第一章

簡介

MongoDB 是個強大有彈性且可擴充的資料庫。它結合了水平擴展（scale out），以及如次要索引、範圍查詢、排序、聚集以及地理資訊索引等特色。本章包含造就 MongoDB 的主要設計決策。

容易使用

MongoDB 是一個**文件導向**（*document-oriented*）資料庫，它並不是關聯式資料庫。不使用關聯式資料庫的最主要原因，是因為這樣可以使得水平擴展更為容易，但當然還有其他的優點。

文件導向資料庫的基礎概念是將「列」（row）的概念用更彈性的模型：「文件」（document）來表達。為了要允許能夠內嵌文件以及陣列，文件導向的作法能夠用單一筆記錄來表示複雜的階級關係。這樣可讓使用物件導向程式語言的開發者，用更自然的方法規劃他們的資料。

MongoDB 也沒有預先定義的綱要（schema）：文件的鍵與值並沒有固定的型態或大小。因為沒有固定的綱要，所以當有需要新增或移除欄位時就變得更簡單。通常來說，這樣會讓開發變得更快速。也能更容易的作任何實驗。開發者能夠嘗試各種不同的資料模型，然後選擇最好的一種繼續開發。

碁峰 www.gotop.com.tw

設計為可擴充

應用程式的資料集合大小會以無法想像的速度成長。因為可使用頻寬的增加和便宜的儲存裝置，使得就算是小規模的應用程式也需要儲存以往許多資料庫無法處理的資料量。上兆位元組的資料，前所未有的資料量，現在都已經很普遍了。

因為開發者要儲存的資料量持續成長，會面臨到困難的抉擇：要如何擴充資料庫？擴充資料庫有兩種方式：垂直擴充（scale up），也就是換成更強大的機器；水平擴充（scale out），也就是在多台機器上放置分割過後的資料。垂直擴充通常較簡單達成，但它有一些缺點：大型機器通常價格昂貴，並且最終會達到其物理極限，以至於花再多的錢也無法再購買到更高階的機器。另外一種方法就是水平擴充：增加儲存空間，或是增加讀取和寫入動作的吞吐量，然後購買額外的伺服器，並且將它們加入到叢集中。這樣既便宜又擁有更好的擴充性，然而要管理數千台機器一定比管理一台機器要困難。

MongoDB 就是設計用來水平擴充的。它的文件導向資料模型能夠讓資料簡單的分割到多台伺服器上。MongoDB 自動會負責叢集中的資料平衡以及負載平衡，自動重新分配文件，並且將讀取與寫入導向對的機器，如圖 1-1 所示。

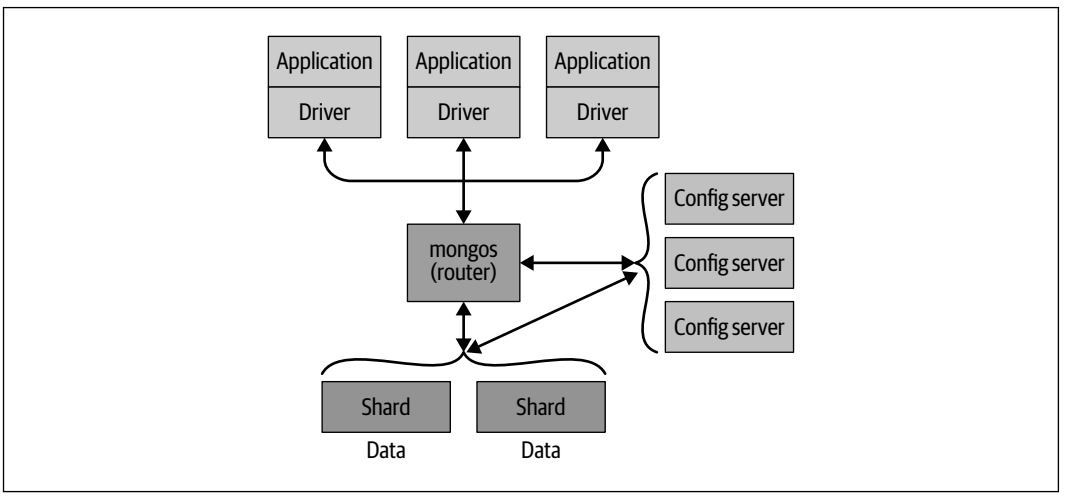

圖 1-1　在多個伺服器上使用分片來將 MongoDB 水平擴充

應用程式會知道 MongoDB 叢集的拓撲，或者它是否實際上是一個叢集，而不是在資料庫連線另一端的單一節點。這讓開發者能夠專注於應用程式的開發，而不是想著要如何擴充。相同地，舉例來說，若現存的部署因為要能夠支援更多的負載而需要被擴充，導致要變更拓撲，仍然能夠維持相同的應用程式邏輯。

碁峰 www.gotop.com.tw

第三章

建立、更新以及刪除文件

本章涵蓋了將資料移入或移出資料庫的基本知識，包含以下的內容：

- 在一個集合中新增文件
- 從一個集合中移除文件
- 更新現存的文件
- 在安全性層級與速度之間為這些動作正確地作出抉擇

插入文件

插入是將資料新增至 MongoDB 的基本方法。要將一個文件插入至一個集合中，使用集合的 `insertOne` 方法：

```
> db.movies.insertOne({"title" : "Stand by Me"})
```

這麼做會新增 `"_id"` 鍵至文件中（假如並未存在文件中），並且儲存至 MongoDB 中。

insertMany

當你需要插入多個文件到一個集合內時，可以使用 `insertMany`。這個方法讓你可以將文件的陣列傳入資料庫中。這是極度有效率的，因為你的程式不會讓每個要插入的文件都在資料庫之間來回傳送，而是一次大批的插入。

碁峰 www.gotop.com.tw

在命令列界面中，你能夠做如下的嘗試：

```
> db.movies.drop()
true
> db.movies.insertMany([{"title" : "Ghostbusters"},
...                       {"title" : "E.T."},
...                       {"title" : "Blade Runner"}]);
{
      "acknowledged" : true,
       "insertedIds" : [
           ObjectId("572630ba11722fac4b6b4996"),
           ObjectId("572630ba11722fac4b6b4997"),
           ObjectId("572630ba11722fac4b6b4998")
      ]
}
> db.movies.find()
{ "_id" : ObjectId("572630ba11722fac4b6b4996"), "title" : "Ghostbusters" }
{ "_id" : ObjectId("572630ba11722fac4b6b4997"), "title" : "E.T." }
{ "_id" : ObjectId("572630ba11722fac4b6b4998"), "title" : "Blade Runner" }
```

一次傳入數十個、數百個或甚至數千個文件，可以大大地加快插入的速度。

當你要插入多個文件到單一個集合時，`insertMany` 就非常有用。若你只是要匯入原始資料（例如從一個資料來源或是 MySQL），有像 *mongoimport* 的命令列工具可以取代批次插入用來匯入資料。另一方面來說，通常在存入 MongoDB 之前會作資料的轉換（將日期資訊轉換為日期型態或是加上自己定義的 `"_id"`），所以 `insertMany` 也可以被用來匯入資料。

目前版本的 MongoDB 並不允許大於 48MB 的訊息，所以在單一的批次插入中是有大小限制的。若你嘗試插入超過 48MB 的內容，許多驅動程式會將內容分割成多個 48MB 大小的批次插入。詳情請見你的驅動程式文件。

當使用 `insertMany` 來執行大量插入時，若陣列中有個文件產生了某個錯誤，結果會因為你選的動作是有序的還是無序的而有所不同。`insertMany` 中的第二個參數可以指定一個操作選項的文件。在選項文件中將 `"ordered"` 鍵設為 `true` 能夠確保文件被插入的順序會跟傳入的順序是相同的。若設為 `"false"` 則 MongoDB 可能會因為要提升效率而重組插入順序。若沒有任何的排序被指定，則預設會是有序地插入。對有序插入來說，傳到 `insertMany` 的陣列就定義了插入的順序。若有個文件產生了插入錯誤，那麼在該文件之後的任何文件都不會被插入。若是無序插入，MongoDB 則會嘗試插入所有的文件，不管有沒有文件發生錯誤。

碁峰 www.gotop.com.tw

在這個範例中，因為預設是有序插入，只有前兩個文件會被插入。第三個文件會產生錯誤，因為你不能插入兩個擁有相同 "_id" 的文件：

```
> db.movies.insertMany([
    ... {"_id" : 0, "title" : "Top Gun"},
    ... {"_id" : 1, "title" : "Back to the Future"},
    ... {"_id" : 1, "title" : "Gremlins"},
    ... {"_id" : 2, "title" : "Aliens"}])
2019-04-22T12:27:57.278-0400 E QUERY [js] BulkWriteError: write
error at item 2 in bulk operation :
BulkWriteError({
    "writeErrors" : [
        {
            "index" : 2,
            "code" : 11000,
            "errmsg" : "E11000 duplicate key error collection:
            test.movies index: _id_ dup key: { _id: 1.0 }",
            "op" : {
                "_id" : 1,
                "title" : "Gremlins"
            }
          }
        ],
"writeConcernErrors" : [ ],
"nInserted" : 2,
"nUpserted" : 0,
"nMatched" : 0,
"nModified" : 0,
"nRemoved" : 0,
"upserted" : [ ]
})
BulkWriteError@src/mongo/shell/bulk_api.js:367:48
BulkWriteResult/this.toError@src/mongo/shell/bulk_api.js:332:24
Bulk/this.execute@src/mongo/shell/bulk_api.js:1186:23
DBCollection.prototype.insertMany@src/mongo/shell/crud_api.js:314:5
@(shell):1:1
```

若我們指定為無序插入，在陣列中的第一個、第二個跟第四個文件將會被插入。只有第三個文件插入失敗，因為擁有重複的 "_id"：

```
> db.movies.insertMany([
... {"_id" : 3, "title" : "Sixteen Candles"},
... {"_id" : 4, "title" : "The Terminator"},
... {"_id" : 4, "title" : "The Princess Bride"},
... {"_id" : 5, "title" : "Scarface"}],
... {"ordered" : false})
```


碁峰 www.gotop.com.tw

第四章

查詢

本章會詳細的介紹查詢。主要涵蓋的部分如下：

- 你能夠作範圍查詢、集合查詢、不等式以及更多使用 `$` 字元的條件式。
- 查詢會回傳一個資料庫游標（cursor），在你需要時會延遲地批次回傳文件。
- 有許多子動作可以在游標上執行，包含跳過一定數量的結果、限制回傳的結果數量以及排序結果。

簡介查詢

`find` 方法用來在 MongoDB 中執行查詢。查詢會回傳在集合中文件的子集合，有可能沒有任何文件，也有可能是整個集合文件。`find` 方法的第一個參數決定了哪些文件要被回傳，它是一個指定什麼查詢條件要被執行的文件。

一個空的查詢文件（如 `{}`）會找到集合內的所有文件。若沒有傳入查詢文件至 `find` 中，它預設就會傳入 `{}`。舉例來說，如下：

```
> db.c.find()
```

會回傳在集合 *c* 中的所有文件（並且是批次的回傳這些文件）。

當我們開始在查詢文件中加入鍵值對時，便開始限制我們的搜尋。對於多數的型態，通常都可以正確運作：數字匹配數字、布林值匹配布林值，而字串匹配字串。對簡單型態

碁峰 www.gotop.com.tw

的查詢，只要直接指定想要找的值即可。舉例來說，要找到所有 "age" 值為 27 的文件，只要在查詢文件中加入一個鍵值對：

```
> db.users.find({"age" : 27})
```

若我們想要找到一個字串，如鍵為 "username" 且值為 "joe" 的文件，可以使用下面的鍵值對：

```
> db.users.find({"username" : "joe"})
```

多重條件的查詢，可以藉由將許多的鍵值對加入到查詢文件中達成，它們將會被解釋為「條件 1 AND 條件 2 AND... AND 條件 N」。舉例來說，想要找到所有 "age" 為 27 且 "username" 為 "joe" 的使用者，可以如下的查詢：

```
> db.users.find({"username" : "joe", "age" : 27})
```

指定要回傳的鍵

有時候你並不需要文件中所有的鍵值對都被回傳。若在這種狀況下，你可以在 find（或是 findOne）的第二個參數中，指定你想要回傳的鍵。這麼做可以減少資料傳輸的量、時間以及在客戶端解碼文件所需要的記憶體。

舉例來說，若你有一個使用者集合，但是只對 "username" 以及 "email" 這兩個鍵有興趣，那麼使用下面的查詢就可以得到只有這些鍵的回傳：

```
> db.users.find({}, {"username" : 1, "email" : 1})
{
    "_id" : ObjectId("4ba0f0dfd22aa494fd523620"),
    "username" : "joe",
    "email" : "joe@example.com"
}
```

如你在上面的輸出中所見的，"_id" 鍵預設會被回傳，就算它並沒有被指定也一樣。

你也能夠在第二個參數中指定某些鍵值對不要在查詢結果中出現。舉例來說，你有個擁有許多鍵的文件，而你唯一知道的事情，就是你永遠不會需要 "fatal_weakness" 鍵被回傳：

```
> db.users.find({}, {"fatal_weakness" : 0})
```

這也可以用來防止 "_id" 鍵被回傳：

```
> db.users.find({}, {"username" : 1, "_id" : 0})
{
```


碁峰 www.gotop.com.tw

第五章

索引

本章介紹 MongoDB 的索引。索引可以讓查詢變得更快速。它們是應用程式開發的重要部分，並且對特定種類的查詢來說甚至是必要的。本章將會涵蓋：

- 索引是什麼，以及為什麼你會想要使用它們
- 如何選擇要索引的欄位
- 要如何實際使用，並且要如何評估索引的使用
- 建立以及移除索引的管理細節

如你將會看到的，為你的集合選擇正確的索引對效能的影響是非常大的。

索引簡介

資料庫索引就像是書的索引。資料庫可以走捷徑，只要查看有序的清單就可以參照到內容，如同看書的索引找內容而不用查看整本書。這讓 MongoDB 在查詢數量級的資料時更快速。

不使用索引的查詢稱作**集合掃描**，代表伺服器必須要「查看過整本書」來找到查詢的結果。這個程序基本上就跟要尋找一本沒有索引的書中的某個資訊一樣：你要從第一頁開始，然後開始閱讀整本書。通常來說，你要避免伺服器作集合掃描，因為這個程序對於大的集合來說是非常緩慢的。

讓我們來看一個範例。開始我們先建立一個有一百萬個文件（或是一千萬，或是一億，假如你很有耐心的話）的集合：

碁峰 www.gotop.com.tw

要被搜尋，以及額外的資訊，如是否有排序等。基於這些資訊，系統會辨識出一組索引，當作可能可以滿足查詢的候選索引。

假設有一個查詢到來，五個索引中有三個被辨識為該查詢的候選索引。MongoDB 接著會建立三個查詢計畫，為每一個索引建立一個計畫，然後用三個平行的執行緒執行查詢，每一個都使用不同的索引。在此的目的是要看哪個能夠最快的回傳結果。

視覺上，我們能夠將它想成是個競賽，如圖 5-1。這個概念是，第一個到達目標狀態的查詢計畫就是贏家。但更重要的是，若之後遇到相同樣貌的查詢。它將會被選為要被使用的索引。這些查詢計畫會互相競賽一陣子（圖中為「試驗期間」），結束之後每場比賽的結果將會用來計算整體的贏家計畫為何。

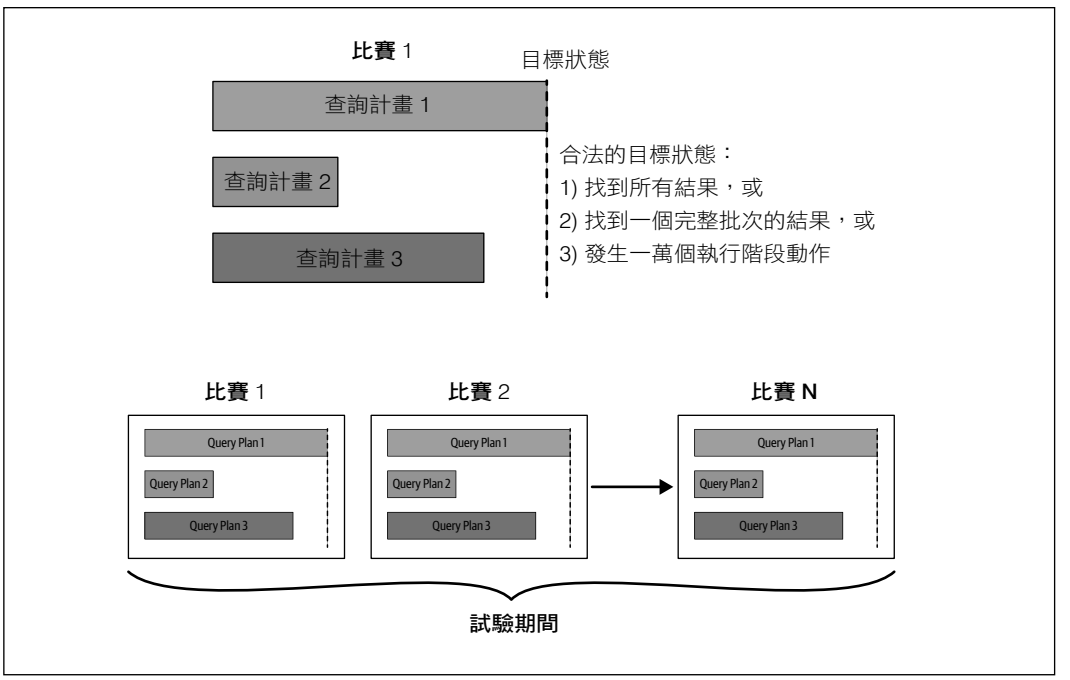

圖 5-1　以競賽視覺化 MongoDB 查詢計畫器如何選擇索引

要贏得比賽，一個查詢的執行緒必須要是第一個排序好回傳所有查詢結果，或是排序好回傳一個試驗數量的結果。排序的部分很重要，讓我們知道在記憶體內執行排序是多麼沒有效率。

碁峰 www.gotop.com.tw

在不同的查詢計畫之間比賽的最大價值是，之後有相同樣貌的查詢時，MongoDB 伺服器就會知道要選擇什麼索引了。伺服器會維護查詢計畫的快取。獲勝的查詢計畫會被存在快取中，未來有該樣貌的查詢就可以直接使用。一段時間後，集合會改變而索引也會跟著改變，最終查詢計畫可能會被移出快取，然後 MongoDB 會再次用可能的查詢計畫，對現在的集合以及索引作實驗，找出最好的計畫。還有一些其他可能讓查詢計劃被移出快取的事件，如重建索引、新增或刪除索引，或是直接將計畫快取清除。最後，重啟 *mongod* 程序也會讓查詢計畫快取被清除。

使用組合索引

在前一節中，我們已經使用了組合索引，也就是包含多於一個鍵的索引。組合索引會比一般的單一鍵索引要複雜一些，但它是非常強大的。本節會更詳細地闡述它。

在此，我們將會用一個範例給你一些概念，讓你要設計組合索引時，知道必須要思考哪些事情。目標就是要讓讀取以及寫入的動作盡可能的有效率，但如同許多的事情一樣，這需要一些事前思考以及一些實驗。

要確定我們能夠得到正確適當的索引，在真實世界的負載下測試索引然後調整，是必要的。然而，當我們在設計索引時，有一些良好慣例是可以遵循的。

首先，我們需要考慮索引的選擇性。對於一個查詢，我們對於能夠提供最少要被掃瞄的紀錄數量的索引感到興趣。因為所有的動作必須要滿足查詢，所以我們需要考量選擇性，有時候要作出取捨。舉例來說，我們會需要考慮排序是如何被處理的。

讓我們來看一個範例。我們會使用一個約有一百萬筆記錄的學生資料集合。資料集合中的文件是如下組成：

```
{
    "_id" : ObjectId("585d817db4743f74e2da067c"),
    "student_id" : 0,
    "scores" : [
    {
        "type" : "exam",
        "score" : 38.05000060199827
    },
    {
        "type" : "quiz",
        "score" : 79.45079445008987
    },
    {
        "type" : "homework",
```


碁峰 www.gotop.com.tw

查詢種類	幾何種類
$nearSphere（GeoJSON 點、2dsphere 索引）	球體
$nearSphere（傳統座標、2d 索引）*	球體
$geoWithin : {$geometry: …}	球體
$geoWithin : {$box: …}	平面
$geoWithin : {$polygon: …}	平面
$geoWithin : {$center: …}	平面
$geoWithin : {$centerSphere: …}	球體
$geoIntersects	球體

* 改用 GeoJSON 點

也要注意的是，2d 索引對於平面幾何跟單純在球體上計算距離（例如，使用 $nearSphere）這兩者都支援。然而，使用球體幾何的查詢搭配 2dsphere 索引會更有效率且準確。

還有要注意的是，$geoNear 運算子是一個聚集運算子。聚集框架會在第七章中被討論。除了 $near 查詢運算子之外，$geoNear 聚集運算子跟 geoNear 這個特別指令讓我們能夠查詢附近的位置。記住，假如集合有使用 MongoDB 的擴充解決方案（見第十五章），也就是集合是使用分片的分散式分佈，$near 查詢運算子就沒辦法用在它上面。

geoNear 指令和 $geoNear 聚集運算子要求，集合中至多只能擁有一個 2dsphere 索引並且至多只能擁有一個 2d 索引，而地理資訊查詢運算子（如 $near 和 $geoWithin）則可以擁有多個地理資訊索引。

geoNear 指令和 $geoNear 聚集運算子對於地理資訊索引會有限制，是因為不論是 geoNear 指令還是 $geoNear 語法，都沒有包含位置欄位。因此，在多個 2d 索引或多個 2dsphere 索引之間選擇索引是很混淆不清的。

地理空間查詢運算子並沒有如此的限制； 這些運算子會使用一個位置欄位，來消除混淆不清的狀態。

扭曲

球體幾何在視覺化到地圖上時看起來會被扭曲，這是因為將三維球體（如地球）投影到平面上自然產生的行為。

碁峰 www.gotop.com.tw

搜尋其他語言

當一個文件被插入（或是索引首次被建立）時，MongoDB 會查看索引的欄位然後為所有字去尾，將其盡量精簡。然而，不同的語言會用不同的方式為詞彙去尾，所以你必須要指定索引或文件的語言為何。text 索引可以指定 "default_language" 選項，該選項預設值為 "english"，但可以被設定為數種語言（參照線上文件（*https://oreil.ly/eUt0Z*）會有最新的列表）。

舉例來說，要建立一個法語的索引，我們可以：

```
> db.users.createIndex({"profil" : "text",
                        "intérêts" : "text"},
                       {"default_language" : "french"})
```

然後法語就會用在文字的去尾，除非有其他設定。你可以以單一文件為單位，藉由描述文件語言的 "language" 欄位，來指定去尾的語言：

```
> db.users.insert({"username" : "swedishChef",
... "profile" : "Bork de bork", language : "swedish"})
```

最高限度集合

在 MongoDB 中，「正常的」集合是動態地被建立以及自動地成長大小以適應額外的資料。MongoDB 也支援另一種不同型態的集合，稱作**最高限度集合**（*capped collection*），它是事先被建立且大小固定的（見圖 6-4）。

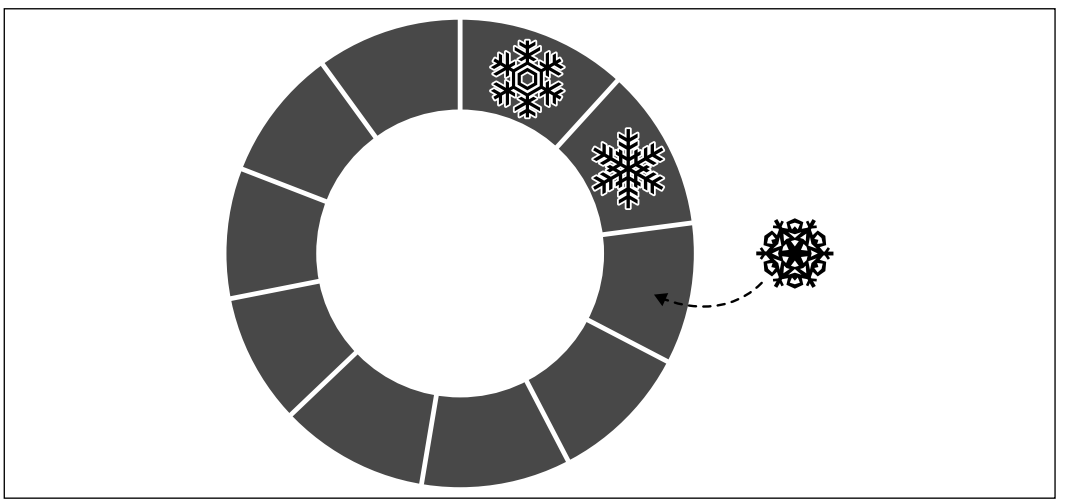

圖 6-4　被插入在佇列最後的新文件

碁峰 www.gotop.com.tw