Prática em Swift

Professor Pedro Henrique
prof.pedrohenrique.iossdk@gmail.com

Agenda

- Revisão Core Data Básico
  - NSFetchedResultsController;
- Core Data Avançado
  - Stack para ir além do básico;
  - Lightweight Migration (adicionando uma nova versão do modelo)

Core Data

- O Core Data é a abordagem padrão para banco de dados local na plataforma iOS;
  - Consiste de um framework extremamente poderoso, que fornece um banco de dados orientado à objetos e uma API igualmente poderosa para interação com o banco.

Core Data

- Oferece um meio para criar sua árvore de objetos suportada por um banco de dados;
- Na frieza dos bits, os dados podem ser armazenados em SQLite, XML ou mesmo em memória, mas, na maioria dos casos, isso é transparente para o programador.

Como funciona o Core Data?

- Em quatro passos básicos:

1. Criar um mapeamento visual, usando o Xcode, onde a relação entre entidades de objetos é definida;
2. Criar objetos (registros) e queries através de API orientada à objetos;
3. Acessar as ”colunas” da ”tabela” usando as propriedades do objeto (registro);
4. Ser feliz! (ou partir para o avançado, caso seja necessário)

Core Data

- Para acessar todas as coisas maravilhosas do Core Data, precisamos de um objeto especial chamado **contexto**: o **NSManagedObjectContext**.
  - Ele é o centralizador de todas as interações com o Core Data.

Quando a gente deixa o básico de lado, podemos acabar precisando de mais de uma instância de contexto. Vamos ver em breve!

Core Data - CRUD

- Create
- Retrieve
- Update
- Delete

Core Data - Create

```
if let entitdadeUsuario = NSEntityDescription
        .entityForName("Usuario", inManagedObjectContext: ctx) {

    let usuario = Usuario(entity: entitdadeUsuario, insertIntoManagedObjectContext: ctx)

    usuario.name = "Pedro"
    //preencher as propriedades da entitdade...


    do {
        try ctx.save()
    }catch let error {
        print("Erro ao salvar contexto: \(error)")
    }

}
```

Core Data - Retrieve

```
let fetchRequest = NSFetchRequest(entityName: "Usuario")
fetchRequest.predicate = NSPredicate(format: "name ==[cd] %@", "pedro")
let ordenarPorNome = NSSortDescriptor(key: "name", ascending: true)
fetchRequest.sortDescriptors = [ordenarPorNome]
fetchRequest.resultType = .ManagedObjectResultType

do {
    let resultado = try ctx.executeFetchRequest(fetchRequest) as! [Usuario]

    print("Foram obtidos \(resultado.count) resultados.")

}catch let error {
    print("Erro ao executar fetch request: \(error)")
}
```

- http://nshipster.com/nspredicate/
- https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSPredicate_Class/
- https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/Predicates/Articles/pSyntax.html

Core Data - Update

```
for usr in resultado {
    usr.name = usr.name?.stringByAppendingString(" alterado hoje")
}

do {
    try ctx.save()
}catch let error {
    print("Erro ao salvar contexto: \(error)")
}
```

O update consiste, basicamente, em alterar um ou mais objetos persistentes e salvar o contexto em seguida.

Core Data - Delete

```
ctx.deleteObject(usr)
do {
    try ctx.save()
}catch let error {
    print("Erro ao salvar contexto: \(error)")
}
```

Assim como no update, no delete, basta deletar o objeto e salvar o contexto em seguida.

NSFetchedResultsController

- Quando se programa para OSX, existe a facilidade do data-binding;
- Para o iOS, entretanto, esta facilidade não está disponível;
- Suprindo esta falta, a Apple nos oferece o NSFetchedResultsController para fazer a ligação da camada Model com a camada View.
  - Core Data e UITableView, por exemplo.

NSFetchedResultsController

- Oferece um enorme ganho de performance quando se lida com um volume maior de dados;
  - Os dados são carregados do Modelo para a View de forma preguiçosa (*lazy load*), ou seja, apenas os dados exibidos na tela são carregados do banco;
- Oferece uma série de facilidades para lidar com as operações do CRUD e atualizar a View em tempo real;
- É fácil de usar!

Links do ❤️

- https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreData/nsfetchedresultscontroller.html
- https://www.raywenderlich.com/999/core-data-tutorial-for-ios-how-to-use-nsfetchedresultscontroller
- http://www.learncoredata.com/nsfetchedresultscontroller-swift/

Core Data Avançado

- Indo além do básico;
- Migração de versão do modelo (lightweight);

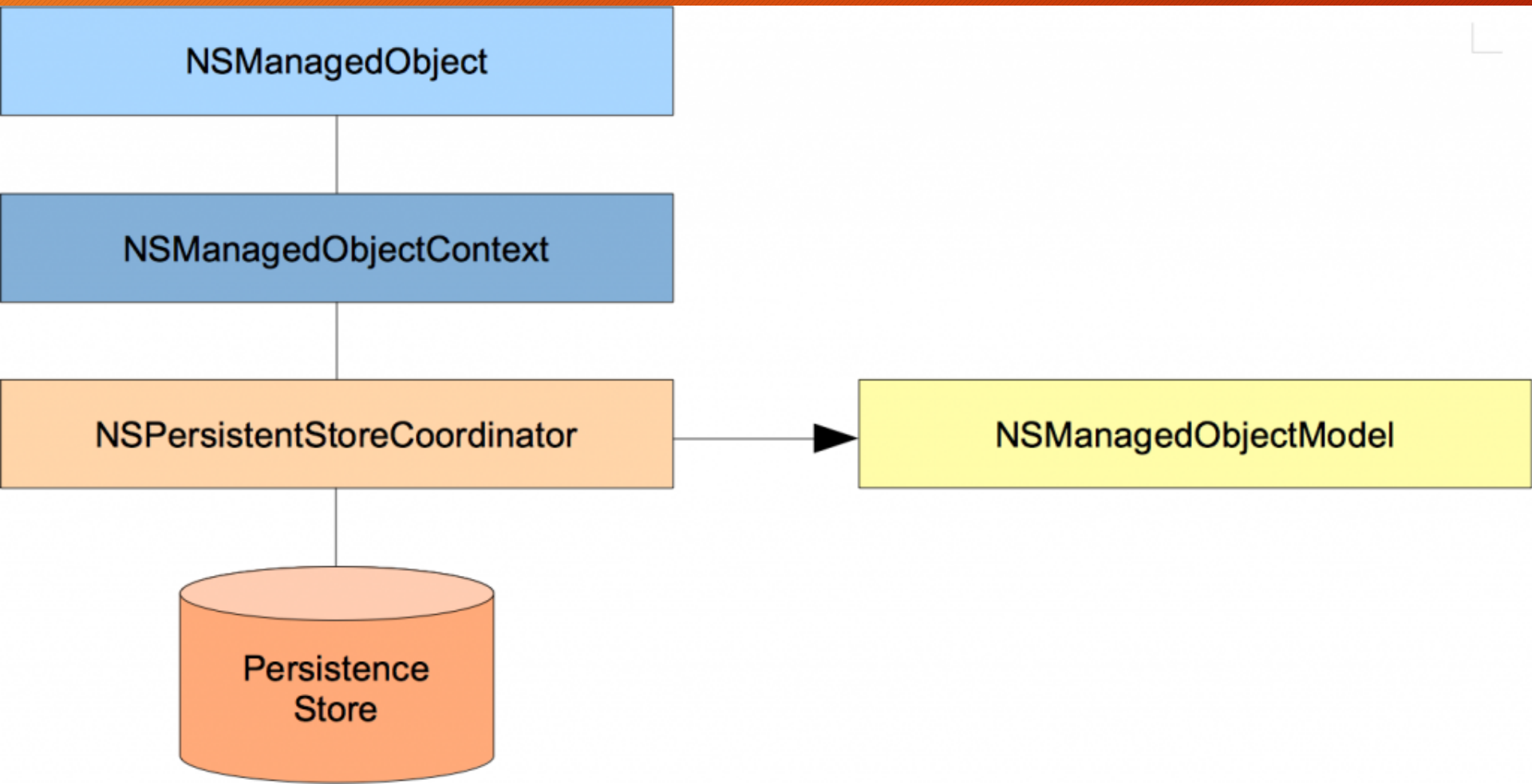
Core Data Avançado

- Por padrão, quando se cria o projeto no Xcode, a stack de Core Data que vem implementada é feita para funcionar apenas na Main Queue (thread principal);
- Isso é bom, quando o aplicativo realiza 100% das operações no Core Data com a interação do usuário, ou seja, na thread principal;
- Isso é muito ruim, caso o aplicativo tenha a necessidade de realizar processamento em background.

Core Data Avançado

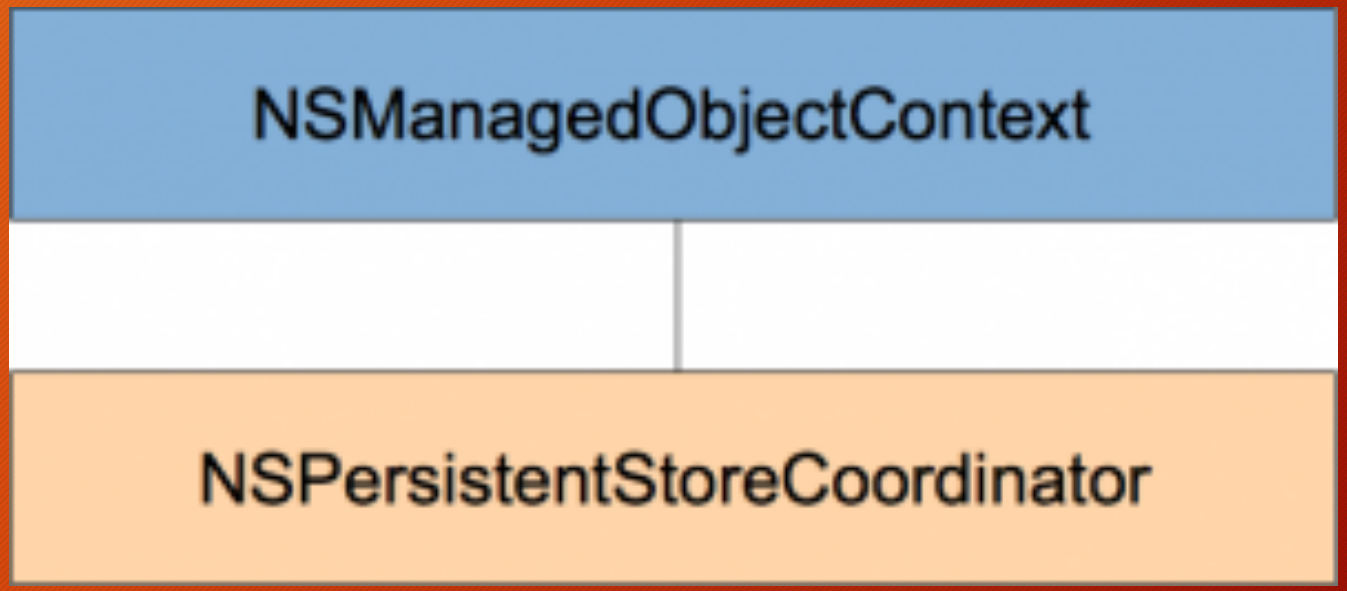
- Para resolver o problema de trabalhar em múltiplas threads com o Core Data, exitem algumas alternativas;
- Mas antes, vamos ver como o básico funciona:

Arquitetura Básica do Core Data



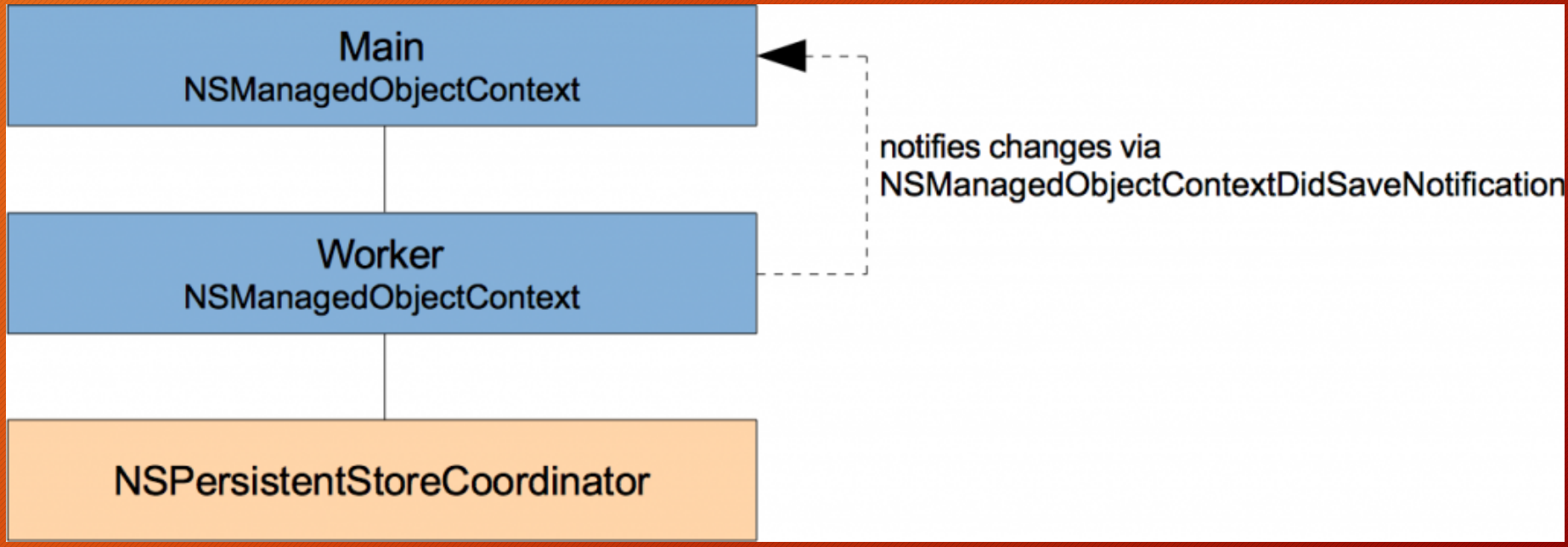
Stack mais comum para usar o Core Data

- Tudo na main thread;
- Esta opção, vocês já sabem fazer.



Stack Avançada para lidar com trabalho em background e na thread principal

- Trabalhar com a hierarquia de contextos (novidade que surgiu no iOS 5);
- Vamos ver esta opção na prática.



Core Data Avançado

- Não são apenas estas as opções!
- Vejam estes artigos que falam a respeito:

- https://blog.codecentric.de/en/2014/11/concurrency-coredata/

- http://martiancraft.com/blog/2015/03/core-data-stack/

- http://floriankugler.com/2013/04/29/concurrent-core-data-stack-performance-shootout/

Core Data Avançado

- Qual é a melhor forma de fazer?

Core Data Avançado

- Tudo vai depender da complexidade das soluções adotadas no desenvolvimento do seu aplicativo.
- Para muitas situações, o básico já é bom o suficiente;
- Para outras, pode ser necessário ir além do básico e implementar alguma das técnicas descritas nos artigos (ou a que vamos fazer juntos hoje);
- Para outras, ainda, pode ser necessário criar algo novo.

Lightweight Migration

- Quando precisamos modificar o modelo de dados de um aplicativo que já foi lançado na loja;

Lightweight Migration

- Quando a migração leve funciona?
  - Quando o iOS consegue ser capaz de inferir as mudanças no modelo, ou seja:

  - Adicionou ou removeu: atributo, entidade ou relacionamento;
  - Tornou um atributo obrigatório, com valor padrão;
  - Tornou um atributo não obrigatório;
  - Renomeou uma entidade com o uso de um identificador de renomeação;

Tipos de Migração

- Lightweight Migrations (ideal)
  - Requer o mínimo esforço por parte do programador. O trabalho se resume em criar o novo modelo e dar alguns cliques. A migração acontece de forma automágica;
- Manual Migrations
  - Ainda com a ajuda do Xcode, aqui você precisa especificar como os seus dados serão migrados, majoritariamente através de um arquivo do tipo Mapping Model (onde é descrito o de-para)
- Custom Manual Migrations
  - Mesma coisa do anterior, adicionando a capacidade de incluir transformação de dados no meio do processo. Além do arquivo Mapping Model, é necessário implementar uma classe do tipo NSEntityMigrationPolicy para fazer as transormações
- Fully Manual Migrations
  - Como o nome já diz, é quando a migração de dados é feita sem qualquer ajuda do ferramental do Xcode. Ou seja, você precisa programar tudo, desde de detectar a versão da base até aplicar todas as mudanças necessárias.

Links do ❤️

- https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreDataVersioning/Articles/vmLightweightMigration.html
- https://www.raywenderlich.com/27657/how-to-perform-a-lightweight-core-data-migration
- https://www.raywenderlich.com/114084/core-data-migrations-tutorial-lightweight-migrations
- http://code.tutsplus.com/tutorials/core-data-and-swift-migrations--cms-25084

Vamos Exercitar!