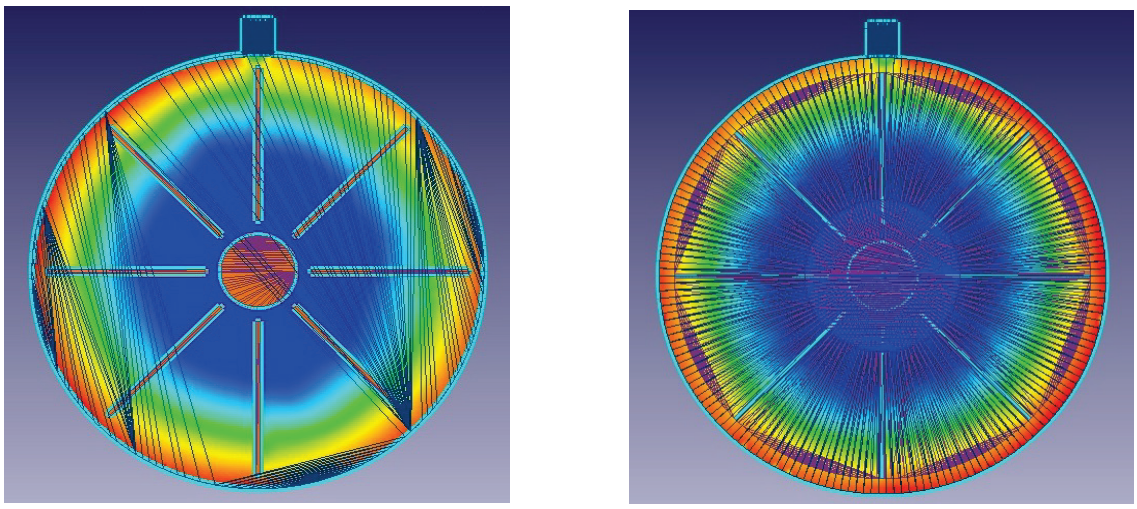

Рис. 4. Схема розподілення температурного поля

в корпусі гідродинамічного перетворювача

Отриманий результат показав, що при швидкості подачі води 1 м/с та частоті обертання робочого колеса 1200 об/хв вода може нагріватися до температури 45°С.

Висновки Нагрівання рідини в гідравлічному перетворювачі енергії відбувається за рахунок сили тертя та процесів вихроутворення, інтенсивність яких можна змінювати регулюючи частоту обертання диска перетворювача та змінюючи розміри внутрішніх параметрів корпусу. При відношенні величини ширини зазору між диском та корпусом перетворювача до діаметру робочого диску (*B/D*) у діапазоні 0,02-0,05 досягається максимальна ефективність роботи перетворювача.

Література

1. Арабаджев А.М. Почему электроотопление – путь к энергосбережению // Энергосбережение. – 2004. – № 5. – С. 6–8.

2. Берковский Б. М., Кузминов В. А. Возобновляемые источники энергии на службе человека. – М.: Наука, 1987.

3. Пфлейдерер К. Лопаточные машины для жидкостей и газов – М.: Машгиз, 1960.

Статья отправлена: 04.04.2017 г.

ЦИТ: ua117-098
DOI: 10.21893/2415-7538.2016-05-1-098
УДК 004.4

Загорський П.М.

ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПОБУДОВИ АРХІТЕКТУРИ МОБІЛЬНОГО ДОДАТКУ ДЛЯ iOS ЗА РАХУНОК ВИДІЛЕННЯ БІЗНЕС ЛОГІКИ В ОКРЕМИЙ ПОТІК ВИКОНАННЯ ЗІ ЗБЕРЕЖЕННЯМ ПЛАВНОСТІ ІНТЕРФЕЙСУ

НТУУ "КПІ" ім. Ігоря Сікорського

Zahorskyi P.M.

MOBILE APPLICATION ARCHITECTURE IMPROVEMENT THROUGHT SPLITTING BUISENESS LOGIC INTO SEPARATED THREADS WITHOUT ANY REDUCE IN UI PERFORMANCE

NTUU "Igor Sikorsky Kyiv Polytechnic Institute"

Аннотация. В данній статті запропоновано метод побудови архітектури мобільного додатку iOS. Основною проблемою у час стрімкого розвитку мобільних додадатків та зі збільшенням кількості активних користувачів є продуктивність та паралелльність виконання задач. Основна особливість представленної архітектури полягає у виділенні бізнес логіки та усіх функцій де вона використовується у окремий потік виконання. Цей підхід дозволить вирішити проблему гальмування плавності інтерфейсу мобільного додатку з використання великої кількості задач, та збільшити кількість та трудомісткість одночасно виконуваних операцій.

Ключевые слова: архітектура, потік виконання, інтерфейс, iOS, об'єкт планування

Abstract. In this article we describe the method of architecture constructing of mobile application iOS. The main problem during the rapid rise of mobile applications use and the increasing number of active users is paralellnist productivity and performance problems. The main feature of architectural representation involves the separation of business logic and all functions where it is used in a separate thread of execution. This approach will solve the problem inhibition of smooth interface of mobile applications using a large number of problems and increase the number and complexity of concurrent transactions.

Key words: architecture, execution thread, iOS, schedulable object

Вступление

На даний момент все більше і більше користувачів надають перевагу мобільним додаткам, ніж використовуванню web-сайтів. Усі сервіси переходять на додатки, і динаміка росте. У зв'язку з цим на додатки покладається все більше логіки та задач, не просто відображення інформації, а набагато більше, наприклад завантаження великої кількості фотографій, файлів, виконання великої кількості бізнес задач одночасно.

Основной текст

При збільшенні складності і навантаження звичайні додатки потребують модифікацій у своїй архітектурі та у паттернах розробки. Також, однією з найважливіших частин мобільного додатку є інтерфейс, чим більш чуйним є представлення та графіка мобільного додатку, тим більш зручно користувачу використовувати цей продукт. Чуйність інтерфейсу залежить в першу чергу від кількості та складності функціоналу додатку або окремої його частини, отже, чим більшу кількість функцій має певний контролер (англ. View Controller) (виконавчий модуль який поєднує представлення (англ. View) та логіку виконання задач та функцій) тим більше часу необхідно йому на виконання логіки та бізнес задач.

Під поняттям "плавного інтерфейсу" у випадку мобільної розробки під iOS

платформи, будемо розуміти такий інтерфейс, який може оновлюватись 60 разів за секунду. Інакше кажучи :

$$60\ FPS = 16{,}7\ ms\ /\ frame$$

де fps (frames per second) - це швидкість, з якою мобільний девайс iOS формує зображення та відображає послідовні у відповідній послідовності, що називаються кадрами, та ms/frame – загальний час обчислнь та візуалізації фрейму. [1]

Інакше кажучи, кожен рівень виконання повинен встигати завершити усі свої задачі за 16,7 мілісекунд. Для загального розуміння, візьмемо час, який необхідний для відмалювання не складного вікна інтерфейсу, це ~ 10 мілісекунд. Отже, методом не складних підрахунків усі задачі бізнес логіки мають бути виконані за 6,7 мілісекунд, не дуже багато.

Отже, ми яскраво бачимо проблему – якщо використовувати стандартні методи та архітектуру, часу для відпрацювання бізнес логіки середньої складності не вистачає, а якщо бути точним – вистачає, але з урізанням плавності інтерфейсу. Провевши певні дослідження, було виявлено декілька способів вирішення проблеми.

- оптимізація самих алгоритмів бізнес логіки, та оптимізація структур данних
- виділення проблемного моменту(важкого для виконання сценарія) з головного потоку виконання
- виділення з головного потоку виконання усіх функцій додатку, за виключенням перетворень інтерфейсів та рендерінга їх самих

Третій спосіб здається найбільш логічним та грунтовним, але для того щоб зупинитись на ньому, потрібно порівняти його з першими двома.

Оптимізація алгоритмів. У кожної оптимізації є своя границя. Звичайно, можно скоротити увесь функціонал, збільшити кількість процесів до 5-10 та кількість об’єктів що будуть завантажуватись / використовуватись, але як правило, при ускладненні логіки (для будь якого сервісу який має мобільний додаток, збільшення функціоналу та його складность є лиш питанням часу) знайдеться такий сценарій, який нівелює усю оптимізацію. Також, з оптимізацією якості “продуктивності”, стовідсотково буде страждати користувацький інтерфейс.

Виділення у окремий потік виконання. Цей підхід має певну низку недоліків, а саме:

- потрібно постійно слідкувати за зв’язками об’єктів для своєчасного рефакторингу однопотокової реалізації на багатопотокову, та навпаки
- потокобезпечні методи важко реалізовувати, в першу чергу, через те, що маючи певний рівень прикладної логіки необхідно враховувати саму специфіку багатопотокового программування
- найбільш вагомою причиною є те, що банальне використання примітивів синхронізації потоків, може тільки ускладнити відмалювання інтерфейсу, а саме, зробити його ще більш тяжким та повільним, на відміну від однопоточної реалізації

Перебравши можливі варіанти вирішення задачі гладкого переходу та

відображення інтерфейсу мобільного додатку, ми зупинилися на останньому способі реалізації. Вже існуючий паттерн розробки SchedulableObject допоможе нам ефективно та якісно реалізувати цей метод, та досягти мети. [2]

Під час розробки серверної та мобільної/веб частини важку для інтерфейсу або девайсу бізнес логіку прийнято виділяти в окремий процес. Взаємодія між процесами все ж залишається однопотоковою. На жаль, у світі мобільної розробки під iOS ми маємо лише один процес, однак ми можемо використовувати той самий принцип, але у мікро постановці, зробивши невеликі трансформації – замінити окремий процес окремим потоком виконання, а механізм зв'язку – не прямими викликами.

На справі, данну задачу ми можемо сформулювати більш чітко, розбивши її на певні пункти:

1 Етап: Створити один окремий працюючий поток виконання

2 Етап: Проассоціювати весь цикл обробки данних та певних сценаріїв / подій і спеціально створений об'єкт, який буде слідкувати за цими діями та отримувати повідомлення які нам будуть необхідні – такий об'єкт буде називатись планувальником(scheduler далі)

3 Етап: Зв'язати кожен об'єкт який може бути змінено у час виконання певного сценарію з одним з schedulers. Чим більше об'єктів буде пов'язано з scheduler який планує взаємодію між ними, тим більше часу залишиться у головного потока на найголовніше завдання у поставленній нами задачі – відмалювання користувацького інтерфейса.

4 Етап: Створити певну логіку зв'язку та виклику об'єктів в залежності від їх належності до певного scheduler. Об'єкти можуть належати як до одного scheduler(зв'язок виконується прямим викликом), так і до двох різних (у такому випадку, звязок виконується не прямим викликом через scheduler іншого об'єкта)

Давайте розглянемо алгоритм більш детальніше. Для цього ми зможемо розбити його на складові, та дати кожному з них характеристику. Першою складовою є модуль "Події". Блоки(кложури, замикання) найбільш комфортна форма абстракції для будь якого Event в iOS. У вигляді коду ця форма представлена нижче.

```
typealias Event = () -> Void
```

Наступним є – черга подій. Найбільш складний і найбільш важливий модуль. Так як подіїї у чергу будуть потрапляти з різних потоків виконання, то черга потребує потікобезпечної імплементації, адже thread-safety(безпека потоків виконання) є одним з головних вимог до нашої архітектури.

```
class EventQueue {
   private let semaphore = DispatchSemaphore(value: 1)
   private var events = [Event]()

   func pushEvent(event: @escaping Event) {
      semaphore.wait()
      events.append(event)
      semaphore.signal()
   }
```

```
    func resetEvents() -> [Event] {
        semaphore.wait()
        let currentEvents = events
        events = [Event]()
        semaphore.signal()
        return currentEvents
    }
}
```

Цикл обробки сигналів та повідомлень подій. Імплементує послідовну(і тільки, це є дуже важливо) обробку черги подій, це гарантує що к об'єктам які реалізовані, виклики будуть йти з одного потоку виконання. Для реалізації цієї властивості в iOS SDK є свій компонент NSRunLoop, але реалізацію можно представити як:

```
class RunLoop {
    let eventQueue = EventQueue()
    var disposed = false

    @objc func run() {
        while !disposed {
            for event in eventQueue.resetEvents() {
                event()
            }
            Thread.sleep(forTimeInterval: 0.1)
        }
    }
}
```

Іншим дуже важливим об'єктом є сам потік виконання. Набільш низькорівнева реалізація – NSThread в iOS SDK. На справі, більшість розробників надають перевагу більш високорівевим примітивам, таким як NSOperationQueue та Grand Central Dispatch, останній є найбільш популярним, та найбільш простим у реалізації.

Нарешті, нам потрібно розглянути останні два компоненти, планувальник та обьект планування, тобто класс який планує та направляє певну операцію, бізнесс процесс, функцію у потрібний нам потік виконання, та само об'єкт який подорожує по потокам.

Планувальник. Як вже було зазначено вище – основною задачею є роспізнавання операції та направлення її у потрібний та валідний шлях(потік), завдяки цьому об'єкту клієнтський код виконує методи классів та об'єктів.

```
class Scheduler {
    private let runLoop = RunLoop()
    private let thread: Thread

    init() {
        self.thread = Thread(target:runLoop,
                             selector:#selector(RunLoop.run),
                             object:nil)
        thread.start()
    }
     func schedule(event: @escaping Event) {
        runLoop.eventQueue.pushEvent(event: event)
```

```
    }

    func dispose() {
        runLoop.disposed = true
    }
}
```

Об'єкт планування. Це стандартний інтерфейс, для не прямого виклику, може бути реалізований у два способи: агрегат та базовий класс. У данному випадку, реалізовано останнім методом.

```
class SchedulableObject<T> {
    private let object: T
    private let scheduler: Scheduler

    init(object: T, scheduler: Scheduler) {
        self.object = object
        self.scheduler = scheduler
    }

    func schedule(event: @escaping (T) -> Void) {
        scheduler.schedule {
            event(self.object)
        }
    }
}
```

Розглянувши усі компоненти, та зрозумівши сутність методу вирішення поставленної задачі, можна винести два головних правила взаємодії об'єктів між собою та у середині планувальника.

1. Якщо клієнт об'єкта планування і об'єкт який викликаються мають зв'язок з одним і тим самим об'єктом планувальника, то метод визивається на пряму, простим чином і не потребує зайвих не прямих викликів.

2. Якщо клієнт об'єкта планування і об'єкт який викликаються мають зв'язки з різними планувальниками, то зв'язок та виклик методів виконується не прямо, методом відправки повідомлень та сигналів.

Заключение и выводы

Таким чином, за домогою цього метода ми досягнемо системного рішення проблеми важкозатратності бізнесс логіки у головному потоці данних. Отримана архітектура, буде дуже просто масштабуватися за двох причин. По перше, як вже було зазначено, кількість робочих потоків не залежить від кількості сервісів, по друге – вся тяжка праця багатопоточних процесів, перекладена с прикладних классів до інфраструктурних.

Литература:

1. [Электронный ресурс]. - Режим доступа: https://medium.com/ios-os-x-development/perfect-smooth-scrolling-in-uitableviews-fd609d5275a5

2. [Электронный ресурс]. - Режим доступа: https://github.com/pavelosipov/POSSchedulableObject

3. [Электронный ресурс]. - Режим доступа: https://habrahabr.ru/company/mailru/blog/317440/

4. [Электронный ресурс]. - Режим доступа: https://www.raywenderlich.com/124311/asyncdisplaykit-2-0-tutorial-getting-started

Статья отправлена: 05.04.2017 г.

ЦИТ: ua117-099
DOI: 10.21893/2415-7538.2016-05-1-099
УДК 528.9: 555.3/.9 (470.345)

Муженикова О.И., Манухова О. М., Шпак Д.Д.

СОЗДАНИЕ КАРТЫ МЕСТОРОЖДЕНИЙ ПОЛЕЗНЫХ ИСКОПАЕМЫХ РЕСПУБЛИКИ МОРДОВИЯ

Национальный исследовательский Мордовский государственный университет имени Н. П. Огарева, Саранск, ул. Большевистская, 68, 430005

UDC 528.9: 555.3/.9 (470.345)

Muzhenikova O. I., Manukova O. M., Shpak D. D.

CREATE A MAP OF MINERAL DEPOSITS IN THE REPUBLIC OF MORDOVIA

National Research Mordovia State University named by N. P. Ogarev Russia, Saransk, Str. Bolshevistskaya, 68, 430005

Аннотация. Указывается, что создание карты полезных ископаемых включает в себя не только определение тематики, структуры слоёв, но и оформление картографического изображения. Подчеркивается важность разработки условных знаков. Правильный выбор условных знаков обеспечит хорошую читаемость и наглядность карты.

Ключевые слова: условный знак, карта, геоинформационные технологии, полезные ископаемые, значок, форма.

Abstract. Specifies that the creation of a map of mineral resources includes not only the definition of the themes, structure of layers, but the design of maps. Stresses the importance of developing symbols. The right choice of symbols will provide a good readability and visibility maps.

Key words: symbol, map, GIS technology minerals, icon, form.

Интенсивное развитие информационных, геоинформационных и педагогических технологий диктует от конкурентоспособного на рынке труда выпускника — бакалавра географии — владения соответствующими знаниями, умениями, навыками. Информационная компетентность становится отличительным признаком качества образования [1–2;12;17-18]. Информационную компетентность студента определяется как качество личности, представляющее собой совокупность знаний, умений и ценностного отношения к эффективному осуществлению различных видов информационной деятельности и использованию новых информационных технологий для решения социально-значимых задач, возникающих в реальных ситуациях профессиональной и повседневной жизни человека в обществе [13].

В новом ФГОС ВО по направлению подготовки 05.03.02 «География» (уровень бакалавриата) формирование рассматриваемой информационной