

# 一朵西兰花

不求深刻,但求简单.

[🏠 首页](#)[📁 归档](#)

## Core Data, FMDB, Realm 性能对比

📅 Mar 16, 2016 | 🏳️ Hits



iOS 持久化其实也没多少选择. 面试题经常会被问到, 但是无非也就下面几种: 文件写入沙盒, NSUserDefaults, NSKeyedArchiver (归档), SQLite 3, 基于 SQLite 的一些数据库封装. 最近工作需要评估一下 iOS 常用的几种数据库的性能, 所以有了这篇文章.

### 介绍

#### FMDB

直接在 iOS 中使用原生的 SQLite API 时, 需要使用大量的 C 语言. 这样体验并不友好, 所以出现了 FMDB 这样使用 Objective-C 封装的 SQLite API 的数据库框架. 这样使 SQLite 操作更简单易懂. 并且它对于多线程的并发操作进行了处理, 所以是线程安全的.

#### Realm

Realm 是由 Y Combinator 公司孵化的一款支持运行在手机、平板和可穿戴设备上的嵌入式数据库(旨在取代 CoreData 和 SQLite). Realm 并不是基于 CoreData, 也不是基于 SQLite 所构建 ORM 的. 它拥有自己的数据库存储引擎, 可以高效且快速地完成数据库的构建操作. Realm 支持 iOS、OS X (Objective-C 和 Swift) 以及 Android. Realm 文件可以跨平台共享. Realm 还提供了[中文文档](#).

#### Core Data

Core Data 是一个模型层的技术. Core Data 本身既不是数据库也不是数据库访问框架. 相反, Core Data 是一个完整的数据模型解决方案. 我简单理解为对持久层的封装, 使得我们可以通过可视化建立数据模型, 简化数据存取. Core Data 不是一个 ORM, 也不像 FMDB 是一个 SQL wrapper.

### 使用方法

#### FMDB

集成:推荐使用Cocoapods进行安装,在Podfile中添加pod ‘FMDB’ 即可.

FMDB有三个主要的类

- FMDatabase: FMDatabase 对象就代表一个单独的SQLite数据库,用来执行 SQLite 语句.
- FMResultSet: 存放 FMDatabase 执行查询后的结果集.
- FMDatabaseQueue: 用于在多线程中执行多个查询或更新,它是线程安全的.

创建: create database: 在沙盒内创建一个 FMDBModel.sqlite 文件.

```
1 NSString* dbPath = [[NSString stringWithFormat:@"%s", [[NSFileManager defaultManager] URL]
2
3 FMDatabase* database = [[FMDatabase alloc] initWithPath:dbPath];
```

dbPath 的路径有三种情况:

1. 具体文件路径: 如果不存在会自动创建.
2. @"": 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.

查: 查询 ID < 1000 的所有数据

```
1 FMResultSet *resultSet = [db executeQuery:[NSString stringWithFormat:@"SELECT * FROM t_me:
  ◀
```

增,删,改 : 除了 executeQuery(查询) 外所有操作都是 executeUpdate(更新)

```
1 [db executeUpdate: @"INSERT INTO t_messageModel (sender_id, msg_type, sender_avatar, send
  ◀
```

SQLite 本身支持事务处理,FMDB作为sqlite的上层封装也对事务进行了支持.

```
1 [db open];
2 [db beginTransaction];
3
4 BOOL isRollBack = NO;
5 @try {
6     for (FMDBMessageModel *model in array) {
7         BOOL isInsertSucceed = [db executeUpdate:@"INSERT INTO t_messageModel (sender_id,
8                                     model.sender_id,
9                                     model.msg_type,
```

```
10         model.sender_avatar,  
11         model.sender_nickname,  
12         @(model.msg_id),  
13         model.content,  
14         @(model.operate_time),  
15         @(model.create_time),  
16         model.conversation_id];  
17     }  
18 }  
19  
20 @catch (NSEException *exception) {  
21     isRollBack = YES;  
22     [db rollback];  
23 }  
24  
25 @finally {  
26     if (!isRollBack) {  
27         [db commit];  
28     }  
29 }  
30 [db close];
```

## Realm

### 官方文档

集成：推荐使用Cocoapods进行安装，在Podfile中添加pod ‘Realm’ 即可。

### 辅助工具

#### 1、Realm Browser

Realm官方非常贴心的向开发者提供了一个用于查看和编辑 Realm 数据的工具 Realm Browser。

精选

排行榜

类别

已购项目

更新

Q realm brower

Realm Browser

4+

Realm Browser is a viewer and editor for .realm data store files. It allows developers implementing Realm in their apps to easily view and debug the contents of the .realm files their apps have created.

Features:...

版本 0.97.0 中的新功能

Thanks again for using Realm Browser!

...

打开

Realm 网站

Realm Browser 支持

信息

类别: 软件开发工具

更新日期: 2016年01月14日

版本: 0.97.0

价格: 免费

大小: 2.9 MB

家人共享: 可使用

语言: 英语

开发商: Realm ApS

© Realm Inc. 2014-2015

限4岁以上

兼容性: OS X 10.9 或更高版本, 64 位处理器

default

CLASSES

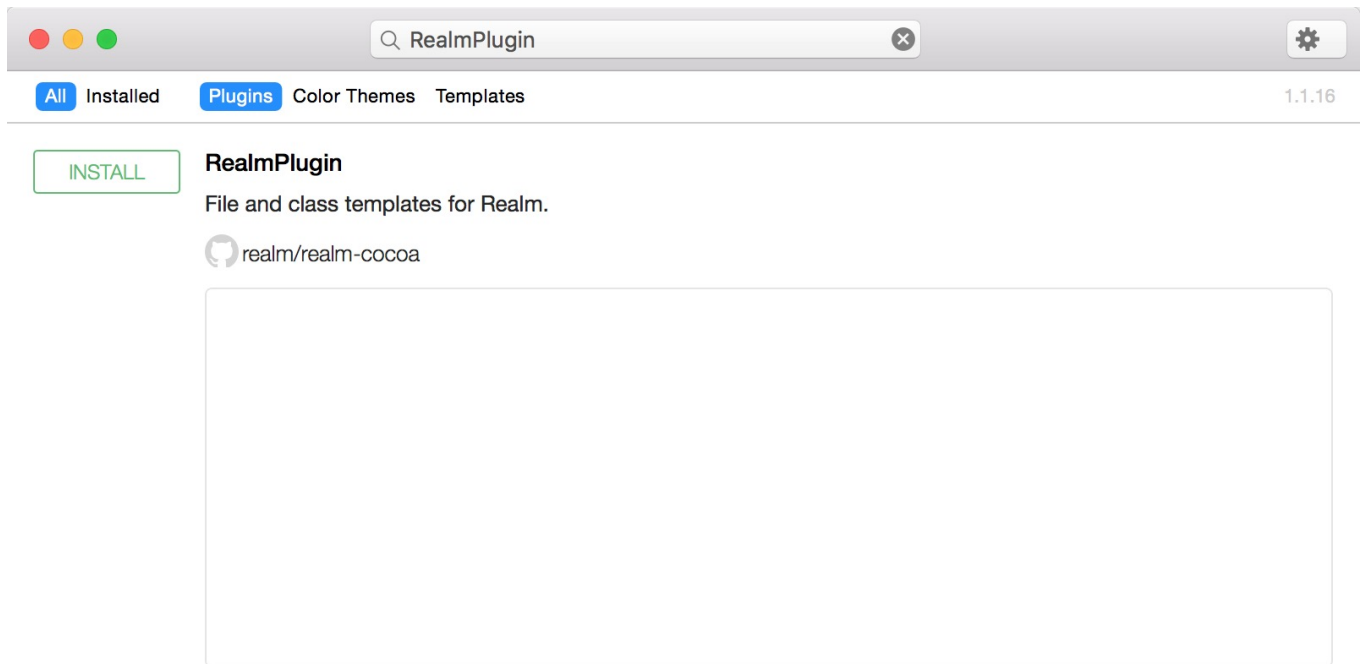
Dog

Owner

name	height	birthdate	vaccinated	owner
String	Float	Date	Boolean	<Owner>
Bilko Fleabaggin	50.000	Sep 8, 2009, 8:26:57 AM	<input checked="" type="checkbox"/>	(Tim)
Deputy Dawg	114.000	Jan 26, 2010, 6:58:03 PM	<input type="checkbox"/>	(JP)
Harry Pawler	114.000	May 13, 2005, 1:36:30 AM	<input checked="" type="checkbox"/>	(Aron)
Jabba the Mutt	98.000	May 16, 2009, 12:41:32 PM	<input type="checkbox"/>	(Joni)
McGruff the Crime Dog	86.000	Sep 14, 2005, 9:38:56 PM	<input checked="" type="checkbox"/>	(Alex)
Nerf Herder	44.000	Feb 23, 2001, 5:34:55 AM	<input type="checkbox"/>	(Michael)
Defense Secretary Waggle	63.000	Jun 8, 2006, 6:03:41 AM	<input checked="" type="checkbox"/>	(Adam)
Salacious B. Crumb	101.000	Jan 18, 2010, 7:51:19 PM	<input checked="" type="checkbox"/>	(Samuel)
Earl Yippington III	44.000	Aug 26, 2001, 9:37:30 PM	<input type="checkbox"/>	(Kristen)
Pickles McPorkchop	71.000	Jan 2, 2008, 6:44:51 PM	<input checked="" type="checkbox"/>	(Emily)
Sir Yaps-a-lot	128.000	Jun 18, 2004, 11:26:05 PM	<input checked="" type="checkbox"/>	(Katsam)
Rudy Loosebooty	77.000	Oct 9, 2002, 6:09:45 AM	<input checked="" type="checkbox"/>	(Morgan)
Madame Barklounder	78.000	Apr 4, 2009, 9:01:06 PM	<input type="checkbox"/>	(Blaine)

2、Xcode Plugin

在Realm中使用到最多的是Realm Model(继承自RLMObject的类，后面有介绍)。官方提供了一个Xcode的插件让我们在创建模型变得非常轻松



直接通过 Alcatraz ，搜索 RealmPlugin 直接安装.

创建：和 FMDB 一样, 有三种方式创建数据库. 一种是存储在默认路径下的数据库, 一种是我们可以自己指定数据库文件的存储路径和只读属性, 另外还可以使用内存数据库.

- 默认Realm数据库

```
1 | RLMRealm *realm = [RLMRealm defaultRealm]
```

- 自定义Realm数据库

```
1 | NSString *docPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomain]
```

```

2   NSString *dbPath = [docPath stringByAppendingPathComponent:@"db/db.realm"];
3   RLMRealm *realm = [RLMRealm realmWithPath:dbPath];
4   // 或者 只读类型
5   RLMRealm *realm = [RLMRealm realmWithPath:dbPath readOnly:YES error:nil];

```

- 内存数据库

```

1   Realm *realm = [RLMRealm inMemoryRealmWithIdentifier:@"memoryDatabase"];

```

## Model

Realm 数据模型是基于标准 Objective - C 类来进行定义的，使用属性来完成模型的具体定义。通过简单的继承 RLMObject 或者一个已经存在的模型类，您就可以创建一个新的 Realm 数据模型对象。Realm 模型对象在形式上基本上与其他 Objective - C 对象相同。

```

1   //主键
2   + (NSString *)primaryKey {
3   }
4   //需要添加索引的属性
5   + (NSArray *)indexedProperties {}
6   //默认属性值
7   + (NSDictionary *)defaultPropertyValues {
8   }
9   //忽略的字段
10  + (NSArray *)ignoredProperties {
11  }

```

在 .m 文件中，Realm 提供了以上方法供对属性进行自定义。

## 数据增删改查

对对象的所有更改（添加，修改和删除）都必须通过写入事务(transaction)完成。

### 1) 增：

```

1   RLMRealm *realm = [RLMRealm defaultRealm];
2   [realm beginWriteTransaction];
3   [realm addObject:model];
4   [realm commitWriteTransaction];

```

### 2) 删

```

1   [realm beginWriteTransaction];

```

```
2 [realm deleteObject:model];
3 // 或者 删除全部
4 [realm deleteAllObjects];
5 [realm commitWriteTransaction];
```

### 3) 改

Realm 提供了一系列用以更新数据的方式，这些方式都有着各自所适应的情景。请选择最符合您当前需求的方式来使用：

#### 3.1 内容直接更新

您可以在写入事务中通过设置某个对象的属性从而完成对象的更新操作。

```
1 [realm beginWriteTransaction];
2 model.name = @"broccoli";
3 [realm commitWriteTransaction];
```

#### 3.2 通过主键更新

如果您的数据模型中设置了主键的话，那么您可以使用 `+ [RLMObject createOrUpdateInRealm:withValue:]` 来更新对象，或者当对象不存在时插入新的对象。

```
1 [realm beginWriteTransaction];
2 [Model createOrUpdateInRealm:realm withValue:model];
3 [realm commitWriteTransaction];
```

#### 3.3 通过 KVC 更新

```
1 RLMResults *models = [RealmMessageModel allObjects];
2 [self.mananger transactionWithBlock:^(
3     [models setValue:@"broccoli" forKeyPath:@"nickname"];
4 )];
```

### 4) 查

#### 4.1 查询全部数据

```
1 RLMResults *models = [RealmModel allObjects];
2 // 或者 指定数据库
3 RLMResults *models = [RealmModel allObjectsInRealm:realm];
```

## 4.2 条件查询

```

1  RLMResults *results = [RealmModel objectsWhere:@"nickname = 'broccoli'"];
2  // 或者使用谓词
3  NSPredicate *predicate = [NSPredicate predicateWithFormat:@"nickname = '%@'", @"broccoli"];
4  RLMResults *results = [RealmModel objectsWithPredicate:predicate];

```

## 4.3 条件排序

筛选出来的结果按照num字段进行递增排序：

```

1  RLMResults *results = [[RealmModel objectsWhere:@"nickname = 'broccoli'"] sortedResultsController];

```

## 4.4 链式查询

在 nickname 是 broccoli 的结果中查询 age 为 23 的结果集。

```

1  RLMResults *nameResults = [RealmModel objectsWhere:@"nickname = 'broccoli'"];
2  RLMResults *results = [nameResults objectsWhere:@"age = 23"];

```

## Core Data

新建一个 Core Data 工程，在 Choose a template for your new project 时，勾选 Use Core Data 。



Choose options for your new project:

Product Name:

Organization Name:

Organization Identifier:

Bundle Identifier:

Language:

Devices:

☒ Use Core Data  
☐ Include Unit Tests  
☐ Include UI Tests

Cancel Previous Next

项目建立后会发现 Frameworks 中已经有了 CoreData.framework 一项，并且还多了一个 \*.xcdatamodeld 文件，该文件定义了数据模型结构，你可以使用XCode内置的可视化建模工具进行构建。点开它你会发现左侧有三项：Entities，Fetch Request、Configurations。Entities：可以简单的将 Entity 理解为数据库中的一张表。点击下方 Add Entity 新增 Entities。表中 Attributes 相当于一张表的列属性，可以设置其数据类型，默认值，最大最小值等，类似数据库可视化建表。Fetch Request 相当于 SQL 查询语句的包装类，用 NSFetchRequest 封装一次数据请求，通过 Fetch Request 将 NSFetchRequest 作为模板存储在被管理对象模型。

因为我们在工程创建时，勾选了 Use Core Data，在 AppDelegate.h 中多了几个 property，method

```
1  #import <UIKit/UIKit.h>
2  #import <CoreData/CoreData.h>
3
4  @interface AppDelegate : UIResponder <UIApplicationDelegate>
5
6  @property (strong, nonatomic) UIWindow *window;
7
8  @property (readonly, strong, nonatomic) NSManagedObjectContext *managedObjectContext;
9  @property (readonly, strong, nonatomic) NSManagedObjectModel *managedObjectModel;
```

```

10  @property (readonly, strong, nonatomic) NSPersistentStoreCoordinator *persistentStoreCoo
11
12  - (void)saveContext;
13  - (NSURL *)applicationDocumentsDirectory;
14
15  @end

```

NSManagedObjectModel 可以把它看做为一个 Core Data 的 schema。生成这个类的来源就是在 xCode 里的.xcdatamodeld文件，我们可以可视化的对这个文件进行操作，实际上这个文件也就相当于数据库的 schema，这个文件编译后就是.momd或.mom文件

NSPersistentStoreCoordinator 是一个位于本地存储文件与缓存层(NSManagedObjectContext)之间的一个持久化层，他是真实操作数据库本地文件。

NSManagedObjectContext 是一个被管理数据的上下文，他实际上是对你所有数据库操作的一个缓存层，把你所有的操作都先缓存起来避免大量磁盘 IO 造成不流畅，你在操作完数据库后调用其save方法，就可以把你的数据库操作提交给持久化层(NSPersistentStoreCoordinator)，由持久化层一次性写入数据库文件。

## NSManagedObjectContext 创建

```

1  - (void)configureContext {
2      _context = [[NSManagedObjectContext alloc] initWithConcurrencyType:NSPrivateQueueConc
3      NSPersistentStoreCoordinator *store = [[NSPersistentStoreCoordinator alloc] initWithM
4
5      NSString *sqlitePath = [[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSI
6
7      NSError *error = nil;
8
9      [store addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:[NSURL file
10
11     if (error) {
12         NSLog(@"%@", error);
13     }
14
15     _context.persistentStoreCoordinator = store;
16 }

```

## 数据增删改查

### 1) 增

```

1  - (void)insert:(CoreDataMessageModel *)model {
2      CoreDataMessageModel *message = [NSEntityDescription insertNewObjectForEntityForName
3
4      message.sender_id = model.sender_id;

```

```

5      message.msg_type = model.msg_type;
6      message.sender_avatar = model.sender_avatar;
7      message.sender_nickname = model.sender_nickname;
8      message.msg_id = model.msg_id;
9      message.content = model.content;
10     message.operate_time = model.operate_time;
11     message.create_time = model.create_time;
12     message.conversation_id = model.conversation_id;
13
14     NSError *error = nil;
15     [_context save:&error];
16
17     if (error) {
18         NSLog(@"%@", error);
19     }
20 }

```

## 2) 删

```

1  - (void)deleteMultiple {
2      NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@"CoreDataMessage"];
3
4      request.predicate = [NSPredicate predicateWithFormat:@"msg_id > %@", @"0"];
5
6      NSArray *resultSet = [_context executeFetchRequest:request error:nil];
7
8      for (CoreDataMessageModel *message in resultSet) {
9          [_context deleteObject:message];
10     }
11
12     [_context save:nil];
13 }

```

## 3) 改

### 改单条数据

```

1  - (void)updateSingle:(CoreDataMessageModel *)model {
2      NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@"CoreDataMessage"];
3      request.predicate = [NSPredicate predicateWithFormat:@"msg_id = %@", model.msg_id];
4
5      NSArray *resultSet = [_context executeFetchRequest:request error:nil];
6
7      for (CoreDataMessageModel *message in resultSet) {
8          message.sender_id = model.sender_id;
9          message.msg_type = model.msg_type;
10         message.sender_avatar = model.sender_avatar;

```

```

11         message.sender_nickname = model.sender_nickname;
12         message.msg_id = model.msg_id;
13         message.content = model.content;
14         message.operate_time = model.operate_time;
15         message.create_time = model.create_time;
16         message.conversation_id = model.conversation_id;
17     }
18
19     [_context save:nil];
20 }

```

## 改多条数据

```

1 - (void)updateMultiple {
2     NSBatchUpdateRequest *batchUpdate = [NSBatchUpdateRequest batchUpdateRequestWithEntityName:@"CoreDataMessage"];
3
4     batchUpdate.propertiesToUpdate = @{@"sender_nickname" : @"broccoli"};
5     batchUpdate.affectedStores = _context.persistentStoreCoordinator.persistentStores;
6
7     batchUpdate.resultType = NSUpdatedObjectsCountResultType;
8
9     @try {
10         [_context executeRequest:batchUpdate error:nil];
11     }
12     @catch (NSException *exception) {
13         NSLog(@"Core Data 批量更新出错: exception %@", exception);
14     }
15     @finally {
16
17     }
18 }

```

## 4) 查

```

1 - (NSArray *)selectSingle {
2     NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@"CoreDataMessage"];
3
4     request.predicate = [NSPredicate predicateWithFormat:@"msg_id < %@", @"10000"];
5
6     return [_context executeFetchRequest:request error:nil];
7 }

```

## 性能比较

测试机: iPhone 6 iOS 系统版本: 9.2.1

循环插入10000次, 每次插入一条数据:

1	FMDB	121634 ms;
2	Realm	58925 ms;
3	CoreData	74987 ms;

批量插入10000条数据:

1	FMDB	410 ms;
2	Realm	457 ms;
3	CoreData	不支持批量操作;

循环更新10000次, 每次更新一条数据:

1	FMDB	5869 ms;
2	Realm	44761 ms;
3	CoreData	86231 ms;

批量更新10000条数据为:

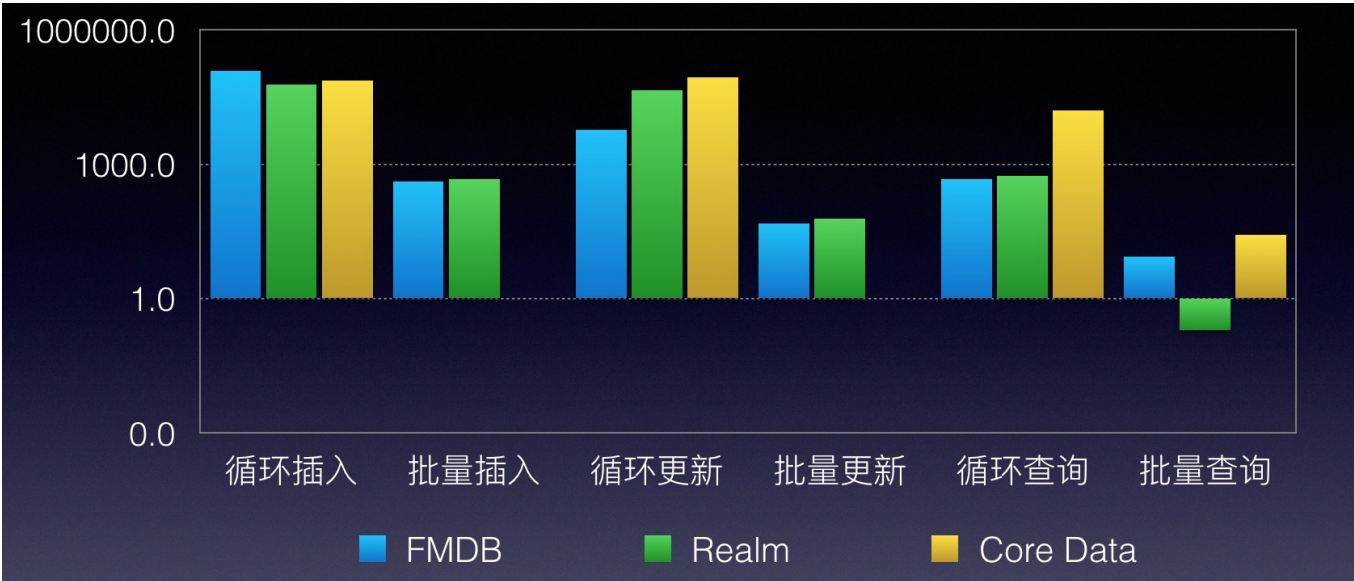
1	FMDB	48 ms;
2	Realm	60 ms;
3	CoreData	90 ms;

循环查询10000条数据, 每次查询一条数据:

1	FMDB	461 ms;
2	Realm	547 ms;
3	CoreData	16072 ms;

批量查询10000条数据:

1	FMDB	8.6 ms;
2	Realm	0.2 ms;
3	CoreData	27 ms;



注释： CoreData 在 iOS 8 才开始支持批量操作（批量更新 (Bench Update) NSBatchUpdateRequest ）

App 使用

多看阅读, 手机淘宝, 手机 QQ, 微博, iOS 中的 News 使用了 FMDB.

IM 的需求

IM 中使用数据库主要的需求是：大量的单次写入一条数据，小批量的读取数据以及小批量的写入.

各自优缺点

FMDB

优点	缺点
对多线程的并发操作进行处理，所以是线程安全的, FMDB是轻量级的框架，使用灵活。	不支持 ORM 需要手写 SQLite 语句

Realm

优点	缺点
入门门槛低, 支持 NSPredicate, 支持 ORM	关联关系弱, 强制内省容错机制导致存储文件不断变大, 没有细粒化通知, 增加包体积


Core Data

优点	缺点
系统自带，官方支持；强大的关联关系；精细化的通知	入门门槛高；不支持批量操作

引用

- yulingtianxia' s blog
- realm 官网
- CoreData VS Realm

 Database

 分享到