

ChatGPT ▾

◆ 取得 Plus 版本 ×

↑ 分享

...

C# 多個執行緒存取 同一個 SQLITE 檔案

在 C# 中讓 **多個執行緒 (threads)** 同時存取 **同一個 SQLite 資料庫檔案** 是可以做到的，但需要非常謹慎地設定與使用，否則會出現錯誤，例如：

```
SQLite Error 5: database is locked
```

✖ 一、SQLite 的多執行緒原理

SQLite 是輕量級資料庫，預設是針對單執行緒或單進程優化的。

要讓多執行緒同時操作，需要考慮以下幾層設定：

層級	重點說明
SQLite 引擎層	是否開啟「多執行緒模式」或「序列化模式」
Connection 層	每個執行緒要不要共用同一個連線 (Connection)
Transaction 層	同步機制：同時讀、同時寫會被鎖定
檔案層	SQLite 寫入時是以整個檔案鎖定為單位 (非 row lock)

✓ 二、建議設定方式

◆ 1. 使用 WAL 模式 (Write-Ahead Logging)

這是多執行緒存取同一 SQLite 檔案最重要的一步。

它允許多執行緒同時讀取資料 (只有寫入會鎖) 。

csharp

복사

```
using var connection = new SqliteConnection("Data Source=mydb.db;");
connection.Open();

using var command = connection.CreateCommand();
command.CommandText = "PRAGMA journal_mode=WAL;";
command.ExecuteNonQuery();
```

WAL 模式可同時多讀單寫，是多執行緒場景的首選。

◆ 2. 每個執行緒使用「自己的連線」

不要共用同一個 `SqliteConnection` 。

因為同一個連線物件不是 `thread-safe` 。

csharp

복사

```
Parallel.For(0, 10, i =>
{
    using var conn = new SqliteConnection("Data Source=mydb.db;");
    conn.Open();

    using var cmd = conn.CreateCommand();
```

```
cmd.CommandText = $"INSERT INTO log_table (msg) VALUES ('Thread {i}')";
cmd.ExecuteNonQuery();
});
```

◆ 3. 設定連線字串中的同步與快取策略

csharp

 複製程式碼

```
var connStr = new SqliteConnectionStringBuilder
{
    DataSource = "mydb.db",
    Mode = SqliteOpenMode.ReadWriteCreate,
    Cache = SqliteCacheMode.Shared
}.ToString();
```

`Cache=Shared` 可以讓同一進程內多個連線共享頁面快取，減少鎖定衝突。

◆ 4. 如果仍發生鎖定，可以加上重試機制

SQLite 在寫入衝突時可能丟出 `SQLiteBusyException`。

你可以加入簡單的重試：

csharp

 複製程式碼

```
for (int retry = 0; retry < 3; retry++)
{
```

```
try
{
    using var conn = new SqliteConnection(connStr);
    conn.Open();
    using var cmd = conn.CreateCommand();
    cmd.CommandText = "INSERT INTO log_table (msg) VALUES ('test')";
    cmd.ExecuteNonQuery();
    break; // success
}
catch (SqliteException ex) when (ex.SqliteErrorCode == 5) // database is Locked
{
    Thread.Sleep(100 * (retry + 1));
}
}
```

⚙️ 三、SQLite 執行緒模式 (僅供理解)

可在開啟資料庫時查詢：

sql

복製程式碼

```
PRAGMA threads;
```

SQLite 有三種模式 (通常不用手動切換) :

模式	說明
single-thread	不可多執行緒

模式	說明
multi-thread	同一 connection 不可共用，不同 connection 可同時操作
serialized	完全 thread-safe (但效能略低)

C# 的 `Microsoft.Data.Sqlite` 預設是使用 **multi-thread**，已足夠安全。

🧠 四、整合建議

如果你要多執行緒存取 SQLite，同時確保穩定性與效能：

- WAL 模式
- 每個執行緒各自建立連線
- 使用 `Cache=Shared`
- 適度加上重試機制
- 避免長時間 Transaction 鎖定整張表

是否要我幫你示範一個

👉 「多執行緒同時寫入 SQLite」的完整可執行 C# 範例（含 WAL、重試、平行寫入）？



「多執行緒同時寫入 SQLite」的完整可執行 C# 範例（含 WAL、重試、平行寫入）

太好了 🤝

以下是完整可執行的 **C# 範例**，示範「多執行緒同時寫入同一個 SQLite 檔案」。

具備：

- ✓ WAL 模式
- ✓ 每執行緒各自開連線
- ✓ 寫入重試機制
- ✓ 平行執行 (Parallel.For)



完整範例程式 (.NET 7+ 可執行)

csharp

복사程式碼

```
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Data.Sqlite;

class Program
{
    static void Main()
```

```
{  
    const string dbFile = "multi_thread_test.db";  
  
    // 若檔案不存在就初始化資料庫  
    if (!File.Exists(dbFile))  
    {  
        InitializeDatabase(dbFile);  
    }  
  
    // 啟用 WAL 模式 (支援多執行緒多讀單寫)  
    using (var conn = new SqliteConnection($"Data Source={dbFile}"))  
    {  
        conn.Open();  
        using var cmd = conn.CreateCommand();  
        cmd.CommandText = "PRAGMA journal_mode=WAL;";  
        var mode = cmd.ExecuteScalar();  
        Console.WriteLine($"WAL 模式啟用結果: {mode}");  
    }  
  
    Console.WriteLine("開始平行寫入...");  
  
    var connStr = new SqliteConnectionStringBuilder  
    {  
        DataSource = dbFile,  
        Cache = SqliteCacheMode.Shared,  
        Mode = SqliteOpenMode.ReadWrite  
    }.ToString();  
  
    // 使用 20 個執行緒平行寫入資料
```

```
Parallel.For(0, 20, i =>
{
    for (int j = 0; j < 5; j++)
    {
        WriteWithRetry(connStr, i, j);
    }
});

Console.WriteLine("寫入完成。");

}

static void InitializeDatabase(string dbFile)
{
    using var conn = new SqliteConnection($"Data Source={dbFile}");
    conn.Open();
    using var cmd = conn.CreateCommand();
    cmd.CommandText = @"
        CREATE TABLE log_table (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            thread_id INTEGER,
            msg TEXT,
            created_at TEXT
        );
    ";
    cmd.ExecuteNonQuery();
    conn.Close();

    Console.WriteLine("資料庫已初始化。");
}
```

```
static void WriteWithRetry(string connStr, int threadId, int seq)
{
    const int maxRetry = 5;
    for (int retry = 0; retry < maxRetry; retry++)
    {
        try
        {
            using var conn = new SqliteConnection(connStr);
            conn.Open();
            using var cmd = conn.CreateCommand();

            cmd.CommandText = "INSERT INTO log_table (thread_id, msg, created_at) VALUES (@t,
            cmd.Parameters.AddWithValue("@t", threadId);
            cmd.Parameters.AddWithValue("@m", $"Thread {threadId} - record {seq}");
            cmd.ExecuteNonQuery();

            return; // 成功就跳出
        }
        catch (SqliteException ex) when (ex.SqliteErrorCode == 5) // database is Locked
        {
            int delay = 100 * (retry + 1);
            Console.WriteLine($"[Thread {threadId}] 資料庫被鎖定，重試第 {retry + 1} 次 (延遲 {d
            Thread.Sleep(delay);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"[Thread {threadId}] 錯誤: {ex.Message}");
            return;
        }
    }
}
```

```
        }  
    }  
}  
}
```

■ 執行結果範例

執行後可能看到類似輸出：

csharp

□ 複製程式碼

資料庫已初始化。

WAL 模式啟用結果: wal

開始平行寫入...

[Thread 7] 資料庫被鎖定，重試第 1 次 (延遲 100ms)

[Thread 15] 資料庫被鎖定，重試第 2 次 (延遲 200ms)

寫入完成。

執行完畢後，可用 SQLite 檢查：

sql

□ 複製程式碼

```
SELECT COUNT(*), thread_id FROM log_table GROUP BY thread_id;
```

應該會看到 20 條執行緒都成功插入記錄。

 重點回顧

設定	說明
WAL 模式	允許多讀單寫，不會整體鎖定檔案
Cache=Shared	同一進程共享快取，降低鎖定衝突
獨立連線	每個執行緒各自開連線，避免非 thread-safe 問題
重試機制	捕捉 SQLite Error 5: database is locked 時稍作延遲再試

是否需要我幫你再加上

👉 多執行緒同時「讀寫混合」操作（例如有些執行緒查詢、有些執行緒寫入）的版本？



```
System.Exception: 'You need to call SQLitePCL.raw.SetProvider(). If  
you are using a bundle package, this is done by calling  
SQLitePCL.Batteries.Init().'
```

這個錯誤是因為 **SQLitePCL** 的初始化沒有正確設定。這通常會出現在使用 `Microsoft.Data.Sqlite` 或其他 SQLite 包的時候，特別是當 SQLite 的底層提供者 (`SQLitePCL.raw`) 還沒被初始化的情況。

為了解決這個問題，你需要在程式啟動時明確初始化 `SQLitePCL.Batteries`，這樣才能使用 SQLite。

🛠 解決方法

在你的 `Main` 方法或程式的初始化部分，加入以下程式碼來初始化 `SQLitePCL.Batteries`：

csharp

복製程式碼

```
using SQLitePCL;

class Program
{
    static void Main()
    {
        // 初始化 SQLitePCL 底層提供者
        Batteries.Init();

        const string dbFile = "multi_thread_test.db";
        // 剩下的程式邏輯...
    }
}
```

💡 為何需要這個初始化？

`SQLitePCL.raw` 是 SQLite 的低階提供者，`SQLitePCL.Batteries.Init()` 會自動幫你設置並載入必要的 SQLite 驅動程式。若沒這麼做，會出現你現在看到的錯誤訊息。

🔗 完整範例 (更新後)

csharp

 複製程式碼

```
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Data.Sqlite;
using SQLitePCL; // 記得引用 SQLitePCL 命名空間

class Program
{
    static void Main()
    {
        // 初始化 SQLitePCL 底層提供者
        Batteries.Init();

        const string dbFile = "multi_thread_test.db";

        // 若檔案不存在就初始化資料庫
        if (!File.Exists(dbFile))
        {
            InitializeDatabase(dbFile);
        }

        // 啟用 WAL 模式 (支援多執行緒多讀單寫)
        using (var conn = new SqliteConnection($"Data Source={dbFile}"))
        {
            conn.Open();
            using var cmd = conn.CreateCommand();
```

```
cmd.CommandText = "PRAGMA journal_mode=WAL;"  
var mode = cmd.ExecuteScalar();  
Console.WriteLine($"WAL 模式啟用結果: {mode}");  
  
}  
  
Console.WriteLine("開始平行寫入...");  
  
var connStr = new SqliteConnectionStringBuilder  
{  
    DataSource = dbFile,  
    Cache = SqliteCacheMode.Shared,  
    Mode = SqliteOpenMode.ReadWrite  
}.ToString();  
  
// 使用 20 個執行緒平行寫入資料  
Parallel.For(0, 20, i =>  
{  
    for (int j = 0; j < 5; j++)  
    {  
        WriteWithRetry(connStr, i, j);  
    }  
});  
  
Console.WriteLine("寫入完成。");  
}  
  
static void InitializeDatabase(string dbFile)  
{  
    using var conn = new SqliteConnection($"Data Source={dbFile}");
```

```
conn.Open();
using var cmd = conn.CreateCommand();
cmd.CommandText = @"
CREATE TABLE log_table (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    thread_id INTEGER,
    msg TEXT,
    created_at TEXT
);
";
cmd.ExecuteNonQuery();
conn.Close();

Console.WriteLine("資料庫已初始化。");
}

static void WriteWithRetry(string connStr, int threadId, int seq)
{
    const int maxRetry = 5;
    for (int retry = 0; retry < maxRetry; retry++)
    {
        try
        {
            using var conn = new SqliteConnection(connStr);
            conn.Open();
            using var cmd = conn.CreateCommand();

            cmd.CommandText = "INSERT INTO log_table (thread_id, msg, created_at) VALUES (@t,
            cmd.Parameters.AddWithValue("@t", threadId);
```

```

        cmd.Parameters.AddWithValue("@m", $"Thread {threadId} - record {seq}");
        cmd.ExecuteNonQuery();

        return; // 成功就跳出
    }
    catch (SqliteException ex) when (ex.SqliteErrorCode == 5) // database is Locked
    {
        int delay = 100 * (retry + 1);
        Console.WriteLine($"[Thread {threadId}] 資料庫被鎖定，重試第 {retry + 1} 次 (延遲 {delay} 毫秒)");
        Thread.Sleep(delay);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"[Thread {threadId}] 錯誤: {ex.Message}");
        return;
    }
}
}
}

```

套件安裝

如果還沒安裝 `SQLitePCLRaw.bundle_e_sqlite3`，可以在 NuGet 安裝：

bash

 複製程式碼

```
dotnet add package SQLitePCLRaw.bundle_e_sqlite3
```

這個包會包括 SQLite 所需的所有依賴。

現在，重新執行這個程式應該能順利初始化並執行多執行緒存取 SQLite 的操作了！

如果還有其他問題，隨時告訴我。



上面的範例是使用 Microsoft.Data.Sqlite.Core 和
SQLitePCLRaw.bundle_e_sqlite3 元件

如果改用 System.Data.SQLite.Core 可以執行嗎

如果可以 請給我對應範例

非常好問題 🌟

你提到要把範例改成使用 System.Data.SQLite.Core，這是完全可行的，只是有幾個關鍵差異需要注意。



一、差異重點

項目	Microsoft.Data.Sqlite	System.Data.SQLite
套件來源	微軟官方 (EF Core 同步)	SQLite 官方 (原生 .NET 封裝)

項目	Microsoft.Data.Sqlite	System.Data.SQLite
初始化需求	需 SQLitePCLRaw.bundle_e_sqlite3 + Batteries.Init()	不需要額外初始化
命名空間	Microsoft.Data.Sqlite	System.Data.SQLite
類別名稱	SqliteConnection / SqliteCommand	SQLiteConnection / SQLiteCommand
相容性	適合 .NET 5/6/7	較傳統，但穩定；支援 .NET Framework & .NET 6/7

✓ 二、改寫後可執行範例（使用 System.Data.SQLite.Core ）

這個版本不需要 SQLitePCL 初始化，也不需要 bundle_e_sqlite3 。

請先安裝套件：

bash

複製程式碼

```
dotnet add package System.Data.SQLite.Core
```

💻 完整範例

csharp

複製程式碼

```
using System;
using System.IO;
```

```
using System.Threading;
using System.Threading.Tasks;
using System.Data.SQLite;

class Program
{
    static void Main()
    {
        const string dbFile = "multi_thread_sqlite_test.db";

        // 若檔案不存在就初始化資料庫
        if (!File.Exists(dbFile))
        {
            InitializeDatabase(dbFile);
        }

        // 啟用 WAL 模式 (多執行緒寫入推薦)
        using (var conn = new SQLiteConnection($"Data Source={dbFile};Version=3;"))
        {
            conn.Open();
            using var cmd = new SQLiteCommand("PRAGMA journal_mode=WAL;", conn);
            var result = cmd.ExecuteScalar();
            Console.WriteLine($"WAL 模式啟用結果: {result}");

            Console.WriteLine("開始平行寫入...");
        }

        string connStr = $"Data Source={dbFile};Version=3;Cache=Shared;Journal Mode=WAL;";
    }
}
```

```
// 使用 20 個執行緒同時寫入資料
Parallel.For(0, 20, i =>
{
    for (int j = 0; j < 5; j++)
    {
        WriteWithRetry(connStr, i, j);
    }
});

Console.WriteLine("寫入完成。");
}

static void InitializeDatabase(string dbFile)
{
    SQLiteConnection.CreateFile(dbFile);

    using var conn = new SQLiteConnection($"Data Source={dbFile};Version=3;");
    conn.Open();

    using var cmd = new SQLiteCommand(@"
        CREATE TABLE log_table (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            thread_id INTEGER,
            msg TEXT,
            created_at TEXT
        );
    ", conn);
    cmd.ExecuteNonQuery();
}
```

```
        Console.WriteLine("資料庫已初始化。");
    }

    static void WriteWithRetry(string connStr, int threadId, int seq)
    {
        const int maxRetry = 5;

        for (int retry = 0; retry < maxRetry; retry++)
        {
            try
            {
                using var conn = new SQLiteConnection(connStr);
                conn.Open();

                using var cmd = new SQLiteCommand(
                    "INSERT INTO log_table (thread_id, msg, created_at) VALUES (@t, @m, datetime('
                    conn);
                cmd.Parameters.AddWithValue("@t", threadId);
                cmd.Parameters.AddWithValue("@m", $"Thread {threadId} - record {seq}");
                cmd.ExecuteNonQuery();

                return; // 成功就離開
            }
            catch (SQLiteException ex)
            {
                if (ex.ErrorCode == (int)SQLiteErrorCode.Busy) // 資料庫被鎖定
                {
                    int delay = 100 * (retry + 1);
                    Console.WriteLine($"[Thread {threadId}] 資料庫被鎖定，重試第 {retry + 1} 次 (延遲 {delay} 毫秒)");
                }
            }
        }
    }
}
```

```
        Thread.Sleep(delay);
        continue;
    }

    Console.WriteLine($"[Thread {threadId}] SQLite 錯誤: {ex.Message}");
    return;
}

catch (Exception ex)
{
    Console.WriteLine($"[Thread {threadId}] 一般錯誤: {ex.Message}");
    return;
}
}
```

⚙️ 三、說明重點

- ◆ WAL 模式

sql

COPY 複製程式碼

```
PRAGMA journal_mode=WAL;
```

開啟後允許「多讀單寫」，非常適合多執行緒存取。

◆ 連線字串建議

csharp

 複製程式碼

```
string connStr = "Data Source=mydb.db;Version=3;Cache=Shared;Journal Mode=WAL;"
```

- Cache=Shared : 讓同一進程的多連線共享快取
 - Journal Mode=WAL : 確保在每次開啟連線時仍維持 WAL 模式
-

◆ 鎖定重試機制

System.Data.SQLite 在多執行緒時可能仍丟出 `SQLiteErrorCode.Busy`，
這時透過延遲重試即可減少衝突。

四、建議選擇