**REDIS**

**Um banco de dados chave-valor avançado**

**Sumário**

1. O que é o Redis?........................................................................................3
2. Como usar o Redis?...................................................................................3
3. Instalando o Redis......................................................................................4
   1. Unix.......................................................................................5
   2. Windows................................................................................5
4. Configurando o Redis.................................................................................6
5. Persistência de dados.................................................................................7
   1. RDB.......................................................................................8
   2. AOF........................................................................................8
6. Executando o Redis....................................................................................9
7. Definindo um banco de dados....................................................................9
8. Tipos de dados..........................................................................................10
   1. Strings.................................................................................10
   2. List……................................................................................10
   3. Hash.....................................................................................11
   4. Set........................................................................................11
   5. Sorted Set............................................................................11
9. Operações com Redis…....…………………………………………………...12
10. Monitoramento..........................................................................................13
11. Otimização................................................................................................14
    1. Redis 32 bit ou 64 bit...........................................................16
    2. Dados comprimidos.............................................................16
    3. Nomes de chave curtos.......................................................17
    4. Tempo de expiração............................................................17
    5. Comandos multi argumentos...............................................17
    6. Queries lentas.....................................................................17
    7. Particionamento de dados...................................................17
12. Segurança.................................................................................................18
13. Backup......................................................................................................18
14. Replicação................................................................................................19
    1. Configuração master-slave..................................................19
    2. Sentinel................................................................................21
    3. Configurando e executando o sentinel................................22
15. Bibliografia................................................................................................26

**1.** **O que é Redis?**

O Redis (Remote Dictionary Server) é um banco de dados não relacional open source e foi criado em 2009 por Salvatore Sanfilippo mais conhecido na internet como antirez, para aumentar a performance da aplicação LLOOGG, uma aplicação que analisa logs web em tempo real.

A velocidade, flexibilidade e simplicidade são as principais características do Redis, isso é possível devido ao armazenamento de dados em memória e seu suporte a uma grande variedade de tipos de dados como Strings, Hashes, Sets, Sorted Sets e Lists. É importante enfatizar que o Redis tem outras características como a atomicidade entre seus comandos e a persistência de dados, a atomicidade é garantida porque o Redis é executado como uma aplicação single-threaded (apenas um comando é executado por vez) e a persistência de dados pode ser obtida através da configuração do servidor Redis.

Por isso, o Redis ganhou popularidade e hoje é utilizado por pequenas e grandes empresas para solucionar pequenas e grandes tarefas. Empresas como Engine Yard, Github, Craigslist, Disqus, Digg, and Blizzard fazem parte de uma grande lista de empresas que adotaram o Redis como solução de seus projetos.

**2. Como usar Redis?**

O Redis pode ser utilizado de muitas formas, além de poder ser utilizado como um banco de dados nosql, na maioria das vezes é utilizado como um repositório de dados em memória também conhecido como Cache. O armazenamento de dados em cache é válido quando existem dados que são requisitados e que não sofrem atualizações constantemente, poupando tempo e uso desnecessário de hardware.

**3. Instalando Redis**

O Redis está disponível para todas as plataformas Unix, como o Linux e Mac OS, mas não é suportado oficialmente em plataformas Windows, porém é possível instalar uma versão experimental baseada na versão 3.0 do Redis, mantida pelo grupo Microsoft Open Tech. A instalação nos sistemas Unix ou Windows em geral é bem fácil.

**3.1 Unix**

A instalação do Redis pode ser feita de varias maneiras, apartir do código fonte que está disponível em [www.redis.io/download](http://www.redis.io/download) ou realizar a instalação por algum gerenciador de pacotes presente nos sistemas Unix, como **apt-get** (debian), **yum** (Fedora), **HomeBrew** (Mac OS) e etc.

Para a instalação através do código fonte deve se seguir os seguintes passos:

*1 - wget http://download.redis.io/releases/redis-2.8.x.tar.gz*

*2 - tar zxvf redis-2.8.x.tar.gz*

*3 - cd redis-2.8.x*

*4 - make*

Na linha 1, foi utilizado a linha de comando *wget* para fazer o download do pacote Redis, em plataformas Mac OS devemos substituir o comando wget pelo:

*curl* [*http://download.redis.io/releases/redis-2.8.x.tar.gz*](http://download.redis.io/releases/redis-2.8.x.tar.gz) *-o redis-2.8.x.tar.gz*

Na linha 2, foi utilizado o comando *tar* para descompactar o pacote Redis.

Na linha 3, foi utilizado o comando *cd* para entrar na pasta que foi descompactado.

Na linha 4, foi utilizado o comando *make* para compilar o Redis. A compilação pode demorar um pouco para terminar, mas após o fim da execução desse comando, o redis já está pronto para uso.

Para a instalação através de gerenciadores de pacotes:

* Debian/Ubuntu

*sudo apt-get install redis-server*

* Fedora/Redhat/CentOS

*sudo yum install redis*

**3.2 Windows**

Como dito anteriormente, o Redis não tem suporte oficial para a plataforma Windows e por se tratar de uma versão experimental, não é recomendada a utilização em ambientes de produção. Mas podemos obter a versão não oficial do redis para windows em um arquivo compactado através do repositório <https://github.com/MSOpenTech/redis/archive/3.0.zip>, após o download, vamos descompacta-lo em uma pasta chamada *redis*, dentro da pasta irá existir um diretório chamado *bin* com os arquivos redisbin.zip e o redisbin64.zip que devem ser descompactados de acordo com a sua plataforma 32 ou 64 bits.

**4. Configurando o Redis**

O Redis é capaz de iniciar sem especificar qualquer arquivo de configuração, porém só é recomendado para fins de desenvolvimento e teste. Portanto, existem 3 maneiras de configurar o Redis. A primeira é através de um arquivo de configuração também conhecido como redis.conf, a outra é através de argumentos passados para o comando redis-server ao iniciar a execução e por último é através dos comando CONFIG SET e CONFIG GET que altera as configurações sem a necessidade de parar ou reiniciar o servidor do Redis. Todos os parâmetros de configuração no arquivo redis.conf também podem ser utilizados nas outras duas formas de configuração. Para utilizar um arquivo de configuração, basta utilizá-lo como parâmetro do comando redis-server.

*./redis-server redis.conf*

Para editar o arquivo redis.conf encontrado na pasta raiz da instalação utilize um editor de texto como vim, nano, gedit, notepad e etc. Ao abrir veremos uma grande variedade de diretivas disponíveis para a configuração de um servidor Redis. A seguir veja alguns dos mais importante parâmetros contido no redis.conf, os valores que se encontram entre os parênteses são os valores padrão da diretiva.

* **daemonize** **(no):** Define se o Redis vai ser executado como um daemon;
* **pidfile** **(/var/run/redis.pid):** Local onde será gerado o arquivo PID quando o parâmetro daemonize for yes;
* **port** **(6379):** A porta TCP que o Redis irá utilizar para aceitar conexões;
* **timeout** **(0):** Tempo de inatividade em segundos para que a conexão de um cliente seja fechada. O valor “0” desliga esse comportamento;
* **maxclients (10000):** Número máximo de clientes conectados;
* **maxmemory (0):** Valor máximo de memória que o Redis pode utilizar. O valor padrão 0 define que não existe limite;
* **slowlog-log-slower-than (10000):** Gerar log de queries com tempo de execução maior que 10.000 microssegundos.
* **slaveof:** Parâmetro para identifica o node master em uma arquitetura master-slave de replicação, aqui inserimos o ip e porta da instancia master.
* **requirepass:** Defini-se o password para executar operações no servidor Redis

Como dito anteriormente também podemos utilizar os mesmos parâmetros ao iniciarmos o servidor Redis, veja o exemplo abaixo.

*./redis-server --port 6380 --slaveof 127.0.0.1 6379*

As diretivas que foram apresentados anteriormente é uma pequena amostra dos parâmetros que se encontra no redis.conf . Para verificar todos os parâmetros possíveis, consulte o arquivo redis.conf, pois ele é muito bem documentado e de fácil compreensão.

**5. Persistencia de dados**

O Redis possui três modos diferentes de persistir seus dados, que são: RDB, AOF e o mesclado. Cada tipo de persistência funciona diferente uma da outra, sendo que o Redis permite mesclar os tipos e também desabilitar qualquer tipo de persistência e manter os dados somente em memória enquanto o servidor do Redis estiver em funcionamento. Se a forma AOF estiver ativa, independente da forma RDB também estar ativa ou não, o Redis irá utilizar o arquivo AOF por padrão, porque este modo

tem uma melhor garantia de durabilidade dos dados. Agora iremos entender um pouco mais sobre os modos de persistencia do Redis.

**5.1 RDB**

Esse é o modo padrão de persistência dados do Redis, conhecida como a forma mais simples de persistencia. Ao utilizar-la, o Redis persiste os dados de forma assíncrona, armazenando snapshots do seus dados em intervalos de tempo específicos. O problema desse modo é se ocorrer algum problema entre um intervalo e outro, os dados manipulados nesse tempo serão perdidos. Abaixo segue alguns parâmetros de configuração do redis.conf para o modo de persistencia RDB.

* **save x y:** define quando o Redis irá fazer snapshot dos dados na memória para o disco. Neste caso, o Redis irá persistir os dados a cada x segundos se pelo menos y chaves forem alteradas. O arquivo de configuração pode conter várias regras de save.
* **dbfilename dump.rdb:** nome do arquivo usado pelo Redis para persistir seus dados no formato RDB;
* **dir ./:** Caminho onde o arquivo do parâmetro dbfilename será criado.
* **stop-writes-on-bgsave-error:** Por padrão essa opção vem ativada “yes” e serve para interromper o processo de escrita de dados no Redis caso ocorra algum problema com o processo de armazenar os dados em disco. Esse recurso pode ser desabilitado mudando o valor desse parâmetro para "no”.
* **rdbcompression:** Por padrão essa opção vem ativada “yes” e serve para comprimir as strings utilizando o algoritmo de compressão LZF. Essa opção pode ser desabilitada com “no”, mas saiba que se fizer isso, o tamanho dos dados armazenados no Redis será maior.
* **rdbchecksum:** Por padrão essa opção vem ativada “yes” e serve para fazer com que o Redis adicione um checksum de 65 bits no final do arquivo. Isso ajuda a tornar a persistência mais resistente a problemas de corrupção de dados.

**5.2 AOF**

Esse modo escreve em um arquivo cada operação de escrita executada e armazena esses logs em um único arquivo. Desta forma, quando o Redis for reiniciado, os dados serão reconstruídos a partir desse arquivo de log. Essa forma de persistência é mais indicada para situações em que nenhum dado pode ser perdido em caso de desastre, porém isso pode impactar o desempenho do Redis. Abaixo segue alguns parâmetros de configuração do redis.conf para o modo de persistencia RDB.

* **appendonly:** Por padrão essa opção vem desativada “no” e serve para ativar a persistencia em AOF caso defina valor desse parâmetro como “yes”.
* **appendfilename "appendonly.aof ”:** nome do arquivo usado pelo Redis para persistir seus dados no formato AOF.
* **appendfsync:** Esse parâmetro define como a chamada ao método fsync() será feita. Esse método é responsável por informar ao sistema operacional para escrever os dados no disco em vez de aguardar e armazenar mais dados na saída de buffer. Existem três possíveis valores para esse parâmetro que são: “always”, “everysec” e “no”. O valor always define que o fsync() seja chamado a cada escrita, o valor everysec define que o fsync() seja chamado a cada segundo e no define que o fsync() nunca seja chamado.
* **no-appendfsync-on-rewrite no:** Por padrão essa opção vem desativada “no” e serve este parâmetro deve ser ativado (yes) somente se houver algum problema de latência, caso contrário o valor padrão (no) é mais recomendado por questões de durabilidade dos dados.

**6. Executando o Redis**

Agora que já vimos como e onde configurar o Redis, para iniciar o servidor nos sistemas Unix deve se executar o arquivo redis-server que se encontra dentro da pasta com os binários que foram extraídos na hora da instalação, para a execução do banco de dados execute o seguinte comando *./redis-server*. Já para inicializar na plataforma Windows deve se executar o arquivo redis-server.exe que foi extraido dos arquivos redisbin.zip ou redisbin64.zip de acordo com a sua plataforma escolhida.

**7. Definindo um banco de dados**

O Redis também possui o conceito de database assim como um banco de dados tradicional, porém ele não faz referência a um database pelo seu nome, mas sim através de números. Por padrão, o Redis sempre irá utilizar o database 0 (zero) caso nenhum outro seja especificado. O Redis também já deixa disponíveis outros 15 databases para serem utilizados. Esse limite pode facilmente ser alterado no arquivo de configuração do Redis (redis.conf) através do parâmetro databases.

O comando para especificar o database a ser utilizado é o SELECT. Ele sempre recebe um número referente ao database. Sempre que uma nova conexão é iniciada, o Redis automaticamente define o database 0 como o ativo.

**8. Tipos de dados do Redis**

O Redis é capaz de manipular 5 tipos de dados, são eles: strings, lists, hashes, sets e sorted sets mais a frente iremos entender quando e como utilizar cada um deles como solução para cada problema específico...

**8.1 Strings**

Strings é tipo mais basico que o Redis suporta, Strings são “binary safe”, ou seja, trata a entrada de dados como um fluxo bruto de bytes e ignora todos os aspectos textuais do dado, isso significa que a string do Redis pode conter qualquer dado, de uma instância de JPEG até um Objeto Ruby Serializado. Uma String pode ter um tamanho máximo de até 512 Megabytes. Abaixo segue algumas das operações que podemos fazer com uma string Redis.

* Os Comandos de inserção, recuperação e remoção dos valores da string como: SET, MSET, GET, MGET, DEL.
* Os Comandos de incremento e decremento dos valores da String como: [INCR](http://redis.io/commands/incr), [DECR](http://redis.io/commands/decr), [INCRBY](http://redis.io/commands/incrby) e DECRBY.
* O Comando para anexar um valor a uma string já existente é o [APPEN](http://redis.io/commands/append)D

Este são apenas alguns dos comandos que o tipo string do Redis suporta, existem muito outros que podemos conferir todos na documentação.

**8.2 List**

O tipo List do Redis nada mais é do que uma lista de strings ordenada pela ordem de inserção, o Redis permite a inserção de elemento na lista tanto no lado esquerdo (head) como no direita (tail), desta forma, podemos tratar uma lista como uma estrutura de Fila com o conceito FIFO (first in, first out) ou como uma pilha com o conceito FILO (first in, last out), o tamanho maximo de uma lista é de 232 - 1 elements (4294967295, mais de 4 bilhoes de elementos por lista). Abaixo segue algumas das operações que podemos fazer com uma Lista Redis.

* Os comandos de inserção, LPUSH inseri o novo elemento no lado esquerdo da lista e RPUSH inseri no lado direito
* Os comandos de remoção, LPOP removi o elemento do lado esquerdo da lista e RPOP remove do lado direito.
* O comando LLEN retorna o tamanho da lista.
* O comando LTRIM limita o tamanho da lista no intervalo especificado.

**8.3 Hash**

Hashes no Redis mapeia a chave de um string e seus valores, por isso eles são o tipo de dados perfeito para representar objetos por exemplo, um usuário com os campos como nome, sobrenome, idade deve ser armazenado como hash, pois desta forma, o objeto irá consumir pouco espaço e será permitido armazenar milhões de objectos em uma pequena instancia do Redis. Todo Hash pode armazenar mais que 4 bilhões de chave-valor. Abaixo segue algumas das operações que podemos fazer com um Hash Redis.

**8.4 Set**

O Set do Redis é uma coleção desordenada de Strings. É possível adicionar, remover e testar a existência de membros em O (1) (tempo constante, independentemente do número de elementos contidos no interior do Set). Esse tipo tem a propriedade desejável de não permitir que elementos se repitam. Em termos práticos, isto significa que a adição de um membro não requer uma verificação se existe. Uma coisa muito interessante sobre o Set é que eles suportam uma série de comandos do lado do servidor para computar conjuntos a partir de conjuntos já existentes, de modo que você pode fazer uniões, intersecções, e diferenças de conjuntos em tempo muito curto. O número máximo de membros em um conjunto pode chegar a mais de 4.000 milhões de membros por set.

**8.5 Sorted Set**

O Sorted Set Redis são, semelhantes ao Set Redis, os membros não podem se repetir. A diferença é que cada membro de um Sorted Set está associada um score, que é utilizado, a fim de ordenar Sorted Set, desde o menor a maior pontuação. Como os membros são únicos, a pontuação pode se repetir. Sorted Set pode adicionar, remover ou atualizar os elementos de uma maneira muito rápida (em um tempo proporcional ao logaritmo do número de elementos). Como os elementos estão em em ordem, você também pode obter faixas de pontuação ou por rankear de uma forma muito mais rápida. O Acesso ao meio de um Sorted Set também é muito rápido.

**9. Operações com Redis**

Imagine que toda vez que um usuário acessa o site do Netflix procurando um filme, ele tenha que esperar a aplicação varrer toda a base de dados, para só depois ele conseguir visualizar a página e decidir qual séries, shows ou filmes deve assistir. Seria um processo muito massante. É muito melhor que a aplicação retorne apenas um número que seja significativo. A pesquisa deve estar focada naquilo que o usuário está procurando. É desnecessária uma pesquisa mais complexa.

É interessante conseguirmos armazenar uma chave - "Categoria" - e um valor - "Comédia". Tal valor pode ser um número, uma palavra, ou uma lista de nomes.

Queremos, então, armazenar em algum lugar um conjunto de chaves e valores, de modo que a busca seja extremamente rápida, otimizada. Para isso, vamos utilizar o Redis.

Para armazenar, um valor no Redis, execute:

*SET "total\_de\_filmes" 1400*

Para obter o valor da chave "total\_de\_filmes", execute:

*GET "total\_de\_filmes"*

*1400*

Para remover uma chave utilize o comando DEL:

*DEL "total\_de\_filmes"*

O Redis também possui um padrão que serve para criar uma estrutura de chaves e ele é definido com ":" (dois pontos). Como exemplo é possível armazenar os resultados da mega-sena:

*SET resultado:29-05-2016:megasena "2, 15, 18, 25, 28, 32"*

Caso seja necessário armazenar diversas duplas chave-valor o comando "MSET" deve ser utilizado:

*MSET resultado:03-05-2016:megasena "1, 3, 17, 19, 24, 26" resultado:23-04-2016:megasena "15, 18, 20, 32, 37, 41" resultado:15-04-2016:megasena "10, 15, 18, 22, 35, 43"*

**10. Monitoramento**

Vamos aprender a monitorar todos os comandos que estão sendo executado no Redis. Para isso, iremos precisar iniciar uma instância do cliente Redis (redis-cli). Com o servidor do Redis funcionando, inicie uma instância e execute o comando MONITOR. Repare que, ao executá-lo, a conexão ficará bloqueada no mesmo instante. Veja a seguir o exemplo:

*127.0.0.1:6379> MONITOR*

*OK*

Apartir do momento que o servidor que está sendo monitorado começa a executar comandos, nosso cliente que monitora os processos começa a gerar outputs na tela com informações sobre cada comando, veja um exemplo:

*1395367119.521877 [0 127.0.0.1:62970] "keys" "\*"*

*1395367134.675322 [0 127.0.0.1:62970] "get" "numero-de-acessos"*

*1395367167.664193 [0 127.0.0.1:62970] "set" "sentido-da-vida" "42"*

*1395367185.199500 [0 127.0.0.1:62970] "del" "sentido-da-vida"*

Monitorar os comandos que estão sendo executados no Redis pode ser muito útil para identificar erros nos comandos que estão sendo enviados ao Redis por outras aplicações. Porém monitorar o Redis pode gerar um impacto significativo no desempenho. Apesar do Redis ser uma tecnologia simples de usar e até mesmo de configurar, ainda precisamos assegurar que as outras aplicações que o estão usando estejam fazendo isso da forma correta. O Redis fornece um comando muito útil que retorna informações e estatística sobre o servidor, veja um exemplo do uso dele.

*127.0.0.1:6379> info*

*# Server*

*redis\_version:2.8.7*

*redis\_git\_sha1:00000000*

*redis\_git\_dirty:0*

*redis\_build\_id:3d45c4c63ce96089*

*redis\_mode:standalone*

*os:Darwin 13.1.0 x86\_64*

*arch\_bits:64*

*multiplexing\_api:kqueue*

*gcc\_version:4.2.1*

*process\_id:45020*

*run\_id:1306d00fe1f7ed9312b897bace78720add3264eb*

*tcp\_port:6379*

*uptime\_in\_seconds:39*

*uptime\_in\_days:0*

*hz:10*

*lru\_clock:1139377*

*config\_file:*

*# Clients*

*connected\_clients:1*

*client\_longest\_output\_list:0*

*client\_biggest\_input\_buf:0*

*blocked\_clients:0*

*# Memory*

*used\_memory:1002704*

*used\_memory\_human:979.20K*

*used\_memory\_rss:1736704*

*used\_memory\_peak:968624*

*used\_memory\_peak\_human:945.92K*

*used\_memory\_lua:33792*

*mem\_fragmentation\_ratio:1.73*

*mem\_allocator:libc*

*# Persistence*

*loading:0*

*rdb\_changes\_since\_last\_save:0*

*rdb\_bgsave\_in\_progress:0*

*rdb\_last\_save\_time:1395514058*

*rdb\_last\_bgsave\_status:ok*

*rdb\_last\_bgsave\_time\_sec:-1*

*Rdb\_current\_bgsave\_time\_sec:-1*

*aof\_enabled:0*

*aof\_rewrite\_in\_progress:0*

*aof\_rewrite\_scheduled:0*

*aof\_last\_rewrite\_time\_sec:-1*

*aof\_current\_rewrite\_time\_sec:-1*

*aof\_last\_bgrewrite\_status:ok*

*# Stats*

*total\_connections\_received:1*

*total\_commands\_processed:0*

*instantaneous\_ops\_per\_sec:0*

*rejected\_connections:0*

*sync\_full:0*

*sync\_partial\_ok:0*

*sync\_partial\_err:0*

*expired\_keys:0*

*evicted\_keys:0*

*keyspace\_hits:0*

*Keyspace\_misses:0*

*pubsub\_channels:0*

*pubsub\_patterns:0*

*latest\_fork\_usec:0*

*# Replication*

*role:master*

*connected\_slaves:0*

*master\_repl\_offset:0*

*repl\_backlog\_active:0*

*repl\_backlog\_size:1048576*

*repl\_backlog\_first\_byte\_offset:0*

*repl\_backlog\_histlen:0*

*#Cpu*

*used\_cpu\_sys:0.05*

*used\_cpu\_user:0.03*

*used\_cpu\_sys\_children:0.00*

*used\_cpu\_user\_children:0.00*

*# Keyspace*

*db0:keys=2,expires=0,avg\_ttl=0*

Como dito anteriormente, o comando INFO apresenta um relatório completo sobre o servidor Redis, repare que as informações apresentadas estão separadas em blocos.

**11. Otimização**

A documentação do Redis disponibiliza algumas dicas para a otimização de memória do servidor, ajudando a evitar problemas e também a melhorar o desempenho. Vamos a elas então:

**11.1 Redis 32 bit ou 64 bit**

Se você conseguir mensurar a quantidade de dados que você pretende armazenar no Redis e esses dados não ultrapassarem 4GB, então nesse caso é recomendado que você utilize uma versão do Redis compilada pra 32 bit. Isso porque, ao usarmos 32 bit, os ponteiros irão possuir a metade do tamanho deum ponteiro de uma instância do Redis compilada para 64 bit e, com isso, o Redis irá utilizar menos memória para armazenar cada chave.

**11.2 Hashes**

Imagine que estamos armazenando os dados de um usuário em três dife-

rentes chaves ( user:name:XXX, user:email:XXX e user:phone:XXX, onde XXX representa o código do usuário), quando cada chave dessa é armazenada no Redis, essas chaves irão possuir metadados adicionais relacionados a elas.

Agora quando convertemos essas chaves em um único hash e adicionamos os dados (name,email e phone) como campos (fields) do hash, esses campos são armazenados em forma de dicionário como texto puro sem nenhum metadado adicional. Sendo assim, conseguimos diminuir o consumo de memória do Redis, otimizar e organizar os dados de uma forma mais eficiente.

**11.3 Dados Comprimidos**

Lembre-se que, quanto mais conteúdo armazenado em uma chave, maior será a memória utilizada pelo Redis. Por este motivo, utilize alguma forma eficiente de compactar os dados antes de armazená-los.

**11.4 Nomes de chave curtos**

Assim como devemos diminuir o tamanho do valor armazenado em uma chave sempre que possível, também devemos aplicar essa regra para o nome das chaves, pois eles também representam uma parte do consumo de memória.

**11.5 Tempo de expiração**

Defina um tempo de expiração para as chaves do Redis sempre que possível, pois assim você garante que terá em memória somente dados relevantes.

**11.6 Comandos multi argumentos**

Utilize comandos multi argumentos para evitar diversas requisições ao servidor do Redis como os comandos HGET, HSET e etc.

**11.7 Queries Lentas**

O Redis fornece um comando chamado SLOWLOG GET que serve para identificar de uma forma rápida e simples de queries que demoram mais tempo do que o esperado pelo Redis. Por padrão, o Redis assume o valor de 10000 microssegundos, mas é possível definir um valor diferente no arquivo de configuração do Redis através do parâmetro slowlog-log-slower-than. É importante lembrar que o tempo de execução que o Redis utiliza para definir de um comando não inclui operações de IO nem o tempo de envio da resposta ao cliente, mas sim apenas o tempo utilizado para executar o comando.

**11.8 Particionamento de dados**

Se a quantidade de dados que precisamos armazenar no Redis é muito grande, podemos particioná-los entre várias instâncias do Redis, cada uma em um computador diferente.

**12. Segurança**

A proposta do Redis é ser rápido na questão de uso e simples no quesito de configuração e administração. Isso também se reflete quanto à sua forma de lidar com segurança. De uma forma geral, o Redis foi criado para ser utilizado através de conexões confiáveis e em ambientes seguros. Com isso em mente, uma forma de garantirmos segurança no Redis é fazer com ele seja acessado apenas por clientes (aplicações) conhecidos dentro de um ambiente controlado e confiável. Em outras palavras, que apenas clientes dentro de uma rede interna possam conectar-se ao servidor do Redis e que o Redis não seja exposto diretamente para internet. Isso pode ser feito utilizando um firewall para bloquear acessos externos à porta TCP do Redis. Fora isso, ainda podemos adicionar mais uma camada de segurança ao Redis por meio de autenticação. O Redis não fornece nada muito elaborado como um perfil para cada tipo de usuário e coisas desse tipo pois isso iria contra a ideia de ser simples. Por isso, a autenticação é feita simplesmente utilizando uma senha que fica armazenada diretamente no seu arquivo de configuração apresentado na seção 9.1.

O parâmetro onde podemos definir a senha é o requirepass e após definirmos um senha com ele, todo cliente precisa executar o comando AUTH para obter autorização do servidor. Vamos realizar um pequeno teste. Começaremos editando o arquivo redis.conf, que se encontra na pasta raiz da instalação do Redis. Adicionemos o seguinte parâmetro:

**13. Backup**

Para realizar backup dos dados contidos no Redis em geral é uma tarefa bem simples, basta habilitar a persistência RDB e realizar uma cópia do arquivo definido no parâmetro de configuração dbfilename. Não se preocupe ao fazer a copia do arquivo enquanto o Redis estiver sendo executado, pois o Redis nunca escreve diretamente nesse arquivo, mas sim em um outro arquivo temporário, que é renomeado para o arquivo RDB quando a persistência estiver completa.

**14. Replicação**

O processo de replicação do Redis tem a arquitetura master-slave, na qual uma ou mais instâncias do tipo slave, cada slave nada mais é do que uma cópia idêntica dos dados contidos na instância do tipo master. a seguir, alguma das características mais importantes sobre replicação no Redis:

* O processo de replicação é feito de forma assíncrona;
* Uma instância master pode ter várias instâncias slaves;
* Uma instância slave é capaz de receber conexões de outros slaves e assim torna-se master apenas dos slaves (similar a uma estrutura de grafos)
* O processo de replicação dos dados é feito de forma não bloqueante tanto para a instância master quanto para as slaves. Com isso, o master continua processando comandos normalmente enquanto a replicação é feita.

A replicação de dados é muito útil quando precisamos garantir escalabilidade de uma aplicação, pois podemos, por exemplo, deixar que aplicação realize comandos de escrita na instância master enquanto os comandos de leitura são processados por um ou mais slaves. Com uma abordagem desse tipo conseguimos balancear a carga que seria direccionada apenas para o master entre os slaves. Uma outra utilidade para replicação é quando precisamos ter redundância de dados.

**14.1 Configuração master-slave**

Configurar a replicação no Redis é muito simples. vamos utilizar o CLI e a própria linha de comando para fazer a configuração da replicação. Lembrando que podemos fazer toda a configuração diretamente no arquivo de configuração do Redis. Vamos iniciar uma instância do Redis na porta 6379 e outra no 6380. Veja como fazer isso no exemplo a seguir:

*./redis-server --port 6379*

*./redis-server --port 6380*

A instância na porta 6379 será a master e instancia na porta 6380 a slave.Nesse momento, temos duas instâncias do Redis sendo executadas, mas ainda não estão configuradas como uma replica master e slave, e por isso ambas são do tipo master e independentes. Vamos utilizar a linha de comando para acessar a instância da porta 6380 e configurá-la como slave da instância na porta 6379. Veja como fazer isso:

*./redis-cli -p 6380*

*127.0.0.1:6380> SLAVEOF 127.0.0.1 6379*

Ao executar a propriedade slaveof <master ip> <master port> do arquivo de configuração ou ainda o parâmetro --slaveof do comando de terminal redis-server para realizar essa configuração. Para confirmar que tudo está correto, vamos executar o comando INFO replication e verificar quais informações ele nos fornece.

*127.0.0.1:6380> INFO replication*

*#replication*

*role:slave*

*master\_host:127.0.0.1*

*master\_port:6379*

*master\_link\_status:up*

*master\_last\_io\_seconds\_ago:4*

*master\_sync\_in\_progress:0*

*slave\_repl\_offset:211*

*slave\_priority:100*

*slave\_read\_only:1*

*connected\_slaves:0*

*master\_repl\_offset:0*

*repl\_backlog\_active:0*

*repl\_backlog\_size:1048576*

*repl\_backlog\_first\_byte\_offset:0*

*repl\_backlog\_histlen:0*

Podemos verificar que a propriedade role dessa instância está definida como slave e a conexão (master\_link\_status) com o master (master\_host e master\_port) está funcionando (up).

Desde a versão 2.6 do Redis, por padrão instâncias do tipo slave são “read-only” aceitam somente leitura. Podemos alterar esse comportamento mudando o parâmetro de configuração slave-read-only para no. Agora vamos no conectar a instancia master porta 6379 e verificar as informações sobre replicação que o comando INFO nos retorna.

*127.0.0.1:6379> INFO replication*

*# Replication*

*role:master*

*connected\_slaves:1*

*slave0:ip=127.0.0.1,port=6380,state=online,offset=981,lag=1*

*master\_repl\_offset:995*

*repl\_backlog\_active:1*

*repl\_backlog\_size:1048576*

*repl\_backlog\_first\_byte\_offset:2*

*repl\_backlog\_histlen:994*

A propriedade connected\_slaves informa quantos slaves estão conectados a instância master e quais são eles. Repare que apenas o slave0 está conectado ao master.

**14.2 Sentinel**

Apesar da configuração de uma réplica no Redis ser bastante simples, precisamos configurar uma sentinela para gerenciar a relação entre as instâncias master e slave para eleger um novo master caso ocorra algum problema (“failover”) com a atual instância master.

Sentinel é uma solução para o problema de “failover” automático e de alta disponibilidade no Redis. Vamos conhecer a seguir as quatro principais responsabilidades do Sentinel, para melhor entendermos sua aplicação em um ambiente de réplicas do Redis.

* **Monitoramento:** Verifica constantemente se as instâncias master e slave estão funcionando corretamente;
* **Notificação:** Notificar o administrador do sistema, ou uma outra aplicação, que ocorreu algum problema com alguma instância do Redis que está sendo monitorada;
* **Failover automático:** Caso o master não esteja funcionando corretamente, é tarefa do Sentinel iniciar um processo de failover e eleger um slave para master, reconfigurar os outros slaves caso existam para considerar o novo master e informar as aplicações quem é o novo master;
* **Provedor de dados de configuração:** Sentinel prove o endereço da instância master do Redis, e também para avisa quando alguma mudança ocorre.

Podemos executar vários processos do Sentinel em um ou mais computadores. A seguir algumas regras utilizadas por ele.

* Um cluster de Sentinel pode realizar o failover do master mesmo se alguma instância do Sentinel estiver falhando.
* Somente uma única instância do Sentinel não consegue realizar o failover sem que as outras instâncias do Sentinel autorizem.
* Qualquer das instâncias do Sentinel podem ser conectadas para obter os dados de configuração do master.

**14.3 Configurando e executando o Sentinel**

Antes de executarmos o sentinel precisamos editar um arquivo de configuração, este arquivo de configuração possui parâmetros específicos do Sentinel. Veja um exemplo abaixo de como configurar toda uma arquitetura Redis com replicas e sentinelas para prover escalabilidade e redundância de dados para sua aplicação.

Vamos iniciar novamente duas instâncias do Redis uma master na porta 6379 e outra slave na porta 6380

*./redis-server --port 6379*

*./redis-server --port 6380**--slaveof 127.0.0.1 6379*

As instâncias master e slave já estão prontas. Notem que ao iniciarmos a instancia slave além do parametro --port utilizamos o parametro --slaveof para determinarmos qual é a instancia master para o slave.

Vamos utilizar duas instâncias do Sentinel, pois esta é a quantidade mínima de instâncias necessárias para realizar o processo de “failover” de forma automática. Como dito anteriormente é preciso criar um arquivo de configuração para o Sentinel, então vamos criar um arquivo chamado sentinel1.conf no mesmo local do arquivo redis.conf, e adicionar o seguinte conteúdo:

*port 26379*

*sentinel monitor mymaster 127.0.0.1 6379 2*

*sentinel down-after-milliseconds mymaster 10000*

Resumindo, esse arquivo de configuração determina ao Sentinel que ele será executado na porta 26379, será responsável por monitorar a instância master do Redis no ip 127.0.0.1 na porta 6379 e alertar se o master está inacessível por mais de *10000* milisegundos, ou seja, 10 segundos sem comunicação com ele.

Agora vamos criar um outro arquivo chamado sentinel2.conf e preencher com um conteúdo semelhante

*port 26380*

*sentinel monitor mymaster 127.0.0.1 6379 2*

*sentinel down-after-milliseconds mymaster 10000*

Após a criação dos arquivos de configuração vamos iniciar o Sentinel, podemos fazer isso de duas formas: uma é utilizando o comando redis-sentinel e a outra é utilizando o próprio comando do Redis redis-server com o parâmetro --sentinel. Como teremos que

iniciar duas instâncias do Sentinel, vamos utilizar as duas formas. Veja isso a seguir:

*./redis-sentinel sentinel1.conf*

*./redis-server sentinel2.conf --sentinel*

Pronto, agora já temos toda a nossa infraestrutura devidamente configurada e funcionando. Caso ocorra qualquer problema com a instância master do Redis, o Sentinel irá detectar e dará inicio ao processo de failover elegendo ou slave para ser o novo master. Então vamos ver na prática, para isso vamos conectar no sentinel utilizando o seguinte comando.

*./redis-cli -p 26379*

Após se conectar ao Sentinel, execute o comando INFO sentinel para exibir algumas informações sobre o sentinel.

*127.0.0.1:26379> INFO sentinel*

*# Sentinel*

*sentinel\_masters:1*

*sentinel\_tilt:0*

*sentinel\_running\_scripts:0*

*sentinel\_scripts\_queue\_length:0*

*master0:name=mymaster,status=ok,address=127.0.0.1:6379,slaves=1,sentinels=2*

Através dessas informações podemos saber que esse Sentinel está monitorando um único master ( sentinel\_masters:1) de nome mymaster. Esse master possui um único slave ( slaves=1) e está devidamente acessível (status=ok). Também é possível saber que existem duas instâncias do Sentinel ( sentinels=2) monitorando o master de nome mymaster. Agora vamos nos conectar à instância master do Redis e forçar um “shutdown” da seguinte forma:

*./redis-cli -p 6379*

*127.0.0.1:6379> SHUTDOWN*

*127.0.0.1:6379> exit*

Após alguns segundos, o Sentinel vai detectar que a instância master do Redis está inacessível e vai marcá-la como sdown. Depois, a outra instância do Sentinel vai realizar a mesma marca e iniciar o processo de eleição do novo master. Depois de mais alguns segundos, você poderá notar que o slave passou a ser master através do seguinte output apresentado pelo Sentinel:

*+switch-master mymaster 127.0.0.1 6379 127.0.0.1 6380*

Para confirmar que o failover ocorreu de fato, vamos conectar novamente ao Sentinel pelo CLI e consultar seus dados:

*src/redis-cli -p 26379*

*127.0.0.1:26379> INFO sentinel*

# Sentinel

sentinel\_masters:1

sentinel\_tilt:0

sentinel\_running\_scripts:0

sentinel\_scripts\_queue\_length:0

master0:name=mymaster,status=ok,address=127.0.0.1:6380,slaves=1,sentinels=2

E sim, o master agora é a instância que está rodando na porta 6380 e tudo está funcionando conforme o esperado. Mas o que aconteceria se a antiga instância master voltar a ser executada? Vamos ver isso agora:

*./redis-server --port 6379*

*./redis-cli --p 6379*

*127.0.0.1:6379> INFO replication*

*# Replication*

*role:slave*

*master\_host:127.0.0.1*

*master\_port:6380*

*master\_link\_status:up*

*master\_last\_io\_seconds\_ago:1*

*master\_sync\_in\_progress:0*

*slave\_repl\_offset:12253*

*slave\_priority:100*

*slave\_read\_only:1*

*connected\_slaves:0*

*master\_repl\_offset:0*

*repl\_backlog\_active:0*

*repl\_backlog\_size:1048576*

*repl\_backlog\_first\_byte\_offset:0*

*repl\_backlog\_histlen:0*

Veja que interessante, ao executarmos novamente a antiga instância master, o Sentinel automaticamente a promoveu e a configurou como slave da instância master corrente.

**15. Referencias bibliograficas**

[1] Citrusbyte. Redis data types. http://redis.io/topics/data-types.

[2] Salvatore Sanfilippo. Antirez weblog. http://antirez.com/.

[3] Redis Site. Redis memory optimization. http://bit.ly/1lZYIP8.

[4] Redis Site. Redis sentinel documentation. http://bit.ly/1hbN9UC.

[5] R. Lazoti. Armazenando dados com redis. 1ª ed. Casa do codigo.

[6] T. Macedo e F. Oliveira. Redis Cookbook. 1ª ed. O’Reilly Media, Inc.