Objective-C

Курс лекций и семинаров для студентов,
желающих научиться программировать под iPhone

Осень-Зима 2013

Лекция №11

Data persistence: SQLite, Core Data, User Defaults

Автор: Дмитрий Волков, iPhone Developer, Sibers

Sibers®

Что вы узнаете сегодня?

- Data persistence: Sandbox, NSFileManager, NSUserDefaults, NSCoding
- NSCoding: Implementation and Usage
- SQLite
- CoreData: NSManagedObject and Fetch requests

Data persistence

Скорее всего, ваше iOS-приложение будет сохранять какие-либо данные между своими запусками:

- Настройки приложения
- Логин / пароль и еще какие-либо данные из API
- Медиа-ресурсы (изображения, аудио, видео)
- Состояние UI
- И многое другое

Для хранения таких данных используется накопитель данных устройства (в нашем случае — flash-память), позволяющая сохранять и записывать данные.

Data persistence

iOS предоставляет как возможность доступа к файловой системе и сохранения данные на диск, так и работу с базами данных.

В общем случае, доступны следующие типы хранилищ:

- NSUserDefaults
- Keychain
- SQLite
- CoreData
- NSFileManager / NSFileHandle

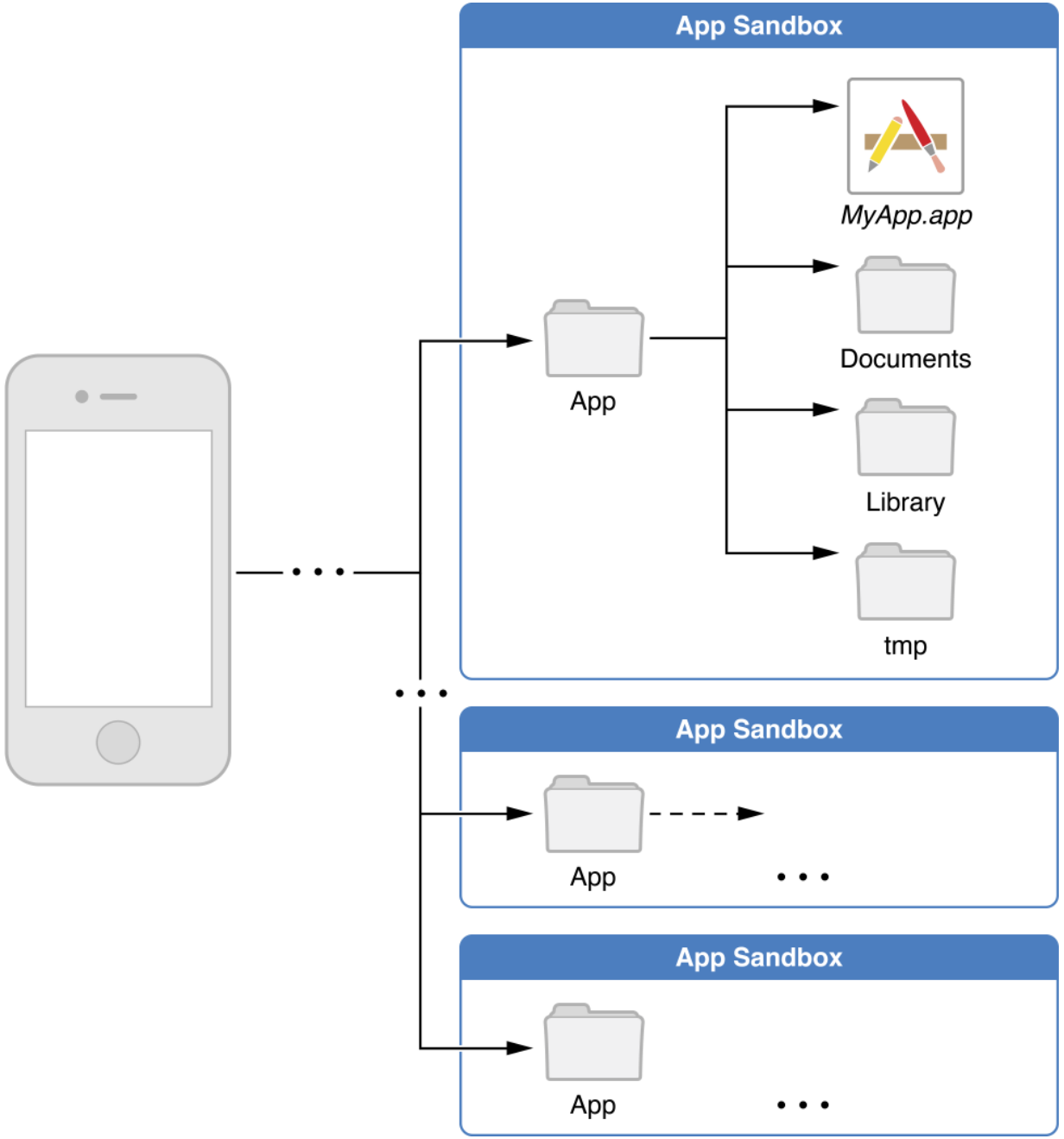
Data persistence: Sandbox

ВАЖНО: Каждое iOS-приложение работает в своей sandbox-директории, к которой вы и имеете доступ. Вы не можете получать доступа к другим частям файловой системы.

Вы имеете доступ к следующим директориям:

- **Documents**
- **Library**
- **tmp**

Директория, в которой находятся ресурсы, которые вы добавили в Ваш проект, .xib файлы, исполняемый файл приложения и другие ресурсы программы — главный «бандл» приложения(**NSBundle**). ***Read only***.


App Sandbox
MyApp.app
Documents
App
Library
tmp
App Sandbox
App
App Sandbox
App

Data persistence: NSFileManager

- Одним из простейших подходов хранения данных является их сохранение в файлы.
- Для работы с файловой системой и управления файлами и директориями, используется класс **NSFileManager**.

NSFileManager — класс-синглтон и несинглтон, позволяет проводить базовые операции с файлами:

- Копирование, удаление, создание, переименование
- Получение и изменение атрибутов (дата создания / изменения и тд.)
- Получение иерархии файловой системы

Data persistence: NSFileManager

```
NSFileManager* sharedFM = [NSFileManager defaultManager];

    NSArray* possibleURLs = [sharedFM URLsForDirectory:NSDocumentDirectory
inDomains:NSUserDomainMask];

NSURL* directoryURL = [possibleURLS firstObject];


NSArray* paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                                     NSUSerDomainMask,
                                                     YES);

NSString* docDirPath = [paths firstObject];
```

Data persistence: NSFileManager

```
NSFileManager* fm = [NSFileManager defaultManager];
NSString* newFilePath = …;
NSString* copiedFilePath = …;
NSData* contents = …;

[fm createFileAtPath:newFilePath contents:contents attributes:nil];
[fm copyFileAtPath:newFilePath toPath:copiedFilePath error:nil];
[fm removeFileAtPath:newFilePath error:nil];
```

Data persistence: NSUserDefaults

- **NSUserDefaults** — просто key-value хранилище данных, доступное из приложения.
- Данные не кодируются и хранятся в простом файле в Library/Preferences/com.applicationbundle.name
- Используется для хранения каких-либо настроек приложения или простых данных пользователя.

Data Persistence: NSUserDefaults

```
NSUserDefaults* defs = [NSUserDefaults standartUserDefaults];

NSString* username = @”User”;
[defs setObject:username forKey:@”com.app.username”];
username = [defs objectForKey:@”com.app.username”];

NSData* data = …;
[defs setObject:data forKey:@”com.app.data”];
[defs setBOOL:YES forKey:@”com.app.musicEnabled”];
[defs setFloat:0.5 forKey:@”com.app.musicVolume”];
```

Data persistence: NSCoding

- Для сохранения объектов, которые вы создаете в приложении, используется протокол **NSCoding**.
- Объекты, которые требуется сериализовать, реализовывают протокол NSCoding.
- После этого, эти объекты можно преобразовывать в тип **NSData** с помощью класса **NSKeyedArchiver** и работать с ними как с набором байт.
- Многие системные классы реализуют этот протокол.
- Коллекции также реализуют его, и при сериализации коллекции сериализуются так же все объекты, которые она содержит.

NSCoding: Implementation

```
@interface SerializedObject : NSObject<NSCoding>
@property(nonatomic) NSInteger count;
@property(nonatomic, copy) NSString* name;
@property(nonatomic, copy) NSNumber* number;
@end
```

NSCoding: Implementation

```
@implementation SerializedObject
- (void) encodeWithCoder:(NSCoder*) coder
{
    [coder encodeObject:self.name forKey:@”name”];
    [coder encodeInteger:self.count forKey:@”count”];
    [coder encodeObject:self.number forKey:@”number”];
}
```

NSCoding: Implementation

```
- (id) initWithCoder:(NSCoder*) coder
{
    self = [self init];
    self.name = [coder decodeObjectForKey:@"name"];
    self.count = [coder decodeIntegerForKey:@"count"];
    self.number = [coder decodeObjectForKey:@"number"];
    return self;
}

@end
```

NSCoding: Usage

```
SerializedObject* object = [SerializedObject new];

Object.name = @"object";

Object.count = 10;

Object.number = @(58.8);

NSData* objectData = [NSKeyedArchiver
archivedDataWithRootObject:object];

[NSUserDefaults standartUserDefaults] setObject:objectData
forKey:@"object"];

SerializedObject* deserialized = [NSKeyedUnarchiver
unarchiveObjectWithData:objectData];
```

SQLite

SQLite — упрощенная open-source реляционная база данных для встраиваемых систем.

- В отличие от десктопных БД (MySQL, MSSQL), SQLite не является отдельным процессом, а поставляется в виде подключаемой библиотеки.
- База данных SQLite — часть вашей программы.
- Все общение с базой (не только выполнение запросов) происходит через C-based API.
- Все данные хранятся в одном файле (таблицы, записи, индексы)

SQLite

Так же на Mac OS X поставляется консольная утилита sqlite3, позволяющая создавать таблицы и выполнять запросы к выбранному файлу БД.

SQLite обладает следующими ограничениями:

- Отсутствуют views
- Отсутствуют хранимые процедуры на уровне БД
- Отсутствуют курсоры
- Многопоточность — несколько читателей, один писатель.
- Нет возможности изменять поля таблиц.

SQLite: Data types

Поддерживаемые типы данных:

- Целые числа
- Вещественные числа
- Текст
- Бинарные данные

SQLite поддерживает «динамические» типы данных, например, в целочисленное поле БД можно записывать текст.

На iOS / Mac OS X используется C API для работы с SQLite.

http://www.sqlite.org

SQLite

```
NSString* dbPath = …;
sqlite3* database;
if(sqlite3_open([dbPath UTF8String], &database)!=SQLITE_OK)
{
    NSLog(@”failed to open DB at path %@, error: %s”, dbPath, sqlilte3_errmsg(database));
    sqlite3_close(database);
}
```

SQLite

```
sqlite3* database;
sqlite3_stmt* statement;
const char* query = “SELECT id, name FROM students”;
sqlite3_prepare_v2(database, query, -1, &statement, NULL);
while(sqlite3_step(statement)==SQLITE_ROW)
{
    int studentID = sqlite3_column_int(statement, 0);
    char* studentName = sqlite3_column_text(statement, 1);
}
sqlite3_finalize(statement);
```

CoreData

CoreData — фреймворк Apple, доступный в Mac OS X с 10.4 и в iOS с 3.0.

CoreData — фреймворк для управления и хранения объектов и связей между ними.

- Использует различные конечные форматы для хранения данных — XML, бинарные файлы, SQLite таблицы.
- Предоставляет высокоуровневый интерфейс для работы с объектами, выборки объектов из хранилища данных и установления отношений между ними.
- CoreData генерирует SQLite запросы, управляет временем жизни своих объектов, минимизирует потребление памяти.

CoreData

- При работе с CoreData, вы моделируете свои объекты, назначая им атрибуты (будущие поля в БД), настройки валидации этих полей, тип данных поля
- Добавляете отношения между объектами: один-к-одному, один-ко-многим (определяете тип связи между различными объектами)
- Создаете «Fetch Requests» — запрос на фильтрацию данных, который выполняет CoreData.
- Для «Fetch Requests» используются объекты класса NSPredicate, которые и описывают точное выражение, которое следует применить к каждой записи для фильтрации.

CoreData

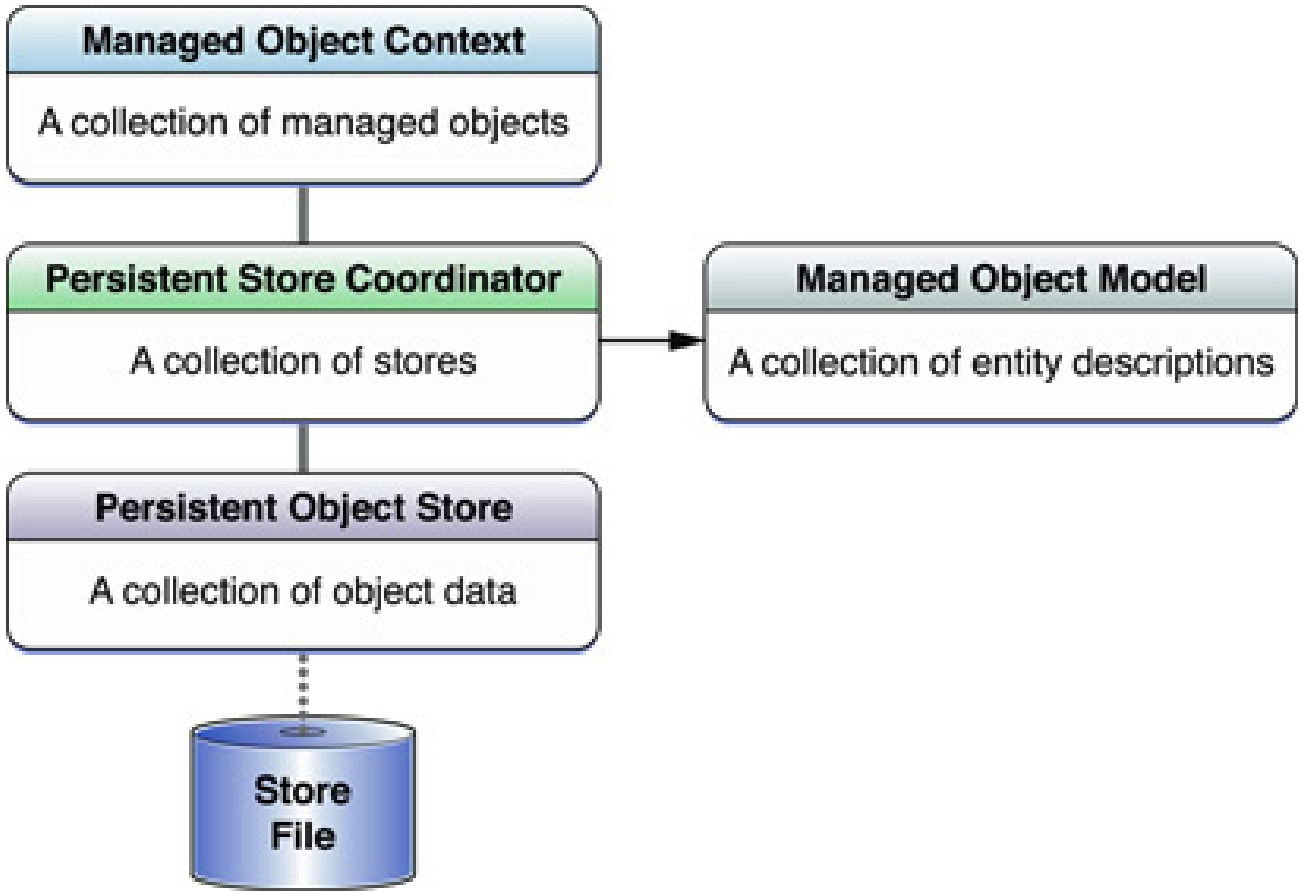
CoreData предоставляет довольно обширную иерархию классов, основными из которых являются:

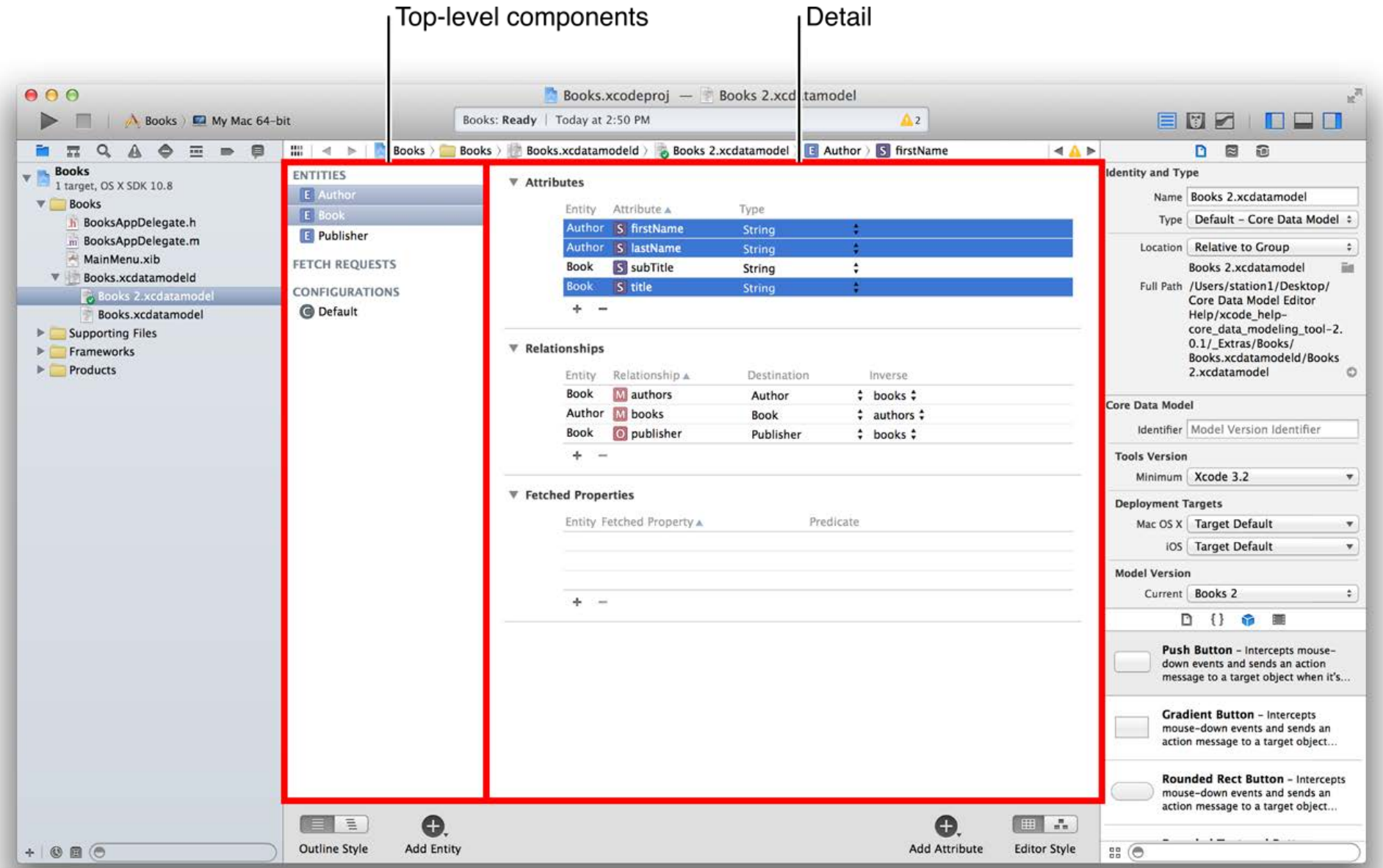
- **NSPersistentStore** — абстрактный базовый класс, описывающий конечное хранилище объектов (база данных sqlite, XML-файл и т.д.)
- **NSManagedObjectModel** — описывает схему вашей БД. Содержит описание объектов, их связей, заранее подготовленные fetch requests.
- **NSPersistentStoreCoordinator** — класс, связывающий описание модели данных с их конечным представлением в хранилище данных.

CoreData

- **NSManagedObject** – модельный объект, описывающий одну запись из базы данных.
- **NSManagedObjectContext** – объект, управляющий набором объектов типа NSManagedObject. Управляет их временем жизни, памятью, а так же передает запросы на сохранение ипредоставление объектов в NSPersistentStoreCoordinator.
- **NSFetchRequest** – запрос на получение объектов. Использует NSPredicate для описания запроса. Возвращает результаты запроса.
- **NSPredicate** – предикат, описывающий запрос.

CoreData




Top-level components
Detail
Books.xcodeproj — Books 2.xcdatamodel
Books: Ready | Today at 2:50 PM
Books
1 target, OS X SDK 10.8
Books
BooksAppDelegate.h
BooksAppDelegate.m
MainMenu.xib
Books.xcdatamodeld
Books 2.xcdatamodel
Books.xcdatamodel
Supporting Files
Frameworks
Products
ENTITIES
Author
Book
Publisher
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Entity Attribute Type
Author firstName String
Author lastName String
Book subTitle String
Book title String
Relationships
Entity Relationship Destination Inverse
Book authors Author books
Author books Book authors
Book publisher Publisher books
Fetched Properties
Entity Fetched Property Predicate
Identity and Type
Name Books 2.xcdatamodel
Type Default - Core Data Model
Location Relative to Group
Books 2.xcdatamodel
Full Path /Users/station1/Desktop/Core Data Model Editor Help/xcode_help-core_data_modeling_tool-2.0.1/_Extras/Books/Books.xcdatamodeld/Books 2.xcdatamodel
Core Data Model
Identifier Model Version Identifier
Tools Version
Minimum Xcode 3.2
Deployment Targets
Mac OS X Target Default
iOS Target Default
Model Version
Current Books 2
Push Button - Intercepts mouse-down events and sends an action message to a target object when it's...
Gradient Button - Intercepts mouse-down events and sends an action message to a target object...
Rounded Rect Button - Intercepts mouse-down events and sends an action message to a target object...
Outline Style
Add Entity
Add Attribute
Editor Style

CoreData: NSManagedObject

```
@interface Author : NSManagedObject
@property(nonatomic, strong) NSString* firstName;
@property(nonatomic, strong) NSString* lastName;
@property(nonatmic, strong) Publisher* publisher;
 @end
@interface Author(OneToManyAcessorMethods)
@property(nonatomic, strong) NSSet* books;
- (void) addBookObject:(Book*) book;
- (void) removeBookObject:(Book*) book;
- (void) addBookObjects:(NSSet*) books;
- (void) removeBookObjects:(NSSet*) books;
@end
```

CoreData: Fetch requests

```
NSManagedObjectContext* moc = …;

NSEntityDescription* description = [NSEntityDescription
entityForName:@”Author” inManagedContext:moc];

NSFetchRequest* request = [NSFetchRequest new];

[request setEntity:description];

NSPredicate* predicate = [NSPredicate predicateWithFormat:@”firstName
LIKE %@”, authorName];

[request setPredicate:predicate];

NSArray* results = [moc executeFetchRequest:request error:nil];
```

Заключение

iOS предоставляет простые и понятные API для сериализации объектов.

CoreData — высокоуровневный фреймворк для управления и сохранения наборов объектов, позволяющий не задумываться, какое именно хранилище используется (xml, SQLite, in-memory store).

В iOS так же входит библиотека SQLite, позволяющая напрямую встраивать базы данных SQLite в свое приложение.

Спасибо за внимание!