DB相關

- 【Redis】相關
- 【Redis】相關指令
- 資料庫型態比較
- DB 相關資源
- 【Debian】連線MariaDB
- 【Redis】設定相關
- 【Redis】安裝
- 【Redis相關】RedisSDK
- DB 欄位命名與比較

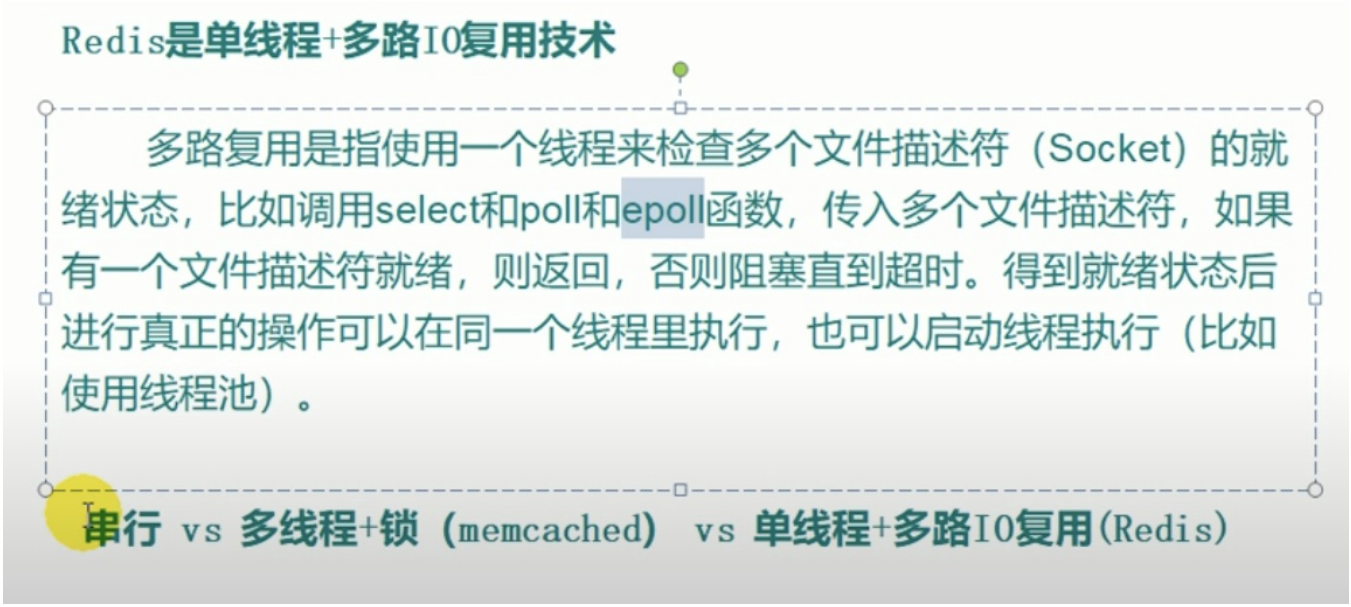
【Redis】相關

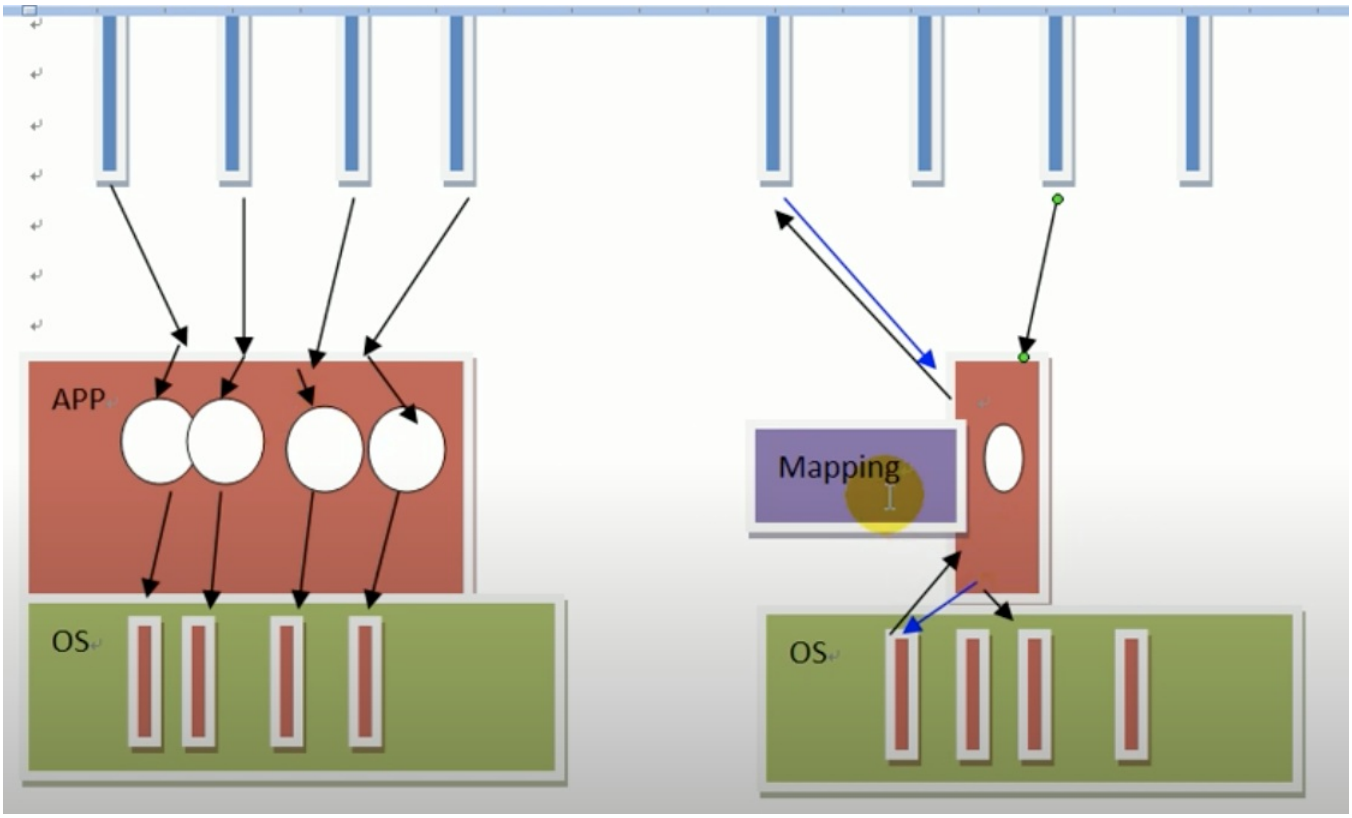
0 - 15 共16個資料庫 0為預設，使用select 切換

Redis 為何快

使用記憶體，單執行序，多路io複用

- NIO
- BIO
- AIO





【尚硅谷】2021最新Redis 6教程分布式□，秒□实□

- youtube
  https://www.youtube.com/playlist?list=PLmOn9nNkQxJHsATQDTW7Ci6X-Rcypgk2G

- 相關筆記
  https://s8jl-my.sharepoint.com/personal/atguigu_s8jl_onmicrosoft_com/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fatguigu%5Fs8jl%5Fonmicrosoft%5Fcom%2FDocuments%2F%E5%B0%9A%E7%A1%85%E8%B

【Redis】相關指令

相關指令

連線

```
# redis-cli -h {$host} -p {$port} -a password
# ping 檢測服務正常
$redis-cli -h 127.0.0.1 -p 6379 -a "mypass"
redis 127.0.0.1:6379>
redis 127.0.0.1:6379> PING

PONG
```

使用redis-cli 進入redis

```
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]>
```

進入db (具有16個

數據庫（DB）。這些數據庫編號從0到15，用於存儲不同的數據)

```
root@62952ea40eb3:/data# redis-cli
127.0.0.1:6379>
```

查詢key (線上環境勿用， 請用scan)

```
#keys pattern
127.0.0.1:6379[2]> keys *
1) "v5.25.1_autocompleteKeywordV3_ASU"
2) "v5.25.1_autocompleteKeywordV3_AU"
3) "v5.25.1_autocompleteKeywordV3_ASUS"
4) "v5.25.1_autocompleteKeywordV3_AUS"
5) "v5.25.1_autocompleteKeywordV3_AS"
6) "v5.25.1_autocompleteKeywordV3_A"
7) "v5.25.1_autocompleteKeywordV3_AUU"
127.0.0.1:6379[2]>

########################################
127.0.0.1:6379[2]> keys *autocompleteKeywordV3*
1) "v5.25.1_autocompleteKeywordV3_\xe9\x80\x81\xe5\xa5\xbd\xe7\xa6\xae"
2) "v5.25.1_autocompleteKeywordV3_\xe5\xb8\xb8\xe6\x98\xa5"
3) "v5.25.1_autocompleteKeywordV3_ASUS"
4) "v5.25.1_autocompleteKeywordV3_\xe5\xb8\xb8"
5) "v5.25.1_autocompleteKeywordV3_\xe7\x9b\xb4\xe6\x92\xad"
6) "v5.25.1_autocompleteKeywordV3_\xe7\x9b\xb4"
7) "v5.25.1_autocompleteKeywordV3_\xe9\x80\x81\xe5\xa5\xbd"
8) "v5.25.1_autocompleteKeywordV3_\xe7\x9b\xb4\xe6\x92\xad\xe6\xb8\x85\xe5\x96\xae"
9) "v5.25.1_autocompleteKeywordV3_\xe9\x80\x81"
```

查詢 sacn

```
# scan {inidex} match {pattern}
#返回分□兩个部分如上面的代□中， 1）代表下一次迭代的游□，2）代表本次迭代的□果集
#，注意如果返回游□□0就代表全部匹配完成。
127.0.0.1:6379> scan 0 MATCH tony*
1) "42"
2)  1) "tony25"
    2) "tony2519"
    3) "tony2529"
    4) "tony2510"
    5) "tony2523"
    6) "tony255"
```

```
    7) "tony2514"
    8) "tony256"
    9) "tony2511"
   10) "tony15"
127.0.0.1:6379> scan 42 MATCH tony* COUNT 1000
1) "0"
2)  1) "tony3513"
    2) "tony359"
    3) "tony4521"
    4) "tony356"
    5) "tony30"
    6) "tony320"
    7) "tony3"
    8) "tony312"

#批次刪除key
#redis-cli --scan --pattern "key前□*" | xargs -L {筆數} redis-cli del
redis-cli --scan --pattern "tony*" | xargs -L 1000 redis-cli del
```

查詢資料

```
#get {key}
127.0.0.1:6379[2]> get v5.25.1_autocompleteKeywordV3_ASUS
"\x1f\x8b\b\x00\x00\x00\x00\x00\x00\x00\xc4W\xfdo\xdbD\x18\xfeW,\xf7\xd7.\xb9;;\x8e]\t\xa1.t\xa5b\x0b\xd5R`R\xd5\x1f\\\xe7\xe6Z\x8
\xf5\xb1s\xe3}\xd5\xddH\xbe\x8d\x04P\x04\x12\x94\x10\x02\x05ID\x17.(\xc5\"
(\x14\x059\x80u}M\xa3\x98\xfc\x9c\xe7\xf8x\x96\xc7u\xac\xf9\x01\xd8\x8aa\xd2o\xcbq\x92\xcb\x86\xeb\xf1s\xab\r^S=\\V\x83\x18?
_\xf9\xa8\xc2\xf5n?
\xe9>|D\x81\xb6\x89S5,\x9d\xce\xb7\xbf\xfe\xac\xdb\xfa\xb2sp\xd0k\xfd\xd8\xfe\xe3`\x18+\x91\x1aq\xe8\x82\x19A\x90\xa4K\x97h\xc0
\x10\x05\x10\xfc\xe4lK\x7f\xd7{\aJ\xb2,\t\xb0\x88\xa8\xc8@\x99\xaa\r6\xa5\x11\x8dVn\xd8\x945\x04q\xe4c\xb5\xe6\x87\x1b\x94\x00\x
<\x1f\x911\xd0@\xa4\xee\x10\xdf.\x13\x1a\x01\x90\xc2x\xb8N\r\xcc\xab\xae\xefr[\xc6\x16Y'd\x93\xb3\x1d\xc2A\x89#5\\\xa5K4\x8b\xa
\xb0\x92\x8a\xe1N\xf1h\xc4Y?
1\xac*\xd9\xa6+\xe1Y\xb1\x83\xce\xe19j\x14k\xf0T\xbf\x8d\x1d\xcf\xc0\x83<\xe6\xbc\xe7]\xa6C\xfe\xd5_\x0f:?
<\xee\xfe\xf6\xa2\xfb\xeci\xd0}\xa2\xf9\xd54\x85\xb5Y\xde\"#\x0233\xf3\x00@e\x91\x1e1*\xd1\x91\xc4\xaf5\xd7\x9a\xe1aG:\x96,\x0f\
\x0c\x92\x85\xc3\xa8Y\r3\x0eflx\xbdAz\x99!\x01\xd2\x0b\x0e\n\xb4\x97\xf3\x93v1\xcc\xba\xae\x88\x05p]Mv\x0e\xa56\x0e\xb1:'\x06\x1
iNJ\xaa7)\xac\x12#,\xe6\x0e\xb9\x83\xad\xb0\xcd\xd2\x9a\xf5\xeb\x82\b\xd0Nr\x94b\xaa\x06D\xe6&0\n\x98\x85\x9d\x8d\xf5\x81\xab\x
\x16Nl)I)_J\x12\xab1\x87\x80\xcc\\
<\xff:\xa7\xabf\xf0\xaa\x8b\xc9\xc80EFf\xbe\xf5S\x88\xe3\x8dxu\xe5\x1a'\x02z\xd5\xb7\xbf8x\x83\x04\xd3v\xbf\xce\xe1\xf3n\xebV\xa6\


# 設定ttl
# expire {key} {秒數}
expire v5.25.1_autocompleteKeywordV3_ASUS 60

#獲得值的ttl
#ttl {key}
ttl v5.25.1_autocompleteKeywordV3_ASUS
```

查詢該DB 資料總數

```
# dbsize 該db儲存數
127.0.0.1:6379[2]> dbsize
(integer) 9
```

刪除該DB 所有資料

```
#清除目前db 所有資料
# flushdb

# 原本db有8筆資料
127.0.0.1:6379[2]> dbsize
(integer) 8

# 清除db
127.0.0.1:6379[2]> flushdb
OK

# 重新查詢已無資料
127.0.0.1:6379[2]> dbsize
```

```
(integer) 0
```

刪除該db資料

```
# del key1 key2 ...
127.0.0.1:6379[2]> keys *
1) "v5.25.1_autocompleteKeywordV3_\xe9\x80\x81\xe5\xa5\xbd\xe7\xa6\xae"
2) "v5.25.1_autocompleteKeywordV3_\xe5\xb8\xb8\xe6\x98\xa5"
3) "v5.25.1_autocompleteKeywordV3_ASUS"
4) "v5.25.1_autocompleteKeywordV3_\xe5\xb8\xb8"
5) "v5.25.1_autocompleteKeywordV3_\xe7\x9b\xb4\xe6\x92\xad"
6) "v5.25.1_autocompleteKeywordV3_\xe7\x9b\xb4"
7) "v5.25.1_autocompleteKeywordV3_\xe9\x80\x81\xe5\xa5\xbd"
8) "v5.25.1_autocompleteKeywordV3_\xe7\x9b\xb4\xe6\x92\xad\xe6\xb8\x85\xe5\x96\xae"
9) "v5.25.1_autocompleteKeywordV3_\xe9\x80\x81"
127.0.0.1:6379[2]> del v5.25.1_autocompleteKeywordV3_ASUS
(integer) 1
127.0.0.1:6379[2]> keys *
1) "v5.25.1_autocompleteKeywordV3_\xe9\x80\x81\xe5\xa5\xbd\xe7\xa6\xae"
2) "v5.25.1_autocompleteKeywordV3_\xe5\xb8\xb8\xe6\x98\xa5"
3) "v5.25.1_autocompleteKeywordV3_\xe5\xb8\xb8"
4) "v5.25.1_autocompleteKeywordV3_\xe7\x9b\xb4\xe6\x92\xad"
5) "v5.25.1_autocompleteKeywordV3_\xe7\x9b\xb4"
6) "v5.25.1_autocompleteKeywordV3_\xe9\x80\x81\xe5\xa5\xbd"
7) "v5.25.1_autocompleteKeywordV3_\xe7\x9b\xb4\xe6\x92\xad\xe6\xb8\x85\xe5\x96\xae"
8) "v5.25.1_autocompleteKeywordV3_\xe9\x80\x81"
127.0.0.1:6379[2]>
```

刪除所有redis資料

```
# 清除整台redis 資料
127.0.0.1:6379[2]> flushall
OK
```

寫入資料(字串)


```
set <key> <value>: 寫入資料
NX：當數據庫中的鍵不存在時，可以將鍵-值添加到數據庫
XX：當數據庫中的鍵存在時，可以將鍵-值添加到數據庫，與 NX 參數互斥
EX：鍵的超時秒數
PX：鍵的超時毫秒數，與 EX 互斥
get <key>：查詢對應鍵值
append <key> <value>：將給定的 <value> 追加到原值的末尾
strlen <key>：獲得值的長度
setnx <key> <value>：只有在鍵不存在時設置鍵的值

incr <key>：將鍵中儲存的數字值增加1
只能對數字值操作，如果為空，新增值為1
decr <key>：將鍵中儲存的數字值減少1
只能對數字值操作，如果為空，新增值為-1
incrby / decrby <key> <步長>：將鍵中儲存的數字值增減。自定義步長。

mset <key1> <value1> <key2> <value2> ...：同時設置一個或多個鍵-值對
mget <key1> <key2> <key3> ...：同時獲取一個或多個值
msetnx <key1> <value1> <key2> <value2> ...：同時設置一個或多個鍵-值對，當且僅當所有給定的鍵都不存在。原子性，有一個失敗則都失敗

getrange <key> <起始位置> <結束位置>：獲得值的範圍，類似於 Java 中的 substring，前包，後包
setrange <key> <起始位置> <value>：用 <value> 覆蓋 <key> 所儲存的字符串值，從 <起始位置> 開始（索引從0開始）。

setex <key> <過期時間> <value>：設置鍵值的同時，設置過期時間，單位秒。
getset <key> <value>：以新換舊，設置了新值同時獲得舊值。
```

寫入資料(List)

lpush/rpush <key> <value1> <value2> <value3> ...：從左邊/右邊插入一個或多個值。
lpop/rpop <key>：從左邊/右邊吐出一個值。值在鍵在，值光鍵亡。

rpoplpush <key1> <key2>：從 <key1> 列表右邊吐出一個值，插到 <key2> 列表左邊。

lrange <key> <start> <stop>：按照索引下標獲得元素（從左到右）。
lrange mylist 0 -1：0 左邊第一個，-1 右邊第一個，（0-1 表示獲取所有）。
lindex <key> <index>：按照索引下標獲得元素（從左到右）。
llen <key>：獲得列表長度。

linsert <key> before <value> <newvalue>：在 <value> 的後面插入 <newvalue>。
lrem <key> <n> <value>：從左邊刪除 n 個 value（從左到右）。
lset <key> <index> <value>：將列表 key 下標為 index 的值替換成 value。

寫入資料(Set)

sadd <key> <value1> <value2> ...：將一個或多個 member 元素加入到集合 key 中，已經存在的 member 元素將被忽略。
smembers <key>：取出該集合的所有值。
sismember <key> <value>：判斷集合 <key> 是否含有該 <value> 值，有1，沒有0。
scard <key>：返回該集合的元素個數。
srem <key> <value1> <value2> ...：刪除集合中的某個元素。
spop <key>：隨機從該集合中吐出一個值。
srandmember <key> <n>：隨機從該集合中取出 n 個值。不會從集合中刪除。
smove <source> <destination> <value>：將集合中一個值從一個集合移動到另一個集合。
sinter <key1> <key2>：返回兩個集合的交集元素。
sunion <key1> <key2>：返回兩個集合的聯集元素。
sdiff <key1> <key2>：返回兩個集合的差集元素（key1 中的，不包含 key2 中的）。

寫入資料(Zset)

zadd <key> <score1> <value1> <score2> <value2>...：將一個或多個 member 元素及其 score 值加入到有序集 key 當中。
zrange <key> <start> <stop> [WITHSCORES]：返回有序集 key 中，下標在 <start> 到 <stop> 之間的元素。帶有 WITHSCORES，可以讓分數一起和值返回到結果集。
zrangebyscore <key> <minmax> [withscores] [limit offset count]：返回有序集 key 中，所有 score 值介於 min 和 max 之間（包括等於 min 或 max）的成員。有序集成員按 score 值遞增（從小到大）次序排列。
zrevrangebyscore <key> <maxmin> [withscores] [limit offset count]：同上，改為從大到小排列。
zincrby <key> <increment> <value>：為元素的 score 加上增量。
zrem <key> <value>：刪除該集合下，指定值的元素。
zcount <key> <min> <max>：統計該集合，分數區間內的元素個數。
zrank <key> <value>：返回該值在集合中的排名，從0開始。

寫入資料(Hash)

hset <key> <field> <value>：給 <key> 集合中的 <field> 鍵賦值 <value>
hget <key1> <field>：從 <key1> 集合 <field> 取出 value
hmset <key1> <field1> <value1> <field2> <value2>...：批量設置 hash 的值
hexists <key1> <field>：查看哈希表 key 中，給定域 field 是否存在。
hkeys <key>：列出該 hash 集合的所有 field
hvals <key>：列出該 hash 集合的所有 value
hincrby <key> <field> <increment>：為哈希表 key 中的域 field 的值加上增量 1 或 -1
hsetnx <key> <field> <value>：將哈希表 key 中的域 field 的值設置為 value，當且僅當域 field 不存在。

相關資源

- 在線練習 https://try.redis.io/
- 文檔總覽 http://www.redis.cn/documentation.html
- 命令文檔 http://www.redis.cn/commands.html
- docker中安装并配置redis https://cloud.tencent.com/developer/article/1670205
- 【Redis】用Docker-Compose搭建Redis Sentinel
- 【Redis】用Docker-Compose搭建Redis主从复制

資料庫型態比較

DBMS DATA TYPE

DATA TYPE	SQL 92	Oracle	MySQL	POSTGRE	M$ SQL	Common
BOOLEAN	BOOLEAN			BOOL		
CHARACTER	CHAR VARCHAR	CHAR VARCHAR2 NVARCHAR2 LONG TEXT ENUM SET	CHAR VARCHAR BLOB TEXT ENUM SET	CHAR VARCHAR TEXT NAME CHARACTER	CHAR VARCHAR TEXT	CHAR VARCHAR TEXT
BIT	BIT BIT VARYING				BIT	
EXACT NUMERIC	INTEGER SMALLINT DECIMAL NUMERIC	INTEGER NUMBER NUMERIC SMALLINT DECIMAL	INTEGER SMALLINT DECIMAL NUMERIC	INTEGER DECIMAL INT2 INT4 INT8 NUMERIC SERIAL	INTEGER BIGINT SMALLINT TINYINT NUMERIC DECIMAL	INTEGER NUMERIC DECIMAL
APPROXIMATE NUMERIC	FLOAT REAL DOUBLE PRECISION	FLOAT REAL DOUBLE PRECISION	FLOAT REAL DOUBLE PRECISION	FLOAT REAL DOUBLE PRECISION FLOAT4 FLOAT8	FLOAT REAL DOUBLE PRECISION	FLOAT REAL DOUBLE RECISION
DATETIME	DATE TIME TIMESTAMP	DATE TIME DATETIME TIMESTAMP YEAR	DATE TIME DATETIME TIMESTAMP YEAR	DATE TIME TIMETZ TIMESTAMP CURRENT EPOCH INFINITY INVALID NOW TODAY TOMORROW YESTERDAY	DATE TIME SMALLDATATIME TIMESTAMP UNIQUEIDENTIFIER	TIMESTAMP DATE
INTERVAL	INTERVAL			INTERVAL		
BINARY		ROWID NROWID RAW LONG RAW			BINARY VARBINARY IMAGE	
LARGE OBJECTS	CHARACTER LARGE OBJECTBINARY LARGE OBJECT	CHARACTER LARGE OBJECT BINARY LARGE OBJECT NATIONAL LARGE OBJECT				
NET				CIDR INET		
Geometric				BOX CIRCLE LINE LSEG PATH POINT POLYGON SERIAL		

來源：資料型態整理 Oracle, MySQL, PostgreSQL, M$ SQL - DBMS DATA TYPE - Soul & Shell Blog (toright.com)

DB 相關資源

- DB 排名 https://db-engines.com/en/ranking

【Debian】連線MariaDB

在Debian上安裝ODBC連接MariaDB，您可以按照以下步驟進行操作：

1. 確保您的Debian系統已連接到互聯網，並且具有管理員權限（或以root用戶身份登錄）。
2. 打開終端，執行以下命令更新匯總列表：
   ```
   sudo apt update
   ```
3. 安裝ODBC驅動程序及相關工具：
   ```
   sudo apt install unixODBC unixODBC-dev
   ```
4. 下載並安裝MariaDB的ODBC驅動程序。首先，訪問MariaDB的官方下載頁面（https://downloads.mariadb.org/connector-odbc/）並選擇適合您Debian版本的驅動程序。然後使用wget命令下載驅動程序的安裝包。請選擇以下命令中的URL替換為您的驅動程序的下載鏈接：
   ```
   wget <□動程序的下□□接>
   ```
5. 解壓驅動程序的安裝包。使用以下命令，將`<package-name>`替換為您下載的安裝包的名稱：
   ```
   tar xvf <package-name>.tar.gz
   ```
6. 進入解壓後的目錄：
   ```
   cd <package-name>
   ```
7. 使用以下命令執行驅動程序的安裝。請注意，這裡的命令可能會因驅動程序的版本和您系統的配置而有所不同。確保按照驅動程序的安裝說明進行操作。
   ```
   sudo ./configure
   sudo make
   sudo make install
   ```
8. 配置ODBC連接。使用文本編輯器（如nano或vi）打開ODBC配置文件`/etc/odbcinst.ini`：
   ```
   sudo nano /etc/odbcinst.ini
   ```
9. 在配置文件中添加以下內容，以指定MariaDB的ODBC驅動程序：
   ```
   [MariaDB]
   Description = MariaDB ODBC driver
   Driver = <□動程序的完整路□>
   ```
   命令`<□動程序的完整路□>`替換為您安裝的驅動程序的實際路徑。保存並關閉文件。
10. 配置ODBC數據源。使用文本編輯器打開ODBC數據源配置文件`/etc/odbc.ini`：
    ```
    sudo nano /etc/odbc.ini
    ```
11. 在文件中添加以下內容，以指定您的MariaDB數據庫連接信息：
    ```
    [ODBC Data Source]
    Driver = MariaDB
    Server = <□据□服务器地址>
    Port = <□据□端口号>
    Database = <□据□名□>
    User = <用户名>
    Password = <密□>
    ```
    將`<□据□服务器地址>`、`<□据□端口号>`、`<□据□名□>`、`<用户名>`和`<密□>`替換為您實際的數據庫連接信息。保存並關閉文件。
12. 現在，您應該能夠使用ODBC連接到MariaDB數據庫了。您可以使用ODBC工具或編程語言中的ODBC庫來建立連接並執行查詢。

請注意，這些步驟提供了基本的安裝和配置過程，具體取決於您使用的驅動程序版本和系統配置。您可以根據驅動程序的安裝說明進行操作，並在遇到問題時參考相關文檔和資源。

【Redis】設定相關

/etc/redis.conf

or

/usr/local/etc/redis/redis.conf

- ## 開放外部連線

```
# 只支持本地
bind 127.0.0.1 -::1
protected-mode yes

# 要支援遠端存取註解 bind 127.0.0.1 -::1
# protected-mode no
# bind 127.0.0.1 -::1
protected-mode no
```

其他設定

```
# 未完成三向交握＋已完成三向交握上限
tcp-backlog 511

# Close the connection after a client is idle for N seconds (0 to disable)
# 0: 永不過期
timeout 0

# 檢測連線存在的間隔時間
tcp-keepalive 300

# 使用daemon啟動
daemonize yes

# pid檔案位置
# 如果指定了pid文件，Redis在啟動時會將其寫入指定位置，並在退出時刪除它。
# 當服務器以非守護模式運行時，如果在配置中未指定pid文件，則不會創建pid文件。當服務器以守護模式運行時，即使未指定，也將使用pid文件，默認為 "/var/run/redis.pid"。
# 創建pid文件是盡力而為的：如果Redis無法創建它，不會發生任何壞事，服務器將正常啟動和運行。
# 請注意，在現代Linux系統上，應改用 "/run/redis.pid"，因為這更符合規範。
pidfile /var/run/redis_6379.pid

# log 級別
# debug (a lot of information, useful for development/testing)
# verbose (many rarely useful info, but not a mess like the debug level)
# notice (moderately verbose, what you want in production probably)
# warning (only very important / critical messages are logged)
loglevel notice

# 設定db數量
databases 16

# 設置密碼
# requirepass {密碼}
requirepass foobared

# 使用命令臨時設定密碼(服務重啟失效)
127.0.0.1:6379>config set requirepass "123456"
# 登入
127.0.0.1:6379>auth 123456

# 最大連接數(預設無)
maxclients 10000
# 最大記憶體使用量(預設無)
maxmemory <bytes>
```

持久化(快照)(RDB)

```
# 持久化(快照)(REB)
# 有可能最後一次資料丟失
# 寫入檔名
dbfilename dump.rdb
# 寫入目錄(啟動redis命令當前目錄)
dir ./
# 快照設定

# 3600秒內有1個key異動
# save 3600 1
# 300秒內有100個key異動
# save 300 100
# 60秒內有10000個key異動
# save 60 10000

# Redis 無法寫入檔案，關閉寫入操作
stop-writes-on-bgsave-error yes
# 文件進行壓縮
rdbcompression yes
# 文件檢查(影響10%效能)
rdbchecksum yes

# 如何恢復 ：
# 關閉 redis 將原先dump.rdb 放回執行目錄下，啟動 redis，會自動載入資料
```

持久化(AOF)

使用log方式存放，只記錄寫的指令，不紀錄讀

RDB & AOF 同時開啟，只會載入AOF資料

```
# 開啟AOF模式
# 預設 no
appendonly yes
# 檔案名稱
appendfilename "appendonly.aof"
# 目錄名稱(上層預設為啟動目錄)
appenddirname "appendonlydir"
# aof 同步頻率
# appendfsync always|everysec|no
# always:  即時同步(每次寫入同步)
# everysec：每秒同步
# no: 同步時機交給作業系統
# http://antirez.com/post/redis-persistence-demystified.html
appendfsync everysec

# no-appendfsync-on-rewrite 預設no
# 重寫壓縮指令 將多筆設定令重寫成一筆指令
# set A1 1
# set B1 3
# set A1 1 B1 3 => 壓縮重寫
no-appendfsync-on-rewrite yes
# 觸發比例
auto-aof-rewrite-percentage 100
# 觸發檔案大小
auto-aof-rewrite-min-size 64mb

# 如何恢復 ：
# 關閉 redis 將原先appendonly.aof 放回執行目錄下，啟動 redis，會自動載入資料

# aof 檔案損壞
# /usr/local/bin/redis-check-aof --fix appendonly.aof
# 備份該檔
# 重啟服務
```

RDB & AOF 使用時機

- **官方推薦 都開**
- **資料不嚴謹，可單獨使用RDB**
- **不建議單獨使用AOF，可能有BUG**
- Redis 只做cache 用，可都不開

---

sentinel config

```
sentinel monitor mymaster 192.168.1.1 6379 2
sentinel auth-pass mymaster 12345678
```

https://hsiehjenhsuan.medium.com/spring-boot-%E4%BD%BF%E7%94%A8-lettuce-%E8%A8%AD%E5%AE%9A%E5%A4%9A%E5%80%8B-redis-%E9%80%A3%E7%B7%9A-55307dc6a480

【Redis】安装

https://redis.io/docs/install/install-redis/install-redis-from-source/

```
# download
wget https://download.redis.io/redis-stable.tar.gz

tar -xzvf redis-stable.tar.gz
cd redis-stable
make

sudo make install

# run server
redis-server
```

【Redis相關】RedisSDK

```
import org.apache.commons.beanutils.BeanUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.dao.DataAccessException;
import org.springframework.data.redis.connection.RedisConnection;
import org.springframework.data.redis.core.RedisCallback;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.util.CollectionUtils;

import java.io.Serializable;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.TimeUnit;

/***
 *  @date 2018/9/25 15:07
 * @description:
 */
public class RedisSDK {

    /**
     * 日誌
     */
    public static final Logger LOG = LoggerFactory.getLogger(RedisSDK.class);

    private RedisTemplate<String, Object> redisTemplate;

    private static final String TEMP_STR1 = ",value:";

    public void setRedisTemplate(RedisTemplate<String, Object> redisTemplate) {
        this.redisTemplate = redisTemplate;
    }

    //==============================common============================

    /**
     * 指定緩存失效時間
     *
     * @param key  鍵
     * @param time 時間(秒)
     * @return
     */
    public boolean expire(String key, long time) {
        try {
            if (time > 0) {
                redisTemplate.expire(key, time, TimeUnit.SECONDS);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 根據key 獲取過期時間
     *
     * @param key 鍵 不能為null
     * @return 時間(秒) 返回0代表為永久有效
     */
    public long getExpire(String key) {
        return redisTemplate.getExpire(key, TimeUnit.SECONDS);
    }
```

```
/**
 * 判斷key是否存在
 *
 * @param key 鍵
 * @return true 存在 false不存在
 */
public boolean hasKey(String key) {
    try {
        return redisTemplate.hasKey(key);
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * 刪除緩存
 *
 * @param key 可以傳一個值 或多個
 */
@SuppressWarnings("unchecked")
public void del(String... key) {
    if (key != null &amp;amp;&amp;amp; key.length > 0) {
        if (key.length == 1) {
            redisTemplate.delete(key[0]);
        } else {
            redisTemplate.delete(CollectionUtils.arrayToList(key));
        }
    }
}

//============================操作String=============================

/**
 * 普通緩存獲取
 *
 * @param key 鍵
 * @return 值
 */
public Object get(String key) {
    return key == null ? null : redisTemplate.opsForValue().get(key);
}

public String getString(String key) {
    Object result = get(key);
    return result == null ? null : result.toString();
}


/**
 * 普通緩存放入
 *
 * @param key   鍵
 * @param value 值
 * @return true成功 false失敗
 */
public boolean set(String key, Object value) {
    try {
        redisTemplate.opsForValue().set(key, value);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }

}

/**
```

```
     * 判斷是否存在
     * @param key
     * @return
     */
    public boolean exists(String key) {
        try {
            return redisTemplate.execute(new RedisCallback<Boolean>() {
                @Override
                public Boolean doInRedis(RedisConnection redisConnection) throws DataAccessException {
                    return redisConnection.exists(key.getBytes());
                }
            });
        } catch (Exception e) {
            LOG.error("", e);
        }
        return false;
    }

    /**
     * 普通緩存放入並設置時間
     *
     * @param key   鍵
     * @param value 值
     * @param time  時間(秒) time要大於0 如果time小於等於0 將設置無限期
     * @return true成功 false 失敗
     */
    public boolean set(String key, Object value, long time) {
        try {
            if (time > 0) {
                redisTemplate.opsForValue().set(key, value, time, TimeUnit.SECONDS);
            } else {
                set(key, value);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 遞增
     *
     * @param key 鍵
     * @param by  要增加幾(大於0)
     * @return
     */
    public long incr(String key, long by) {
        if (by < 0) {
            throw new RuntimeException("遞增因子必須大於0");
        }
        return redisTemplate.opsForValue().increment(key, by);
    }

    /**
     * 遞減
     *
     * @param key 鍵
     * @param by  要減少幾(小於0)
     * @return
     */
    public long decr(String key, long by) {
        if (by < 0) {
            throw new RuntimeException("遞減因子必須大於0");
        }
        return redisTemplate.opsForValue().increment(key, -by);
    }
```

```
//================================操作Map=================================

/**
 * HashGet
 *
 * @param key  鍵 不能為null
 * @param item 項 不能為null
 * @return 值
 */
public Object hget(String key, String item) {
    return redisTemplate.opsForHash().get(key, item);
}

/**
 * 獲取hashKey對應的所有鍵值
 *
 * @param key 鍵
 * @return 對應的多個鍵值
 */
public Map<Object, Object> hmget(String key) {
    return redisTemplate.opsForHash().entries(key);
}

/**
 * HashSet
 *
 * @param key 鍵
 * @param map 對應多個鍵值
 * @return true 成功 false 失敗
 */
public boolean hmset(String key, Map<String, Object> map) {
    try {
        redisTemplate.opsForHash().putAll(key, map);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * HashSet 並設置時間
 *
 * @param key  鍵
 * @param map  對應多個鍵值
 * @param time 時間(秒)
 * @return true成功 false失敗
 */
public boolean hmset(String key, Map<String, Object> map, long time) {
    try {
        redisTemplate.opsForHash().putAll(key, map);
        if (time > 0) {
            expire(key, time);
        }
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * 向一張hash表中放入數據,如果不存在將創建
 *
 * @param key   鍵
 * @param item  項
 * @param value 值
 * @return true 成功 false失敗
 */
```

```
public boolean hset(String key, String item, Object value) {
    try {
        redisTemplate.opsForHash().put(key, item, value);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * 向一張hash表中放入數據,如果不存在將創建
 *
 * @param key   鍵
 * @param item  項
 * @param value 值
 * @param time  時間(秒)  注意:如果已存在的hash表有時間,這裡將會替換原有的時間
 * @return true 成功 false失敗
 */
public boolean hset(String key, String item, Object value, long time) {
    try {
        redisTemplate.opsForHash().put(key, item, value);
        if (time > 0) {
            expire(key, time);
        }
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

/**
 * 刪除hash表中的值
 *
 * @param key  鍵 不能為null
 * @param item 項 可以使多個 不能為null
 */
public void hdel(String key, Object... item) {
    redisTemplate.opsForHash().delete(key, item);
}

/**
 * 判斷hash表中是否有該項的值
 *
 * @param key  鍵 不能為null
 * @param item 項 不能為null
 * @return true 存在 false不存在
 */
public boolean hHasKey(String key, String item) {
    return redisTemplate.opsForHash().hasKey(key, item);
}

/**
 * hash遞增 如果不存在,就會創建一個 並把新增後的值返回
 *
 * @param key  鍵
 * @param item 項
 * @param by   要增加幾(大於0)
 * @return
 */
public double hincr(String key, String item, double by) {
    return redisTemplate.opsForHash().increment(key, item, by);
}

/**
 * hash遞減
 *
```

```
     * @param key  鍵
     * @param item 項
     * @param by   要減少記(小於0)
     * @return
     */
    public double hdecr(String key, String item, double by) {
        return redisTemplate.opsForHash().increment(key, item, -by);
    }


    /**
     * @param key
     * @return
     */
    public <T> T hgetAll(String key, Class<T> clazz) {
        try {
            Map<Object, Object> values = hmget(key);
            LOG.info("redis獲取值,key:" + key + TEMP_STR1 + JSONObject.toJSONString(values));
            String json = JSONObject.toJSONString(values);
            Map<String, String> all = (Map) JSONObject.parse(json);
            T t = clazz.newInstance();
            BeanUtils.populate(t, all);
            return t;
        } catch (Exception e) {
            LOG.info("redis獲取值,key:" + key + "失敗");
        }
        return null;
    }

    //=============================操作set=============================

    /**
     * 根據key獲取Set中的所有值
     *
     * @param key 鍵
     * @return
     */
    public Set<Object> sGet(String key) {
        try {
            return redisTemplate.opsForSet().members(key);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * 根據value從一個set中查詢,是否存在
     *
     * @param key   鍵
     * @param value 值
     * @return true 存在 false不存在
     */
    public boolean sHasKey(String key, Object value) {
        try {
            return redisTemplate.opsForSet().isMember(key, value);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 將數據放入set緩存
     *
     * @param key    鍵
     * @param values 值 可以是多個
     * @return 成功個數
     */
```

```
public long sSet(String key, Object... values) {
    try {
        return redisTemplate.opsForSet().add(key, values);
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

/**
 * 將set數據放入緩存
 *
 * @param key    鍵
 * @param time   時間(秒)
 * @param values 值 可以是多個
 * @return 成功個數
 */
public long sSetAndTime(String key, long time, Object... values) {
    try {
        Long count = redisTemplate.opsForSet().add(key, values);
        if (time > 0) {
            expire(key, time);
        }
        return count == null ? 0L : count;
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

/**
 * 獲取set緩存的長度
 *
 * @param key 鍵
 * @return
 */
public long sGetSetSize(String key) {
    try {
        Long size = redisTemplate.opsForSet().size(key);
        return size == null ? 0L : size;
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}

/**
 * 移除值為value的
 *
 * @param key    鍵
 * @param values 值 可以是多個
 * @return 移除的個數
 */
public long setRemove(String key, Object... values) {
    try {
        Long count = redisTemplate.opsForSet().remove(key, values);
        return count == null ? 0L : count;
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
}


//================================操作list=================================

/**
 * 獲取list緩存的內容
```

```
     *
     * @param key   鍵
     * @param start 開始
     * @param end   結束  0 到 -1代表所有值
     * @return
     */
    public List<Object> lGet(String key, long start, long end) {
        try {
            return redisTemplate.opsForList().range(key, start, end);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * 獲取list緩存的長度
     *
     * @param key 鍵
     * @return
     */
    public long lGetListSize(String key) {
        try {
            return redisTemplate.opsForList().size(key);
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * 通過索引 獲取list中的值
     *
     * @param key   鍵
     * @param index 索引  index>=0時, 0 表頭,1 第二個元素,依次類推;index<0時,-1,表尾,-2倒數第二個元素,依次類推
     * @return
     */
    public Object lGetIndex(String key, long index) {
        try {
            return redisTemplate.opsForList().index(key, index);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * 將list放入緩存
     *
     * @param key   鍵
     * @param value 值
     * @return
     */
    public boolean lSet(String key, Object value) {
        try {
            redisTemplate.opsForList().rightPush(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 將list放入緩存
     *
     * @param key   鍵
     * @param value 值
     * @param time  時間(秒)
```

```
     * @return
     */
    public boolean lSet(String key, Object value, long time) {
        try {
            redisTemplate.opsForList().rightPush(key, value);
            if (time > 0) {
                expire(key, time);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 將list放入緩存
     *
     * @param key   鍵
     * @param value 值
     * @return
     */
    public boolean lSet(String key, List<Object> value) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 將list放入緩存
     *
     * @param key   鍵
     * @param value 值
     * @param time  時間(秒)
     * @return
     */
    public boolean lSet(String key, List<Object> value, long time) {
        try {
            redisTemplate.opsForList().rightPushAll(key, value);
            if (time > 0) {
                expire(key, time);
            }
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * 根據索引修改list中的某條數據
     *
     * @param key   鍵
     * @param index 索引
     * @param value 值
     * @return
     */
    public boolean lUpdateIndex(String key, long index, Object value) {
        try {
            redisTemplate.opsForList().set(key, index, value);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
```

```
        }
    }

    /**
     * 移除N個值為value
     *
     * @param key   鍵
     * @param count 移除多少個
     * @param value 值
     * @return 移除的個數
     */
    public long lRemove(String key, long count, Object value) {
        try {
            Long remove = redisTemplate.opsForList().remove(key, count, value);
            return remove;
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * 發佈消息
     *
     * @param channel
     * @param message
     */
    public void sendMessage(String channel, Serializable message) {
        redisTemplate.convertAndSend(channel, message);
    }


}
```

DB 欄位命名與比較

資料庫欄位命名的選擇對於系統的可讀性、一致性和維護性有著重要的影響。不同的資料庫可能有不同的命名慣例，以下是常見的命名方式及其在各種資料庫中的應用。

以下是常見命名方式與對應資料庫的整理表格：

命名方式	命名特徵	應用資料庫
駝峰式命名法	單字首字母小寫，後續單字首字母大寫， 如：`firstName`	**MongoDB**、**Cassandra**：與程式碼結合較緊密的 NoSQL 資料庫常使用
蛇形命名法	單字之間用底線 `_` 分隔，全小寫，如：`first_name`	**PostgreSQL**、**MySQL**、**Oracle**、**SQLite**：傳統 SQL 資料庫中最常見
帕斯卡命名法	每個單字首字母大寫，沒有分隔符號，如：`FirstName`	**Microsoft SQL Server**、**Entity Framework**：與程式碼結合時偶爾使用
中線命名法	單字之間用連字符 `-` 分隔，全小寫，如：`first-name`	**Elasticsearch**：主要用於索引和類型，但不常用於欄位
全小寫命名法	全部字母小寫，無分隔符號，如：`firstname`	**MySQL**、**SQLite**：為了避免大小寫區分問題，常用於輕量或無區分大小寫的系統
前綴或後綴命名法	欄位名稱加上描述性前綴或後綴， 如：`user_id`、`created_at`	**PostgreSQL**、**MySQL**、**SQL Server**、**MongoDB**：常用於表示主鍵、外鍵或時間戳等特定欄位

此表格總結了各種命名方式的特徵和常用的資料庫，選擇命名方式時，應根據資料庫特性和開發需求來做決定，並保持專案中的一致性。

常見欄位命名方式及應用

1. **駝峰式命名法（Camel Case）**

- **命名法**：單字首字母小寫，接下來每個單字的首字母大寫。
- **範例**：`firstName`、`lastName`、`orderDate`
- **應用資料庫**：
  - **MongoDB**：作為 NoSQL 資料庫，MongoDB 使用 JSON/BSON 結構，開發者常使用駝峰式命名與程式碼一致。
  - **Cassandra**：由於 Cassandra 支援靈活的資料模式，駝峰式命名法也被頻繁採用，尤其在與程式碼整合時。

2. **蛇形命名法（Snake Case）**

- **命名法**：單字之間用底線 `_` 分隔，所有字母小寫。
- **範例**：`first_name`、`last_name`、`order_date`
- **應用資料庫**：
  - **PostgreSQL**：最常見的命名方式，與 SQL 語法兼容性好，避免大小寫問題。
  - **MySQL**：由於 MySQL 預設不區分大小寫，蛇形命名法常被使用以保持命名的一致性和可讀性。
  - **Oracle**：儘管 Oracle 支援大小寫區分，蛇形命名法依然流行。
  - **SQLite**：輕量級資料庫中也常見此命名方式，方便管理。

3. **帕斯卡命名法（Pascal Case）**

- **命名法**：每個單字首字母大寫，沒有分隔符號。
- **範例**：`FirstName`、`LastName`、`OrderDate`
- **應用資料庫**：
  - **Microsoft SQL Server**：部分開發者，特別是使用 Microsoft 技術堆疊的，會選擇帕斯卡命名法以與程式命名風格保持一致。
  - **Entity Framework**：當與 SQL Server 或其他資料庫結合時，帕斯卡命名法常用於映射資料庫欄位和程式屬性名稱。

4. **中線命名法（Kebab Case）**

- **命名法**：單字之間用連字符 `-` 分隔，所有字母小寫。
- **範例**：`first-name`、`last-name`、`order-date`
- **應用資料庫**：
  - **Elasticsearch**：儘管大部分資料庫不支援欄位中使用連字符，中線命名偶爾會出現在 Elasticsearch 的索引或類型中，但在欄位名稱中較少見。

5. **全小寫命名法**

- **命名法**：所有字母均小寫，無任何分隔符號。
- **範例**：`firstname`、`lastname`、`orderdate`
- **應用資料庫**：
  - **MySQL**：由於 MySQL 不區分大小寫，許多開發者會選擇全小寫命名來避免混淆。
  - **SQLite**：小型應用中，開發者常使用全小寫命名來簡化欄位管理。

6. **前綴或後綴命名法**

- **命名法**：在欄位名稱前或後添加描述性前綴或後綴。
- **範例**：`user_id`、`created_at`
- **應用資料庫**：
  - **PostgreSQL**、**MySQL**、**SQL Server**：此命名法常見於資料表中，特別是主鍵、外鍵或時間戳等欄位，如 `id`、`created_at`。
  - **MongoDB**：在 NoSQL 環境中，這種命名法也被用來標記欄位的特定用途，便於區分不同類型的資料。

小結

- **SQL 資料庫（PostgreSQL、MySQL、SQL Server、Oracle）**：推薦使用 **蛇形命名法**，這是一種最常見的標準化命名方式，能在 SQL 查詢中避免大小寫問題，並提高可讀性。
- **NoSQL 資料庫（MongoDB、Cassandra、Elasticsearch）**：**駝峰式命名法** 和 **全小寫命名法** 較為常見，這些資料庫的靈活數據模型和與 JSON 的緊密結合，使得駝峰式命名更符合程式開發習慣。

選擇合適的命名方式應根據具體資料庫、開發團隊的需求和維護便利性來決定，並保持整個專案中的一致性。