Redis

原文：ENG-18-Redis.md

Drogon 支援 Redis，一款高速記憶體資料庫，可用於資料庫快取或訊息代理。Drogon 的 Redis 連線皆為非同步，確保高併發效能。

Redis 支援需安裝 `hiredis` 套件。若建置時未安裝 hiredis，則無法使用 Redis 功能。

建立 client

可透過 `app()` 動態建立與取得 Redis client：

```
app().createRedisClient("127.0.0.1", 6379);
...
// app.run() 之後
RedisClientPtr redisClient = app().getRedisClient();
```

亦可於設定檔建立 Redis client：

```
    "redis_clients": [
        {
            //name: client 名稱，預設 'default'
            //"name":"",
            //host: 伺服器 IP，預設 127.0.0.1
            "host": "127.0.0.1",
            //port: 伺服器埠號，預設 6379
            "port": 6379,
            //passwd: 預設空字串
            "passwd": "",
            //db index: 預設 0
            "db": 0,
            //is_fast: 預設 false，true 時效能更高但不可呼叫同步介面，且僅限 IO 執行緒與主執行緒使用
            "is_fast": false,
            //number_of_connections: 預設 1，is_fast 為 true 時為每 IO 執行緒連線數，否則為總連線數
            "number_of_connections": 1,
            //timeout: 預設 -1.0（秒），命令執行逾時，0 或負值表示無逾時
            "timeout": -1.0
        }
    ]
```

使用 Redis

execCommandAsync 以非同步方式執行 Redis 指令，至少需三個參數：成功與失敗 callback，以及指令本身。指令可為 C 風格格式字串，後續為格式字串參數。例如設定 key `name` 為 `drogon`：

```
redisClient->execCommandAsync(
    [](const drogon::nosql::RedisResult &r) {},
    [](const std::exception &err) {
        LOG_ERROR << "something failed!!! " << err.what();
    },
    "set name drogon");
```

或設定 `myid` 為 `587d-4709-86e4`

```
redisClient->execCommandAsync(
    [](const drogon::nosql::RedisResult &r) {},
    [](const std::exception &err) {
        LOG_ERROR << "something failed!!! " << err.what();
    },
    "set myid %s", "587d-4709-86e4");
```

同樣 execCommandAsync 也可讀取 Redis 資料：

```
redisClient->execCommandAsync(
    [](const drogon::nosql::RedisResult &r) {
        if (r.type() == RedisResultType::kNil)
            LOG_INFO << "找不到 key 'name' 對應的值";
        else
            LOG_INFO << "Name is " << r.asString();
    },
    [](const std::exception &err) {
        LOG_ERROR << "something failed!!! " << err.what();
    },
    "get name");
```

交易（Transaction）

Redis 交易可一次執行多個指令，所有指令依序執行，期間不會插入其他 client 指令。注意交易非原子性，收到 exec 指令後即執行，若有指令失敗，剩餘指令仍會執行，且不支援 rollback。

newTransactionAsync 建立新交易，交易物件用法同 RedisClient，最後以 RedisTransaction::execute 執行交易。

```
redisClient->newTransactionAsync([](const RedisTransactionPtr &transPtr) {
    transPtr->execCommandAsync(
        [](const drogon::nosql::RedisResult &r) { /* 指令成功 */ },
        [](const std::exception &err) { /* 指令失敗 */ },
    "set name drogon");

    transPtr->execute(
        [](const drogon::nosql::RedisResult &r) { /* 交易成功 */ },
```

```
        [](const std::exception &err) { /* 交易失敗 */ });
});
```

協程

Redis client 支援協程。建議使用 GCC 11 或更新版本，並以 `cmake -DCMAKE_CXX_FLAGS="-std=c++20"` 啟用。詳見 協程。

```
try
{
    auto transaction = co_await redisClient->newTransactionCoro();
    co_await transaction->execCommandCoro("set zzz 123");
    co_await transaction->execCommandCoro("set mening 42");
    co_await transaction->executeCoro();
}
catch(const std::exception& e)
{
    LOG_ERROR << "Redis failed: " << e.what();
}
```

下一步: 測試框架