iPhoneアプリケーションチュートリアル

iPhone

2009-08-10

Apple Inc.
© 2009 Apple Inc.
All rights reserved.

本書の一部あるいは全部を Apple Inc. から書面による事前の許諾を得ることなく複写複製（コピー）することを禁じます。また、製品に付属のソフトウェアは同梱のソフトウェア使用許諾契約書に記載の条件のもとでお使いください。書類を個人で使用する場合に限り 1 台のコンピュータに保管すること、またその書類にアップルの著作権表示が含まれる限り、個人的な利用を目的に書類を複製することを認めます。

Apple ロゴは、米国その他の国で登録された Apple Inc. の商標です。

キーボードから入力可能な Apple ロゴについても、これを Apple Inc. からの書面による事前の許諾なしに商業的な目的で使用すると、連邦および州の商標法および不正競争防止法違反となる場合があります。

本書に記載されているテクノロジーに関しては、明示または黙示を問わず、使用を許諾しません。本書に記載されているテクノロジーに関するすべての知的財産権は、Apple Inc. が保有しています。 本書は、Apple ブランドのコンピュータ用のアプリケーション開発に使用を限定します。

本書には正確な情報を記載するように努めました。ただし、誤植や制作上の誤記がないことを保証するものではありません。

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

アップルジャパン株式会社
〒163-1450 東京都新宿区西新宿
3 丁目20 番2 号
東京オペラシティタワー
http://www.apple.com/jp/

Apple, the Apple logo, Cocoa, iPod, Mac, Mac OS, Objective-C, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

Instruments and iPhone are trademarks of Apple Inc.

Apple Inc. は本書の内容を確認しておりますが、本書に関して、明示的であるか黙示的であるかを問わず、その品質、正確さ、市場性、または特定の目的に対する適合性に関して何らかの保証または表明を行うものではありません。その結果、本書は「現状有姿のまま」提供され、本書の品質または正確さに関連して発生するすべての損害は、購入者であるお客様が負うものとします。

いかなる場合も、Apple Inc. は、本書の内容に含まれる瑕疵または不正確さによって生じる直接的、間接的、特殊的、偶発的、または結果的損害に対する賠償請求には一切応じません。そのような損害の可能性があらかじめ指摘されている場合においても同様です。

上記の損害に対する保証および救済は、口頭や書面によるか、または明示的や黙示的であるかを問わず、唯一のものであり、その他一切の保証にかわるものです。 Apple Inc. の販売店、代理店、または従業員には、この保証に関する規定に何らかの変更、拡張、または追加を加える権限は与えられていません。

一部の国や地域では、黙示あるいは偶発的または結果的損害に対する賠償の免責または制限が認められていないため、上記の制限や免責がお客様に適用されない場合があります。 この保証はお客様に特定の法的権利を与え、地域によってはその他の権利がお客様に与えられる場合もあります。

目次

序章 **はじめに 7**

この書類の構成 7

第 1 章 **チュートリアルの概要とデザインパターン 9**

チュートリアルの概要 9
デザインパターン 10
    Delegation 10
    Model-View-Controller 11
    Target-Action 11

第 2 章 **プロジェクトの作成 13**

Xcode 13
アプリケーションのブートストラップ 15
まとめ 18

第 3 章 **View Controllerの追加 19**

View Controllerクラスの追加 19
View Controllerプロパティの追加 21
View Controllerインスタンスの作成 22
ビューのセットアップ 24
管理処理 24
実装のソースリスト 25
アプリケーションのテスト 25
まとめ 26

第 4 章 **nibファイルの調査 27**

Interface Builder 27
nibファイルの調査 27
    File's Owner 28
    Viewアウトレット 29
nibファイルのロード 30
アプリケーションのテスト 30
まとめ 31

第 5 章 **ビューの設定 33**

ユーザインターフェイス要素の追加 33

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

View Controllerのインターフェイス宣言 36
接続の作成 37
テスト 39
まとめ 40

第 6 章 **View Controllerの実装 41**

プロパティ 41
changeGreeting:メソッド 42
テキストフィールドのデリゲート 42
まとめ 43

第 7 章 **トラブルシューティング 45**

コードおよびコンパイラの警告 45
nibファイルの接続の確認 45
メソッド名のデリゲート 45

第 8 章 **次に学ぶこと 47**

ユーザインターフェイス 47
ユーザインターフェイス要素のプログラムによる作成 47
デバイスへのインストール 48
追加機能 48

付録 A **コードリスト 51**

HelloWorldAppDelegate 51
  ヘッダファイル：HelloWorldAppDelegate.h 51
  実装ファイル：HelloWorldAppDelegate.m 51
MyViewController 52
  ヘッダファイル：MyViewController.h 52
  実装ファイル：MyViewController.m 52

改訂履歴 **書類の改訂履歴 55**

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

第 2 章 **プロジェクトの作成 13**

図 2-1 アプリケーションのブートストラップ 16

第 3 章 **View Controllerの追加 19**

図 3-1 MyViewController 21

第 5 章 **ビューの設定 33**

図 5-1 ユーザインターフェイス要素を含むビューとガイド線の表示 34

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

はじめに

このチュートリアルでは、簡単なiPhoneアプリケーションの作成方法を示します。この文書の目的は、利用可能なすべての機能を網羅的に解説することではなく、テクノロジーをいくつか紹介し、開発プロセスの基礎となる情報を提供することです。

Cocoa Touchを使用してiPhone向けの開発を始めたばかりの人は、是非この文書をお読みください。また、この文書を読むためには、一般的なコンピュータプログラミングの基礎、特にObjective-C言語に多少慣れている必要があります。これまでにObjective-Cを使用したことのない方は、少なくとも『*Learning Objective-C: A Primer*』を読んでおいてください。

ここでの目標は、洗練されたアプリケーションを作成することではなく、以下を理解することです。

- Xcodeを使用してプロジェクトを作成および管理する方法
- すべてのiPhone開発の基礎となる基本デザインパターンおよびテクノロジー
- Interface Builderの使いかたの基礎
- 標準のユーザインターフェイスコントロールを使用して、アプリケーションをユーザ入力に応答させる方法

もう1つの目標は、iPhoneの開発ツールおよびテクノロジーを十分に理解するために読む必要があるその他の文書を示すことです。

> **重要：** このチュートリアルを進めるには、iPhone Dev Centerから入手できるiPhone SDKとデベロッパツールをインストールする必要があります。
>
> この文書では、iPhone SDK v3.0で利用可能なツールについて説明します（Xcodeのバージョンが3.1.3以降であること確認してください）。

この書類の構成

この文書は次の各章で構成されています。

- 「チュートリアルの概要とデザインパターン」 （9 ページ）
- 「プロジェクトの作成」 （13 ページ）
- 「View Controllerの追加」 （19 ページ）
- 「nibファイルの調査」 （27 ページ）
- 「ビューの設定」 （33 ページ）
- 「View Controllerの実装」 （41 ページ）
- 「トラブルシューティング」 （45 ページ）

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

- 「次に学ぶこと」 (47 ページ)

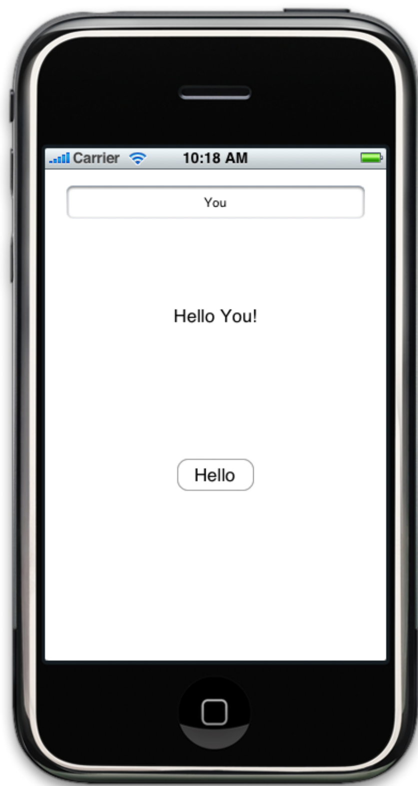
2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

チュートリアルの概要とデザインパターン

この章では、これから作成するアプリケーションの概要と、そこで使用するデザインパターンについて説明します。

チュートリアルの概要

このチュートリアルでは、非常に簡単なアプリケーションを作成します。このアプリケーションには、テキストフィールド、ラベル、およびボタンがそれぞれ1つあります。テキストフィールドに名前を入力してボタンを押すと、ラベルのテキストが更新されて“Hello, <Name>!”と表示されます。



これは、非常に簡単なアプリケーションですが、Cocoa Touchを使用したすべてのiPhone開発の基礎となるデザインパターン、ツール、およびテクニックが盛り込まれています。Cocoa Touchは、UIKitフレームワークとFoundationフレームワークから構成されています。これらは、iPhone OSでグラフィカルなイベント駆動型アプリケーションを実装するために必要な基本的なツールおよび基盤を提供します。また、ユーザの連絡先情報など、デバイス機能にアクセスするための主要なサービスを提供するフレームワークがほかにもいくつか含まれています。Cocoa Touch、およびiPhone OSに

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

おけるその位置付けの詳細については、『*iPhone OS Technology Overview*』を参照してください。これから使用する主なデザインパターンについては、「デザインパターン」（10 ページ）で説明します。

このチュートリアルでは、ユーザインターフェイスについては、ほとんど触れません。ただし、プレゼンテーションは、優れたiPhoneアプリケーションのために極めて重要な要素です。本格的なアプリケーションのために、ユーザインターフェイスを強化する方法を理解するには、『*iPhone Human Interface Guidelines*』を読んで、このチュートリアルに基づくサンプルコード(HelloWorld)を研究してください。

デベロッパにとっては、View Controllerの仕組みと、それがiPhoneアプリケーションのアーキテクチャにどのように組み込まれるのかを理解するための出発点になるでしょう。

デザインパターン

『*Cocoa Fundamentals Guide*』のデザインパターンの章をまだ読んでいない場合は、必ずお読みください。ただし、これから使用する主なパターンは、以下のとおりです。

- Delegation
- Model-View-Controller
- Target-Action

以下は、これらのパターンの簡単な概要と、アプリケーションでの使用場所です。

Delegation

Delegationパターンは、あるオブジェクトが、デリゲートとして指定された別のオブジェクトに定期的にメッセージを送信して、入力を要求したり、イベントの発生を通知したりするデザインパターンです。このパターンは、クラス継承の代替手段として、再利用可能なオブジェクトの機能を拡張するために使用します。

このアプリケーションでは、メイン起動ルーチンが終了したことと、カスタム設定を開始できることを、アプリケーションオブジェクトがデリゲートに通知します。また、ビューのセットアップと管理を行うコントローラのインスタンス作成をデリゲートに依頼します。さらに、テキストフィールドは、ユーザが「改行(Return)」キーをタップしたときに、それをデリゲート（この場合は、同じコントローラ）に通知します。

デリゲートのメソッド群は、通常、1つのプロトコルにまとめられます。プロトコルは、基本的には、単なるメソッドのリストです。クラスがプロトコルに従っていれば、そのクラスが、プロトコルの必須（任意の場合もある）メソッドを実装していることが保証されます。デリゲートプロトコルは、1つのオブジェクトがデリゲートに送信できるすべてのメッセージを定義しています。プロトコル、およびそれがObjective-Cにおいて果たす役割の詳細については、『*The Objective-C 2.0 Programming Language*』の「Protocols」の章を参照してください。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

Model-View-Controller

Model-View-Controller (MVC)デザインパターンは、アプリケーション内のオブジェクトに、次の3つの役割を与えます。

Modelオブジェクトは、ゲームにおける宇宙船(SpaceShips)やロケット(Rockets)、生産性型アプリケーションにおけるToDo項目や連絡先(Contacts) 、描画アプリケーションにおける円(Circles)や正方形(Squares)などのデータを表します。

このアプリケーションでは、データは非常に単純で、単なる文字列です。また、この文字列は、1つのメソッドの外では実際には使われません。したがって、厳密に言えば、必要さえありません。しかし、ここで重要なのは原則です。ほかのアプリケーションでは、Modelはもっと複雑で、さまざまな場所からアクセスされます。

Viewオブジェクトは、データの表示方法を知っており、ユーザがデータを編集できるようにしていることがあります。

このアプリケーションでは、ほかのいくつかのビュー（ユーザからの入力情報を取得するテキストフィールド、ユーザ入力に基づいてテキストを表示するもう1つのテキストフィールド、および、後者のテキストの更新をユーザが指示するためのボタン）を含むメインビューが必要です。

Controllerオブジェクトは、ModelとViewの仲介役をします。

このアプリケーションでは、Controllerオブジェクトが、入力テキストフィールドからデータを取得して文字列に格納し、もう1つのフィールドを適切に更新します。この更新は、ボタンによって送信されるアクションの結果として、開始されます。

Target-Action

Target-Actionは、ユーザイベント（クリックやタップなど）に応じて、コントロールオブジェクト（ボタン、スライダなど）から別のオブジェクト（ターゲット）にメッセージを送信（アクション）して、メッセージを受け取ったオブジェクトがそれを解釈してアプリケーション固有の命令として処理するメカニズムです。

このアプリケーションでは、ボタンがタップされると、そのボタンが、ユーザ入力に基づいてModelとViewを更新するように、Controllerに指示します。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

プロジェクトの作成

この章では、Xcodeを使用してプロジェクトを作成し、アプリケーションがどのように起動するかを学びます。

Xcode

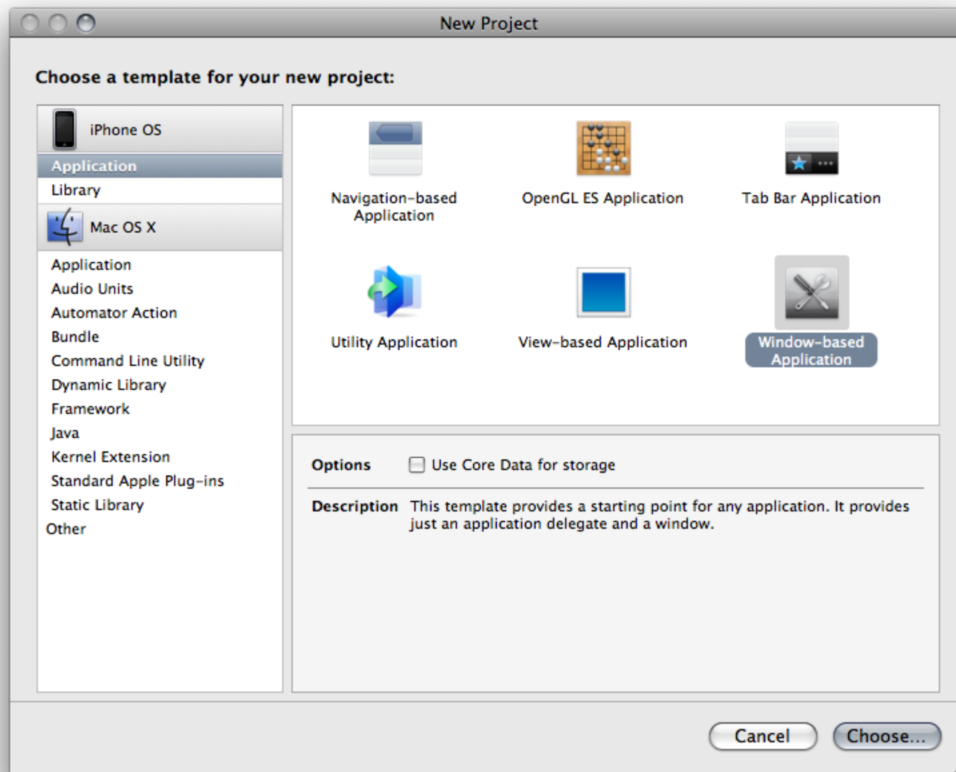
iPhone用のアプリケーションを作成するために使用する主たるツールは、Xcode（AppleのIDE（統合開発環境））です。また、Xcodeを使用して、Cocoaユーティリティやコマンドラインユーティリティなど、その他にもさまざまなプロジェクトを作成できます。

注： 表記規則として、このチュートリアルの中で、読者が実行しなければならない手順を含む段落の先頭には>> が付いています（その後に、箇条書きリストが続く場合もあります）。

コードのリストでは、Xcodeのテンプレートファイルに含まれているコメントは示しません。

>>Xcode（デフォルトでは、`/Developer/Applications`にあります）を起動します。次に、「ファイル(File)」＞「新規プロジェクト(New Project)」を選んで、新規プロジェクトを作成します。次と同様の新しいウインドウが表示されます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.



注： 「Use Core Data for storage」オプションが表示されない場合、iPhone OS SDKのバージョン3.0がインストールされていることを確認してください。Xcodeのバージョンは3.1.3以降である必要があります。

>> 「Window-Based Application」を選択して、「選択(Choose)」をクリックします（ストレージにCore Dataを使用するオプション（「Use Core Data for storage」）を選択*しないで*ください。この例では、Core Dataは使用しません）。

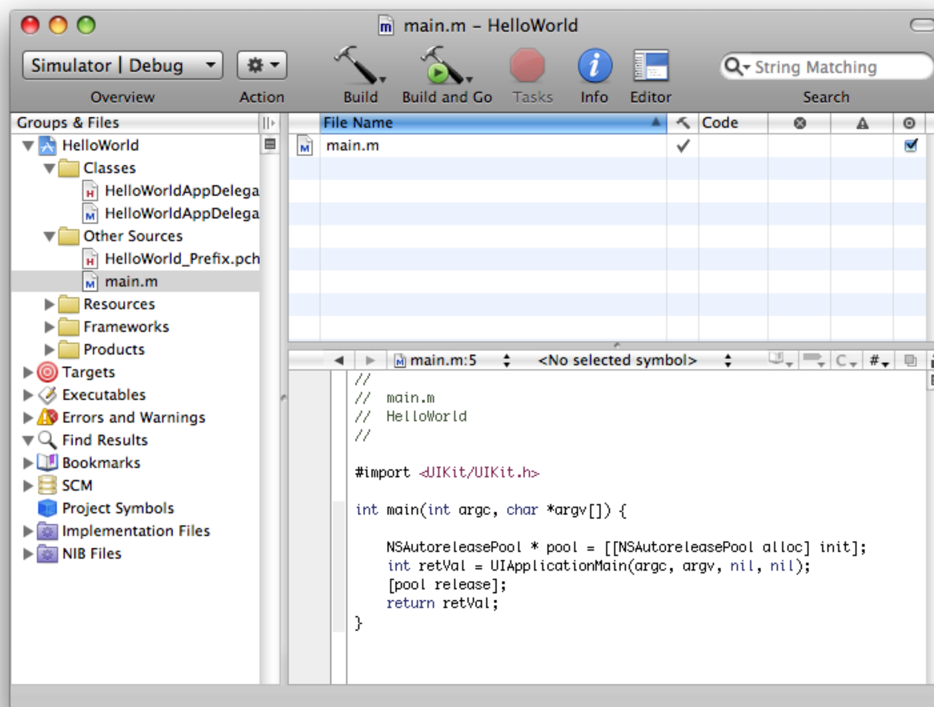
プロジェクトの保存先を選択するための画面が表示されます。

>>プロジェクト用として適切な場所（デスクトップまたは独自のプロジェクトディレクトリなど）を選択して、プロジェクトに名前（`HelloWorld`など）を設定し、「保存(Save)」をクリックします。

注： このチュートリアルでは、プロジェクトに`HelloWorld`という名前を付けたという前提で説明を進めます。したがって、アプリケーションデリゲートクラスは`HelloWorldAppDelegate`という名前になります。プロジェクトに別の名前を付けた場合は、アプリケーションデリゲートクラスは、*プロジェクト名*`AppDelegate`という名前になります。

次のような新規プロジェクトウインドウが表示されます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.



Xcodeをまだ使用したことがない場合は、少し時間を取ってXcodeを操作してみてください。プロジェクトウインドウの構成と、ファイルの編集および保存などの基本的なタスクの実行方法を理解するには、『*Xcode Workspace Guide*』をお読みください。これで、アプリケーションをビルドして実行し、Simulatorでどのように見えるかを確認できます。

>>「ビルド(Build)」>「ビルドして進行（実行）(Build and Go (Run))」を選ぶか、ツールバーの「ビルドして進行(Build and Go)」ボタンをクリックします。

iPhone Simulatorアプリケーションが自動的に起動されます。アプリケーションが起動されると、単純な白い画面が表示されます。白い画面がどのようにして表示されるかを理解するには、アプリケーションがどのように起動されるかを理解する必要があります。

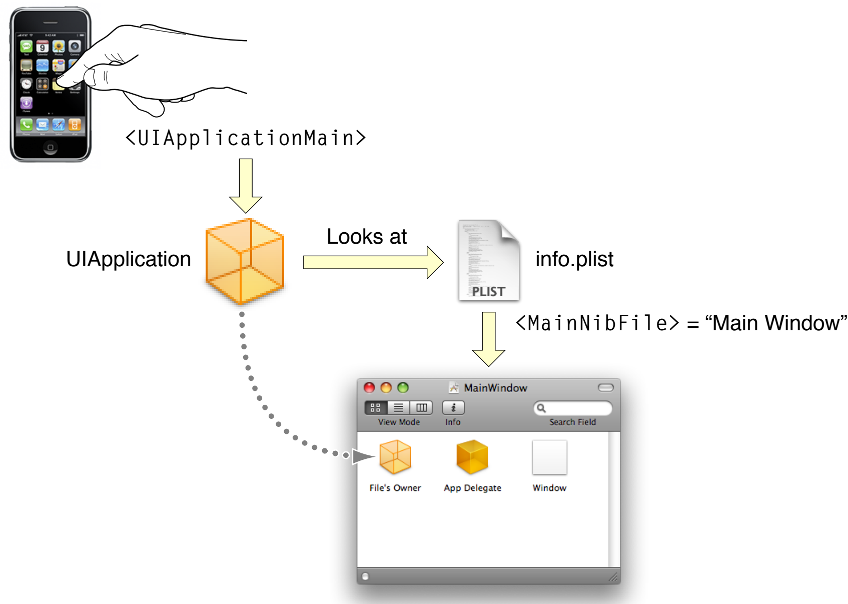
>> Simulatorを終了します。

アプリケーションのブートストラップ

先ほど作成したテンプレートプロジェクトによって、基本的なアプリケーション環境はすでにセットアップされています。すなわち、アプリケーションオブジェクトの作成、ウインドウサーバへの接続、実行ループの開始などを行います。図2-1に示すように、ほとんどの作業は`UIApplicationMain`関数によって実行されます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

図 2-1 アプリケーションのブートストラップ



main.mのmain関数は、次のようにしてUIApplicationMain関数を呼び出します。

```
int retVal = UIApplicationMain(argc, argv, nil, nil);
```

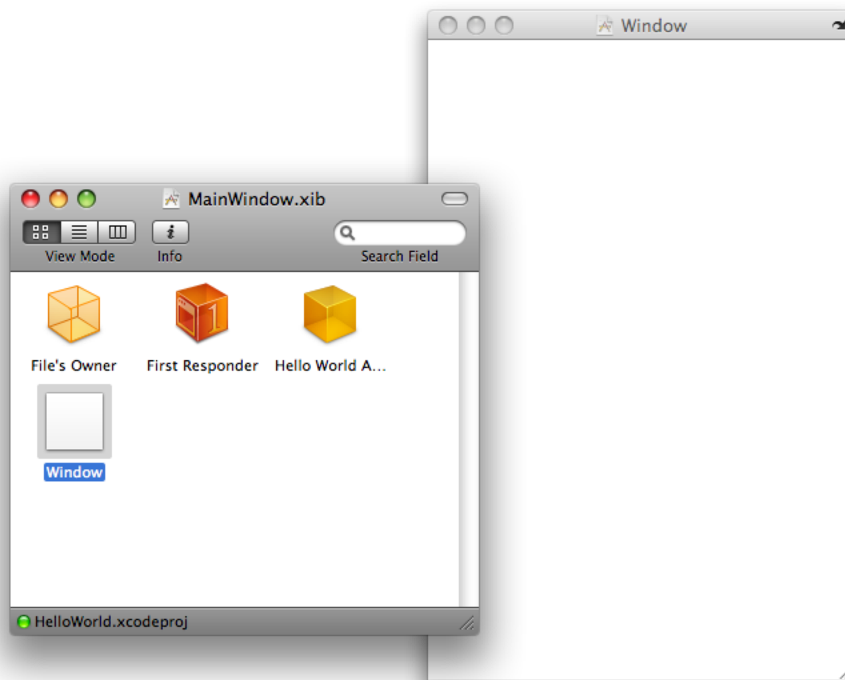
これによって、UIApplicationのインスタンスが作成されます。また、アプリケーションのInfo.plistプロパティリストファイルがスキャンされます。Info.plistファイルは、アプリケーションについての情報（アプリケーション名、アイコンなど）を含む辞書です。このファイルには、アプリケーションオブジェクトがロードする**nibファイル**の名前（NSMainNibFileキーで指定されている）が含まれている場合もあります。nibファイルには、ユーザインターフェイス要素およびその他のオブジェクトのアーカイブが含まれています。これらについては、このチュートリアルの後半で詳しく説明します。プロジェクトのInfo.plistファイルには、次のような記述があります。

```
<key>NSMainNibFile</key>
<string>MainWindow</string>
```

これは、アプリケーションの起動時に、MainWindowというnibファイルがロードされることを意味します。

>> このnibファイルを参照するには、プロジェクトウインドウのResourcesグループのMainWindow.xibをダブルクリックします（このファイルの拡張子は“xib”ですが、慣習として“nibファイル”と呼ばれています）。Interface Builderが起動し、このファイルが開きます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

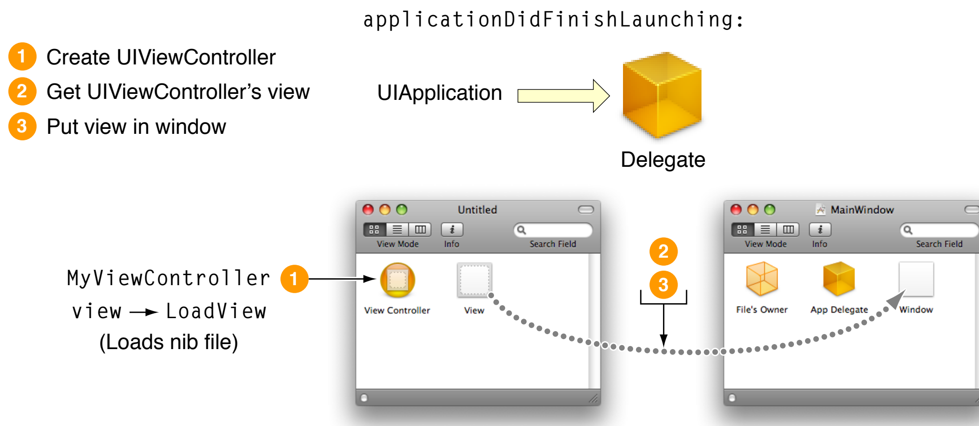


Interface Builder書類には、以下の4つの項目が含まれています。

- **File's Owner**プロキシオブジェクト。File's Ownerオブジェクトは、実際には`UIApplication`のインスタンスです。File's Ownerについては、後半の「File's Owner」（28 ページ）で説明します。
- **First Responder**プロキシオブジェクト。First Responderは、このチュートリアルには使用しません。このオブジェクトの詳細については、『*iPhone Application Programming Guide*』の「Event Handling」をお読みください。
- アプリケーションの**デリゲート**として設定された`HelloWorldAppDelegate`のインスタンス。デリゲートについては、次のセクションで説明します。
- ウインドウ(Window)。ウインドウは白に設定された背景を持ち、起動時に表示されるように設定されています。アプリケーションが起動された時に表示されるのが、このウインドウです。

アプリケーションの起動が完了すると、カスタマイズを行うことができます。一般的なパターン（および次の章で説明するもの）を次の図に示します。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.



アプリケーションオブジェクトは、セットアップが完了すると、applicationDidFinishLaunching:メッセージをデリゲートに送信します。デリゲートは通常、ユーザインターフェイス自身を構成するのではなく、**View Controller**オブジェクト（ビューを管理するための特別なコントローラ。これは、「Model-View-Controller」（11 ページ）で説明されているように、Model-View-Controllerデザインパターンに従っています）を作成します。デリゲートは、そのビューのView Controllerを要求し（要求に応じてView Controllerが作成する）、ウインドウのサブビューとして追加します。

まとめ

この章では、新規プロジェクトを作成し、アプリケーションの起動プロセスの仕組みを学びました。次の章では、View Controllerのインスタンスを定義して作成します。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

View Controllerの追加

このアプリケーションでは、2つのクラスを必要とします。1つはXcodeのアプリケーションテンプレートが提供する、アプリケーションデリゲートクラスです。このクラスのインスタンスは、nibファイル内に作成されます。もう1つはView Controllerクラスです。このクラスを実装し、そのインスタンスを作成する必要があります。

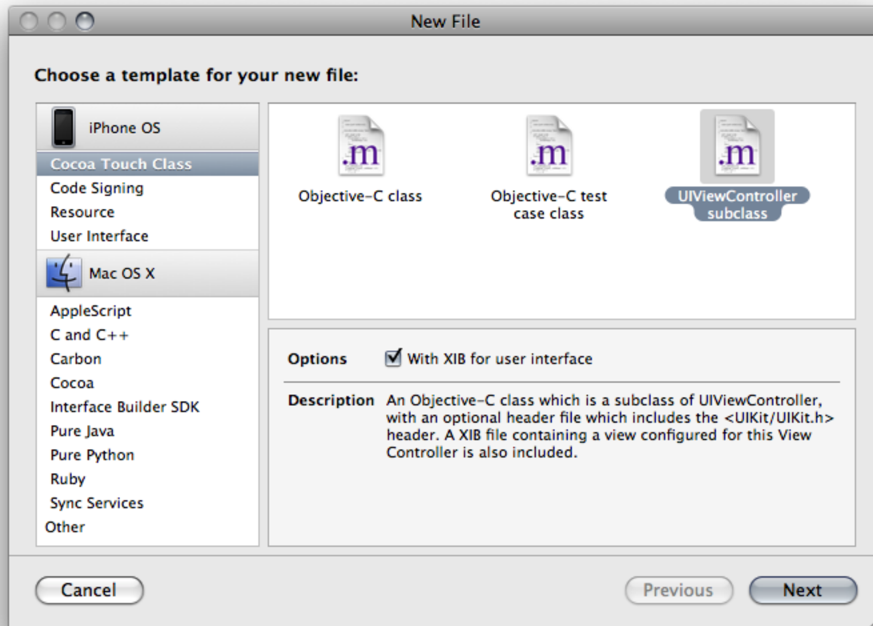
View Controllerクラスの追加

View Controllerオブジェクトは、ほとんどのiPhoneアプリケーションで中心的な役割を果たします。その名のとおり、このオブジェクトはビューの管理を担当します。しかし、iPhoneでは、それに加えてナビゲーションおよびメモリ管理も支援します。ここでは、後者の機能は使用しませんが、今後の開発のために、それらの機能を認識しておくことは重要です。UIKitにはUIViewControllerという特別なクラスがあります。このクラスは、View Controllerに期待されるデフォルトの動作のほとんどをカプセル化しています。独自のアプリケーション用に、この動作をカスタマイズするには、サブクラスを作成する必要があります。

>>Xcodeで、プロジェクトオーガナイザから、プロジェクト（「グループとファイル(Groups & Files)」リストの最上位にあるHelloWorld）またはClassesグループフォルダのどちらかを選択します。新規ファイルが現在の選択に追加されます。

>> 「新規ファイル(New File)」ウインドウで、「ファイル(File)」＞「新規ファイル(New File)」を選びます。Cocoa Touch Classesグループを選択して、UIViewController subclassを選択します。「オプション(Options)」セクションでは、「With XIB for user interface」を選びます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.



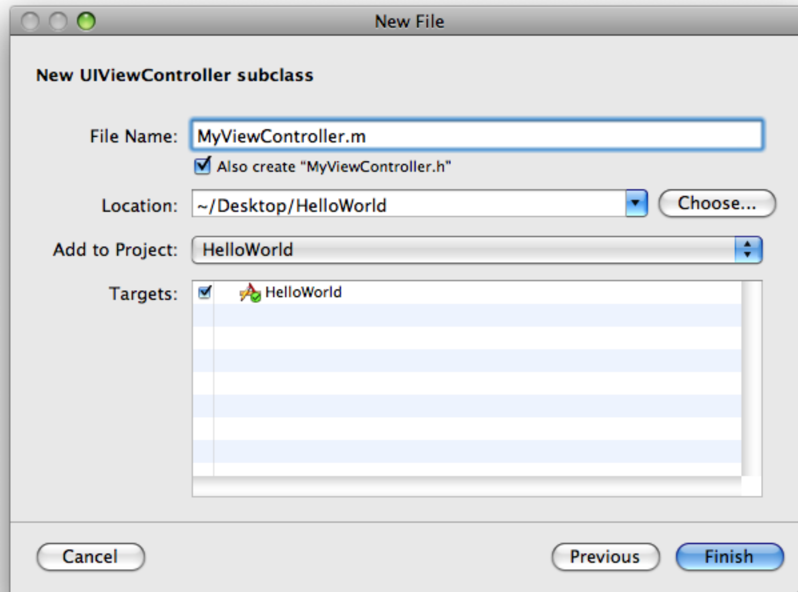
注： 「With XIB for user interface」オプションが表示されない場合、iPhone OS SDKのバージョン3.0がインストールされていることを確認してください。Xcodeのバージョンは3.1.3以降である必要があります。

「With XIB for user interface」を選択することは、nibファイルを作成してView Controllerに関連付け、プロジェクトに追加することを意味します（nibファイルについては、次の章で詳細に説明します）。

>> 「次へ(Next)」をクリックし、次の画面で、このファイルに新しい名前（`MyViewController`など）を付けます（慣習として、クラス名は大文字で始めます）。以下に示すように、`.m`ファイルと`.h`の両方が作成されて、これらのファイルがプロジェクトに追加されます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

図 3-1 MyViewController



>> 「完了(Finish)」を押して、ファイルがプロジェクトに追加されたことを確認します。

この新規ソースファイルを見ると、さまざまなメソッドのスタブ実装がすでに存在することがわかります。さしあたり、これらだけで十分です。次の作業は、このクラスのインスタンスを作成することです。

View Controllerプロパティの追加

アプリケーションが実行されている間、View Controllerを確実に存続させるには、それを、アプリケーションデリゲート（これも、アプリケーションが実行されている間、存続します）のインスタンス変数に追加します。その理由を理解するには、『*Memory Management Programming Guide for Cocoa*』を参照してください。

このインスタンス変数は、`MyViewController`クラスのインスタンスになります。ただし、`MyViewController`クラスについてコンパイラに指示せずにこの変数を宣言すると、コンパイルエラーになります。ヘッダファイルをインポートすることもできますが、一般に、Cocoaでは**前方宣言**を行います。これは、`MyViewController`がどこかで定義されるので、今はそれをチェックする必要がないことをコンパイラに約束するものです（これによって、2つのクラスが相互参照する必要がある場合や、お互いにヘッダファイルをインクルードしている場合に生じる循環を回避することもできます）。そして、ヘッダファイル自体を実装ファイルにインポートします。

>> アプリケーションデリゲートヘッダファイル（`HelloWorldAppDelegate.h`）で、`HelloWorldAppDelegate`のインターフェイス宣言の前にこの前方宣言を追加します。

```
@class MyViewController;
```


2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

>> 中括弧の間に次の行を追加することによって、インスタンス変数を追加します。

```
MyViewController *myViewController;
```

>> また、閉じ中括弧の後の@endの前に、次のようなプロパティ宣言を追加します。

```
@property (nonatomic, retain) MyViewController *myViewController;
```

プロパティについては、『*The Objective-C 2.0 Programming Language*』の「Declared Properties」の章を参照してください。基本的に、この宣言は、HelloWorldAppDelegateのインスタンスがプロパティを持ち、このプロパティには、getterメソッドとsetterメソッドであるmyViewControllerとsetMyViewController:を使用してアクセスできることを表します。また、このインスタンスは、このプロパティを保持する(retain)ことを表します（保持については、後で詳しく説明します）。

作業が正しく進んでいれば、HelloWorldAppDelegateクラスのインターフェイスファイル（HelloWorldAppDelegate.h）は次のようになります（コメントは示しません）。

```
#import <UIKit/UIKit.h>

@class MyViewController;

@interface HelloWorldAppDelegate :NSObject <UIApplicationDelegate> {
    UIWindow *window;
    MyViewController *myViewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) MyViewController *myViewController;

@end
```

これで、View Controllerのインスタンスを作成できます。

View Controllerインスタンスの作成

アプリケーションデリゲートにView Controllerプロパティを追加したら、View Controllerのインスタンスを実際に作成して、プロパティの変数として設定する必要があります。

>> アプリケーションデリゲートクラスの実装ファイル（HelloWorldAppDelegate.m）に、MyViewControllerのインスタンスを作成します。それには、applicationDidFinishLaunching:メソッドの実装の最初の文として、次のコードを追加します。

```
MyViewController *aViewController = [[MyViewController alloc]
            initWithNibName:@"MyViewController" bundle:[NSBundle mainBundle]];
[self setMyViewController:aViewController];
[aViewController release];
```

このわずか3行で、非常にたくさんのことが行われます。それを以下に示します。

- View Controllerクラスのインスタンスを作成して初期化します。
- アクセサメソッドを使用して、この新規View ControllerをmyViewControllerインスタンス変数に設定します。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

setMyViewController:を個別に宣言しておらず、これはプロパティ宣言の一部として暗黙的に含まれていることに注意してください（「View Controllerプロパティの追加」（21 ページ）を参照）。

- メモリ管理ルールに従って、このView Controllerを解放します。

allocを使用してView Controllerオブジェクトを作成したら、initWithNibName:bundle:を使用してそれを初期化します。このinitメソッドでは、1番目に、このコントローラがロードするnibファイルの名前を指定し、2番目に、nibファイルが含まれているバンドルを指定します。バンドルは、ファイルシステム内の場所を抽象化したもので、アプリケーションで使用されるコードおよびリソースをグループ化します。ファイルシステム内のリソースの場所を自分で指定することに対してバンドルを使用することの利点は、バンドルでは便利で簡単なAPIが提供され（バンドルオブジェクトは、名前だけリソースを見つけることができます）、ローカリゼーションも考慮されている点です。バンドルの詳細については、『*Resource Programming Guide*』を参照してください。

慣習として、allocメソッドを使用して作成したオブジェクトは、作成者が所有します（『*Memory Management Programming Guide for Cocoa*』の「Memory Management Rules」を参照）。次のことも行います。

- 自分が作成したオブジェクトの所有権を放棄する。
- 通常は、初期化メソッドとは別の場所で、アクセサメソッドを使用して、インスタンス変数を設定する。

この実装の2行目では、アクセサメソッドを使用してインスタンス変数を設定しています。次に、3行目で、releaseを使用して、所有権を放棄しています。

上記を実装する方法は、ほかにもいくつかあります。たとえば、この3行を、次の2行に置き換えることもできます。

```
MyViewController *aViewController = [[[MyViewController alloc]
            initWithNibName:@"MyViewController" bundle:[NSBundle mainBundle]]
autorelease];
[self setMyViewController:aViewController];
```

このバージョンでは、新規View Controllerの所有権を放棄する方法として、autoreleaseを使用しています。所有権の放棄は、以降のどこかのタイミングで行われます。これを理解するには、『*Memory Management Programming Guide for Cocoa*』の「Autorelease Pools」をお読みください。ただし、一般的には、autoreleaseの使用はできる限り避けてください。それは、releaseよりもリソースを消費する操作だからです。

最後の行を次と置き換えることもできます。

```
self.myViewController = aViewController;
```

ドット記法は、元の実装におけるものとまったく同じアクセサメソッド(setMyViewController:)を呼び出します。このドット記法は、単に構文を簡略化する手法で、特に、ネストした式を使用する場合に簡略になります。どちらの構文を選ぶかは、主として個人の好みによりますが、ドット記法には、プロパティと組み合わせて使用した場合の利点が、ほかにもいくつかあります。詳細については、『*The Objective-C 2.0 Programming Language*』の「Declared Properties」を参照してください。ドット構文の詳細については、『*The Objective-C 2.0 Programming Language*』の「Objects, Classes, and Messaging」にある「Dot Syntax」を参照してください。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

ビューのセットアップ

View Controllerは、依頼があると、ビューの管理と設定を担当します。したがって、ウインドウのコンテンツビューを直接作成するのではなく、View Controllerにそのビューの作成を依頼して、それをウインドウのサブビューとして追加します。

>> View Controllerを解放した後に、次の行を追加します。

```
UIView *controllersView = [myViewController view];
[window addSubview:controllersView];
```

同じことを、次の1行で実行することもできます。

```
[window addSubview:[myViewController view]];
```

ただし、2行に分けた方が、1行の場合よりも、メモリ管理の側面（先に示したメモリ管理の側面の対極の側面）を強調するのに役立ちます。『*Memory Management Programming Guide for Cocoa*』の「Memory Management Rules」に列挙されているメソッドを使用して、Controller Viewを作成したわけではないので、作成者は、返されたオブジェクトを所有していません。したがって、それをウインドウに単純に渡したら、後は忘れてかまいません（オブジェクトを解放する必要はありません）。

テンプレートから次の最終行を指定します。

```
[window makeKeyAndVisible];
```

ビューが完成したウインドウが画面上に表示されます。実際のコンテンツが表示される前にブランクの画面がユーザに表示されないように、ウインドウが表示される前にビューを追加します。

管理処理

やり残している作業がもう少しあります。View Controllerのヘッダファイルをインポートして、アクセサメソッドを合成する必要があります。また、メモリ管理ルールに従って、deallocメソッド内でView Controllerを忘れずに解放します。

>> アプリケーションデリゲートクラスの実装ファイル(HelloWorldAppDelegate.m)で、以下のことを行います。

- ファイルの先頭で、MyViewControllerのヘッダファイルをインポートします。

  ```
  #import "MyViewController.h"
  ```

- このクラスの@implementationブロックで、このView Controllerのアクセサメソッドを合成するようにコンパイラに指示します。

  ```
  @synthesize myViewController;
  ```

- deallocメソッドの最初の文で、View Controllerを解放します。

  ```
  [myViewController release];
  ```


2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

実装のソースリスト

作業が正しく進んでいれば、HelloWorldAppDelegateクラスの実装(HelloWorldAppDelegate.m)は以下のようになります。

```
#import "MyViewController.h"
#import "HelloWorldAppDelegate.h"

@implementation HelloWorldAppDelegate

@synthesize window;
@synthesize myViewController;


- (void)applicationDidFinishLaunching:(UIApplication *)application {

    MyViewController *aViewController = [[MyViewController alloc]
              initWithNibName:@"MyViewController" bundle:[NSBundle mainBundle]];
    [self setMyViewController:aViewController];
    [aViewController release];

    UIView *controllersView = [myViewController view];
    [window addSubview:controllersView];
    [window makeKeyAndVisible];
}


- (void)dealloc {
    [myViewController release];
    [window release];
    [super dealloc];
}

@end
```

アプリケーションのテスト

これでアプリケーションのテストができます。

>>プロジェクトをコンパイルして実行します（「ビルド(Build)」>「ビルドして実行(Build and Run)」を選ぶか、Xcodeツールバーで「ビルドして実行(Build and Run)」をクリックします）。

アプリケーションはエラーなしでコンパイルされ、iPhone Simulatorに再び白い画面が表示されます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

まとめ

この章では、新たにView Controllerクラスとそれに付随するnibファイルを追加しました。アプリケーションデリゲートで、View Controller用のインスタンス変数およびアクセサメソッドを宣言しました。また、アクセサメソッドを合成し、その他の管理処理も実行しました。しかし最も重要なのは、View Controllerのインスタンスを作成して、そのビューをウインドウに渡したことです。次の章では、Interface Builderを使用して、このコントローラがビューをロードするために使用するnibファイルを設定します。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

nibファイルの調査

Interface Builderアプリケーションを使用して、nibファイルを作成し、設定します。ここでは、2つの重要な概念である、アウトレットとFile's Ownerプロキシオブジェクトを紹介します。

Interface Builder

Interface Builderは、ユーザインターフェイスを作成するために使用するアプリケーションです。Interface Builderは、ソースコードは作成しませんが、これを利用してオブジェクトを直接操作して、それらをnibファイルと呼ばれるアーカイブに保存できます。

用語の定義： Interface Builder書類の拡張子は".xib"ですが、歴史的に拡張子は".nib"（「NextStep Interface Builder」の略）であったため、口語的に「nibファイル」と呼ばれます。

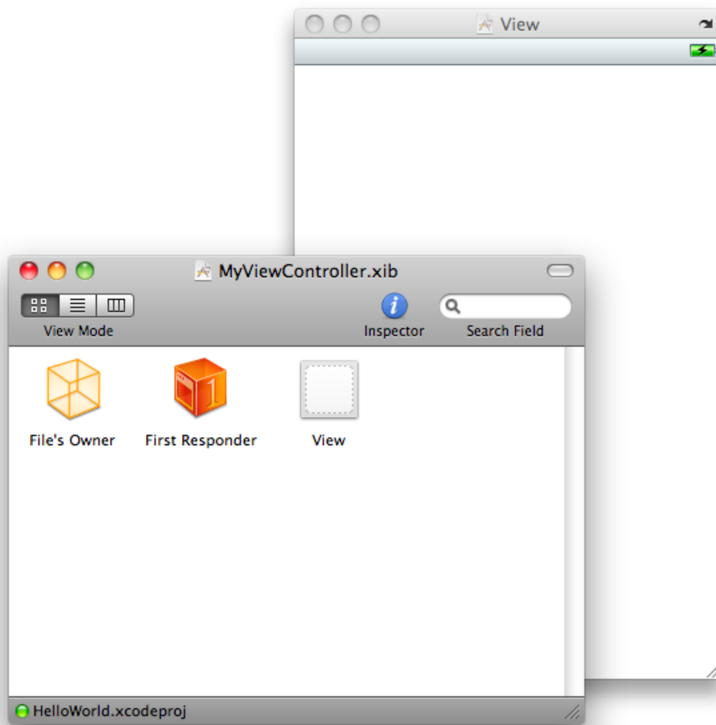
実行時にnibファイルがロードされると、これらのオブジェクトが展開されて、このファイルを保存したときと同じ状態に（オブジェクト間のすべての接続関係も含めて）復元されます。Interface Builderの詳細については、『*Interface Builder User Guide*』をお読みください。

nibファイルの調査

>> Xcodeで、View ControllerのXIBファイル(`MyViewController.xib`)をダブルクリックして、Interface Builderのファイルを開きます。

ファイルには、File's Ownerプロキシ、First Responderプロキシ、およびViewの3つのオブジェクトが含まれています。Viewは、編集できるように、独立したウインドウに表示されます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.



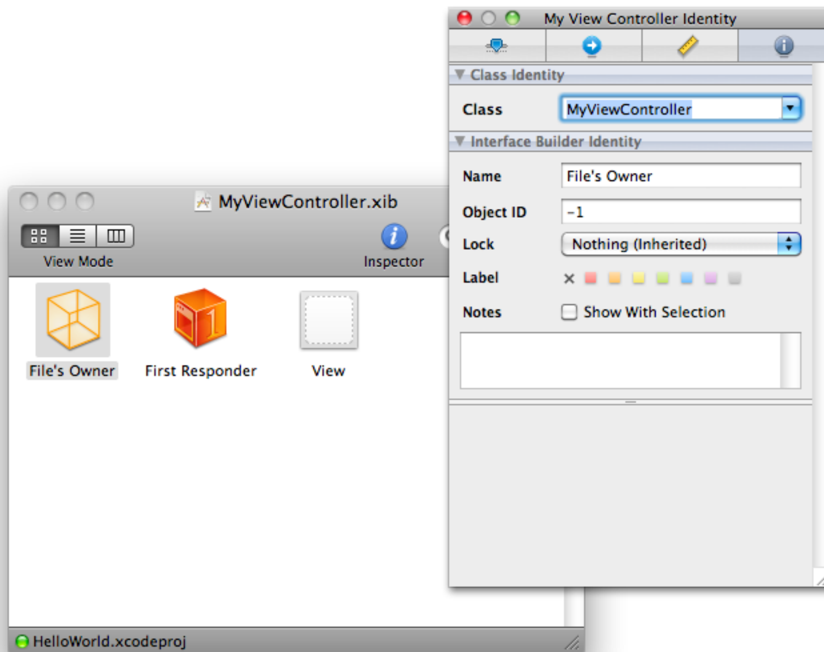
File’s Owner

Interface Builderの書類では、インターフェイスに追加するその他のオブジェクトとは対照的に、File’s Ownerオブジェクトは、nibファイルのロード時に作成されません。これは、ユーザインターフェイスの所有者になるために設定するオブジェクトであり、通常は、インターフェイスをロードするオブジェクトです。これについては、『*Resource Programming Guide*』で詳しく説明しています。アプリケーションでは、File’s Ownerは`MyViewController`のインスタンスになります。

File’s Ownerに対して適切な接続を行えるようにするには、Interface Builderが、オブジェクトFile’s Ownerがどのような種類のものであるか知る必要があります。「Identity」インスペクタを使用して、オブジェクトのクラスをInterface Builderに通知します。View Controllerのクラスとともにnibファイルを作成するときに実際に設定しましたが、ここでインスペクタを表示すると参考になります。

>> Interface Builderの書類ウインドウで「File’s Owner」アイコンを選択し、「Tools」＞「Identity Inspector」を選んで、次のようなIdentityインスペクタを表示します。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.



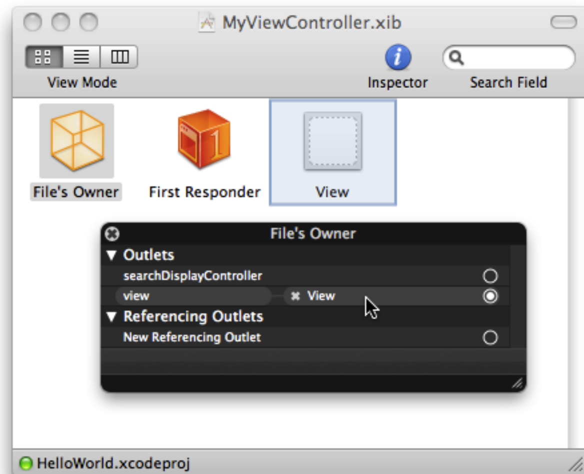
「Class Identity」セクションの「Class」フィールドに、`MyViewController`と表示されているはずです。これは、File's Ownerが`MyViewController`のインスタンスになるという、Interface Builderに対する単なる約束であることを理解することが重要です。単なるクラスの設定は、File's Ownerがそのクラスのインスタンスになることを保証するものではありません。nibファイルをロードすると、File's Ownerは設定したどのようなオブジェクトにもなります。異なるクラスのインスタンスの場合、nibファイルで行われる接続は正しく確立されません。

Viewアウトレット

インスペクタパネルを使用して、オブジェクトの接続の検索、作成、切断を行うことができます。

>> Interface Builder書類ウインドウで、Controlキーを押しながらFile's OwnerをクリックしてFile's Ownerの接続を表示する半透明のパネルを表示します。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.



この時点で作成できる接続は、View Controllerの`view`アウトレットだけです。**アウトレット**とは、単に、nibファイル内の項目に接続している属性（通常はインスタンス変数）です。アウトレット接続は、nibファイルがロードされ`UIView`インスタンスが展開されたときに、View Controllerの`view`インスタンス変数がそのビューに設定されることを意味します。

nibファイルのロード

View Controllerは、`loadView`メソッドの中で、nibファイルを自動的にロードします。`initWithNibName:bundle:`にロードする最初の引数として、nibファイルの名前を指定したことを思い出してください（「View Controllerインスタンスの作成」（22 ページ）を参照）。通常、`loadView`メソッドは、View Controllerの存続期間中に一度だけ呼び出されて、そのビューを作成するために使用されます。View Controllerの`view`メソッドを呼び出すと、ビューがまだ作成されていなければ、コントローラは自動的に`loadView`メソッドを呼び出します（メモリ不足の警告を受け取った結果、View Controllerがビューを破棄している場合は、必要であれば、`loadView`が再度呼び出されて、そのビューが再作成されます）。

View Controllerのビューをプログラムで作成する場合は、`loadView`をオーバーライドして、独自の実装でビューを作成できます。

`initWithNibName:bundle:`を使用してView Controllerを初期化するときに、ビューのロード後に追加の設定を行う場合は、View Controllerの`viewDidLoad`メソッドをオーバーライドします。

`NSBundle`のインスタンスを使用して、nibファイルを自分でロードすることもできます。nibファイルのロードの詳細については、『*Resource Programming Guide*』を参照してください。

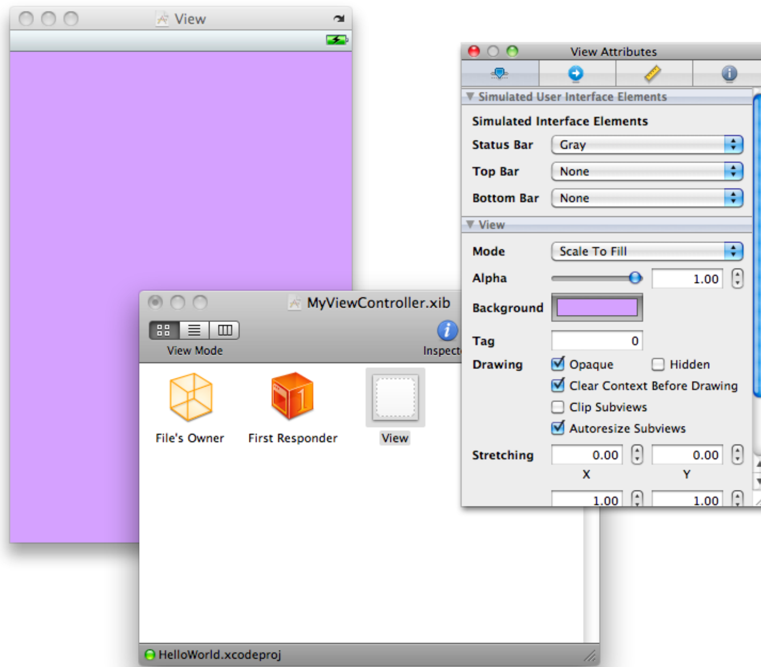
アプリケーションのテスト

アプリケーションが正しく動作していることを確認するには、ビューの背景色を白以外の色に設定して、アプリケーションの起動後に新しい色が表示されることを確認できます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

>> Interface Builderで、ビューを選択し、「Tools」>「Attributes Inspector」を選んで、「Attributes」インスペクタを表示します。

>> Backgroundカラーウェルのフレームをクリックして、「Colors」パネルを表示し、違う色を選択します。



>> nibファイルを保存します。

>> プロジェクトをコンパイルして実行できます（ツールバーの「ビルドして進行(Build and Go)」ボタンをクリックします）。

アプリケーションはエラーなしでコンパイルされ、iPhone Simulatorに再び適切な色の画面が表示されます。

>> ビューの背景色を白に復元して、nibファイルを保存します。

まとめ

この章では、nibファイルを調査して、アウトレットについて学習し、ビューの背景色を設定しました。また、リソースのロードについて、およびView Controllerがどのようにnibファイルをロードするかについても学びました。

次の章では、ビューにコントロールを追加します。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

ビューの設定

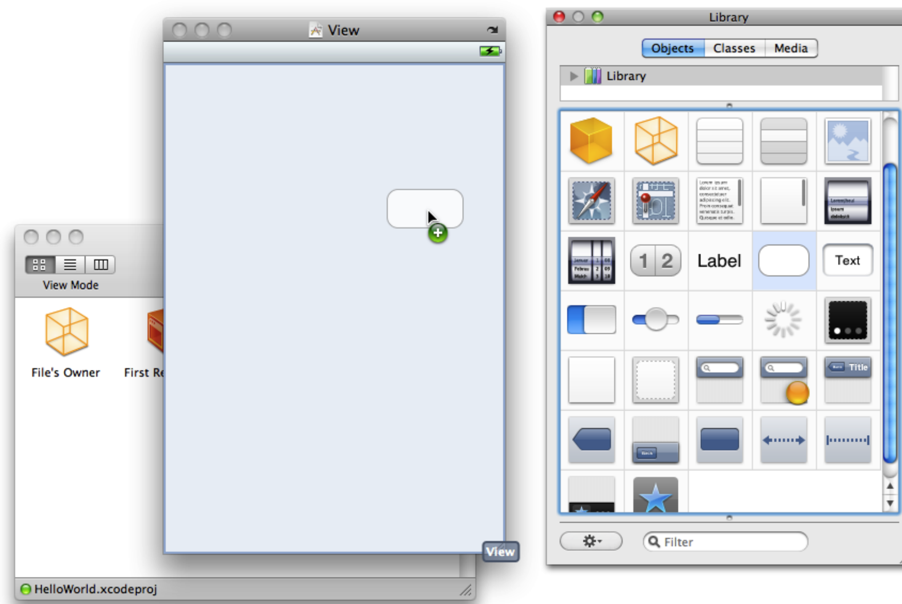
Interface Builderには、nibファイルに追加できるオブジェクトのライブラリがあります。これらの中には、ボタン、テキストフィールドなどのユーザインターフェイス要素もあれば、View ControllerなどのControllerオブジェクトもあります。すでに、nibファイルにはビューが含まれているので、今度は、ボタンとテキストフィールドを追加する必要があります。

ユーザインターフェイス要素の追加

ユーザインターフェイス要素をInterface Builderライブラリからドラッグして追加します。

>> Interface Builderで、「Tools」＞「Library」を選び、ライブラリウインドウを表示します。

描画アプリケーションと同様に、ライブラリからビューアイテムをドラッグして、View上にドロップできます。



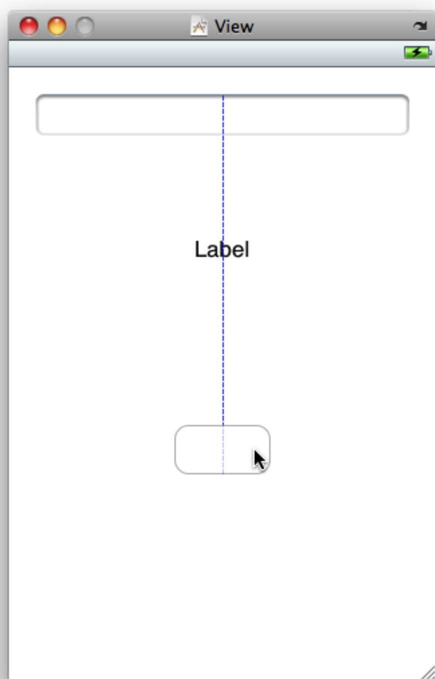
>>テキストフィールド（`UITextField`）、ラベル（`UILabel`）、ボタン（`UIButton`）をビューへ追加します。

次に、サイズ変更ハンドルを使用して、アイテムを適切なサイズに変更できます。また、ドラッグして位置を変更することもします。View内でアイテムを移動すると、位置合わせガイドが青い破線として表示されます。

>> 要素を次のようにレイアウトします。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

図 5-1 ユーザインターフェイス要素を含むビューとガイド線の表示

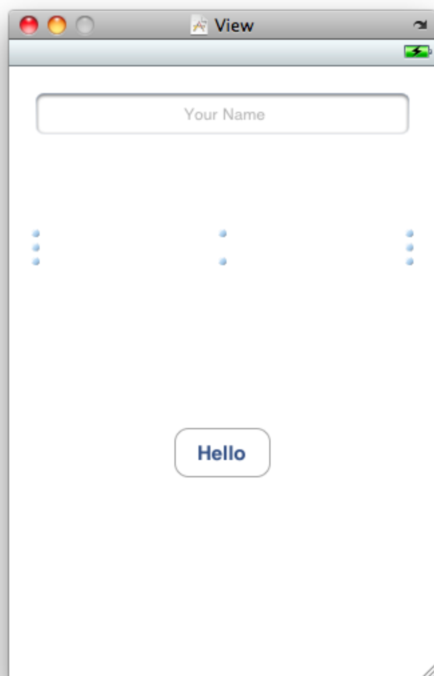


>> 以下の手順で変更を加えます。

1. テキストフィールドにプレースフォルダ文字列“`Your Name`”を追加します。それには、「Text Field Attributes」インスペクタで、この文字列を入力します。
2. ラベルのサイズを変更して、Viewの幅まで広げます。
3. 「Label Attributes」インスペクタを使用するか、ラベル内のテキストを直接選択（ダブルクリックして選択）してDeleteを押すことによって、ラベルからテキスト（「ラベル」）を削除します。
4. ボタン内をダブルクリックして、“`Hello`”とタイプし、ボタンにタイトルを追加します。
5. インスペクタを使用して、テキストフィールドとラベルを中央揃えに設定します。

その結果、ビューは次のようになります。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.



>> ラベルの「Label Attributes」インスペクタの「View」セクションで、「Clear Context Before Drawing」を選択します。これによって、あいさつが更新されたときに、新しい文字列が描画される前に、以前の文字列が削除されます。この設定を行わないと、文字列が重ね書きされます。

テキストフィールドには、ほかにもいくつかの変更を加えます。最初に適用する変更はテキストフィールドに該当することは明白かもしれませんが、その他はそれほど明白ではありません。まず、名前の頭文字が自動的に大文字になるようにします。次に、テキストフィールドに関連付けられているキーボードを、名前の入力用に設定して、キーボードに「Done」ボタンが表示されるようにします。

ここでの指針は、いつキーボードを画面に表示するか、また、テキストフィールドには何が入るかが分かっているということです。したがって、実行時にキーボードがユーザのタスクに最も適合するように、テキストフィールドを設計します。これらの設定はすべて、テキスト入力特性を使用して行います。

>> Interface Builderで、テキストフィールドを選択して「Attributes」インスペクタを表示します。「Text Input Traits」セクションで、次のように設定します。

- 「Capitalize」ポップアップメニューから「Words」を選択します。
- 「Keyboard」のタイプポップアップメニューから「Default」を選択します。
- Keyboardの「Return Key」ポップアップメニューから「Done」を選択します。

>> ファイルを保存します。

Xcodeでアプリケーションをビルドして実行すると、アプリケーションの起動時に、このユーザインターフェイス要素が配置したとおりに表示されます。ボタンをクリックすると、それがハイライト状態になり、テキストフィールド内をタップすると、キーボードが表示されます。ただし、この

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

時点では、キーボードが表示された後に、それを閉じる手段がありません。これを改善して、その他の機能を追加するためには、View Controllerとの間に適切な接続を作成する必要があります。次のセクションではこれらについて説明します。

View Controllerのインターフェイス宣言

View Controllerからユーザインターフェイスへの接続を作成するには、アウトレットを指定する必要があります（アウトレットは、単なるインスタンス変数です）。また、その非常に単純なModelオブジェクトである文字列のための宣言が必要です。

>>Xcodeを使用して、MyViewController.h内のMyViewControllerクラスに次のインスタンス変数を追加します。

```
UITextField *textField;
UILabel *label;
NSString *string;
```

>>次に、このインスタンス変数のプロパティ宣言と、changeGreeting:アクションメソッドの宣言を追加する必要があります。

```
@property (nonatomic, retain) IBOutlet UITextField *textField;
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, copy) NSString *string;
- (IBAction)changeGreeting:(id)sender;
```

IBOutletは、このインスタンス変数またはプロパティをアウトレットとして扱うように、Interface Builderに指示するためだけに使用される特殊なキーワードです。実際には何も定義されないので、コンパイル時には何の影響もありません。

IBActionは、このメソッドをターゲット／アクション接続のアクションとして扱うように、Interface Builderに指示するためだけに使用される特殊なキーワードです。これはvoidとして定義されます。

また、View Controllerは、それ自体がこのテキストフィールドのデリゲートになるため、UITextFieldDelegateプロトコルを採用しなければなりません。クラスがプロトコルを採用することを指定するには、インターフェイスで、そのクラスの継承元のクラスの名前の後に、角括弧（<>）で囲んでプロトコル名を追加します。

>>UIViewControllerの後に<UITextFieldDelegate>を追加して、UIViewControllerオブジェクトがUITextFieldDelegateプロトコルを採用することを指定します。

インターフェイスファイルは次のようになります。

```
#import <UIKit/UIKit.h>

@interface MyViewController :UIViewController <UITextFieldDelegate> {
    UITextField *textField;
    UILabel *label;
    NSString *string;
}
@property (nonatomic, retain) IBOutlet UITextField *textField;
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, copy) NSString *string;
- (IBAction)changeGreeting:(id)sender;
@end
```


2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

>> MyViewController.hを保存して、Interface Builderに変更を適用します。

このセクションの終わりでプロジェクトをテストするために、実装ファイル(MyViewController.m)に、スタブのchangeGreeting:メソッドを実装します。

>> @implementation MyViewControllerの行の後に、次のコードを追加します。

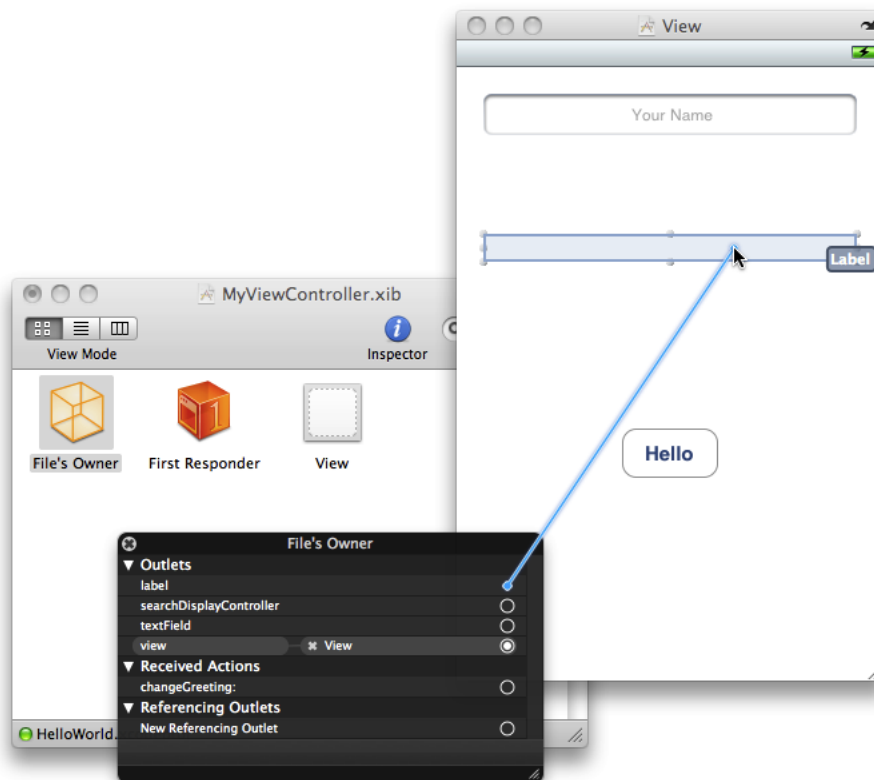
```
- (IBAction)changeGreeting:(id)sender {
}
```

>> ファイルを保存します。

接続の作成

View Controllerのアウトレットとアクションを定義したので、nibファイルに接続を確立できます。

>> labelアウトレットおよびtextFieldアウトレットを接続します。Controlキーを押しながらFile's Ownerをクリックして半透明のパネルを表示して利用可能なアウトレットとアクションをすべて表示します。このリストの右にある円から目的の要素にドラッグして接続を作成します。

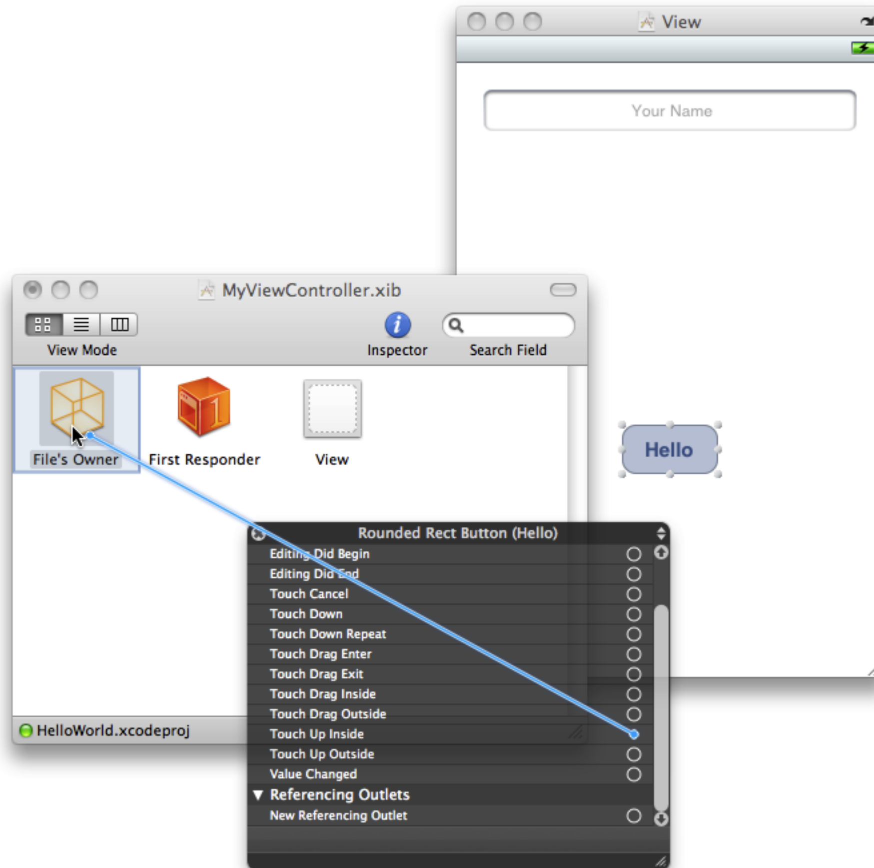


パネルの右下角にあるリサイズハンドルをドラッグすることによって、パネルをリサイズして一度に表示するアウトレットやアクションを多くしたり少なくしたりできます。アウトレットやアクションを表示するための十分なスペースがない場合、リスト表示内を移動するためのスクローラがパネルに表示されます。

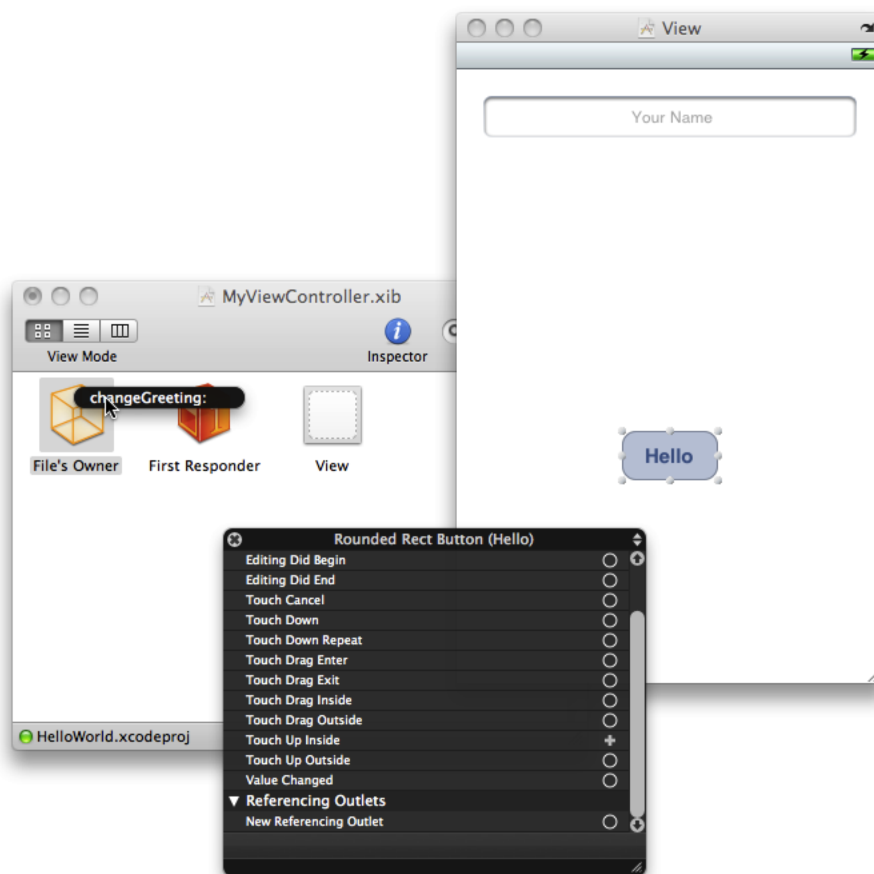
2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

Interface Builderは、誤った種類の要素に対しては接続を作成しません。たとえば、`label`プロパティは`UILabel`のインスタンスであると宣言しているため、`label`アウトレットをテキストフィールドに接続することはできません。

>>ボタンのアクションメソッドを設定します。それには、Controlキーを押しながらボタンをクリックしてインスペクタを表示して、「Touch Up Inside Events」リストの開いている円からFile's Ownerアイコンにドラッグし、File's Ownerの上に表示される半透明のパネルの`changeGreeting:`を選択します（「Touch Up Inside」接続を表示するためにインスペクタをスクロールする必要があるかもしれません）。



© 2009 Apple Inc. All Rights Reserved.



これは、アプリケーションを実行するとき、ボタン内で指を持ち上げると、ボタンからFile's Ownerオブジェクトに`changeGreeting:`メッセージが送信されることを意味します（すべてのコントロールイベントの定義については、`UIControl`を参照してください）。

ユーザがキーボードの「Return」ボタンをタップすると、テキストフィールドはそのデリゲートにメッセージを送信します。このコールバックを使用して、キーボードを閉じることができます（「テキストフィールドのデリゲート」（42 ページ）を参照）。

>>テキストフィールドのデリゲートをFile's Owner (View Controller)に設定します。それには、Controlを押しながらテキストフィールドからFile's Ownerにドラッグして、表示された半透明のパネルから`delegate`を選択します。

テスト

これで、アプリケーションをテストできます。

>> プロジェクトのビルドと実行を行います。

プロパティのアクセサメソッドをまだ実装していないため、ここでコンパイラの警告がいくつか表示されます。これらについては次の章で修正します。ボタンが動作することが分かります（ボタンをタップすると、ボタンがハイライト状態になります）。また、テキストフィールド内をタッチす

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

ると、キーボードが表示されてテキストを入力できます。ただし、このキーボードを閉じる手段はありません。そのためには、それに関連するデリゲートメソッドを実装する必要があります。これについても、次の章で行います。

まとめ

View Controllerクラスのインターフェイスに、インスタンス変数とプロパティの宣言、およびアクションメソッドの宣言を追加しました。このアクションメソッドのスタブ実装を、このクラスの実装に追加しました。また、nibファイルも設定しました。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

View Controllerの実装

View Controllerを実装するには、いくつかの手順があります。インスタンス変数を処理し（メモリ管理を含む）、changeGreeting:メソッドを実装し、ユーザが「Done」をタップしたらキーボードが閉じるようにする必要があります。

プロパティ

まず、アクセサメソッドを合成するように、コンパイラに指示する必要があります。

>>MyViewController.mファイルで、@implementation MyViewControllerの行の後に次のコードを追加します。

```
@synthesize textField;
@synthesize label;
@synthesize string;
```

このコードは、インターフェイスファイルで指定した仕様に従って、これらのプロパティのアクセサメソッドを合成するようにコンパイラに指示するものです。たとえば、stringプロパティの宣言は@property (nonatomic, copy) NSString *string;です。したがって、コンパイラは、- (NSString *)stringと- (void)setString:(NSString *)newStringの2つのアクセサメソッドを生成します。setString:メソッドでは、渡された文字列の複製が作成されます。これは、**カプセル化**を保証するのに役立ちます（渡された文字列は変更可能である可能性があるので、コントローラにはその複製を持たせます）。カプセル化の詳細については、『*Object-Oriented Programming with Objective-C*』の「Mechanisms Of Abstraction」を参照してください。

すべてのプロパティ宣言で、View Controllerがインスタンス変数を所有する定義になっているため、deallocメソッド内で、所有権を放棄しなければなりません（retainおよびcopyは所有権を持つことを意味します。『*Memory Management Programming Guide for Cocoa*』の「Memory Management Rules」を参照）。

>>MyViewController.mファイルで、親の実装を呼び出す前にインスタンス変数を解放するようにdeallocメソッドを更新します。

```
- (void)dealloc {
    [textField release];
    [label release];
    [string release];
    [super dealloc];
}
```


2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

changeGreeting:メソッド

ボタンは、タップされると、View ControllerにchangeGreeting:メッセージを送信します。次に、View Controllerは、テキストフィールドから文字列を取得して、ラベルを適切に更新します。

>> MyViewController.mファイルで、changeGreeting:メソッドの実装を次のように完成させます。

```
- (IBAction)changeGreeting:(id)sender {

    self.string = textField.text;

    NSString *nameString = string;
    if ([nameString length] == 0) {
        nameString = @"World";
    }
    NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@!",
nameString];
    label.text = greeting;
    [greeting release];
}
```

このメソッドは、いくつかの部分に分かれます。

- `self.string = textField.text;`

  これは、テキストフィールドからテキストを取得して、その結果をコントローラのstringインスタンス変数に代入します。

  この例では、この文字列インスタンス変数をほかのどこかで使用するわけではありません。ただし、その役割を理解することは重要です。この文字列インスタンス変数は、View Controllerが管理している非常に単純なModelオブジェクトです。一般に、コントローラは、アプリケーションデータを、固有のModelオブジェクト内に保持しなければなりません。アプリケーションデータをユーザインターフェイス要素内に格納してはいけません。

- @"World"は、NSStringのインスタンスで表される文字列定数です。
- initWithFormat:メソッドは、printf関数と同様に、書式文字列で指定された書式に従って新規文字列を作成します。%@は、文字列オブジェクトが置き換えられることを表します。文字列の詳細については、『*String Programming Guide for Cocoa*』を参照してください。

テキストフィールドのデリゲート

アプリケーションをビルドして実行すると、ボタンをタップしたときに、ラベルに"Hello, World!"と表示されます。テキストフィールドを選択して、入力を開始すると、テキスト入力が完了したことを示す方法とキーボードを閉じる方法がないことをに気付きます。

iPhoneアプリケーションでは、キーボードは、テキスト入力を許可する要素がファーストレスポンダになると自動的に表示され、要素がファーストレスポンダステータスでなくなると自動的に閉じられます（ファーストレスポンダの詳細については、『*iPhone Application Programming Guide*』の

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

「Event Handling」を参照してください）。キーボードと直接通信する方法はありません。しかし、テキストエントリ要素のファーストレスポンダステータスの切り替えによる副次的な効果として、キーボードを表示／非表示にすることができます。

このアプリケーションでは、ユーザがテキストフィールド内をタップしたときに、テキストフィールドがファーストレスポンダになり、つまりキーボードが表示されます。ユーザがキーボードの「Done」ボタンをタップしたときに、キーボードが非表示になります。

UITextFieldDelegateプロトコルには、ユーザが「Return」ボタンをタップしたときにテキストフィールドを呼び出す（ボタンに表示されるテキストがどのようなものであっても）、textFieldShouldReturn:メソッドが含まれています。View Controllerはテキストフィールドのデリゲートとして設定したため（「接続の作成」 （37 ページ）を参照）、このメソッドを実装して、resignFirstResponderメッセージを送信する（キーボードを閉じる効果を持つ）ことによって、テキストフィールドからファーストレスポンダステータスを強制的になくすことができます。

>>MyViewController.mファイルで、textFieldShouldReturn:メソッドを次のように実装します。

```
- (BOOL)textFieldShouldReturn:(UITextField *)theTextField {
    if (theTextField == textField) {
        [textField resignFirstResponder];
    }
    return YES;
}
```

このアプリケーションでは、テキストフィールドが1つしかないので、実際にはtheTextField == textFieldのテストを含める必要はありません。ただし、このオブジェクトが、同じタイプの複数のオブジェクトのデリゲートになる可能性があり、それらを区別する必要がある場合は、このテストが必要になります。

>>アプリケーションをビルドして実行します。アプリケーションは期待どおりの動作をします（名前の入力が完了して「Done」をタップすると、キーボードが閉じます。次に「Hello」ボタンをタップすると、ラベルに“Hello, <Your Name>!”と表示されます）。

アプリケーションが期待どおりに動作しない場合は、トラブルシューティングが必要です（「トラブルシューティング」 （45 ページ）を参照）。

まとめ

View Controllerの実装が完了しました。これで、初めてのiPhoneアプリケーションは完成です。おめでとうございます。

ここで少し時間を取って、View Controllerが、アプリケーションのアーキテクチャ全体においてどのような位置付けにあるか考えてみてください。おそらく、これから記述するほとんどのiPhoneアプリケーションで、View Controllerを使用することでしょう。

次の章では一息入れて、次に何を行うべきかについて考えます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

トラブルシューティング

このセクションでは、発生する可能性のある一般的な問題を解決するための、いくつかの方法について説明します。

コードおよびコンパイラの警告

アプリケーションが期待どおりに動作しない場合は、まず、自分のコードと「コードリスト」 （51 ページ）に示す完全なリストを比較します。

コードは、どのような警告も表示されることなく、コンパイルされなければなりません。Objective-C言語は、非常に柔軟な言語なので、コンパイラから出力されるメッセージがせいぜい警告どまりである場合もあります。通常、警告は、エラーと同等に扱う必要があります。

nibファイルの接続の確認

デベロッパとして、アプリケーションが正しく動作しない場合は、ソースコードにバグがないかチェックするのが本能です。しかし、Cocoaには、もう1つチェックしなければならない部分があります。アプリケーションの設定の多くの部分が、nibファイルに"エンコード"されている場合があるのです。正しい接続ができていなければ、アプリケーションは期待どおりに動作しません。

ボタンがタップされたときにテキストが更新されない場合は、ボタンのアクションがView Controllerに接続されていないか、View Controllerのアウトレットがテキストフィールドまたはラベルに接続されていない可能性があります。

「Done」をタップしたときに、キーボードが閉じない場合は、テキストフィールドのデリゲートを接続していない可能性があります（「接続の作成」 （37 ページ）を参照）。デリゲートを接続している場合は、もっと些細な問題が原因かもしれません（「メソッド名のデリゲート」 （45 ページ）を参照）。

メソッド名のデリゲート

デリゲートに関してよくある間違いは、デリゲートメソッド名のスペル間違いです。たとえデリゲートオブジェクトを正しく設定しても、そのデリゲートが、正確な名前を持つメソッドを実装していなければ、それを呼び出すことはできません。通常は、デリゲートメソッドの宣言をこの文書からコピーして貼り付けるのが一番良い方法です。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

次に学ぶこと

この章では、iPhone開発について、次に何を学ぶべきかの方向性を提案します。

ユーザインターフェイス

このチュートリアルでは、非常に簡単なiPhoneアプリケーションを作成しました。Cocoa Touchは、充実した開発環境を提供していますが、これまでは、表面をかじったにすぎません。ここから、さらに探求を進めましょう。このアプリケーションをスタート地点にします。第1章で述べたように、ユーザインターフェイスは、優れたiPhoneアプリケーションにとっては極めて重要です。ユーザインターフェイスの強化を試みます。画像と色を要素に追加します。背景画像とアプリケーションのアイコンを追加します。Interface Builderのインスペクタを表示して、ほかにどのような要素を設定できるかを調べましょう。

ユーザインターフェイス要素のプログラムによる作成

このチュートリアルでは、Interface Builderを使用してユーザインターフェイスを作成しました。Interface Builderを利用すると、ユーザインターフェイス要素をすばやく簡単に組み合わせることができます。しかし、ユーザインターフェイス要素をコードで作成する必要がある場合もあります（たとえば、独自のTable Viewセルを作成する場合、通常、サブビューはプログラムで作成してレイアウトします）。

まず、`MyViewController` nibファイルを開き、Viewからテキストフィールドを削除します。

View Controllerに対するビュー階層全体をコードで作成する場合は、`loadView`をオーバーライドします。ただし、ここでは、nibファイルをロードしてから、追加設定を実行します（もう1つのビューを追加します）。したがって、代わりに、`viewDidLoad`をオーバーライドします（`viewDidLoad`メソッドは、nibファイルを使用してメインビューをロードするか、`loadView`をオーバーライドするかにかかわらず、使用できる一般的なオーバライドポイントです）。

`MyViewController.m`で、次のような`viewDidLoad`の実装を追加します。

```
- (void)viewDidLoad {

    CGRect frame = CGRectMake(20.0, 68.0, 280.0, 31.0);
    UITextField *aTextField = [[UITextField alloc] initWithFrame:frame];
    self.textField = aTextField;
    [aTextField release];

    textField.textAlignment = UITextAlignmentCenter;
    textField.borderStyle = UITextBorderStyleRoundedRect;

    textField.autocapitalizationType = UITextAutocapitalizationTypeWords;
    textField.keyboardType = UIKeyboardTypeDefault;
```


2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

```
    textField.returnKeyType = UIReturnKeyDone;
    textField.delegate = self;
    [self.view addSubview:textField];
}
```

Interface Builderでテキストフィールドを作成して設定するのは簡単でしたが、それに比べると、非常にたくさんのコードがあります。

アプリケーションをビルドして実行します。アプリケーションが以前と同様に動作することを確認します。

デバイスへのインストール

適切なデバイス（iPhoneまたはiPod touch）が30ピンのUSBケーブルでコンピュータに接続されていて、iPhoneデベロッパプログラムから有効な証明書を入手している場合、プロジェクトのアクティブなSDKを（「iPhone OS Simulator」の代わりに）「iPhone OS」に設定し、プロジェクトをビルドして実行します。コードが正常にコンパイルされると、Xcodeはアプリケーションをデバイスへ自動的にアップロードします。詳細については、『*iPhone Development Guide*』を参照してください。

追加機能

次に、機能の拡張を試みます。それには、以下のようなさまざまな方向性があります。

- あらかじめ作成されたユーザインターフェイスコントロールを配置するキャンバスとしてViewを使用する代わりに、独自のコンテンツを描画したり、タッチイベントに応答したりするカスタムビューを記述します。MoveMe、Metronomeなどの例を参考にしてください。
- このアプリケーションでは、Interface Builderを使用してユーザインターフェイスをレイアウトしましたが、実際には、多くのアプリケーションがTable Viewを使用してインターフェイスをレイアウトします。Table Viewを使用すると、画面の境界をはみ出すようなインターフェイスを簡単に作成できます。その結果、ユーザは追加要素を表示するために簡単にスクロールが行えます。まず、Table Viewを使用して簡単なリストを作成する方法を調べます。サンプルコードプロジェクト（TableViewSuiteを含む）がいくつかあるので、これを参考にして独自のインターフェイスを作成します。
- Navigation ControllerとTab Bar Controllerが提供するアーキテクチャは、ドリルダウンスタイルのインターフェイスの作成を可能にし、アプリケーション内のほかのビューをユーザに選択できるようにします。Navigation Controllerは、しばしばTable Viewと連携して動作します。ただし、Navigation ControllerとTab Bar Controllerは、いずれもView Controllerと一緒に動作します。Navigation Controllerを使用するサンプルアプリケーション（SimpleDrillDownなど）を参考にし、それらを基礎にして自分のアプリケーションを作成してください。
- アプリケーションをローカライズすることによって、市場規模を拡大できることがよくあります。国際化は、アプリケーションをローカライズできるようにするためのプロセスです。国際化の詳細については、『*Internationalization Programming Topics*』を参照してください。
- パフォーマンスは、iPhoneの優れたユーザ体験のためには極めて重要です。Mac OS Xが提供しているさまざまなパフォーマンスツール（特に、Instruments）を使用して必要なリソースを最小にするようにアプリケーションを調整する方法を学びましょう。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

一番大切なことは、新しいアイディアを試して実験することです。参考になるサンプルコードはたくさんあります。また、概念とインターフェイスのプログラミングについて理解するには、ドキュメントが役立ちます。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

コードリスト

この付録では、定義する2つのクラスのコードリストを提供します。このコードリストには、ファイルテンプレートのコメントおよびその他のメソッドの実装を示しません。

HelloWorldAppDelegate

ヘッダファイル：HelloWorldAppDelegate.h

```
#import <UIKit/UIKit.h>

@class MyViewController;

@interface HelloWorldAppDelegate :NSObject <UIApplicationDelegate> {
    UIWindow *window;
    MyViewController *myViewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) MyViewController *myViewController;

@end
```

実装ファイル：HelloWorldAppDelegate.m

```
#import "MyViewController.h"
#import "HelloWorldAppDelegate.h"

@implementation HelloWorldAppDelegate

@synthesize window;
@synthesize myViewController;


- (void)applicationDidFinishLaunching:(UIApplication *)application {

    MyViewController *aViewController = [[MyViewController alloc]
              initWithNibName:@"MyViewController" bundle:[NSBundle mainBundle]];
    [self setMyViewController:aViewController];
    [aViewController release];

    UIView *controllersView = [myViewController view];
    [window addSubview:controllersView];
    [window makeKeyAndVisible];
}
```

 | © 2009 Apple Inc. All Rights Reserved.

```
- (void)dealloc {
    [myViewController release];
    [window release];
    [super dealloc];
}

@end
```

MyViewController

ヘッダファイル：MyViewController.h

```
#import <UIKit/UIKit.h>

@interface MyViewController :UIViewController <UITextFieldDelegate> {
    UITextField *textField;
    UILabel *label;
    NSString *string;
}

@property (nonatomic, retain) IBOutlet UITextField *textField;
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, copy) NSString *string;

- (IBAction)changeGreeting:(id)sender;

@end
```

実装ファイル：MyViewController.m

```
#import "MyViewController.h"

@implementation MyViewController

@synthesize textField;
@synthesize label;
@synthesize string;

- (IBAction)changeGreeting:(id)sender {

    self.string = textField.text;

    NSString *nameString = string;
    if ([nameString length] == 0) {
        nameString = @"World";
    }
    NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@!",
nameString];
    label.text = greeting;
    [greeting release];
```


2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

```
}

- (BOOL)textFieldShouldReturn:(UITextField *)theTextField {
    if (theTextField == textField) {
        [textField resignFirstResponder];
    }
    return YES;
}

- (void)dealloc {
    [textField release];
    [label release];
    [string release];
    [super dealloc];
}

// テンプレートのその他のメソッド（省略）
@end
```


2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

書類の改訂履歴

この表は「*iPhoneアプリケーションチュートリアル*」の改訂履歴です。

日付	メモ
2009-08-10	アクセサメソッドを直接使用した例のドット構文への参照を削除しました。
2009-06-15	iPhone OS 3.0向けに更新しました。
2009-01-06	誤植を修正しました。
2008-10-15	ターゲットアクションデザインパターンの説明を分かりやくしました。
2008-09-09	細かい点を訂正し、説明を分かりやすくしました。
2008-06-09	iPhone向けアプリケーションの開発を紹介する新規文書。

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.

書類の改訂履歴

2009-08-10 | © 2009 Apple Inc. All Rights Reserved.