Monday, October 31, 2016

CoreData Refresher

High Level Core Data Stack Overview

NSManagedObjectContext

NSManaged Object

Entity: Person

Λ

Managed Object Model V

A collection of entity descriptions < - > **NSPersistentStoreCoordinator**

Λ

V

NSPersistStore

Λ

V

SQLite

Λ

V

File System

red = inside NSPersistentContainer (iOS 10)

Ex: Entity: Person (NSManagedObject Entitites)

- Attributes
- name = PersonA
- age = 30
- Insert entity into the managed object context(but not yet saved).. call saveContext() to send to NSPersistentStore -> then to SQLite

- In iOS 10 is now inside the NSPersistentContainer
- Autogen iOS10 or set Non/Manual to then manually generate files

---

NSFetchedResultsController

- Works directly with CoreData to make it easier to make the fetched results when you make a fetch request.
- It collects the information of the results without having to bring all of it to memory at the same time.
- As you access, objects are placed in memory in batches to match the access patterns and object previously accessed are disposed of, keeps memory low.
- FRC work in conjunction with views with collection of objects such as table views and collection views. Their data source present their data in sections in rows.
- FRC can efficiently analyze the result request and precompute the results in the sections to save even more memory.
- All this is in the NSFetchedResultsControllerDelegate
- var fetchedResultsController: NSFetchedResultsController<Item>!

Easy Getter for Persistent Store's view context

- In appdelegate:
- let appDelegate = UIApplication.shared.delegate as! AppDelegate
- let context = ad.persistentContainer.viewContext

---

iOS 10 syntax for fetch requests

- let fetchRequest: NSFetchRequest<Item> = Item.fetchRequest()
- let someKindOfSort = NSSortDescriptor(key: "date", ascending: true)
- fetchRequest.sortDescriptors = [someKindOfSort]
- frc.delegate = self;
- do {

-

- try self.frc.performFetch()
- } catch {
- }

NSFetchedResultsControllerDelegate conforms to:

- controllerWillChangeContent
  - beginUpdates for collection
- controllerDidChangeContent
  - endUpdates for collection
- Listen for changes in data with…
- controller(_ controller: NSFetchedResultsController<NSFetchRequestResult>, didChange anObject: Any, at indexPath: IndexPath?, for type:
  - Then handle .insert/.delete/.update/.move for type

Saving a context

- let item = Item(context: context) // context from appdelegate
- now set item properties
- ad.saveContext   // app delegate save context
- **For saving with relationships:**
- item.toImage = image
- item.toStore = // store item here
- ad.saveContext

Fetch from context

- let fetchReq: NSFetchRequest<Store> = Store.fetchRequest()
- then do-try-catch, with self.stores = try context.fetch(fetchReq)

Removal

- get the item to delete, then
- context.delete(item)
- ad.saveContext() // save changes using app delegates saveContext()

-