iOS数据持久化技术 - By Dorayo

何为数据持久化

能将内存中的`数据模型`转换为`存储模型`，并能在将来需要时将`存储模型`还原为`数据模型`的机制

说明：通俗将也就是将数据保存在非易失性设备中，并且能在需要的时候恢复。苹果中也就是从内存->闪存的过程

---

iOS开发中数据持久化的方法

- Raw File APIs (C语言文件操作、iOS的NSFileManager)
- NSUserDefaults
- 属性列表(PList)
- 对象归档
- SQlite3
- Core Data

通过代码获取沙盒相关路径

```
//获取根目录
    NSString *homePath = NSHomeDirectory();
    NSLog(@"Home目录：%@",homePath);

//获取Documents文件夹目录,第一个参数是说明获取Doucments文件夹目录，第二个参数说明是在当前应用沙盒中获取，所有应用沙盒目录组成一个数组结构的数据存放
    NSArray *docPath = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,NSUserDomainMask, YES);
    NSString *documentsPath = [docPath objectAtIndex:0];
    NSLog(@"Documents目录：%@",documentsPath);

 //获取Cache目录
    NSArray *cacPath = NSSearchPathForDirectoriesInDomains(NSCachesDirectory, NSUserDomainMask, YES);
    NSString *cachePath = [cacPath objectAtIndex:0];
    NSLog(@"Cache目录：%@",cachePath);

 //Library目录
    NSArray *libsPath = NSSearchPathForDirectoriesInDomains(NSLibraryDirectory, NSUserDomainMask, YES);
    NSString *libPath = [libsPath objectAtIndex:0];
    NSLog(@"Library目录：%@",libPath);

 //temp目录
    NSString *tempPath = NSTemporaryDirectory();
    NSLog(@"temp目录：%@",tempPath);
```

Raw File APIs

C语言文件操作

1. 写入文件

```
- (void)saveData
{
    // 文件路径
    const char *filePath = [_path UTF8String];

    // 打开文件
    FILE *fp = fopen(filePath, "w+");
    if (NULL == fp) {
        perror("fopen");
        return;
    }

    // 将_textFiled的内容写入文件
    const char *content = [_textField.text UTF8String];
    long size = _textField.text.length;

    size_t count = fwrite(content, size, 1, fp);

    if (count > 0) {
        NSLog(@"Saved data successfully");
    }

    fclose(fp);
}
```

2. 读取文件

```
- (void)loadData
{
    // 文件路径
    const char *filePath = [_path UTF8String];
    NSLog(@"%s", filePath);

    // 打开文件
    FILE *fp = fopen(filePath, "r");
    if (fp == NULL) {
        perror("fopen");
        return;
    }

    // 读取文件到内存
    char buf[BUFSIZE] = {0};
    fseek(fp, 0, SEEK_END);
    long size = ftell(fp);
    rewind(fp);
    fread(buf, size, 1, fp);

    // 赋值给_textField
    NSString *str = [NSString stringWithUTF8String:buf];
    if (str != NULL && ![str isEqualToString:@""]) {
        _textField.text = str;
    }

    fclose(fp);
}
```

NSFileManager

1. 创建文件夹和文件

```
 // Document目录路径
 NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirect
ory, NSUserDomainMask, YES);
 NSString *documentsDirectory = paths[0];
 NSLog(@"documentsDirectory%@",documentsDirectory);

 // 创建test目录
 NSFileManager *fileManager = [NSFileManager defaultManager];
 NSString *testDirectory = [documentsDirectory stringByAppendingPathCo
mponent:@"test"];
 [fileManager createDirectoryAtPath:testDirectory withIntermediateDire
ctories:YES attributes:nil error:nil];

 // 初始化文件路径
 self.path = [testDirectory stringByAppendingPathComponent:kTmpFileNam
e];
 NSLog(@"path:%@", _path);

 // 创建文件
 NSString *content = nil;
 if (![fileManager fileExistsAtPath:_path]) {
     [fileManager createFileAtPath:_path contents:[content dataUsingEn
coding:NSUTF8StringEncoding] attributes:nil];
 }
```

2. 写入文件

```
 - (void)saveData
 {
     NSLog(@"content to save:%@", _textField.text);
     NSError *error;
     [_textField.text writeToFile:_path atomically:YES encoding:NSUTF8
StringEncoding error:&error];
     if (error) {
         NSLog(@"%@", error);
         return;
     }
     NSLog(@"Save data successfully");
 }
```

3. 读取文件

```
 - (void)loadData
 {
     NSError *error;

     NSString *content = [[NSString alloc] initWithContentsOfFile:_pat
h encoding:NSUTF8StringEncoding error:&error];
     if (error) {
         NSLog(@"%@", error);
         return;
     }
     NSLog(@"content:%@", content);

     _textField.text = content;

 }
```

NSUserDefaults

- 直接使用原始的文件操作API，不管是C语言的还是OC的都不太方便
- Cocoa会为每个app自动创建一个数据库，用来存储App本身的偏好设置，如：开关值，音量值之类的少量信息
- NSUserDefaults使用时用 [NSUserDefaults standardUserDefaults] 接口获取单例对象
- NSUserDefaults本质上是以Key-Value形式存成plist文件，放在App的Library/Preferences目录下
- 这个文件是不安全的，所以千万不要用NSUserDefaults来存储密码之类的敏感信息,`用户名密码应该使用KeyChains来存储`

---

1. 保存数据

```
 - (IBAction)saveConfig:(id)sender {

     NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaul
ts];

     [userDefaults setBool:self.toggle.on forKey:@"toggle"];

     float progress = [self.progressTextField.text floatValue];
     [userDefaults setFloat:progress forKey:@"progress"];
     [userDefaults setObject:self.inputTextField.text forKey:@"input"]
;

     // keeps the in-memory cache in sync with a user's defaults datab
ase
     [userDefaults synchronize];
 }
```

2. 读取数据

```
 - (void)loadConfig
 {
     NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaul
ts];

     self.toggle.on = [userDefaults boolForKey:@"toggle"];
     self.progressView.progress = [userDefaults floatForKey:@"progress
"];
     self.progressTextField.text = [NSString stringWithFormat:@"%.2f",
 self.progressView.progress];
     self.inputTextField.text = [userDefaults stringForKey:@"input"];
 }
```

说明：对NSUserDefaults单例对象的操作，实质上还是对PList文件(Library/Preferences/<Application BundleIdentifier>.plist)的读写，只是Apple帮我们封装好了读写方法。

关于Setting Bundle

- Setting Bundle的概念更多地应该是在App的配置选择上
- Setting Bundle可以给用户提供一个从《设置》应用里去配置应用程序的方式
- 从开发者的角度来看，一般需要频繁修改的配置选项，如游戏的音量和控制选项等最好放到app内部的设置页里，而类似于邮箱应用中的邮件地址和服务器的设置等不需要频繁更改的配置项可以放到Setting Bundle里
- 从《设置》应用中进行设置，实际上是操作iOS配置系统中的应用程序域(Application

Domain)，是持久的

- iOS的配置系统中存在如下一些域，将来查询时严格按照如下列出域的顺序进行查找

Domain	State
NSArgumentDomain	volatile
Application (Identified by the app's identifier)	persistent
NSGlobalDomain	persistent
Languages (Identified by the language names)	volatile
NSRegistrationDomain	volatile

- registerDefaults:方法是在NSRegistrationDomain域上进行配置的，所以仅仅是存在于内存中的，易失的

属性列表(PList)

- NSUserDefaults只能读写Library/Preferences/<Application BundleIdentifier>.plist这个文件
- PList文件是XML格式的，只能存放固定数据格式的对象
- PList文件支持的数据格式有NSString, NSNumber, Boolean, NSDate, NSData, NSArray,和NSDictionary。其中，Boolean格式事实上以[NSNumber numberOfBool:YES/NO];这样的形式表示。NSNumber支持float和int两种格式。

---

1. 创建PList文件

```
 _path = [NSTemporaryDirectory() stringByAppendingString:@"custom.plis
t"];

 NSFileManager *fileManager = [NSFileManager defaultManager];

 if (![fileManager fileExistsAtPath:_path]) {
     // 使用NSMutableDictionary的写文件接口自动创建一个文件
     NSMutableDictionary *dict = [NSMutableDictionary dictionary];
     dict[@"textField"] = @"hello,world!";
     if (![dict writeToFile:_path atomically:YES]) {
         NSLog(@"Error!!!");
     }
 }
```

2. 写入PList文件

```
- (IBAction)saveData:(id)sender {

    NSMutableDictionary *dict = [NSMutableDictionary dictionary];
    dict[@"textField"] = _textField.text;
    if (![dict writeToFile:_path atomically:YES]) {
        NSLog(@"Error!!!");
    }
}
```

3. 读取文件

```
- (void)loadData
{
    NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithCo
ntentsOfFile:_path];
    NSString *content = dict[@"textField"];
    if (content && content.length > 0) {
        _textField.text = content;
    }
}
```

对象归档（NSCoding+NSKeyedArchiver）

- NSUserDefaults和Plist文件支持常用数据类型，但是不支持自定义的数据对象
- Cocoa提供了NSCoding和NSKeyArchiver两个工具类，可以把我们自定义的对象编码成二进制数据流，然后存进文件里面

---

1. NSCoding协议

```
/**
 *  解档
 *
 *  @param aDecoder 解码器
 *
 *  @return 解档之后会生成一个该类的对象
 */
- (id)initWithCoder:(NSCoder *)aDecoder
{
    self = [super init];
    if (self) {
        self.name = [aDecoder decodeObjectForKey:kNameKey];
        self.age = [aDecoder decodeIntForKey:kAgeKey];
        self.studyID = [aDecoder decodeObjectForKey:kStudyIDKey];
    }

    return self;
}

/**
 *  归档
 *
 *  @param aCoder 编码器
 */
- (void)encodeWithCoder:(NSCoder *)aCoder
{
    [aCoder encodeObject:_name forKey:kNameKey];
    [aCoder encodeInt:_age forKey:kAgeKey];
    [aCoder encodeObject:_studyID forKey:kStudyIDKey];
}
```

2. 保存数据

```
- (IBAction)saveData:(id)sender {

    _student = [[YMStudent alloc] init];

    _student.name = _name.text;
    _student.age = [_age.text intValue];
    _student.studyID = _studyID.text;

    NSLog(@"%@", _student);

    if ([NSKeyedArchiver archiveRootObject:_student toFile:_path]) {
        NSLog(@"Archive successfully!");
    }
}
```

3. 读取数据

```
- (void)loadData
{
    NSLog(@"%@", _path);
    NSFileManager *fileManager = [NSFileManager defaultManager];
    if (![fileManager fileExistsAtPath:_path]) {
        NSLog(@"Student.data does not exsist!");
        return;
    }
    _student = [NSKeyedUnarchiver unarchiveObjectWithFile:_path];

    if (_student == nil) {
        NSLog(@"No data!");
        return;
    }

    NSLog(@"Unarchive successfully!");
    _name.text = _student.name;
    _age.text = [@(_student.age) stringValue];
    _studyID.text = _student.studyID;
}
```

Sqlite3

1. 创建数据库引擎单例

```
 + (YMSqlDBManager *)shareInstance
 {
     static dispatch_once_t once;
     dispatch_once(&once, ^{
         _dbManager = [[self alloc] init];
     });
     return _dbManager;
 }
```

2. 打开数据库

```
- (BOOL)openDB
{
    if (_dbSqlite) {
        return YES;
    }

    // 数据库路径
    NSString *dbPath = [[self documentPath] stringByAppendingPathCompo
nent:kDBFileName];


    int result = sqlite3_open([dbPath UTF8String], &_dbSqlite);
    if (result != SQLITE_OK) {
        NSLog(@"%s", sqlite3_errmsg(_dbSqlite));
        return NO;
    }

    NSLog(@"Open database ok");
    return YES;
}
```

3. 关闭数据库

```
- (BOOL)closeDB
{
    if (sqlite3_close(_dbSqlite) != SQLITE_OK) {
        NSLog(@"Close database failed!");
        return NO;
    }
    _dbSqlite = nil;
    return YES;
}
```

4. 创建学生记录表

```
 - (BOOL)createTable
 {
     // 1. 打开数据库
     [self openDB];

     // 2. 编写SQL
     NSString *sql = @"CREATE TABLE IF NOT EXSIST student(number INTEG
ER PRIMARY KEY, name TEXT NOT NULL, age INTEGER, sex TEXT, icon BLOB)"
;

     // 3. 执行SQL语句
     char *errmsg;
     int result = sqlite3_exec(_dbSqlite, [sql UTF8String], NULL, NULL
, &errmsg);
     if (result != SQLITE_OK) {
         NSLog(@"%s", errmsg);
         return NO;
     }

     NSLog(@"Create student table ok");

     // 4. 关闭数据库
     [self closeDB];
     return YES;
 }
```

5. 插入学生记录

```
 - (BOOL)insertStudent:(YMStudentModel *)student
 {
     // 1. 打开数据库
     [self openDB];

     // 2. 编写SQL
     NSString *sql = @"INSERT INTO student(number, name, age, sex, ico
n) VALUES (?, ?, ?, ?, ?)";

     //3. 编译sql语句
     sqlite3_stmt *stmt;
     int result = sqlite3_prepare_v2(_dbSqlite, [sql UTF8String], -1,
&stmt, NULL);
     if (result != SQLITE_OK) {
         NSLog(@"Error: sqlite3_prepare_v2");
         return NO;
     }

     // 4. 绑定sql语句中的参数 即: 替换掉问号
     sqlite3_bind_int(stmt, 1, student.number);
     sqlite3_bind_text(stmt, 2, [student.name UTF8String], -1, NULL);
     sqlite3_bind_int(stmt, 3, student.age);
     sqlite3_bind_text(stmt, 4, [student.sex UTF8String], -1, NULL);
     sqlite3_bind_blob(stmt, 5, [UIImagePNGRepresentation(student.icon
) bytes], (int)UIImagePNGRepresentation(student.icon).length, NULL);


     // 5. 执行
     sqlite3_step(stmt);

     // 6. 释放预编译语句对象
     sqlite3_finalize(stmt);

     // 7. 关闭数据库
     [self closeDB];

     NSLog(@"Insert ok");
     return YES;
 }
```

6. 删除学生记录

```
 - (BOOL)deleteStudentByNumber:(int)number
 {
     // 1. 打开数据库
     [self openDB];

     // 2. 编写SQL
     NSString *sql = @"DELETE FROM student WHERE number = ?";

     //3. 编译sql语句
     sqlite3_stmt *stmt;
     int result = sqlite3_prepare_v2(_dbSqlite, [sql UTF8String], -1,
&stmt, NULL);
     if (result != SQLITE_OK) {
         NSLog(@"Error: sqlite3_prepare_v2");
         return NO;
     }

     // 4. 绑定sql语句中的参数 即：替换掉问好
     sqlite3_bind_int(stmt, 1, number);

     // 5. 执行
     sqlite3_step(stmt);

     // 6. 释放预编译语句对象
     sqlite3_finalize(stmt);

     // 7. 关闭数据库
     [self closeDB];

     NSLog(@"Delete ok");
     return YES;
 }
```

7. 更新学生记录

```
 - (BOOL)updateStudent:(YMStudentModel *)student ByNumber:(int)number
 {
     // 1. 打开数据库
     [self openDB];

     // 2. 编写SQL
     NSString *sql = @"UPDATE student SET name = ?, age = ?, icon = ?
WHERE number = ?";

     //3. 编译sql语句
     sqlite3_stmt *stmt;
     int result = sqlite3_prepare_v2(_dbSqlite, [sql UTF8String], -1,
&stmt, NULL);
     if (result != SQLITE_OK) {
         NSLog(@"Error: sqlite3_prepare_v2");
         return NO;
     }

     // 4. 绑定sql语句中的参数 即：替换掉问好
     sqlite3_bind_text(stmt, 1, [student.name UTF8String], -1, NULL);
     sqlite3_bind_int(stmt, 2, student.age);
     sqlite3_bind_blob(stmt, 3, [UIImagePNGRepresentation(student.icon
) bytes], (int)UIImagePNGRepresentation(student.icon).length, NULL);
     sqlite3_bind_int(stmt, 4, student.number);


     // 5. 执行
     sqlite3_step(stmt);

     // 6. 释放预编译语句对象
     sqlite3_finalize(stmt);

     // 7. 关闭数据库
     [self closeDB];

     NSLog(@"Update ok");
     return YES;
 }
```

8. 查询所有数据

从上一步的step的结果中提取出模型对象

```
- (YMStudentModel *)extractModelFrom:(sqlite3_stmt *)stmt
{
```

```
    // 1 提取学号
    int number = sqlite3_column_int(stmt, 0);

    // 2 提取姓名
    const unsigned char *name = sqlite3_column_text(stmt, 1);

    NSString *nameStr;
    if (name != NULL) {
        nameStr = [NSString stringWithCString:(const char *)name encod
ing:NSUTF8StringEncoding];
    }

    // 3 提取年龄
    int age = sqlite3_column_int(stmt, 2);

    // 4 提取性别
    const unsigned char *sex = sqlite3_column_text(stmt, 3);

    NSString *sexStr;
    if (sex != NULL) {
        sexStr = [NSString stringWithCString:(const char *)sex encodin
g:NSUTF8StringEncoding];
    }

    // 5 提取头像
    const void *iconData = sqlite3_column_blob(stmt, 4);
    int iconLength = sqlite3_column_bytes(stmt, 4);
    UIImage *icon = [UIImage imageWithData:[NSData dataWithBytes:iconD
ata length:iconLength]];

    YMStudentModel *model = [[YMStudentModel alloc] init];
    model.number = number;
    model.name = nameStr;
    model.age = age;
    model.sex = sexStr;
    model.icon = icon;

    return model;
}


- (NSArray *)selectAllData
{
    // 1. 打开数据库
    [self openDB];
```

```
    // 2. 编写SQL
    NSString *sql = @"SELECT * from student";

    // 3. 编译SQL语句
    sqlite3_stmt *stmt;
    int result = sqlite3_prepare_v2(_dbSqlite, [sql UTF8String], -1, &
stmt, NULL);
    if (result != SQLITE_OK) {
        NSLog(@"Error: sqlite3_prepare_v2");
        return nil;
    }

    // 4. 执行SQL语句
    NSMutableArray *array = [NSMutableArray array];
    while (sqlite3_step(stmt) == SQLITE_ROW) {
        YMStudentModel *model;
        // 5. 按列提取执行结果
        model = [self extractModelFrom:stmt];
        if (model) {
            [array addObject:model];
        }
    }

    // 6. 释放预编译语句对象
    sqlite3_finalize(stmt);

    // 7. 关闭数据库
    [self closeDB];

    // 8. 返回结果
    return array;
}
```

9. 根据学号查询指定记录

```
- (YMStudentModel *)selectStudentByNumber:(int)number
{
    // 1. 打开数据库
    [self openDB];

    // 2. 编写SQL
    NSString *sql = @"SELECT * FROM student WHERE number = ?";

    // 3. 编译SQL语句
    sqlite3_stmt *stmt;
    int result = sqlite3_prepare_v2(_dbSqlite, [sql UTF8String], -1,
&stmt, NULL);
    if (result != SQLITE_OK) {
        NSLog(@"Error: sqlite3_prepare_v2");
        return nil;
    }

    // 4. 执行SQL语句
    sqlite3_bind_int(stmt, 1, number);
    YMStudentModel *model;
    if (sqlite3_step(stmt) == SQLITE_ROW) {
        // 5. 按列提取执行结果
        model = [self extractModelFrom:stmt];
    }

    // 6. 释放预编译语句对象
    sqlite3_finalize(stmt);

    // 7. 关闭数据库
    [self closeDB];

    // 8. 返回结果
    return model;
}
```

10. 根据学生姓名查询指定记录

```
- (NSArray *)selectStudentsByName:(NSString *)name
{
    // 1. 打开数据库
    [self openDB];

    // 2. 编写SQL
    NSString *sql = @"SELECT name FROM student WHERE name = ?";

    // 3. 编译SQL语句
    sqlite3_stmt *stmt;
    int result = sqlite3_prepare_v2(_dbSqlite, [sql UTF8String], -1, &
stmt, NULL);
    if (result != SQLITE_OK) {
        NSLog(@"Error: sqlite3_prepare_v2");
        return nil;
    }

    // 4. 执行SQL语句
    sqlite3_bind_text(stmt, 1, [name UTF8String], -1, NULL);

    YMStudentModel *model;
    NSMutableArray *array = [NSMutableArray array];

    while (sqlite3_step(stmt) == SQLITE_ROW) {
        // 5. 按列提取执行结果
        model = [self extractModelFrom:stmt];
        if (model) {
            [array addObject:model];
        }
    }

    // 6. 释放预编译语句对象
    sqlite3_finalize(stmt);

    // 7. 关闭数据库
    [self closeDB];

    // 8. 返回结果
    return array;

}
```