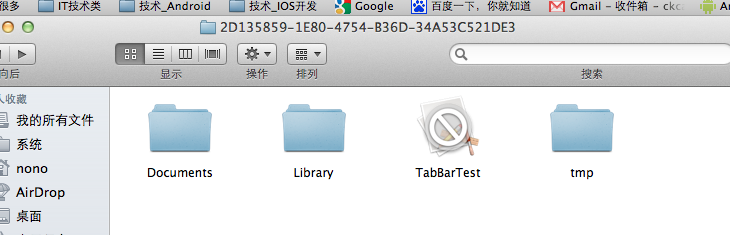
[IOS学习九：ios开发之数据的持久化存储机制](http://blog.csdn.net/nono_love_lilith/article/details/7539659)

分类： [IOS开发学习](http://blog.csdn.net/Nono_Love_Lilith/article/category/1119876)

2012-05-13 10:12 4404人阅读 评论(5) [收藏](javascript:void(0);) 举报

[ios](http://blog.csdn.net/tag/details.html?tag=ios)[存储](http://blog.csdn.net/tag/details.html?tag=%e5%ad%98%e5%82%a8)[sqlite](http://blog.csdn.net/tag/details.html?tag=sqlite)[数据库](http://blog.csdn.net/tag/details.html?tag=%e6%95%b0%e6%8d%ae%e5%ba%93)[database](http://blog.csdn.net/tag/details.html?tag=database)[sql](http://blog.csdn.net/tag/details.html?tag=sql)

IOS中数据的持久化保存这块内容，类似于Android中文件的几种常见的存储方式。 对于数据的持久化存储，ios中一般提供了4种不同的机制。 1.属性列表 2.对象归档 3.数据库存储（SQLite3） 4.苹果公司提供的持久性工具Core Data。   其实储存的形式无非就这么几种，而我们还必须要关心的是，这些文件会被放置在那个文件下，然后如何读取。 也就是说：IOS上数据存储，我们要了解的两点，数据存储格式（也就是存储机制），数据存储位置。 1》文件如何存储（如上面4点） 2》文件存储在哪里。 对于数据的操作，其实我们关心的是操作的速率。 就好比在Adnroid中偏好存储，数据库存储，io存储一样。 我大致问了我们公司新来的ios哥们，他说他们培训机构基本对数据操作这块就讲了属性列表和数据库，以及普通的文件存储（比如音视频图这些多媒体数据）。 我就只好先看看书了。   一：应用文件目录 首先我们来看了解下ios数据存储位置，因为只有知道位置路径我们才能去读取数据，而数据的持久化机制不过是针对操作速率来考虑的， 比如我们大致知道属性列表（既键值对形式）的存储熟虑应该高于数据库高于io文件流存储。 我们在选择用何种机制存储数据，主要也是看数据的形式。   一个ios应用安装后大致会有如下文件夹及其对应路径：



在mac上看模拟器中应用路径： /Users/nono/Library/Application Support/iPhone Simulator/5.1/Applications/2D135859-1E80-4754-B36D-34A53C521DE3  你在finder中的home下可能找不到Library这个目录，因为貌似是影藏起来了（我这机器上是，在终端可以看到）。 最后那一窜的类似序列号的东西就是ios自动给应用生成的一组应用唯一识别码最为了应用的home目录名。 其下面就是上图所示了。 书上对这些文件夹介绍：

Document：应用程序将其数据存储在这个文件夹下，基于NSUserDefaults的首选项的设置除外。

简单理解是，基本上我们要操作的一些数据都是存储在这个文件夹下面的

TIPS:这边提下一点，对于ios系统这么分配文件夹，是因为在设备进行同步时，ITunes有选择性的意识来备份文件。

比如我们可以猜到，tmp下的应该就不会备份了。

对于Document文件夹目录路径的获取，API提供了这么一种方法：

**[cpp]** view plaincopy

1. NSArray \*paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
2. NSString \*docPath = [paths objectAtIndex:0];

Library:基于NSUserDefault首选项设置存储在其下Preferences文件夹中，简单来说，这个文件夹一般你很少操作到。

书上对于这部分基本没介绍。估计对于初级部分是跳过了。

Tmp：应用临时存储文件，当不需要时，应用负责删除其下的文件数据。

该文件也提供了目录获取方法：

**[cpp]** view plaincopy

1. NSString \*tmpDoc = NSTemporaryDirectory();

应用程序文件：这个基本没提到书上，但是我们大致可以猜测，这就是整个应用程序的程序文件夹吧。

好了，以上我们大致解决了我们提到的第一个点，文件存储目录

二：数据存储机制

1.属性列表

这个其实我们早见过，plist就是，感觉用来存储键值对小数据是最合适，因为速率很高。

这个存储机制很简单，对于前面我们使用过了在plist文件来读取数据填充一些列表，只不过那会plist文件存储位置不同，

用的是Mainbundle什么的来返回文件夹，其实这边我也推测，上面提到有个应用程序文件夹，它下面的文件就是这么来读取的~（反正暂时不管他）

这边不过就是改变了存储位置，数据操作还是一样的

**[cpp]** view plaincopy

1. NSArray \*paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
2. NSString \*docPath = [paths objectAtIndex:0];
3. NSString \*myFile = [docPath stringByAppendingPathComponent:@"my.list"];
4. //读取文件
5. NSArray \*array = [[NSArray alloc] initWithContentsOfFile:myFile];
6. //操作完若修改了数据则，写入文件
7. [array writeToFile:myFile atomically:YES];

2.对象归档

上面的属性列表存储机制，我们都知道，这个机制支持NSArray,NSDictionary,NSData,NSString,NSNumber,NSDate 等等

这些对象直接写入plist文件中。

那么对于一些复杂对象，我要保存整个这个对象数据呢？

反正我是这么觉得，这个机制很像java中的对象整体序列化。当然，这些数据在读取是就需要遵循一种墨守成规的协议了。

首先我们定义的对象类，必须实现NSCoding和NSCopying协议（额，网上说后面这个不实现也可以，我猜是他对象没有copy操作，因此没出错）书本上反正是实现了这两个协议 然后归档中用到的操作类 NSKeyedArchiver 这边我们定义一个对象，h文件中定义两属性，申明要实现的NSCoding和NSCopying协议 实现文件

**[cpp]** view plaincopy

1. //
2. //  TestObj.m
3. //  DataStorageTest
4. //
5. //  Created by Nono on 12-5-12.
6. //  Copyright (c) 2012年 NonoWithLilith. All rights reserved.
7. //
9. #import "TestObj.h"
11. @implementation TestObj
12. @synthesize stringA = stringA;
13. @synthesize stringB = stringB;
15. #pragma mark -
16. #pragma NSCoding协议实现实现
17. - (**void**)encodeWithCoder:(NSCoder \*)aCoder
18. {   //encoder
19. [aCoder encodeObject:stringAforKey:@"1"];
20. [aCoder encodeObject:stringBforKey:@"2"];
21. }
22. - (id)initWithCoder:(NSCoder \*)aDecoder
23. {
24. //decoder
25. **if** (self = [superinit]) {
26. stringA = [[aDecoder decodeObjectForKey:@"1"] retain];
27. stringB = [[aDecoder decodeObjectForKey:@"2"] retain];
28. }
29. returnself;
30. }
32. #pragma NSCopying协议实现
33. - (id)copyWithZone:(NSZone \*)zone
34. {
35. TestObj \*copy = [[[selfclass] allocWithZone:zone] init];
36. copy.stringA = [[self.stringAcopyWithZone:zone] autorelease];
37. copy.stringB = [[self.stringBcopyWithZone:zone] autorelease];
38. **return** copy;
39. }
40. @end

然后是对对象归档的读取和写入

**[cpp]** view plaincopy

1. //读取归档文件
2. NSData \*data = [[NSMutableDataalloc] initWithContentsOfFile:myFile];
3. NSKeyedUnarchiver \*unarchiver = [[NSKeyedUnarchiveralloc] initForReadingWithData:data];
4. TestObj \* test = [unarchiver decodeObjectForKey:@"data"];
5. [unarchiver finishDecoding];
6. [data release];
7. [unarchiver release];
9. //写入归档文件
10. NSMutableData \*data1 = [[NSMutableDataalloc] init];
11. NSKeyedArchiver \*archiver = [[NSKeyedArchiveralloc] initForWritingWithMutableData:data1];
12. [archiver encodeObject:test forKey:@"data"];
13. [archiver finishEncoding];
14. [data writeToFile:myFile atomically:YES];
15. [data1 release];
16. [archiver release];
17. [test release];

但是问了下新同事，据说这个用到也是蛮少，至少他目前。

但是，我看了下，觉得这个和Android 中Parcelable

太尼玛像似了

三.数据库存储

和Android一样，ios中也是用了SQLite3这种嵌入式数据库。

这个网上例子是很多了。我这边就大致看下了数据库的打开，

数据库表创建，查询，插入

**[cpp]** view plaincopy

1. //数据库操作
2. sqlite3 \*database;
3. // const NSString \* dbname = @"mydb"
4. **int** result;
5. //打开一个指定路径的现有的数据库，如果没有则会新建一个db库
6. result =  sqlite3\_open([myFile UTF8String], &database);
7. **if** (result != SQLITE\_OK) {
8. sqlite3\_close(database);
9. }
11. //创建一个db表
12. **char** \*errorMsg;
13. NSString \*sql\_create\_table = @"CREATE TABLE IF NOT EXISTS NONOTABLE 省略~~~~~~~~~~~~~";
14. **int** result1 ;
15. //sqlite\_exec用了针对sqlite3运行任何不要返回数据的命令，它用于执行更新，插入和删除。简单来说，这个方法执行的都是一些无需返回数据（虽然我们可能获取一个状态值。）。
16. result1 = sqlite3\_exec(database, [sql\_create\_table UTF8String], NULL, NULL, &errorMsg);
18. //检索查询操作
19. **int** result2 ;
20. sqlite3\_stmt \*statment;
21. NSString \*sql\_selected = @"查询语句";
22. result2 = sqlite3\_prepare\_v2(database, [sql\_selected UTF8String], -1, &statment, nil);
23. **if**(result2 == SQLITE\_OK){
24. //单步操作
25. **while** (sqlite3\_step(statment) == SQLITE\_ROW) {
26. **int** row = sqlite3\_column\_int(statment, 0);
27. **char** \* rpwData = sqlite3\_column\_text(statment, 1);
28. }
29. sqlite3\_finalize(statment);
30. }

33. //绑定变量，既就是插入操作的一种变种，比如我么那上面提到sqlite\_exec可以执行插入操作，插入内容直接是写在sql字窜里，但是考虑到字窜涉及到无效的符号以及会一些严重的注入漏洞（比如以前听过的引号符号）。
34. NSString \*sql\_bind = @"insert into foo value(?,?)";
35. result2 = sqlite3\_prepare\_v2(database, [sql\_selected UTF8String], -1, &statment, nil);
36. **if**(result2 == SQLITE\_OK){
37. sqlite3\_bind\_int(statment, 1, 235);
38. sqlite3\_bind\_text(statment, 2, "test", -1, nil);
39. sqlite3\_finalize(statment);
40. }
41. **if** (sqlite3\_step(statment) != SQLITE\_DONE)
42. NSLog(@"error");
43. sqlite3\_finalize(statment);

46. sqlite3\_close(database);

关于更多的，大伙可以自行百度，因为数据库的操作语法太怪异了，书上说是基本是基于c的，本人没学过c。看得有点心烦~

4。Core Data存储机制

大致浏览下基本感觉就是将对象归档搞成了可视化和简单化。

这块内容比较多。网上资料也挺丰富的。

暂时不做介绍了。

总结下：其实对于ios数据存储，最常用和主要要掌握的就是属性列表和数据库，因为两个是出镜率比较高的。

其他可能在数据存明显体现出储优势时，我们会去考虑用另外两种机制。

基础的来说，必须掌握属性列表和sqlite的操作存储。

[**syxChina**](http://www.cnblogs.com/syxchina/)

个人开发者

1. [CnBlogs](http://www.cnblogs.com/)
2. [Home](http://www.cnblogs.com/syxchina/)
3. [New Post](http://www.cnblogs.com/syxchina/admin/EditPosts.aspx?opt=1)
4. [Contact](http://space.cnblogs.com/msg/send/syx278250658)
5. [Admin](http://www.cnblogs.com/syxchina/admin/EditPosts.aspx)
6. [Rss](http://www.cnblogs.com/syxchina/rss)

Posts - 262  Articles - 0  Comments - 91

[**IOS之数据持久化**](http://www.cnblogs.com/syxchina/archive/2012/09/17/2689830.html)

9.1 数据持久化概述

9.2 iOS应用程序目录结构

9.3 读写属性列表

9.4 对象归档

9.5 访问SQLite

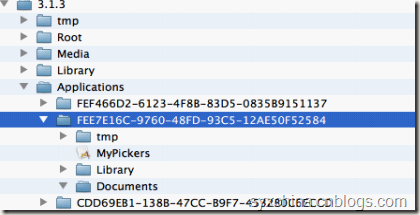
**9.1 数据持久化概述**

iOS中可以有四种持久化数据的方式： 属性列表、对象归档、SQLite3和Core Data

**9.2 iOS应用程序目录结构**

iOS应用程序运行在Mac os模拟器时候，有一下临时目录模拟器3.1.3为例子：

/Users/tony/Library/Application Support/iPhone Simulator/3.1.3/Applications

[](http://images.cnblogs.com/cnblogs_com/syxchina/201209/201209172255407517.png)

IOS应用程序采用沙盒原理设计，ios每个应用程序都有自己的3个目录(Document,Library,tmp)，互相之间不能访问。

Documents存放应用程序的数据。

Library目录下面还有Preferences和Caches目录，Preferences目录存放应用程序的使用偏好，Caches目录与Documents很相 似可以存放应用程序的数据。

tmp目录供应用程序存储临时文件。

**9.3 读写属性列表**

读取Documents目录下文件

可以获得应用程序的Documents文件夹。

NSArray\* myPaths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

NSString\* myDocPath = [myPaths objectAtIndex:0];

获取文件的完整路径。

- (NSString\*)filePath:(NSString\*)fileName {

NSArray\* myPaths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

NSString\* myDocPath = [myPaths objectAtIndex:0];

NSString\* filePath = [myDocPath stringByAppendingPathComponent:fileName];

return filePath;

}

获取tmp目录

获取应用程序的tmp目录要比获取Documents目录容易的多。使用函数NSTemporaryDirectory ()可以获得tmp目录路径。

NSString\* tempPath = NSTemporaryDirectory();

获取文件的完整路径。

NSString\* tempFile = [tempPath stringByAppendingPathComponent:@"properties.plist"];

属性列表文件实例 ：PropertesList

[](http://images.cnblogs.com/cnblogs_com/syxchina/201209/201209172255422109.png)

PropertesListViewController.h

[](javascript:void(0);)

#import "Student.h"

@interface ViewController : UIViewController

@property (retain, nonatomic) IBOutlet UITextField \*studentNo;

@property (retain, nonatomic) IBOutlet UITextField \*studentName;

@property (retain, nonatomic) IBOutlet UITextField \*studentClass;

- (IBAction)save:(id)sender;

- (IBAction)load:(id)sender;

- (IBAction)endEditing:(id)sender;

- (IBAction)saveToArchiver:(id)sender;

- (IBAction)loadFromArchiver:(id)sender;

- (NSString\*)filePath:(NSString\*)fileName;

@end

[](javascript:void(0);)

PropertesListViewController.m

[](javascript:void(0);)

@synthesize studentNo;

@synthesize studentName;

@synthesize studentClass;

- (NSString\*)filePath:(NSString\*)fileName {

NSArray\* myPaths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

NSString\* myDocPath = [myPaths objectAtIndex:0];

NSString\* filePath = [myDocPath stringByAppendingPathComponent:fileName];

return filePath;

}

- (IBAction)save:(id)sender {

NSString\* fileName = [self filePath:@"properties.plist"];

NSLog(fileName);

NSMutableArray\* data = [[NSMutableArray alloc]init];

[data addObject:studentNo.text];

[data addObject:studentName.text];

[data addObject:studentClass.text];

[data writeToFile:fileName atomically:YES];

}

- (IBAction)load:(id)sender {

NSString\* fileName = [self filePath:@"properties.plist"];

if ([[NSFileManager defaultManager]fileExistsAtPath:fileName]) {

NSArray\* data = [[NSArray alloc]initWithContentsOfFile:fileName];

studentNo.text = [data objectAtIndex:0];

studentName.text = [data objectAtIndex:1];

studentClass.text = [data objectAtIndex:2];

[data release];

}

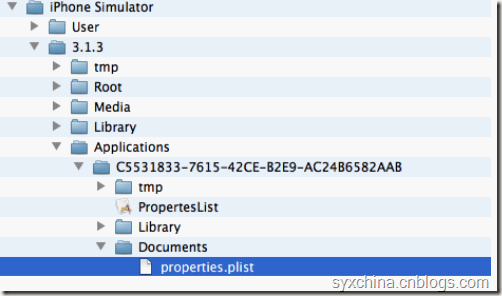
}

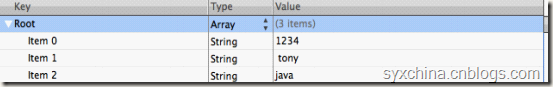
- (IBAction)endEditing:(id)sender {

[sender resignFirstResponder];

}

[](javascript:void(0);)

[](http://images.cnblogs.com/cnblogs_com/syxchina/201209/201209172255466193.png)

[](http://images.cnblogs.com/cnblogs_com/syxchina/201209/201209172255478177.png)

**9.4 对象归档**

对象归档实例：Encoding

对象归档

“归档”是值另一种形式的序列化，对模型对象进行归档的技术可以轻松将复杂的对象写入文件，然后再从中读取它们，只要在类中实现的每个属性都是基本数据类型（如int或float）或都是符合NSCoding协议的某个类的实例，你就可以对你的对象进行完整归档。

实现NSCoding协议

NSCoding协议声明了两个方法： -(void)encodeWithCoder:(NSCoder \*)aCoder，是将对象写入到文件中。

-(id)initWithCoder:(NSCoder \*)aDecoder，是将文件中数据读入到对象中。

实现NSCopying协议

NSCopying协议声明了一个方法： -(id)copyWithZone:(NSZone \*)zone ，是将对象复制方法。

Student.h

[](javascript:void(0);)

@interface Student : NSObject<NSCoding, NSCopying>

@property (retain, nonatomic) NSString\* studentNo;

@property (retain, nonatomic) NSString\* studentName;

@property (retain, nonatomic) NSString\* studentClass;

@end

[](javascript:void(0);)

Student.m

[](javascript:void(0);)

#import "Student.h"

@implementation Student

@synthesize studentNo = \_studentNo;

@synthesize studentName = \_studentName;

@synthesize studentClass = \_studentClass;

#pragma mark NSCopying

- (id)copyWithZone:(NSZone \*)zone {

Student\* copy = [[[self class]allocWithZone:zone]init];

copy.studentNo = [\_studentNo copyWithZone:zone];

copy.studentName = [\_studentName copyWithZone:zone];

copy.studentClass = [\_studentClass copyWithZone:zone];

return copy;

}

#pragma mark NSCoding

- (void)encodeWithCoder:(NSCoder \*)aCoder {

[aCoder encodeObject:\_studentNo forKey:@"studentNo"];

[aCoder encodeObject:\_studentName forKey:@"studentName"];

[aCoder encodeObject:\_studentClass forKey:@"studentClass"];

}

- (id)initWithCoder:(NSCoder \*)aDecoder {

\_studentNo = [aDecoder decodeObjectForKey:@"studentNo"];

\_studentName = [aDecoder decodeObjectForKey:@"studentName"];

\_studentClass = [aDecoder decodeObjectForKey:@"studentClass"];

return self;

}

-(NSString\*)description {

return [[[NSString alloc]initWithFormat:@"no:%@ name:%@ class:%@", \_studentNo, \_studentName, \_studentClass]autorelease];

}

- (void)dealloc {

[\_studentName release];

[\_studentClass release];

[\_studentNo release];

[super dealloc];

}

@end

[](javascript:void(0);)

EncodingViewController.h

详细见上。

EncodingViewController.m

[](javascript:void(0);)

- (IBAction)saveToArchiver:(id)sender {

NSString\* fileName = [self filePath:@"student.archiver"];

NSMutableData\* data = [NSMutableData data];

NSKeyedArchiver\* archiver = [[NSKeyedArchiver alloc]initForWritingWithMutableData:data];

Student\* student = [[Student alloc]init];

student.studentNo = studentNo.text;

student.studentName = studentName.text;

student.studentClass = studentClass.text;

[archiver encodeObject:student forKey:@"myStudent"];

[archiver finishEncoding];

[data writeToFile:fileName atomically:YES];

[archiver release];

[student release];

}

[](javascript:void(0);)

NSMutableData \* theData = [NSMutableData data];用于包含编码的数据。

NSKeyedArchiver \* archiver = [[NSKeyedArchiver alloc] initForWritingWithMutableData:theData];创建NSKeyedArchiver实例，用于将对象归档到此theData实例中。

[archiver encodeObject:student forKey:@"mystudent"]; 使用“键－值”对编码来对希望包含在归档中的对象进行归档。

[theData writeToFile:filename atomically:YES]; 写入数据到归档文件。

EncodingViewController.m

[](javascript:void(0);)

- (IBAction)loadFromArchiver:(id)sender {

NSString\* fileName = [self filePath:@"student.archiver"];

NSData\* data = [NSData dataWithContentsOfFile:fileName];

if ([data length] > 0) {

NSKeyedUnarchiver\* unArchiver = [[NSKeyedUnarchiver alloc]initForReadingWithData:data];

Student\* student = [unArchiver decodeObjectForKey:@"myStudent"];

studentNo.text = student.studentNo;

studentName.text = student.studentName;

studentClass.text = student.studentClass;

[unArchiver finishDecoding];

[unArchiver release];

}

}

[](javascript:void(0);)

NSData \* theData =[NSData dataWithContentsOfFile:filename];从归档文件中获得NSData实例。

NSKeyedUnarchiver \* archiver = [[NSKeyedUnarchiver alloc] initForReadingWithData:theData];

创建一个NSKeyedUnarchiver实例对数据进行解码。Student \*student = [archiver decodeObjectForKey:@"mystudent"];

使用与归档编码使用相同的键对象进行解码。

**9.5 访问SQLite**

**SQLite数据库**

SQLite是一个开源的嵌入式关系数据库，它在2000年由D. Richard Hipp发布，它的减少应用程序管理数据的开销，SQLite可移植性好，很容易使用，很小，高效而且可靠。

SQLite嵌入到使用它的应用程序中，它们共用相同的进程空间，而不是单独的一个进程。从外部看，它并不像一个RDBMS，但在进程内部，它却是完整的，自包含的数据库引擎。 嵌入式数据库的一大好处就是在你的程序内部不需要网络配置，也不需要管理。因为客户端和服务器在同一进程空间运行。SQLite 的数据库权限只依赖于文件系统，没有用户帐户的概念。SQLite 有数据库级锁定，没有网络服务器。它需要的内存，其它开销很小，适合用于嵌入式设备。你需要做的仅仅是把它正确的编译到你的程序。

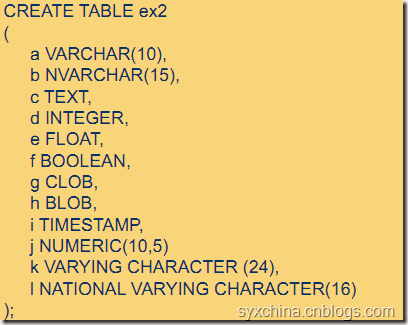
**SQLite数据类型**

SQLite是无类型的，这意味着你可以保存任何类型的数据到你所想要保存的任何表的任何列中, 无

论这列声明的数据类型是什么，对于SQLite来说对字段不指定类型是完全有效的，如:

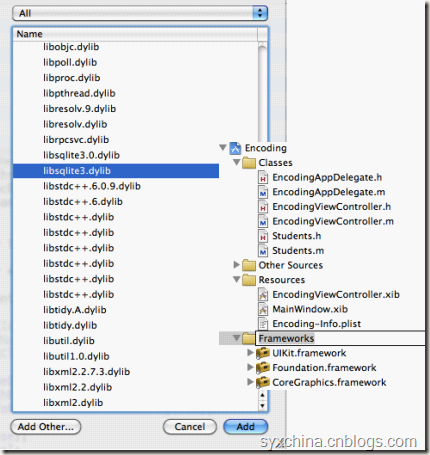
Create Table ex1(a, b, c);

SQLite允许忽略数据类型，但是仍然建议在你的Create Table语句中指定数据类型， 因为数据类型对于你和其他的程序员交流， 或者你准备换掉你的数据库引擎。 SQLite支持常见的数据类型， 如：

[](http://images.cnblogs.com/cnblogs_com/syxchina/201209/201209172255497395.png)

在iOS中使用SQLite3

为了能够在iOS中使用SQLite3需要是将libsqlite3.dylib类库添加到Xcode工程中，在工程的Frameworks（框架） 文件夹右键添加存在Frameworks

[](http://images.cnblogs.com/cnblogs_com/syxchina/201209/201209172255509936.png)

或者导航到 /Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/ iPhoneSimulator<version>.sdk/usr/lib 目录下面找到libsqlite3.dylib.

实例：StudentSQLite3

[](http://images.cnblogs.com/cnblogs_com/syxchina/201209/201209172255507145.png)

StudentSQLite3ViewController.h

[](javascript:void(0);)

#import "sqlite3.h"

#define DATA\_FILE @"data.sqlite3"

#define TABLE\_NAME @"student"

#define FIELDS\_NAME\_SID @"studentId"

#define FIELDS\_NAME\_SNAME @"studentName"

#define FIELDS\_NAME\_SCLASS @"studentClass"

@interface ViewController : UIViewController {

sqlite3\* db;

}

@property (retain, nonatomic) IBOutlet UITextField \*studentId;

@property (retain, nonatomic) IBOutlet UITextField \*studentName;

@property (retain, nonatomic) IBOutlet UITextField \*studentClass;

- (IBAction)saveFromSqlite:(id)sender;

- (IBAction)loadFromSqlite:(id)sender;

-(NSString\*)dataFile;

-(IBAction)textFieldDoneEditing:(id)sender;

@end

[](javascript:void(0);)

StudentSQLite3ViewController.m

[](javascript:void(0);)

@synthesize studentId;

@synthesize studentName;

@synthesize studentClass;

-(NSString\*)dataFile {

NSArray\* myPaths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

NSString\* myDocPath = [myPaths objectAtIndex:0];

NSString\* fileName = [myDocPath stringByAppendingFormat:DATA\_FILE];

return fileName;

}

[](javascript:void(0);)

无参数SQLite3处理过程

1、打开数据库sqlite3\_open。

2、创建数据库表和执行SQL语句sqlite3\_exec。

3、释放资源sqlite3\_close。

**创建数据库**

[](javascript:void(0);)

- (void)viewDidLoad {

[super viewDidLoad];

NSString\* fileName = [self dataFile];

NSLog(@"%@", fileName);

if (sqlite3\_open([fileName UTF8String], &db) != SQLITE\_OK) {

sqlite3\_close(db);

NSAssert(NO, @"OPEN SQLITE DATABASE ERROR！");

} else {

char\* error;

NSString\* createSQL = [NSString stringWithFormat:@"CREATE TABLE IF NOT EXISTS %@(%@ TEXT PRIMARY KEY, %@ TEXT, %@% TEXT);",

　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　TABLE\_NAME, FIELDS\_NAME\_SID, FIELDS\_NAME\_SNAME, FIELDS\_NAME\_SCLASS];

if (sqlite3\_exec(db, [createSQL UTF8String], NULL, NULL, &error)) {

sqlite3\_close(db);

NSAssert1(NO, @"CREATE TABLE ERROR", error);

} else {

sqlite3\_close(db);

}

}

}

[](javascript:void(0);)

sqlite3\_open([[self dataFilePath] UTF8String], &db) != SQLITE\_OK sqlite3\_open打开数据库，注意：在sqlite3中的函数都是使用C字符串[self dataFilePath] UTF8String]是将NSString字符串转换为C字符串，&db是sqlite3指针（\* db）的地址。

该函数sqlite3\_open返回SQLITE\_OK打开成功。

sqlite3\_exec(db, [tablesql UTF8String], NULL, NULL, &err) != SQLITE\_OK

sqlite3\_exec是执行任何不带返回值sql语句，第2个参数是要执行的sql语句，第3个参数是要回调函数，第4个参数是要回调函数的参数，第5个参数是执行出错的字符串。

sqlite3\_close(db); 是关闭数据库。

NSAssert是断言函数，当断言失败时候打印信息。

NSAssert1是带有一个参数的NSAssert函数，此外还有NSAssert2等函数。

有参数的SQLite3处理过程

1、打开数据库sqlite3\_open。

2、预处理SQL语句sqlite3\_prepare\_v2。

3、绑定参数sqlite3\_bind\_text。

4、执行语句sqlite3\_step(statement) 。

5、释放资源sqlite3\_finalize࿨sqlite3\_close。

**数据保存**

[](javascript:void(0);)

- (IBAction)saveFromSqlite:(id)sender {

NSString\* fileName = [self dataFile];

NSLog(@"%@", fileName);

if (sqlite3\_open([fileName UTF8String], &db)) {

sqlite3\_close(db);

NSAssert(NO, @"OPEN DATABASE ERROR");

} else {

NSString\* sqlStr = [NSString stringWithFormat:@"INSERT OR REPLACE INTO %@(%@, %@, %@) VALUES(?, ?, ?)",TABLE\_NAME, FIELDS\_NAME\_SID, FIELDS\_NAME\_SNAME, FIELDS\_NAME\_SCLASS];

sqlite3\_stmt\* statement;

//预处理过程

if (sqlite3\_prepare(db, [sqlStr UTF8String], -1, &statement, NULL) == SQLITE\_OK) {

//绑定参数开始

sqlite3\_bind\_text(statement, 1, [studentId.text UTF8String], -1, NULL);

sqlite3\_bind\_text(statement, 2, [studentName.text UTF8String], -1, NULL);

sqlite3\_bind\_text(statement, 3, [studentClass.text UTF8String], -1, NULL);

//执行插入

if (sqlite3\_step(statement) != SQLITE\_DONE) {

NSAssert(0, @"INSERT DATABASE ERROR!");

}

}

sqlite3\_finalize(statement);

sqlite3\_close(db);

}

}

[](javascript:void(0);)

sqlite3\_prepare\_v2(db, [sqlStr UTF8String], -1, &statement, nil) == SQLITE\_OK

sqlite3\_prepare\_v2执行sql语句，第3个参数-1代表全部sql字符串长度，第4个参数&statement是sqlite3\_stmt指针（\* statement）的地址，第5个参数是sql语句没有被执行的部分语句。

sqlite3\_bind\_text(statement, 1, [studentId.text UTF8String], -1, NULL);

是绑定参数，第2个参数为序号（从1开始），第3个参数为字符串值，第4个参数为字符串长度。 第5个参数为一个函数指针，SQLITE3执行完操作后回调此函数，通常用于释放字符串占用的内存。

sqlite3\_step(statement) != SQLITE\_DONE判断是否执行完成sql语句执行。

sqlite3\_finalize(statement)和sqlite3\_close(db)释放资源。

**查询数据**

[](javascript:void(0);)

- (IBAction)loadFromSqlite:(id)sender {

NSString\* fileName = [self dataFile];

NSLog(@"%@", fileName);

if (sqlite3\_open([fileName UTF8String], &db) != SQLITE\_OK) {

sqlite3\_close(db);

NSAssert(NO, @"OPEN DATABASE ERROR!");

} else {

NSString\* sqlStr = [NSString stringWithFormat:@"SELECT %@,%@,%@ FROM %@ WHERE %@=?",

FIELDS\_NAME\_SID, FIELDS\_NAME\_SNAME, FIELDS\_NAME\_SCLASS, TABLE\_NAME, FIELDS\_NAME\_SID];

sqlite3\_stmt\* statement;

//预处理过程

if (sqlite3\_prepare\_v2(db, [sqlStr UTF8String], -1, &statement, NULL) == SQLITE\_OK) {

//绑定参数开始

sqlite3\_bind\_text(statement, 1, "1000", -1, NULL);

//执行

while (sqlite3\_step(statement) == SQLITE\_ROW) {

char\* field1 = (char\*)sqlite3\_column\_text(statement, 0);

NSString\* field1Str = [[NSString alloc]initWithUTF8String:field1];

studentId.text = field1Str;

char\* field2 = (char\*)sqlite3\_column\_text(statement, 1);

NSString\* field2Str = [[NSString alloc]initWithUTF8String:field2];

studentName.text = field2Str;

char\* field3 = (char\*)sqlite3\_column\_text(statement, 2);

NSString\* field3Str = [[NSString alloc]initWithUTF8String:field3];

studentClass.text = field3Str;

[field1Str release];

[field2Str release];

[field3Str release];

}

}

sqlite3\_finalize(statement);

sqlite3\_close(db);

}

}

[](javascript:void(0);)

while (sqlite3\_step(statement) == SQLITE\_ROW) sqlite3\_step(statement) == SQLITE\_ROW单步执行并判断sql语句执行的状态。

char \*field1 = (char \*) sqlite3\_column\_text(statement, 0); sqlite3\_column\_text(statement, 0);取出字段值，第2个参数是列的顺序，序号是从0开始。

NSString \*field1Str = [[NSString alloc] initWithUTF8String: field1];构建NSSting字符串。

其它部分代码

[](javascript:void(0);)

-(IBAction)textFieldDoneEditing:(id)sender {

[sender resignFirstResponder];

}

- (void)viewDidUnload

{

[self setStudentId:nil];

[self setStudentName:nil];

[self setStudentClass:nil];

[super viewDidUnload];

}

- (void)dealloc {

[studentId release];

[studentName release];

[studentClass release];

[super dealloc];

}

[](javascript:void(0);)