第19章 基本動作の開発

まずは、Twitterクライアントして基本的な動作部分から制作していきましょう。なお、サンプルソースをダウンロードしてXcodeでプロジェクトを開いてから読み進めることをおすすめします。

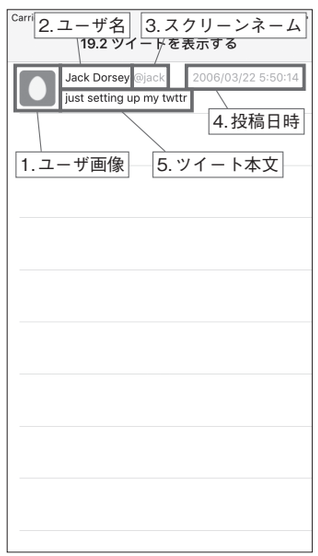
19.1 Twitterクライアントを作る

Twitterは140文字以下の「ツイート」の投稿を共有するWebサービスです。ツイートはある程度複雑なデータ構造を持っていますので、これらをRealmで保存して表示できるアプリを作っていきます。

Twitter APIについて

TwitterはツイートのJSON）を取得するためのAPIが用意されています。サンプルではAPIの通信部分を簡易的にするために、APIから返される想定のJSONをSwiftのオブジェクト（Array、Dictionary、Stringなど）に変換したJSONオブジェクトを静的に用意し、仮想のリクエストからJSONオブジェクトを取得する流れを作っています。最後の、20.4 その後の開発で実際にTwitter APIとの通信部分に対応しています。

○図19.1：ツイートを構成する情報



19.2 ツイートを表示する

まずはシンプルに1つのツイートを表示します。

ツイートを構成するのに最低限含まれている情報は、ユーザ名、スクリーンネーム（一意の文字列ID）、ユーザ画像URL、投稿日時、ツイート本文があります（**図19.1**）。

【サンプル】
サンプル/19-02_ツイートを表示する/TwitterTweet.xcodeproj

ユーザのモデルクラスを定義する

リスト19.1はTwitter APIから取得できるユーザのDictionary（JSONオブジェクト）の例です。

ユーザのDictionaryを元にモデル定義を行います（**リスト19.2**、参照 5.1：モデル定義とは）。ポイントはプロパティ名をDictionaryのキーと同一にすることです。Realmに保存するユーザが重複しないようにidをプライマリキーにします。

○リスト19.1：ユーザのJSONオブジェクト（TwitterJSON.swift）

```
// ユーザのJSONオブジェクト(必要な部分だけを抜粋)
static let user: [String: Any] = [
    "id": 12, // ユーザID
    "name": "Jack Dorsey", // ユーザ名
    "screen_name": "jack", // スクリーンネーム(変更可能な一意の文字列ID)
    "profile_image_url_https": "https://<長いので紙面上は省略>.png" // ユーザ画像URL
]
```

○リスト19.2：ユーザのモデル定義（User.swift）

```
class User: Object {
    dynamic var id = 0 // 一意の数値ID(変更不可でTwitterアカウントを作成時に自動で与えられる)
    dynamic var name = "" // 名前
    dynamic var screen_name = "" // スクリーンネーム(変更可能な一意の文字列ID)
    dynamic var profile_image_url_https = "" // プロフィール画像URL

    override class func primaryKey() -> String? {
        return "id"    // idをプライマリキーに指定する
    }
}
```

ツイートのモデルクラスを定義する

リスト19.3はTwitter APIから取得できるツイートのDictionary（JSONオブジェクト）の例です。ツイートのDictionaryには必ずユーザのDictionaryが含まれています。

ツイートのDictionaryを元にモデル定義を行います（**リスト19.4**）。ユーザと同様にプロパティ名をDictionaryのキーと同一にします。Realmに保存するツイートが重複しないようにidをプライマリキーとします。

ツイートを追加する

ツイートのDictionaryからツイートモデルクラスを生成し、初期化してRealmに追加します（参照 7.3：モデルオブジェクトの追加）。

リスト19.5では起動のたびにRealmに追加するようになっています。ツイートモデルクラスにはプライマリキーを定義しているため、add(_:update:)のupdateをtrueにして追加する必要があります。updateがtrueでない場合は、新規でモデルを追加する挙動となり一意であるはずのツイートモデルが重複することになってしまうので、例外が発生します。Realmでは開発中にモデル定義に反する挙動（今回の例だとプライマリキーがあるのに重複させるadd()を使用している）に気づくことができるように例外が発生するようになっています。

サンプルのJSONオブジェクトは静的なので変更が起きることはないですが、仮にプライマリキーのid以外の要素が更新された場合は適切に上書き更新されます。

Appendix A.1

クラス

ここでは、「Object」「Realm」「Results」「List」「LinkingObjects」「Migration」「Schema」「ObjectSchema」「Property」「RealmOptional」「AnyRealmCollection」「RLMIterator」「ThreadSafeReference」クラスを説明しています。

A.1.1 Object

【宣言】

```
open class Object: RLMObjectBase
```

Objectクラスは、Realmのモデルオブジェクトを定義するために使用されるクラスです（**リストA.1.1**）。Objectクラスを継承したサブクラスが、Realmのモデルとして扱われます。そのため、Objectクラスを直接インスタンス化して使うことはありません。なお、プロパティ定義でサポートされている型は**表A.1.1**のとおりです。

○リストA.1.1：モデル定義例

```
class Person: Object { // RealmSwift.Objectクラスを継承する必要がある
    dynamic var bool = false            // Bool
    dynamic var int = 0                 // Int
    dynamic var float: Float = 0        // Float
    dynamic var double: Double = 0      // Double

    dynamic var string = ""             // String
    dynamic var date = Date()           // Date
    dynamic var data = Data()           // Data

    let boolOptional = RealmOptional<Bool>()     // Boolのオプショナル型
    let intOptional = RealmOptional<Int>()       // Intのオプショナル型
    let floatOptional = RealmOptional<Float>()   // Floatのオプショナル型
    let doubleOptional = RealmOptional<Double>() // Doubleのオプショナル型

    dynamic var stringOptional: String?          // Stringのオプショナル型
    dynamic var dateOptional: Date?              // Dateのオプショナル型
    dynamic var dataOptional: Data?              // Dataのオプショナル型

    dynamic var dog: Dog? // 1対1の関連
    let cats = List<Cat>() // 1対多の関連
}

class Dog: Object {
    let persons = LinkingObjects(fromType: Person.self,
                                 property: "dog") // 逆方向の関連
}
class Cat: Object {
    let persons = LinkingObjects(fromType: Person.self,
                                 property: "cats") // 逆方向の関連
}
```


【プロパティ定義の制約】

- 整数型は、Int、Int8、Int16、Int32、Int64が定義可能
- 小数型を使用する場合は、FloatまたはDoubleを使用する。CGFloat型はプラットフォーム（CPUアーキテクチャ）によって実際の定義が変わるため使用しない
- RealmOptional、List、LinkingObjects はletで、それ以外はすべてdynamic varで定義する必要がある
- Bool、Int、Float、Doubleのオプショナル型を定義するには、ラッパークラスのRealmOptionalクラスを使用する必要がある
- lazy（遅延ストアドプロパティ）は使用できない

○表A.1.1：プロパティ定義でサポートされているすべての型

型	非オプショナル型の定義	オプショナル型の定義
Bool	dynamic var value = false	let value = RealmOptional<Bool>()
Int	dynamic var value = 0	let value = RealmOptional<Int>()
Float	dynamic var value: Float = 0	let value = RealmOptional<Float>()
Double	dynamic var value: Double = 0	let value = RealmOptional<Double>()
String, NSString	dynamic var value = ""	dynamic var value: String?
Date, NSDate	dynamic var value = Date()	dynamic var value: Date?
Data, NSData	dynamic var value = Data()	dynamic var value: Data?
Objectのサブクラス	非オプショナル型にはできません	dynamic var value: Class?
List<T>	let value = List<Class>()	オプショナル型にはできません
LinkingObjects<T>	let value = LinkingObjects(fromType: Class.self, property: "property")	オプショナル型にはできません

初期化子

■ init()

アンマネージドオブジェクトを生成します（参照 7.3：アンマネージドオブジェクト／マネージドオブジェクト）。

【宣言】

```
public override required init()
```

■ init(value:)

指定の値で初期化し、アンマネージドオブジェクトを生成します（参照 7.3：アンマネージドオブジェクト／マネージドオブジェクト）。

【宣言】

```
public init(value: Any)
```