Redis モジュール機能の解説

OSC2019 .Enterprise

SRA OSS, Inc. 日本支社 山本博之

はじめに

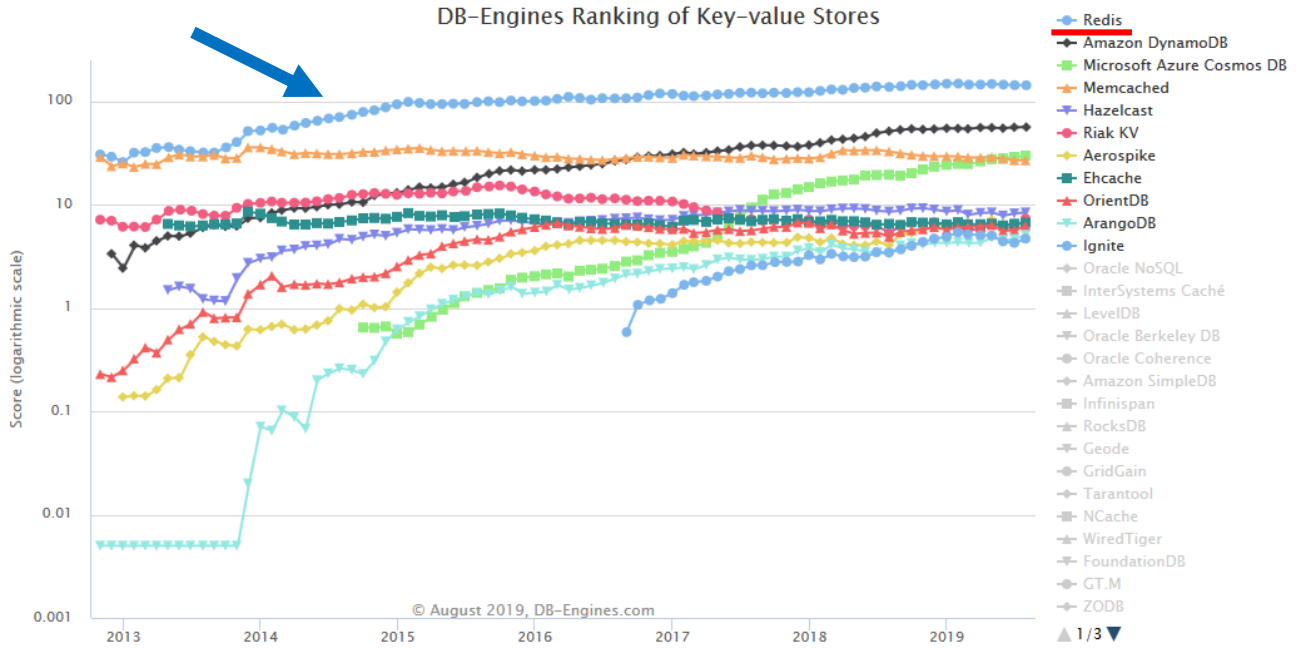
- Redisについて
- Redisモジュールとは
- Redisモジュールで可能なこと
- Redisモジュールのライセンス変更について
- Redisモジュールの作り方
- 公開されているRedisモジュールの紹介

Redisについて

- データの永続化が可能なインメモリデータベース
- Key-valueストア
- 多様なデータ型に対応
  - 文字列、ハッシュ、リスト etc.
- メッセージング機能も備える
  - ストリーム、Pub/Sub、Blocked list
- 冗長化、クラスタ機能を標準で備える
  - Replication、Sentinel、Cluster

Redisの人気度

- db-engines.comのKey-valueストアランキングで1位
  - https://db-engines.com/en/ranking/key-value+store
- NoSQLジャンルではMongoDB、Elasticsearchに次いで3位



Redisモジュールとは

- Redisに外部モジュールをロードすることで機能拡張できる仕組み
- モジュールはC/C++で実装

Redisモジュールで可能なこと

- 新しいコマンドの作成
- 新しいデータ型の作成
- Redisの既存の機能をモジュールから呼び出す
- レプリケーション、クラスタにも対応できる
  - RDB、AOFの永続化処理を実装すればOK

Redisモジュールのライセンス変更

- Redis Labs社が開発、提供するモジュールのライセンスを変更
  - クラウド業者のフリーライドを防ぐことが狙い
  - 対象は RediSearch, RedisGraph, RedisJSON, RedisML, RedisBloom など
  - AGPL
    → (2018/8) Apache v2.0 modified with Commons Clause
    → (2019/2) Redis Source Available License (RSAL)

- **Redis本体のライセンスは変更なし**
  - 引き続き3条項BSDライセンスが適用
  - 自分で開発したRedisモジュールにも特に制限はなし

RSALの制限事項

- ソースコードの利用・改変・再配布はOK
- Redisモジュールを使ったサービスや製品の販売はOK
- **ただし以下の開発に使用することは禁止**
  - データベース
  - キャッシュエンジン
  - ストリーム処理エンジン
  - 検索エンジン
  - インデックスエンジン
  - 機械学習(ディープラーニング含む)、AIエンジン
  - RedisのAPIやRedis Modules APIを公開する製品やサービス

Redisモジュールの作り方

- 必要なもの
  - Redisの最新版ソースコードに含まれる redismodule.h のみ (ライブラリ等は不要なので楽!)

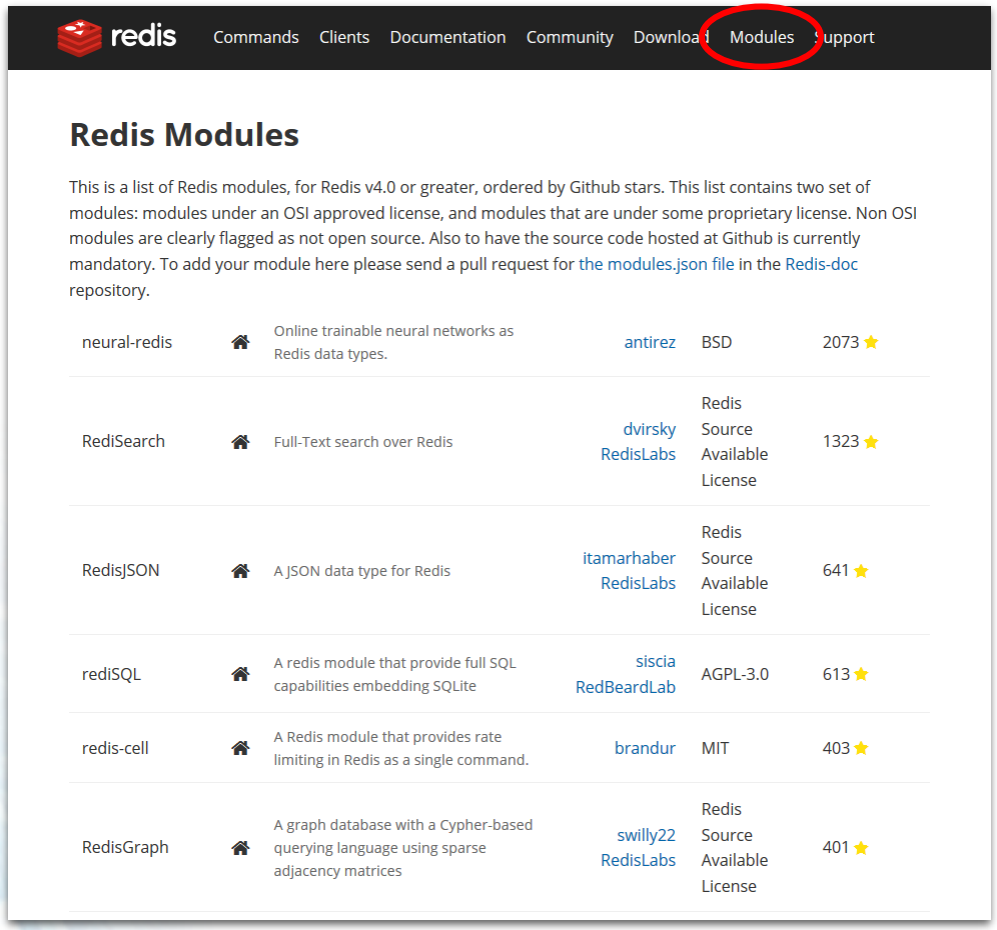
- コンパイル、配置、ロード

```
# gcc --shared -fPIC helloworld.c -o helloworld.so
# cp helloworld.so /var/lib/redis/helloworld.so
# redis-cli module load /var/lib/redis/helloworld.so

(設定ファイルに書く場合は loadmodule /var/lib/redis/helloworld.so)
```

※レプリケーション、クラスタ構成の場合はすべてのサーバに配置する必要がある

公開されているRedisモジュール



Redis Modules
https://redis.io/modules

公開されているRedisモジュールの紹介

- RediSearch … 全文検索
- RedisAI/RedisML/neural-redis … 機械学習系
- RedisGraph … グラフDB
- RediSQL … SQLエンジン

RediSearch

- https://github.com/RediSearch/RediSearch.git
- 全文検索エンジン
- 多言語対応
  - 中国語にも対応しているが、日本語は未対応
- Redis Labs社開発
- RSALライセンス

RediSearchのビルド

- cmake v3以降が必要

```
$ sudo yum install cmake3 --enablerepo=epel
$ git clone https://github.com/RediSearch/RediSearch.git
$ cd RediSearch; mkdir build; cd build
$ cmake3 .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
$ make
$ sudo cp redisearch.so /var/lib/redis/
```

- モジュールの読み込み

```
$ redis-cli module load /var/lib/redis/redisearch.so
```

もしくは redis.conf に以下を追加

```
loadmodule /var/lib/redis/redisearch.so
```

RediSearchを使ってみる

- スキーマを作成

```
127.0.0.1:6379> FT.CREATE myidx schema title text weight 5.0
body text url text
```

- ドキュメントを追加

```
127.0.0.1:6379> FT.ADD myidx doc1 1.0 fields title "Hello
World" body "This is a test document." url "http://localhost"
```

- 検索

```
127.0.0.1:6379> FT.SEARCH myidx "test" limit 0 10
1) (integer) 1
2) "doc1"
3) 1) title
   2) "Hello World"
   3) body
   4) "This is a test document."
   5) url
   6) "http://localhost"
```

RediSearchを使ってみる

- あらかじめ分かち書きをしておけば日本語も一応使える

```
127.0.0.1:6379> FT.CREATE jpidx schema title text weight 5.0 body
text title.orig text weight 0.0 body.orig text weight 0.0
127.0.0.1:6379> FT.ADD jpidx d2 1.0 fields title "日本語 の テスト"
body "これ は 試験 用 の 文章 です 。" title.orig "日本語のテスト"
body.orig "これは試験用の文章です。"
127.0.0.1:6379> FT.SEARCH jpidx 日本語
1
d2
title
日本語 の テスト
body
これ は 試験 用 の 文章 です 。
body.orig
これは試験用の文章です。
title.orig
日本語のテスト
```

RedisAI/RedisML

- RedisAI: ディープラーニング / 機械学習モデルを実装
  - https://oss.redislabs.com/redisai/
  - バックエンドを選択可能(TensorFlow, PyTorch, ONNXRuntime)
  - GPU使用可能
  - 実用性を求める場合はこちら
  - Redis Labs社開発 / RSALライセンス
- RedisML: 機械学習モデルを実装
  - https://oss.redislabs.com/redisml/
  - RedisAIで置き換えられる予定
  - Redis Labs社開発 / RSALライセンス

neural-redis

- neural-redis: 機械学習モデルを実装
  - https://github.com/antirez/neural-redis
  - RedisAI/RedisMLに比べるとシンプルなので取っ付きやすい
  - 他のライブラリに依存せず単体で動作
  - α版というステータス
  - Salvatore Sanfilippo (Redis開発者)が開発
  - 3条項BSDライセンス

neural-redisのビルド

- ソース取得、make、配置

```
$ git clone https://github.com/antirez/neural-redis
$ cd neural-redis/
$ make sse
$ sudo cp neuralredis.so /var/lib/redis/
```

- モジュールの読み込み

```
$ redis-cli module load /var/lib/redis/neuralredis.so
```

neural-redis (1)回帰分析

- 入力データから数値の予測を行う

- 足し算の答えの予測の例 (a + b = c)

neural-redis (1)回帰分析

- ニューラルネットワークの作成

```
127.0.0.1:6379> NR.CREATE net REGRESSOR 2 3 -> 1 NORMALIZE DATASET 50
TEST 10
(integer) 13
```

2入力、1出力で作成

- 学習データを投入

```
127.0.0.1:6379> NR.OBSERVE net 1 2 -> 3
127.0.0.1:6379> NR.OBSERVE net 4 5 -> 9
127.0.0.1:6379> NR.OBSERVE net 3 4 -> 7
127.0.0.1:6379> NR.OBSERVE net 1 1 -> 2
127.0.0.1:6379> NR.OBSERVE net 2 2 -> 4
127.0.0.1:6379> NR.OBSERVE net 0 9 -> 9
127.0.0.1:6379> NR.OBSERVE net 7 5 -> 12
127.0.0.1:6379> NR.OBSERVE net 3 1 -> 4
127.0.0.1:6379> NR.OBSERVE net 5 6 -> 11
```

足し算の例を学習させる

neural-redis (1)回帰分析

- トレーニング開始

```
127.0.0.1:6379> NR.TRAIN net AUTOSTOP
Training has started
```

- トレーニング結果のテスト

```
127.0.0.1:6379> NR.RUN net 1 1
1) "2.0505485534667969"
127.0.0.1:6379> NR.RUN net 4 6
1) "10.419864654541016"
127.0.0.1:6379> NR.RUN net 2 3
1) "4.8927350044250488"
```

それなりに近そうな結果は出た

neural-redis (2)分類

- 入力データをグループ分けする

- タイタニック号の乗客の生存予測
  - neural-redisに同梱されているサンプルの1つ
  - 乗客の各種情報から生存確率を求める
  - 入力パラメータ: 乗客の各種属性
    - チケットクラス、性別、年齢、兄弟・親子の乗船数、チケット料金
  - 分類ラベル: 生存 or 死亡

neural-redis (2)分類

- 付属のRubyスクリプトで学習データ(CSV形式の乗客名簿)の投入とトレーニング

```
$ cd neural-redis/examples
$ ruby titanic.rb
Before training
185 prediction errors on 290 items
Still training...
After training
47 prediction errors on 290 items
```

neural-redis (2)分類

- トレーニング結果のテスト
- ファーストクラス、女性、30歳、兄弟・親子同乗なしの場合

```
127.0.0.1:6379> NR.RUN mynet 1 0 0 0 1 30 0 0 200
1) "0.06613945960998535"
2) "0.9496233463287353"
```

95%の確率で生存

- 70歳の場合

```
127.0.0.1:6379> NR.RUN mynet 1 0 0 0 1 70 0 0 200
1) "0.08922017365694046"
2) "0.925819993019104"
```

92.5%に減少

- 70歳、サードクラス、格安チケットの場合

```
127.0.0.1:6379> NR.RUN mynet 0 0 1 0 1 70 0 0 20
1) "0.6508185863494873"
2) "0.3200186192989349"
```

32%に減少

RedisGraph

- https://oss.redislabs.com/redisgraph/
- RedisにグラフDB機能を追加
- neo4jのグラフクエリー言語Cypherを採用
- Redis Labs社開発
- RSALライセンス

RedisGraphの実行

- dockerイメージを使用

```
# docker run -p 6379:6379 -it --rm redislabs/redisgraph
```

RedisGraphを使ってみる

- グラフを作成 (例: 人同士の関係)

```
127.0.0.1:6379> GRAPH.QUERY People
 "CREATE (:Person {name:'John'})-[:friend]->(:Person {name:'Fred'}),
         (:Person {name:'Lisa'})-[:friend]->(:Person {name:'Mary'})"
1) 1) "Labels added: 1"
   2) "Nodes created: 4"
   3) "Properties set: 4"
   4) "Relationships created: 2"
   5) "Query internal execution time: 3.208423 milliseconds"
```

RedisGraphを使ってみる

- クエリ実行 (例: Fred の友人を検索)

```
127.0.0.1:6379> GRAPH.QUERY People
 "MATCH (p1:Person)-[:friend]->(p2:Person)
  WHERE p2.name = 'Fred'
  RETURN p1.name, p2.name"
1) 1) "p1.name"
   2) "p2.name"
2) 1) 1) "John"
      2) "Fred"
3) 1) "Query internal execution time: 0.326652 milliseconds"
```

RedisGraphを使ってみる

- クエリ実行 (例: Mary の友人の数をカウント)

```
127.0.0.1:6379> GRAPH.QUERY People
 "MATCH (p1:Person)-[:friend]->(p2:Person {name:'Mary'})
  RETURN count(p1)"
1) 1) "count(p1)"
2) 1) 1) (integer) 1
3) 1) "Query internal execution time: 0.290916 milliseconds"
```

RediSQL

- https://redisql.com/
- https://redisql.redbeardlab.com/rediSQL/
- SQLiteベースのインメモリSQLエンジン
- Rust言語で実装
- Red Beard Lab社開発
- MITライクなカスタムライセンス
- フリー版は統計情報をサーバに送る必要がある(PRO版は不要)

RediSQLの実行

- dockerイメージを使用

```
# docker run -it --net host siscia/redisql
```

※直接外部ネットワークにつながっている必要あり

RediSQLを使ってみる

- DB作成

```
127.0.0.1:6379> REDISQL.CREATE_DB DB
```

- テーブル作成

```
127.0.0.1:6379> REDISQL.EXEC DB "CREATE TABLE foo(A INT, B TEXT);"
```

- データをテーブルに挿入

```
127.0.0.1:6379> REDISQL.EXEC DB "INSERT INTO foo VALUES(3, 'bar');"
```

- テーブルからデータを取得

```
127.0.0.1:6379> REDISQL.EXEC DB "SELECT * FROM foo;"
1) 1) (integer) 3
   2) "bar"
```

まとめ

- RedisモジュールでRedisの自由な機能拡張が可能
- できることはほぼ無制限
  - Redisサーバのメモリやストレージの制限くらい
  - RedisをKey-valueストア以外のあらゆる用途に使えるようになる
- 既に様々な高機能なモジュールが提供されており
  Redisを多様な用途に利用可能
  - 全文検索、機械学習、グラフDB、etc...
- モジュールのライセンスの制限には注意

オープンソースとともに

SRA OSS, INC.