Redis

Definição

O Redis é um banco de dados NoSQL escrito em C. O Remote Dictionary Server é um banco de dados chave-valor cujo o armazenamento é feito na memória, fazendo com que a escrita e a leitura sejam rápidos.

Mas qual a diferença entre ele o cache? O que acontece quando o banco cai? Haverá perda total dos dados?

Características

- é de código aberto
- permite tempos de respostas em consultas inferiores a um milissegundo

Redis vs Demais Banco de Dados chave-valor

Redis é uma evolução dos demais bancos de dados chave-valor, pois os valores podem armazenar tipos de dados mais complexos com operações atômicas definidas sob estes dados.

Redis é um banco de dados em memória mas que também é persistido no disco rígido, uma vez que representa um diferente ***trade-off*** que comabina uma alta velocidade de escrita e leitura através da limitação do conjunto de dados que não podem ser maiores que a memória.

Outra vantagem de bancos de dados em memória é que a representação de estruturas de dados mais complexos torna-se mais simples de manipular comparada a mesma estrutura em disco. Assim, REDIS pdoe fazer muito com uma pequena complexidade interna.

Casos de uso populares do Redis

Armazenamento em cache

O Redis é uma excelente escolha para a implementação de caches de memória altamente disponíveis para diminuir a latência de acesso aos dados, aumentar a produtividade e aliviar a sobrecarga de aplicativos e bancos de dados relacionais ou NoSQL. O Redis pode fornecer itens frequentemente solicitados com um tempo de resposta inferior a um milissegundo. Além disso, pode escalar facilmente para processar cargas maiores sem necessidade de aumentar o back-end de alto custo. Entre os exemplos mais comuns de armazenamento em cache com o Redis, estão resultados de consultas de banco de dados, sessões persistentes, páginas web e objetos frequentemente usados como imagens, arquivos e metadados.

Chat, sistemas de mensagens e filas

O Redis oferece suporte a Pub/Sub com correspondência de padrões e a várias estruturas de dados como listas, conjuntos ordenados e hashes. Assim, o Redis pode oferecer suporte a salas de chat de alta performance, streams de comentários em tempo real, feeds de mídia social e intercomunicação de servidores. A estrutura de dados Redis List facilita a implementação de uma fila leve. As listas oferecem operações atômicas e recursos de bloqueio e são adequadas para vários aplicativos que exigem um agente de mensagens ou uma lista circular confiável.

Placares de jogos

O Redis é uma escolha comum de desenvolvedores de jogos que precisam criar placares em tempo real. Basta usar a estrutura de dados Sorted Set do Redis que disponibiliza a especificidade de elementos enquanto mantém a lista classificada de acordo com suas pontuações. A criação de uma lista classificada em tempo real é tão fácil quanto atualizar a pontuação de um usuário toda vez que ela muda. Você também pode usar conjuntos classificados para processar dados de séries temporais usando time stamps como pontuação.

Armazenamento de sessões

O Redis é um datastore na memória com alta disponibilidade e persistência, escolhido frequentemente por desenvolvedores de aplicativos para armazenar e gerenciar dados de

sessão para aplicativos na escala da Internet. O Redis oferece a latência inferior a um milissegundo, a escala e a resiliência necessárias para gerenciar dados de sessão como perfis de usuário, credenciais, estados de sessão e personalizações específicas de usuários.

Streaming de mídia avançada

O Redis oferece um datastore rápido na memória para viabilizar casos de uso de streaming ao vivo. O Redis pode ser usado para armazenar metadados sobre perfis de usuários, visualização de históricos e informações/tokens de autenticação para milhões de usuários, bem como armazenar arquivos manifesto para possibilitar que CDNs façam streaming de vídeo para milhões de usuários de dispositivos móveis e desktops ao mesmo tempo.

Dados geoespaciais

O Redis oferece estruturas e operadores de dados na memória para uso específico, o que permite gerenciar em tempo real dados geoespaciais em grande escala e velocidade. Comandos como GEOADD, GEODIST, GEORADIUS e GEORADIUSBYMEMBER para armazenar, processar e analisar dados geoespaciais em tempo real facilitam e agilizam o uso de recursos geoespaciais com o Redis. Você pode usar o Redis para adicionar aos aplicativos recursos baseados em localização como tempo de percurso, distância do percurso e pontos de interesse.

Machine Learning

Aplicativos modernos e orientados a dados exigem machine learning para processar rapidamente um grande volume e variedade de dados e automatizar a tomada de decisões. Para casos de uso como detecção de fraudes em jogos e serviços financeiros, fazer ofertas em tempo real em tecnologia de anúncios e matchmaking para encontros e transporte solidário, a capacidade de processar dados ao vivo e tomar decisões em dezenas de milissegundos é fundamental. O Redis oferece um datastore ágil na memória para criar, treinar e implantar rapidamente modelos de Machine Learning.

Análise em tempo real

O Redis pode ser usado com soluções de streaming como Apache Kafka e Amazon Kinesis, atuando como datastore na memória para consumir, processar e analisar dados em tempo

real com latência inferior a um milissegundo. O Redis é uma escolha ideal para casos de uso de análises em tempo real, como análises de mídia social, direcionamento de anúncios, personalização e IoT.

Conexão

Os comandos REDIS são usados, juntamente, em um servidor REDIS.

Para rodar comandos em um servidor REDIS, você precisa de um ***client***. O redis-cli é um cliente incorporado no próprio pacote de instalação do servidor REDIS.

Para iniciar o redis-cli, abra o terminal e digite redis-cli. Este comando irá conectá-lo ao servidor local e partir deste momento você poderá rodar qualquer comando.

```
$redis-cli
redis 127.0.0.1:6379>
redis 127.0.0.1:6379> PING
PONG
```

No exemplo acima, nós conectamos a um servidor REDIS rodando na máquina local e executamos o comando PING que checa se o servidor está rodando corretamente ou não.

Conexão e Segurança

Redis pode ser seguro, uma vez que qualquer cliente necessita se autenticar antes de executar algum comando. Para tornar o REDIS seguro, você precisa atribuir uma senha ao arquivo de configuração.

```
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) ""
```

Por ***default***, esta propriedade está em branco, o que significa que não há nenhuma senha definida nesta instância do REDIS. Se assim desejar, você pode alterar esta propriedade através da execução do seguinte comando:

```
127.0.0.1:6379> CONFIG set requirepass "tutorialspoint"
OK
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) "tutorialspoint"
```

Depois de definir a senha, se qualquer cliente executar o comando sem autenticação, será retornado um erro. Para evitar este erro, será preciso que o cliente autentique seu acesso através do comando AUTH:

```
127.0.0.1:6379> AUTH password
Example
127.0.0.1:6379> AUTH "tutorialspoint"
OK
127.0.0.1:6379> SET mykey "Test value"
OK
127.0.0.1:6379> GET mykey
"Test value"
```

Conexão em um Servidor REDIS Remoto

Para executuar comandos em um servidor REDIS remoto, você precisa se conectar a este servidor através do mesmo cliente ***redis-cli***.

```
$ redis-cli -h host -p port -a password
```

Este exemplo abaixo mostra como se conectar ao um servidor REDIS remoto, usando seu host, sua porta e uma senha.

```
$ redis-cli -h 127.0.0.1 -p 6379 -a "mypass"
redis 127.0.0.1:6379>
redis 127.0.0.1:6379> PING
PONG
```

Tipos

Ao contrário de outros datastores de chave-valor que oferecem estruturas de dados limitadas,

o Redis tem uma grande variedade de estruturas de dados para atender às necessidades de suas aplicações. Os tipos de dados do Redis incluem:

- **Strings** – dados em texto ou binários com tamanho de até 512 MB
- **Listas** – uma coleção de strings na ordem em que foram adicionadas
- **Conjuntos** – uma coleção não ordenada de strings com a capacidade de fazer a intersecção, união e diferenciação de outros tipos de conjunto
- **Conjuntos ordenados** – conjuntos ordenados por um valor
- **Hashes** – uma estrutura de dados para armazenar uma lista de campos e valores
- **Bitmaps** – um tipo de dados que oferece operações de nível de bits
- **HyperLogLogs** – uma estrutura de dados probabilística para estimar os itens únicos em um conjunto de dados
- **Transmissões** – uma fila de mensagens de estrutura de dados de log
- **Dados geoespaciais** – um mapa de registros com base em longitude/latitude
- **JSON** - um objeto aninhado e semiestruturado de valores nomeados que suportam números, strings, booleanos, matrizes e outros objetos

Strings

Criar uma chave e armazenar um valor:

```
$ redis-cli
> set mykey somevalue
OK
> get mykey
"somevalue"
```

Incrementar uma chave:

```
> set counter 100
OK
> incr counter
(integer) 101
> incr counter
(integer) 102
> incrby counter 50
(integer) 152
```

Salvar várias chaves no mesmo comando (MSET):

```
> mset a 10 b 20 c 30
OK
```

Retornar várias chaves no mesmo comando (MGET):

```
> mget a b c
1) "10"
2) "20"
3) "30"
> get a
"10"
```

Testar se existe uma chave:

```
> set mykey hello
OK
> exists mykey
(integer) 1
> del mykey
(integer) 1
> exists mykey
(integer) 0
```

Obter o tipo de uma chave:

```
> set mykey x
OK
> type mykey
string
> del mykey
(integer) 1
> type mykey
none
```

Expiração de uma Chave (*Key expiration*):

```
> set key some-value
OK
> expire key 5
(integer) 1
> get key (immediately)
"some-value"
> get key (after some time)
(nil)
```

Testar tempo restante de uma chave (*ttl*):

```
> set key 100 ex 10
OK
> ttl key
(integer) 9
```

O exemplo acima cria uma chave (***key***) com uma ***string*** com valor 100 que irá expirar em 10 segundos. O commando ***ttl*** é chamado com objetivo de checar o tempo restante de vida desta chave.

+ Alguns Comandos:

DEL <key>
Este comando delete uma chave, se existe.

EXISTS <key>
Este comando checa se a chave (key) existe ou não.

EXPIRE <key> <seconds>
Define o tempo que uma determinada chave era expirar.

EXPIREAT <key> <timestamp>
Define o tempo que uma determinada chave era expirar (formato Unix Timestamp).

PEXPIRE <key> <milliseconds>
Define o tempo que uma determinada chave era expirar (milisegundos).

KEYS <pattern>
Encontra todas as chaves que respeitam um padrão (***pattern***) pelo usuário.

MOVE <key> <db>

Move uma chave (key) para outro banco de dados (db).

PERSIST <key>
Remove o período de experiração de uma chave (key).

PTTL <key>
Retorna o tempo restante para expirar uma chave (key) em milisegundos.

TTL <key>
Retorna o tempo restante para expirar uma chave (key).

RANDOMKEY
Retorna um chave aleatória.

RENAME <key> <newkey>
Muda o nome de uma chave (key) para um novo nome (newkey).

RENAMENX <key> <newkey>
Renomeia a chave (key), se a nova chave (newkey) não existe.

TYPE <key>
Retorna o tipo do dado armazenado em uma chave (key).

GETRANGE <key> <start> <end>
Retorna uma parte de uma ***string*** armazenada em um chave (key).

SETEX <key> <seconds> <value>
Define no mesmo campo o valor de um chave (key) e seu tempo de expiração.

SETNX <key> <value>
Define o valor de uma chave (key), somente se esta chave não existe.

SETRANGE <key> <offset> <value>
Sobrescreve parte de uma ***string*** de uma chave (key) começando por uma determinada posição (offset).

STRLEN <key>
Retorna o tamanho (quantidade caracteres) do valor armazenado em uma chave (key).

MSETNX <key> <value> [<key> <value>...]

Define múltiplas chaves para múltiplos valores somente se nenhuma das chaves existe.

INCR <key>
Incrementa o valor inteiro de uma chave (key)

INCRBY <key> <increment>
Incrementa o valor inteiro de uma chave com um determinado valor incremental (increment).

DECR <key>
Decrementa o valor inteiro de uma chave.

DECRBY <key> <decrement>
Decrementa o valor inteiro de uma chave em um determinado valor de decremento (decrement).

APPEND <key> <value>
Acrescenta um string em uma chave (key).

Listas

LPUSH adiciona um novo elemento à lista (***head***) e **RPUSH** adiciona um novo elemento à lista (***tail***). Finalmente, o comando **LRANGE** extraí um intervalo (***range***) de elementos da lista:

```
> rpush mylist A
(integer) 1
> rpush mylist B
(integer) 2
> lpush mylist first
(integer) 3
> lrange mylist 0 -1
1) "first"
2) "A"
3) "B"
```

Como pode ser visto, RPUSH adiciona elementos à direita da lista, enquanto o comando LPUSH adiciona elementos à esquerda.

É possível adicionar vários elementos em uma mesma chamada de comando:

```
> rpush mylist 1 2 3 4 5 "foo bar"
(integer) 9
> lrange mylist 0 -1
1) "first"
2) "A"
3) "B"
4) "1"
5) "2"
6) "3"
7) "4"
8) "5"
9) "foo bar"
```

Uma importante operação em listas é a habilidade de remover elementos com a operação ***pop***. Pop é a operação que retorna o elemento de uma lista e, ao mesmo tempo, o elimina desta lista.

```
> rpush mylist a b c
(integer) 3
> rpop mylist
"c"
> rpop mylist
"b"
> rpop mylist
"a"
```

Hashes

O comando **HSET** cria um hash com múltiplos campos, enquanto o comando **HGET** recupera um simples campo. O comando **HMGET** é similar ao comando **HGET** mas retorna um ***array*** de valores:

```
> hset user:1000 username antirez birthyear 1977 verified 1
(integer) 3
> hget user:1000 username
"antirez"
> hget user:1000 birthyear
"1977"
> hgetall user:1000
1) "username"
2) "antirez"
3) "birthyear"
4) "1977"
5) "verified"
6) "1"
```

O comando **HINCRBY** permite executar operações em campos individuais:

```
 > hincrby user:1000 birthyear 10
(integer) 1987
> hincrby user:1000 birthyear 10
(integer) 1997
```

Sets

Os Sets no REDIS são coleções de ***strings*** não ordenadas. O comando **SAD** adiciona novos elementos ao conjunto (**set**). Isto também permite fazer uma grande quantidade de operações nestes conjuntos como testar se um dado elemento existe, interseções, união ou diferença entre múltiplos conjuntos:

```
> sadd myset 1 2 3
(integer) 3
> smembers myset
1. 3
2. 1
3. 2
```

O comando ***sismember*** testa se um valor pertence ao um determinado conjunto:

```
> sismember myset 3
(integer) 1
> sismember myset 30
(integer) 0
```

Mais um exemplo:

```
redis 127.0.0.1:6379> sadd tutoriallist redis
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist mongodb
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist rabitmq
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist rabitmq
(integer) 0
redis 127.0.0.1:6379> smembers tutoriallist

1) "rabitmq"
2) "mongodb"
3) "redis"
```

Sorted Sets

Sorted Sets são um tipo de dado similar a um mix entre os tipos Set e Hash. São conjuntos ordenados compostos de únicos e não-repetidos ***strings*** de elementos já que em alguns casos é interessante ordenar um conjunto.

Entretanto, enquanto elementos dentro do conjunto ***Set*** não são ordenados, cada elemento em um conjunto ordenado é associados a valor de ponto flutuante, chamado de escore (isto é porque o otipo é também similar a um hash, onde cada elemento é mapeado em um valor).

```
redis 127.0.0.1:6379> zadd tutoriallist 0 redis
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 mongodb
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 rabitmq
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 rabitmq
(integer) 0
redis 127.0.0.1:6379> ZRANGEBYSCORE tutoriallist 0 1000

1) "redis"
2) "mongodb"
3) "rabitmq"
```

Restauração e Backup

Backup

O comando SAVE é usado para criar um backup do banco de dados corrente.

```
$ redis-cli
127.0.0.1:6379> keys *
1) "c"
2) "a"
3) "b"
4) "igor"
127.0.0.1:6379> save
OK
127.0.0.1:6379> CONFIG get dir
1) "dir"
2) "/var/lib/redis"
```

```
$ sudo su
root@vostro3360:/home/iapereira# cd /var/lib/redis/
root@vostro3360:/var/lib/redis# ls
dump.rdb
```

Restauração

Para restaurar um dump, mova o arquivo de backup (***dump.db***) para o diretório do REDIS e inicie o servidor. Para saber qual é o diretório usado pelo REDIS para o armazenamento dos arquivos de dump, use o comando CONFIG:

```
127.0.0.1:6379> CONFIG get dir
1) "dir"
2) "/var/lib/redis"
```

De acordo com a saída do comando CONFIG acima, ficamos sabendo que o diretório /var/lib/redis é o diretório onde o servidor do REDIS está instalado.

Bgsave

O comando BGSAVE é um comando alternativo para a criação de backups no REDIS. Este

comando irá iniciar o processo de backup e rodar este processo em background.

```
127.0.0.1:6379> BGSAVE
Background saving started
```

Transações

As transações permitem executar um grupo de comandos em um único passo.

Todos os comandos de uma transação são, sequencialmente, executados como um única operação isolada.

Transações no Redis são também operações atômicas, ou seja, signica que todos - ou nenhum - dos comandos de um transação são processados.

Transações no REDIS são iniciadas pelo comando MULTI e então necessitam de uma lista de comandos que devem ser executados pela transação. Devemos usar o comando EXEC para encerrar o bloco de uma transação iniciada pelo comando MULTI.

```
redis 127.0.0.1:6379> MULTI
OK
List of commands here
redis 127.0.0.1:6379> EXEC
```