УДК 004.043

Е.И. Бородай, инженер-программист,
Г.В. Табунщик, канд. техн. наук, доцент,
Запорожский национальный технический университет, Украина
icydreams3@gmail.com, galina.tabunshchik@gmail.com

Особенности сервисов обработки данных для мобильных устройств

Авторами рассмотрены особенности реализации сервисов обработки данных для мобильных устройств. Предложена реализация архитектуры сервиса хранения данных с использованием технологии plist. В статье приведены результаты исследования наиболее популярных сервисов и реализации предложенной авторами.

Ключевые слова: iOS, хранение данных, архитектура, сервис, plist, Core Data, XML, SQL Lite

Актуальность

На бирже труда, в данный момент, в сфере информационных технологий востребовано около 46 % программистов для мобильных платформ от общего числа программистов [1].

При возрастающих требованиях и увеличении количества мультимедии в мобильных устройствах, главной задачей программиста является обеспечение быстрой работы с данными при жестком ограничении на объем места, занимаемого этими данными.

Данная задача является одной из первостепенных и ее решение необходимо во многих приложениях, начиная от игр последнего поколения, над которыми работают сотни программистов в течении нескольких месяцев и заканчивая рекламными приложениями, которые пишут один, два программиста за несколько недель.

Поэтому актуальным является анализ шаблонов и архитектур наиболее используемых сервисов для мобильных платформ с дальнейшей их оптимизацией для возникающих задач.

Постановка задачи

Существует очень много определений архитектуры сервисов мобильных приложений. В данном случае под архитектурой сервисов мобильных приложений будем понимать структуру, определяющую их компоненты, функции и взаимодействие [2].

К базовым задачам мобильных приложений относятся [2]:

- аутентификация и авторизация (Authentication and Authorization);
- кэширование и состояние (Caching and State);
- взаимодействие (Communication);
- композиция (Composition);
- параллельные вычисления и транзакции (Concurrency and Transactions);
- управление конфигурацией (Configuration Management);
- связанность и сцепление (Coupling and Cohesion);
- доступ к данным (Data Access);
- работа с исключениями (Exception Management);
- ведение логов и мониторинг (Logging and Instrumentation);
- взаимодействие с пользователем (User Experience);
- проверка данных (Validation);
- поток операций (Workflow).

Без хранения данных не может обойтись ни одно приложение. Сложность реализации данного сервиса для мобильных платформ неуклонно растет даже при небольшом увеличении сложности проекта.

Например, рассмотрим возникающие задачи для таких приложений как ежедневные игры, приложение взаимодействия с социальной сетью и приложение «Словарь».

В стандартных играх задача хранения данных состоит в сохранении и загрузке данных о «спящих» (неиспользуемых) объектах, с целью освобождения места в оперативной памяти для отработки операций с объектами реального времени. Главным критерием при решении данных задач является скорость записи и загрузки данных, так как пользователь ожидает плавную работу и быстрый отклик от игрового приложения.

В приложениях взаимодействия с социальной сетью основной задачей является сохранение кэша данных для уменьшения

количества обращений в сеть, а также для быстрой загрузки стандартных компонентов. При поиске решения требуется найти баланс между скоростью работы приложения и объемом места, занимаемым приложением.

В приложениях типа «Словарь», происходит обращение к локальной базе данных для поиска перевода. В данном случае задачей реализации является надстройка над базой данных, которая хранит десятки тысяч слов. Поэтому основное внимание уделяется именно уменьшению объема места занимаемого приложением, ведь пользователь не будет использовать словарь объемом в половину имеющихся ресурсов.

Цель работы – реализация сервиса, позволяющего оптимизировать работу с данными для мобильных приложений.

Задачи – исследовать существующие сервисы доступа к данным, реализовать алгоритм, позволяющий увеличить скорость обработки данных с минимальными затратами процессорного времени.

Исследование сервисов доступа к данным для iOS

Выполним анализ наиболее широко используемых сервисов для iOS, направленных на работу с данными: SQLLite, Core Data, и бинарных данных.

1. SQLLite – легковесная встраиваемая реляционная база данных(БД), которая имеет синтаксис, схожий с чистым SQL. Она используется в большинстве мобильных приложений и легко переносима между мобильными операционными системами [8]. Библиотека написана на чистом C, что обеспечивает быстродействие будущему приложению.

2. Core Data – фреймворк, предоставляемый компанией Apple, который идет в комплекте с iOS. Данный фреймворк способен работать с разными видами данных таких как: XML, binary, SQL Lite и inMemory.

Особенностями Core Data является возможность работы со сложными графами объектов, выполнение операций с множественными связями и операций с подтверждением запросов.

Так как Core Data является надстройкой на SQL Lite, то стандартные операции, такие как удаление или добавление объекта будут работать медленнее. Другим недостатком является то, что Core Data является стандартом OS X и не может взаимодействовать с другими операционными системами.

3. Сервис непосредственного хранения бинарных данных на носители устройства в формате raw (формат данных, при котором хренение производится в том состоянии, в котором и используется).

В качестве критериев для дальнейшего анализа были выбраны:

- скорость работы определенной реализации при разных условиях выполнения операций и разных поставленных задачах;
- интерфейс для работы с БД;
- безопасность хранения информации;
- объем памяти, занимаемый на накопителе.

Для тестирования работы сервисов хранения информации был создан класс Shop, в который была помещена структура данных, представленная на рисунке ниже.

```
14 @interface Shop: NSObject
15 {
16     NSNumber* id;
17     NSString* name;
18     NSString* address;
19     NSDate*   updateDate;
20     NSNumber* latitude;
21     NSNumber* longitude;
22     NSArray* specialization;
23 }
```

Рисунок 1 – Структура данных класса Shop

Объектов класса Shop в программе насчитывалось около 10 тысяч. Целевая функция — снижение отклика программы.

В таблице 1 занесены время, требуемое на выполнение операций и объем места занимаемый БД.

Таблица 1 – Результаты работы сервисов хранения данных

Операция \ сервис	Core Data	SQL Lite	Binary storage
Обновление списка (ms)	1.78	1.32	4.31
Поиск по имени (ms)	0.71–1.11	0.52 – 0.97	3.19 – 7.13
Поиск в массивах (ms)	0.54	0.94	3.57
Добавление в базу данных (ms)	0.23	0.12	2.23
Удаление с базы данных (ms)	0.53	0.34	2.53
Место занимаемое на накопители устройства (kB)	32	27	96

Как видно из результатов тестирования, Core Data проигрывает в скорости SQL Lite при стандартных запросах. Это происходит потому, что вначале Core Data обращается к своим сервисам, а после уже выполняет запрос к БД. При решении задачи поиска по имени время на выполнение запросов будет идентично для обоих

случаях, поэтому отсутствует дополнительное обращение к сторонним сервисам. В случае работы с массивами данных Core Data эффективнее почти в два раза, благодаря уменьшению места, отведенного под хранение массивов данных. При использовании Core Data не требуется вручную разбивать массив на данные и сохранять каждый элемент, а также это уменьшает процессорное время которое на выполнение самих операций с БД. Разница в занимаемом данными пространстве при использовании сервисов, описанных выше, также зависит только от того что Core Data является фреймворком (надстройкой) над SQL Lite, а значит и сохранят некоторые структуры данных требуемые для корректной работы с БД.

Метод, используемый для хранения данных в бинарном виде проигрывает по сравнению с остальными. Основной его недостаток это отсутствие индексации как таковой. Даже если ее выполнять программно, то она не сможет показать результаты лучше, чем индексация выполняемая операционной системой на носителе информации. Так как данные сохраняются в чистом виде, а не с использованием алгоритмов сжатия данных, как в SQL Lite или Core Data, то увеличивается место для хранения почти в три раза. Также данные хранятся не отдельно друг от друга, а в едином массиве данных, и чтобы получить доступ к определенному полю требуется выполнить чтение всего блока данных, и только после происходит обращение к его полям. Данный метод можно использовать в случае если используемые данные либо однотипные, либо неделимые (например видео, аудио файлы, названия объектов которых будут представлены сразу).

Кроме вышеперечисленного еще одним существенным недостатком Core Data и SQL Lite является огромное количество кода для обработки как самой БД, так и запросов для нее.

Алгоритм доступа к данным:

Исходя из данных, полученных в результате анализа, было решено использовать специальные файлы формата plist, которые iOS использует для хранение данных конфигурации. Основным преимуществом является то, что эти файлы индексируются первыми в программах, а значит обращение к ним будет выполняться в первую очередь.

Шаблон PList предназначен для оптимизации кодов табличных списков, основанных на информации из базы данных The Plant List. Каждый вызов шаблона соответствует одной строке такой таблицы. Формат основан на языке разметки XML и Core Foundation DTD. Список параметров в файле представлен в виде пар <Ключ – Значение>. Данные в файле могут храниться как в текстовом, так и в двоичном виде [3]. Например: <key> Ключ </key> <string> Значение(строка) </string>.

Как упоминалось выше, кроме скорости обработки запроса немаловажным критерием является объем памяти, занимаемый данными. Поэтому было решено использовать сжатие.

Для каждой сжимаемой последовательности символов однократно либо в каждый момент времени собирается статистика её встречаемости в кодируемых данных. На основе этой статистики вычисляется вероятность значения очередного кодируемого символа (либо последовательности символов). После этого применяется кодирование Хаффмана [6], для представления часто встречающихся последовательностей короткими кодовыми словами, а редко встречающихся – более длинными.

Алгоритм работы сервиса обработки данных:

- ссоздание шаблона хранения данных с помощью plist;
- организация записи данных на носитель по типу данных;
- индексация результатов записи;
- использование алгоритма кодирования Хаффмана.

На рисунке 2 представлен пример файла формата .plist, используемый для тестирования разработанного сервиса.

Key	Type	Value
id	Number	0
name	String	
address	String	
updateDate	Date	Mar 6, 2012 12
latitude	Number	0
longitude	Number	0
▶ specialization	Array	(0 items)

Рисунок 2 – Организация данных с использованием сервиса хранения данных в файлах формата .plist.

Для анализа эффективности разработанного сервиса была использована такая же схема данных, что и в предыдущих случаях, результаты введены в таблицу 3.

Таблица 3 – Результаты работы сервиса хранения данных с использованием plist и сжатием

операция/ сервис	Plist без сжатия	Plist с сжатием
Обновление списка (ms)	1.13	1.21
Поиск по имени (ms)	0.34 – 0.63	0.42 – 0.76
Поиск в массивах (ms)	0.31	0.43
Добавление в базу данных (ms)	0.07	0.12
Удаление с базы данных (ms)	0.15	0.24
Место занимаемое на накопители устройства (kB)	43	38

Как видно из таблицы 3, разработанный сервис почти по всем параметрам имеет преимущества над стандартными методами.

Такие результаты обусловлены тем, что приложение в iOS индексирует данные в plist файлах в первую очередь, а значит, и поиск доступа с ним будет осуществляться быстрее. Также данные файлы лишены ненужных в данной задаче связей между схемами данных, таких как таблицы. Также быстрая скорость работы обусловлена тем, что библиотека взаимодействия с plist написана на языке, на котором написана сама операционная система, поэтому для взаимодействия сервиса с операционной системой не требуется преобразовывать код в «родной» для ядра.

Разработанный сервис показывает худшие результаты по объему занимаемого места на диске. В таблице 4 представлено соотношение работы разработанного сервиса с наилучшими результатами стандартных сервисов.

Таблица 4 – Результаты работы табличного сервиса plist

операция/ сервис	Без сжатия	С сжатием
Обновление списка (ms)	+ 15%	+ 8,5%
Поиск по имени (ms)	+ 44%	+ 21%
Поиск в массивах (ms)	+ 40%	+ 20%
Добавление в базу данных (ms)	+ 7%	0 %
Удаление с базы данных (ms)	+ 51%	+ 25%
Место занимаемое на накопители устройства (kB)	- 53%	- 21%

Данный сервис наиболее эффективен при работе с данными, имеющими стандартные для iOS форматы, такие как NSString, NSNumber, NSArray и т.д. Также эффективен при работе с данными в виде структур, представленных неделимым массивом данным, которые не имеют релятивных связей, таких как таблицы по ключам.

Выводы

Научная новизна работы – исследованы существующие методы доступа к данным. Разработан метод обработки данных, основанный на оптимизации кодов табличных листов и кодирования данных, что позволило уменьшить отклик системы и объем занимаемых ресурсов.

Практическая ценность работы состоит в том, что реализован сервис, предназначенный для обработки данных для iOS, демонстрирующий на 15% быстрее процессорное время затраченное на обработку данных по сравнению с существующими аналогами.

Данную архитектуру сервиса хранения данных возможно применять в классах задач, которые требуют баланса между производительность и местом занимаемым базой данных, таких как приложения работы с социальными сетями или рекламные приложения.

Список литературы

1. Сайт трудоустройства в Украине [Электронный ресурс]. – URL: http://jobs/articles/4550/ (дата обращения: 21.01.2012).

2. Архитектура приложений – горячие точки [Электронный ресурс]: сайт. - URL: http://habrahabr.ru/post/40660/ (дата обращения: 15.02.2012).

3. Pilone D., Head First iPhone Development / D. Pilone , T. Pilone. – O'Reilly Media, Inc., 2009. – 560c.

4. Introduction to Property Lists [Электронный ресурс]: сайт. - URL: https://developer.apple.com/library/mac/#documentation /Cocoa/Conceptual/PropertyLists/Introduction/Introduction.html (дата обращения: 03.02.2012).

5. Кнастер С., Objective-C 2.0 и программирование для Mac / С. Кнастер, М. Далримпл. – М: ООО"И.Д. Вильямс", 2010. – 320с.
6. Дамримпл М., Разработка приложений для iPhone, iPad, iPod Touch с использованием iOS SDK / М. Дамримпл, Д. Ламарч. – М: Вильямс, 2011. – 624с.
7. Isted T. Core Data for iOS: Developing Data-Driven Application for the iPad, iPhone & iPod Touch / T. Isted, T. Harrington. – Advisor Wesley Professional, 2001. – 304с.
8. Kreibich A., Using SQLite / A. Kreibich. – O'Reilly Media, 2010. – 528с.

Надійшла до редакції 31.03.2012

Є.І. БОРОДАЙ, Г.В. ТАБУНЩИК
Запорізький національний технічний університет

E.I. BORODAI, G.V. TABUNSHCHIK
Zaporizhzhya National Technical University

Особливості сервісів обробки даних для мобільних пристроїв.

Авторами розглянуті особливості реалізації сервісів обробки даних для мобільних пристроїв. Запропоновано реалізацію архітектури сервісу зберігання даних з використанням технології plist. У статті наведені результати дослідження найбільш популярних сервісів і реалізації запропонованої авторами.

Ключові слова: iOS, зберігання даних, архітектура, сервіс, plist, Core Data, XML, SQL Lite

Architectural Features of Data Processing for Mobile Devices.

The paper discusses the peculiarities of data services implementation for mobile devices. The implementation of the data storage service architecture with plist technology is considered. The paper provides the results of studying most popular services and application implementation.

Keywords: iOS, data storage, architecture, service, plist, Core Data, XML, SQL Lite