INCETRO

Что такое паттерн DAO и почему вы его полюбите

INCETRO

Дмитрий Савинов

iOS Developer

INCETRO

План

1. Проблематика
2. Определение и основные понятия
3. Ставим задачу
4. Подходы к решению поставленной задачи
5. DAO изнутри
6. DAO снаружи
7. Рассмотрим разные БД
8. Пошагово разберем как внедрить DAO
9. Подведем итоги

Знакомство с DAO

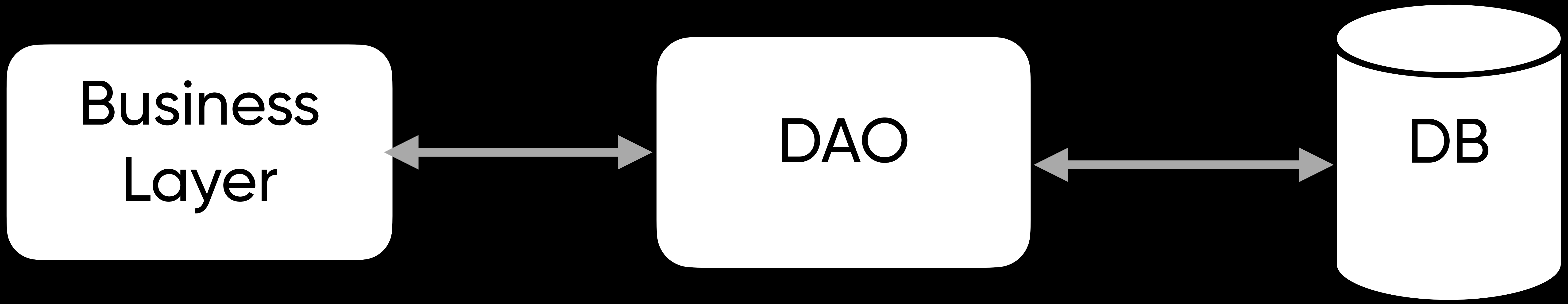
INCETRO

🤔🤨🧐

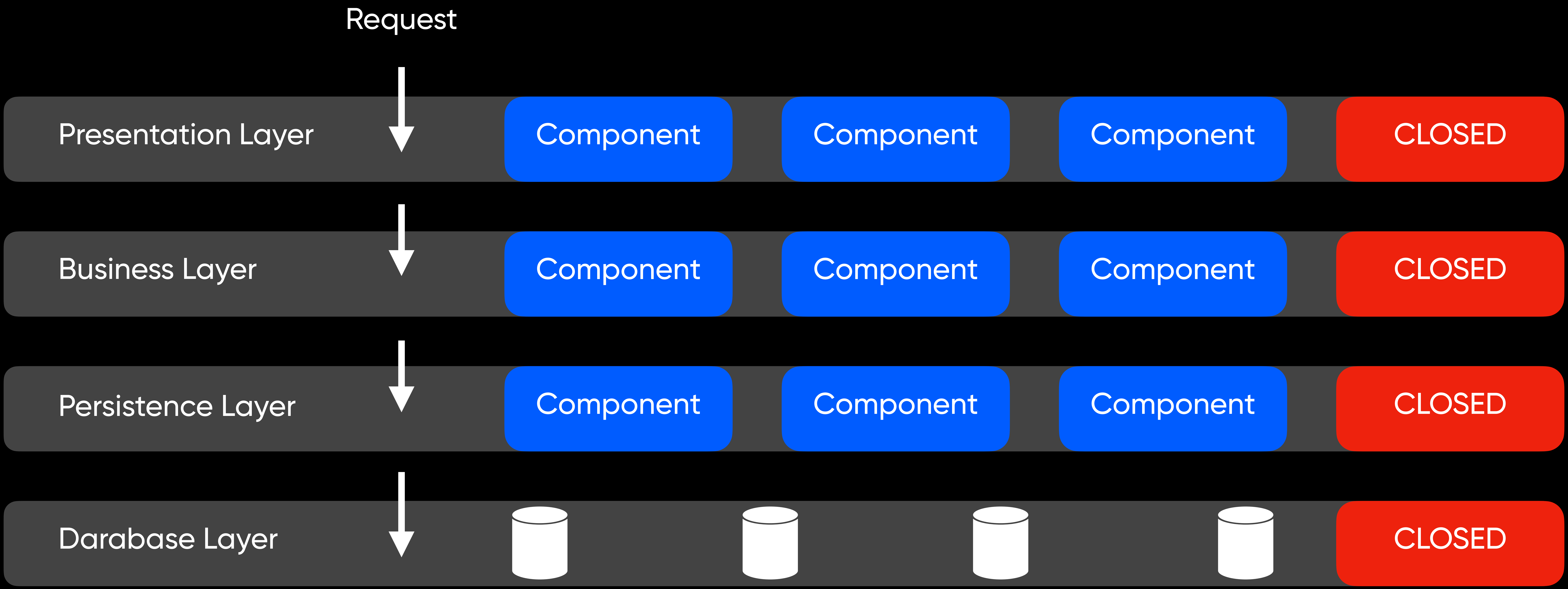
А зачем оно нам надо?

INCETRO

Да кто такой этот ваш DAO?



Persistence Layer



INCETRO

PlainObject

Plain Object — кто это такой?

```swift
// MARK: - UserPlainObject

public struct UserPlainObject {

    // MARK: - Properties

    public let id: Int
    public let name: String
    public let bio: String?
    public let username: String
    public let email: String?
    public let phone: String
    public let birthDate: Date?
    public let gender: Gender?
}
```

INCETRO

Plain Object — кто это такой?

```swift
// MARK: - UserPlainObject

public struct UserPlainObject {

    // MARK: - Properties

    public let id: Int
    public let name: String
    public let bio: String?
    public let username: String
    public let email: String?
    public let phone: String
    public let birthDate: Date?
    public let gender: Gender?
}
```

Пример

```swift
// MARK: - ProjectPlainObject

public struct ProjectPlainObject {

    // MARK: - Properties

    public let id: Int
    public let name: String
    public let workers: [ProjectWorkerPlainObject]
    public let roles: [RolePlainObject]
    public let budget: Int
    public let isArchived: Bool
    public let workTime: Double
    public let averageRate: Double
    public let currency: String
}
```

Еще один пример

```swift
// MARK: - CommentPlainObject

public struct CommentPlainObject {

    // MARK: - Properties

    public let id: Int
    public let imageURL: URL?
    public let username: String
    public let content: String
    public let likesCount: Int
    public let isLiked: Bool
    public let publicationDate: Date
}
```

INCETRO

PlainObject – друг?

INCETRO

Конечно, да

structure > class

let = безопасность

INCETRO

Организуем PersistanceLayer

INCETRO

Организуем свой Persistance Layer

```
// MARK: - DialogPlainObject

public struct DialogPlainObject {

    // MARK: - Properties

    public let id: Int
    public let isPinned: Bool
    public let messages: [MessagePlainObject]
}
```

INCETRO

Организуем свой Persistance Layer

```
// MARK: - DialogPlainObject

public struct DialogPlainObject {

    // MARK: - Properties

    public let id: Int
    public let isPinned: Bool
    public let messages: [MessagePlainObject]
}
```

Организуем свой Persistance Layer

```swift
// MARK: - MessagePlainObject

public struct MessagePlainObject {

    // MARK: - Properties

    public let id: Int
    public let date: Date
    public let text: String
    public let senderId: Int
    public let receiverId: Int
    public let type: Int
    public let isIncoming: Bool
    public let isRead: Bool
}
```

INCETRO

Есть 3 пути

INCETRO

Путь первый

Отказаться от Plain и сделать мутабельную модель

```swift
// MARK: - DialogModelObject

final class DialogModelObject: Object {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var isPinned = false
    let messages = List<MessageModelObject>()

    // MARK: - Object

    override class func primaryKey() -> String? {
        "id"
    }
}
```

```swift
// MARK: - MessageModelObject

final class MessageModelObject: Object {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var date = Date()
    @objc dynamic var text = ""
    @objc dynamic var receiverId = 0
    @objc dynamic var senderId = 0
    @objc dynamic var type = 0
    @objc dynamic var isIncoming = false
    @objc dynamic var isRead = false

    // MARK: - Object

    override class func primaryKey() -> String? {
        "id"
    }
}
```

INCETRO

Путь первый

Отказаться от Plain и сделать мутабельную модель

```swift
// MARK: - DialogModelObject

final class DialogModelObject: Object {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var isPinned = false
    let messages = List<MessageModelObject>()

    // MARK: - Object

    override class func primaryKey() -> String? {
        "id"
    }
}
```

```swift
// MARK: - MessageModelObject

final class MessageModelObject: Object {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var date = Date()
    @objc dynamic var text = ""
    @objc dynamic var receiverId = 0
    @objc dynamic var senderId = 0
    @objc dynamic var type = 0
    @objc dynamic var isIncoming = false
    @objc dynamic var isRead = false

    // MARK: - Object

    override class func primaryKey() -> String? {
        "id"
    }
}
```

INCETRO

Путь первый

```swift
let realm = try! Realm()

let dialogs = realm.objects(DialogModelObject.self)

try! realm.write {
    dialogs[0].isPinned = true
}
```

Путь первый

```
let realm = try! Realm()

let dialogs = realm.objects(DialogModelObject.self)

try! realm.write {
    dialogs[0].isPinned = true
}
```

INCETRO

Путь первый

```
let realm = try! Realm()

let dialogs = realm.objects(DialogModelObject.self)

try! realm.write {
    dialogs[0].isPinned = true
}
```

INCETRO

Путь первый

```swift
struct ContentView: View {

    @State var dialogs: [DialogModelObject]

    var body: some View {
        Form {
            ForEach(dialogs) { dialog in
                DialogView(dialog)
            }
        }
    }
}
```

Осталось 2 пути

INCETRO

Путь второй

Сделать отдельную модель для базы и при необходимости преобразовывать модель базы в плейн и наоборот

```swift
// MARK: - DialogPlainObject

public struct DialogPlainObject {

    // MARK: - Properties

    public let id: Int
    public let isPinned: Bool
    public let messages: [MessagePlainObject]
}
```

↔

```swift
// MARK: - DialogModelObject

final class DialogModelObject: Object {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var isPinned = false
    let messages = List<MessageModelObject>()

    // MARK: - Object

    override class func primaryKey() -> String? {
        "id"
    }
}
```

INCETRO

Путь второй

```swift
// MARK: - MessagePlainObject

public struct MessagePlainObject {

    // MARK: - Properties

    public let id: Int
    public let date: Date
    public let text: String
    public let senderId: Int
    public let receiverId: Int
    public let type: Int
    public let isIncoming: Bool
    public let isRead: Bool
}
```

⟷

```swift
// MARK: - MessageModelObject

final class MessageModelObject: Object {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var date = Date()
    @objc dynamic var text = ""
    @objc dynamic var receiverId = 0
    @objc dynamic var senderId = 0
    @objc dynamic var type = 0
    @objc dynamic var isIncoming = false
    @objc dynamic var isRead = false

    // MARK: - Object

    override class func primaryKey() -> String? {
        "id"
    }
}
```

INCETRO

Путь второй

```swift
// MARK: - MessageModelObject

final class MessageModelObject: RealmModel {

    ...

    func toPlainObject() -> MessagePlainObject {
        MessagePlainObject(
            id: id,
            date: date,
            text: text,
            senderId: senderId,
            receiverId: receiverId,
            type: type,
            isIncoming: isIncoming,
            isRead: isRead
        )
    }
}
```

```swift
// MARK: - MessagePlainObject

public struct MessagePlainObject {

    ...

    public func toDatabaseModel() -> MessageModelObject {
        let messageModel = MessageModelObject()
        messageModel.id = id
        messageModel.date = date
        messageModel.text = text
        messageModel.senderId = senderId
        messageModel.receiverId = receiverId
        messageModel.type = type
        messageModel.isIncoming = isIncoming
        messageModel.isRead = isRead
        return messageModel
    }
}
```

Путь второй

```swift
// MARK: - DialogServiceImplementation

public final class DialogServiceImplementation: DialogService {

    // MARK: - Properties

    private let realm: Realm

    // MARK: - Initializers

    public init(realm: Realm) {
        self.realm = realm
    }

    public func obtainDialogs() -> [DialogPlainObject] {
        realm.objects(DialogModelObject.self).map { $0.toPlainObject() }
    }

    public func persistDialog(dialog: DialogPlainObject) throws {
        try realm.write {
            realm.add(dialog.toDatabaseModel(), update: .modified)
        }
    }
}
```

INCETRO

Путь второй

```swift
// MARK: - DialogServiceImplementation

public final class DialogServiceImplementation: DialogService {

    // MARK: - Properties

    private let realm: Realm

    // MARK: - Initializers

    public init(realm: Realm) {
        self.realm = realm
    }

    public func obtainDialogs() -> [DialogPlainObject] {
        realm.objects(DialogModelObject.self).map { $0.toPlainObject() }
    }

    public func persistDialog(dialog: DialogPlainObject) throws {
        try realm.write {
            realm.add(dialog.toDatabaseModel(), update: .modified)
        }
    }
}
```

INCETRO

Путь второй

```swift
// MARK: - DialogServiceImplementation

public final class DialogServiceImplementation: DialogService {

    // MARK: - Properties

    private let realm: Realm

    // MARK: - Initializers

    public init(realm: Realm) {
        self.realm = realm
    }

    public func obtainDialogs() -> [DialogPlainObject] {
        realm.objects(DialogModelObject.self).map { $0.toPlainObject() }
    }

    public func persistDialog(dialog: DialogPlainObject) throws {
        try realm.write {
            realm.add(dialog.toDatabaseModel(), update: .modified)
        }
    }
}
```

INCETRO

Путь второй

```swift
// MARK: - DialogServiceImplementation

public final class DialogServiceImplementation: DialogService {

    // MARK: - Properties

    private let realm: Realm

    // MARK: - Initializers

    public init(realm: Realm) {
        self.realm = realm
    }

    public func obtainDialogs() -> [DialogPlainObject] {
        realm.objects(DialogModelObject.self).map { $0.toPlainObject() }
    }

    public func persistDialog(dialog: DialogPlainObject) throws {
        try realm.write {
            realm.add(dialog.toDatabaseModel(), update: .modified)
        }
    }
}
```

INCETRO

Есть только 1 путь

INCETRO

DAO

INCETRO

Путь воина

Воспользоваться DAO

```swift
// MARK: - DialogPlainObject

public struct DialogPlainObject: Plain {

    // MARK: - Plain

    public var uniqueId: UniqueID {
        UniqueID(value: id)
    }

    // MARK: - Properties

    public let id: Int
    public let isPinned: Bool
    public let messages: [MessagePlainObject]
}
```

⟷

```swift
// MARK: - DialogModelObject

final class DialogModelObject: RealmModel {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var isPinned = false
    let messages = List<MessageModelObject>()
}
```

INCETRO

Путь воина

```swift
// MARK: - MessagePlainObject

public struct MessagePlainObject {

    ...

    public func toDatabaseModel() -> MessageModelObject {
        let messageModel = MessageModelObject()
        messageModel.id = id
        messageModel.date = date
        messageModel.text = text
        messageModel.senderId = senderId
        messageModel.receiverId = receiverId
        messageModel.type = type
        messageModel.isIncoming = isIncoming
        messageModel.isRead = isRead
        return messageModel
    }
}
```

⟷

```swift
// MARK: - MessageModelObject

final class MessageModelObject: RealmModel {

    ...

    func toPlainObject() -> MessagePlainObject {
        MessagePlainObject(
            id: id,
            date: date,
            text: text,
            senderId: senderId,
            receiverId: receiverId,
            type: type,
            isIncoming: isIncoming,
            isRead: isRead
        )
    }
}
```

INCETRO

Проблема методов to...

```swift
func toDatabaseModel() -> MessageModelObject {
    let configuration = AssembliesHolder.container.resolve(RealmConfiguration.self).unwrap()
    let messageStorage = RealmStorage<MessageModelObject>(configuration: configuration)
    let messageModel = try! messageStorage.read(byPrimaryKey: uniqueId.rawValue) ?? MessageModelObject()
    if messageModel.uniqueId.isEmpty {
        messageModel.uniqueId = uniqueId.rawValue
    }
    messageModel.id = id
    messageModel.date = date
    messageModel.text = text
    messageModel.senderId = senderId
    messageModel.receiverId = receiverId
    messageModel.type = type
    messageModel.isIncoming = isIncoming
    messageModel.isRead = isRead
    return messageModel
}
```

INCETRO

Проблема методов to...

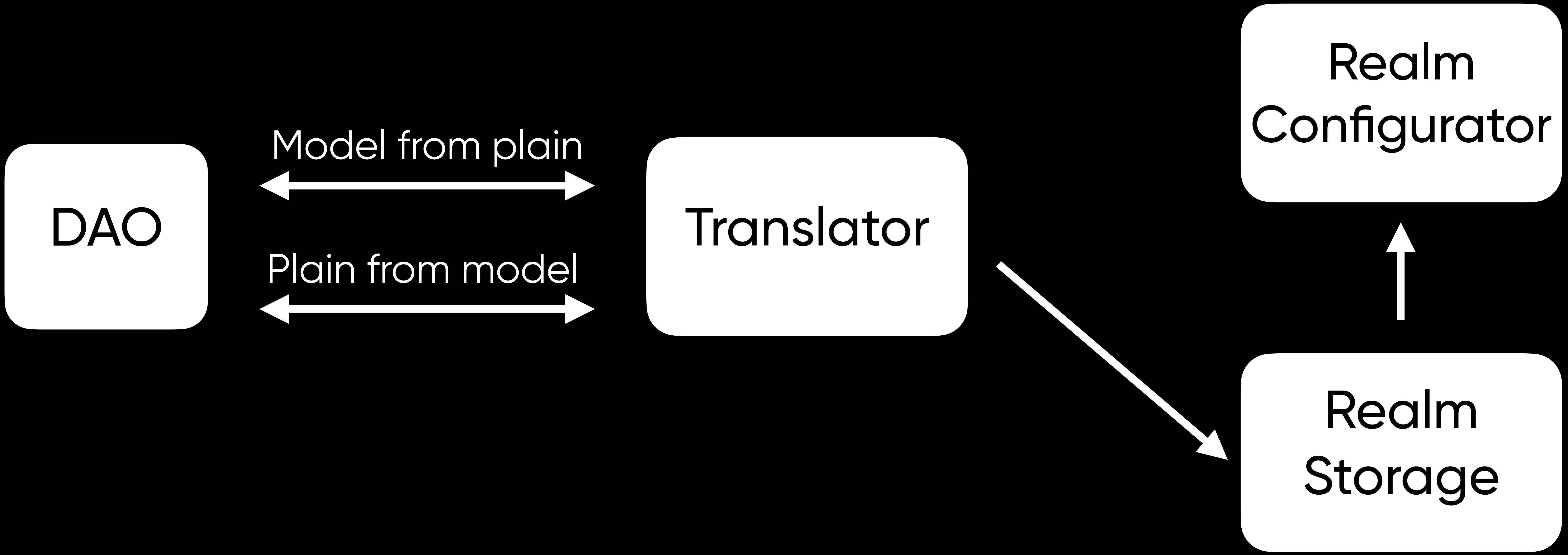
```swift
func toDatabaseModel() -> MessageModelObject {
    let configuration = AssembliesHolder.container.resolve(RealmConfiguration.self).unwrap()
    let messageStorage = RealmStorage<MessageModelObject>(configuration: configuration)
    let messageModel = try! messageStorage.read(byPrimaryKey: uniqueId.rawValue) ?? MessageModelObject()
    if messageModel.uniqueId.isEmpty {
        messageModel.uniqueId = uniqueId.rawValue
    }
    messageModel.id = id
    messageModel.date = date
    messageModel.text = text
    messageModel.senderId = senderId
    messageModel.receiverId = receiverId
    messageModel.type = type
    messageModel.isIncoming = isIncoming
    messageModel.isRead = isRead
    return messageModel
}
```

INCETRO

Translator

INCETRO

Да кто такой этот ваш DAO?



Translator – пример

```swift
// MARK: - MessageTranslator

final class MessagesTranslator {

    // MARK: - Aliases

    typealias PlainModel = MessagePlainObject
    typealias DatabaseModel = MessageModelObject

    // MARK: - Properties

    private lazy var messageStorage = RealmStorage<MessageModelObject>(configuration: self.configuration)
    private let configuration: RealmConfiguration

    // MARK: - Initializers

    public init(configuration: RealmConfiguration) {
        self.configuration = configuration
    }
}
```

Translator – пример

```
// MARK: - MessageTranslator

final class MessagesTranslator {

    // MARK: - Aliases

    typealias PlainModel = MessagePlainObject
    typealias DatabaseModel = MessageModelObject

    // MARK: - Properties

    private lazy var messageStorage = RealmStorage<MessageModelObject>(configuration: self.configuration)
    private let configuration: RealmConfiguration

    // MARK: - Initializers

    public init(configuration: RealmConfiguration) {
        self.configuration = configuration
    }
}
```

INCETRO

Translator – пример

```swift
// MARK: - Translator

extension MessagesTranslator: Translator {

    func translate(model: DatabaseModel) throws -> PlainModel {
        MessagePlainObject(
            id: Int(model.uniqueId) ?? 0,
            date: model.date,
            text: model.text,
            senderId: model.senderId,
            receiverId: model.receiverId,
            type: model.type,
            isIncoming: model.isIncoming,
            isRead: model.isRead
        )
    }

    ...
}
```

INCETRO

Translator – пример

```
// MARK: - Translator

extension MessagesTranslator: Translator {

    ...

    func translate(plain: PlainModel) throws -> DatabaseModel {
        let model = try messageStorage.read(byPrimaryKey: plain.uniqueId.rawValue) ?? DatabaseModel()
        try translate(from: plain, to: model)
        return model
    }

    func translate(from plain: PlainModel, to databaseModel: DatabaseModel) throws {
        if databaseModel.uniqueId.isEmpty {
            databaseModel.uniqueId = plain.uniqueId.rawValue
        }
        databaseModel.date = plain.date
        databaseModel.isIncoming = plain.isIncoming
        databaseModel.isRead = plain.isRead
        databaseModel.senderId = plain.senderId
        databaseModel.text = plain.text
        databaseModel.type = plain.type
        databaseModel.receiverId = plain.receiverId
    }
}
```

INCETRO

Translator — пример с вложенностью

```swift
// MARK: - DialogPlainObject

public struct DialogPlainObject {

    // MARK: - Properties

    public let id: Int
    public let isPinned: Bool
    public let messages: [MessagePlainObject]
}
```

INCETRO

Translator – пример с вложенностью

```swift
// MARK: - Translator

extension DialogTranslator: Translator {

    func translate(model: DatabaseModel) throws -> PlainModel {
        DialogPlainObject(
            id: Int(model.uniqueId) ?? 0,
            isPinned: model.isPinned,
            messages: try MessagesTranslator(configuration: configuration).translate(
                models: Array(model.messages)
            )
        )
    }

    func translate(plain: PlainModel) throws -> DatabaseModel {
        let model = try dialogStorage.read(byPrimaryKey: plain.uniqueId.rawValue) ?? DatabaseModel()
        try translate(from: plain, to: model)
        return model
    }

    func translate(from plain: PlainModel, to databaseModel: DatabaseModel) throws {
        if databaseModel.uniqueId.isEmpty {
            databaseModel.uniqueId = plain.uniqueId.rawValue
        }
        databaseModel.isPinned = plain.isPinned
        databaseModel.messages.removeAll()
        databaseModel.messages.append(
            objectsIn: try MessagesTranslator(configuration: configuration).translate(
                plains: plain.messages
            )
        )
    }
}
```

INCETRO

Translator — пример с вложенностью

```swift
// MARK: - Translator

extension DialogTranslator: Translator {

    func translate(model: DatabaseModel) throws -> PlainModel {
        DialogPlainObject(
            id: Int(model.uniqueId) ?? 0,
            isPinned: model.isPinned,
            messages: try MessagesTranslator(configuration: configuration).translate(
                models: Array(model.messages)
            )
        )
    }

    func translate(plain: PlainModel) throws -> DatabaseModel {
        let model = try dialogStorage.read(byPrimaryKey: plain.uniqueId.rawValue) ?? DatabaseModel()
        try translate(from: plain, to: model)
        return model
    }

    func translate(from plain: PlainModel, to databaseModel: DatabaseModel) throws {
        if databaseModel.uniqueId.isEmpty {
            databaseModel.uniqueId = plain.uniqueId.rawValue
        }
        databaseModel.isPinned = plain.isPinned
        databaseModel.messages.removeAll()
        databaseModel.messages.append(
            objectsIn: try MessagesTranslator(configuration: configuration).translate(
                plains: plain.messages
            )
        )
    }
}
```

INCETRO

Translator – пример с вложенностью

```swift
// MARK: - Translator

extension DialogTranslator: Translator {

    func translate(model: DatabaseModel) throws -> PlainModel {
        DialogPlainObject(
            id: Int(model.uniqueId) ?? 0,
            isPinned: model.isPinned,
            messages: try MessagesTranslator(configuration: configuration).translate(
                models: Array(model.messages)
            )
        )
    }

    func translate(plain: PlainModel) throws -> DatabaseModel {
        let model = try dialogStorage.read(byPrimaryKey: plain.uniqueId.rawValue) ?? DatabaseModel()
        try translate(from: plain, to: model)
        return model
    }

    func translate(from plain: PlainModel, to databaseModel: DatabaseModel) throws {
        if databaseModel.uniqueId.isEmpty {
            databaseModel.uniqueId = plain.uniqueId.rawValue
        }
        databaseModel.isPinned = plain.isPinned
        databaseModel.messages.removeAll()
        databaseModel.messages.append(
            objectsIn: try MessagesTranslator(configuration: configuration).translate(
                plains: plain.messages
            )
        )
    }
}
```

INCETRO

DAO

INCETRO

DAO

```swift
/// Create DAO instance
let dao = DAO(
    storage: RealmStorage<DialogModelObject>(configuration: configuration),
    translator: DialogTranslator(configuration: configuration)
)

/// Obtain dialogs (from backend for example)
let dialogs = dialogService.obtainDialogs()

/// Save dialogs
try dao.persist(dialogs)

/// Obtain some info from database
let count = try dao.read().count

/// Erase all dialogs from database
try dao.erase()
```

INCETRO

DAO

```
/// Create DAO instance
let dao = DAO(
    storage: RealmStorage<DialogModelObject>(configuration: configuration),
    translator: DialogTranslator(configuration: configuration)
)

/// Obtain dialogs (from backend for example)
let dialogs = dialogService.obtainDialogs()

/// Save dialogs
try dao.persist(dialogs)

/// Obtain some info from database
let count = try dao.read().count

/// Erase all dialogs from database
try dao.erase()
```

INCETRO

DAO

```
/// Create DAO instance
let dao = DAO(
    storage: RealmStorage<DialogModelObject>(configuration: configuration),
    translator: DialogTranslator(configuration: configuration)
)

/// Obtain dialogs (from backend for example)
let dialogs = dialogService.obtainDialogs()

/// Save dialogs
try dao.persist(dialogs)

/// Obtain some info from database
let count = try dao.read().count

/// Erase all dialogs from database
try dao.erase()
```

INCETRO

DAO

```
/// Create DAO instance
let dao = DAO(
    storage: RealmStorage<DialogModelObject>(configuration: configuration),
    translator: DialogTranslator(configuration: configuration)
)

/// Obtain dialogs (from backend for example)
let dialogs = dialogService.obtainDialogs()

/// Save dialogs
try dao.persist(dialogs)

/// Obtain some info from database
let count = try dao.read().count

/// Erase all dialogs from database
try dao.erase()
```

Что может DAO?

```
// MARK: - Create

public func create(_ plain: Plain) throws {
    try storage.create { model in
        try translator.translate(from: plain, to: model)
    }
}
```

INCETRO

Что может DAO?

```swift
// MARK: - Read

public func read(predicatedBy predicate: NSPredicate, orderedBy name: String, ascending: Bool) throws -> [Plain] {
    let sorter = SortDescriptor(key: name, ascending: ascending)
    let models = try storage.read(predicatedBy: predicate, includeSubentities: true, sortDescriptors: [sorter])
    let plains = try translator.translate(models: models)
    return plains
}
```

INCETRO

Что может DAO?

```
// MARK: - Read

public func read(predicatedBy predicate: NSPredicate, orderedBy name: String, ascending: Bool) throws -> [Plain] {
    let sorter = SortDescriptor(key: name, ascending: ascending)
    let models = try storage.read(predicatedBy: predicate, includeSubentities: true, sortDescriptors: [sorter])
    let plains = try translator.translate(models: models)
    return plains
}
```

INCETRO

Что может DAO?

```
// MARK: - Update

public func persist(_ plains: [Plain]) throws {
    try plains.forEach(persist)
}
```

INCETRO

Что может DAO?

```swift
// MARK: - Delete

public func erase(byPrimaryKeys primaryKeys: [UniqueID]) throws {
    let predicate = NSPredicate(format: "uniqueId IN %@", primaryKeys.map(\.rawValue))
    try erase(predicatedBy: predicate)
}
```

INCETRO

А что кроме Realm?

Realm? А иначе можно?

```swift
// MARK: - UserModelObject

final class UserModelObject: NSManagedObject, Storable {

    public typealias PrimaryType = Int64
    @NSManaged public var id: Int64
    @NSManaged public var name: String
    @NSManaged public var age: Int16
}
```

INCETRO

CoreData тоже на месте

```swift
/// Create CoreData storage instance
let configuration = CoreStorageConfig(containerName: "Name of container also is filename for `*.xcdatamodeld` file.")

/// Create DAO instance
let dao = DAO(
    storage: CoreStorage(with: configuration, model: UserModelObject.self),
    translator: UserTranslator(configuration: configuration)
)

/// Obtain users (from backend for example)
let users = userService.obtainDialogs()

/// Save users
try dao.persist(users)

/// Obtain some info from database
let count = try dao.read().count

/// Erase all users from database
try dao.erase()
```

INCETRO

Пошагово внедряем DAO

INCETRO

Алгоритм

- Создать иерархию PlainObject-ов
- Написать соответствующую ей иерархию Model-ей
- Реализовать Translator-ы для каждой пары Plain-Model
- Пользоваться экземпляром DAO

Алгоритм

Plain

```swift
// MARK: - DialogPlainObject

public struct DialogPlainObject: Plain {

    // MARK: - Plain

    public var uniqueId: UniqueID {
        UniqueID(value: id)
    }

    // MARK: - Properties

    public let id: Int
    public let isPinned: Bool
    public let messages: [MessagePlainObject]
}
```

INCETRO

Алгоритм

Plain

```swift
// MARK: - DialogPlainObject

public struct DialogPlainObject: Plain {

    // MARK: - Plain

    public var uniqueId: UniqueID {
        UniqueID(value: id)
    }

    // MARK: - Properties

    public let id: Int
    public let isPinned: Bool
    public let messages: [MessagePlainObject]
}
```

INCETRO

Plain – как протокол

```swift
// MARK: - Plain

/// Parent for all plain objects
public protocol Plain {

    /// Unique identifier
    var uniqueId: UniqueID { get }
}

public extension Plain {

    /// Comparison function
    ///
    /// - Parameter other: entity compare with.
    /// - Returns: result of comparison.
    func equals<T>(_ other: T) -> Bool where T: Plain {
        return self.uniqueId == other.uniqueId
    }
}
```

INCETRO

Алгоритм

Model

```
// MARK: - DialogModelObject

final class DialogModelObject: RealmModel {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var isPinned = false
    let messages = List<MessageModelObject>()
}
```

INCETRO

Алгоритм

Model

```
// MARK: - DialogModelObject

final class DialogModelObject: RealmModel {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var isPinned = false
    let messages = List<MessageModelObject>()
}
```

INCETRO

RealmModel

```swift
final class DialogModelObject: RealmModel {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var isPinned = false
    let messages = List<MessageModelObject>()
}
```

⟷

```swift
// MARK: - DialogModelObject

final class DialogModelObject: Object {

    // MARK: - Properties

    @objc dynamic var id = 0
    @objc dynamic var isPinned = false
    let messages = List<MessageModelObject>()

    // MARK: - Object

    override class func primaryKey() -> String? {
        "id"
    }
}
```

INCETRO

Алгоритм

Translator

```swift
// MARK: - DialogTranslator

final class DialogTranslator {

    // MARK: - Aliases

    typealias PlainModel = DialogPlainObject
    typealias DatabaseModel = DialogModelObject

    // MARK: - Properties

    private lazy var dialogStorage = RealmStorage<DialogModelObject>(configuration: self.configuration)
    private let configuration: RealmConfiguration

    // MARK: - Initializers

    public init(configuration: RealmConfiguration) {
        self.configuration = configuration
    }
}

// MARK: - Translator

extension DialogTranslator: Translator {

    func translate(model: DatabaseModel) throws -> PlainModel {
        ...
    }

    func translate(plain: PlainModel) throws -> DatabaseModel {
        ...
    }

    func translate(from plain: PlainModel, to databaseModel: DatabaseModel) throws {
        ...
    }
}
```

Итог

DAO

```swift
/// Create DAO instance
let dao = DAO(
    storage: RealmStorage<DialogModelObject>(configuration: configuration),
    translator: DialogTranslator(configuration: configuration)
)

/// Obtain dialogs (from backend for example)
let dialogs = dialogService.obtainDialogs()

/// Save dialogs
try dao.persist(dialogs)

/// Obtain some info from database
let count = try dao.read().count

/// Erase all dialogs from database
try dao.erase()
```

INCETRO

Плюсы

1. Легкость при смене БД
2. Паттерн DAO делает упор на низкую связь между различными компонентами приложения.
3. Простота написания Unit тестов
4. Изолированная база данных

Ссылки на github.com

Monreau

/Incetro/Monreau

DAO

/Incetro/DAO

DAO от Роботов

/RedMadRobot/DAO

Минусы

1. Невозможность использовать особенности конкретной БД

2. Несколько соединений с БД

3. Дублирование кода

Спасибо за внимание!

INCETRO

INCETRO