Лекция №6

Хранение данных. Часть 2

Короткий Егор

образование

О чем будем говорить

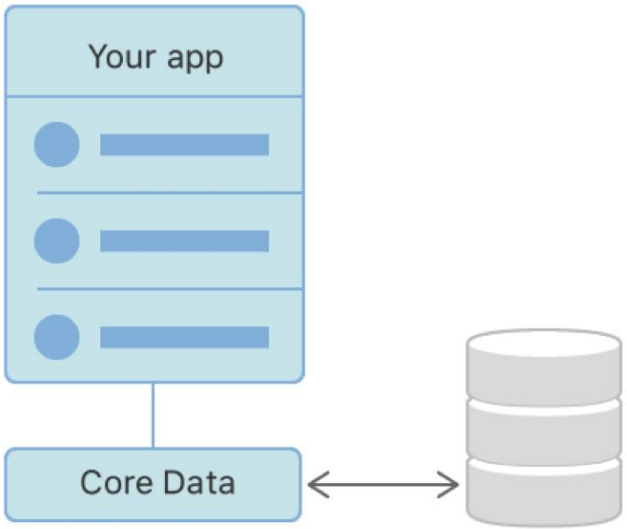
- CoreData
- Realm
- SQLite

CoreData

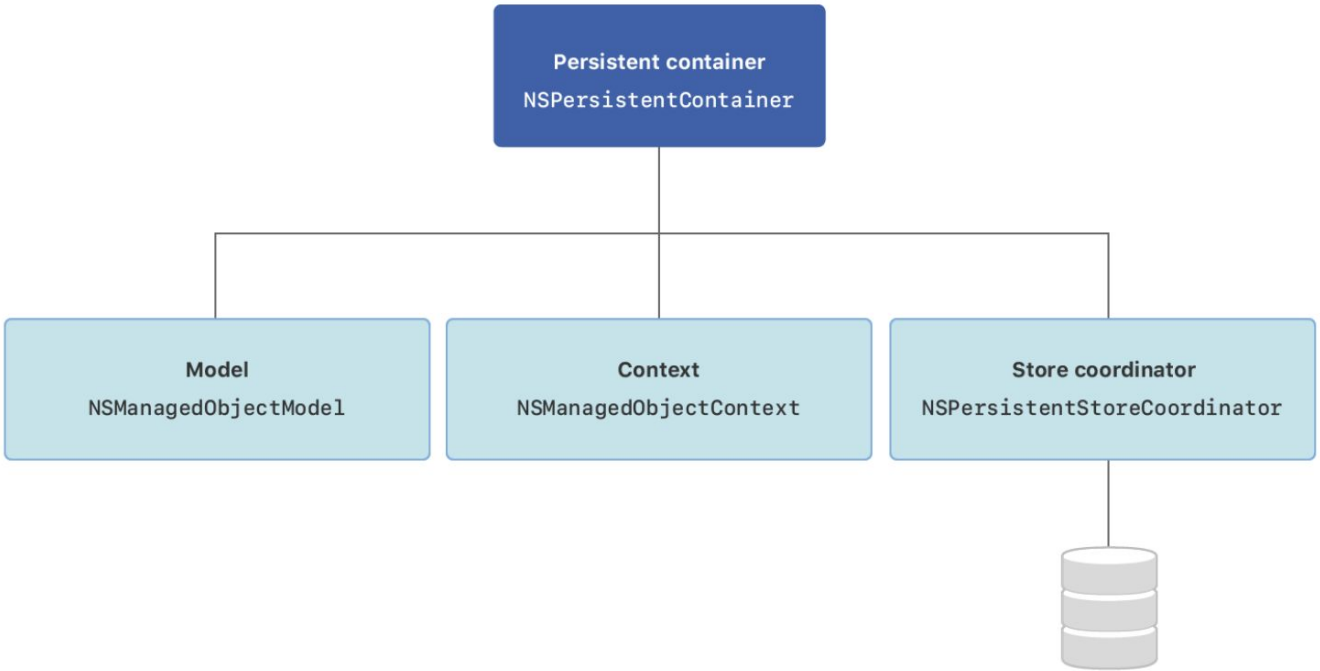
Хранение деревьев объектов

- хранилище
- не является реляционной БД
- несколько типов хранилища:
  - Sqlite
  - Xml
  - память

CoreData



CoreData


Persistent container
NSPersistentContainer
Model
NSManagedObjectModel
Context
NSManagedObjectContext
Store coordinator
NSPersistentStoreCoordinator

CoreData: основные понятия

- **NSManagedObjectModel** — описание структуры данных (объекты, их атрибуты, связи).
- **NSPersistentStore** — конкретная разновидность хранилища (**SQLite**, Binary, In-Memory + свои).
- **NSPersistentStoreCoordinator**
  - отвечает за создание/извлечение существующих данных из persistent store;
  - передаёт их запросившему контексту.
- **NSManagedObjectContext**
  - при извлечении объектов из persistent store создаётся объектный граф;
  - все изменения выполняются в контексте;
  - если контекст сохраняют, то изменения сохраняются в persistent store.

CoreData

- **NSManagedObject** — базовый класс для объектов
- **NSFetchRequest** — запрос на выборку объектов
- Все операции делаются через контекст
- Объекты нельзя передавать между потоками напрямую
    - Можно только через ID объекта (objectID)

CoreData CRUD

- **CREATE**
- **READ**
- **UPDATE**
- **DELETE**

CoreData. Чтение

```
func fetch<T>(request: NSFetchRequest<T>) -> [T] where T: NSManagedObject {
    return (try? viewContext.fetch(request)) ?? []
}
```

```
func count<T: NSManagedObject>(request: NSFetchRequest<T>) -> Int {
    return (try? viewContext.count(for: request)) ?? 0
}
```

CoreData. Создание

```
func create<T: NSManagedObject>(entityName: String, configureBlock: @escaping (T) -> ())  {
    viewContext.performAndWait {
        guard let object = NSEntityDescription.insertNewObject(forEntityName: entityName,
                                                                into: viewContext) as? T else {
            return
        }

        configureBlock(object)

        try? viewContext.save()
    }
}
```

CoreData. Обновление

```
func update<T>(request: NSFetchRequest<T>, configureBlock: @escaping (T) -> ()) where T: NSManagedObject {
    let objects = fetch(request: request)

    guard let object = objects.first else {
        return
    }

    configureBlock(object)

    viewContext.performAndWait {
        try? viewContext.save()
    }
}
```

CoreData. Удаление

```
func deleteAll(request: NSFetchRequest<NSFetchRequestResult>) {
    let batchRequest = NSBatchDeleteRequest(fetchRequest: request)

    viewContext.performAndWait {
        _ = try? viewContext.execute(batchRequest)
        try? viewContext.save()
    }
}
```

```
func delete<T>(request: NSFetchRequest<T>) where T: NSManagedObject {
    let objects = fetch(request: request)

    objects.forEach({ viewContext.delete($0) })
}
```

CoreData

NSFetchedResultsController

- Выбирает данные для представления в виде UITableView
- Поддерживает секции
- Реагирует на изменения NSFetchRequest и вызывает делегат с описанием этих изменений

CoreData FRC

```
private lazy var fetchedResultsController: NSFetchedResultsController<TestItem> = {
    let request: NSFetchRequest<TestItem> = TestItem.fetchRequest()

    let sortDescriptor = NSSortDescriptor(key: #keyPath(TestItem.value), ascending: true)
    request.sortDescriptors = [sortDescriptor]

    let frc = NSFetchedResultsController(fetchRequest: request,
                                         managedObjectContext: dataManager.mainQueueContext,
                                         sectionNameKeyPath: nil,
                                         cacheName: nil)
    frc.delegate = self

    return frc
}()
```

CoreData FRC. Делегат изменений

```
extension AutoFetchViewController: NSFetchedResultsControllerDelegate {

    func controllerWillChangeContent(_ controller: NSFetchedResultsController<NSFetchRequestResult>) {
        tableView.beginUpdates()
    }

    func controller(_ controller: NSFetchedResultsController<NSFetchRequestResult>, didChange anObject: Any, at indexPath: IndexPath?, for type: NSFetchedResultsChangeType,
        newIndexPath: IndexPath?) {
        let rowAnimation = UITableView.RowAnimation.automatic

        switch type {
        case .insert:
            if let newIndexPath_ = newIndexPath {
                tableView.insertRows(at: [newIndexPath_], with: rowAnimation)
            }
        case .delete:
            if let indexPath_ = indexPath {
                tableView.deleteRows(at: [indexPath_], with: rowAnimation)
            }
        case .update:
            if let indexPath_ = indexPath {
                tableView.reloadRows(at: [indexPath_], with: rowAnimation)
            }
        case .move:
            if let oldIndexPath = indexPath, let newIndexPath_ = newIndexPath {
                tableView.moveRow(at: oldIndexPath, to: newIndexPath_)
            }
        @unknown default:
            break
        }
    }

    func controllerDidChangeContent(_ controller: NSFetchedResultsController<NSFetchRequestResult>) {
        tableView.endUpdates()
    }

}
```

NSPredicate

Класс, предназначенный для задания требуемых условий фильтрации

Его можно создать:

- из строки с условием
- блоком кода, который отдаёт Bool в зависимости от того, подходит объект или нет
- как логическую комбинацию других предикатов
- подробнее см. ссылки

Миграции в CoreData

https://vk.cc/cfMHaH

Realm

- Тоже объектное хранилище
- Использует свой собственный высокопроизводительный бэкенд хранения
- Есть для ObjectiveC, Swift, Java, JS (если у вас swift + objc, то надо использовать objc-версию)

Realm

- **Object** — базовый класс для моделей данных
- Связи моделируют отдельным классом **List**
- Условия в запросах делаются тоже через **NSPredicate**
- Объекты realm не должны передаваться между потоками
- Аналог **NSFetchedResultController** — это нотификации

Realm. Базовые операции

```swift
import RealmSwift

class Person: Object {

    @objc dynamic var firstName = ""
    @objc dynamic var lastName = ""
}

extension Person {

    var fullName: String {
        firstName + " " + lastName
    }

}
```

```swift
private var dataSource: Results<Person>!

private func loadInitialData() {
    dataSource = try? Realm().objects(Person.self)
}
```

```swift
private func save(with text: String) {
    let components = text.components(separatedBy: " ")
    let person = Person()
    person.firstName = components.first ?? ""
    person.lastName = components.last ?? ""

    let realm = dataSource.realm
    try? realm?.write {
        realm?.add(person)
    }
}
```

Realm. Базовые операции

```
private var notificationToken: NotificationToken?

func observe() {
    notificationToken = dataSource.observe({ [weak self] changes in
        guard let self = self else { return }

        switch changes {
        case let .initial(results):
            self.dataSource = results
        case let .update(_, deletions, insertions, modifications):
            self.tableView.beginUpdates()
            self.tableView.deleteRows(at: deletions.map({ IndexPath(row: $0, section: 0)}),
                                      with: .automatic)
            self.tableView.insertRows(at: insertions.map({ IndexPath(row: $0, section: 0) }),
                                      with: .automatic)
            self.tableView.reloadRows(at: modifications.map({ IndexPath(row: $0, section: 0) }),
                                      with: .automatic)
            self.tableView.endUpdates()
        case let .error(error):
            fatalError(error.localizedDescription)
        }
    })
```

SQLite

- Низкий уровень
- C-API
- Реляционная база

Для облегчения работы есть несколько библиотек

Полезные ссылки

- https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/index.html#//apple_ref/doc/uid/TP40001075-CH2-SW1 — Apple CoreData guide
- https://realm.io/docs — документация Realm
- https://www.raywenderlich.com/7569-getting-started-with-core-data-tutorial — CoreData tutorial
- https://www.raywenderlich.com/9220-realm-tutorial-getting-started — Realm tutorial
- https://github.com/ccgus/fmdb — SQLite Obj-C wrapper
- https://github.com/stephencelis/SQLite.swift — SQLite Swift wrapper
- https://github.com/yapstudios/YapDatabase — YapDatabase
- https://nshipster.com/nspredicate/ — статья про NSPredicate
- https://habr.com/ru/post/351116/ — Keychain

QA

образование