Noveo
Software University

6. Хранение данных

Noveo University — iOS

Александр Горбунов

Сегодня

- Папки приложения
- NSFileManager, NSURL
- NSCoding
- NSUserDefaults
- SQLite

Способы хранения данных

- UserDefaults
- Файлы
  - plist
  - NSCoding
  - БД SQLite
  - кастомные форматы
- CoreData

NSUserDefaults

Наиболее простой способ сохранить / загрузить данные. Используется в основном для хранения настроек приложения и других мелких сущностей. Работает только с ограниченным набором стандартных типов.

```
NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaults];

[userDefaults setInteger:42 forKey:@"MyInteger"];
[userDefaults setObject:@"SomeString" forKey:@"MyString"];
[userDefaults setBool:YES forKey:@"MyFlag"];

[userDefaults synchronize];

BOOL myFlag = [userDefaults boolForKey:@"MyFlag"];
NSInteger myInt = [userDefaults integerForKey:@"MyInteger"];
NSString *myString = [userDefaults objectForKey:@"MyString"];
```

Файловая система iOS

Каждое приложение выполняется в "песочнице", в том числе имеет доступ только к папкам своей песочницы. У каждого приложения есть следующий набор папок:

- *NSDocumentDirectory* — документы (автоматически архивируются, видны пользователю в iTunes)
- *NSLibraryDirectory* — настройки, БД и т.д. (автоматически архивируются, но не видны пользователю)
- *NSCachesDirectory* — файлы, которые гарантированно могут быть скачаны снова (не архивируются и могут быть автоматически удалены системой при нехватке дискового пространства)
- *NSDownloadsDirectory* — не архивируются, но и удаляются системой автоматически
- ...

Файловая система iOS

Получение пути до папки документов приложения:

```
NSArray *docsDirs = NSSearchPathForDirectoriesInDomains(
    NSDocumentDirectory, NSUserDomainMask, YES);

NSString *docsDir = docsDirs.firstObject;
```

Ресурсы приложения

Ресурсы приложения хранятся отдельно от остальных данных и доступны только для чтения.

По умолчанию все ресурсы приложения попадают в одну папку, и должны иметь разные имена. Можно явно указать какие папки проекта должны сохранять структуру при копировании в ресурсы.

Ресурсы приложения

Получение пути к ресурсу по имени файла:

```
[[NSBundle mainBundle] pathForResource:@"filename" ofType:@"ext"];
```

NSFileManager

`NSFileManager` позволяет получать список, размеры и атрибуты, проверять наличие по заданному пути, создавать, переносить и удалять файлы и папки, создавать и удалять символические ссылки, а также управлять хранением бэкапов файлов в iCloud.

NSFileManager

Создаём директорию и получаем новый список файлов и каталогов в домашней директории:

```
NSArray *docsDirs = NSSearchPathForDirectoriesInDomains(
    NSDocumentDirectory, NSUserDomainMask, YES);
NSString *docsDir = docsDirs.firstObject;
NSString *path = [docsDir stringByAppendingPathComponent:@"MyDir"];

NSError *error = nil;

[[NSFileManager defaultManager] createDirectoryAtPath:path
    withIntermediateDirectories:YES attributes:nil error:&error];

NSArray *contents = [[NSFileManager defaultManager]
    contentsOfDirectoryAtPath:docsDir error:&error];
```

Встроенные механизмы сериализации

Сохранение и загрузка строки в / из файла:

```
[@"Awesome string" writeToFile:filepath atomically:YES
    encoding:NSUTF8StringEncoding error:&error];
```

```
NSString *string = [NSString stringWithContentsOfFile:filepath
    encoding:NSUTF8StringEncoding error:&error];
```

Встроенные механизмы сериализации

Сохранение и загрузка массива в / из файла:

```
[@[@2, @3, @4] writeToFile:filepath atomically:YES];
```

```
NSArray *array = [NSArray arrayWithContentsOfFile:filepath];
```

Встроенные механизмы сериализации

Сохранение и загрузка словаря в / из файла:

```
[@{@"2": @2, @"3": @3} writeToFile:filepath atomically:YES];
```

```
NSDictionary *dictionary = [NSDictionary dictionaryWithContentsOfFile:filepath];
```

Формат PList

PList (Property List) — стандартный для iOS и OS X формат хранения простых типов данных, поддерживающий следующие типы:

- `NSString` — UTF-8 строка
- `NSNumber` — строка с десятичным числом
- `NSNumber` — булево значение `<true/>` или `<false/>`
- `NSDate` — ISO 8601 представление
- `NSData` — Base64 представление
- `NSArray` — `<array>`, может содержать любое количество других объектов
- `NSDictionary` — `<dict>`, чередование `<key>` и других объектов

Формат PList

На самом деле это обычный XML:

```
<plist version="1.0">
<dict>
	<key>CFBundleDevelopmentRegion</key>
	<string>en</string>

	<key>LSRequiresIPhoneOS</key>
	<true/>

	<key>UIRequiredDeviceCapabilities</key>
	<array>
		<string>armv7</string>
	</array>
</dict>
</plist>
```

NSCoding

`NSCoding` — протокол, абстрагирующий возможность сериализации и десериализации любых объектов.

`NSCoder` — абстрактный класс, представляющий набор методов для сериализации / десериализации объектов.

`NSKeyedArchiver` / `NSKeyedUnarchiver` — конкретные реализации NSCoder, отвечающие за сериализацию / десериализацию полей объекта по ключам и предоставляющие возможность использовать файл как хранилище.

NSCoding

Пример сериализации/десериализации коллекции объектов:

```
@interface Department ()
@property (nonatomic, copy) NSArray *people;
@end

static NSString *peopleFilename() { return /*...*/; }

@implementation Department

- (void)loadPeople
{
    self.people = [NSKeyedUnarchiver unarchiveObjectWithFile:peopleFilename()];
}

- (void)savePeople
{
    [NSKeyedArchiver archiveRootObject:self.people
        toFile:peopleFilename()];
}

@end
```

NSCoding

Пример реализации протокола NSCoding:

```
@interface Person : NSObject <NSCoding>
@property (nonatomic, copy) NSString *name;
@property (nonatomic, assign) NSInteger age;
@end

@implementation Person
- (void)encodeWithCoder:(NSCoder *)coder
{
    [coder encodeObject:self.name forKey:@"name"];
    [coder encodeInteger:self.age forKey:@"age"];
}

- (instancetype)initWithCoder:(NSCoder *)decoder
{
    if ((self = [self init])) {
        _name = [decoder decodeObjectForKey:@"name"];
        _age = [decoder decodeIntegerForKey:@"age"];
    }
    return self;
}
@end
```

NSCoding

Только программист отвечает за соответствие ключей. Если в очередной версии программы появится новое поле, и мы попытаемся загрузить новый объект из старого файла, возникнет исключение.

NSCoding

Как избежать ошибок при загрузке объекта:

- Проверять наличие соответствующего поля методом

```
- (BOOL)containsValueForKey:(NSString *)key
```

- Не использовать строковые литералы для ключей, а использовать например следующую обёртку, позволяющую компилятору находить ошибки в ключах:

```
#define STR_PROP( prop ) NSStringFromSelector(@selector(prop))
[coder encodeObject:self.someProp forKey:STR_PROP(someProp)];
```

- Хранить номер версии формата объекта и перед загрузкой остальных полей выбирать десериализатор соответствующей версии.

DB vs NSCoding

- БД хранит записи, а не объекты.
- При работе с БД объём вспомогательного кода больше.
- При работе с БД за надёжность и производительность отвечает отдельный компонент.
- БД хорошо подходит для работы с большим количеством однородных данных.
- БД включает продвинутые возможности поиска, фильтров и других "хитрых" запросов.

SQLite

SQLite — реляционная БД, ограниченно поддерживающая SQL.

- Популярный стандарт с open-source реализацией. Написан на C. Поддерживается практически на всех платформах.
- Очень надёжная реализация (объём тестов в >1000 раз больше самой реализации).
- Богатая документация.
- Стандартно поддерживается в iOS SDK.
- Упрощённая схема общения между клиентским кодом и движком БД (движок встраивается прямо в программу). Это упрощает код и ускоряет общение с БД.
- Упрощённая схема физического хранения данных (всё в одном блокируемом файле).
- Тип столбца не определяет тип хранимого значения, то есть в любой столбец можно занести любое значение.

SQLite

```
NSString *databasePath = ...;
char *dbPath = [databasePath UTF8String];
sqlite3 *contactDB;
sqlite3_open(dbPath, &contactDB);

char *queryStatement = [@"SELECT name FROM people WHERE age=30" UTF8String];
sqlite3_stmt *statement;
sqlite3_prepare_v2(contactDB, queryStatement, -1, &statement, NULL);

while (sqlite3_step(statement) == SQLITE_ROW) {
    NSString *name = [[NSString alloc] initWithUTF8String:
        (const char *)sqlite3_column_text(statement, 0)];

    // Обработка полученного значения name.
}

sqlite3_finalize(statement);
sqlite3_close(_contactDB);
```

FMDB

- Сторонняя библиотека с открытым исходным кодом.
- Обёртка поверх SQLite.
- Предоставляет API для SQLite на Objective-C (не C).
- Удобное приведение данных к нужному типу.
- Удобное асинхронное выполнение транзакций в блоках.

FMDB

Пример запроса (создание таблицы):

```
NSString *dbPath = ...;

NSString *createTableQuery = @"CREATE TABLE people (id INTEGER  PRIMARY KEY DEFAULT NUL
    TEXT DEFAULT NULL, age INTEGER DEFAULT NULL)";

FMDatabase *database = [FMDatabase databaseWithPath:dbPath];
[database open];

[database executeUpdate:createTableQuery];

[database close];
```

FMDB

Пример запроса (вставка нескольких строк):

```
FMDatabaseQueue *queue = [[FMDatabaseQueue alloc] initWithPath:dbPath];

[queue inDatabase:^(FMDatabase *db) {
    [db beginTransaction];

    for (Person *person in people) {
        [db executeUpdate:@"INSERT INTO people (name, age) VALUES (?, ?)",
            person.name, @(person.age)];
    }

    [db commit];
}];
```

FMDB

Пример запроса (чтение нескольких строк):

```
FMDatabaseQueue *queue = [[FMDatabaseQueue alloc] initWithPath:dbPath];

[queue inDatabase:^(FMDatabase *db) {
        FMResultSet *results = [db executeQuery:
            @"SELECT * FROM people WHERE age=?", @30];

        while ([results next]) {
            NSString *name = [results stringForColumn:@"name"];
            int age = [results intForColumn:@"age"];
            //...
        }
    }];
```

CoreData

Системный фреймворк для работы с *объектной* базой данных.

Низкоуровневым контейнером для CoreData выступает как правило SQLite-файл, хотя в общем случае можно реализовать любое кастомное физическое хранилище.

Характерная особенность CoreData — автоматизированная подгрузка и выгрузка данных при обращении к ним.

CoreData

Когда стоит задуматься об использовании CoreData:

- Необходимо поддерживать хранение нескольких состояний объекта (например отмена действий).
- Необходимо хранить объекты, находящиеся в очень сложных отношениях.
- Необходимо упростить ту или иную рутину — миграция модели, ленивая загрузка, глубокое удаление ...
- CoreData хорошо ложится на окружающую инфраструктуру (например синхронизация через iColud).

CoreData

Как правило на CoreData смотрят как на последний вариант.

- Очень сложно в целом.
- Неудобное API.
- Сложности с многопоточностью.
- **Очень** медленно при большом количестве объектов или большом количестве одновременных операций.
- Данные хранятся в некрасивой автогенерированной БД.