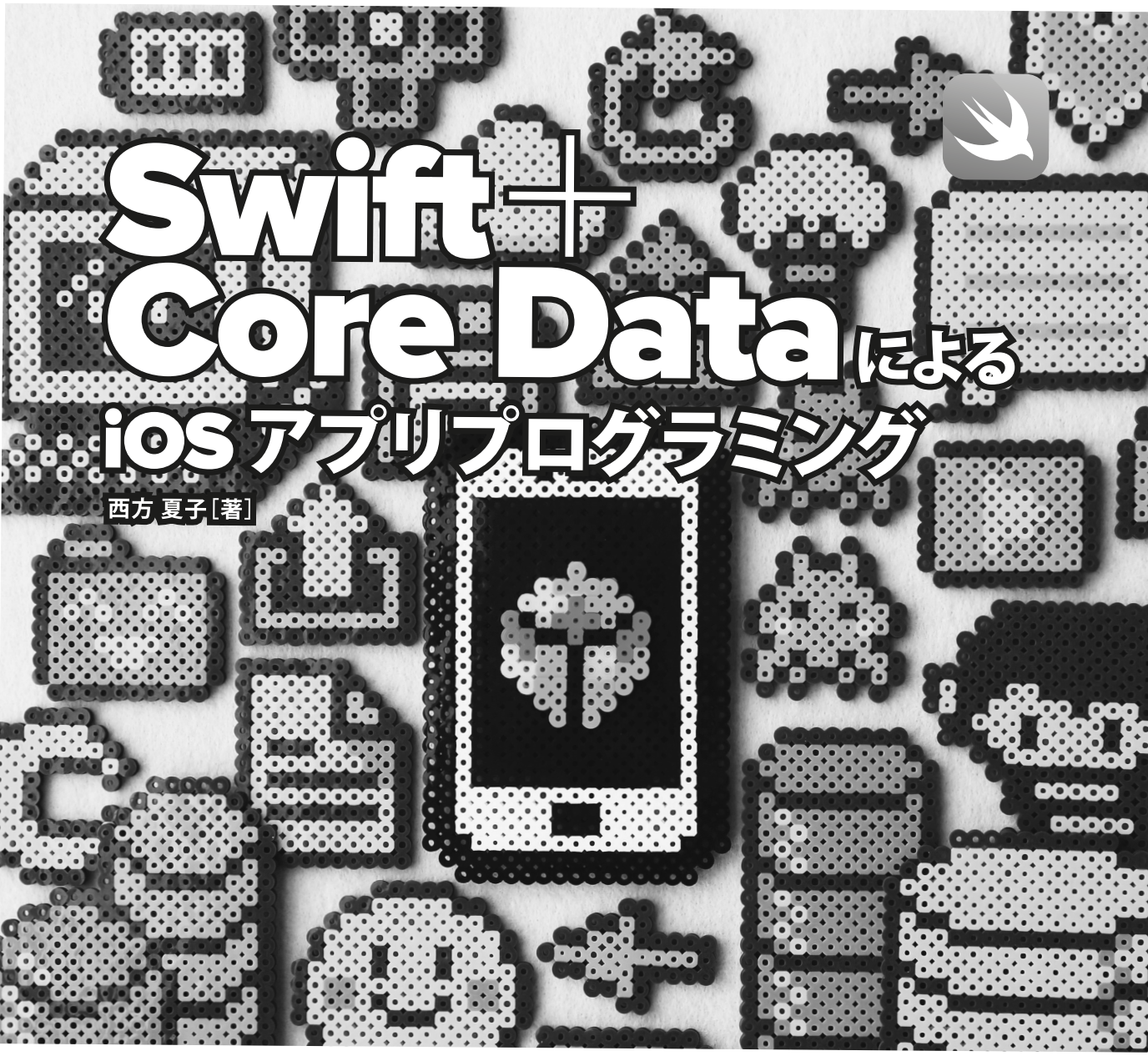
Swift＋Core Dataによるiosアプリプログラミング

西方 夏子［著］

Core Dataの
「仕組み」と「使い方」
をマスターして、
正しいiOSアプリ
設計を身につけよう！

Core Dataはオブジェクト管理の仕組みを提供するフレームワーク。本書ではCore Dataの仕組みから、ユーザインターフェイスとの連携方法を重点的に解説します。
チュートリアル形式で、アプリを作成しながら、Core Dataの基礎と使い方の両方が習得できます。

マイナビ



Swift＋Core Dataによるiosアプリプログラミング

西方 夏子［著］

マイナビ

■本書のサポートサイト

本書のサンプルファイル、補足情報、訂正情報などを掲載してあります。適宜ご参照ください。

http://book.mynavi.jp/supportsite/detail/9784839954888.html

●本書は 2016 年 1 月段階での情報に基づいて執筆されています。
本書に登場する製品やソフトウェア、サービスのバージョン、画面、機能、URL、製品のスペックなどの情報は、すべてその原稿執筆時点でのものです。
執筆以降に変更されている可能性がありますので、ご了承ください。

●本書に記載された内容は、情報の提供のみを目的としております。
したがって、本書を用いての運用はすべてお客様自身の責任と判断において行ってください。

●本書の制作にあたっては正確な記述につとめましたが、著者や出版社のいずれも、本書の内容に関してなんらかの保証をするものではなく、内容に関するいかなる運用結果についてもいっさいの責任を負いません。あらかじめご了承ください。

●本書中の会社名や商品名は、該当する各社の商標または登録商標です。
本書中では TM および R マークは省略させていただいております。

はじめに

長年にわたり進化を続けている iOS SDK ですが、歴史が長いフレームワークの 1 つに Core Data が挙げられます。Core Data は強力なフレームワークである一方、複雑なフレームワークでもあります。それが故に、敬遠している開発者も多いのではないでしょうか。

近年、Xcode のプロジェクトテンプレートでは、Core Data の使用がデフォルトで有効となっています。Core Data が改良を重ね、多くの開発者にとって使いやすいフレームワークになったことを意味しています。

Core Data はオブジェクト管理の仕組みを提供するフレームワークですが、iOS アプリは、オブジェクト（データ）だけでは成り立ちません。必ず用意しなくてはならないのがユーザーインターフェイス（画面）です。Core Data は画面と連携してこそ意味を持つのです。

本書では、Core Data そのものの仕組みに留まらず、Core Data とユーザーインターフェイスとの連携を重点的に解説します。道具としての Core Data の性能が分かったとしても、使い方が分からなければ意味がないからです。使用方法の習得には実際に手を動かすことが一番です。そこで、チュートリアル形式で解説を進めます。アプリを完成に近付けながら、Core Data の基礎と実践が習得できます。

特に躓きがちな箇所では、問題点を明確にするためにエラーの内容を確認しながら、コードを修正するなど、より実践に即した解説に努めています。

ユーザーインターフェイスの実装に不慣れでもスムーズに読み進めるように、チュートリアルで扱うユーザーインターフェイス実装の基本事項を別章にまとめています。また、Objective-C と Swift ではプログラミング言語の性質に違いがあるため、実装時の注意点も異なります。本書では Swift をベースとして解説しています（執筆時点での最新バージョンは Swift 2.1 ／ Xcode 7.2 ／ iOS 9.2）。

本書を通じて、多くの読者がアプリ開発の幅を広げ、開発者としてのスキルアップに繋がる手助けができれば、これにまさる喜びはありません。

2016 年春
西方 夏子

管理オブジェクトモデル

Core Data スタック

管理オブジェクト

属性

関係、フェッチ済みプロパティ

管理オブジェクトコンテキスト

検証

フェッチ

Chapter 01

概論

本書で解説するCore Dataは、データ管理を主目的とする秀逸なフレームワークです。ユーザーインターフェイスとの連携を前提にするなど、必ず表示画面が存在するiOSのアプリ開発では重要な役割を担います。

本章では、実際のアプリ開発におけるCore Dataの位置付けと共に、その主な機能の全体像を概説します。全貌の紹介と共に本書で習得するCore Dataがいかに優れたフレームワークで、実用的であるかを説明します。

Section
1-1 iOSアプリとCore Data

iOSアプリの構成は、画面表示を司るビュー、データ管理を担当するモデル、そしてビューとモデルを制御するコントローラに分けて考えることができます。本書で解説する「Core Data」は、この中のモデル層を管理するフレームワークです。充実した機能が搭載されていることはもちろんのこと、ビューやコントローラとの親和性の高さも特徴です。

Core Dataは秀逸なフレームワークですが、単独では意味を成しません。ビューやコントローラと共に協調して初めて、その力を発揮します。本節では、iOSアプリの構成において、Core Dataがどのような位置付けにあるかを説明します。

Core Dataの位置付け

iOSアプリに必ず1つは存在するのが画面です（図1.1）。アプリを起動すると、初期画面が表示されます。表示後は、他の画面へ遷移することもあれば、この初期画面のみで完結することもあります。いずれにせよ、iOSアプリにおいて画面の表示は必須の要素です。

iOSアプリは表示画面を通じてユーザーとやりとりします。例えば、必要な情報を表示することも、タップ操作で入力を受け取ることも、アプリとユーザーとのやり取りといえます。

アプリとユーザーとのやり取りで、表示内容やタップ操作に伴って処理する内容を制御するのはコントローラの役目です。ビューコントローラと呼ばれる専用コントローラが、これらの処理を一手に引き受けます。また、画面に表示するコンテンツやユーザーから受け取る入力内容（タップ操作やテキスト入力など）は、それぞれ「データ」として管理されます。これらのデータは次回のアプリ実行時にも使用できるように、必要に応じてデバイス内に保存します。

iOSアプリでの保存処理は「永続化」と呼ばれ、Core Dataにも永続化の機能が備わっています。また、個々に管理される「データ」構造を記したものを「モデル」と呼びます。アプリ内で扱う「データ」は、メモリ上に存在する「オブジェクト」として表現されます（図1.1）。

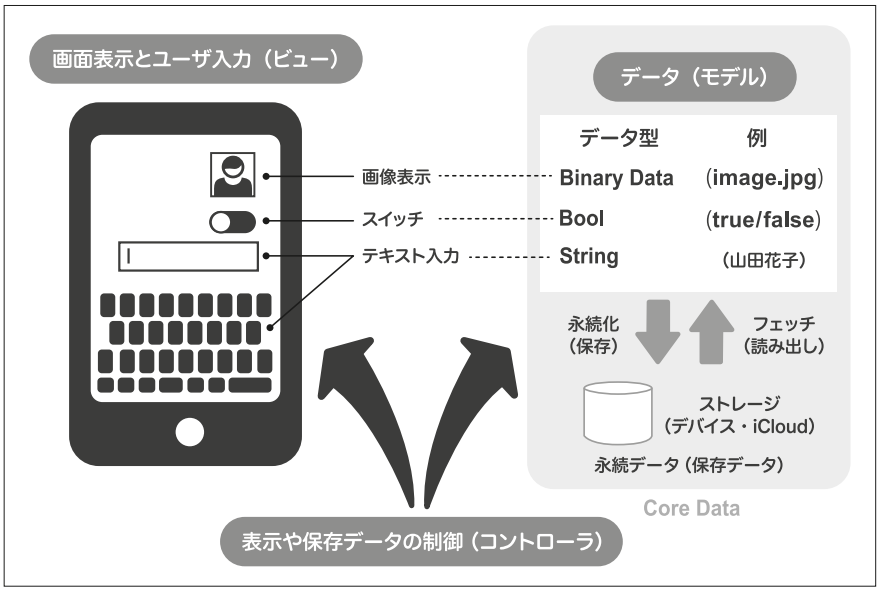


図 1.1：iOS アプリの構成と Core Data の位置付け

Core Data の主な役割は、「モデル」の構築と「オブジェクト」の管理、そして「永続化」です。すなわち Core Data はデータ管理のためのフレームワークです。

なお、Core Data では永続化するファイル形式として SQLite が利用できます。それが故に、「Core Data ＝データベース」と短絡してしまいがちです。しかし、SQLite（データベース）は、Core Data の機能のごく一部であることを覚えておいてください。

Core Data の歴史を少し振り返ってみましょう。Core Data が iOS に登場したのは、iOS 3（iPhone OS 3）からです。しかし、初期バージョンではバグが多く、特に UI との連携はお世辞にも使いやすいものとはいえませんでした。また、Core Data の使用がデフォルトとなっている Xcode のテンプレートもなく、上級者向けのフレームワークという印象がありました。

しかし、iOS がバージョンアップを重ね iOS 9 となった現在、多くのバグは修正され、誰にでも使えるフレームワークとなっています（執筆時現在）。Xcode 7 では 2 つのテンプレートで Core Data の使用がデフォルトになっています。[Master-Detail Application] と [Single View Application] です。

前者の [Master-Detail Application] はテーブルビューを使用したアプリです。また、後者の [Single View Application] を利用すれば、例えば、Keynote などのプレゼンテーションアプリの作成にも、Core Data を有効に活用できます。

Section
1-2 Core Data の魅力

データ管理のための数多くの機能を搭載する Core Data は、その多機能さからも分かる通り、複雑なフレームワークです。この複雑なフレームワークをわざわざ使うには、相応の理由があります。Core Data を利用するメリットを一言で表すと開発効率の向上です。確かに複雑で扱いにくい点があることは否定できませんが、それでも、Core Data を使うことで開発効率は格段に向上します。本節でその具体例を挙げましょう。

UI の更新と連動したオブジェクトの管理

前節でも述べた通り、iOS アプリに画面は不可欠です。データを管理する際にも画面との連携が重要です。表示内容と実際に保存されている内容に相違があってはならないためです。

Core Data を利用すると、簡単にデータ管理と画面管理を連携することが可能です。特に「Fetched Results Controller」と呼ばれる専用コントローラを使用すれば、データ管理とテーブル表示の連携が非常にシンプルになります。

変更の管理

データ変更の管理も忘れてはいけません。データの編集は常に確定されるわけではなく、時には編集内容がキャンセルされる場合もあります。また、「取り消し」(Undo) や「やり直し」(Redo) の実装が必要なケースもあります。こうした変更の管理機能を自前で実装するには、多くのコードを記述する必要に迫られます。しかし、Core Data を使えば、元々管理機能が組み込まれているため、新たに用意する必要はありません。

検証とエラーハンドリング

データの編集時にどのような値でも設定できるわけではありません。例えば、電話番号をテキストとして入力するケースでは、数字とハイフン（もしくは数字だけ）に限定する必要がありますし、何らかの個数を数値として入力するときに負の値は不適合です。このように、データの値が適合しているかどうかを調べることを検証と呼びます。また、検証の結果が不適合（検証エラー）となった場合は、ユーザーにエラー内容を表示するなどの処理が必要です。

Core Data にはこうした検証やエラーハンドリングの仕組みも用意されています。アプリ側では、この仕組みを使ってアプリ固有の機能を実装していきます。

モデルエディタの使用

Xcode には、Core Data が使用するデータモデル（データ構造を記したもの）を編集する、「モデルエディタ」が用意されています。モデルエディタを使用することで、データモデルを直感的に編集できます。下図にモデルエディタの表示例を示します（図 1.2）。

データ構造を視覚的に確認しながら実装できるため、ソースコードでの記述に比べて、格段に実装速度が向上します。

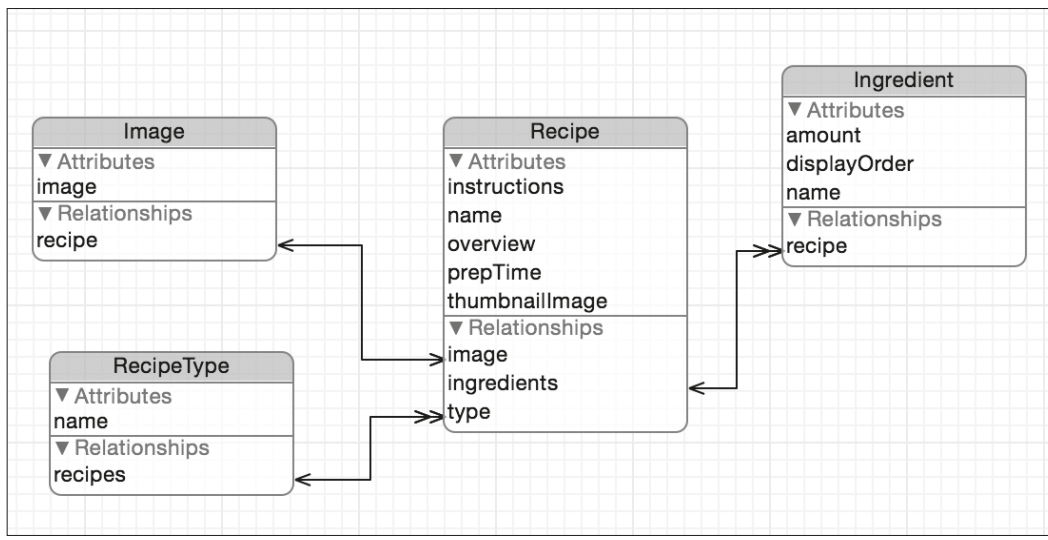


図 1.2：モデルエディタの表示例（アップル公式サンプル iPhoneCoreDataRecipes より）

パフォーマンスへの配慮

取り扱うデータ量が大規模になると、パフォーマンスのケアも必要となります。メモリ使用効率にせよ CPU 負荷にせよ、軽快な動作にはこれらの効率化は避けて通れません。Core Data では元々、膨大なデータ量を扱うことが想定されているためパフォーマンスへの配慮も十分です。

例えば、オブジェクトグラフ全体をメモリ上に展開する代わりに、参照する内容だけをメモリ上に配置するなど、効率化を図る機能が標準で備わっています。

並列処理

パフォーマンスへの配慮で欠かせないのが並列処理です。並列処理が必要な理由の 1 つに、UI の更新が挙げられます。iOS アプリでは原則として、UI の更新はメインスレッドからしか行えません。そのため、UI 更新に影響するオブジェクトもメインスレッドで扱う必要があります。

ところが、扱うデータの規模が大きくなってくると、すべてのデータをメインスレッドで処理することが困難になります。メインスレッドで CPU 負荷が掛かる処理を行うと、画面の更新やユーザーイベントの受け取りが一時的に止まってしまうためです。

そこで必要に応じてデータ処理の一部をバックグラウンドで行います。Core Data には並列処理の仕組みも備わっているため心配はいりません。ただし、Core Data の各モジュールはスレッドセーフではないため、並列処理の正しいステップを踏む必要があります。

本項で紹介したのは、Core Data の機能の一部です。実際には Core Data の機能はさらに多岐にわたります。アプリのモデル層を実装するほとんどの場合で、本項で紹介した機能が必要になります。もちろん、必須ではない機能もありますが、アプリのクオリティが向上することは間違いありません。また、Core Data には、モデル層に欠かせないこれらの機能があらかじめ用意されているため、開発者はアプリ固有の実装に注力できます。Core Data を使うことで開発効率が向上するのはこのためです。

ちなみに、Apple の公式ドキュメント「Core Data Programming Guide」には、Core Data を利用することで、モデル層の実装に必要なコードを 50%〜 70%削減できると記述されています。Core Data にはモデル層に必要な機能が詰め込まれているのです。

Section
1-3 機能と役割・関連モジュール

Core Data はデータ管理のためのフレームワークであり、その主な役割は、モデルの構築とオブジェクトの管理、そしてデータの永続化に大別されます。本節では、各役割を具体的に説明すると共に、関連モジュールも紹介します。各モジュールの詳細は「Chapter 03 全体構造と関連モジュール」（P.065）で解説します。

1-3-1 データモデルの構築

データモデルとは、アプリで扱うデータの構造を定義したものです。Objective-C や Swift の一般的なクラスでデータモデルを構築することもできますが、Core Data には、データモデル専用の NSManagedObjectModel クラスが用意されています。「管理オブジェクトモデル（Managed Object Model）」と呼ばれるものです。

管理オブジェクトモデルは、Xcode 付属のモデルエディタを使用して編集が可能です。モデルエディタでは、管理オブジェクトモデルに必要な各要素を UML（Unified Model Language）図のように表示したり（P.005 図 1.2）、テーブル表示することが可能です。モデル全体を視覚的に確認できるため、実装時の漏れや間違いを早期に発見できます。また、大規模開発では情報共有のツールとしても役立ちます。

モデルエディタでは、モデルの詳細、例えば、値のデフォルト値や有効範囲なども設定できるため、コードによる追加実装なしでも十分実用的です。もし、モデルエディタなしで管理オブジェクトモデルを実装すると、大量のコードを記述する必要に迫られます。ちなみに、その一例は「11-4-1 管理オブジェクトモデルの作成」（P.350）で、Playground で Core Data を扱う方法として後述しています。

データモデルは個々のデータそのものではなく、データの構造です。アプリで扱うデータには何通りの型があり、それぞれどのような値が設定できるか、さらにはデータ間の関係を示すものがデータモデルです。個々の実データはデータモデルを元に生成され、メモリ上ではこれが「オブジェクト」として扱われます。オブジェクトに関しては次項で説明します。

1-3-2 オブジェクトの管理

メモリ上に展開されたデータはオブジェクトとして扱われます。Core Data で扱うオブジェクトは「管理オブジェクト（Managed Object）」と呼ばれる特殊なオブジェクトで、NSManagedObject クラスで表現されます。

管理オブジェクトは、管理オブジェクトモデルを元に生成されるため、設定できる値や他のオブジェクトとの関係が既に定義されています。この定義を元に、値の検証（有効範囲内にあるかなど）やエラーハンドリングが可能となります。

管理オブジェクトを扱う上で重要なのが、「管理オブジェクトコンテキスト（Managed Object Context）」です。NSManagedObjectContext クラスで表現されます。管理オブジェクトコンテキストは、いわば管理オブジェクトのキャンバスのようなものです。管理オブジェクトのライフサイクルは、管理オブジェクトコンテキストに追加されたときに始まり、管理オブジェクトコンテキストから削除されたときに終わります。

また、管理オブジェクトコンテキストは複数を生成可能です。並列処理の実現にも、複数の管理オブジェクトコンテキストが必要となります。このとき、管理オブジェクトコンテキスト間のやり取りや、UI との連携が重要になることはいうまでもありません。

管理オブジェクトコンテキストを複数用意するのは、並列処理のためだけではありません。管理オブジェクトのライフサイクルは、登録されている管理オブジェクトコンテキストに支配されます。複数の管理オブジェクトコンテキストがある場合、あるコンテキスト上に施された変更が、他のコンテキストに影響を及ぼすことはありません。この特徴を利用して、異なるオブジェクトの変更や保存を独立して制御できます。変更内容の保存やキャンセルのタイミングはユーザー体験に大きく影響しますが、Core Data を使えば柔軟な設計が可能です。

管理オブジェクトの更新管理も管理オブジェクトコンテキストの役目です。変更の取り消し（Undo）とやり直し（Redo）も、管理オブジェクトコンテキストが持つ Undo Manager で実現されます。管理オブジェクトコンテキストは、登録されている管理オブジェクトの状態を常に把握しています。追加・削除を含む変更内容は、管理オブジェクトコンテキストが保存されるまで追跡され続けます。

管理オブジェクトコンテキストを保存することは、そこに登録されている管理オブジェクトの永続化を意味します。永続化に関しては次項で解説します。

1-3-3　永続化

管理オブジェクトコンテキストを保存すると、登録されている管理オブジェクトの状態が、ファイル（一部例外あり）に保存されます。これを永続化と呼びます。

Core Data では永続ストアの形式を選択できます。そして、オブジェクト管理に影響することなく、ストア形式（ファイルタイプ）やストレージ（デバイス内または iCloud）を変更できることもまた、Core Data の特徴の 1 つです。

表 1.1 に Core Data が取り扱うストア形式を示します。

表 1.1：Core Data のストア形式

ストア	種類	速度	メモリへの展開
SQLite	永続	高速	必要部分のみ
バイナリ	永続	高速	全て
XML（OS X のみ）	永続	低速	全て
メモリ内	メモリ	高速	全て

メモリ内ストアを除き、すべてのストアは永続化されます。メモリ内ストアを使用すると、実際にはデータは永続化されませんが、Core Data では便宜上、上記 4 種類のストアのことを「永続ストア」と呼びます。

ストア形式として最も一般的なのは SQLite ストアです。SQLite ストアの最大の特徴は、オブジェクトグラフで必要な部分だけ（アクセスのあった部分だけ）をメモリ上に展開することです。そのための処理はすべて Core Data で内部的に行われます。この機能のおかげで、場合によっては使用メモリの大幅削減が可能です。SQLite ストア以外のストア形式では、オブジェクトグラフ全体をメモリ上に展開します。

SQLite ストアではオブジェクトグラフの一部のみをメモリ上に展開するのに対し、バイナリストアではオブジェクトグラフ全体をメモリ上に展開します。また、保存時も全データが一度に書き込まれます。データ量が多くなることが予想されるのであれば、バイナリストアではなく SQLite ストアの使用を検討すべきです。

また、他のストア形式とは違い、メモリ内ストアには永続性がありません。一度アプリを終了するとデータはすべて消えてしまいます。そのため、実際のアプリでの実用性は低いですが、単体テストで利用されることがあります。

実はiOSには、Core Data以外にも永続化の手法が用意されています。設定を保存するための「User Defaults」と、任意のオブジェクトをプロパティリストに変換して保存するための「Keyed Archiver」です。それぞれの永続化方法は用途が限定されており、目的に応じて適切な方法を選ぶ必要があります。

User Defaultsは、キーと値の組み合わせで設定データを保存します。Setting Bundleと連携しており、Setting Bundleを通じて設定した値をアプリ側から読み出すことができることも特徴の1つです。アプリの設定内容を永続化したい場合は、User Defaultsの使用が最適です。

前述のUser Defaultsが設定の保存に適しているのに対して、Keyed Archiverはユーザーデータの保存に適しています。Keyed Archiverを使えば、任意のオブジェクトの保存が可能です。保存の際にオブジェクトはプロパティリストに変換されます。

オブジェクト間に相関関係がなく、単独データとして利用するケースなどでは、Keyed Archiverも十分実用的です。しかし、保存すべきデータ数が多くなるとデータの管理が大変になります。また、データ間に相関がある場合、問題はより複雑になります。1つのデータを読み込むために、芋づる式に複数のオブジェクトを読み出す必要に迫られるためです。

さらに、検索やソートはどうでしょうか。この場合は全データを読み出す必要があります。データ数が多くなればなるほど、メモリ不足への懸念も大きくなります。

上記の問題点が少しでも危惧されるのであれば、永続化の手法として最適なのはCore Dataです。User DefaultsやKeyed Archiverは、一見簡単に実装可能と思いがちですが、複雑な機能の実現しようと試みた途端に破綻してしまいます。一方のCore Dataは、導入の手間は他の手法に比べてかかるものの、一度取り入れてしまえば、データ管理に必要な機能のほとんどが揃っているところが魅力です。目的に応じて、適切な永続化方法を選択してください。

Chapter 02

UI 開発の基礎

iOS アプリでは画面表示が不可欠です。本書では主にチュートリアル形式で Core Data の解説を進めますが、もちろんその中でも画面表示は必要です。

本章ではチュートリアルに先駆け、iOS アプリでの UI 実装の基本事項を解説します。UIKit の概要にはじまり、UI の実装に欠かせない Storyboard や Auto Layout の仕組み、テーブルビューの利用方法を説明します。特にテーブルビューはチュートリアルでも使用するため理解を深めてください。

Section

2-1 UIKitの機能と役割

本書で解説するCore Dataがモデルを司るフレームワークなら、UIKitはUIを司るフレームワークです。iOSアプリを構成するにはユーザーインターフェイスが必須です。画面表示はもちろんのこと、ユーザーからのタッチイベントを受け取って反応することも重要です。これらの処理を一手に引き受けているのがUIKitです。

本節では、UIKitの主な機能を紹介し、特に重要なビューとビューコントローラ、そしてイベントハンドリングの概要を解説します。

2-1-1 UIKitとは

アプリの起動処理は、アプリケーションオブジェクト（UIApplication）とアプリケーションデリゲート（UIApplicationDelegate）で行われます。起動状態はアプリケーションデリゲートを通じて通知されます。

この起動処理で、UIKitは画面を初期化してイベントハンドリングを開始します。

UIKitを取り扱う上でもっとも注意しなくてはならないのは、UIKitがスレッドセーフではないことです。UIKitのほとんどのクラスはメインスレッドからのアクセスしか許可されてなく、他のスレッドからアクセスすると例外が発生するなど予期せぬ挙動を起こします。

一方、本書のメインテーマであるCore Dataは、常にメインスレッドで動作するとは限りません。Core DataとUIKitを連携させる際、UIKitへのアクセスは必ずメインスレッドで行われるように注意してください。

UIKitの主なクラスを紹介します。

画面表示

画面表示に欠かせないのはビュー（UIView）とビューコントローラ（UIViewController）です。UIView は画面表示のためのオブジェクトで、数多くのサブクラスが用意されています。

UIView のサブクラスには、画面上に配置するボタンなどの部品も含まれます。これらの部品には、UIButton（ボタン）や UITextField（テキスト入力欄）、UILabel（テキスト表示）などの名前が付けられています。

本書ではいくつかの UI 部品を取り扱いますが、すべて Storyboard を用いて実装します。ドラッグ&ドロップで簡単に実装できるため、部品ごとの細かい知識は不要です。より詳細な知識が必要な場合は、Apple の公式ドキュメント「UIKit User Interface Catalog」などを参照してください。

UIViewController は、画面上にビューを配置したり、ユーザーからのイベントを受けるなど、その名の通り、コントローラの役割を担うオブジェクトです。

iOS には標準でいくつかのビューコントローラが用意されています。テーブルビューを表示する UITableViewController や、コレクションビューを表示する UICollectionViewController などです。。

なお、ビューとビューコントローラに関しては、それぞれ「2-1-2 ビュー」（P.014）「2-1-3 ビューコントローラ」（P.016）で掘り下げて説明します。

イベントハンドリング

イベントハンドリングでは、UIResponder と呼ばれるクラスが活躍します。UIResponder はユーザーからのイベントを受け取り処理します。実は UIView や UIViewController は UIResponder のサブクラスで、ユーザーイベントを受け取る仕組みが用意されています。

本書では、UIView や UIViewController の一部でイベントハンドリングを行いますが、タッチイベントやジェスチャを単独で扱うことはありません。UIView と UIViewController を理解していれば、チュートリアルを読み進めるには十分です。

なお、ビューやビューコントローラを利用したイベントハンドリングは、「2-1-4 イベントハンドリング」（P.019）で説明します。

Storyboard

Storyboardとは、画面そのものや画面と画面の接続情報などを管理するXcodeのツールです。本書ではStoryboardの使用を前提に解説を進めます。

また、Storyboardはツールと思われがちですが、実際にはUIKitに含まれるオブジェクトです。Storyboardエディタで編集した内容は、UIStoryboardクラスのオブジェクトとして表現されます。

なお、Storyboardに関しては、「2-2 StoryboardとAuto Layout」(P.021)で詳述します。

状態保存と復元

UIKitには、ビューの状態を保存・復元する機能が備わっています。アプリがバックグラウンド待機中にシステムにより強制終了された場合でも、最後に終了したときの状態を復元する機能で、State Preservationと呼ばれます。

非常に便利でユーザー体験の向上にも寄与している機能ですが、使用方法を間違えるとアプリが起動しなくなるなどの問題を引き起こします。特にCore Dataと連携させるときには、ビューとモデルとの不整合が生じやすいため、注意深く利用する必要があります。チュートリアルの実装では、State Preservationは利用しません。

その他

UIKitには本項で紹介した以外にも数多くの機能やモジュールが含まれています。本書ではすべてを紹介できないため、詳しくはAppleの公式ドキュメントなどを参照してください。

2-1-2 ビュー

ビューはUIViewまたはそのサブクラスで表現されます。画面の矩形領域にコンテンツを描画したり、この領域上のタッチイベントを処理するのがビューの主な役割です。

また、画面に表示するボタンなど、UI部品もすべてUIViewのサブクラスです。UI部品はStoryboardエディタで簡単に配置でき、凝った実装ではない限り取り扱いも容易です。

UI部品は大別すると次の4種類に分類できます。

・コンテンツの表示
・データコレクションの表示
・ユーザーイベントの受け付け（UIコントロール）
・バーの表示

テキストを表示するUILabelは、コンテンツ表示のためのビューです。これに対して、ユーザーからの入力を受け取るテキスト入力欄（UITextField）やボタン（UIButton）はUIコントロールに分類されます。

データコレクションを表示するビューには、テーブルビュー（UITableView）とコレクションビュー（UICollectionView）があります。前者はデータを縦方向にのみ並べることが可能です。後者は表示するデータのレイアウトは任意です。

チュートリアルではテーブルビューを用いて画面を構成します。テーブルビューの実装方法は、「2-3 テーブルビュー」（P.042）で解説します。

画面の上下にはバーの表示が可能ですが、これらのバーもビューの一種です。バーには、タブバー、ナビゲーションバー、ツールバー、検索バーがあります。

バー上にはUIBarButtonItemと呼ばれるボタンを配置可能です。外観は通常のボタンとよく似ていますが、実はUIBarButtonItemはビューではありません。NSObjectのサブクラスであるため一般的なビューとは異なりますが、イベントハンドリングの方法は変わりません。

なお、画面最上部に通信状況や時刻などを表示する細長いバーは、ステータスバーと呼ばれますが、他のバーとは取り扱いが異なります。表示するコンテンツはシステムで管理され、アプリ側で制御するものではないためです。アプリ側ではステータスバーの色や表示・非表示の切り替えが可能です。本書で言及する「バー」は、ステータスバーを含まないアプリ側で制御可能なバーのみを指します。

ちなみにビューの配置には、座標で指定する方法と、Auto Layoutと呼ばれる手法で自動的にレイアウトする方法があります。本書ではAuto Layoutを用いてビューのレイアウトを制御します。Auto Layoutは「2-2 StoryboardとAuto Layout」（P.021）で解説します。

2-1-3 ビューコントローラ

前項で説明したビューは、画面上に表示される部品そのものですが、ビューの表示はビューコントローラにより制御されます。ビューコントローラは、UIKit の中で最も規模の大きいモジュールです。役割も広範囲に渡ります。

ビューコントローラの主な機能を以下に挙げます。

・ビューの表示とイベント処理
・画面遷移に関する処理
・デバイス回転時の処理
・アプリの状態保存に関する処理
・メモリ管理

本項では、特に重要なビュー（画面）の表示とイベント処理に関して、ビューコントローラのライフサイクルと関連付けて解説します。なお、以降の説明は、Storyboard の使用を前提としていますが、Storyboard を使わずに構成したアプリでは、起動処理が少々異なります。

ビューとビューコントローラの初期化

アプリの起動時には Storyboard ファイルがロードされ、最初の画面のビューコントローラがインスタンス化されます。さらに、ビューコントローラが管理するビューがロードされ、初期画面を構成します。この時点で既にビューコントローラもビューも初期化されていますが、開発者が初期化処理に手を加えることも可能です。

ビューコントローラには生成に関するメソッドがいくつか用意されているので、これらのメソッドをオーバーライドすることで、独自の実装を追加できます。下記に特に重要なメソッドを示します（コード 2.1）。

コード 2.1：ビューコントローラの生成に関するメソッド（UIViewController）

```
// Storyboard ファイル（または Nib ファイル）からのロードが完了
func awakeFromNib()

// コンテンツビューを生成（または Storyboard からビューをロード）
func loadView()
```

```
// ビューのロードが完了
func viewDidLoad()
```

Storyboard を利用したビューコントローラでは、表示すべきビューは Storyboard から読み込まれます。この処理を行うのが loadView() メソッドです。したがって、サブクラスでは原則として loadView() をオーバーライドしません。

また、loadView() メソッドはビューコントローラにより内部的に呼ばれるメソッドで、ビューをロードする必要がある際に呼び出されます。ビューのロードが必要になるのは、UIViewController の view プロパティにアクセスがあったときです。つまり、view プロパティに初めてアクセスがあったときに、ビューコントローラがビューを読み込む設計です。

画面表示

ビューのロードが完了すると、viewDidLoad() が呼ばれます。画面表示に必要なオブジェクトはこの段階で初期化します。

ビューがロードされたら次は画面への表示です。画面の表示・消去に関してもいくつかのメソッドが用意されており、サブクラスではこれらをオーバーライドして処理を追加できます。下記コード例に、ビューコントローラのビューの表示・消去に関するメソッドを示します（コード 2.2）。

コード 2.2：ビューの表示と消去に関するメソッド（UIViewController）

```
// 表示処理に入る前に呼ばれる
func viewWillAppear(_ animated: Bool)

// 表示が完了したら呼ばれる
func viewDidAppear(_ animated: Bool)

// 消去処理に入る前に呼ばれる
func viewWillDisappear(_ animated: Bool)

// 消去が完了したら呼ばれる
func viewDidDisappear(_ animated: Bool)
```

ビューコントローラのビューが画面上に表示されることは、そのビューがビュー階層に追加されることを意味します。画面に表示するすべてのビューは、UIWindow と呼ばれる土台となるビューの上に階層構造を構築します。

viewWillAppear(_:) はビューがビュー階層に追加される直前に呼ばれ、ビュー階層への追加が完了すると、viewDidAppear(_:) が呼ばれます。消去の場合も同様です。

画面表示と消去のタイミングで何かしらの処理を追加したい場合には、前記コード例（コード2.2）に示したメソッドをオーバーライドします。

ここで注意したいのは、上記のメソッドが呼ばれるタイミングは、あくまでもビュー階層に変化があったときです。画面遷移したように見えても、ビュー階層からビューが消去されていなければ、viewWillDisappear(_:) や viewDidDisappear(_:) は呼ばれません。例えば、表示画面を覆うように次の画面が現れるモーダルビュー表示では、元の画面が完全に隠れない限り、ビュー階層から消去されません。

コンテナビューコントローラ

モーダルビュー表示と並んでよく利用される画面遷移に、コンテナビューコントローラを利用するものがあります。あるビューコントローラが複数のビューコントローラを管理し、「子」となるビューコントローラを次々に切り替えながら表示する画面遷移です。大元となるビューコントローラをコンテナビューコントローラと呼びます。

iOS には、標準のコンテナビューコントローラがいくつか用意されています。代表的なのは、ナビゲーションコントローラ（UINavigationController）とタブバーコントローラ（UITabBarController）です。また、画面サイズが多様化し、画面を分割して表示するためのスプリットビューコントローラ（UISplitViewController）も主流となってきました。いずれも UIViewController のサブクラスです。

ナビゲーションコントローラでは、ドリルダウン形式の画面表示を実現します。標準アプリでは、「設定」アプリがドリルダウン形式を採用しています。

ナビゲーションコントローラで管理される画面では、上部にナビゲーションバーが表示され、［戻る］ボタンの他にいくつかのバーボタンアイテムを表示可能です。

ちなみに、チュートリアルでは、モーダルビュー表示とナビゲーションコントローラの両方を使用します。いずれも画面遷移は Storyboard で実装します。

2-1-4　イベントハンドリング

UI コントロール(UIView)または UIBarButtonItem のイベントハンドリングには、ターゲット・アクションパターンが利用されています。

イベントを受け取った UI コントロールは、「ターゲット」として設定されているオブジェクトに実装された「アクション」を呼び出します。そのためにターゲットとなるクラスでは、アクションメソッドを実装しておく必要があります。

ターゲットとアクションの設定には Storyboard エディタを使うと簡単ですが、コードでも記述できます。コードで実装する場合、次のメソッドを使用します。

コード 2.3：ターゲットとアクションの設定（UIControl）

```
func addTarget(_ target: AnyObject?, action action: Selector,
               forControlEvents controlEvents: UIControlEvents)
```

上記コード例の controlEvents にはアクションを起こすイベントを指定します。例えば、ビューの矩形領域から指が離れたことを意味する TouchUpInside や、コントロールの状態が変化したことを意味する ValueChanged などがあります。UIControlEvent の他の値は、リファレンスガイドなどを参照してください。

action にセットするアクションメソッドは、オプションとして送信元オブジェクトである sender と検出されたイベントを引数に持てます。アクションメソッドの一般的な形式は、下記コードに示す通り、sender のみを引数に持つ形式です（コード 2.4）。

コード 2.4：アクションメソッドの一般的な形式

```
func action(sender: AnyObject)
```

Storyboard エディタを使用してターゲットとアクションの設定を行う場合でも、コントロールイベントを指定する必要があります。

UIBarButtonItem によるイベントハンドリングも、UIControl のイベントハンドリングと大きな違いはありません。ここでもターゲット・アクションパターンを用いています。

レスポンダチェーン

イベントハンドリングに関する重要な概念として、レスポンダチェーンがあります。

レスポンダチェーンを一言でいうと「誰がイベントを処理するか」というバトンの受け渡しです。バトンを持っている UIResponder オブジェクトがイベントの処理権を持ち、必要に応じてバトンをリレーしていきます。このリレーの順序を決めているのがレスポンダチェーンです。トップバッターのことをファーストレスポンダと呼び、ファーストレスポンダが最初にユーザーイベントを処理します。

ファーストレスポンダはテキスト入力とも関係します。テキスト入力のためのビューには UITextField（1 行）と UITextView（複数行）があります。テキスト入力欄がタップされると、UIKit はタップされたビューをファーストレスポンダに設定します。そうすることで、キーボードが表示され入力の受付が開始されます。

プログラムでテキスト入力を開始または終了するには、UITextField または UITextView をファーストレスポンダにしたり、逆にファーストレスポンダではなくすことで実現可能です。

ファーストレスポンダの切り替えには、下記のメソッドを利用します（コード 2.5）。なお、ビューをファーストレスポンダにするには、ビュー階層が確立されている必要があります。

コード 2.5：ファーストレスポンダの切り替えメソッド（UIResponder）

```
// ファーストレスポンダにする
func becomeFirstResponder() -> Bool

// ファーストレスポンダではなくする
func resignFirstResponder() -> Bool
```

また、ビューコントローラでテキスト入力の開始を制御するには、viewDidAppear(_:) で becomeFirstResponder() を呼び出します。これより前のタイミングでは、まだビュー階層が完成していないため、対象ビューをファーストレスポンダに設定することはできません。

Section

2-2 Storyboard と Auto Layout

Storyboard の登場は歴史を遡ること数年、iOS 5 の時代です。Storyboard の登場でアプリ開発は格段に楽になりました。Storyboard を利用すると、画面構成や画面遷移、個々の画面内の部品まで、簡単に実装できます。コード量の削減に繋がることはいうまでもなく、また、昨今の多様化するデバイスサイズにも柔軟に対応できます。本書のチュートリアルでも Storyboard の使用を前提としています。

2-2-1 Storyboard エディタの使い方

Storyboard エディタは 3 パートに分かれています。左から Document Outline、エディタ、ユーティリティです（図 2.1）。右側のユーティリティエリアはさらに上下に分割され、上側にはエディタ上の部品を設定するユーティリティ、下側には部品庫でもある Object Library などが表示されます。

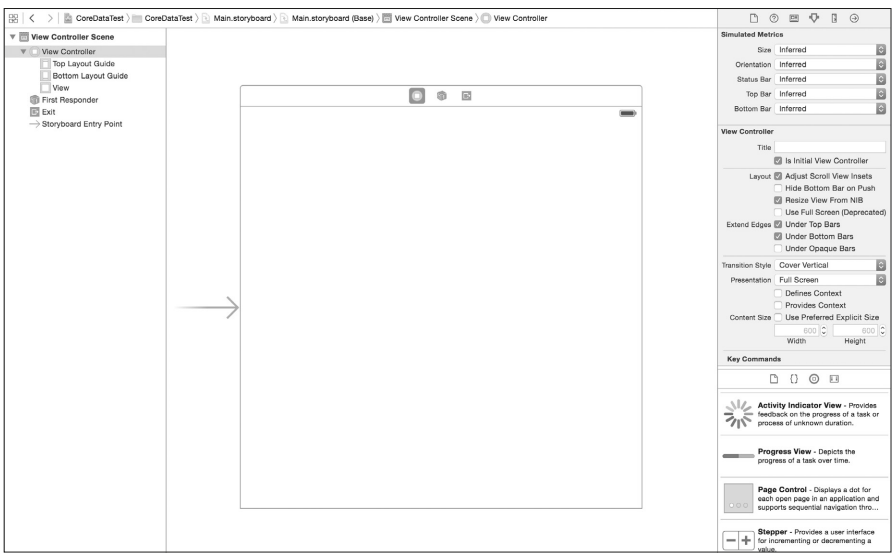


図 2.1：Storyboard エディタ

ユーティリティエリアの機能

下図はユーティリティエリア上部にある 6 つのボタンです（図 2.2）。これらのボタンで、Inspector と呼ばれる各種設定画面の表示を切り替えます。

図 2.2：Inspector ボタン

左側のボタンから順に、次の Inspector が表示されます。

- File Inspector
- Quick Help Inspector
- Identity Inspector
- Attributes Inspector
- Size Inspector
- Connections Inspector

頻繁に利用されるのが、「Identity Inspector」、「Attributes Inspector」、「Size Inspector」の 3 つです。チュートリアルでも頻繁に利用するため覚えておいてください。

Identity Inspector は、主にクラス名の指定で使います（図 2.3）。

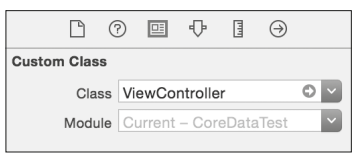


図 2.3：Identity Inspector

Attirubtes Inspector は、ビューやビューコントローラの初期設定をします。例えば、UILabel の Attributes Inspector は次図の通りです（図 2.4）。

図には表示されていませんが、UILabel のスーパークラスである UIView に関する設定もこの画面の下に続きます。

Size Inspector では、サイズに関する設定をします（図 2.5）。

ビューのレイアウトを Auto Layout に任せるのであれば、[Constraints] より下の部分が重要です。上部にある frame の設定エリアで直接レイアウト指定を行っても、Auto Layout により再配置されます。

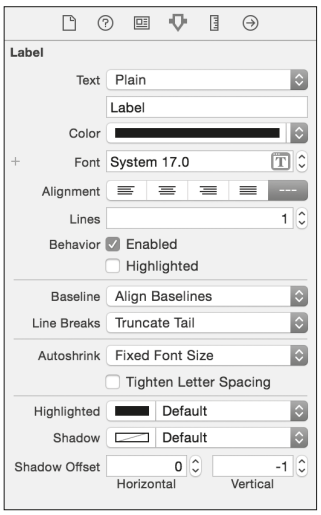


図 2.4：Attributes Inspector（UILabel）

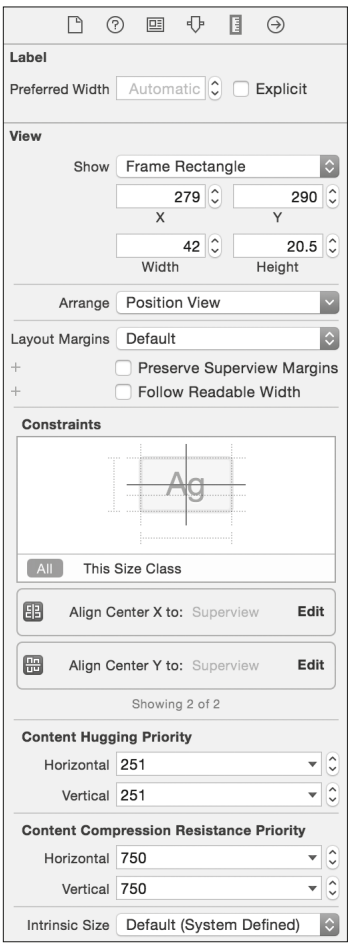


図 2.5：Size Inspector

試し読みはお楽しみ
いただけましたか？

ここからはManatee
おすすめの商品を
ご紹介します。

Tech Book Zone Manatee

3.6
2017

おすすめ電子書籍

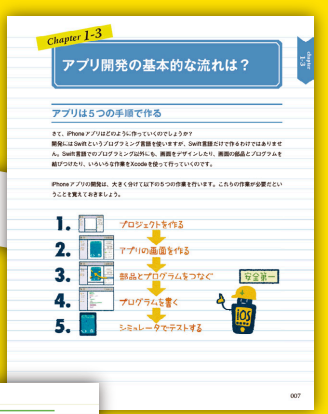
Tech Book Zone Manatee

1

体験型の本書でプログラミングの第1歩を踏みだそう!

『やさしくはじめるiPhoneアプリ作りの教科書【Swift 3&Xcode 8.2対応】』は、iPhoneアプリを作ってみたい初心者のための入門書です。プログラミングが初めての人、苦手意識がある人でも楽しく学んでいけるよう、なるべくやさしく、イラストや図をたくさん使って解説しています。本書では実際にサンプルアプリを作りながら学んでいきますが、イラストによる解説で、一歩ずつ丁寧に、iPhoneアプリ作りの基本と楽しさを学べます。

イラストによる解説で、プログラミングがはじめての人でも学べる



作成するサンプルアプリはシンプルで、意味を理解しながら作っていける

やさしくはじめる iPhoneアプリ作りの教科書【Swift 3&Xcode 8.2 対応】

Swift (iOS開発)

マイナビ出版
森巧尚(著者)、
まつむらまきお(イラスト)
312 ページ　価格：3,002 円(PDF)

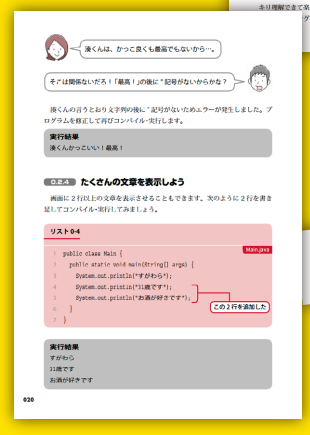
2

豊富なイラストで「なぜ?」を解消! Javaの第一歩を踏み出そう

『スッキリわかるJava入門 第2版』は、Javaの基礎から初学者には難しいとされるオブジェクト指向まで、膨らむ疑問にしっかり対応しました。Javaプログラミングの「なぜ?」がわかる解説と約300点の豊富なイラストで、楽しく・詳しく・スッキリとマスターできる構成となっています。「なんとなくJavaを使っているけれど、オブジェクト指向の理解には自信がない」「学習の途中で挫折してしまった」という方にもおススメです。

300ものイラストで楽しく・詳しく・スッキリとマスター!

「プログラミング」



会話のやりとりの中にも、開発現場でのヒントが詰め込まれている

スッキリわかる Java 入門 第2版

Java

インプレス
中山清喬・国本大悟(著者)
658 ページ
価格：2,376 円(PDF)

幅広いジャンルで活躍！ C# がキホンから学べる本

3

基礎からわかる C#

本書はプログラミングの経験がある人を対象とした、プログラミング言語「C#」の入門書です。C# の概要から基本的な文法、特徴的な機能まで、わかりやすく解説しています。C# 6.0 の新機能についても解説しています。

シーアンドアール研究所
西村誠（著者）
168 ページ　価格：1,944 円（PDF）

&

プログラミング未経験でも Android アプリを開発！

4

イラストでよくわかる Android アプリのつくり方 Android Studio 対応版

親しみやすいイラストや、ステップバイステップでの丁寧な解説が基本コンセプト。開発環境「Android Studio」に対応し、Android のプログラムを作りながら、自然に Java というプログラム言語の知識が身につきます。

インプレス
羽山博・めじろまち（著者）
価格：2,138 円（PDF）

&

JavaScript を網羅的に取り上げた骨太の 1 冊

5

JavaScript 逆引きハンドブック

JavaScript の逆引きリファレンスの決定版。JavaScript の機能を網羅的に取り上げていて、骨太の 1 冊になっています。JavaScript の基本的な処理や便利な Tips はもちろん、HTML5 の API についても数多く掲載しています。

シーアンドアール研究所
古籏一浩（著者）
993 ページ　価格：3,694 円（PDF）

初めてのウェブ開発も安心 Ruby の文法を基礎から解説

6

改訂 3 版 基礎 Ruby on Rails

Ruby の文法やオブジェクト指向の考え方を初歩から解説。アプリケーションのモックアップ作り、データベースを導入し、ログイン・ログアウト機能を加え、最終的にはメンバーや記事の管理ページまでできあがります。

インプレス
黒田努・佐藤和人（著者）
536 ページ　価格：3,240 円（PDF）

&

初歩から順に理解できる PHP とデータベース

7

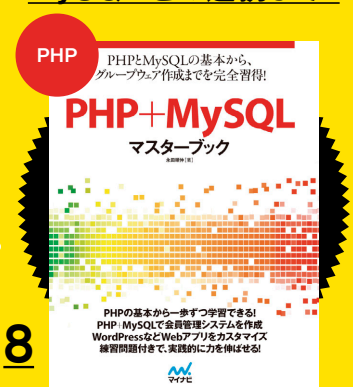
いちばんやさしい PHP の教本

PHP とデータベースの基本を順番に学んで、実践的なプログラムを完成させていく PHP の入門書です。大事なポイントや勘違いしやすいポイントは講師がフォロー。セミナーを受けている感覚で読み進められます。

インプレス
柏岡秀男・池田友子（著者）
240 ページ　価格：1,836 円（PDF）

&

PHP5.4 の基本から MySQL との連携まで！



8

PHP+MySQL マスターブック

この一冊で PHP と MySQL の基本と Web アプリケーションの構築法について学習できる実践的なプログラミング入門です。現場必須のプログラム構築法、API の活用法から、セキュリティ技術まで詳しく解説します。

マイナビ出版
永田順伸（著者）
384 ページ　価格：2,916 円（PDF）

はじめてプログラミングに触れる前に読んでおこう

9

プログラミングの基本から手取り足取りじっくり解説

10

必ずアルゴリズムの意味がわかるようになる入門書!

11

プログラミングの世界へようこそ

全くの初心者がプログラミングを勉強したいとき、さまざまな疑問が湧いてきます。「どの言語を覚えればいい？」「文系でも大丈夫？」本書はプログラミングに触れる前に知っておきたい基本をイラスト付きで解説します。

マイナビ出版
尾川一行・中川聡(著者)
192 ページ　価格：1,933 円(PDF)

目指せプログラマー！プログラミング超入門

本書は Windows 開発の標準ツールとも言える「Visual Studio」を使い、C# というプログラミング言語でプログラミングの基本を学びます。最終的には、ちょっとしたアクションゲームが作れるくらいになるのが目標です。

マイナビ出版
掌田津耶乃(著者)
312 ページ　価格：2,074 円(PDF)

楽しく学ぶ アルゴリズムとプログラミングの図鑑

図解とイラストを豊富に使ったアルゴリズムの入門書。アルゴリズムとは『問題を解決するための考え方』です。それが分かってきたら、8 種類のプログラミング言語を使ったサンプルプログラムを実際に試しましょう。

マイナビ出版　森巧尚(著者)、まつむらまきお(イラスト)
300 ページ　価格：2,689 円(PDF)

プログラミングの初心者がPython 3 を学ぶのに最適

12

Go 言語の基礎から応用までポイントがよくわかる

13

R 言語の機能を目的から見つけ出せる！

14

基礎 Python 基礎シリーズ

プログラミングの初心者を対象にした Python 3 の入門書です。変数の取り扱いから、リスト、タプルといった Python 固有のデータの操作、制御構造や関数などについて、初心者でも基礎から学習できるように説明しました。

インプレス
大津真(著者)
312 ページ　価格：2,894 円(EPUB)

改訂 2 版 基礎からわかる Go 言語

Google が開発したプログラミング言語「Go」の基礎から応用までをわかりやすく解説した 1 冊です。最新の Go 1.4 のバージョンに対応して改訂しました。Linux、Mac OS X、Windows の各環境に対応しています。

シーアンドアール研究所　古川昇(著者)
240 ページ　価格：2,138 円(EPUB)

改訂 3 版 R 言語逆引きハンドブック

本書では、最新バージョンの R 3.3.0 に対応し、R 言語の機能を目的から探すことができます。統計が注目を集めるなか、R を利用するユーザーも増えています。初心者でも使えるように、導入から丁寧に解説しています。

シーアンドアール研究所
石田基広(著者)
800 ページ　価格：4,860 円(PDF)