持久化

# 概念

所谓的数据持久化，就是将数据保存到硬盘中，使得在应用程序或机器重启后可以继续访问之前保存的数据，既可以减少用户流量消耗，也可以增强APP的体验效果。

# 工具

## 分类

1. 属性列表（plist存储）：
2. 偏好设置（NSUserDefaults）
3. 归档序列化存储
4. CoreData
5. SQLite
6. Keychain
7. YYCache

# 详细分类

## plist文件

系统提供的最简便的key-value本地存储方案，适合比较轻量的数据存储，比如一些业务flag。主要原因还是其底层是用plist文件存储的，在数据量逐步变大后，可能会发生性能问题。

1. 属性列表（plist文件-property list），
2. NSUserDefaults 作为iOS中一种轻量级数据本地化方式，简单易用，经常用于存储一些应用相关属性记录。本质也是plist文件。路径为沙盒Document目录平级的/Library/Prefereces里。具体来说NSUserDefaults是iOS系统提供的一个单例类(iOS提供了若干个单例类)，通过类方法standardUserDefaults可以获取NSUserDefaults单例。
3. NSUserDefaults 只能存储常见的基本类型，不能存储自定义对象。

## SQLite

底层利用到数据的存储方案，比较适用数据量大，有查询，排序等需求的存储场景，缺点就是开发略复杂一些。

1. CoreData
2. SQLite

## 归档序列化存储

### 归档

//1.获取文件路径

NSString \*docPath=[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];

//2、添加储存的文件名

NSString \*path = [docPath stringByAppendingPathComponent:@"data.archiver"];

//3、将一个对象保存到文件中

BOOL flag = [NSKeyedArchiver archiveRootObject:@”name” toFile:path];

这种方式可以对字符串、数字等进行归档，当然也可以对NSArray与NSDictionary进行归档。返回值Flag标志着是否归档成功，YES为成功，NO为失败。

### 解档

//1.获取文件路径

NSString \*docPath=[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];

NSString \*path=[docPath stringByAppendingPathComponent:@"data.archiver"];

NSLog(@"path=%@",path);

//2.从文件中读取对象

[NSKeyedUnarchiver unarchiveObjectWithFile:path]

### 多对象归档

同样是使用NSKeyedArchiver进行归档，不同的是同时归档多个对象，这里我们举例放入了一个CGPoint点、字符串、整数（当然很多类型都可以的，例如UIImage、float等等），使用encodeObject:(nullable id)objv forKey:(NSString \*)key方法进行归档，最后通过writeToFile方法写入文件

//准备数据

CGPoint point = CGPointMake(1.0, 2.0);

NSString \*origin = @"坐标原点";

NSInteger value = 10;

NSString \*docPath=[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];

NSString \*path = [docPath stringByAppendingPathComponent:@"multi.archiver"];

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

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

//对多个对象进行归档

[archvier encodeCGPoint:point forKey:@"kPoint"];

[archvier encodeObject: origin forKey:@"kOrigin"];

[archvier encodeInteger:value forKey:@"kValue"];

[archvier finishEncoding];

[data writeToFile:path atomically:YES];

### 多对象解档

从路径中获得数据构造NSKeyedUnarchiver实例，使用encodeObject: forKey:方法获得文件中的对象。

NSMutableData \*dataR = [[NSMutableData alloc]initWithContentsOfFile:multiHomePath];

NSKeyedUnarchiver \*unarchiver = [[NSKeyedUnarchiver alloc] initForReadingWithData:dateR];

CGPoint pointR = [unarchiver decodeCGPointForKey:@"kPoint"];

NSString \*infoR = [unarchiver decodeObjectForKey:@"kInfo"];

NSInteger valueR = [unarchiver decodeIntegerForKey:@"kValue"];

[unarchiver finishDecoding];

NSLog(@"%f,%f,%@,%d",pointR.x,pointR.y,infoR,valueR);

### 自定义对象归档

自定义类中实现NSCoding协议或NSSecureCoding协议。

1. 如果需要归档的类是某个自定义类的子类时，就需要在归档和解档之前先实现父类的归档和解档方法。即 [super encodeWithCoder:aCoder] 和 [super initWithCoder:aDecoder] 方法。
2. 必须遵循并实现NSCoding协议 。
3. 归档和解归档其中任意对象都需要归档和解归档整个文件 。
4. 归档后的文件是加密的，保存文件的扩展名可以任意指定。

示例：

// MyDog.h

// NSSecureCoding

#import <Foundation/Foundation.h>

@interface MyDog : NSObject<NSSecureCoding>

@property(nonatomic, strong) NSString \*name;

@property(nonatomic, assign) NSInteger age;

@end

//

// MyDog.m

// NSSecureCoding

#import "MyDog.h"

@implementation MyDog

+ (BOOL)supportsSecureCoding {

return YES; //支持加密编码

}

//解码方法

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

if (self = [super init]) {

if (aDecoder) {

\_name = [aDecoder decodeObjectOfClass:[NSString class] forKey:@"name"];

\_age = [aDecoder decodeIntegerForKey:@"age"];

}

}

return self;

}

//编码方法

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

[aCoder encodeObject:\_name forKey:@"name"];

[aCoder encodeInteger:\_age forKey:@"age"];

}

@end

Class 解码的选择NSKeyedUnarchiver 编码的选择NSKeyedArchiver

NSMutableData \*data = [NSMutableData data];

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

[archiver setRequiresSecureCoding:YES];

[archiver encodeObject:customClassObject forKey:NSKeyedArchiveRootObjectKey];

[archiver finishEncoding];

解码

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

[unarchiver setRequiresSecureCoding:YES];

MyDog \*dog = (MyDog \*)unarchiver decodeObjectOfClass:[ MyDog class] forKey:NSKeyedArchiveRootObjectKey];

## Keychain

keychain(钥匙串)存储在iOS系统中,并且恢复iPhone会使keychain的内容也恢复.但是删除App是不会影响keychain.  
不同App之间Keychain是不能相互访问的，但是可以通过配置keychain-sharing来解决。

图形用户界面, 文本, 应用程序

描述已自动生成

首先使用keychain 要先导入 Security.framework 这个系统framework，然后在文件里 import 头文件，由于这个框架是c语言风格的接口API，会让人很疑惑

Keychain里可以存储若干条目（item），每个条目都属于某一个类别（class），以下是常见的几种类别：

1. **kSecClassInternetPassword** 属于该类别的条目往往用来存储上网登录密码，远程服务器密码等
2. **kSecClassGenericPassword**存储一些通用的密码，比如数据库密码，vpn连接的密码等等
3. **kSecClassCertificate**，**kSecClassKey**和**kSecClassIdentity**这三类条目往往用于建立基于证书，秘钥和公钥系统的安全连接。

## YYCache

高性能的线程安全方案, YYCache采取的方案是大于20k的采取直接存储文件，然后在sqlite里面存元信息（比如说文件路径），小于20k的直接存储到sqlite里面。LRU算法，基于存储时长、数量、大小的缓存控制策略等。