NoSQL for Cache 資料庫設計建議與開發準則

適用聲明

- 本文件（以下簡稱 開發準則）適用於 104corp DBA 團隊所管理的 NoSQL 資料庫用於暫存或快取場景。
- 適用範疇包含(且不限於)：
  - Redis Standalone 7.0+
  - AWS ElastiCache for Redis OSS
  - 其他具快取特性的 NoSQL 解決方案
    - 高頻讀寫暫存資料（Session / Cache）
    - 輕量級消息佇列與 Pub/Sub 系統
  - 請持續參考 104 Technology Standards (https://104cloud.sharepoint.com/:x:/s/EA/EQdML4ub_DZDpkICjMWi1kYBjWWWzuMkrSiu746lh55r8A?e=CeMAEv) & 104 Technology Radar (https://radar.thoughtworks.com/?documentId=https%3A%2F%2Fdocs.google.com%2Fspreadsheets%2Fd%2F1IMGTRbV_V_c9QErkgp4JK4iqZiav_WS5COByLtISA90%2F)
- 註記：
  - 本文件強調「短期暫存 / 非關鍵持久化數據」，為減緩 RDBMS 負載之輔助層。
  - 本文件為指引性建議，提供之建議 不具溯及力 ，適用於新開發、重構與需求調整情境。
  - 產品端可依據業務需求與 DBA 討論適度調整；經討論後之調整建議記錄於產品技術文件與架構設計文件。

準則調整與維護辦法

- 本文件由 DBA 團隊負責維護與版本控管。
- 特殊需求應提前與 DBA / 架構團隊評估，並納入架構設計文件。
- 每季檢視及更新快取策略，並應搭配 DBA Quarterly Routine Tasks 季度週期作業 (https://104corp.atlassian.net/browse/ITDBA-3379) 檢討是否存在「快取即資料庫」的錯誤使用並提出修正。

溝通管道：若產品端有相關建議變更提案，請直接聯繫負責 DBA（E-mail 或 Teams），DBA 會在季度例會

修訂歷史

版本	日期	修訂者	審核者	變更摘要 & 影響章節	關聯 Ticket
1.0	2025/10/07	DBA Team	DBA Team	初版發布	ITDBA-3379 (https://104corp.atlassian.net/browse/ITDB 3379)

標準化設計建議與開發準則

說明：本文件針對 NoSQL for Cache 操作設計提供標準化建議、正確 / 不正確範例與對產品應用的影響分析，方便開發/運維落地執行與稽核。

Global

- [SHOULD] 命名與 Key 設計
  - 建議：table/key 使用小寫、底線或冒號分隔；禁止特殊字元（空格、換行、引號）；Key 長度建議 ≤64 字元；高頻 Key 加前綴便於辨識用（cache:, session:, temp:）。

    不正確範例："user 1000"、user\nid、very_long_namespace_name_exceeding_sixt
    正確範例：user:1000、cache:session:550e8400-e29b-41d4-a716-446655440000

- [MUST] 權限最小化
  - 建議：Application 權限必須嚴格限制，禁止執行 Command categories: -@admin (https://redis.io/docs/latest/operate/oss_and_stack/management/security/acl/#command-categories), -@dangerous (https://redis.io/docs/latest/operate/oss_and_stack/management/security/acl/#command-categories)。線上營運環境的應用帳號不得有這兩類命令權限。

    正確範例: 使用 SCAN 迭代（redis-cli --scan --pattern "prefix:*" 或 SCAN cur
    不正確範例: SHUTDOWN ; KEYS * ; FLUSHALL、CONFIG（全域/危險操作）

- [SHOULD] Cluster / Sharding 考量
  - 建議：在 Redis Cluster 環境設計 key 前綴與 hash tag（{}）確保原子操作跨 slot；避免跨 slot multi-key 操作。

範例: 要 multi-key 原子操作時使用 hash tag: user:{1000}:cart 與 user:{1000}:

- [SHOULD] Client library 與連線管理
  - 建議：使用穩定 client，啟用 connection pool、short timeouts、合理 retry 與 backoff；避免大量短連線。

增

- [MUST] 資料驗證與正規化
  - 建議：Application 寫入資料前做嚴格驗證。

    不正確範例: 直接把 client 傳來的 JSON 存成 text 未做 schema 驗證

- [MUST] 資料型態與序列化
  - 建議：選用適當 Redis 資料型別（String, Hash, List, Set, Sorted Set）以減少序列化成本。

    不正確範例: 把大量欄位序列化成單一超巨 JSON 儲存在 string

- [SHOULD] Idempotency（冪等性檢查）
  - 建議：所有建立/變更型 API 必須支援 idempotency-key；以 Redis 回應快取緩存，以 RDBMS 的 UNIQUE constraint 作最終安全機制。

    不正確範例: API 多次重試造成重複訂單 insert（無唯一約束）

- [MUST] 批次寫入與 pipeline（避免高頻單鍵阻塞）
  - 建議：大量寫入應使用 pipeline 或 redis-cli --pipe 以降低 RTT；避免單一 key 的高頻更新導致熱點，應採分批（batch size）、節流（rate limit）與鍵分散策略。

    正確範例: 用 redis-cli --pipe 批次寫入，非原子，需分批與節流

- [MUST] TTL 與快取策略（Cache keys 必設 TTL）
  - 建議：所有臨時/快取類鍵必須設 TTL；若無 TTL，需有自動回收/清理機制。避免永久無 TTL，減少記憶體累積與漫長記憶體回收。

    不正確範例: SET cache:user:1000 "..." （無 EX）

刪

- [MUST] 禁止於 Production 使用破壞性/全域命令
  - 建議：Production 環境不得由應用帳號執行 FLUSHALL、FLUSHDB 刪除無用資料。
- [SHOULD] 使用非阻塞 (Non-blocking) 刪除（UNLINK 優先於 DEL）
  - 建議：對大物件或大量 key 刪除，優先使用 UNLINK（非同步釋放）以避免 Redis 主執行緒阻塞；僅在確定小量 key 時使用 DEL。

```
redis> SET key1 "Hello"
"OK"
redis> SET key2 "World"
"OK"
redis> UNLINK key1 key2 key3
(integer) 2
```

- [SHOULD] 使用軟刪除或 tombstone 防止快取穿透/回填 race
  - 建議：對重要資料採 soft-delete（DB 標記 + cache 設 tombstone short TTL 或 version）而非直接刪除，以降低查無資料時重建壓力與穿透。

```
範例: SET cache:user:1000:tombstone "deleted" EX 300 ; 或將 value 加上 ver
```

- [SHOULD] Cluster 情境下避免跨 slot multi-key 刪除
  - 建議：在 Redis Cluster，multi-key 操作必須在同 slot。刪除多個相關 key 時使用 hash tag（{}）確保同 slot，或逐 slot 分批處理。

```
範例: user:{1000}:profile user:{1000}:cart 可以同 slot 一起刪 ; 否則用 SCAN
```

改

- [SHOULD] 使用 Hash 或細粒度 key 儲存可部分更新的欄位
  - 建議：避免把頻繁變動欄位全部序列化成大 JSON；用 Hash 存欄位可原子更新某一欄且節省頻寬。

```
範例：
redis-cli HSET user:1000 name "alice"
redis-cli HINCRBY user:1000 counter 1
```

- [SHOULD] 考量記憶體與存取效率

- 建議：Hash 在欄位數少且多樣時節省空間，但當欄位非常多或 value 很大時，可能比多 key 更不適。

  範例：小值多欄用 `Hash`；大值用多 `key`

查

- [MUST] 避免 KEYS 一次性大操作（使用 SCAN + 分批 pipeline）
  - 建議：禁止在生產使用 KEYS 等全域/破壞性指令；請用 SCAN 並配合 COUNT 與 pipeline 分批處理。

    正確範例: 使用 `SCAN cursor MATCH "prefix:*" COUNT 1000`，將結果分批處理於 `pipe`

- [SHOULD] Cluster 情境：資料批次作業建議。
  - 建議：對每個 master 單獨執行 SCAN，分批以 COUNT + pipeline/UNLINK 處理，勿用 KEYS 或掃 replica。

    `KEYS` 為 `O(N)` 且會阻塞主執行緒；掃 `Replica` 可能不一致且增加 `Replication/IO` 負載。

- [SHOULD] 使用資料類型特定掃描（HSCAN/SSCAN/ZSCAN）
  - 建議：若目標為 hash、set、zset 的成員或 field，優先用 HSCAN/SSCAN/ZSCAN，避免用 SCAN + GET 反覆讀取整個 key。
  - 對於 Large Value 成員，建議用 COUNT 值並以 pipeline 或批次命令分批執行。

```
# Hash 字段掃描
redis-cli HSCAN myhash 0 MATCH 'field:*' COUNT 200
# Set 成員掃描
redis-cli SSCAN myset 0 MATCH 'member:*' COUNT 200
# ZSet 成員掃描
redis-cli ZSCAN myzset 0 MATCH 'member:*' COUNT 200
```