Swift 優雅的使用 FMDB

快速入手SQLite

Nick Lin

FMDB

iOS SQLite資料庫管理 第一首選(Github 星星破萬)

支援多線程存取資料庫

支援寫入錯誤退回寫入動作

支援事務型態操作

如何在 Swift 中使用 FMDB

首先先從Github下載回來
拉近專案

建立Bridging-Header.h
#import "FMDB.h"

build Phases -> Link Binary With Libraries 添加
libsqlite3.tbd

前置作業搞定收工

FMDB 的常用類別

FMDatabaseQueue
資料庫存取(多線程安全)

FMDatebase
資料庫存取

FMResultSet
資料庫取出的資料格式

FMDB 基本語法

FMDatabaseQueue

// 建立資料庫或者是取得資料庫
FMDatabaseQueue(path: dbPath)

// 建立一個block 執行資料庫操作 單次指令 單次操作
FMDatabaseQueue.inDatabase { (db) in // 操作內容 }

// 建立一個事務 所有指令 一次操作
FMDatabaseQueue.inTransaction { (db,nil) in // 操作內容 }

FMDB 基本語法

FMDatabase

// 資料庫更新(新增,修改,刪除)
FMDatabase.executeUpdate(SQL , values: [value])

// 資料庫查詢
FMDatabase.executeQuery("SELECT * FROM user", values: nil)

除了查詢之外，所有對於資料庫的操作都是更新

資料庫語法簡介

新增

```
INSERT INTO
user ( gender , year )
values ( man , 1999)
```

刪除

```
DELETE FROM user
WHERE gender = man
```

修改

```
UPDATE user
SET year = 2000
WHERE gender = man
```

搜尋

```
SELECT *
FROM user
WHERE gender = man AND
year = 1999
```

有沒有人發現，沒有講到要怎樣建立資料庫表格

既然都用第三方來協助使用資料庫了

當然也可以使用第三方來建立資料庫表格啊

個人目前是使用 MAMP 來建立表格及檢視資料庫內容
https://www.mamp.info/en/

有免費版跟付費版(其實架站才是 MAMP 的主打)

先來看看基本的 FMDB 對資料庫的操作

```
// 這是新增
FMDatabaseQueue().inDatabase { (db) in
    do {
        try db.executeUpdate( "INSERT INTO user(name,gender,year) values (?,?,?)"
            , values: [ "Nick" , "Man" , 1999 ] )
    }catch{
        print(error)
    }
}

// 這是修改
FMDatabaseQueue().inDatabase { (db) in
    do {
        try db.executeUpdate("UPDATE user SET name = ? WHERE gender = ?"
            , values: [ "Nick Lin" , "Man" ])
    }catch{
        print(error)
    }
}
```

```swift
// 這是刪除
FMDatabaseQueue().inDatabase { (db) in
    do {
        try db.executeUpdate("DELETE FROM user WHERE year = ? "
            , values: [ 1999 ])
    }catch{ print(error) }
}

// 這是讀取
FMDatabaseQueue().inDatabase { (db) in
    do {
        let rs = try db.executeQuery("SELECT * FROM user WHERE gender = ? "
            , values: [ "Man" ])
        while rs.next(){
            let user = ???
            user.name       = rs.boolForColumn("name")
            user.gender     = rs.stringForColumn("gender")
            user.year       = rs.intForColumn("year")
            // 再把 user 拿來使用
        }
    }catch{  print(error) }
}
```

使用Class當成資料結構 還可以順便寫Func

```swift
// MARK:- 使用者資料表
class DB_user : NSObject {
    var id              : Int32!
    var school          : String!
    var department      : String!
    var grade           : String!
    var uClass          : String!
    var birthday        : String!
    var theme           : Int32!
    var create_time     : String!
    var edit_time       : String!
}
```

```swift
// MARK:- 使用者資料表 功能
extension DB_user{

    // 新增一筆資料
    static func instertData(data:DB_user){…}

    // 更新一筆資料
    static func updateData(data:DB_user){…}

    // 刪除一筆資料
    static func deleteData(data:DB_user){…}

    // 讀取符合搜尋條件的資料
    static func loadMatchData(Match:String,
                              value:[AnyObject])->[DB_user]{…}

    // 讀取全部資料
    static func loadAllData()->[DB_user]{…}
}
```

用Class當資料結構，看起來不錯耶

BUT！（人生最厲害就是這個 BUT！）

你有十個表格，你就要寫十個表格的每一個 Func

有一百個表格，你就 GG 了

哇！

有沒有方法可以不要寫那麼多，又可以方便使用呢

謎之音:

若是沒有,你就是來騙台錢的!