redis

Euller Henrique Bandeira Oliveira

O que é o Redis?

O Redis (Remote Dictionary Server: Servidor remoto de dicionário) é um banco de dados não relacional de chave-valor que a priori armazena/busca seus dados na memória RAM e a posteriori pode armazenar seus dados no HD/SSD como um backup

Benefícios

Benefícios

1. Performance

- Os dados residem na memória RAM, tal fato permite que o acesso a eles ocorra com uma baixa latência (microssegundos) e com uma alta taxa de transferência.
- Pode suportar uma quantidade de operações maior (milhões por segundo) e tempos de respostas mais rápidos (em média menos de um milissegundo) do que um banco de dados relacional.

Benefícios

2. Estrutura De Dados Flexíveis

- Oferece:
  - Uma grande variedade de estruturas de dados:
    - Strings: Dados em texto ou binário com tamanho de até 512mb
    - Listas: Coleção de strings na ordem em que foram adicionadas
    - Conjuntos: Coleção não ordenada de strings (Operações: intersecção, união e diferenciação)
    - Conjuntos Ordenados: Coleção ordenada de strings Hashes: Lista de chave-valor
    - Bitmaps: Oferece operações de nível de bits
    - HyperLogLogs: Estrutura de dados probabilística para estimar os itens únicos em um conjunto de dados
    - Streams: Fila de mensagens de logs
    - Dados Geoespaciais: Mapa de registros com base em longitude/latitude
    - Json: Objeto aninhado semiestruturado de valores nomeados que suportam números, strings, booleandos, matrizes e outros objetos
    -

Benefícios

3. Simplicidade E Facilidade De Uso

- Permite que um código tradicionalmente complexo seja escrito com mais facilidade
- Poucas linhas de código são necessárias para armazenar e buscar e dados
- Comandos são mais simples do que comandos pertencentes a bancos de dados relacionais
- Ex: Com apenas uma linha de código, é possível inserir vários dados no redis por meio da estrutura de dados hash

Benefícios

4. Replicação E Persistência

- Replicação:

- Utiliza uma arquitetura de réplica principal
- Oferece suporte à replicação assíncrona (Os dados não são replicados instantamente)
- Permite a replicação do servidor/replica principal e de seus dados
- Oferece maior performance de leitura (as solicitações são distribuídas entre vários servidores)
- Se o servidor/replica principal ficar indisponível, uma das réplicas se torna o servidor/replica principal

- Persistência:
- Oferece:
- Suporte a snapshots point-in-time (Cria cópias instantâneas (snapshots) do estado atual dos dados em um determinado ponto no tempo e as armazena no disco)
- Suporte a Append Only File(AOF) (Toda alteração que ocorre na memória RAM, também ocorre no disco/SSD)
- Tais métodos permitem a restauração rápida dos dados do Redis no caso de uma interrupção

Benefícios

5. Alta Disponibilidade E Escalabilidade

- Oferece:
  - Uma arquitetura de réplica principal em:
    - Um único nó principal
    - Uma topologia de clusters (Vários clusters em que cada um possui vários nós)
  - Alta disponibilidade (Capacidade de um sistema funcionar quando for necessário)
  - Alta Performance (Capacidade de um sistema funcionar com velocidade etc)
  - Alta Confiabilidade (Capacidade de um sistema funcionar sem a ocorrência de falhas críticas por um determinado período)
  - Escalabilidade vertical (Aumenta o hardware de uma máquina)
  - Escalabilidade horizontal (Aumenta a quantidade de “máquinas”)

Benefícios

6. Versatilidade E Facilidade De Uso

- Oferece:
  - Várias ferramentas que tornam o desenvolvimento e as operações mais rápidas e fáceis:
    - Pub/Sub: Publica mensagens em canais para que sejam recebidas por assinantes
    - Chaves com tempo de vida útil (TTL): Ajuda a evitar que o banco de dados seja sobrecarregado com itens desnecessários
    - Contadores atômicos: Garantem que operações concorrentes não criem resultados incompatíveis
    - Lua: Uma linguagem de script potente e leve

Benefícios

7. Código Aberto

- Possui o suporte da comunidade
- Não há o aprisionamento a nenhum fornecedor ou tecnologia
- É baseado em padrões abertos
- É compatível com formatos de dados abertos

Casos De Uso

Casos de uso

1. Armazenamento em cache

A principal funcionalidade do redis é o cache.
Objetivos do uso do cache:

- Diminuir a latência de acesso aos dados
- Aliviar a sobrecarga de bancos de dados relacionais (Ex: PostgreSQL) ou não relacionais (ex: MongoDb)

Motivos do uso do redis para implementar o cache:

- Dados frequentemente solicitados são retornados em menos de um milissegundo
- É capaz de se escalar facilmente para processar um alto volume de dados

Exemplos de uso:

- Armazenamento do resultado de consultas realizadas em um banco de dado relacional
- Armazenamento de sessões
- Armazenamento de html, css, js, imagens, arquivos e metadados

Casos de uso

2. Armazenamento de sessões

O redis é extremamente indicado para gerenciar sessões de sites

Motivos do uso do redis para armazenar e gerenciar dados de sessão:

- Oferece:
  - Latência baixa (Inferior a um milissegundo)
  - Escalabilidade
  - Resiliência (Capacidade do sistema funcionar com a ocorrência de falhas Obs: Tal capacidade é obtida por meio da replicação)
  - Alta disponibilidade
  - Persistência

Exemplos de uso:

- Perfis de usuário
- Credenciais
- Estados de sessão
- Personalização específicas de usuários
- Aplicações online:
- Jogos
- E commerce
- Redes sociais

Forma de uso:

- Crie uma chave-valor e determine o tempo
- de vida (TTL) dela para que ela possa ser
- utilizada como chave de sessão.

Casos de uso

3. Machine Learning

Objetivos do uso do machine learning:

- Processar rapidamente um grande volume e variedade de dados
- Automatizar decisões

Motivos do uso do redis para implementar machine learning:

- Oferece:
  - Banco de dados ágil (Armazenado na memória RAM) que pode ser usado para criar, treinar e implantar modelos de machine learning
  - Capacidade de processar dados ao vivo
  - Capacidade de decisão em milissegundos

Exemplos de uso:

- Detecção de fraudes (jogos e serviços financeiros)
- Ofertas em tempo real (anúncios) (Ex: GoogleAds)
- Matchmaking para encontros (Ex: Tinder)
- Transporte solidário (Ex: Wase)

Casos de uso

4. Dados Geoespaciais

Motivos do uso do redis para armazenar dados geospaciais:

- Oferece:
  - Estruturas e operadores que armazenam, processam e analisam dados geoespaciais
  - Ex: GEOADD, GEODIST, GEORADIUS E GEORADIUSBYMEMBER
- Permite gerenciar em tempo real tais dados em grande escala e velocidade

Exemplo de uso:

- Adição de recursos relacionados a localização em um site/app, tais como:
  - Tempo de percurso
  - Distância do percurso
  - Pontos de interesse

Casos de uso

5. Análise em tempo real

Motivos do uso do redis para análises em tempo real:

- Oferece: Banco de dados NoSQL (Armazenado na mémoria RAM) para consumir, processar e analisar dados em tempo real com baixa latência
- Integração com o Apache Kafka e Amazon Kinesis

Exemplos de uso:

- Análises de redes sociais
- Direcionamento de anúncios
- Personalização
- Internet das coisas

Casos de uso

6. Limite de taxa

Motivos do uso do redis para limitação de taxa:

- Oferece: Cálculo da taxa dos eventos
- Aceleração da taxa dos eventos

Forma de uso:

- Associe um contador radis a uma chave de API do cliente (endpoint) para que o número de solicitações de acesso em um determinado período seja contado.
- Se um determinado limite for excedido, realize a ação que você julgar necessária. (Ex: Escalonamento horizontal)

Exemplos de uso:

- Limitar:
  - Números de publicações em um fórum
  - Utilização de recursos
  - O impacto de remetentes de spam (Um remetente é considerado um remetente de spam se ele enviar muitos emails)

Casos de uso

7. Placares de jogos

Desenvolvedores de jogos costumam escolher o redis para criar placares em tempo real

Motivos do uso do redis para geração de placares de jogos:

- Oferece:
  - Estrutura de dados Sorted Set (Ordena a lista de pontuações automaticamente)

Exemplos de uso:

- Criação de uma lista classificada em tempo real
- Atualização da pontuação de um usuário a cada mudança
- Processamento de dados de séries temporais (usa o time-stamps como pontuação) a partir de conjuntos classificados

Casos de uso

8. Chat, sistemas de mensagens e filas

Motivos do uso do redis para criação de chats, sistemas de mensagens e filas:

- Oferece:
  - Suporte a Pub/Sub
  - Estrutura de dados: Listas, conjuntos ordenados e hashes
    - Listas: Facilita a implementação de uma fila leve
      - Operações atômicas (LPOP, RPOP, LPUSH E RPUSH)
      - Recursos de bloqueio (BLPOP, BRPOP, BRPOPLPPUSH)
      - Adequadas para apps que exigem:
        - Agente de mensagens
        - Lista circular confiável

Exemplos de uso:

- Salas de chat de alta performance
- Streams de comentários em tempo real
- Feeds de redes sociais
- Intercomunicação de servidores

Casos de uso

9. Streaming de mídia avançada

Motivos do uso do redis para implementar o streaming ao vivo:

- Oferece: Banco de dados NoSQL rápido (Armazenado na memória RAM)

Exemplos de uso:

- Armazenagem:
  - Metadados de perfis de usuários
  - Históricos
  - Informações/tokens de autenticação
  - Arquivos de manifesto
- Tais dados possibilitam que CDNs façam streaming de vídeo para milhões de usuários ao mesmo tempo

Redis Json

RedisJson

Problema

- Por padrão, um Json é armazenado no Redis como uma string
- Portanto, o json é buscado e salvo no redis em sua totalidade

- Se for necessário fazer uma busca/atualização do valor de um campo em um Json/Json Aninhado
- |
- v
- Buscar a string com formato json, transforma-la em um objeto, manipular o objeto, transformar o objeto em uma string com formato json e salvar o json no Redis

- Problema:
- Aumenta o tráfego de rede e consequentemente a latência, pois o Json inteiro deve ser buscado e salvo frequentemente
- Piora a performance da aplicação, pois ela recebe uma função que normalmente o banco de dados receberia
- Aumenta a complexidade do código, já que ele adquire mais responsabilidade

RedisJson

Solução

- Módulo RedisJson
  - Permite tratar dados em formato Json como uma estrutura de dados nativa do Redis
  - Similar a um banco não relacional baseado em documentos (ex: MongoDB), contudo possui menos funções
  - Apresenta uma sintaxe simples de entender
  - Adiciona uma série de comandos que facilitam a manipulação de dados em formado Json no Redis

Obs:

- Limite de tamanho do documento (Estrutura de dados Json): 64MB
- Limite de profundidade do agrupamento (Limite de aninhamento de um json): 128

Exemplo

Docker+Java+Spring+Redis+Redis Json

Exemplo

Pom.xml

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

```xml
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-redis</artifactId>
</dependency>
```

```xml
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>4.2.3</version>
</dependency>
```

Exemplo

Docker

```yaml
version: "3.8"

services:

  api_pagamento:
    container_name: api_pagamento
    image: api_pagamento:1.0
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8081:8081"
    depends_on:
      - postgresql
      - redis-json
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://postgresql:5432/db
      - SPRING_DATASOURCE_USERNAME=euller
      - SPRING_DATASOURCE_PASSWORD=12345
      - SPRING_JPA_HIBERNATE_DDL_AUTO=update
      - SPRING_JPA_SHOW_SQL=true
      - SPRING_JPA_HIBERNATE_FORMAT_SQL=true
      - SPRING_REDIS_HOST=redis-json
      - SPRING_REDIS_PORT=6379
    networks:
      - api_pagamento-network
```

```yaml
  postgresql:
    image: postgres:latest
    container_name: postgresql
    ports:
      - 5432:5432
    environment:
      - POSTGRES_DB=db
      - POSTGRES_USER=euller
      - POSTGRES_PASSWORD=12345
    volumes:
      - ./data/postgres:/var/lib/postgresql/data
    networks:
      - api_pagamento-network

  redis-json:
      image: redislabs/rejson:latest
      container_name: redis-json
      ports:
        - 6379:6379
      volumes:
        - ./data/redis:/data
      networks:
        - api_pagamento-network
networks:
  api_pagamento-network:
    driver: bridge
```

```
FROM maven:3.8.3-openjdk-17  as build
COPY src /app/src
COPY pom.xml /app
RUN mvn -f /app/pom.xml clean package

FROM openjdk:17-jdk
COPY --from=build /app/target/*.jar /app.jar
ENTRYPOINT ["java","-Dspring.profiles.active=stack", "-jar","/app.jar"]
```

Exemplo

Terminal Docker+Redis

1. Clone o repósitorio
2. Instale o docker (https://www.docker.com/products/docker-desktop/)
3. Abra o docker
4. Abra o terminal
5. Navegue até a api_pagamento_redis_json
6. Digite docker-compose up -d
    i. O jar da aplicaçào será gerado, executado e inserido em uma imagem
    ii. A imagem da aplicação será executada, ou seja, se tornará um container
    iii. A imagem do postgresql será executada, ou seja, se tornará um container
    iv. A imagem do redis-json será executada, ou seja, se tornará um container

1. Abra o terminal
2. Digite docker exec -it redis-json bash para abrir o terminal do container redis-json
3. Digite redis-cli --raw (--raw: exibe os dados codificados em UTF8)
4. Digite keys * para exibir as keys salvas
5. Digite JSON.GET <-nome-da-key-> para exibir o valor de determinada key

Exemplo

Spring Service

```java
@Override
public TransacaoDTO pagar(Transacao transacao) throws InsercaoNaoPermitidaException {

    if(transacao.getDescricao().getStatus() == null && transacao.getDescricao().getNsu() == null && transacao.getDescricao().getCodigoAutorizacao() == null && transacao.g

        transacao.getDescricao().setNsu("1234567890");
        transacao.getDescricao().setCodigoAutorizacao("147258369");
        transacao.getDescricao().setStatus(StatusEnum.AUTORIZADO);

        //Save on Database
        Transacao transacaoSave = transacaoRepository.save(transacao);

        //Save on Cache
        transacaoCacheRepository.setKey("size::transacoes", transacaoSave.getId().toString());

        return (TransacaoDTO) Mapper.convert(transacaoSave, TransacaoDTO.class);

    }else{
        throw new InsercaoNaoPermitidaException();
    }

}
```

Exemplo

Spring Service

```
public TransacaoDTO estornar(Long id) throws TransacaoInexistenteException {

    //Get of cache
    TransacaoDTO transacaoDTO = transacaoCacheRepository.getJsonKey("transacao::"+id.toString());
    if(transacaoDTO != null) {
        if(transacaoDTO.getDescricao().getStatus() == StatusEnum.NEGADO){
            return transacaoDTO;
        }
    }

    //Get of Database, Save on DataBase
    Transacao transacao = (Transacao) Mapper.convert(getById(id), Transacao.class);
    transacao.getDescricao().setStatus(StatusEnum.NEGADO);

    descricaoRepository.save(transacao.getDescricao());

    //Save on cache
    transacaoDTO = transacaoCacheRepository.updateJsonKey("transacao::"+id, "descricao.status", StatusEnum.NEGADO.toString());
    if(transacaoDTO!=null){
        return transacaoDTO;
    }else{
        throw new TransacaoInexistenteException();
    }

}
```

Exemplo

Spring Service

```java
@Override
public TransacaoDTO getById(Long id) throws TransacaoInexistenteException {

    //Get of Cache
    TransacaoDTO transacaoDTO = transacaoCacheRepository.getJsonKey("transacao::"+id.toString());
    if(transacaoDTO != null) {
        return transacaoDTO;
    }

    //Get of DataBase
    transacaoDTO = (TransacaoDTO) transacaoRepository.findById(id).map(t -> Mapper.convert(t, TransacaoDTO.class)).orElse(null);
    if(transacaoDTO != null){
        //Save on Cache
        transacaoCacheRepository.setJsonKey("transacao::"+transacaoDTO.getId(), transacaoDTO);
        return transacaoDTO;
    }else{
        throw new TransacaoInexistenteException();
    }

}
```

Exemplo

Spring Service

```java
@Override
public List<TransacaoDTO> getAll() throws TransacaoInexistenteException {

    //Get of Cache
    List<TransacaoDTO> transacoesDTO = transacaoCacheRepository.getAllJsonKey();
    Long sizeDB = transacaoCacheRepository.getKey("size::transacoes");
    if(sizeDB != null) {
        if(transacoesDTO.size() == sizeDB) {
            return transacoesDTO;
        }
    }

    //Get of Database
    transacoesDTO = transacaoRepository
            .findAll()
            .stream()
            .map(t -> {
                TransacaoDTO transacaoDTO = (TransacaoDTO) Mapper.convert(t, TransacaoDTO.class);
                //Save on Cache
                transacaoCacheRepository.setJsonKey("transacao::"+t.getId(), transacaoDTO);
                return transacaoDTO;
            })
            .collect(Collectors.toList());
    if(transacoesDTO.size() != 0){
        return transacoesDTO;
    }else{
        throw new TransacaoInexistenteException();
    }

}
```

Exemplo

Spring Repository

```
//Jedis: Jedis é um cliente Java para Redis projetado para desempenho e facilidade de uso.
```

```
private final JedisPooled jedis;
private final Gson gson = new Gson();
```

```
public void setKey(String key, String value) {
    jedis.set(key, value);
}
```

```
public Long getKey(String key) {
    String value = jedis.get(key);
    if (value != null) {
        return Long.parseLong(value);
    }
    return null;
}
```

```
public boolean verifyIfExistJsonKey(String key) {
    //jedis.exist(): Verifica se a chave existe no Redis.
    return jedis.exists(key);
}
```

Exemplo

Spring Repository

```java
public void setJsonKey(String key, TransacaoDTO value) {
    if(!verifyIfExistJsonKey(key)) {
        //jedis.jsonSet(key, value): Cria uma chave e associa um valor a ela
        jedis.jsonSet(key, gson.toJson(value));
        //jedis.expire(): Define o tempo de vida da chave.
        jedis.expire(key, 60 * 5);
    }
}
```

```java
public TransacaoDTO getJsonKey(String key){
    if (verifyIfExistJsonKey(key)){
        //jedis.jsonGet(key): Retorna o valor associado a determinada chave
        return (TransacaoDTO) Mapper.convert(jedis.jsonGet(key), TransacaoDTO.class);
    }
    return null;
}
```

```java
public TransacaoDTO updateJsonKey(String key, String field, String value){
   if (verifyIfExistJsonKey(key)) {
       //jedis.jsonSet(key, Path.of(field), value): Atualiza o valor de determinado campo de determinada chave.
       jedis.jsonSet(key, Path.of(field), value);
       return (TransacaoDTO) Mapper.convert(jedis.jsonGet(key), TransacaoDTO.class);
   }
   return null;
}
```

```java
public List<TransacaoDTO> getAllJsonKey() {
    //jedis.keys("transacao::*"): Retorna todas as chaves que começam com "transacao::"
    return jedis.keys("transacao::*")
            .stream()
            .map( key -> getJsonKey(key))
            .sorted(Comparator.comparing(TransacaoDTO::getId))
            .collect(Collectors.toList());

}
```


redis

Referências

https://github.com/EullerHenrique/api_pagamento_redis_json
https://aws.amazon.com/pt/redis/
https://aws.amazon.com/pt/elasticache/what-is-redis/
https://medium.com/@1fabiopereira/redisjson-manipulando-json-como-tipo-nativo-no-redis-3736e1fba832
https://redis.io/docs/stack/json/
https://docs.aws.amazon.com/pt_br/memorydb/latest/devguide/json-document-overview.html#json-nesting-depth-limit
https://redis.io/commands/?group=json
https://paulo-coelho.github.io/ds_notes/