**iOS开发数据库篇—FMDB简单介绍**

**一、简单说明**

1.什么是FMDB

FMDB是iOS平台的SQLite数据库框架

FMDB以OC的方式封装了SQLite的C语言API

2.FMDB的优点

使用起来更加面向对象，省去了很多麻烦、冗余的C语言代码

对比苹果自带的Core Data框架，更加轻量级和灵活

提供了多线程安全的数据库操作方法，有效地防止数据混乱

3.FMDB的github地址

https://github.com/ccgus/fmdb

**二、核心类**

FMDB有三个主要的类

（1）FMDatabase

一个FMDatabase对象就代表一个单独的SQLite数据库

用来执行SQL语句

（2）FMResultSet

使用FMDatabase执行查询后的结果集

（3）FMDatabaseQueue

用于在多线程中执行多个查询或更新，它是线程安全的

**三、打开数据库**

通过指定SQLite数据库文件路径来创建FMDatabase对象

FMDatabase \*db = [FMDatabase databaseWithPath:path];

if (![db open]) {

NSLog(@"数据库打开失败！");

}

文件路径有三种情况

（1）具体文件路径

如果不存在会自动创建

（2）空字符串@""

会在临时目录创建一个空的数据库

当FMDatabase连接关闭时，数据库文件也被删除

（3）nil

会创建一个内存中临时数据库，当FMDatabase连接关闭时，数据库会被销毁

**四、执行更新**

在FMDB中，除查询以外的所有操作，都称为“更新”

create、drop、insert、update、delete等

使用executeUpdate:方法执行更新

- (BOOL)executeUpdate:(NSString\*)sql, ...

- (BOOL)executeUpdateWithFormat:(NSString\*)format, ...

- (BOOL)executeUpdate:(NSString\*)sql withArgumentsInArray:(NSArray \*)arguments

示例

[db executeUpdate:@"UPDATE t\_student SET age = ? WHERE name = ?;", @20, @"Jack"]

**五、执行查询**

查询方法

- (FMResultSet \*)executeQuery:(NSString\*)sql, ...

- (FMResultSet \*)executeQueryWithFormat:(NSString\*)format, ...

- (FMResultSet \*)executeQuery:(NSString \*)sql withArgumentsInArray:(NSArray \*)arguments

示例

// 查询数据

FMResultSet \*rs = [db executeQuery:@"SELECT \* FROM t\_student"];

// 遍历结果集

while ([rs next]) {

NSString \*name = [rs stringForColumn:@"name"];

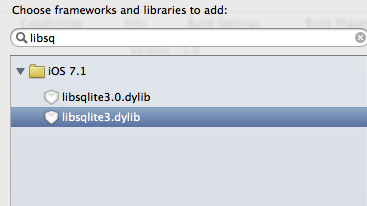
int age = [rs intForColumn:@"age"];

double score = [rs doubleForColumn:@"score"];

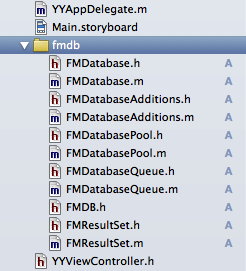
}

**六、代码示例**

1.新建一个项目，导入libsqlite3库，并在项目中包含主头文件

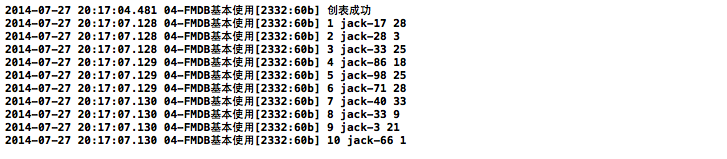


2.下载第三方框架FMDB



|  |
| --- |
| //  // YYViewController.m  // 04-FMDB基本使用  //  // Created by apple on 14-7-27.  // Copyright (c) 2014年 wendingding. All rights reserved.  //  #import "YYViewController.h"  #import "FMDB.h"  @interface YYViewController ()  @property(nonatomic,strong)FMDatabase \*db;  @end  @implementation YYViewController  - (void)viewDidLoad  {  [super viewDidLoad];  //1.获得数据库文件的路径  NSString \*doc=[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) lastObject];  NSString \*fileName=[doc stringByAppendingPathComponent:@"student.sqlite"];    //2.获得数据库  FMDatabase \*db=[FMDatabase databaseWithPath:fileName];    //3.打开数据库  if ([db open]) {  //4.创表  BOOL result=[db executeUpdate:@"CREATE TABLE IF NOT EXISTS t\_student (id integer PRIMARY KEY AUTOINCREMENT, name text NOT NULL, age integer NOT NULL);"];  if (result) {  NSLog(@"创表成功");  }else  {  NSLog(@"创表失败");  }  }  self.db=db;  }  -(void)touchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event  {  [self delete];  [self insert];  [self query];  }  //插入数据  -(void)insert  {  for (int i = 0; i<10; i++) {  NSString \*name = [NSString stringWithFormat:@"jack-%d", arc4random\_uniform(100)];  // executeUpdate : 不确定的参数用?来占位  [self.db executeUpdate:@"INSERT INTO t\_student (name, age) VALUES (?, ?);", name, @(arc4random\_uniform(40))];  // [self.db executeUpdate:@"INSERT INTO t\_student (name, age) VALUES (?, ?);" withArgumentsInArray:@[name, @(arc4random\_uniform(40))]];    // executeUpdateWithFormat : 不确定的参数用%@、%d等来占位  // [self.db executeUpdateWithFormat:@"INSERT INTO t\_student (name, age) VALUES (%@, %d);", name, arc4random\_uniform(40)];  }  }  //删除数据  -(void)delete  {  // [self.db executeUpdate:@"DELETE FROM t\_student;"];  [self.db executeUpdate:@"DROP TABLE IF EXISTS t\_student;"];  [self.db executeUpdate:@"CREATE TABLE IF NOT EXISTS t\_student (id integer PRIMARY KEY AUTOINCREMENT, name text NOT NULL, age integer NOT NULL);"];  }  //查询  - (void)query  {  // 1.执行查询语句  FMResultSet \*resultSet = [self.db executeQuery:@"SELECT \* FROM t\_student"];    // 2.遍历结果  while ([resultSet next]) {  int ID = [resultSet intForColumn:@"id"];  NSString \*name = [resultSet stringForColumn:@"name"];  int age = [resultSet intForColumn:@"age"];  NSLog(@"%d %@ %d", ID, name, age);  }  }  @end |

打印查看结果：



提示：

如果ID设置为逐渐，且设置为自动增长的话，那么把表中的数据删除后，重新插入新的数据，ID的编号不是从0开始，而是接着之前的ID进行编号。

注意：

不要写成下面的形式，不要加''，直接使用%@,它会自动认为这是一个字符串。

