УДК 004.65:004.05:004.415

РАЗРАБОТКА ПРИЛОЖЕНИЙ ПОД ПЛАТФОРМУ IOS С ПОМОЩЬЮ ТЕХНОЛОГИИ CORE DATA

© 2015 Н.В. Корнеев, В.А. Гончаров

Поволжский государственный университет сервиса, г. Тольятти

Статья поступила в редакцию 01.02.2015

В статье раскрыты основные подходы новой технологии Core Data. Сформулированы принципы оптимизации процессов управления данными на платформе iOS с помощью технологии Core Data, а также вспомогательных сторонних библиотек Mogenerator и Magical Record. Разработаны методы устраняющие проблемы инициализации стека .xdatamodel с сущностями и атрибутами, базирующиеся на создании наследования от класса Event и написание кода уже в новом классе, который не зависит от генерации. Сформулированы принципы оптимизации процессов выборки данных из базы данных с помощью технологии Active Record фреймворка Ruby on Rails.
Ключевые слова: Core Data, Mogenerator, Magical Record, разработка iOS приложений с базой данных.

Фреймворк Core Data предоставляет обобщенные и автоматизированные решения задач, связанных с жизненным циклом и управлением объектами, и является своеобразной надстройкой над SQL [1, 2]. С помощью него можно избавиться от прямых SQL запросов и вместо них использовать более удобный интерфейс программирования приложений - API.

Возможности Core Data:

Отслеживание изменений и отмены изменений. Технологии Core Data обеспечивают встроенное управление отмены (undo) и повтора (redo).

Поддержка связей (relationship). Core Data управляет распространением изменений, в том числе поддержание непротиворечивости и уникальности между объектами.

Futures (faulting) - Core Data может уменьшить объем потребляемой оперативной памяти вашей программы с помощью отложенной загрузки объектов. Он также поддерживает частично функции (futures, и copy-on-write) обмена данными.

Автоматическая проверка значений свойств. В Core data управляемые объекты расширяются стандартным KVC методами проверки, которые гарантируют, что отдельные значения лежат в пределах допустимых диапазонов или что значения подходят по замыслу.

Схемы миграции. Реализовывать миграцию данных это одна из сложных задач. В Core Data средства миграции для схем упрощают задачу и предлагают для этого удобный API, что в большинстве случаях делает его очень эффективным инструментом.

Корнеев Николай Владимирович, доктор технических наук, профессор кафедры «Информационный и электронный сервис». E-mail: niccyper@mail.ru
Гончаров Владимир Александрович, аспирант. E-mail: vladimir1631@yandex.ru

Возможность интеграции с приложением контроллера для синхронизации пользовательского интерфейса. Для этого Core Data предоставляет специальный класс – NS Fetched Results Controller class на iOS, и интегрируется с методом Cocoa-привязки на OS X.

Полная, автоматическая поддержка KVC и KVO паттернов.

Группирование, фильтрация и упорядочивание данных в памяти и в пользовательском интерфейсе.

Автоматическая поддержка хранения объектов во внешних хранилищ (xml, plist, SQLite).

Создание сложных запросов. Вместо того, чтобы писать SQL, есть возможность создавать сложные запросы, используя объект NS Predicate для выборки запроса. NS Predicate обеспечивает поддержку основных функций, коррелированные подзапросы, и другие возможности SQL. С помощью основных данных, он также поддерживает Unicode поиск, сортировку и регулярные выражения.

Правила слияния. Core Data предоставляет встроенные функции для отслеживания версий и поддержки автоматического разрешения конфликтов.

Все указанные возможности требуют практического изучения и анализа, с точки зрения поддержки уровня моделей приложения, объема строк кода, приемлемого уровня качества с использованием модульного тестирования, которые ежедневно используется миллионами разработчиками в широком спектре приложений [7, 8].

Для наглядного анализа технологий Core Data создадим тестовый проект. Для этого зайдем в Xcode и создадим новый проект Master-Detail Application. Назовем проект Example Magical Record и поставим галочку напротив Use Core Data, чтобы автоматически сгенерировался проект с примером кода (рис. 1).

Если далее нажать кнопку Next среда автоматически сгенерирует простой пример для работы с фреймворком. Содержащийся в примере код подвергнем анализу: в нем присутствует инициализация стека .xdatamodel с сущностями и атрибутами, программный код для работы с базой данных, но работа с сущностью Event реализована через паттерн Key-Value Coding. Такое решение является не всегда удобным для использования в профессиональном программировании - нам нужно сгенерировать .h и .m файлы проекта для получения доступа к его свойствам и методам. Для этого нам нужно выполнить следующие операции:

1. Выбрать File->New->File... ;
2. Из списка для iOS выбрать Core Data -> NS Managed Object Subclass;
3. Выбрать созданную нами модель Data Model (в нашем случае - это Example Magical Record.xcdata model);
4. Выбрать сущности. Созданная нами сущность – Event, выбираем ее;
5. Выбираем папку для сохранения файлов.

Теперь мы имеем сгенерированную модель Event. Научившись создавать интерактивно Core Data модели можно приступать к программной реализации. Добавим метод для сравнения промежутка времени между Event:

```
- (NSTimeInterval)intervalToEvent:(Event *)event
{
   return [self.timeStamp
timeIntervalSinceDate:event.timeStamp];
}
```

Также добавим атрибут count с типом integer16 и попробуем заново сгенерировать файлы. Полученные недостатки выражаются в следующем:

Необходимость генерировать заново исходники и заменять старые после изменений.

После генерации – изменений не произошло и внедренный атрибут count не появился.

Решить указанные проблемы можно следующим образом:

Переименовать сущность и сгенерировать новую, но в данном случае нам придется переименовывать все в своем коде.

Удалить старые исходники и создать новые, но в этой ситуации нужно будет мигрировать написанный код в новый модуль.

Описанные выше решения не профессиональны и скорее могут быть использованы в случае обучения студентов, они в значительной степени увеличивают время на разработку проекта. Оптимальный вариантом является создание наследования от класса Event и написание кода уже в новом классе, который не зависит от генерации.

Для решения этой проблемы и был разработан script Mogenerator [3]. Ниже описаны шаги по его установке:

Зайти по ссылке [6], скачать и установить в систему mac os;

Далее нужно создать скрипт. Создаем файл в корне приложения mogend.sh и напишем в нем скрипт:

```
MACHINE_DIR="${PROJECT_DIR}/
ExampleMagicalRecord/Models/CoreData/Private"
HUMAN_DIR="${PROJECT_DIR}/
ExampleMagicalRecord/Models/CoreData"
INCLUDE_H="${PROJECT_DIR}/
ExampleMagicalRecord/Models/ModelIncludes.h"
echo "machine source path - $MACHINE_DIR/"
echo "human source path - $HUMAN_DIR/"
echo "include.h path - $INCLUDE_H"
mogenerator --model "${INPUT_FILE_PATH}/"
--machine-dir "$MACHINE_DIR" --human-dir
"$HUMAN_DIR" --includeh "$INCLUDE_H"
--template-var arc=true
${DEVELOPER_BIN_DIR}/momc -XD_MOMC_
TARGET_VERSION=10.7 "${INPUT_FILE_PATH}"
"${TARGET_BUILD_DIR}/${EXECUTABLE_FOLDER_
PATH}/${INPUT_FILE_BASE}.momd"
echo "Mogend.sh is done"
```

Попробуем разобраться в коде этого скрипта. В переменной MACHINE_DIR мы указываем, где будут храниться исходники, сгенерированной Core Data (их трогать мы не будем, они нужны

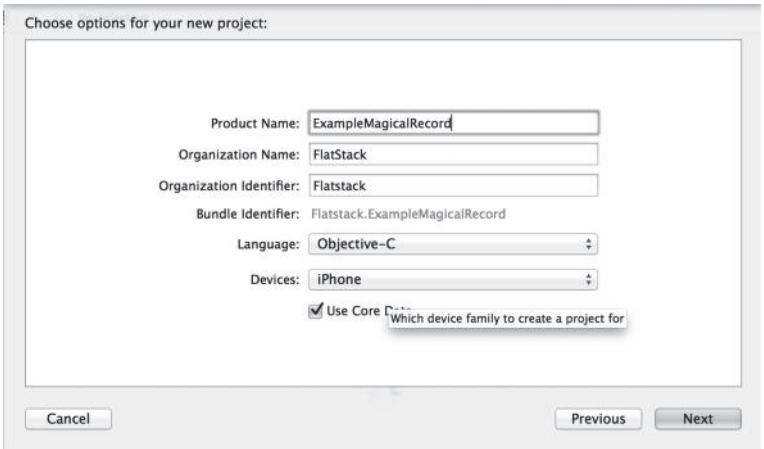

Рис. 1. Окно создания проекта Example Magical Record в Xcode с использованием технологий Core Data

фреймворку), в HUMAN_DIR мы будем хранить исходники для программирования. INCLUDE_H хранит в себе все правила - headers наших сгенерированных классов. Команда mogenerator описывает генерацию файлов. Параметром -- model «${INPUT_FILE_PATH}/» мы сообщаем место нахождения нашего .xcdatamodel, а также параметром --template-var arc=true включаем поддержку ARC. Последняя команда описывает место для сохранения momd файла.

Далее нужно добавить два правила (рис. 2): «Data model version files» и «Data model files» которые содержат путь к нашему скрипту и вид выходных данных.

Следует отметить, что здесь действует классическое правило - если переместить скрипт в другую папку, то в правилах тоже нужно будет обязательно указать новый путь.

Затем нужно указать название классов для сущностей (рис. 3).

Далее для генерирования новых исходников мы компилируем проект. Если при компиляции произошла ошибка, ее можно посмотреть в «Report Navigation» вкладке (рис. 4). Как видно из рис. 4 скрипт создал 2 приватных, сгенерированный для Core Data, и 2 для программирования файлы.

Новые файлы, которые появились в папках ExampleMagicalRecord/Models/CoreData и ExampleMagicalRecord/Models/CoreData/Private, нужно добавить в проект. После добавления в проекте появились сгенерированный приватный класс для Core Data _Event с нашими аттрибутами и класс для программирования Event. Дополнительно Mogenerator сгенерировал методы entityName для получения имени сущности, insertInManagedObjectContext: для быстрой вставки и entityInManagedObjectContext: для получения описания сущности. Также добавился метод для валидация параметра timestamp validateTimeStamp:(id*)value_ error:(NSError**)error_;. добавились 2 категории - CoreDataGeneratedAccessors и CoreDataGeneratedPrimitiveAccessors.

В первом случае создаются методы для удобного доступа к «Relationships», например:

```
- (void)addReviews:(NSOrderedSet*)value_;
- (void)removeReviews:(NSOrderedSet*)value_;
- (void)addReviewsObject:(Review*)value_;
- (void)removeReviewsObject:(Review*)value_;
```

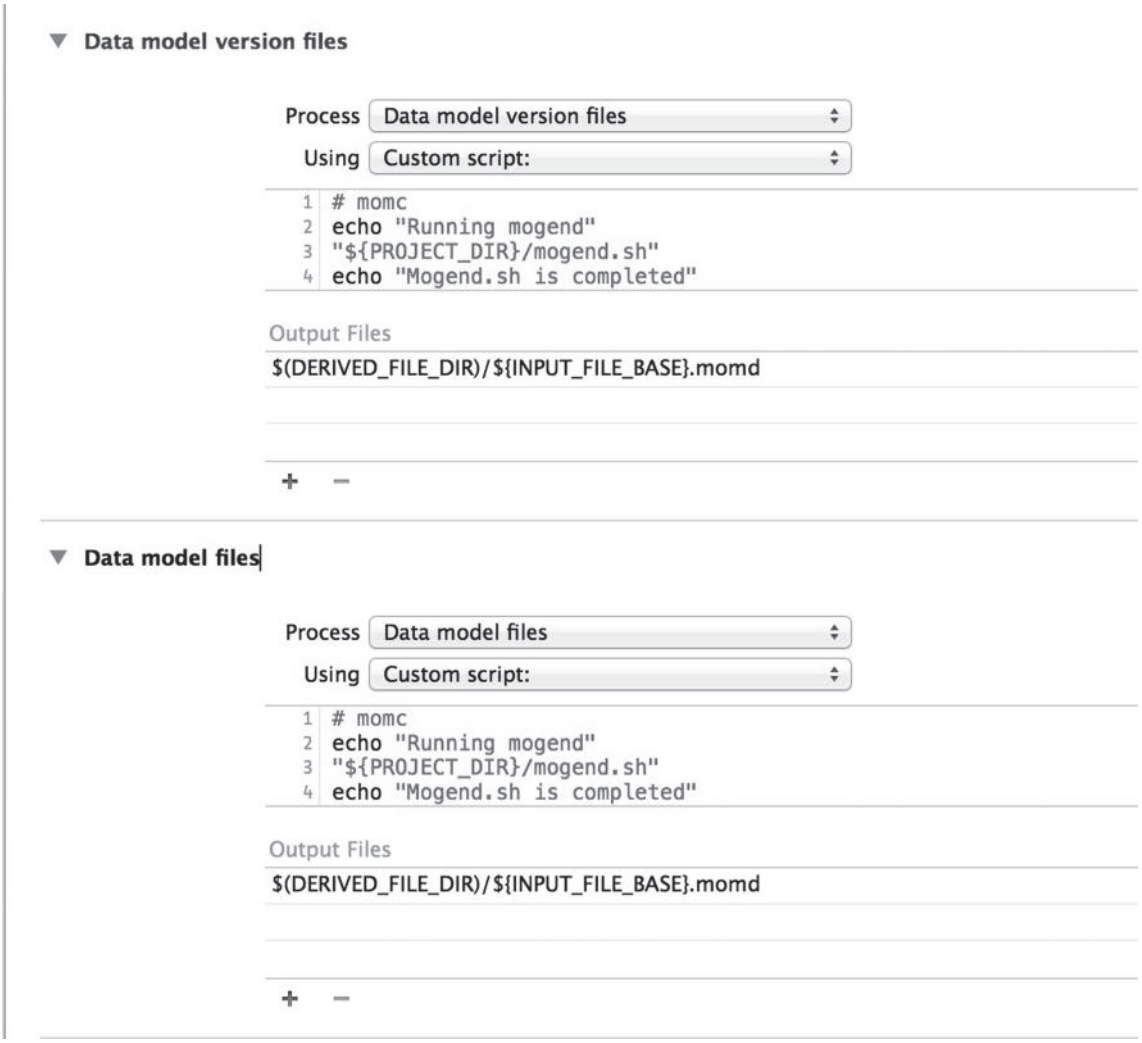

Рис. 2. Окно добавления правил в проект

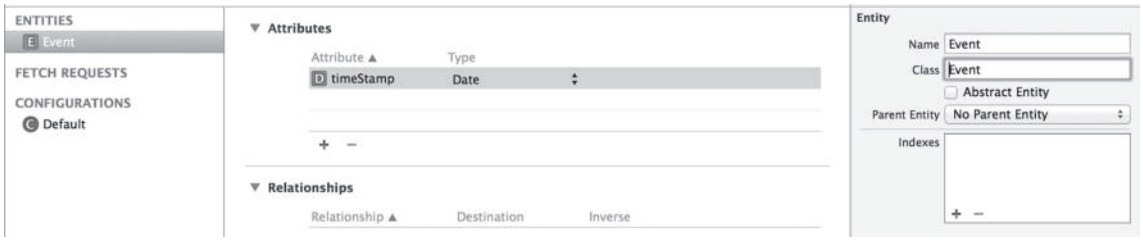

Рис. 3. Окно для добавления классов для сущностей

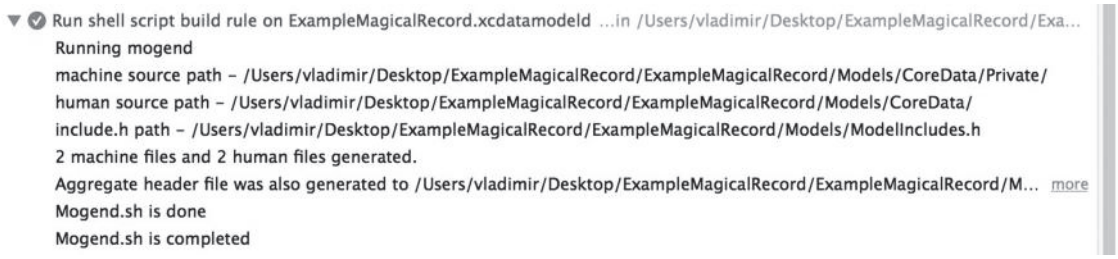

Рис. 4. Вкладка «Report Navigation»

Во втором создались методы для доступа к данным напрямую, а не через accessor:

```
- (NSDate*)primitiveTimeStamp;
- (void)setPrimitiveTimeStamp:(NSDate*)
value;
```

Это дает возможность переопределить метод getter или setter, например, тип данных timestamp можно определить так:

```
- (NSDate *)timeStamp
{
  [super willAccessValueForKey:NSStringFrom
Selector(@selector(timeStamp))];
  NSDate *timestamp      = [self
primitiveTimeStamp];
  [super didAccessValueForKey:NSStringFromS
elector(@selector(timeStamp))];
  if (!timestamp)
  {
    timestamp          = [NSDate date];
    [self setPrimitiveTimeStamp:timestamp];
  }
  return timestamp;
}
```

Для полноты представления возможностей Core Data рассмотрим библиотеку Magical Record [4], как вспомогательный инструмент быстрой разработки приложений.

Magical Record используется для оптимизации процессов выборки данных из базы данных с помощью технологии Active Record фреймворка Ruby on Rails.

Основные цели Magical Record:

Очистить код от реализации стека.

Получить понятные, простые и однострочные выборки.

Использовать модификацию NSFetchRequest, когда необходима оптимизация для запросов.

Проведем анализ указанных возможностей на примере. Для начала необходимо скачать файлы для этой библиотеки с git репозитария. Лучше всего использовать CocoaPods– это позволит нам в несколько шагов установить Magical Record и предоставить простое обновление других зависимых библиотек.

Для установки Magical Record необходимо выполнить следующие действия:

Установить CocoaPods библиотеку. Для этого открываем «Терминал» и пишем команду:

```
sudo gem install cocoapods
```

Создать пустой файл с именем Podfile в корне приложения.

Открыть Podfile и написать в нем нужный нам pod для установки, и ios - минимально допустимую iOS версию приложения:

```
platform :ios, '7.0'
pod 'MagicalRecord'
```

Открыть командную строку, перейти в корневую папку проекта и написать:

```
pod install
```

По окончанию установки в конце будет написано использовать теперь не наш проект, а с расширением .xcworkspace.

Откроем снова наш проект ExampleMagicalRecord и откроем файл AppDelegate.h. Здесь Xcode автоматически сгенерировал 3 свойства и 2 метода:

```
@property (readonly, strong,
nonatomic) NSManagedObjectContext
*managedObjectContext;
@property (readonly, strong,
nonatomic) NSManagedObjectModel
*managedObjectModel;
```

```
@property (readonly, strong,
nonatomic) NSPersistentStoreCoordinator
*persistentStoreCoordinator;

- (void)saveContext;
- (NSURL *)
applicationDocumentsDirectory;
```

Здесь описана инициализация Core Data стека. Разберемся с этим кодом. Сначала мы инициализируем managedObjectModel с нашей momd-моделью, далее инициализация persistentStoreCoordinator, и затем инициализируем главный контекст managedObjectContext. Все это не меньше 50 строк кода. Уберем этот код и сделаем то же самое с помощью Magical Record:

```
- (BOOL)application:(UIApplication *)
application didFinishLaunchingWithOptions:(
NSDictionary *)launchOptions {
    // Override point for customization after
application launch.
    [MagicalRecord setShouldDeleteStoreOn
ModelMismatch:YES];
    [MagicalRecord setupAutoMigratingCore
DataStack];
    return YES;
}
```

Стек автоматически инициализирован в заинкапсулированую static переменную и Core Data можно использовать. Сравним код от Apple для вставки новой сущности:

```
- (void)insertNewObject:(id)sender {
    NSDate *date          = [NSDate date];
    NSManagedObjectContext *context
= [self.fetchedResultsController
managedObjectContext];
    NSEntityDescription *entity = [[self.
fetchedResultsController fetchRequest]
entity];
    NSManagedObject *newManagedObject
= [NSEntityDescription insertNewObjectForEn
tityForName:[entity name] inManagedObjectC
ontext:context];

    // If appropriate, configure the new
managed object.
    // Normally you should use accessor
methods, but using KVC here avoids the need
to add a custom class to the template.
    [newManagedObject setValue:date
forKey:@"timeStamp"];

    // Save the context.
    NSError *error = nil;
    if (![context save:&error]) {
        // Replace this implementation with
code to handle the error appropriately.
        // abort() causes the application to
generate a crash log and terminate. You
should not use this function in a shipping
application, although it may be useful during
development.
        NSLog(@"Unresolved error %@, %@",
error, [error userInfo]);
        abort();
    }
}
```

В случае примера от Apple нам пришлось передать контекст (или каким-то другим способом его получить), получить имя сущности, сделать вставку объекта в контекст, изменить timestamp аттрибут и попытаться сохранить context с новым объектом.

Magical Record вернул нам контекст, который он после изменения сохранит в корневом:

```
[MagicalRecord saveWithBlockAndWait:^(
NSManagedObjectContext *localContext) {
    Event *event          = [Event MR_
createInContext:localContext];
    event.timeStamp       = date;
}];
```

Наша задача - описать изменения в именно в localContext.

Аналогично с удаление объекта с примером от Apple:

```
- (void)tableView:(UITableView *)
tableView commitEditingStyle:(UITa
bleViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)
indexPath {
    if (editingStyle == UITableViewCellEditin
gStyleDelete) {
        NSManagedObjectContext *context
= [self.fetchedResultsController
managedObjectContext];
        [context deleteObject:[self.
fetchedResultsController
objectAtIndexPath:indexPath]];

        NSError *error = nil;
        if (![context save:&error]) {
            // Replace this implementation with
code to handle the error appropriately.
            // abort() causes the application
to generate a crash log and terminate. You
should not use this function in a shipping
application, although it may be useful during
development.
            NSLog(@"Unresolved error %@,
%@", error, [error userInfo]);
            abort();
        }
    }
}
```

и Magical Record:

```
- (void)tableView:(UITableView *)
```

```
tableView commitEditingStyle:(UITa
bleViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)
indexPath {
    if (editingStyle == UITableViewCellEditin
gStyleDelete) {
   [MagicalRecord saveWithBlockAndWait:^(
NSManagedObjectContext *localContext) {
        Event *event    = [[self.
fetchedResultsController
objectAtIndexPath:indexPath] MR_
inContext:localContext];
        [event MR_deleteEntity];
      }];
   }
   }
```

Фильтрация объектов (NSPredicate) и сортировка (NSSortDescription) для сложных запросов поддерживаются, но их использование потребуется для более сложных запросов. Сравним создание функции NSFetchedResultsController с примером от Apple:

```
    - (NSFetchedResultsController *)
fetchedResultsController
   {
     if (_fetchedResultsController != nil) {
       return _fetchedResultsController;
     }

    NSFetchRequest *fetchRequest =
[[NSFetchRequest alloc] init];
     // Edit the entity name as appropriate.
     NSEntityDescription
*entity = [NSEntityDescription
entityForName:@"Event"
inManagedObjectContext:self.
managedObjectContext];
     [fetchRequest setEntity:entity];

     // Set the batch size to a suitable number.
     [fetchRequest setFetchBatchSize:20];

     // Edit the sort key as appropriate.
     NSSortDescriptor *sortDescriptor
= [[NSSortDescriptor alloc]
initWithKey:@"timeStamp" ascending:NO];
     NSArray *sortDescriptors = @
[sortDescriptor];

     [fetchRequest setSortDescriptors:sortDe
scriptors];

     // Edit the section name key path and
cache name if appropriate.
     // nil for section name key path means
"no sections".
     NSFetchedResultsController
*aFetchedResultsController =
[[NSFetchedResultsController alloc]
initWithFetchRequest:fetchRequ
est managedObjectContext:self.
managedObjectContext
sectionNameKeyPath:nil cacheName:nil];
     aFetchedResultsController.delegate =
self;
     self.fetchedResultsController =
aFetchedResultsController;

    NSError *error = nil;
    if (![self.fetchedResultsController
performFetch:&error]) {
     // Replace this implementation with
code to handle the error appropriately.
     // abort() causes the application to
generate a crash log and terminate. You
should not use this function in a shipping
application, although it may be useful during
development.
     NSLog(@"Unresolved error %@, %@",
error, [error userInfo]);
     abort();
    }

    return _fetchedResultsController;
   }
```

и Magical Record:

```
    - (NSFetchedResultsController *)
fetchedResultsController
   {
     if (_fetchedResultsController != nil) {
       return _fetchedResultsController;
     }

    NSFetchedResultsController
*aFetchedResultsController    =
     [Event MR_
fetchAllSortedBy:@"timeStamp"
            ascending:NO
          withPredicate:nil
            groupBy:nil
            delegate:self];

     _fetchedResultsController        =
aFetchedResultsController;

     return _fetchedResultsController;
   }
```

Как видно, наша задача получить все события и отсортировать их по убыванию решается намного быстрее. Magical Record содержит большое количество вспомогательных методов для ускорения разработки. Готовый тестовый проект можно скачать по ссылке [5].

ВЫВОДЫ

С Core Data объем кода, который нужно написать, чтобы поддержать уровень моделей приложения уменьшается от 50% до 70% строк кода. Это связано прежде всего с особенностями, перечисленными выше, обеспечивающими Core Data возможностями, которые не придется реализовывать самостоятельно.

Core Data формирует приемлемый уровень качества, которое обеспечивается за счет модульных тестов, и ежедневно используется миллионами разработчиками в широком спектре приложений. Кроме того, в дополнение к функциям безопасности и обработкам ошибок, он предлагает оптимальное распределение оперативной памяти любого конкурирующего решения. Иными словами, можно потратить много времени, тщательно обрабатывая собственное решение, оптимизированное для конкретной проблемной области, и не получить никаких преимуществ в производительности по сравнению с тем, что предлагает Core Data бесплатно для любого приложения.

В дополнение к преимуществам самого framework, Core Data хорошо интегрируется с OS X инструмента. Встроенные инструменты позволяют создать свою графическую схему, быстро и легко. В разрабатываемом приложении можно использовать инструменты для измерения производительности Core Data и с их помощью отлаживать различные проблемы. Эти аспекты способствуют дальнейшему ускорению разработки и отладки приложений.

При разработке приложений с Core Data необходимо учитывать следующие факты:

Core Data не является реляционной базой данных или системой управления реляционными базами данных (СУБД). Core Data обеспечивает инфраструктуру для управления изменениями, сохранения и извлечения объектов из хранилища.

Core Data не является панацеей. Она не снимает необходимости написания кода. Можно создавать сложные приложения исключительно с помощью моделирования данных Xcode инструментов и Interface Builder, а для создания современных приложений все равно придется писать код.

СПИСОК ЛИТЕРАТУРЫ

1. *Harwani B.M.* Core Data iOS Essentials. – Birminghamv: Packt Publishing, 2011
2. Core Data Programming Guide: [Электронный ресурс] // Apple Inc., 2004-2014. URL:https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreData/Articles/cdTechnologyOverview.html#//apple_ref/doc/uid/TP40009296-SW1. (Дата обращения: 2.10.2014).
3. Mogenerator: [Электронный ресурс] // GitHub, Inc, 2014. URL: https://github.com/rentzsch/mogenerator. (Дата обращения: 2.10.2014).
4. MagicalRecord: [Электронный ресурс] // GitHub, Inc, 2014. URL: https://github.com/magicalpanda/MagicalRecord. (Дата обращения: 2.10.2014).
5. Example Magical Record and MOGenerator: [Электронный ресурс] // GitHub, Inc, 2014. URL: https://github.com/VladimirGoncharov/ExampleMagicalRecord. (Дата обращения: 2.10.2014).
6. Mogenerator generates Objective-C code for your Core Data custom classes: [Электронный ресурс] // GitHub, Inc, 2014. URL: http://rentzsch.github.io/mogenerator/. (Дата обращения: 2.10.2014).
7. *Корнеев Н.В., Гончаров В.А.* Система продаж CRM и сравнение SaaS-моделей // Техника машиностроения. 2014. Т. 21. № 3 (91). С. 60-63.
8. *Корнеев Н.В., Осипов И.В.* Алгоритмические и программные принципы построения и разработки системы расширяемых шаблонов для контроля и оптимизации торговых систем на основе облачной сети распределенных вычислений // Ученые записки РГСУ. 2012. №3. С. 163…169.

APPLICATIONS PROGRAMMING UNDER GANTRY IOS BY MEANS OF PRODUCTION ENGINEERING CORE DATA

© 2015 N.V. Korneev, V.A. Goncharov

Volga Region State University of Service, Togliatti

In paper the basic approaches of new production engineering Core Data are opened. Principles of optimisation of control procedures by data on a gantry iOS by means of production engineering Core Data, and also auxiliary indirect libraries Mogenerator and Magical Record are formulated. Methods eliminating problems of initializing стека.xdatamodel with substances and the attributes, founded on making of inheriting from class-room Event and code writing already in the new class-room which one does not depend on generation are developed. Principles of optimisation of processes of sampling of data from a database by means of production engineering Active Record framework of Ruby on Rails are formulated.

Keywords: Core Data, Mogenerator, Magical Record, development iOS applications with a database.

Nikolay Korneev, Doctor of Technics., Professor at the Information and Electronic Service Department.
E-mail: niccyper@mail.ru
Vladimir Goncharov, post-graduate student.
E-mail: vladimir1631@yandex.ru