvatore Sanfilippo, incarne la puissance de l'open-source et illustre comment une initiative communautaire peut façonner de manière positive le paysage technologique moderne.

Salvatore Sanfilippo, connu sous le pseudonyme Antirez, a contribué de manière significative au domaine des systèmes embarqués avant de se lancer dans le développement de Redis. Il a notamment créé le langage de programmation Tcl adapté aux systèmes embarqués, connu sous le nom de JimTcl. Plus tard, attiré par l'essor du Web 2.0, Salvatore a fondé sa propre entreprise web et a réussi à conclure des accords avec Telecom Italia, le plus grand opérateur téléphonique en Italie.

L'idée de Redis est née alors que Salvatore et son équipe cherchaient à développer un produit pour des statistiques en temps réel. Ils ont rencontré des défis de performance importants avec les bases de données traditionnelles, incapables de gérer efficacement l'ingestion de données et les requêtes en temps réel. Pour résoudre ces problèmes, Salvatore a commencé à explorer des alternatives et a réalisé qu'une base de données en mémoire était nécessaire, particulièrement pour gérer des listes d'items avec des opérations très rapides. Ainsi est né le prototype initial de Redis, d'abord développé en Tcl puis traduit en langage C pour améliorer les performances.

En mars 2009, Salvatore a annoncé Redis en tant que projet open-source sur Hacker News, marquant le début de son engagement passionné dans le développement de Redis. VMware a montré un intérêt marqué pour le projet et a sponsorisé son développement, permettant à Salvatore de travailler à plein temps sur Redis pendant un an. Par la suite, il a rejoint Pivotal où Redis est devenu un élément crucial de leurs systèmes à haute performance à faible latence.

Salvatore a toujours eu pour vision de rendre Redis centré sur les développeurs, en facilitant leur travail avec une documentation claire et une utilisation intuitive. Après des années de développement continu, Redis a connu plusieurs versions majeures avec l'ajout de fonctionnalités telles que les modules API, les structures de données de flux, et le support du protocole RESP3.

Redis, c'est quoi ?

Dans la section précédente, nous avons exploré l'histoire de Redis et les efforts de Salvatore pour créer cette base de données remarquable qui a gagné une immense popularité dans la communauté des développeurs.

Redis, dans sa forme longue, est connu sous le nom de "Remote Dictionary Server" et est construit comme une base de données en mémoire. Fondamentalement, Redis est une base de données de type clé-valeur, mais ses fonctionnalités vont au-delà de cela. Comparé à d'autres bases de données en mémoire, Redis ne se limite pas seulement aux paires clé-valeur simples ; il prend en charge des structures de données plus complexes comme les listes et les ensembles. Cela permet d'effectuer des opérations avancées telles que l'ajout et la suppression d'éléments, les intersections entre ensembles, et bien plus encore.

Dans les chapitres suivants, nous approfondirons nos connaissances sur les structures de données Redis, ses performances, ses cas d'utilisation spécifiques, ainsi que des exemples concrets pour apprendre à l'utiliser efficacement.

Structures et architecture

Dans le domaine des bases de données, Redis se distingue par son architecture particulière et ses choix techniques, orientés vers la performance et la simplicité.

Architecture Fondamentale de Redis

1. **Type Clé-Valeur**:
    - Redis est essentiellement une base de données de type clé-valeur, ce qui signifie que les données sont stockées sous forme de paires clé-valeur où chaque clé est unique.
2. **Base de données en Mémoire**:

- Redis est classé comme une base de données en mémoire, ce qui implique que toutes les données sont conservées dans la mémoire principale (RAM) de l'ordinateur plutôt que sur un disque dur ou un SSD. La mémoire RAM est la forme de stockage la plus rapide, permettant des opérations de lecture et d'écriture ultra-rapides. Cependant, cela limite la quantité de données pouvant être stockée en fonction de la capacité de la RAM disponible.

3. **Écrit en C**:

    - Redis est entièrement écrit en langage C, un langage connu pour sa rapidité et sa robustesse. Cela contribue à l'efficacité et à la vitesse de Redis, en minimisant les surcouches et en permettant un accès direct au matériel.

Avantages et Désavantages des Choix d'Architecture

Avantages :

- **Vitesse Élevée** : En utilisant la RAM, Redis atteint des performances élevées pour les opérations de lecture/écriture.

- **Simplicité** : L'architecture clé-valeur simplifie la gestion des données, ce qui rend Redis facile à comprendre et à utiliser.

- **Flexibilité** : La capacité à stocker différents types de données (chaînes, listes, ensembles, etc.) permet des cas d'utilisation variés.

Désavantages :

- **Coût de la Mémoire** : La mémoire RAM est coûteuse par rapport à d'autres formes de stockage, limitant la capacité de stockage globale par rapport à des bases de données sur disque.

- **Persistance Limitée** : Bien que Redis supporte la persistance des données, cela nécessite une configuration spécifique et peut ne pas être aussi robuste que d'autres systèmes de gestion de bases de données.

- **Évolutivité Verticale** : L'évolutivité de Redis est principalement verticale (ajout de ressources à un serveur unique), ce qui peut être limitant pour des charges de travail très volumineuses.

En conclusion, Redis se distingue par sa simplicité, sa rapidité et sa flexibilité grâce à son architecture en mémoire et à son modèle de données clé-valeur. Ces caractéristiques en font un choix populaire pour les applications nécessitant une manipulation rapide de données, telles que le caching, les sessions utilisateur en temps réel, et d'autres cas où la performance est cruciale.

Redis suit un modèle d'architecture client-serveur où le serveur Redis est responsable du stockage et de la gestion des données, tandis que le client Redis s'authentifie et envoie des commandes au serveur pour exécution.

Architecture Client-Serveur de Redis

1. **Serveur Redis** :
    - Le serveur Redis est le composant central de l'architecture. Il est responsable du stockage des données en mémoire et de leur gestion. Toutes les opérations de lecture, écriture, mise à jour et suppression des données sont effectuées par le serveur Redis.
2. **Client Redis** :
    - Le client Redis est une entité externe qui communique avec le serveur Redis. Son rôle est d'authentifier l'accès au serveur et d'envoyer des commandes Redis spécifiques pour interagir avec les données stockées sur le serveur. Les clients Redis peuvent être des applications, des services ou des bibliothèques intégrées dans d'autres logiciels.
3. **Stockage en Mémoire et Caching** :
    - Redis stocke toutes les données en mémoire RAM, ce qui permet des performances extrêmement rapides pour les opérations de lecture et d'écriture. Cette caractéristique le rend similaire à un système de mise en cache, où les données sont préchargées et accessibles très rapidement.
4. **Différences par rapport à un Système de Mise en Cache Simple** :
    - Contrairement à un simple système de mise en cache, Redis offre également des mécanismes de persistance des données. Cela signifie que les données peuvent être sauvegardées sur un disque dur ou un SSD, assurant ainsi qu'elles ne sont pas perdues lorsque l'application ou le serveur Redis est arrêté ou redémarré.

En résumé, l'architecture client-serveur de Redis se distingue par sa capacité à stocker des données en mémoire avec une vitesse d'accès très élevée, tout en offrant des fonctionnalités de persistance pour assurer la durabilité des données. Cela en fait une solution idéale pour les applications nécessitant à la fois performances et fiabilité dans la gestion des données.

Mécanismes de Persistance de Redis

Redis utilise deux mécanismes principaux de persistance pour assurer la durabilité des données :

1. **Persistance des Fichiers RDB (Redis Database)** :
    - Crée des snapshots périodiques du jeu de données en mémoire et les stocke dans des fichiers RDB sur le disque.
    - Avantages : Moins d'espace disque requis par rapport à la persistance AOF, chargement plus rapide au redémarrage.
    - Inconvénients : Risque de perte de données si un échec survient entre deux snapshots.
2. **Persistance des Fichiers AOF (Append Only File)** :
    - Journalise chaque opération d'écriture reçue par le serveur dans un fichier de journal.
    - Avantages : Garantit que chaque opération d'écriture est enregistrée, minimise le risque de perte de données.
    - Inconvénients : Les fichiers AOF sont souvent plus volumineux que les fichiers RDB en raison de la journalisation continue.

Mise à l'Échelle au-delà d'une Machine Unique

Redis supporte également le clustering pour la mise à l'échelle horizontale :

- **Clustering Redis** :
    - Distribue les données sur plusieurs nœuds Redis pour gérer des ensembles de données plus importants.
    - Chaque nœud est responsable d'un sous-ensemble des données, déterminé par hachage des clés.
    - Avantages : Évite les goulets d'étranglement, offre une tolérance aux pannes et une disponibilité élevée.
    - Considérations : Nécessite une planification soigneuse de la distribution des clés pour un routage efficace des requêtes.

En résumé, Redis utilise la persistance (RDB et AOF) pour assurer la durabilité des données et prend en charge le clustering pour la mise à l'échelle horizontale, permettant de gérer des volumes de données plus importants et des besoins de performances élevés tout en garantissant la robustesse et la disponibilité des données.

Redis est un système de base de données clé-valeur où la clé doit toujours être une chaîne de caractères, mais la valeur peut prendre plusieurs formes. Voici les types de valeurs que Redis peut stocker :

1. **String** : Une simple chaîne de caractères.
2. **List** : Une collection ordonnée de chaînes de caractères.
3. **Set** : Une collection non ordonnée de chaînes de caractères uniques.
4. **Hash** : Un ensemble de champs associés à leurs valeurs.
5. **Sorted Set** : Un ensemble ordonné de chaînes de caractères uniques, trié selon une valeur numérique associée à chaque élément.

Faits Rapides sur Redis :

- **Stockage** : 1 million de chaînes dans Redis occupent environ 85 MB.
- **Mémoire** : Une instance Redis vide utilise environ 3 MB de mémoire.
- **Taille des valeurs** : Les valeurs Redis sont de 64 bits.
- **Nombre de clés** : Théoriquement, Redis peut gérer jusqu'à 2^32 clés dans une instance.

Ces caractéristiques font de Redis un choix polyvalent pour divers cas d'utilisation, allant du cache rapide à des structures de données complexes comme les ensembles et les listes ordonnées.

Structures de Données de Base :

Chaînes (Strings) :

Permet de stocker des données génériques jusqu'à 512 MB par chaîne. Utilisation des Strings : Utile pour stocker des objets JSON ou d'autres données sérialisées, bien que cela limite les opérations directes sur les objets stockés dans Redis. Redis offre ainsi une flexibilité considérable dans la gestion des données, en permettant de choisir le type de valeur le plus adapté à chaque besoin d'application.

Hashes :

Les Hashes dans Redis fonctionnent de manière similaire aux objets à structure simple en JavaScript. Un Hash dans Redis permet de mapper une clé de type chaîne à une valeur de type chaîne.

Utilisation des Hashes Redis :

- **Idéal pour les objets simples** : Les Hashes sont particulièrement adaptés pour stocker des objets peu profonds. Par exemple :
    - Sessions utilisateur.
    - Profils utilisateurs.
    - Informations de visiteurs.

Les Hashes permettent de structurer les données de manière organisée et efficace, offrant un moyen pratique de stocker et de manipuler des données qui peuvent être facilement représentées sous forme de paires clé-valeur.

En résumé, Redis Hashes sont bien adaptés pour stocker des structures de données simples et peu profondes, ce qui les rend idéaux pour de nombreux cas d'utilisation courants dans les applications.

Listes :

Les listes Redis sont similaires à des tableaux de chaînes de caractères où les éléments sont triés dans l'ordre d'insertion. Voici quelques points importants sur leur architecture sous-jacente :

Caractéristiques des listes Redis :

1. **Implémentation avec des listes chaînées** : Les listes Redis sont implémentées à l'aide de listes chaînées, ce qui leur donne les propriétés des listes chaînées. Cela signifie que les opérations en tête (head) et en queue (tail) sont constantes en temps d'exécution, peu importe la taille de la liste.
2. **Accès par index** : L'accès à un élément par son index dans une liste Redis est coûteux en temps car les listes Redis ne sont pas optimisées pour un accès direct par position. Dans de tels cas, il serait préférable d'utiliser des ensembles triés (sorted sets) Redis.
3. **Capacité de stockage** : Une liste Redis peut théoriquement stocker jusqu'à 2^32 - 1 éléments.

Utilisations courantes des listes Redis :

- **Implémentation de files d'attente** : Les listes Redis sont idéales pour implémenter des structures de données de type file d'attente où les éléments sont ajoutés à la fin (queue) et retirés du début (head).
- **Applications dans les réseaux sociaux** : Des plateformes comme Twitter utilisent des listes Redis pour stocker les timelines des utilisateurs et les flux de page d'accueil.

- **Gestion de tâches asynchrones** : Les tâches peuvent être ajoutées dans des listes et traitées dans l'ordre où elles ont été ajoutées, ce qui est particulièrement utile pour les systèmes de traitement de données en lot ou asynchrones.

En résumé, Redis listes sont bien adaptées pour gérer des données structurées séquentiellement, telles que des files d'attente et des listes de tâches, en fournissant des opérations efficaces pour l'ajout, la suppression et l'itération des éléments.

Ensembles (Sets) :

Les ensembles (Sets) dans Redis sont similaires aux listes car ils stockent une collection de chaînes de caractères, mais avec des différences clés : les éléments ne sont pas ordonnés et chaque élément est unique. Voici quelques points importants sur les ensembles Redis :

Caractéristiques des ensembles Redis :

- **Unicité des éléments** : Les ensembles Redis garantissent que chaque élément est unique, ce qui signifie que vous ne pouvez pas avoir plusieurs occurrences du même élément.

- **Opérations efficaces** : Outre l'ajout et la suppression d'éléments, les ensembles Redis supportent des opérations telles que l'union, l'intersection et la différence, toutes exécutées côté serveur et donc très rapides.

Utilisations recommandées des ensembles Redis :

- **Stockage d'entités uniques** : Les ensembles Redis sont idéaux pour stocker des ensembles d'éléments uniques où l'ordre n'a pas d'importance et la vérification de l'existence d'un élément est une opération fréquente.

- **Exemples d'utilisation** : Ils sont souvent utilisés pour enregistrer des ensembles d'utilisateurs uniques qui ont visité un site web, pour gérer des listes d'identifiants uniques dans une application, etc.

En résumé, les ensembles Redis sont une solution efficace pour gérer des collections d'éléments uniques et pour effectuer des opérations ensemblistes rapides et efficaces côté serveur.

Ensembles Triés (Sorted Sets) :

Les ensembles triés (Sorted Sets) dans Redis sont similaires aux ensembles, mais chaque élément est associé à un score numérique qui est utilisé pour trier les éléments. Voici quelques points importants sur les ensembles triés Redis :

Caractéristiques des ensembles triés Redis :

- **Ordre basé sur les scores** : Chaque élément dans un ensemble trié a un score associé, et les éléments sont ordonnés en fonction de ces scores. Les scores peuvent être des nombres entiers ou à virgule flottante, et plusieurs éléments peuvent avoir le même score.

- **Assignation des scores** : Redis ne définit pas les scores par défaut pour les éléments. C'est à l'utilisateur de définir et de manipuler les scores en fonction des besoins spécifiques de l'application.

- **Complexité des opérations** : Les opérations d'insertion, de suppression et de recherche dans un ensemble trié ont une complexité O(log(N)), où N est le nombre d'éléments dans l'ensemble. Cela rend les ensembles triés adaptés aux cas d'utilisation où l'accès rapide à des membres spécifiques en fonction de leur score est crucial.

Utilisations recommandées des ensembles triés Redis :

- **Tableaux de classement (Leaderboards)** : Idéal pour stocker les noms des meilleurs scores ordonnés par leurs scores respectifs.

- **Listes de questions et réponses (Q&A)** : Utilisé dans les plateformes de Q&R pour trier les questions en fonction des votes ou des notes attribuées.

Exemples courants d'utilisation :

- **Classements** : Par exemple, un jeu peut utiliser un ensemble trié pour suivre et afficher les meilleurs scores des joueurs.

- **Plateformes de Q&R** : Comme Stack Overflow, où les questions sont affichées en fonction des votes et des réponses.

En résumé, les ensembles triés Redis sont particulièrement adaptés lorsque vous avez besoin de maintenir un ordre spécifique parmi des éléments tout en permettant un accès rapide aux membres en fonction de leur score.