数据持久化

1. 简介：

iOS 中数据持久化方式，基本上有以下几种：1.属性列表 2.用户偏好 3.对象归档 4.SQLite3 5.Core Data，下面将分别对这几种的具体实现进行讲解。

2. 关于应用沙盒：

(1) 简介：每个 iOS 应用都有自己的应用沙盒(即文件系统目录)，与其他文件系统隔离，应用必须待在自己的沙盒里，其他应用不能访问该沙盒。

(2) 应用沙盒目录：

1) 应用程序包 app：包含了所有资源文件和可执行文件，上架前经过数字签名，上架后不可修改

2) Documents：保存应用运行时生成的需要持久化的数据，如 sqlite、coredata 数据、游戏存档，不要保存从网络上加载的文件，否者无法上架！

3) Library 目录：这目录下有 Caches 和 Preferences

Preferences：包含应用程序便好文件。不应直接创建偏好设置文件，而应使用 NSUserDefaults 类来设置应用程序的偏好

Caches 目录：用于存放体积大又不需要备份的数据，如图片、mp3 文件、离线地图等，必须提供 cache 目录清理的解决方案

4) tmp 目录：存放临时文件，不会被备份，而且这文件夹下数据可能随时被清除，保存后续不需要使用的文件，系统会自动处理，重启手机 tmp 目录会清空，系统磁盘空间不足时，系统也会自动处理

(3) 常见沙盒目录获取方式

1) 沙盒根目录

```
NSSring *home = NSHomeDirectory();
```

2) Document 目录

方式一：利用沙盒根目录拼接目录名，不建议，由于新版本可能改目录名

方式二：利用 NSSearchPathForDirectoriesInDomains

3) tmp 临时目录

```
NSString *tmp = NSTmporaryDirectory();
```

4) Library/Caches 目录

利用 NSSearchPathForDirectoriesInDomains，第二个参数由 NSDocumenDirecory 改为 NSCacheDirectory 即可

3. 属性列表 plist

(1) 简介：可保存一些可序列化类，方便保存和加载数据。可序列化存储类:NSArray、NSDictionary、NSData、NSString、NSDate、NSNumber。

(2) 缺点：无法将自定义对象序列化到 plist；无法存储其他类，如 UIColor、NSURL 等；目前只能将一小部分对象存储在属性列表中

(3) 示例：

```
@property (strong, nonatomic) IBOutletCollection(UITextField) NSArray *lineFields;

@end

@implementation ViewController_Plist

- (void)viewDidLoad {
    [super viewDidLoad];
    self.title = @"Plist";
    NSString *filePath = [self dataFilePath];
    if([[NSFileManager defaultManager] fileExistsAtPath:filePath]){
        NSArray *array = [NSArray arrayWithContentsOfFile:filePath];
        for(int i = 0; i < [array count]; i++){
            [(UILabel *)_lineFields[i] setText:array[i]];
        }
    }
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(applicationWillResignActive) name:UIApplicationWillResignActiveNotification object:
        nil];
    // Do any additional setup after loading the view.
}

- (NSString *)dataFilePath{
    NSString *paths = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];
    return [paths stringByAppendingPathComponent:@"data.plist"];
}

- (void)applicationWillResignActive{
    NSString *filePath = [self dataFilePath];
    NSArray *array = [_lineFields valueForKey:@"text"];
    [array writeToFile:filePath atomically:YES];
}
```

4. 用户偏好 NSUserDefaults

(1) 简介：本质也是一个 plist 文件，也是只可保存一些可序列化存储类，比较方便读写

(2) 示例：

```
- (void)viewDidLoad {
    [super viewDidLoad];
    self.title = @"NSUserDefault";
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(applicationWillResignActive) name:UIApplicationWillResignActiveNotification object:
        nil];
    // Do any additional setup after loading the view.
}
```

```
- (void)viewDidAppear:(BOOL)animated{
    [super viewDidAppear: animated];
    NSUserDefaults *userDef = [NSUserDefaults standardUserDefaults];
    NSString *string = [userDef valueForKey:@"string"];
    if([string length]){
        UIAlertController *controller = [UIAlertController alertControllerWithTitle:@"提示" message:string preferredStyle:UIAlertControllerStyleAlert];
        UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleCancel handler:^(UIAlertAction * _Nonnull action) {

        }];
        [controller addAction:cancelAction];
        [self presentViewController:controller animated:YES completion:nil];
    }
}

- (void)viewDidDisappear:(BOOL)animated{
    [super viewDidDisappear:animated];
    [[NSNotificationCenter defaultCenter] removeObserver:self];
}

- (void)applicationWillResignActive{
    NSUserDefaults *userDef = [NSUserDefaults standardUserDefaults];
    NSString *string = @"fdasfasfasdfasdf";
    [userDef setValue:string forKey:@"string"];
}
```

5. 模型对象归档 NSKeyedArchiver

(1) 简介：可以将对象存储到文件中(可任意拓展名保存)

(2) 相比 CoreData 不能查询，不适合大型或复杂对象处理，存取速较慢。即使很多方面应用可能受益于 CoreData，但就持久化而言，CoreData 比较复杂。

(3) 示例：

```
@interface MYObject : NSObject<NSCoding, NSCopying>

@property (copy, nonatomic) NSString *string;//字符串

@property (assign, nonatomic) NSInteger number;//基本数据类型

@property (strong, nonatomic) MYObject *child;//对象
```

```objc
@implementation MYObject

- (void)encodeWithCoder:(NSCoder *)aCoder{
    [aCoder encodeInteger:_number forKey:@"number"];
    [aCoder encodeObject:_string forKey:@"string"];
    [aCoder encodeObject:_child forKey:@"child"];
}

- (instancetype)initWithCoder:(NSCoder *)aDecoder{
    if(self = [super init]){
    _number = [aDecoder decodeIntegerForKey:@"number"];
    _string = [aDecoder decodeObjectForKey:@"string"];
    _child = [aDecoder decodeObjectForKey:@"child"];
    }
    return self;
}
```

```objc
- (id)copyWithZone:(NSZone *)zone{
    MYObject *object = [[MYObject alloc] init];
    object.number = _number;
    object.string = [_string copyWithZone:zone];
    object.child = [_child copyWithZone:zone];
    return object;
}
```

```objc
- (void)viewDidLoad {
    [super viewDidLoad];
    self.title = @"NSKeyedArchiver";
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(applicationWillResignActive) name:UIApplicationWillResignActiveNotification object:
        nil];
    // Do any additional setup after loading the view.
}

- (void)viewDidAppear:(BOOL)animated{
    [super viewDidAppear:animated];
    MYObject *object = [NSKeyedUnarchiver unarchiveObjectWithFile:[self datafilePath]];
    if(object){
        UIAlertController *controller = [UIAlertController alertControllerWithTitle:@"提示" message:[NSString stringWithFormat:@"%ld岁的%@有个%ld岁的孩子
            叫%@", object.number, object.string, object.child.number, object.child.string] preferredStyle:UIAlertControllerStyleAlert];
        UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"确定" style:UIAlertActionStyleCancel handler:^(UIAlertAction * _Nonnull action) {

        }];
        [controller addAction:cancelAction];
        [self presentViewController:controller animated:YES completion:nil];
    }
}
```

```
- (void)applicationWillResignActive{
    NSString *dataPath = [self datafilePath];
    MYObject *object = [[MYObject alloc] init];
    MYObject *child = [[MYObject alloc] init];

    child.number = 2;
    child.string = @"嗒吡";

    object.number = 24;
    object.string = @"嘣吡";
    object.child = child;
    [NSKeyedArchiver archiveRootObject:object toFile:dataPath];
}

- (NSString *)datafilePath{
    NSString *path = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) firstObject];
    return [path stringByAppendingPathComponent:@"data.suibian"];
}
```

6. SQLite3 之 FMDB 框架

(1) 简介：sqlite 是一个开源嵌入式关系数据库，可移植性好，易使用，很小，高效可靠。只是不建议直接操作 sqlite 库，而采用一些开源第三发库，如 FMDB，对 SQLite 做了不错的封装

(2) 示例：

```
#import <Foundation/Foundation.h>
#import "FMDB.h"

extern NSString *MYDatabase_T_Person;

@interface MYDatabaseManager : NSObject

@property (strong, nonatomic) FMDatabase *database;

/**
 * 获取管理实例
 *
 */
+ (instancetype)defaultManager;

/**
 * 创建数据表
 *
 * @ param sql 数据库查询语句 eg. [NSString stringWithFormat:@"CREATE TABLE IF NOT EXISTS %@ (ID INTEGER PRIMARY KEY AUTOINCREMENT,
   NAME TEXT, PHONE TEXT)", MYDatabase_T_Person];
 */
- (void)createTableUsingSQL:(NSString *)sql;

/**
 * 数据插入
 *
 * @ param sql 数据库查询语句 eg. [NSString stringWithFormat:@"INSERT INTO %@ (NAME) VALUES (?)", MYDatabase_T_Person]
 * @ param arguments 查询语句参数 eg. @[@"哇哈哈哈哈"]
 */
- (BOOL)insertObjectUsingSQL:(NSString *)sql withArgumentsArray:(NSArray *)arguments;
```

```
/**
* 数据删除
*
* @ param sql 数据库查询语句 eg. [NSString stringWithFormat:@"DELETE FROM %@ WHERE ID=?", MYDatabase_T_Person)]
* @ param arguments 查询语句参数 eg. @[@"7"]
*/
- (BOOL)removeObjectUsintSQL:(NSString *)sql withArgumentsArray:(NSArray *)arguments;

/**
* 数据修改
*
* @ param sql 数据库查询语句 eg. [NSString stringWithFormat:@"UPDATE %@ SET NAME=? WHERE ID=?", MYDatabase_T_Person]
* @ param arguments 查询语句参数 eg. @[@"fawefawegfa", @"5"]
*/
- (BOOL)updateObjectUsingSQL:(NSString *)sql withArgumentsArray:(NSArray *)arguments;

/**
* 数据查询
*
* @ note 记得查询结果FMResultSet操作结束后关闭数据库，提前关闭会导致后续操作出现问题
* @ param sql 数据库查询语句 eg. [NSString stringWithFormat:@"SELECT *FROM %@", MYDatabase_T_Person]
* @ param arguments 查询语句参数 eg. @[]
*/
- (FMResultSet *)queryObjectUsingSQL:(NSString *)sql withArgumentsArray:(NSArray *)arguments;

/**
* 关闭数据库
* @ note 数据库查询结果FMResultSet操作结束后调用
*/
- (void)closeDatabaseAfterQueryDown;
```

```
static MYDatabaseManager *instance;

//数据库
static const NSString *kDatabaseName = @"MYDATABASE";

//数据表
const  NSString *MYDatabase_T_Person = @"T_PERSON";

@interface MYDatabaseManager ()

@end

@implementation MYDatabaseManager

+ (instancetype)defaultManager{
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        instance = [[MYDatabaseManager alloc] init];
    });
    return instance;
}

+ (instancetype)allocWithZone:(struct _NSZone *)zone{
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        instance = [super allocWithZone: zone];
    });
    return instance;
}
```

```
- (FMDatabase *)database{
    if(!_database){
        NSString *databasePath = [self getDatabasePath];
            _database = [FMDatabase databaseWithPath:databasePath];
            if(!_database){
                NSLog(@"数据库初始化失败，原因：%@", _database.lastErrorMessage);
            }
    }
    return _database;
}

#pragma mark Public Methods

- (void)createTableUsingSQL:(NSString *)sql{
    if(!self.database)
        return;

    [_database open];
    BOOL isDown= [_database executeStatements:sql];
    [_database close];
    if(!isDown){
      NSLog(@"数据表创建失败，原因：%@", _database.lastErrorMessage);
    }else{
        //          NSLog(@"数据表创建成功");
    }
}
```

```
- (BOOL)insertObjectUsingSQL:(NSString *)sql withArgumentsArray:(NSArray *)arguments{
    if(!self.database)
        return NO;

    [_database open];
    BOOL isDown = [_database executeUpdate:sql withArgumentsInArray:arguments];
    [_database close];
    if(!isDown){
        NSLog(@"数据插入失败，原因：%@", _database.lastErrorMessage);
        return NO;
    }
    //          NSLog(@"数据插入成功");
    return YES;
}

- (BOOL)removeObjectUsintSQL:(NSString *)sql withArgumentsArray:(NSArray *)arguments{
    if(!self.database)
        return NO;

    [_database open];
    BOOL isDown = [_database executeUpdate:sql withArgumentsInArray:arguments];
    [_database close];
    if(!isDown){
        NSLog(@"数据删除失败，原因：%@", _database.lastErrorMessage);
        return NO;
    }
    //          NSLog(@"数据删除成功");
    return YES;
}
```

```
- (BOOL)updateObjectUsingSQL:(NSString *)sql withArgumentsArray:(NSArray *)arguments{
    if(!self.database)
        return NO;

    [_database open];
    BOOL isDown = [_database executeUpdate:sql withArgumentsInArray:arguments];
    [_database close];
    if(!isDown){
        NSLog(@"数据更新失败，原因：%@", _database.lastErrorMessage);
        return NO;
    }
    //          NSLog(@"数据更新成功");
    return YES;
}

- (FMResultSet *)queryObjectUsingSQL:(NSString *)sql withArgumentsArray:(NSArray *)arguments{
    if(!self.database)
        return nil;

    [_database open];
    FMResultSet *resultSet = [_database executeQuery:sql withArgumentsInArray:arguments];
    return resultSet;
}

- (void)closeDatabaseAfterQueryDown{
    if(!self.database)
        return;

    [_database close];
}
```

```
#pragma mark Private Methods

- (NSString *)getDatabasePath{
    NSString *path = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) firstObject];
    return  [path stringByAppendingPathComponent: [NSString stringWithFormat:@"%@.sqlite", kDatabaseName]];
}
```

```
#import "ViewController_FMDB.h"
#import "MYDatabaseManager.h"
#import "MYObjet.h"

@interface ViewController_FMDB ()

@end

@implementation ViewController_FMDB
```

```
- (void)viewDidLoad {
    [super viewDidLoad];
    self.title = @"FMDB";
    //建表操作
    [[MYDatabaseManager defaultManager] createTableUsingSQL:[NSString stringWithFormat:@"CREATE TABLE IF NOT EXISTS %@ (ID INTEGER
        PRIMARY KEY AUTOINCREMENT, NAME TEXT, PHONE TEXT)", MYDatabase_T_Person]];

    //删除操作
    //      [[MYDatabaseManager defaultManager] removeObjectUsintSQL:[NSString stringWithFormat:@"DELETE FROM %@ WHERE ID=?",
        MYDatabase_T_Person] withArgumentsArray:@[@"2"]];
    [[MYDatabaseManager defaultManager] removeObjectUsintSQL:[NSString stringWithFormat:@"DELETE FROM %@ ", MYDatabase_T_Person]
        withArgumentsArray:@[]];

    //插入操作
        [[MYDatabaseManager defaultManager] insertObjectUsingSQL:[NSString stringWithFormat:@"INSERT INTO %@ (NAME, PHONE) VALUES (?,?)",
            MYDatabase_T_Person] withArgumentsArray:@[@"忘忧草止水", @"13123323395"]];

    //更新操作
    //    [[MYDatabaseManager defaultManager] updateObjectUsingSQL:[NSString stringWithFormat:@"UPDATE %@ SET NAME=? WHERE ID=?",
        MYDatabase_T_Person] withArgumentsArray:@[@"哦买噶", @"1"]];

    //查询操作
    FMResultSet *resultSet = [[MYDatabaseManager defaultManager]  queryObjectUsingSQL:[NSString stringWithFormat:@"SELECT *FROM %@",
        MYDatabase_T_Person] withArgumentsArray:@[]];
    while ( [resultSet next]) {
        MYObjet *person = [[MYObjet alloc] init];
        person.name = [resultSet stringForColumn:@"NAME"];
        person.phoneNumber = [resultSet stringForColumn:@"PHONE"];
        [[[UIAlertView alloc] initWithTitle:@"提示" message:[NSString stringWithFormat:@"来自数据库：%@，电话：%@", person.name, person.phoneNumber]
            delegate:nil cancelButtonTitle:@"确定" otherButtonTitles:nil, nil] show];
    }
    [[MYDatabaseManager defaultManager] closeDatabaseAfterQueryDown];
```

7. CoreData 框架

在 CoreData 之前，创建数据模型的传统方式是创建 NSObject 的子类，并让其遵循 NSCoding 和 NSCopying 协议，以便进行归档。CoreData 不需要创建类，而是先在数据模型编辑器中创建一些实体，然后在代码中为这些实体创建托管对象。实体与托管对象就相当于类与类的实例。

(1) 实体由属性构成，属性分 3 种类型

1) 特性：如同实例变量的作用，用于保存数据

2) 关系：用于定义实体间的关系，一对一或一对多

3) 提取属性：关系的备选方法，如一个 person 对象可以有一个 neighbor 的提取属性，用于查找数据存储中与这个 person 的 HomeAddress 中拥有相同邮编的所有 homeAddress 对象，提取属性也是一种能跨越多个数据存储的关系。

(2) 键-值编码

从托管对象中检索存储在 name 特性中的值，需调用：

```
let name = myManagedObject.valueForKey(“name”)
```

为托管对象属性设置新值，可执行：

```
myManagedObject.setValue(“fdafadsf”,forKey:”name”)
```

(3)在上下文中结合

1) 这些托管对象活动区域在持久存储中。默认情况下，CoreData 应用将支持存储实现为存储在 Documents 的 SQLite 数据库。但 CoreData 框架中的类将完成加载和保存数据的所有操作。你只需要操作对象，内部工作由 CoreData 完成。虽然还支持其他方式如二进制文件、xml 等，在几乎所有情况下，还是应采用默认设置，并使用 SQLite 作为持久存储

2) 除了创建持久化存储外(应用委托中实现)，我们通常不会直接操作持久存储，而使用所谓的托管对象上下文。上下文协调对持久存储的访问，同时保存自上次保存对象以来修改过的的属性信息。上下文还支持通过撤销管理器来注册所有更改，意味着可以撤销或回滚至上一次数据。可将多个上下文指向同一个持久存储，但一般应用只会用一个。

3) 许多核心数据调用都需要 NSManagedObjectContext 作为参数，或需在上下文中执行。除了一些更复杂、多线程的应用外，应用委托中都可只使用 managedObjectContext 属性,它是 xcode 项目模板自动为应用创建的上下文。除托管对象上下文和持久存储协调者外，所提供应用委托还包含一个 NSManagedObjectModel 实例，该类负责运行时加载和表示使用 xcode 中数据模型编辑器创建的数据模型。通常，不需直接与该类进行交互。该类由其他 CoreData 类在后台使用，因此可确定数据模型中定义了哪些实体和属性。只要使用所提供的文件创建数据模型，就完全不需要担心这个类。

(4)创建新的托管对象

创建托管对象新实例使用 NSEntityDescription 类中的 insertNewObjectForEntityForName(_,inManagedObjectContext:)工厂方法。NSEntityDescription 的工作是跟踪在应用的数据模型中定义的实体，并让你创建这些实体的实例。此方法创建并返回一个实例，表示内存中的单个实体的正确属性设置的 NSManagedObject 实例；如果将实体配置为使用 NSManagedObject 的子类为实现,则返回该子类的实例。记住!实体类似于类。实体是对象的描述，用于定义特定的实体拥有哪些属性。

Let thing = NSEntityDescription.insertNewObjectForEntityForName(“” inManagedObjectContext:);

此方法除创建管理对象，还负责把对象插入上下文，返回这个对象。此时对象位于上下文中，但还不是持久存储中的一部分，下一次托管对象上下文的 saveContext 方法调用时，此对象将被添加到持久存储内。

(5)获取托管对象

要从持久存储中获取托管对象，可使用 fetch 请求，这是 CoreData 处理预定义查询的方式。例如：可要求返回所有 eyesColor 为 blue 的 person 对象

首先，创建获取请求后，为其指定一个 NSEntityDescription，指定希望检索的一个或多个对象实体

```
let request = NSFetchRequest();
let entityDescription = NSEntityDescription.entityForName(“Thing”,inManagedObjectContext:);
request.entity = entityDescription;
```

也可以使用 NSPredicate 类为获取请求指定条件。谓词类似 SQL 的 where 语句

```
let pred = NSPredicate(format:”name=%@”, argumentArray:nameString);
request.predicate = pred;
```

表示：仅需获取哪些 name 属性为 name string 值的托管对象

可使用 NSManagedObjectContext 中的实例方法来执行获取请求

```
do{
 let objects = try context.executeFetchRequest(request)
    if object.count == 0
    return;
    for oneObject in objects{
       //对象操作，结束后记得 saveContext()
    }
}catch{
 NSLog(“error”, [])
}
```

(6)CoreData 增删改查

1)添加操作

```
var person:NSManagedObject!=nil
```

```
person =
NSEntityDescription.insertNewObjectForEntityForName(entityName,inMa
nagedObjectContext:context) as NSManagedObject

person,.setValue(value, forKey:key);

context.saveContext();
```

2) 删除操作

```
let request = NSFetchReqeust(entityName:entityName)

do{

 let objects = try context.executeFetchRequest(request)

 if objects.count == 0{
 return
    for oneObject in objects{
       context.deleteObject(oneObject as! NSManagedObject)
    }
    context.saveContext()
 }catch{
    NSLog(“error”,[])
 }
```

3) 修改操作、查询操作

```
letpredic = NSPredicate(format:”name=%@”,”abc”);

let request = NSFetchRequest(entityName:entityName)

request.predicate = predict

do{

 let objects = try context.executeFetchRequest(request)

 if objects.count == 0{
 return
    for oneObject in objects{
```

```
            oneObject.setValue("13123323395",forKey:phoneKey)
        }
        context.saveContext()
    }catch{
        NSLog("error",[])
    }
```

(7) CoreData 管理类封装

```
#import <CoreData/CoreData.h>

#define STORE_NAME @"MYDatabase.sqlite"//数据库名

NS_ASSUME_NONNULL_BEGIN

@interface MYCoreDataManager : NSObject

@property (readonly, strong, nonatomic) NSManagedObjectContext *managedObjectContext;
@property (readonly, strong, nonatomic) NSManagedObjectModel *managedObjectModel;
@property (readonly, strong, nonatomic) NSPersistentStoreCoordinator *persistentStoreCoordinator;

/* 获取单例对象 */
+ (MYCoreDataManager *)defaultManager;

/* 添加操作 */
- (NSManagedObject *)createObjectWithEntity:(NSString *)entityName;

/* 删除操作 */
- (void)deleteObject:(NSManagedObject *)object;
- (void)deleteAllObjects:(NSString *)entityName;

/*
 查询操作
 支持谓词predicate和排序描述器sortDescriptor
*/
- (NSArray *)fetchObjectsWithEntity:(NSString *)entityName
                          predicate:(nullable NSPredicate *)predicate
                               sort:(nullable NSArray<NSSortDescriptor*> *)sortDescriptors;

/* 保存修改 */
- (void)saveContext;
```

```objc
#import "MYCoreDataManager.h"

static MYCoreDataManager *shareManager;

@implementation MYCoreDataManager

@synthesize managedObjectContext = _managedObjectContext;
@synthesize managedObjectModel = _managedObjectModel;
@synthesize persistentStoreCoordinator = _persistentStoreCoordinator;

+ (MYCoreDataManager *)defaultManager{
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        shareManager = [[MYCoreDataManager alloc] init];
    });
    return shareManager;
}

+ (instancetype)allocWithZone:(struct _NSZone *)zone{
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        shareManager = [super allocWithZone: zone];
    });
    return shareManager;
}

- (instancetype)init{
    if(self = [super init]){
        [self managedObjectContext];
    }
    return self;
}
```

```objc
- (NSManagedObjectModel *)managedObjectModel {
    // The managed object model for the application. It is a fatal error for the application not to be able to find and load its model.
    if (_managedObjectModel != nil) {
        return _managedObjectModel;
    }
    NSURL *modelURL = [[NSBundle mainBundle] URLForResource:@"Model" withExtension:@"momd"];
    _managedObjectModel = [[NSManagedObjectModel alloc] initWithContentsOfURL:modelURL];
    return _managedObjectModel;
}

- (NSURL *)applicationDocumentsDirectory {
    // The directory the application uses to store the Core Data store file. This code uses a directory named "com.cwn.Core_Data" in the application's documents directory.
    return [[[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask] lastObject];
}
```

```
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {
    // The persistent store coordinator for the application. This implementation creates and returns a coordinator, having added the store for the application to it.
    if (_persistentStoreCoordinator != nil) {
        return _persistentStoreCoordinator;
    }

    // Create the coordinator and store

    _persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:[self managedObjectModel]];
    NSURL *storeURL = [[self applicationDocumentsDirectory] URLByAppendingPathComponent:STORE_NAME];
    NSError *error = nil;
    NSString *failureReason = @"There was an error creating or loading the application's saved data.";
    if (![_persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeURL options:nil error:&error]) {
        // Report any error we got.
        NSMutableDictionary *dict = [NSMutableDictionary dictionary];
        dict[NSLocalizedDescriptionKey] = @"Failed to initialize the application's saved data";
        dict[NSLocalizedFailureReasonErrorKey] = failureReason;
        dict[NSUnderlyingErrorKey] = error;
        error = [NSError errorWithDomain:@"YOUR_ERROR_DOMAIN" code:9999 userInfo:dict];
        // Replace this with code to handle the error appropriately.
        // abort() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.
        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
        abort();
    }

    return _persistentStoreCoordinator;
}
```

```
- (NSManagedObjectContext *)managedObjectContext {
    // Returns the managed object context for the application (which is already bound to the persistent store coordinator for the application.)
    if (_managedObjectContext != nil) {
        return _managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoordinator];
    if (!coordinator) {
        return nil;
    }
    _managedObjectContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:NSMainQueueConcurrencyType];
    [_managedObjectContext setPersistentStoreCoordinator:coordinator];
    return _managedObjectContext;
}

#pragma mark - public methods

- (NSManagedObject *)createObjectWithEntity:(NSString *)entityName{
    NSManagedObjectContext *context = [self managedObjectContext];
    NSManagedObject *object = [NSEntityDescription insertNewObjectForEntityForName:entityName inManagedObjectContext:context];
    return object;
}
```

```objc
- (void)deleteObject:(NSManagedObject *)object{
    NSManagedObjectContext *context = [self managedObjectContext];
    [context deleteObject:object];
}
- (void)deleteAllObjects:(NSString *)entityName{
    NSArray *arr = [self fetchObjectsWithEntity:entityName predicate:nil sort:nil];
    [arr enumerateObjectsUsingBlock:^(NSManagedObject *obj, NSUInteger idx, BOOL * _Nonnull stop) {
        [self deleteObject:obj];
    }];
    [self saveContext];
}

- (NSArray *)fetchObjectsWithEntity:(NSString *)entityName
                          predicate:(nullable NSPredicate *)predicate
                               sort:(nullable NSArray<NSSortDescriptor*> *)sortDescriptors{
    NSManagedObjectContext *context = [self managedObjectContext];
    NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:entityName];

    if(sortDescriptors != nil){
        [request setSortDescriptors:sortDescriptors];
    }

    if(predicate != nil){
        [request setPredicate:predicate];
    }

    NSArray *fetchedObjects = [context executeFetchRequest:request error:nil];
    if(fetchedObjects == nil){
        return [NSArray array];
    }

    return fetchedObjects;
```

```objc
- (void)saveContext {
    NSManagedObjectContext *managedObjectContext = self.managedObjectContext;
    if (managedObjectContext != nil) {
        NSError *error = nil;
        if ([managedObjectContext hasChanges] && ![managedObjectContext save:&error]) {
            // Replace this implementation with code to handle the error appropriately.
            // abort() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.
            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
            abort();
        }
    }
}
```

(8) CoreData管理类 MYCoreDataManager的使用

1) 项目新建 CoreData 模型

2) 将 MYCoreDataManager.m 中 managedObjectModel 的 modelURL 模型名称 改为对应新建的模型名.momd

3) 打开模型编辑器，新建一实体"Person"，添加特性Attributes："name", age

4) 新建管理对象的子类文件

5) 具体使用示例：

```
Person *object=(Person*)[[MYCoreDataManager defaultManager] createObjectWithEntity:@"Person"];
object.name=@"急救模型水";
object.age=[NSNumber numberWithInteger:24];
[[MYCoreDataManager defaultManager] saveContext];
NSArray *objects=[[MYCoreDataManager defaultManager] fetchObjectsWithEntity:@"Person" predicate:nil sort:nil];
NSEnumerator *enumerator=[objects objectEnumerator];
Person *objc;
while(objc=[enumerator nextObject]){
 NSLog(@"%@",objc.name);
 //[[MYCoreDataManager defaultManager] deleteObject:objc];
}
[[MYCoreDataManager defaultManager] saveContext];
```