redislabs
HOME OF REDIS

LIBRO BLANCO

Usos populares de Redis para principiantes

Dave Nielsen

CONTENTS

Introducción a Redis

Redis es un almacenamiento de estructuras de datos en memoria de tipo NoSQL que se suele utilizar como base de datos, caché e intermediario de mensajería. Al contrario que otros almacenamientos en memoria, puede conservar sus datos en disco, y es notablemente versátil gracias a una amplia variedad de estructuras de datos (Sets, Sorted Sets, Hashes, Lists, Strings, Bit Arrays, HyperLogLogs, Geospatial Indexes). Los comandos de Redis permiten a los desarrolladores realizar operaciones de alto rendimiento en esas estructuras con muy poca complejidad. Dicho de otra forma, Redis está diseñado con vistas al rendimiento y la sencillez.

En distintas referencias recientes se ha observado que Redis era la base de datos NoSQL de mayor rendimiento, con hasta 8 veces la productividad y hasta el 80% menos de latencia que otras bases de datos distintas de NoSQL. Redis también se ha establecido como referencia a 1,5 millones de operaciones/segundo a latencias inferiores al milisegundo al tiempo que se ejecutaba en una sola y mera instancia en la nube, lo que supone la menor cantidad de hardware con respecto a otras bases de datos NoSQL. Lea sobre los puntos de referencia de Redis aquí y aquí.

Este libro blanco ofrece una visión general de algunos casos de uso populares para ayudarle a empezar con Redis.

¿Qué significa 'almacenamiento de estructura de datos'?

La mayoría de las bases de datos NoSQL almacenan datos en un solo almacén de datos sin necesidad de un esquema SQL. Por ejemplo, MongoDB almacena la mayoría de sus datos en una estructura de documento, Cassandra en un almacén de datos en forma de columna, los datos de Couchbase son en su mayoría clave/valor con algunas capacidades de almacenamiento de datos de documentos, y Memcached almacena los datos en una estructura de cadena simple. Redis tampoco requiere un esquema SQL, pero lo que hace a Redis diferente es que tiene muchas estructuras de datos de las que elegir.

Estas estructuras de datos están muy optimizadas. Cada una ofrece comandos especializados que ayudan a ejecutar funciones complejas sin esfuerzo y con muy poco código. Simplifican la intrincada funcionalidad, que normalmente habría requerido docenas o cientos de líneas de código y habría consumido un considerable ancho de banda de red para su implementación. Estas estructuras de datos hacen que Redis sea extremadamente potente y permiten que las aplicaciones basadas en Redis manejen volúmenes extremos de operaciones a muy baja latencia.

Las estructuras de datos de Redis son:

- Strings
- Hashes
- Lists
- Sets
- Sorted Sets
- Bit Arrays
- HyperLogLogs
- Geospatial Indexes

Las estructuras de datos de Redis pueden combinarse como bloques de construcción de Lego para formar estructuras de datos más complejas. Y los comandos de Redis les dan un poder increíble. Ahora hablemos de algunos casos de uso populares que permiten estas estructuras de datos.

Usos populares de Redis

Redis se usa a menudo para:

- Gestión de datos de sesión de usuario
- Análisis en tiempo real como contadores/líderes/más vistos/rangos más altos-más bajos
- Recomendaciones de compra o de artículos basadas en características de perfil comunes
- Colas de mensajes para el flujo de trabajo y otros trabajos
- El almacenamiento en caché de datos estáticos e interactivos

Redis también se utiliza para transacciones de alta velocidad, búsquedas geoespaciales, datos de series temporales, bloqueo distribuido y muchos otros casos de uso. Hablaremos de cómo las estructuras de datos permiten varios de estos diversos usos; hay un enlace a otros recursos al final de este artículo.

Redis para gestión de sesiones de usuarios

Imagine que utilizan su sitio web miles o millones de usuarios todos los días. Es necesario almacenar y recuperar rápidamente la información de usuario y de uso, como el nombre, la dirección URL de la foto del perfil, las páginas visitadas, los elementos vistos, el tiempo dedicado a cada página y otros detalles para personalizar la experiencia de cada usuario. Necesitará almacenar estos datos en múltiples variables de sesión y su aplicación web deberá poner esta información a disposición de varios servidores de baja latencia para cumplir con los tiempos de respuesta que sus usuarios esperan.

La forma tradicional de almacenar la información de la sesión es en el nivel de aplicación. Pero esto se queda corto porque requiere un equilibrador de cargas por sesión por delante de su web/app-tier con el fin de reenviar la petición al servidor correcto. Esta estrategia es: 1) compleja, requiere una configuración especial del equilibrador de cargas; 2) no es eficiente, el equilibrador de cargas deja de equilibrar la carga y solo reenvía las solicitudes al servidor web/app que almacena la sesión; 3) no es seguro ya que una vez que el servidor web/app falla se pierde toda la información de la sesión.

Otra opción es que si mantiene todos sus datos de sesión en una base de datos compartida y fiable a la que se pueda acceder desde cada servidor web o de aplicaciones y que admita operaciones de alto rendimiento y baja latencia, su aplicación puede funcionar con seguridad con un volumen alto de usuarios y operaciones.

El uso de bases de datos NoSQL basadas en disco para los datos de sesión introduce una latencia innecesaria, mientras que Memcached, que utiliza una mera estructura de datos de valor clave, no es capaz de ofrecer las posibilidades interactivas que ofrecen las estructuras de datos de Redis. Además, Memcached puede ser poco fiable. Se puede perder toda la caché de sesiones de usuario si el servidor falla.

Así que Redis y específicamente la estructura de datos Hash de Redis es su mejor opción. Le permite almacenar toda su información en múltiples campos, mientras que le da la flexibilidad de utilizar otras estructuras de datos de Redis. Y no solo está disponible en la memoria de baja latencia, sino que puede tener una conmutación inmediata por causa de error en caso de que su servidor se caiga.

La estructura de datos de Redis Hash es una colección de pares clave-valor, así que podría tener un solo ID de usuario o nombre de usuario asociado a múltiples campos como vista de página, tiempo en el sitio, página actual y más. Los comandos HGET y HSET le permiten obtener y actualizar valores individuales; HMGET/HMSET le permite obtener o actualizar múltiples valores. Los campos individuales se pueden incrementar sin leer todo el Hash con comandos como HINCRBY.

Ejemplo 1: Uso de un HASH de Redis para almacenar la información de la sesión de usuario.

```
HMSET usersession:1 userid 8754 name dave ip 10.20.104.31 hits 1
OK
HMGET usersession:1 userid name ip hits
1) “8754”
2) “dave”
3) “10.20.104.31”
4) “1”
HINCRBY usersession:1 hits 1
(integer) 2
HSET usersession:1 lastpage “home”
(integer) 1
HGET usersession:1 lastpage
“home”
HDEL usersession:1 lastpage
(integer) 1
DEL usersession:1
(integer) 1
```

***Figura 1**: Salida de la consola Redis-CLI*

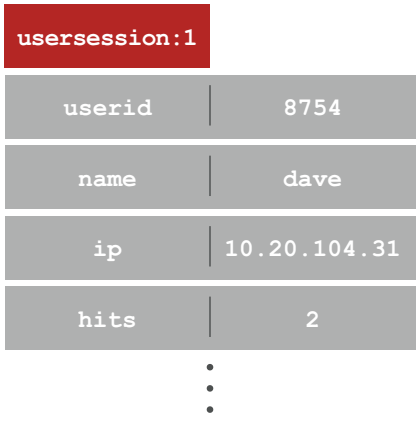

***Figura 2**: Representación de los datos de la sesión de usuario almacenados en una estructura de datos Redis Hash*

Otro aspecto positivo de Redis en esta situación es que se puede usar el comando EXPIRE de Redis para borrar los datos de las sesiones de usuario una vez que estas hayan expirado. Establece un tiempo hasta el lanzamiento (TTL) de la tecla Hash de Redis y lo borra automáticamente después de un tiempo especificado, como 20 minutos.

Redis para análisis en tiempo real

Redis es excepcionalmente bueno para los cálculos analíticos en tiempo real: las mejores puntuaciones, los contribuyentes de mayor rango, los puestos más altos y muchos más. Imagínese que tenga una aplicación distribuida como una subasta, un juego multijugador o una encuesta de usuarios en tiempo real. Muchos usuarios están respondiendo o actuando en tiempo real: aumentando las ofertas, recogiendo puntos o respondiendo con sus opiniones. Su aplicación necesita rastrear y mostrar instantáneamente la oferta más alta, la puntuación más alta o la opinión más alta. Aquí es donde el SORTED SET de Redis desempeña un papel sencillo, pero elegante.

Los Sorted Sets de Redis mantienen automáticamente las listas de puntuaciones de los usuarios actualizadas y en orden. Usará los comandos de Redis, como ZADD, para mantener las puntuaciones actualizadas. ZRANGE y ZREVRANGE le darán los valores superiores o inferiores del Sorted Set. ZRANK también sacará el rango real de un usuario en la pila. Con Sorted Sets no tiene que preocuparse por la eficiencia de la clasificación, el orden de clasificación y las otras muchas decisiones que tiene que tomar con las bases de datos tradicionales.

Ejemplo 2: Uso de SORTED SETS para análisis sencillos en tiempo real. En este ejemplo, su clave de Sorted Set es game:1 y los miembros se añaden con puntuaciones con el comando ZADD. La puntuación de id:3 aumenta en 44000 por el comando ZINCRBY y finalmente se usa ZREVRANGE para encontrar al máximo anotador.

```
ZADD game:1 10000 id:A
(integer) 1
ZADD game:1 34000 id:B
(integer) 1
ZADD game:1 340 id:3
(integer) 1
ZADD game:1 35000 id:4
(integer) 1
ZINCRBY game:1 44000 id:3
(integer) 0
ZREVRANGE game:1 0 -1
id:3
```

***Figura 3**: Game:1 se ordena automáticamente por puntuación y ZREVRANGE devuelve la mayor puntuación*

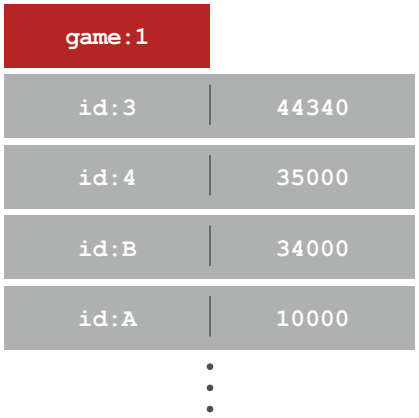

***Figura 4**: Representación de las puntuaciones en tiempo real almacenadas en una estructura de datos de Sorted Set de Redis*

Redis para recomendaciones de usuarios

Es bastante común que los sitios de venta al por menor o de comercio electrónico recomienden artículos comprados por otros que han comprado el mismo artículo. Lo mismo ocurre con la recomendación de artículos que tienen las mismas etiquetas del artículo que está leyendo. También aquí se puede usar Redis para encontrar compras o artículos comunes con facilidad y sencillez.

Los SETS de Redis son colecciones de cadenas sin ordenar, pero lo que los hace verdaderamente potentes son los comandos Set de Redis. Con la mayoría de las bases de datos, comparar más de 2 tablas o colecciones daría problemas de rendimiento. Con Redis, se pueden crear varios Sets con el comando SADD, y luego usar SINTER para encontrar los miembros de la intersección de todos los Sets señalados. También puede usar SISMEMBER para determinar si se puede encontrar un valor en particular en un Set y SMEMBERS o SSCAN (recomendado para sets grandes) para devolver todos los miembros de un Set.

En el caso de recomendación de artículos mencionado anteriormente, puede identificar fácilmente los artículos que se recomendarán mediante los Sets de Redis. Para preparar los datos, se utiliza SADD para añadir cada artículo a uno o más conjuntos de etiquetas. Puede haber cientos de Sets que representen cientos de etiquetas.

Una vez que sus Sets estén rellenados, está listo para calcular una recomendación. Por ejemplo: Si un usuario está viendo article:1 que ha sido etiquetado con tag:1, tag:2, tag:3 y usted quiere recomendarle nuevos artículos. Realizaría 'SINTER tag:1 tag:2 tag:3' para obtener una lista de artículos que también han sido etiquetados con las mismas tres etiquetas.

Aquí hay un ejemplo de las funciones de Redis que se utilizarían para añadir etiquetas a los artículos y luego encontrar los artículos etiquetados con tag:1, tag:2 y tag:3.

Ejemplo 3: Utilice SADD para añadir etiquetas a cada artículo a medida que se crean, SMEMBERs para encontrar todas las etiquetas que tiene un artículo. Si quiere encontrar artículos que tienen etiquetas comunes, crearía un conjunto ordenado por etiqueta y luego usaría SINTER para encontrar artículos que tienen múltiples etiquetas comunes

```
SADD article:3 tag:3 tag:2 tag:1
(integer) 3
SMEMBERS article:3
1) “tag:1”
2) “tag:2”
3) “tag:3
SADD tag:1 article:3 article:1
(integer) 2
SADD tag:2 article:22 article:14 article:3
(integer) 3
SADD tag:3 article:9 article:3 article:2
(integer) 3
SISMEMBER article:3 tag:1
(integer) 1
SINTER tag:1 tag:2 tag:3
1) “article:3”
```

***Figura 5**: Salida de la consola para el ejemplo 3*

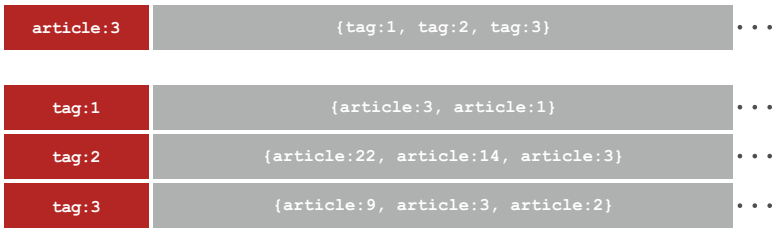

***Figura 6**: Representación de artículos y etiquetas almacenados en un conjunto de estructuras de Set de Redis*

Redis para la gestión de las colas de trabajo

En una encuesta reciente a usuarios, encontramos que el uso de Redis para las colas era un caso de uso muy común. Muchos sistemas de gestión de trabajo populares como Resque, Celery y Sidekiq emplean Redis entre bastidores. Un flujo de trabajo, con tareas que se tienen que procesar secuencialmente, se puede manejar fácilmente con las Lists de Redis. Por ejemplo, su aplicación podría enviar ofertas de usuarios a un sistema de subastas que también tiene que enviar notificaciones por correo electrónico de procesamiento intensivo; o su aplicación podría aceptar un pedido de múltiples productos que también tiene que procesar transacciones con tarjeta de crédito; o su aplicación podría enviar los acontecimientos de la aplicación web a un sistema de supervisión remoto. La forma más eficaz de diseñar su aplicación es aprovechar una cola para almacenar estas tareas de forma que los procesos de los trabajadores las realicen de forma asíncrona.

Las Lists de Redis son colecciones de cadenas ordenadas por orden de inserción, o como se especifique. Por ejemplo, LPUSH añadirá un valor al final «izquierdo» de una List, al que normalmente se denomina principio o parte superior de la Lista. Y RPUSH añadirá el valor al final «derecho» de la lista, al que normalmente se denomina final o fondo de la lista. El RPOP eliminará el último elemento de la lista, mientras que el LPOP eliminará el primer elemento. RPOPLPUSH, por divertido que suene, es una manera eficiente de hacer «saltar» un elemento del extremo derecho de una List y «empujarlo» al extremo izquierdo de otra List.

Ejemplo 4: En el siguiente ejemplo, usamos LPUSH para añadir tareas «naranja», «verde» y «azul» al lado izquierdo (es decir, la parte superior o el principio) de la queue:1. Luego usamos RPUSH para añadir la tarea «roja» al lado derecho (es decir, el fondo o el final) de la misma cola. Entonces usamos RPOPLPUSH para quitar simultáneamente la tarea «roja» del lado derecho de la queue:1 y empujarla al lado izquierdo de la queue:2.

```
LPUSH queue:1 orange
(integer) 1
LPUSH queue:1 green
(integer) 2
LPUSH queue:1 blue
(integer) 3
RPUSH queue:1 red
(integer) 4
RPOPLPUSH queue:1 queue:2
```

***Figura 7**: Salida de la consola de los comandos de la List LPUSH, RPUSH y RPOPLPUSH*

***Figura 8**: Representación de las tareas añadidas a la queue:1 almacenadas en un Set de Redis antes de RPOPLPUSH*

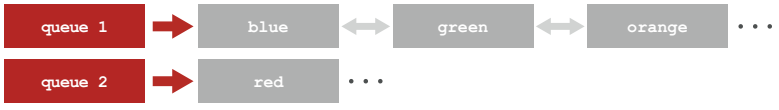

***Figura 9**: Representación de la tarea roja después de trasladarla a la queue:2 con RPOPLPUSH*

Redis para almacenamiento en caché

El almacenamiento en caché es un caso de uso muy conocido de Redis. Lo encontrará entre los primeros casos de uso cuando empiece con Redis. El almacenamiento en caché es una técnica común que se emplea para poder ampliar los sistemas de alto rendimiento. El principio detrás del almacenamiento en caché es sencillo: en lugar de mantener todos sus datos en una base de datos central, se traslada una copia de los datos a los que accede frecuentemente al servidor de Redis para aliviar el cuello de botella número uno de su aplicación: la conexión con su base de datos.

Un nivel de caché es un patrón de diseño de sistema distribuido que ofrece muchas ventajas. Una vez que se ha separado la caché de la aplicación, esta no sólo es más independiente, sino que puede compartir los datos en caché entre los servidores y con otras aplicaciones y (micro)servicios. Algunos casos populares de uso de la memoria caché son:

- Almacenamiento en caché de páginas web estáticas y fragmentos de HTML para reducir la carga de servicio del servidor web y/o de la aplicación
- Activos de páginas de caché (fragmentos de código, archivos multimedia) que se utilizan junto con CDN y sistemas de almacenamiento rápido
- Guardar en caché la información de la sesión de usuario, como se mencionó anteriormente, para prácticamente todo tipo de aplicaciones, SOA y marco de microservicios
- Almacenamiento en caché de objetos para perfiles de usuario y datos de gran actividad (que se actualizan y leen con frecuencia)
- Llamadas a la API y marcas de características de la caché y los mecanismos de aceleración
- Almacenamiento de los resultados de las consultas en las bases de datos

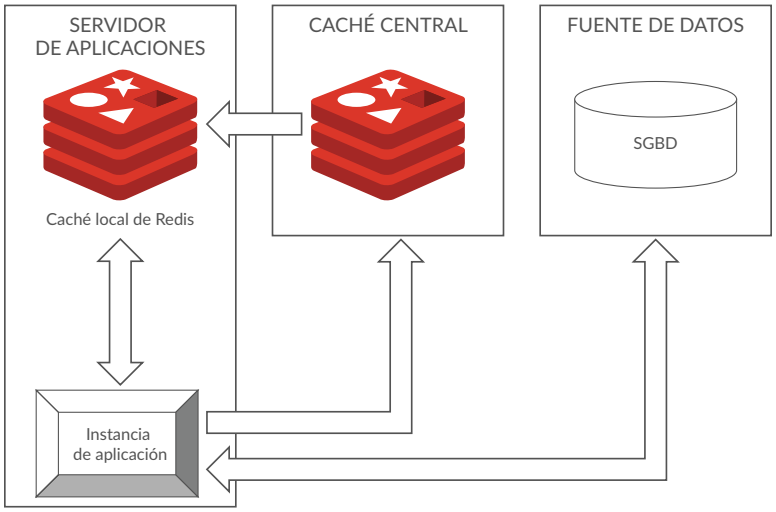

***Figura 10**: Se puede configurar Redis en el mismo servidor que su aplicación, o bien se puede configurar como caché central. Para que disponibilidad sea elevada se suele utilizar un caché central.*

¿Qué vendrá en el futuro?

Redis se usa popularmente como base de datos, caché e intermediario de mensajes. Las estructuras de datos en Redis son similares a algunos modelos de datos de otras bases de datos NoSQL, salvo por que se implementan en memoria con los propósitos simultáneos de alto rendimiento, eficiencia elevada en el espacio de memoria y poco tráfico de red de aplicaciones. Las estructuras de datos ofrecen una flexibilidad inigualable de casos de uso que hacen de Redis una solución extremadamente popular para distintas situaciones de aplicación a escala web.

Para conocer más recursos sobre cómo usar Redis, visualice algunos de nuestros seminarios web grabados o lea el popular libro de Redis del Dr.Josiah Carlson, «Redis in Action» (Redis en acción), disponible en https://redislabs.com/resources/ebook/.

700 E El Camino Real, Suite 250
Mountain View, CA 94040
(415) 930-9666
redislabs.com