GO-13 02: Redis

Описание:

В прошлом модуле вы узнали, что такое in-memory хранилище. Но это не всегда верный подход к хранению данных. В основном при разработке приложения мы хотим хранить данные в памяти, если они нам очень важны и мы часто к ним обращаемся. Чаще всего - это какой-то кеш.
У этого подхода есть ряд проблем:

1. Данные теряются, если сервис аварийно завершил свое выполнение.
2. Оперативная память конечна, иногда ее переполнение может создать для нас ряд понятных проблем - несохранность данных, потеря данных, малый объем возможного хранения данных.
3. Сложно масштабировать, так как данные в разных инстансах будут неконсистентны.
4. Нужно заботиться о race condition.

Решение было найдено в написании отдельной СУБД для хранения данных в памяти, при этом эта СУБД решает проблемы, которые описаны выше. Такое решение называется Redis.
Redis:
Redis - это key value хранилище данных.
Все данные Redis хранит в виде словаря, в котором ключи связаны со своими значениями. Одно из ключевых отличий Redis от других хранилищ данных заключается в том, что значения этих ключей не ограничиваются строками. Поддерживаются следующие абстрактные типы данных: строки, списки, множества, хеш-таблицы, упорядоченные множества.
Redis имеет утилиты, которые облегчают работу:

- redis-server - сам сервер;
- redis-sentinel - мониторинг, failover и другое;
- redis-cli - CLI для работы с Redis;
- redis-benchmark - проверка производительности;
- redis-check-aof и redis-check-dump - утилиты для работы с поврежденными файлами данных.

Полную документацию можно прочитать на официальном сайте.
Также redis позволяет реализовать модель pub/sub. В данном случаи Redis может рассматриваться как аналог Kafka.
Основным инструментом для работы с Redis из Golang кода является вот эта библиотека.
Также существуют и другие способы подключиться к redis. Полный список языков и пакетов хранится вот тут.
Давайте посмотрим, как работает Redis. Для начала подключимся к нему:

```
opt := redis.Options{
```

```
        Addr:     "your redis addr",
        Password: "",
        DB:       0,
    }

redisClient := redis.NewClient(&opt)
```

На выкладке выше представлен стандартный способ подключения к Redis. Addr - это адрес до вашего redis сервера. Стандартный порт для подключения - :6379. Пункт DB указывает на то, что используется стандартная база данных.
Теперь давайте посмотрим, как можно установить и считать значение, положенное в redis:

```
err := redisClient.Set(ctx, "key", "value", 0).Err()
if err != nil {
   log.Error(err)
}

val, err := redisClient.Get(ctx, "key").Result()
if err != nil {
   log.Error(err)
}
fmt.Println("key", val)
```

Как получить и считать значение, теперь ясно, но что будет, если значение пусто? Для этого в Redis есть специальный тип ошибки - redis.Nil. На ней можно проверить пришедшее нам значение.

```
val2, err := redisClient.Get(ctx, "key2").Result()
if err == redis.Nil {
      fmt.Println("key2 does not exist")
} else if err != nil {
   panic(err)
} else {
   fmt.Println("key2", val2)
}
```

Также можно, например, использовать кастомные команды вместо готовых примеров:

```
res, err := redisClient.Do(ctx, "set", "key", "value").Result()
```

Не нужно забывать, что мы работаем с хранилищем, которое использует оперативную память сервера, иногда полезно проверять показатели Redis. Для пакета предоставляется следующий инструмент:

```
redisClient.Info()
```

Redis in Docker-Compose
С использованием кода ниже можно поднять Redis с помощью docker. Просто используйте docker-compose up -d. Если вы захотите поднять его рядом со своим кодом, то обращаться к redis можно прямо по имени контейнера, так как они находятся в одной сети.

```
redis:
    container_name: redis
    image: redis:4.0-alpine
    command:
      - 'redis-server'
      - '--loglevel ${REDIS_LOGLEVEL:-warning}'
      - '--databases 2'
      - '--maxmemory ${REDIS_MAXMEM:-50mb}'
      - '--maxmemory-policy ${REDIS_POLICY:-noeviction}'
      - '--requirepass ${REDIS_PASS}'
    volumes:
      - redis:/data
    ports:
      - "6379:6379"
```

Полезные ссылки:

- Шапаргалки по Redis
- Отличный пример использования Redis в highload

Задание:

За основу мы возьмем код из прошлого задания. Ваша задача будет переписать наш нативный кеш и вместо него использовать redis. Redis нужно поднять рядом с помощью Docker Compose.

1. В вашем проекте module13 сделайте новую ветку 02_task.
2. Переработайте использование нативного кеша, используйте redis вместо него.
3. Для запуска Redis вместе с вашим сервисом используйте Docker Compose.
4. В ответе пришлите ссылку на merge request в ветку master своего проекта ветки 02_task.