iPhone OS View Controllerプログラミングガイド

iPhone

2009-10-19

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/

目次

図、表、リスト

はじめに

ビューオブジェクトを使用してコンテンツを画面に描画するiPhoneアプリケーションにおいて、View Controllerは、これらのビューオブジェクトとアプリケーションの残りの部分とを結び付ける極めて重要な役割を果たします。View Controllerは、Model-View-Controller設計パラダイムにおける従来のControllerオブジェクトに相当しますが、それ以上の機能を提供します。iPhoneアプリケーションでは、View Controllerは、アプリケーションの基本動作を管理するために必要なほとんどのロジックを提供しています。たとえば、View Controllerは、画面でのコンテンツの表示や消去を管理します。また、デバイスの向きの変化に対応して、ビューの向きの変更を管理します。

対象読者

View Contorllerをいつどのように使用するかを理解することは、iPhoneアプリケーションを設計するために極めて重要です。View Controlerは基本的なインフラストラクチャを提供します。これを利用すると、アプリケーションの実装が容易になるだけでなく、アプリケーションがこのプラットフォームのインターフェイスガイドラインに準拠して動作することを保証できます。もちろん、View Controllerが提供する基本機能だけでは十分ではありません。どこかの時点で、カスタムコンテンツを表示するためにカスタムコードが処理を引き継ぐ必要があります。この文書では、View Controllerが提供する動作と、固有のニーズに合わせてこれらの動作をカスタマイズできる場所について説明します。

重要： この文書の内容は、iPhoneアプリケーションにのみ適用されます。Mac OS Xアプリケーションでもある種のView Controllerをサポートしますが、これらは異なる要件と動作を持つ別のクラスを使用して実現されています。

お読みになる前に

この文書をお読みになる前に、少なくとも以下のCocoaの概念について基本的な理解を得ている必要があります。

- XcodeおよびInterface Builderについての基本情報と、アプリケーション開発におけるそれらの役割
- 新規Objective-Cクラスの定義の仕方
- メモリ管理の方法（Objective-Cでオブジェクトを作成したり解放する方法を含む）
- アプリケーションの動作を管理する上でのデリゲートオブジェクトの役割
- Model-View-Controllerパラダイムについての基本的な理解

CocoaおよびObjective-Cについて経験の少ないデベロッパの方は、これらのトピックについての情報を『*Cocoa Fundamentals Guide*』で入手できます。

iPhoneアプリケーションの開発には、Mac OS X v10.5以降を実行するIntelベースのMacintoshコンピュータが必要です。また、iPhone SDKをダウンロードし、インストールする必要があります。iPhone SDKの入手方法については、http://developer.apple.com/iphone/にアクセスしてください。

この書類の構成

この文書は、次の章で構成されています。

- 「View Controllerの概要」（13 ページ）では、View Controllerの概要と、View ControllerがiPhoneアプリケーションで果たす役割について説明します。
- 「カスタムView Controller」（23 ページ）では、すべてのView Controllerがサポートする基本的な動作と、アプリケーションで使用するカスタムView Controllerの作成方法について説明します。
- 「Navigation Controller」（57 ページ）では、Navitation Controllerを作成して設定する方法について説明します。
- 「Tab Bar Controller」（79 ページ）では、Tab Bar Controllerを作成して設定する方法について説明します。
- 「モーダルView Controller」（93 ページ）では、View Controllerをモーダルモードで表示する方法について説明し、その際のガイドラインを示します。
- 「View Controllerインターフェイスの組み合わせ」（101 ページ）では、異なる種類のView Controllerを組み合わせて、高度なユーザインターフェイスを作成する方法について説明します。

フィードバックの提供

ドキュメントに関するフィードバックは、各ページの一番下にある組み込みのフィードバックフォームを使って送ってください。

Appleのソフトウェアやドキュメントのバグを発見した場合は、Appleにお知らせください。また、製品やドキュメントの今後の改訂版に期待する機能についての機能拡張リクエストを提出することもできます。バグ報告や機能拡張リクエストを提出するには、以下のADC WebサイトのBug Reportingページを使用します。

http://developer.apple.com/jp/bugreporter/

バグ報告を行うには、ADCの有効なログイン名とパスワードが必要です。ログイン名は、Bug Reportingページに説明されている手順に従って無料で入手できます。

関連項目

アプリケーションの設計に関する追加情報については、『*iPhone Application Programming Guide*』および『*iPhone Human Interface Guidelines*』を参照してください。

この文書で取り上げたView Controllerクラスの詳細については、『*UIKit Framework Reference*』を参照してください。

View Controllerの概要

View Controllerは、iPhoneアプリケーションの実装に必要な基本的なインフラストラクチャを提供します。この章では、View Controllerがアプリケーションで果たす役割と、View Controllerを使用してさまざまな種類のユーザインターフェイスを実装する方法の概要を説明します。

View Controllerとは

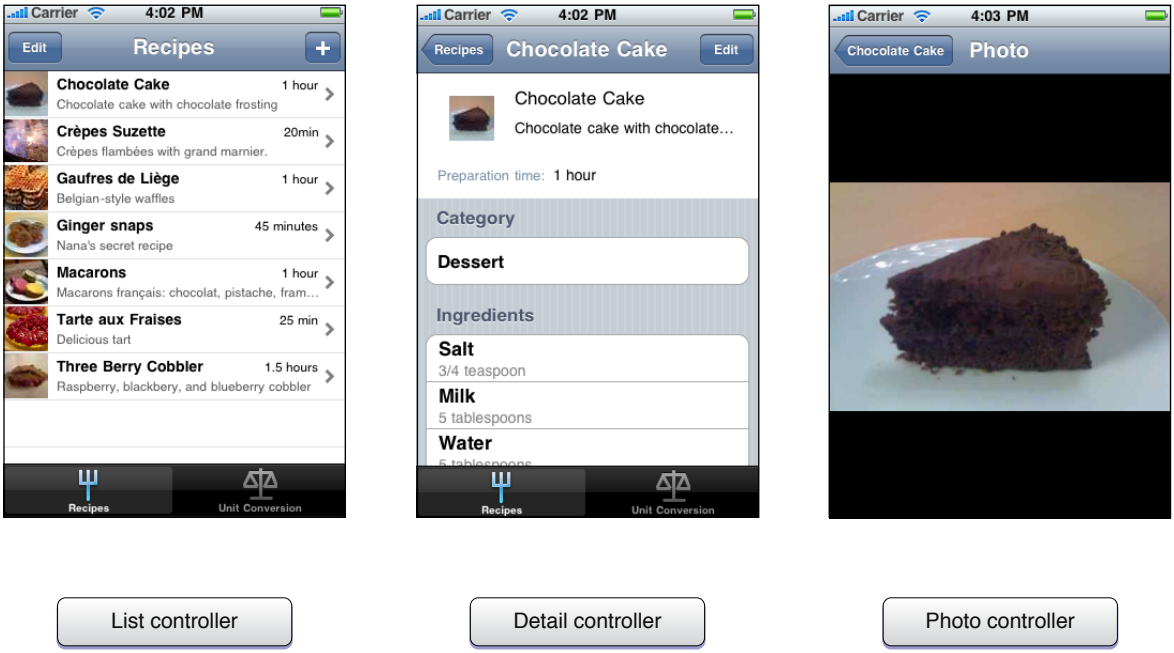
Model-View-Controller (MVC)デザインパターンでは、Controllerオブジェクトは、アプリケーションのデータと、そのデータをユーザに表示するために使うビューやその他の視覚的な要素との橋渡しに必要なカスタムロジックを提供します。iPhoneアプリケーションでは、**View Controller**は、アプリケーションのビューを表示したり管理するために使う特殊なControllerオブジェクトです。View Controllerオブジェクトは、UIKitフレームワークに定義されている`UIViewController`クラスの下位オブジェクトです。

View Controllerは、iPhoneアプリケーションの設計および実装において非常に重要な役割を果たします。iPhone OSベースのデバイスで動作するアプリケーションがコンテンツを表示するために使用できる画面領域は限られているため、ユーザに情報を提示する方法を工夫する必要があります。たくさんのコンテンツを持つアプリケーションでは、一般に、そのコンテンツを複数の画面に分散して、それぞれの画面を適切なタイミングで表示または非表示にします。View Controllerオブジェクトは、表示中のビューを管理したり、ビューの表示と非表示を調整するためのインフラストラクチャを提供します。

アプリケーションでView Controllerを使用すべき理由はたくさんありますが、View Controllerを避けるべき理由はほとんどありません。View Controllerを使用すると、iPhoneアプリケーションで使われる標準的なインターフェイス動作のほとんどを簡単に実装できます。View Controllerは、そのまま使用できるデフォルトの動作を提供しています。また、必要に応じて、その動作をカスタマイズすることもできます。また、View Controllerは、アプリケーションのユーザインターフェイスやコンテンツを構成するための便利な方法も提供しています。

図 1-1に、レシピを管理するアプリケーションの3つの異なる（ただし、関連する）画面の例を示します。最初の画面には、このアプリケーションが管理するレシピのリストが表示されています。レシピの1つをタップすると次の画面が表示されて、そのレシピの詳細が表示されます。この詳細ビュー内のレシピの写真をタップすると、3番目の画面が表示されて、その料理の写真が画面全体に表示されます。これらの各画面を管理しているのは、別々のView Controllerオブジェクトです。それぞれが、対応するビューオブジェクトを表示したり、そのビューにデータを取り込んだり、ビューとのやり取りに応答する仕事を行います。

図 1-1 View Controllerが管理する3つの画面



ビューの表示と管理に加えて、View Controllerを使用して画面から画面へのナビゲーションを管理することもできます。iPhone OSでは、新規の画面を表示するための標準的な方法がいくつかあります。これらすべての方法をView Controllerを使用して実装できます。それぞれについては、この文書の各所で説明します。

View Controllerの種類

ほとんどのiPhoneアプリケーションには少なくとも1つのView Controllerが含まれています。また、複数のView Controllerを持つアプリケーションもあります。大まかに言うと、View Controllerは、アプリケーションで果たす役割に応じて大きく3つのカテゴリに分類されます。

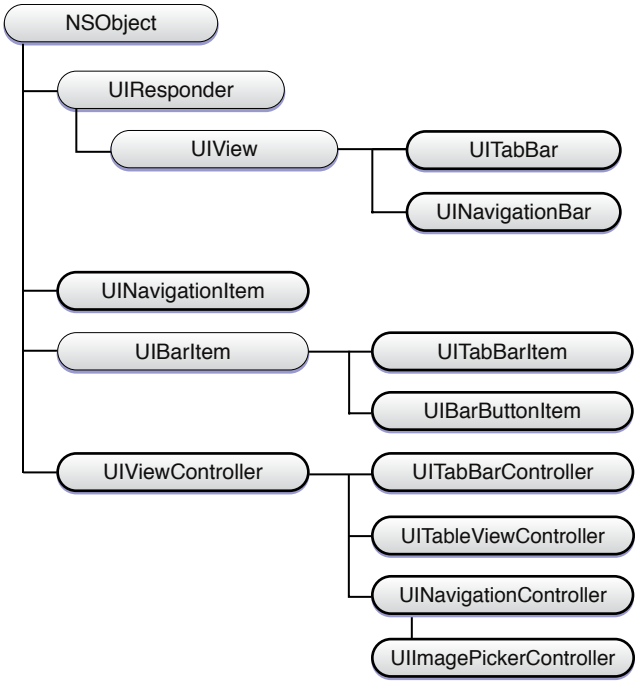
カスタムView Controllerは、何らかのコンテンツを画面に表示する目的で特別に定義するControllerオブジェクトです。ほとんどのiPhoneアプリケーションでは、表示の方法が異なる複数の画面を使用してデータを表示します。たとえば、ある画面はテーブル形式で項目のリストを表示し、別の画面はそのリスト内の1つの項目に関する詳細情報を表示します。このようなアプリケーションに対応するアーキテクチャでは、画面のタイプごとに、データのマーシャリングや表示を管理する別々のView Controllerを作成する必要があります。

コンテナView Controllerは、ほかのView Controllerを管理して、それらの間のナビゲーション関係を定義する特殊なView Controllerオブジェクトです。Navigation ControllerとTab Bar Controllerは、どちらもコンテナView Controllerの例です。通常は、コンテナView Controllerをデベロッパが定義することはありません。代わりに、システムが提供するコンテナView Controllerをそのまま使用します。これは、それぞれのコンテナView Controllerが、特定のタイプのインターフェイスの完全な実装を提供しているからです。

モーダルView Controllerは、別のView Controllerによって特定の方法で表示される任意のView Controller（コンテナかカスタムかは問わない）です。モーダルView Controllerは、アプリケーション内の特定のナビゲーション関係を定義します。View Controllerをモーダルモードで表示する最も一般的な目的は、ユーザに何らかのデータ入力を要求するためです。たとえば、ユーザにフォームへの入力をしてもらったり、ピッカーインターフェイスからオプションを選択してもらうために、View Controllerをモーダルモードで表示します。ただし、モーダルView Controllerにはその他の用途もあります。詳細は、「モーダルView Controller」（93 ページ）で説明します。

図 1-2に、UIKitフレームワークで利用できるView Controllerクラスと、View Controllerと組み合わせて使う主なクラスを示します。通常、これらの補助クラスは、特殊なインターフェイスを実装するためにView Controllerオブジェクトで内部的に使われます。たとえば、`UITabBarController`オブジェクトは、Tab Barインターフェイスに関連付けられたタブを実際に表示する`UITabBar`オブジェクトを管理します。その他のフレームワークでも、特定のタイプのインターフェイスを表示するために追加のView Controllerを定義していることがあります。

図 1-2　UIKitのView Controllerクラス



以降のセクションでは、アプリケーションのコンテンツを編成したり表示したりするために使用するいくつかの種類のView Controllerについてさらに詳しく説明します。

カスタムView Controllerの概要

カスタムView Controllerは、アプリケーションのコンテンツのために連携して動作する主要なオブジェクトです。ほとんどすべてのアプリケーションは、少なくとも1つのカスタムView Controllerを持っています。複雑なアプリケーションでは、複数のカスタムView Controllerを持つこともあります。カスタムView Controllerには、アプリケーションのデータとビューの間のやり取りを支援するために必要なロジックとグルーコードが含まれています。View Controllerは、アプリケーション内のほかのControllerオブジェクト（アプリケーションデリゲートやほかのView Controllerなど）とやり取りをすることもあります。

それぞれのカスタムView Controllerオブジェクトは、*ちょうど一画面に相当するコンテンツ*の管理を担当します。View Controllerと画面が1対1に対応していることは、アプリケーションの設計において非常に重要です。同じ画面の別々の部分を管理するために、複数のカスタムView Controllerを使用するべきではありません。同様に、複数の画面に相当するコンテンツを管理するために、単独のカスタムView Controllerを使用するべきではありません。

注： 1つの画面を複数の領域に分割して、各領域を別々に管理する場合は、View Controllerオブジェクトではなく、汎用のControllerオブジェクト（`NSObject`から派生させたカスタムオブジェクト）を使用して、その画面の各領域を管理します。そして、1つのView Controllerを使用して、それらの汎用Controllerオブジェクトを管理します。このView Controllerは画面全体のやり取りを調整しますが、必要に応じて、管理下にある汎用Controllerオブジェクトにメッセージを転送します。

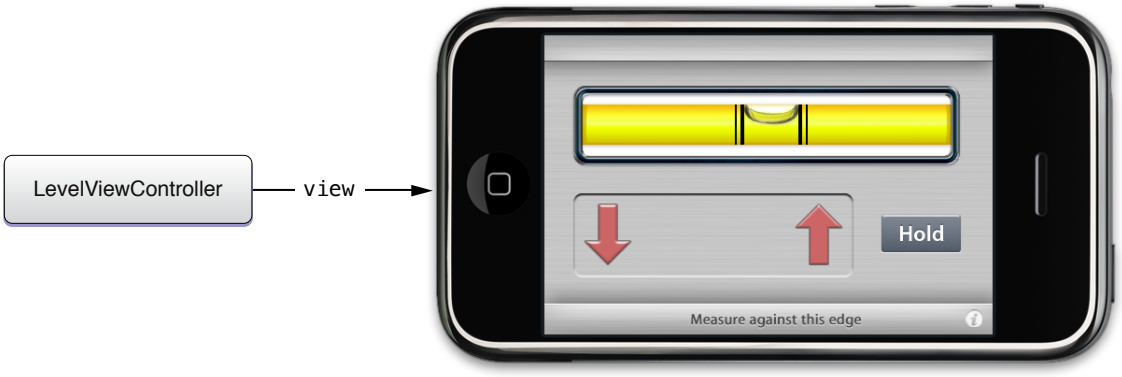
カスタムView Controllerを作成するには、`UIViewController`から直接サブクラスを作成して、そのサブクラスにカスタムコードを追加します。典型的な`UIViewController`のサブクラスの宣言には、次のような要素が含まれます。

- 対応するビューによって表示されるデータを含むオブジェクトを指すメンバ変数
- このView Controllerがやり取りしなければならない主要なビューオブジェクトを指すメンバ変数（アウトレット）
- ビュー階層内のボタンやその他のコントロールに関連付けられたタスクを実行するアクションメソッド
- このView Controllerのカスタム動作を実装するために必要な追加のメソッド

この種のView Controllerのほとんどのコードは、カスタムコンテンツを管理するために使用することから、アプリケーション固有のものになります。一方、すべてのView Controllerがサポートする共通の動作もあります。これらの共通の動作については、`UIViewController`クラスにメソッドが定義されています。デベロッパは、希望の動作を実装するためにこのメソッドをオーバーライドして使用できます。共通の動作には、ビューの管理、インターフェイスの向きの管理、メモリ不足の警告のサポートが含まれます。

図 1-3に、*BubbleLevel*サンプルプロジェクトのカスタムView Controllerの例を示します。このアプリケーションでは、`UIViewController`の直接のサブクラスである`LevelViewController`クラスを定義しています。このクラスは、デバイスの傾斜角度の変化を把握するために加速度センサーのデータを監視し、そのデータを使用して、対応するビューオブジェクトを更新します。このView Controllerの`view`プロパティは、コンテンツを表示している実際のビューオブジェクトの参照を提供します。

図 1-3 BubbleLevelアプリケーションのカスタムView Controller



すべてのView Controllerに必要な標準動作の管理については、「カスタムView Controller」（23 ページ）を参照してください。

Table View Controllerの概要

`UITableViewController`クラスは、テーブル形式のデータを管理するために特別に設計されたカスタムView Controllerです。Table View Controllerを使用しなくてもテーブルの管理はもちろん可能ですが、このクラスは、選択の管理、行の編集、テーブルの設定などの標準的なテーブル関連動作に対するサポート機能を自動的に追加します。この追加サポート機能によって、テーブルベースのインターフェイスを作成して初期化するために記述しなければならないコードの量を最小限に抑えることができます。Table View Controllerは、カスタムView Controllerを使用するのと同じ場所で使用できます。また、Table View Controllerのサブクラスを作成して、追加のカスタム動作を実装することもできます。もちろん、このようなView Controllerで管理されるビュー階層には、Table Viewオブジェクトが1つ含まれていなければなりません。

図 1-4に、Table View Controllerの構成例を示します。このTable View ControllerはカスタムView Controllerであるため、（`view`プロパティに）このインターフェイスのルートビューへのポインタを持っていますが、このインターフェイスに表示されるテーブルビューを指す別のポインタも持っています。

図 1-4 テーブル形式のデータの管理

この文書では、すべてのView Controllerに共通する動作についてのみ説明し、Table View Controllerに固有の情報については言及しません。Table View Controllerのテーブル関連動作に固有の情報については、『*UITableViewController Class Reference*』を参照してください。Table View管理の全般的な情報については、『*Table View Programming Guide for iPhone OS*』を参照してください。

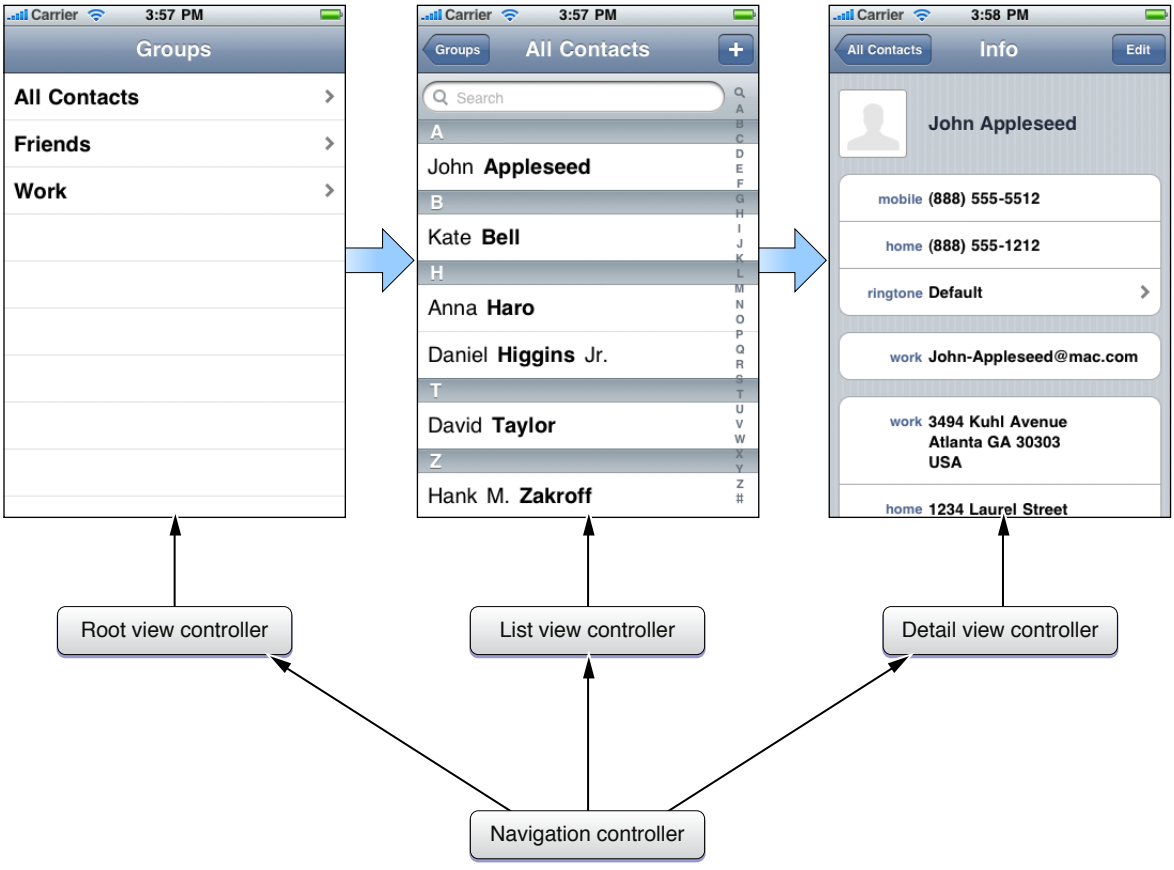
Navigation Controllerの概要

Navigation Controllerは、階層的に構成されたデータを表示するために使用するコンテナView Controllerです。Navigation Controller は`UINavigationController`クラスのインスタンスです。このクラスはそのまま使用できるため、サブクラスを作成する必要がありません。このクラスのメソッドは、カスタムView Controllerのコレクションをスタックベースで管理する機能をサポートしています。このスタックは、階層的なデータの中をユーザが通った経路を反映します。スタックの一番下は出発点を表し、スタックの一番上はデータ内のユーザの現在位置を表します。

Navigation Controllerの主な仕事は、ほかのView Controllerのマネージャとしての役割ですが、いくつかのビューの管理も行います。具体的に言うと、Navigation ControllerはNavigation Barを1つ管理します。Navigation Barには、データ階層内のユーザの現在位置についての情報、前の画面に戻るためのボタン、および現在のView Controllerで必要なカスタムコントロールが表示されます。また、Navigation Controllerは、現在の画面に関連するコマンドを表示するために使うオプションツールバーの管理も行います。ほとんどの場合、これらのビューを直接変更する必要はありませんが、`UIViewController`クラスに定義されているサポート機能を利用して、これらのビューを設定することができます。

図1-5に、「連絡先(Contacts)」アプリケーションのいくつかの画面を示します。このアプリケーションでは、Navigation Controllerを使用して連絡先情報をユーザに表示します。ユーザに対して表示される画面は、それぞれ1つのカスタムView Controllerオブジェクトによって管理されています。このカスタムView Controllerオブジェクトはデータ階層の特定のレベルにある情報を表示します。たとえば、「Root」View Controllerと「List」View Controllerは、テーブル形式の連絡先情報を異なる方法で表示します。「Detail」View Controllerは、まったく異なるタイプの画面を使用して、特定の連絡先の情報を表示します。ユーザがインターフェイス内のコントロールを操作すると、これらのコントロールは、Navigation Controllerに、次のView Controllerを表示したり、現在のView Controllerを閉じたりする指示を伝えます。

図 1-5　階層的なアプリケーションデータのナビゲーション



Navigation Controllerオブジェクトを設定したり使用したりする方法については、「Navigation Controller」（57 ページ）を参照してください。

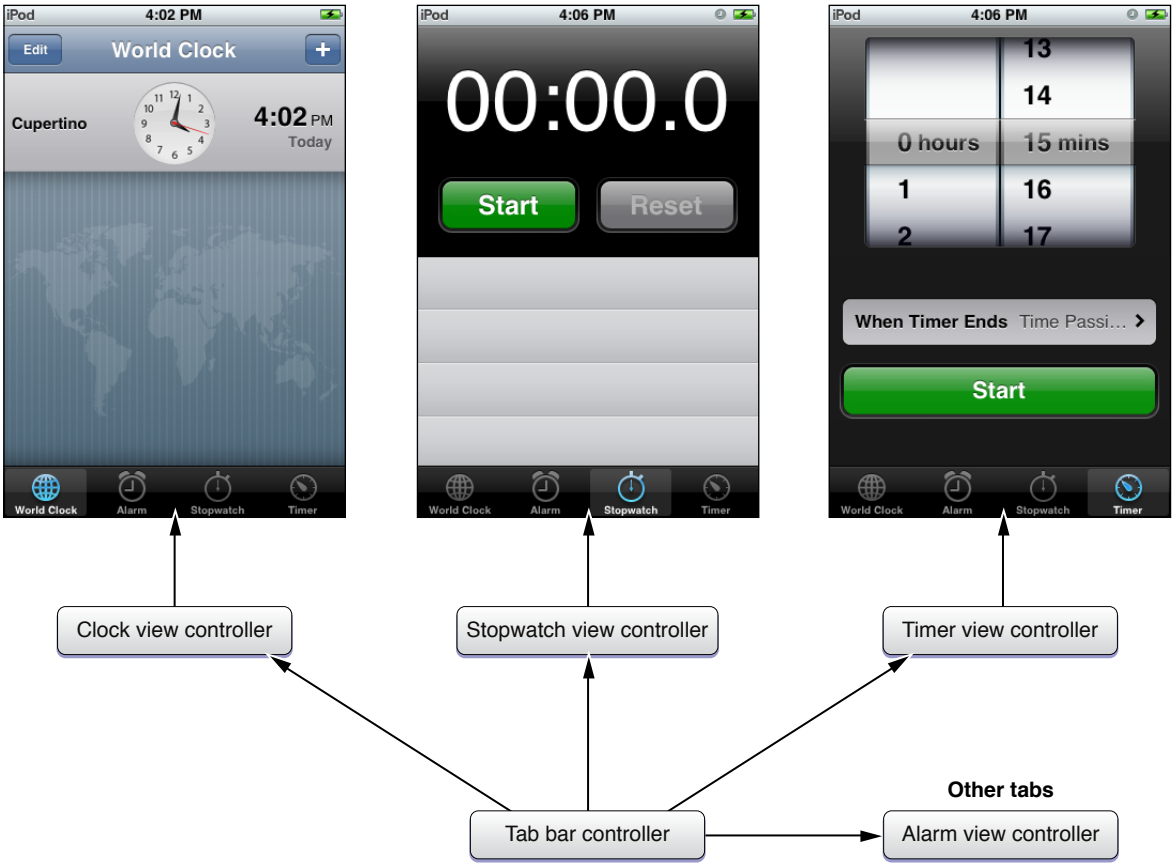
Tab Bar Controllerの概要

Tab Bar Controllerは、アプリケーションを2つ以上の操作モードに分割するために使用するコンテナView Controllerです。Tab Bar Controller は`UITabBarController`クラスのインスタンスです。このクラスはそのまま使用できるため、サブクラスを作成する必要がありません。Tab Bar Controllerのモードは Tab Bar Viewを使用して表示されます。Tab Bar Viewには、サポートするモードごとに1つのタブが表示されます。タブを選択すると、それに対応するView Controllerによって、そのインターフェイスが画面に表示されます。

アプリケーションで異なる種類のデータを表示するときや、同じデータでもまったく異なる方法で表示するときに、Tab Bar Controllerを使用します。Tab Bar Controllerには、Tab Bar Viewでのユーザのタップに応答して自動的にモードを切り替える機能があります。モード数が多くてタブ用のスペースに入りきらないときは、Tab Bar Controllerが、標準では表示しないタブの選択や、表示するタブのカスタマイズも管理します。

図 1-6に、「時計(Clock)」アプリケーションのいくつかのモードと、それに対応するView Controllerとの関係を示します。各モードには、メインコンテンツ領域を管理するルートView Controllerがあります。「時計(Clock)」アプリケーションの場合、「Clock」View Controllerと「Alarm」View Controllerは共に、画面の上端に追加コントロールを持つナビゲーションスタイルのインターフェイスを表示します。その他のモードでは、カスタムView Controllerを使用して、単独の画面を表示します。

図 1-6　「時計(Clock)」アプリケーションのさまざまなモード



Tab Bar Controllerを設定したり使用したりする方法については、「Tab Bar Controller」 （79 ページ）を参照してください。

モーダルView Controllerの概要

モーダルView Controllerとは、特定の種類のView Controllerクラスではなく、View Controllerをユーザへ表示する方法の1つです。コンテナView Controllerは、管理下にあるView Controller間の特定の関係を定義しますが、モーダルView Controllerを利用するとデベロッパがこの関係を定義できます。どんなView Controllerオブジェクトも、ほかのView Controllerオブジェクトをモーダルモードで表示できます。ほとんどの場合は、ユーザから情報を収集したり、特定の目的でユーザの注意を引くためにView Controllerをモーダルモードで表示します。その目的が達成されたら、モーダルView Controllerを閉じて、ユーザがアプリケーション内を移動できるようにします。

図 1-7に、「連絡先(Contacts)」アプリケーションの例を示します。ユーザが新規の連絡先を追加するために「＋」ボタンをクリックすると、「Contacts」View Controllerが「New Contact」View Controllerをモーダルモードで表示します。これによって、この2つのView Controller間に親子関係が生じます。ユーザがこの操作をキャンセルするか、連絡先データベースに保存するのに十分な連絡先情報を入力するまでは、「新規連絡先(New Contact)」画面が表示されたままになります。操作が完了した時点で、「Contacts」View Controllerは子ビューを閉じます。

図 1-7 モーダルView Controllerの表示

モーダルモードで表示されているView Controller自身が、別のView Controllerをモーダルモードで表示できることも注目に値します。モーダルView Controllerを連鎖させることができる機能は、複数のモーダルアクションを順番に実行する必要があるときに役立ちます。たとえば、前の図の「新規連絡先(New Contact)」画面で、ユーザが「写真を追加(Add Photo)」ボタンをタップして既存の画像を選ぶ場合、「New Contact」View Controllerは画像ピッカーインターフェイスをモーダルモードで表示します。連絡先のリストに戻るには、この画像ピッカー画面を閉じてから、「新規連絡先(New Contact)」画面を個別に閉じなければなりません。

モーダルView Controllerの用途とアプリケーションでの表示方法の詳細については、「モーダルView Controller」（93 ページ）を参照してください。

XcodeのiPhoneアプリケーションテンプレート入門

iPhoneアプリケーション用のXcodeプロジェクトテンプレートのほとんどは、少なくとも1つのView Controllerクラスを最初から提供しています。また、複数のView Controllerを提供するテンプレートもあります。これらの初期クラスは、典型的なView Controllerに必要なコードを提供しており、アプリケーションをすぐに書き始められるように設計されています。しかし、テンプレートは単なる出発点に過ぎないということを忘れてはいけません。

テンプレートアプリケーションの目的は、特定の種類のアプリケーションに着手するために最適な方法を示すことです。これから作成しようとするインターフェイスに最も近いテンプレートを利用して開発を始めるのが、常に最も簡単です。たとえば、「株価(Stocks)」アプリケーションや「天気(Weather)」アプリケーションに似たアプリケーションを作成するのであれば、Utility Applicationテンプレートを使用します。一方、Tab Barを使用してアプリケーションを異なるモードに分けるのであれば、Tab Bar Applicationテンプレートを使用します。

View Controllerの基本的な動作を試してみたい場合は、View-based Applicationテンプレートから始めるのがよいでしょう。このタイプのアプリケーションでは、1つのカスタムView Controllerを使用して、アプリケーションのコンテンツを表示します。追加のView Controllerをモーダルモードで表示すれば、この基本動作を拡張できます。

アプリケーションのユーザインターフェイスをゼロから作成したければ、Window-based Applicationテンプレートを使用します。このテンプレートは、最小構成のプロジェクトを提供します。デベロッパは、必要なView Controllerを含めるようにこのプロジェクトに変更を加えることができます。

Xcodeでのプロジェクト作成の詳細については、『*Xcode Project Management Guide*』を参照してください。

カスタムView Controller

カスタムView Controllerは、アプリケーションのコンテンツを表示するために使用します。どのView Controllerも、何らかのコンテンツの表示を管理し、アプリケーションの基盤となるデータオブジェクトに基づいてコンテンツを更新したり同期させたりする仕事を行います。カスタムView Controllerでは、そのために、コンテンツを表示するビューを作成して、そのビューのコンテンツとアプリケーションのデータ構造を同期させるために必要なインフラストラクチャを実装しなければなりません。

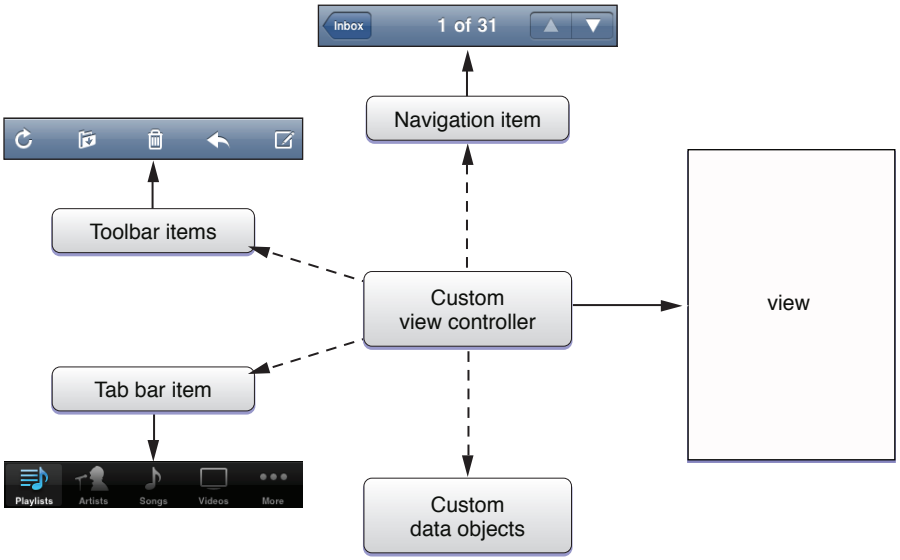
`UIViewController`クラスは、（カスタムかそうでないかを問わず）すべてのView Controllerに必要な基本的な動作を提供します。この章では、このクラスが提供する基本的な動作について説明し、アプリケーションのニーズに合わせてこれらの動作を変更する方法も示します。これらの動作とそれを変更する方法を理解することは、アプリケーションのカスタムView Controllerを実装するために不可欠です。また、どの種類のView Controllerとやり取りをする場合にも役立ちます。

カスタムView Controllerの構造

`UIViewController`クラスは、すべてのカスタムView Controllerを実装するための基本的なインフラストラクチャを提供します。`UIViewController`クラスのインスタンスを作成してビューを表示することもできますが、何か特別なことを実行したい場合は、カスタムサブクラスを定義する必要があります。そのサブクラスで、カスタムメソッドを使用して、ビューにデータを取り込んだり、ボタンやその他のコントロールに対するタップに応答します。ただし、View Controllerのデフォルトの動作に変更を加えたいときは、`UIViewController`クラスのメソッドをオーバーライドする必要があります。また、希望の動作を実装するために、ほかのUIKitクラスとのやり取りが必要になる場合もあります。

図 2-1に、カスタムView Controllerに直接関連付けられる主要なオブジェクトを示します。これらのオブジェクトは、実質的にはView Controller自身が所有し管理します。ビュー（`view`プロパティを介してアクセス可能）は、View Controllerに提供しなければならない唯一のブジェクトです。ただし、ほとんどのView Controllerは、表示する必要があるデータを含むカスタムオブジェクトも持っています。その他のオブジェクトは、Navigation Barインターフェイスや Tab Barインターフェイスなどの機能をサポートする必要があるときにだけ使います。それでも、ほとんどのオブジェクトは、十分なデフォルトの動作を提供しています。

図 2-1 カスタムView Controllerの構造



View Contorllerが単独で動作することはめったにありませんが、カスタムView Controllerは、常に独立したオブジェクトとして設計するべきです。つまり、View Controllerには、1つの画面に相当する分のコンテンツの管理に関わるすべての動作をカプセル化する必要があります。View Controllerには、必要なデータ（または、少なくとも管理下におくデータへの参照）、そのデータの表示に必要なビュー、システム内の変化（向きの変化など）に対応するためのロジック、およびユーザとのやり取りを検証したり処理したりするために必要なコードが含まれていなければなりません。ビュー階層やデータモデルの一部を管理するために必要なすべてのカスタムオブジェクトは、View Controllerで作成し、完全に管理しなければなりません。

カスタムView Controllerの実装チェックリスト

アプリケーションにカスタムView Controllerを実装するには、Xcodeを使用してソースファイルを準備します。ほとんどのiPhoneプロジェクトテンプレートには、少なくとも1つのView Controllerクラスが含まれています。また、必要であれば、Xcodeで新規のView Controllerを作成できます。

独自に作成するカスタムView Controllerで、必ず実行しなければならない作業がいくつかあります。

- View Controllerでロードするビューを設定する（「View Controller用のビューの作成」 （29 ページ）を参照）。
- View Controllerでサポートする向きを決定する（「インターフェイスの向きの管理」 （38 ページ）を参照）。
- View Controllerで管理するメモリをクリーンアップする（「効率的なメモリ管理」 （46 ページ））。

View Controllerのビューを設定するときには、アクションメソッドや、そのビューで使用するアウトレットを定義する必要があるかもしれません。たとえば、ビュー階層にテーブルが含まれている場合は、そのテーブルへのポインタをアウトレットに格納して、後でそれにアクセスできるようにします。同様に、ビュー階層にボタンやその他のコントロールが含まれている場合は、ユーザとのや

り取りに応答して、それらのコントロールから対応するアクションメソッドを呼び出すようにします。View Controllerクラスの定義の繰り返しになりますが、次のような項目をView Controllerクラスに追加する必要があることに気付いたかもしれません。

- 対応するビューによって表示されるデータを含むオブジェクトを指すメンバ変数
- このView Controllerがやり取りしなければならない主要なビューオブジェクトを指すメンバ変数（アウトレット）
- ビュー階層内のボタンやその他のコントロールに関連付けられたタスクを実行するアクションメソッド
- このView Controllerのカスタム動作を実装するために必要な追加のメソッド

重要： アウトレットやその他のメンバ変数をカスタムView Controllerに追加するときは、アクセスしようと考えているそれぞれの変数用の宣言済みプロパティを含めることを強くお勧めします。宣言済みプロパティは、メンバ変数にアクセスするための簡易構文を提供します。また、宣言済みプロパティによって、これらの変数を管理するために必要なグルーコードの大半を書く必要がなくなります。特に、ほかのオブジェクトを参照する場合は、必要に応じてオブジェクトの保持と解放を自動的に行ってくれるため、プロパティを使用すると非常に便利です。

上記の項目は、独自に作成するすべてのカスタムView Controllerクラスに含めることになる可能性が最も高い項目です。しかし、特定の動作を実装するために、その他のメソッドをView Controllerに追加する場合もあります。これらのメソッドの多くは、View Controllerのインフラストラクチャのフックを利用して、次のような一般的な処理を実装します。

- View Controllerのビューが画面に表示されたり、画面から消去されるのに応答して、ビュー階層やアプリケーション状態を調整できます（「表示関連の通知への応答」（49 ページ）を参照）。
- Navigation ControllerやTab Bar Controllerで使われる次のようなオブジェクトを設定できます。
  - Navigation Item（View ControllerをNavigation Controllerインターフェイスと組み合わせて使用する場合）（「Navigation Barの外観のカスタマイズ」（70 ページ）を参照）。
  - Toolbar Item（関連するNavigation Controllerがツールバーを表示する場合）（「ツールバー項目の指定」（77 ページ）を参照）。
  - Tab Bar Item（View ControllerをTab Bar Controllerインターフェイスと組み合わせて使用する場合）（「Tab Barインターフェイスの作成」（82 ページ）を参照）。
- インターフェイスの向きが変化した場合は、ビュー階層を調整できます（「向きの変化への応答」（40 ページ）を参照）。
- ビューまたはそのサブビューで処理されないイベントをキャッチするイベントハンドラを実装できます（『*iPhone Application Programming Guide*』の「Event Handling」を参照）。
- 編集可能なビューを実装できます（「ビューの編集モードの有効化」（52 ページ）を参照）。

ビュー管理サイクルの理解

View Controllerオブジェクトでは、対応するビューの管理は、ロードサイクルとアンロードサイクルという2つの独立したサイクルとして発生します。ロードサイクルは、アプリケーションがView Controllerにビューオブジェクトへのポインタを要求したときに、そのオブジェクトが現在メモリ内に存在しない場合に必ず発生します。ロードサイクルが発生すると、View Controllerは、そのビューをメモリにロードし、今後参照できるようにそのビューへのポインタを保存します。

その後、アプリケーションがメモリ不足の警告を受け取ると、View Controllerはそのビューをアンロードしようとします。アンロードサイクルでは、View Controllerはビューオブジェクトを解放して、ビューのない初期状態に戻ることを試みます。ビューを解放すると、View Controllerは再びビューが要求されてロードサイクルが再度始まるまで、ビューオブジェクトがない状態のままになります。

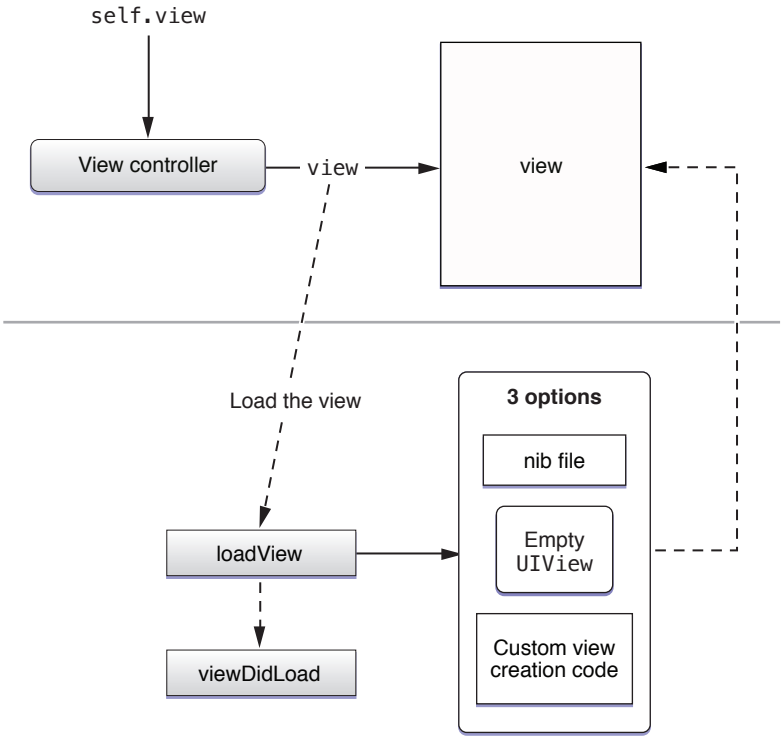
ロードサイクルとアンロードサイクルでは、View Controllerがビューのロードやアンロードに関わるほとんどの処理を実行します。ただし、View Controllerクラスで、ビュー階層内のビューへの参照を格納したり、ロード時にビューにほかの設定を行う必要がある場合は、特定のメソッド（後述）をオーバーライドしてその処理を実行できます。

ロードサイクルでは、次のようなステップが発生します。

1. アプリケーションのどこかで、View Controllerの`view`プロパティ内にあるビューが要求されます。
2. ビューが現在メモリ内に存在しない場合、View Controllerは`loadView`メソッドを呼び出します。
3. `loadView`メソッドは次のいずれかを実行します。
    - このメソッドをオーバーライドした場合には、その実装が、必要なビューをすべて作成して、`view`プロパティに`nil`以外の値を代入する処理を担います。
    - このメソッドをオーバーライドしなかった場合、デフォルトの実装が、View Controllerの`nibName`プロパティと`nibBundle`プロパティを使用して、指定のnibファイルからビューをロードしようとします。指定のnibファイルが見つからない場合は、View Controllerクラスの名前と同じ名前のnibファイルを検索して、そのファイルをロードします。
    - 該当するnibファイルが見つからなければ、メソッドは空の`UIView`オブジェクトを作成して、それを`view`プロパティに代入します。
4. View Controllerは`viewDidLoad`メソッドを呼び出して、ロード時に実行すべきほかの処理があればそれらを実行します。

図 2-2は、ロードサイクルを図示したものです。この図には、ロードサイクルで呼び出されるいくつかのメソッドも含まれています。必要であれば、アプリケーションで`loadView`メソッドと`viewDidLoad`メソッドの両方をオーバーライドして、View Controllerに希望の動作を追加することもできます。

図 2-2 ビューをメモリにロードする

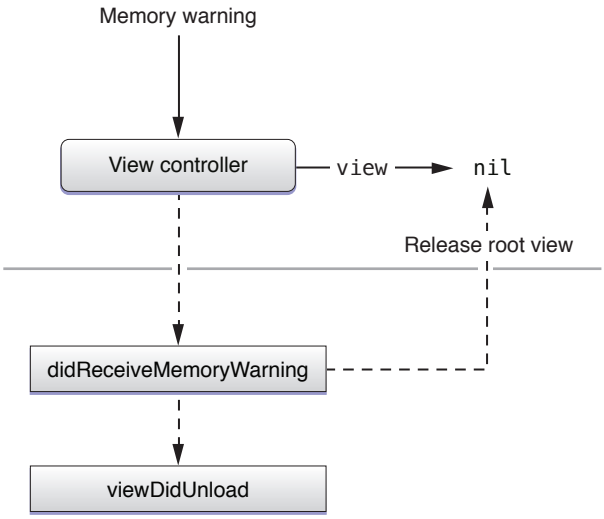


アンロードサイクルでは、次のようなステップが発生します。

1. アプリケーションがシステムからメモリ不足の警告を受け取ります。
2. 各View Controllerは自身の`didReceiveMemoryWarning`メソッドを呼び出します。
    - このメソッドをオーバーライドする場合は、その中で、View Controllerオブジェクトで不要になったすべてのカスタムデータを解放します。このメソッドでは、View Controllerのビューを解放すべきではありません。このメソッドの実装のどこかで`super`を呼び出して、デフォルトの動作を実行しなければなりません。
    - デフォルトの実装では、ビューを解放しても安全であると判断されたときにのみ、ビューが解放されます。
3. View Controllerは、ビューを解放すると、`viewDidUnload`メソッドを呼び出します。このメソッドをオーバーライドして、ビューやビュー階層に必要な追加のクリーンアップ処理を実行することもできます。

図 2-3は、View Controllerのアンロードサイクルを図示したものです。

図 2-3 ビューをメモリからのアンロードする



重要： iPhone OS 3.0以降では、ビューのクリーンアップに関連するコードを置く場所としては、`viewDidUnload`メソッドが望ましい場所です。`didReceiveMemoryWarning`メソッドをオーバーライドして、ビューの解放時に一時的なキャッシュや不要になったその他のプライベートデータを解放することもできます。`didReceiveMemoryWarning`をオーバーライドする場合は、必ず`super`を呼び出して、親のバージョンのメソッドにビューを解放させます。

iPhone OS 2.2以前では、ビュー関連のクリーンアップを実行したり、不要になったプライベートデータ構造を解放するために、`didReceiveMemoryWarning`メソッドを使用しなければなりません。`viewDidUnload`メソッドは、iPhone OS 3.0以降でのみ利用できます。

メモリ不足状態のときのメモリ管理の詳細については、「効率的なメモリ管理」（46 ページ）を参照してください。

カスタムView Controllerクラスの定義

カスタムView Controllerは`UIViewController`のサブクラスで、アプリケーションのコンテンツを表示するために使用します。ほとんどのXcodeプロジェクトテンプレートには、必要に応じて変更可能なカスタムView Controllerクラスが1つ含まれています。カスタムView Controllerを追加作成する必要がある場合は、次の手順を実行します。

1. 「ファイル(File)」＞「新規ファイル(New File)」を選び、新規のソースファイルをプロジェクトに追加します。

   新規の`UIViewController`サブクラスを作成します。「新規ファイル(New File)」ダイアログの「Cocoa Touch Classes」セクションに、このタイプのクラス用のテンプレートがあります。

2. この新しいView Controllerファイルに適切な名前を付けて、プロジェクトに追加します。

3. ソースファイルを保存します。

View Controllerのソースファイルを作成したら、コンテンツを表示するために必要な動作を実装できます。以降の各セクションでは、カスタムView Controllerを使用して実行できる主な処理について説明します。View Controllerの作成の詳細については、『*UIViewController Class Reference*』を参照してください。

View Controller用のビューの作成

View Controllerの主な仕事は、必要に応じてビューをロードしたりアンロードしたりすることです。ほとんどのView Controllerは、それに対応するnibファイルからビューをロードします。nibファイルを使用する利点は、ビューを視覚的に配置したり設定したりできるため、レイアウトの調整がすばやく簡単にできることです。しかし、プログラムによってビューを作成することもできます。

Interface Builderでのビューの作成

Interface Builderは、View Controller用のビューを作成したり設定したりするための直感的な方法を提供します。Interface Builderは、その名のとおり、アプリケーションのインターフェイスをプログラムからではなく視覚的に作成するツールです。このアプリケーションを使用して、ビューとコントロールを直接操作してワークスペースにドラッグし、位置を決めたり、サイズを変更したり、インスペクタウインドウを使用してそれらの属性を変更したりすることによって、ビューとコントロールを組み立てることができます。そして、その結果はnibファイルに保存されます。nibファイルには、デベロッパが組み立てたオブジェクトの集合が、それに対するすべてのカスタマイズ情報と一緒に保存されます。

View Controllerで使用するnibファイルを構成するには、次の2つの方法があります。

- ビューだけをnibファイルに保存して、**分離nibファイル**を作成する。
- ビューとView Controllerを同じnibファイルに保存して、**統合nibファイル**を作成する。

この2つの方法のなかでは、分離nibファイルを使用する方法が圧倒的に望ましい方法です。分離nibファイルは、特にメモリ管理に関しては、よりしっかりしたソリューションを提供します。メモリ不足状態のときは、ビューを所有するView Controllerオブジェクトの状態に影響を与えずに、必要に応じて、分離nibファイルの内容をメモリから削除できます。統合nibファイルの場合は、これと同じことはできません。統合nibファイルの内容は、すべてのnibファイルオブジェクトが不要になるまで、メモリ内に存在しなければなりません。

分離nibファイルにビューを保存する

分離nibファイルを作成する手順には、次の2つの別個の実装ステップが必要になります。

- ビューを含むnibファイルを設定しなければなりません。
- そのnibファイルとView Controllerオブジェクトを関連付けなければなりません。

nibファイルそのものの設定は、比較的簡単です。ゼロからnibファイルを作成する場合は、Interface BuilderのCocoa Touch Viewテンプレートを出発点として使用するべきです。このテンプレートで作成したnibファイルには、File's Ownerプレースホルダと1つのカスタムビューオブジェクトが含まれています。この新規のnibファイルをXcodeプロジェクトに追加し、次の手順に従ってその内容を構成します。

1. File's Ownerプレースホルダのクラス名にView Controllerクラスの名前を設定します。

このView Controllerクラスは、Xcodeプロジェクトであらかじめ作成しておく必要があります。新規のnibファイルの場合は、File's Ownerのクラスはデフォルトで`NSObject`に設定されています（Xcodeプロジェクトテンプレートによって提供されたnibファイルを編集している場合は、このクラスが、すでに適切なView Controllreクラス名に設定されている場合もあります）。

2. File's Ownerプレースホルダの`view`アウトレットが、このnibファイル内の最上位のViewオブジェクトに接続されていることを確認します。

   この`view`アウトレットは、`UIViewController`クラスで定義されており、すべてのView Controllerオブジェクトに継承されます。このアウトレットがFile's Ownerプレースホルダに見つからない場合は、このnibファイルをXcodeプロジェクトに追加してあるかどうかを確認してください。nibファイルとXcodeプロジェクトを関連付けることによって、Interface Builderはそのプロジェクト内のクラスに関する情報を自動的に取得します。この操作は、これらのクラスの利用可能なアウトレットとアクションを知るために必要です。

   このアウトレットを接続し忘れると、nibファイルがロードされたときに、View Controllerの`view`プロパティが`nil`に設定されます。その結果、ビューが画面に表示されません。

3. ビューそのものを作成して、アプリケーションのコンテンツを表示するために必要なすべてのサブビューを追加します。

4. nibファイルを保存します。

nibファイルを作成して、それをXcodeプロジェクトに追加したら、View Controllerオブジェクトをそのnibファイルの名前で初期化する必要があります。View Controllerオブジェクトを初期化する方法は、オブジェクトの作成方法によって異なります。View Controllerをプログラムによって作成する場合は、View Controllerオブジェクトを初期化するときに、`initWithNibName:bundle:`メソッドにnibファイルの名前を渡します。（ビューを含むnibファイルではない）独立のnibファイルからView Controllerオブジェクトをロードする場合は、Interface Builderを使用して、View ControllerオブジェクトのNIB Name属性の値をビューのnibファイルの名前に設定します。

リスト 2-1に、プログラムによってView Cotrollerを作成して初期化する方法の例を示します。この例では、カスタムクラスが同じ名前のnibファイルを使用して、ビューを保存します。View Controllerを初期化したら、この例で実行しているように、適切な方法でView Controllerを使用します（View Controllerをユーザに表示するなど）。

リスト 2-1　プログラムによるView Controllerオブジェクトの作成

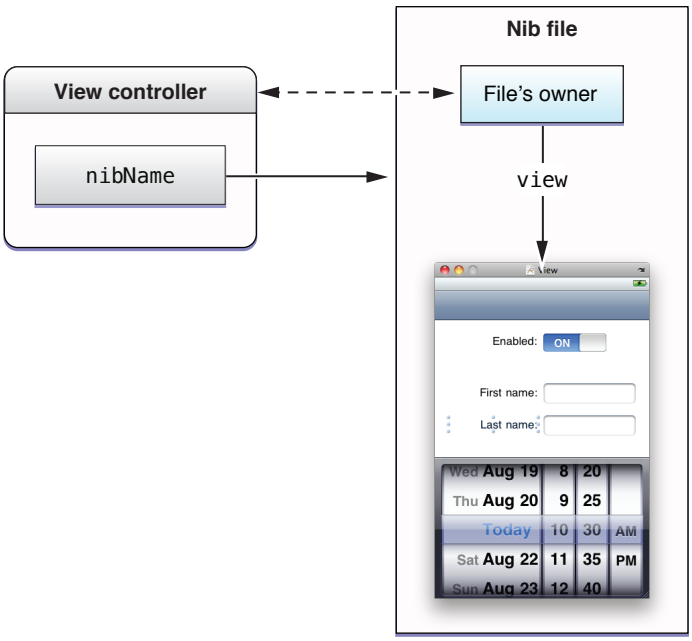
```
- (void)displayModalView
{
   MyViewController* vc = [[MyViewController alloc]
initWithNibName:@"MyViewController"
                                bundle:nil];
   [self presentModalViewController:vc animated:YES];
}
```

分離nibファイルでは、View Controllerオブジェクトの`view`プロパティにアクセスがあったときに、そのビューがメモリ内に現在存在しなければ、自動的にnibファイルのロードが始まります。デフォルトの`loadView`メソッドでは、`nibName`プロパティと`nibBundle`プロパティを使用して指定のnibファイルを見つけて、その内容をメモリにロードします。

図 2-4に、View Controllerの実行時の構成と、ロード前の分離nibファイルを示します。View Controllerの`nibName`プロパティには、nibファイルの名前を表す文字列が格納されています。この文字列を使用して、アプリケーションのバンドル内のnibファイルを見つけます。nibファイル内では、File's

Ownerプレースホルダが View Controllerオブジェクトの代わりに、View Controllerのアウトレットやアクションをnibファイル内のオブジェクトに接続するために使われます。nibファイルがロードされると、View Controllerオブジェクトの`view`プロパティは、nibファイルからのビューを指します。

図 2-4　分離nibファイルからビューをロードする



nibファイルを作成したり、その内容を設定したりする方法の詳細については、『*Interface Builder User Guide*』を参照してください。View Controllerのカスタムアウトレットとアクションの設定の詳細については、「View Controllerのアクションとアウトレットを設定する」（35 ページ）を参照してください。

ビューとView Controllerを同じnibファイルに保存する

アプリケーションが1つの画面しか持たない場合は、その画面のビューとそれを管理するView Controllerの両方を同じnibファイルに保存できます。ビューオブジェクトとカスタムView Controllerオブジェクトを同じnibファイルに保存することは、通常はお勧めしません。メモリ不足状態になったときに、システムがビューをアンロードできなくなることがしばしばあるからです。しかし、ビューそのものをアンロードすることがない場合は、View Controllerオブジェクトと同じnibファイルにビューを含める方が理にかなっている場合もあります。

図 2-5に、ウインドウ内に画面を1つだけ表示するアプリケーションのメインnibファイルを示します。この例では、nibファイルに、カスタムView Controllerオブジェクト(`MyViewController`)とそのView Controllerが管理するビューの両方が含まれています。このドキュメントウインドウでは、View Controllerの内部にビューオブジェクトがネストしている点に注目してください。このようにビューをネストさせる方法は好んで使われます。これによってInterface BuilderがビューとView Controllerの同期を保つことができるからです。

図 2-5 View Controllerをnibファイルに含める

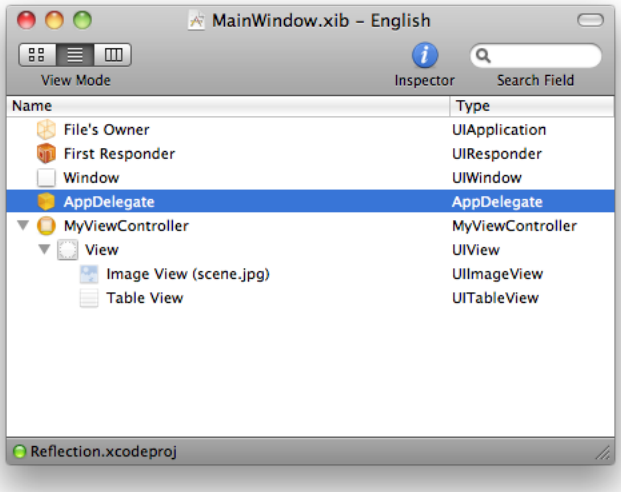


図 2-5に示したnibファイルを設定するには、次の手順を実行します。

1. View Controller (`UIViewController`)オブジェクトをライブラリからInterface Builderのドキュメントウインドウにドラッグします。
2. 次のいずれかの方法で、汎用のViewオブジェクトをView Controllerに追加します。
   - ビューをView Controllerのワークスペースウインドウにドラッグします。
   - ビューをInterface Builderのドキュメントウインドウ内のView Controllerオブジェクトにドラッグします。

   重要： （図 2-5 （32 ページ）に示したように）View Controller内にビューをネストさせるには、必ずビューをView Controllerオブジェクトにドラッグします。Interface Builderでは、View Controllerオブジェクトがステータスバーの存在、半透明効果、および親ウインドウ内のビューの位置に影響を与えるその他の要素を考慮します。ビューをView Controllerオブジェクトにネストさせないと、これらの要素は考慮されないため、ビューが不適切な位置に表示される可能性があります。
3. Image Viewをライブラリからこの汎用ビューにドラッグします。
4. Table Viewをライブラリからこの汎用ビューにドラッグします。
5. nibファイルを保存します。

注： 先の例では、View Controllerのルートビューとして汎用の`UIView`オブジェクトを使用しましたが、任意の1つのビュー（独自に定義した`UIView`のカスタムサブクラスなど）を追加することができます。独自のサブクラスを使用するには、汎用の`UIView`オブジェクトを追加し、「Identity」インスペクタを使用してそのクラスの名前を変更します。

nibファイルの最上位にオブジェクトを追加するときは、必ずこれらのオブジェクトをnibファイル内のどこかに存在するアウトレットに接続しなければなりません。前述のnibファイルの場合は、アプリケーションデリゲートオブジェクトでアウトレットを定義して、そのアウトレットをカスタムView Controllerオブジェクトに接続しています。アウトレットがないと、実行時にコードがView Controllerにアクセスする手段がありません。また、最上位レベルのオブジェクトは保持された後に自動解放されるため、そのオブジェクトを保持しておかないと、それを使用する前に解放されてしまう可能性があります。

View Controllerとそのビューが同じnibファイルに保存されているため、このnibファイルがメモリにロードされると、これら両方が使用できる状態になります。メインnibファイルの場合、通常は、View Controllerのビューをウインドウに追加するために、アプリケーションデリゲートの`applicationDidFinishLaunching:`メソッドに何らかのコードを追加します。これについては、「View Controllerのビューの表示」（48 ページ）で説明します。

Interface Builderでビューの表示属性を設定する

ビューのコンテンツを適切にレイアウトできるように、Interface Builderが提供するコントロールの中には、ビューがNavigation Bar、ToolBar、その他、カスタムコンテンツの位置に影響を与えるオブジェクトを持つかどうかを指定できるものがあります。表 2-1は、設定可能なアイテムと、それらがView Controllerまたはビューに与える影響を一覧表にまとめたものです。分離nibファイルの場合は、ビューオブジェクトの属性を変更することによって、これらのアイテムを設定します。統合nibファイルの場合は、View Controllerオブジェクトの属性を変更することによって、これらのアイテムを設定します。

表 2-1 View Controller用に設定可能なアイテム

設定可能なアイテム	説明
ステータスバー	「Attributes」インスペクタでStatus Bar属性を変更することによって、ステータスバーを表示するかどうか、またアプリケーションに表示するステータスバーの種類を指定できます。この属性は、デザインのためだけに用意されており、これを使用すればビューとコントロールがステータスバー付きで表示されたときにどのように見えるか全体像がわかります。 Status Bar属性の値はnibファイルには保存されません。ステータスバーの実際のスタイル（およびステータスバーを表示するかどうか）は、実行時にアプリケーションのプログラムによって設定しなければなりません。

設定可能なアイテム	説明
ナビゲーションバー	「Attributes」インスペクタのTop Bar属性の値を変更することによって、ビューがNavigation Barを持つかどうかを指定できます。この属性は、デザインのためだけに用意されており、これを使用すればビューとコントロールがNavigation Bar付きで表示されたときにどのように見えるか全体像がわかります。Interface Builderを使用して、さまざまなスタイルのNavigation Bar（プロンプトテキスト用のスペースを持つナビゲーションバーなど）を設定できます。 Top Bar属性の値はnibファイルには保存されません。Navigation Barの実際のスタイル（およびNavigation Barを表示するかどうか）は、それを所有するNavigation Controllerによって管理されます。Navigation Barの設定方法については、「Navigation Barの外観のカスタマイズ」（70 ページ）を参照してください。
タブバー	「Attributes」インスペクタのBottom Bar属性の値を変更することによって、ビューがTab Barを持つかどうかを指定できます。この属性は、主にデザインのために用意されており、これを使用すればTab Barが存在するときにビューとコントロールがどのように見えるか全体像がわかります。 実際にTab Bar Controllerを設定するときは、Tab Bar Itemを、Tab Barインターフェイスのタブに関連付けられている個々のView Controllerに追加できます。Tab Bar ControllerとTab Bar Itemの設定の詳細については、「Tab Barインターフェイスの作成」（82 ページ）を参照してください。
ツールバーアイテム	Navigation Controllerが提供するToolbarをビューで使用するには、「Attributes」インスペクタのBottom Bar属性をToolbarに設定します。この属性は、ナビゲーションインターフェイスを作成するときに、主にデザインのために使います。これを使用すれば、Tab Barが存在するときにビューとコントロールがどのように見えるか全体像がわかります。 実際にNavigation Controllerを設定するときには、View ControllerのToolBar用のBar Button Itemをnibファイルに含めることもできます。ナビゲーションインターフェイス内のツールバーの設定の詳細については、「ナビゲーションツールバーの表示」（76 ページ）を参照してください。
タイトル	統合nibファイルでは、Title属性に適切な値を設定することによって、View Controllerのタイトルを指定できます。Navigation ControllerとTab Bar Controllerは、View Controllerを表示するときに、この属性の値をデフォルト値として使用します。
nib名	View Controllerをnibファイルに保存するときは、NIB Name属性を使用して、そのView Controllerのビューを含む分離nibファイルの名前を指定します。分離nibファイルの設定方法については、「分離nibファイルにビューを保存する」（29 ページ）を参照してください。 実行時にView Controllerがメモリにロードされるときに、この属性の値がView Controllerの`nibName`プロパティに設定されます。

nibファイル内のNavigation ControllerとTab Bar Controllerの設定は、少し複雑です。それについては、次の場所で説明します。

- Navigation Controllerオブジェクトそのものの設定方法については、「ナビゲーションインターフェイスの作成」（60 ページ）を参照してください。
- Tab Bar Controllerオブジェクトそのものの設定方法については、「Tab Barインターフェイスの作成」（82 ページ）を参照してください。

- Navigation ControllerとTab Bar Controllerを組み合わせる場合の設定方法については、「Tab Barインターフェイスへの Navigation Controllerの追加」（101 ページ）を参照してください。

View Controllerのアクションとアウトレットを設定する

使用するnibファイルが分離か統合かに関わらず、View Controllerのアクションとアウトレットの設定方法は本質的に同じです。Interface Builderを使用して、インターフェイス内のビューとコントロールをView Controllerを表すオブジェクトに接続します。統合nibファイルの場合は、View Controllerオブジェクトへの接続を直接作成できます。一方、分離nibファイルの場合は、View Controllerの代わりにFile's Ownerプレースホルダへの接続を作成します。

リスト 2-2に、2つのカスタムアウトレット（`IBOutlet`キーワードで指定されている）と、1つのアクションメソッド（`IBAction`という戻り値型で指定されている）を定義するカスタムの`MyViewController`クラスの定義を示します。これらのアウトレットは、nibファイル内の1つのボタンと1つのテキストフィールドへの参照を格納します。一方、アクションメソッドはそのボタンのタップに応答します。

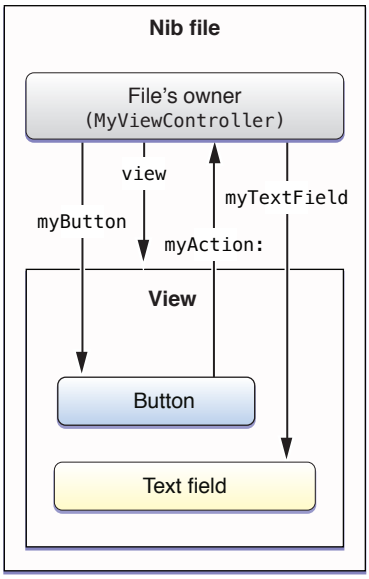
リスト 2-2　カスタムView Controllerクラスの宣言

```
@interface MyViewController :UIViewController
{
  id myButton;
  id myTextField;
}
@property (nonatomic) IBOutlet id myButton;
@property (nonatomic) IBOutlet id myTextField;

- (IBAction)myAction:(id)sender;
```

図 2-6に、このような`MyViewController`クラス内のオブジェクト間に作成する接続を示します。このnibファイルは、「View Controllerのアクションとアウトレットを設定する」（35 ページ）の手順に従って設定されています。File's Ownerプレースホルダのクラスは View Controllerクラスに設定され、File's Ownerの`view`アウトレットは最上位レベルのビューオブジェクトに接続されています。また、このnibファイルには、`MyViewController`クラスのアウトレットおよびアクションと、それに対応するnibファイルオブジェクトの間の接続も含まれています。

図 2-6 MyViewController.nibの内容



前述のように設定したMyViewControllerクラスが作成されてモーダルモードで表示されると、View Controllerのインフラストラクチャがこのnibファイルを自動的にロードし、アウトレットやアクションを再構成します。このように、ビューがユーザに表示されるまでには、View Controllerのアウトレットとアクションが設定されて使用できる状態になります。実行時のコードと設計時のリソースファイルをこのように連動させられることが、nibファイルの強みの1つです。

プログラムによるビューの作成

nibファイルを使用せずに、プログラムによってビューを作成したい場合は、ViewControllerのloadViewメソッドからそれを実行します。プログラムによってビューを作成する場合は、このメソッドをオーバーライドしなければなりません。このメソッドの実装では、次の処理を実行しなければなりません。

1. 画面にちょうど合うサイズのルートビューオブジェクトを作成します。

   このルートビューは、ViewControllerに関連付けられたその他のすべてのビューのコンテナとしての役割を果たします。通常は、このビューに対して、アプリケーションウインドウ（画面一杯に表示される）のサイズに一致するフレームを定義します。ただし、システムステータスバー、Navigation Bar、Tab Barなど各種のビューが存在するときに、View Controllerはこれらに対応してこのフレームのサイズを調整します。

   汎用のUIViewオブジェクト、独自に定義したカスタムビュー、その他、画面一杯に拡張できる任意のビューを使用できます。

2. 追加のサブビューを作成し、それらをルートビューに追加します。ビューごとに、次の処理を実行しなければなりません。

   a. ビューを作成して初期化します。システムビューの場合は、通常、initWithFrame:メソッドを使用して、そのビューの初期のサイズと位置を指定します。

   b. addSubview:メソッドを使用して、そのビューを親ビューに追加します。

c. releaseメソッドを呼び出して、そのビューを解放します。

3. ルートビューをView Controllerのviewプロパティに代入します。

4. ルートビューを解放します。

各ビューを作成した直後にそれを解放するというのは、奇妙に思われるかもしれません。しかし、ビューをビュー階層に追加するか、ビューへの参照を保存したら、解放することこそがまさに行うべきことです。どのオブジェクトも作成時の保持カウントの初期値は1です。親ビューは自動的にサブビューを保持するため、子ビューを親ビューに追加したら子ビューを解放しても安全です。同様に、ルートビューを格納するために使われるviewプロパティも保持のセマンティクスに従って、ビューが解放されるのを防ぎます。したがって、各ビューを解放することは、所有権を適切な場所に移して、後でそのオブジェクトの解放漏れが生じるのを防止することを意味します。

リスト 2-3に、*Metronome*サンプルプロジェクトのMetronomeViewControllerクラスのloadViewメソッドを示します。このメソッドでは、カスタムビューを作成して、カスタムビューをView Controllerのviewプロパティ（ビューを保持する）に代入するなどの基本的なセットアップを実行します。この例では、View ControllerのmetronomeViewプロパティは、ビューへのポインタを格納する追加プロパティですが、このプロパティでは、保持の問題を回避するために代入のセマンティクスを用います。

リスト 2-3　「Metronome」アプリケーションでのプログラムによるビューの作成

```
- (void)loadView {

    self.wantsFullScreenLayout = YES;

    MetronomeView *view = [[MetronomeView alloc] initWithFrame:[UIScreen
mainScreen].applicationFrame];
    view.metronomeViewController = self;
    self.view = view;
    self.metronomeView = view;

    [view release];
}
```

注： loadViewメソッドをオーバーライドして、プログラムによってビューを作成するときは、superを呼び出してはいけません。これを行うと、ビューをロードするデフォルトの動作が開始されます。通常、これはCPUサイクルを浪費するだけです。loadViewメソッドの独自実装では、View Controller用のルートビューとサブビューの作成に必要なすべての処理を実行するべきです。ビューのロード処理の詳細については、「ビュー管理サイクルの理解」（26 ページ）を参照してください。

プロパティやカスタムsetterメソッドを使用して、任意のビューオブジェクトを保持する場合は、必ずviewDidUnloadメソッドを実装してこれらのプロパティをnilに設定することを忘れないでください。特に、アウトレットを使用してビューへの参照を格納したり、これらのアウトレットで、保持セマンティクスを持つプロパティやその他のsetterメソッドを使用する場合は、この点に注意してください。ビューに関連付けられているメモリの管理の詳細については、「効率的なメモリ管理」（46 ページ）を参照してください。

ビューのアンロード後のクリーンアップ

View Controllerのビューは、メモリにロードされたら、メモリ不足の状態が発生するか、View Controller自身が割り当て解除されるまではメモリ内に留まります。メモリ不足状態の場合、`UIViewController`のデフォルト動作では、ビューが現在使われていなければ、`view`プロパティに格納されているビューオブジェクトが解放されます。しかし、カスタムView Controllerクラスで、ビュー階層内のビューへのアウトレットやポインタを格納している場合は、最上位レベルのビューオブジェクトが解放されたときに、これらの参照も解放しなければなりません。それを怠ると、これらのオブジェクトはすぐにメモリから削除されず、その後これらのオブジェクトへのポインタを上書きすると、後でメモリリークの原因となる可能性があります。

View Controllerが必ずビューオブジェクトへの参照をクリーンアップしなければならない場所は、次の2ヶ所です。

- `dealloc`メソッド
- `viewDidUnload`メソッド

宣言済みプロパティを使用してビューへの参照を格納し、そのプロパティが保持セマンティクスを使用している場合、そのプロパティに`nil`値を代入するだけで、そのビューを解放できます。プロパティは、その便利さから、ビューオブジェクトの管理手段として圧倒的に好まれています。プロパティを使用しない場合は、明示的に保持したビューに`release`メッセージを送信してから、それに対応するポインタ値を`nil`に設定しなければなりません。

メモリ管理のベストプラクティスについては、「効率的なメモリ管理」（46 ページ）を参照してください。

インターフェイスの向きの管理

iPhone OSベースのデバイスでは、加速度センサーを利用してデバイスの現在の向きを判別できます。UIKitフレームワークはこの情報を利用して、アプリケーションのユーザインターフェイスを必要に応じてデバイスの向きに合わせることができます。デフォルトでは、アプリケーションは縦長の向きのみをサポートします。必要であれば、それ以外の向きをサポートするようにView Controllerを設定できます。

別の向きをサポートするには、ビューとそれを管理するView Controllerの両方に追加の設定をする必要があります。インターフェイスで複数の向きをサポートするための最も簡単な方法は、次の手順を実行することです。

- View Controllerの`shouldAutorotateToInterfaceOrientation:`メソッドをオーバーライドして、サポートする向きを宣言します（「サポートするインターフェイスの向きの宣言」（39ページ）を参照）。
- View Controllerのビュー階層内の各ビューに自動サイズ調整マスクを設定します（「複数の向きをサポートするためのビューの設定」（40 ページ）を参照）。

ほとんどのアプリケーションでは、この2つのステップで十分です。しかし、ビューの自動サイズ調整動作では、それぞれの向きに必要なレイアウトにならない場合は、View Controllerのその他のメソッドをオーバーライドし、それらを使用して向きが変化したときのレイアウトを調整します。`UIViewController`クラスは、向きの変化のさまざまフェーズに応答したり、必要に応じてビュー（またはアプリケーションのその他の部分）を調整できるように、一連の通知を提供します。これらの通知については、「向きの変化への応答」（40 ページ）で詳しく説明します。

回転処理について

iPhone OSベースのデバイスの向きが変化すると、システムは、変化が生じたことを、関係する部分に知らせるために`UIDeviceOrientationDidChangeNotification`通知を送信します。デフォルトでは、UIKitフレームワークがこの通知をインターセプトし、それを使用してインターフェイスの向きを自動的に更新します。つまり、わずかな例外を除けば、この通知を処理する必要はまったくありません。デベロッパがしなければならないのは、向きの変化に応答できるように、View Controllerクラスに適切なメソッドを実装することだけです。

iPhoneアプリケーションでは、ウインドウオブジェクトが現在の向きの変化に関連する処理のほとんどを実行します。ただし、ウインドウオブジェクトは、アプリケーションのView Controllerと連携しながら、向きが変化したかどうかを判別して、向きが変化した場合は、その変化に応答するために呼び出すメソッドを決定します。具体的に言うと、ウインドウは、一番最近ウインドウに追加また表示されたルートビューを持つView Controllerと連携します。つまり、ウインドウオブジェクトは、「View Controllerのビューの表示」（48 ページ）で説明するメカニズムのいずれかを使用して表示されたビューを持つ最前面のView Controllerとのみ連携します。

実際の回転処理は、対応するView Controllerの実装に応じて、2つの道筋のいずれかに従って進みます。最も一般的な道筋は、1ステップの回転を実行する方法です。ただし、View Controllerは2ステップの回転もポートします。1ステップの回転処理は、iPhone OS 3.0以降で利用できます。2ステップの処理よりも効率的なため、好んで使用されます。どちらの道筋が選択されるかは、View Controllerサブクラスとオーバーライドするメソッドによって決まります。2ステップの処理だけに対応するメソッドをオーバーライドすると、ウインドウオブジェクトは2ステップの処理を使用します。それ以外の場合は、1ステップの処理を使用します。

どちらの回転処理を使用するかに関わらず、View Controllerのメソッドは回転のさまざまな段階で呼び出されるので、タスクを追加実行する機会があります。これらのメソッドを使用して、ビューの表示や非表示、ビューの位置やサイズの変更、アプリケーションのほかの部分への向きの変化の通知が行えます。これらのカスタムメソッドは回転動作中に呼び出されるため、そこで時間のかかる操作を実行することは避けるべきです。また、ビュー階層全体を新しいビューセットに置きかえる操作も避けるべきです。向きの違いに応じて特有のビューを提供する場合は、新規のView Controllerをモーダルモードで表示するなどの、効率的な方法もあります（「代替の横長インターフェイスの作成」（45 ページ）を参照)。

1ステップおよび2ステップの回転処理で発生する一連のステップの詳細については、「向きの変化への応答」（40 ページ）を参照してください。

サポートするインターフェイスの向きの宣言

View Controllerが管理するビューで、デフォルトの縦長以外の向きをサポートする場合は、`shouldAutorotateToInterfaceOrientation:`メソッドをオーバーライドして、そのビューでサポートする向きを指定しなければなりません。設計時には、ビューでサポートする向きを必ず選択して、それらの向きを考慮してコードを実装しなければなりません。実行時の情報に基づいて、サポートする向きを動的に選択する方法にはメリットはありません。たとえ、そのような選択をしても、可能性のあるすべての向きをサポートするために必要なコードを実装しなければなりません。そうであれば、それらの向きをサポートするかどうかを事前に選択しておいても同じです。

リスト 2-4に、デフォルトの縦長と横長左向きをサポートするView Controllerの`shouldAutorotateToInterfaceOrientation:`メソッドの典型的な実装を示します。このメソッドを独自に実装する場合も、これと同じくらい単純にするべきです。

リスト 2-4　shouldAutorotateToInterfaceOrientation:メソッドの実装

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)orientation
{
   if ((orientation == UIInterfaceOrientationPortrait) ||
       (orientation == UIInterfaceOrientationLandscapeLeft))
      return YES;

   return NO;
}
```

注： アプリケーションで両方向の横長をサポートする場合は、*orientation*パラメータを両方の横長定数と明示的に比較する代わりに、UIInterfaceOrientationIsLandscapeマクロを近道として使用できます。同様に、UIKitフレームワークには、両方の縦長変数を識別するUIInterfaceOrientationIsPortraitマクロが定義されています。

複数の向きをサポートするためのビューの設定

ユーザインターフェイスの向きが変わると、自動サイズ変更マスクに従って、影響を受けるビューの境界が自動的に変更されます。すべてのビューのautoresizingMaskプロパティには、スーパービューを規準にしてそのビューの境界がどのように変化するかを表す定数が含まれています。各ビューは自身の境界を調整すると、それぞれのサブビューに、自動サイズ変更動作に基づいてサイズを変更するように指示します。最終的には、ビューの自動サイズ変更動作を適切に設定しておけば、そのビューは自動的に向きの変化に対応します。

ビューの自動サイズ変更動作では、正確に望みどおりのレイアウトにならない場合は、独自のカスタムレイアウトコードを用意してその動作を置き換えたり補うことができます。UIViewControllerクラスには、向きが変化する前、最中、および後に呼び出されるいくつかのメソッドが定義されています。これらのメソッドを使用して、必要に応じてビューのレイアウトを変更できます。

回転処理の最中に呼び出されるメソッドについては、「向きの変化への応答」（40 ページ）を参照してください。ビューの自動サイズ変更プロパティと、それらがビューに与える影響の詳細については、『*iPhone Application Programming Guide*』の「Autoresizing Behaviors」を参照してください。

向きの変化への応答

デバイスの向きが変化したときは、View Controllerがビューの向きを適切に変更することによってそれに応答します。新たな向きをサポートしていれば、View Controllerはその変更に伴って通知を生成、それに応答するコードに実行の機会を与えます。回転通知は、1ステップ処理でも2ステップ処理でも発生します。

向きの変化に応答する理由の1つは、ビュー階層を調整するためです。たとえば、これらの通知を使用して、次のような変更を行うことができます。

- 特定の向きに特有のビューを表示または非表示にします。
- 新しい向きに基づいてビューの位置やサイズを調整します。
- 向きの変化を反映して、アプリケーションのその他の部分を更新します。

UIKitフレームワークは可能であれば必ず1ステップの回転処理を使用してビューを回転します。ただし、1ステップの処理を使用するか、2ステップの処理を使用するかはデベロッパに任されています。2ステップの処理では使われるが、1ステップの処理では使われないメソッドもいくつかありま

す。2ステップ用のメソッドをオーバーライドした場合は、2ステップの処理が使われます。1ステップの処理で使われるメソッドだけをオーバーライドした場合は、1ステップの処理が使われます。以降の各セクションでは、それぞれの処理に関連するメソッドについて説明します。これらのメソッドについての情報（これらのメソッドが2ステップの処理を起動するかどうかを含む）は、『*UIViewController Class Reference*』でも参照できます。

1ステップで向きの変化に応答する

iPhone OS 3.0以降では、1ステップの回転メソッドを使用して、向きの変化が発生する直前または直後に変更を行うことができます。この処理では、次の一連のことが順番に起こります。

1. ウインドウが、デバイスの向きに変化が生じたことを検出します。
2. ウインドウは適切なView Controllerを検索して、その`shouldAutorotateToInterfaceOrientation:`メソッドを呼び出し、新しい向きをサポートするかどうかを判断します。

   コンテナView Controllerは、このメソッドをインターセプトし、独自のヒューリスティクスに基づいて、向きを変更すべきかどうかを判断することも可能です。たとえば、Tab Bar Controllerは、管理下にあるすべてのビューが新しい向きをサポートする場合にのみ、向きを変更できます。
3. 新しい向きがサポートされる場合、ウインドウはView Controllerの`willRotateToInterfaceOrientation:duration:`メソッドを呼び出します。

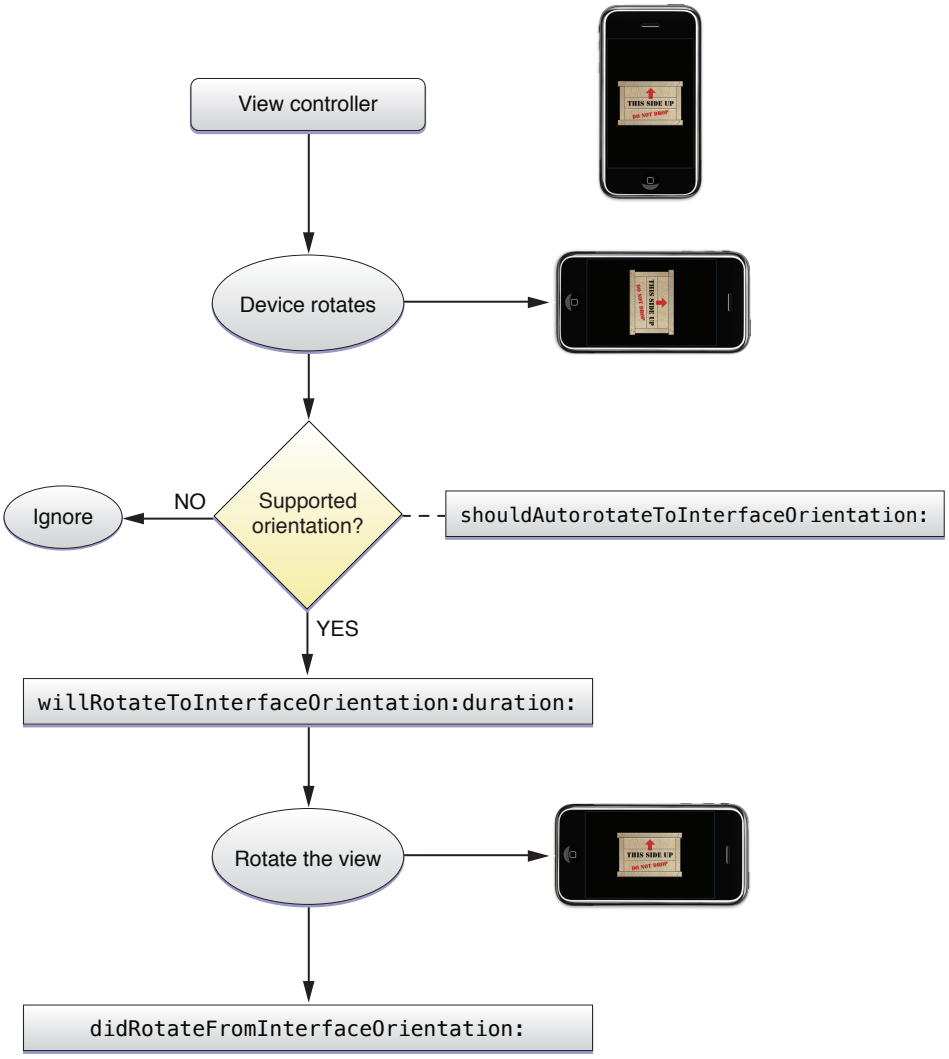
   コンテナView Controllerは、このメッセージを現在表示中のカスタムView Controllerに転送します。カスタムView Controllerでこのメソッドをオーバーライドして、インターフェイスが回転する前に、ビューを非表示にしたり、ビューのレイアウトにその他の変更を加えることもできます。
4. ウインドウは、View Controllerのビューの境界を調整します。

   これによって、ビュー階層内の各ビューは、自動サイズ変更マスクに基づいてサイズ変更されます。
5. ウインドウはView Controllerの`didRotateFromInterfaceOrientation:`メソッドを呼び出します。

   コンテナView Controllerは、このメッセージを現在表示中のカスタムView Controllerに転送します。これは、回転処理の終了を表します。このメソッドを使用して、ビューを表示したり、ビューのレイアウトを変更したり、アプリケーションにその他の変更を加えたりできます。

図 2-7は、上記のステップを図示したものです。また、この図には、回転処理のさまざまな段階でのインターフェイスの外観も示されています。

図 2-7 1ステップのインターフェイス回転処理



2ステップで向きの変化に応答する

すべてのバージョンのiPhone OSで、インターフェイスの向きの変更に応答するために、2ステップの通知を使用できます。2ステップの処理では、2つの独立した回転が実行されます。最初のステップでは、インターフェイスが回転方向に半分だけ回転します。次のステップで、この中間地点から最終的な向きまで回転します。アプリケーションはこの処理を通して、回転の前、最中、後に応答できるように通知を受信します。

2ステップの回転では、次の一連のことが順番に起こります。

1. ウインドウが、デバイスの向きに変化が生じたことを検出します。
2. ウインドウは適切なView Controllerを検索して、その`shouldAutorotateToInterfaceOrientation:`メソッドを呼び出し、新しい向きをサポートするかどうかを判断します。

コンテナView Controllerは、このメソッドをインターセプトし、独自のヒューリスティクスに基づいて、向きを変更すべきかどうかを判断することも可能です。たとえば、Tab Bar Controllerは、管理下にあるすべてのビューが新しい向きをサポートする場合にのみ、向きを変更できます。

3. 新しい向きがサポートされる場合、ウインドウはView Controllerの`willRotateToInterfaceOrientation:duration:`メソッドを呼び出します。

   コンテナView Controllerは、このメッセージを現在表示中のカスタムView Controllerに転送します。このメソッドを使用して、インターフェイスが回転する前に、ビューを非表示にしたり、ビューのレイアウトにその他の変更を加えることができます。

4. ウインドウはView Controllerの`willAnimateFirstHalfOfRotationToInterfaceOrientation:duration:`メソッドを呼び出します。

   コンテナView Controllerは、このメッセージを現在表示中のカスタムView Controllerに転送します。このメソッドを使用して、インターフェイスが回転する前に、ビューを非表示にしたり、ビューのレイアウトにその他の変更を加えることができます。

5. ウインドウは最初の半分の回転を実行します。

   これによって、ビュー階層内の各ビューの境界は、自動サイズ変更動作に基づいて調整されます。ほとんどの回転は縦長モードから横長モードへの変更ですが（したがって、中間地点までは45度回転します）、横長左向きから横長右向きに回転する場合や、縦長で上下逆さまに回転することも可能です。後者の場合は、最初の半分の回転が90度になります。

6. ウインドウはView Controllerの`didAnimateFirstHalfOfRotationToInterfaceOrientation:`メソッドを呼び出します。

   コンテナView Controllerは、このメッセージを現在表示中のカスタムView Controllerに転送します。

7. ウインドウはView Controllerの`willAnimateSecondHalfOfRotationFromInterfaceOrientation:duration:`メソッドを呼び出します。

   コンテナView Controllerは、このメッセージを現在表示中のカスタムView Controllerに転送します。

8. ウインドウは残りの半分の回転を実行します。

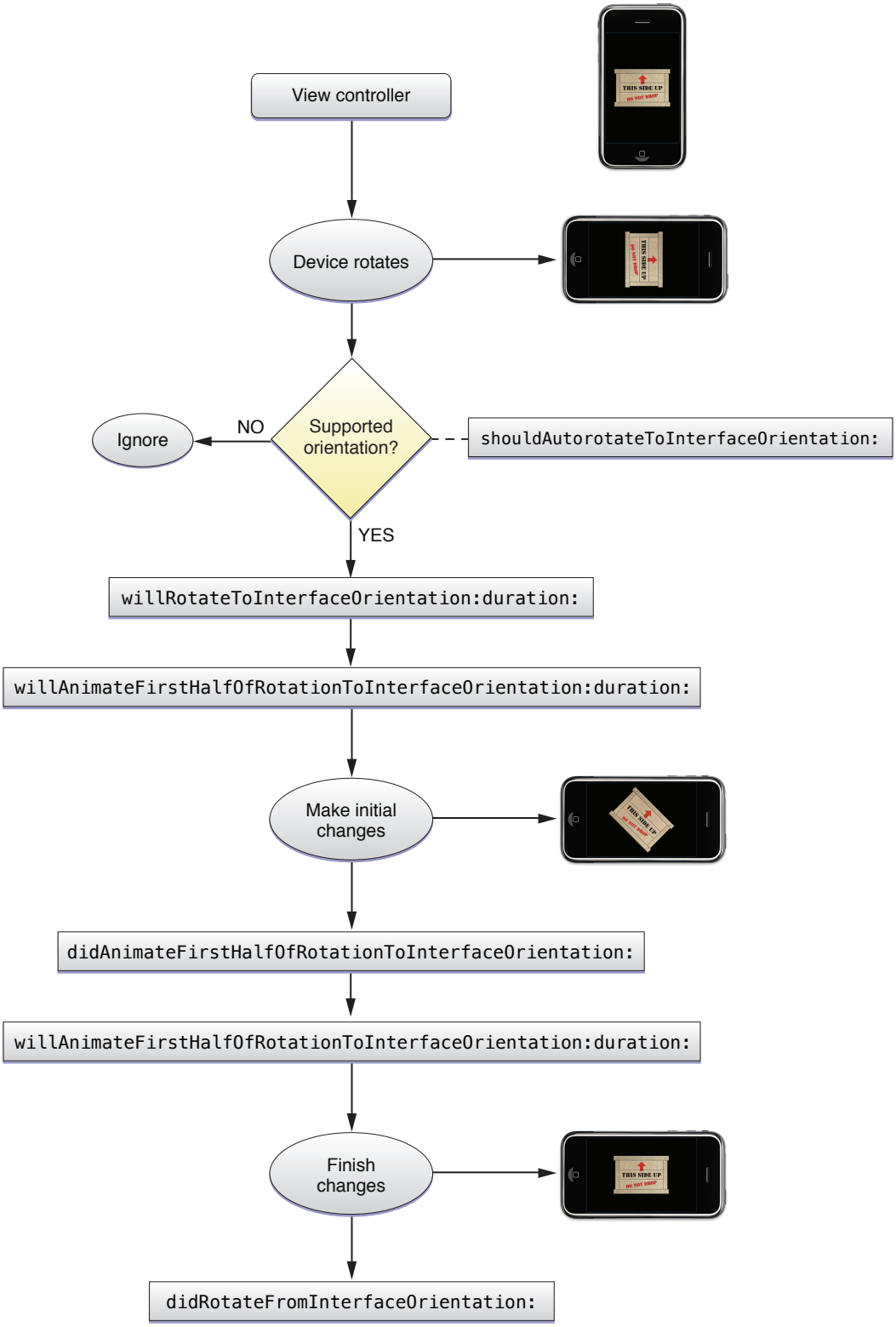
   この回転が終了した時点で、すべてのビューに自動サイズ変更動作が適用されて、ビューが「最終的な」位置に配置されます。

9. ウインドウはView Controllerの`didRotateFromInterfaceOrientation:`メソッドを呼び出します。

   コンテナView Controllerは、このメッセージを現在表示中のカスタムView Controllerに転送します。このメソッドを使用して、回転が終了した後に、ビューを表示したり、ビューの位置やサイズを変更したり、その他の変更を加えることができます。

図 2-8は、上記のステップを図示したものです。各ステップに応じて、ユーザに表示されるビューの外観も示しています。

図 2-8 2ステップのインターフェイス回転処理



代替の横長インターフェイスの作成

デバイスが縦長か横長かによって、同じデータを違う形式で表示したい場合、それを実現するには、2つの別々のView Controllerを使用します。一方のView Controllerは主たる向き（通常は縦長）でのデータの表示を管理し、もう一方のView Controllerは、別の向きでのデータの表示を管理します。2つのView Controllerを使用する方が、向きが変わるたびにビュー構成を大幅に変更するよりも簡単かつ効率的です。これによって、それぞれのView Controllerは、1つの向きでのデータの表示に集中できて、それに応じて要素を管理できます。また、現在の向きを確認するための条件判断によって、View Controllerのコードが煩雑になるのを避けることもできます。

別の横長インターフェイスをサポートするには、次の処理を実行しなければなりません。

- 次の2つのView Controllerオブジェクトを実装します。
  - 1つは縦長のみのインターフェイスを表示します。
  - もう1つは横長のみのインターフェイスを表示します。
- `UIDeviceOrientationDidChangeNotification`通知の受け取りを登録します。このハンドラメソッド内で、デバイスの現在の向きに応じて、代替View Controllerを表示または非表示にします。

View Controllerは、通常は向きの変化を内部的に管理するため、1つの向きの場合にだけ表示されるように各View Controllerに指示しなければなりません。さらに、主たるView Controllerの実装では、デバイスの向きの変化を検出して、適切な向きに変化したときに代替View Controllerを表示する必要があります。主たる向きに戻ったときは、主たるView Controllerが代替View Controllerを消去します。

リスト 2-5に、縦長をサポートする主たるView Controllerに実装しなければならない主要なメソッドを示します。初期化の一部として、このView Controllerは、向き変更通知を共有の`UIDevice`オブジェクトから受信するように登録します。このような通知を受信すると、`orientationChanged:`メソッドは、現在の向きに応じて横長用のView Controllerを表示または消去します。

リスト 2-5　横長用のView Controllerの表示

```
@implementation PortraitViewController
- (id)init
{
   self = [super initWithNibName:@"PortraitView" bundle:nil];
   if (self)
   {
       isShowingLandscapeView = NO;
       self.landscapeViewController = [[[LandscapeViewController alloc]
                           initWithNibName:@"LandscapeView" bundle:nil]
autorelease];

      [[UIDevice currentDevice] beginGeneratingDeviceOrientationNotifications];
       [[NSNotificationCenter defaultCenter] addObserver:self
                                  selector:@selector(orientationChanged:)
                                  name:UIDeviceOrientationDidChangeNotification
                                  object:nil];
   }
   return self;
}

- (void)orientationChanged:(NSNotification *)notification
{
```

```
    UIDeviceOrientation deviceOrientation = [UIDevice currentDevice].orientation;
    if (UIDeviceOrientationIsLandscape(deviceOrientation) &&
        !isShowingLandscapeView)
    {
        [self presentModalViewController:self.landscapeViewController
                                 animated:YES];
        isShowingLandscapeView = YES;
    }
    else if (deviceOrientation == UIDeviceOrientationPortrait &&
             isShowingLandscapeView)
    {
        [self dismissModalViewControllerAnimated:YES];
        isShowingLandscapeView = NO;
    }
}
```

効率的なメモリ管理

View Controllerとメモリ管理と言えば、検討しなければならない問題は次の2つです。

- どのようにしてメモリを効率的に割り当てるか？
- いつ、どのようにメモリを解放するか？

メモリ割り当てについては、完全にデベロッパが判断すべき部分もありますが、`UIViewController`クラスにはメモリ管理タスクに関連するメソッドがいくつかあります。表 2-2は、View Controllerオブジェクト内でメモリの割り当てや解放を行う可能性のある場所と、それぞれの場所で実行しなければならない処理のリストです。

表 2-2　メモリの割り当てや解放を行う場所

タスク	メソッド	説明
View Controllerで必要な重要なデータ構造を割り当てる	初期化メソッド	カスタム初期化メソッドには（`init`という名前であるか、それ以外の名前であるかに関わらず）、常にView Controllerオブジェクトを既知の適切な状態に設定する責任があります。これには、適切な操作を保証するために必要なあらゆるデータ構造の割り当てが含まれます。
ビューオブジェクトを作成する	`loadView`	プログラムによってビューを作成する場合にのみ、`loadView`メソッドのオーバーライドが必要になります。nibファイルからビューをロードする場合は、`initWithNibName: bundle:`メソッドを使用して、適切なnibファイルの情報を使用してView Controllerを初期化するだけです。
ビューに表示するデータの割り当てやロードを行う	`viewDidLoad`	ビューオブジェクトに関連付けられているデータはすべて、`viewDidLoad`メソッドで作成またはロードしなければなりません。このメソッドが呼び出された時点で、ビューオブジェクトの存在と、それが既知の適切な状態にあることが保証されます。

タスク	メソッド	説明
ビューオブジェクトへの参照を解放する	`viewDidUnload` `dealloc`	クラスのアウトレットやその他のメンバ変数を使用して、ビュー階層内のビューオブジェクトを保持する場合は、そのビューが不要になったら、それらのアウトレットを必ず解放しなければなりません。ビューオブジェクトを解放したら、安全のために、必ずアウトレットや変数を`nil`に設定します。ビューが解放されるタイミングの詳細については、「ビュー管理サイクルの理解」（26 ページ）を参照してください。
ビューが表示されていないときに必要のないデータを解放する	`viewDidUnload`	ビュー固有のデータのうち、ビューが再びメモリにロードされたときに簡単に再作成できるデータを解放するには、`viewDidUnload`メソッドを使用します。ただし、データの再作成にあまりに時間がかかる場合は、それに対応するデータオブジェクトをここで解放する必要はありません。代わりに、`didReceiveMemoryWarning`メソッドでこれらのオブジェクトを解放することを検討するべきです。
メモリ不足通知に応答する	`didReceiveMemory-Warning`	View Controllerに関連付けられているカスタムデータ構造のうち、重要でないものを解放するには、このメソッドを使用します。このメソッドは、ビューオブジェクトへの参照を解放するためには使用しませんが、`viewDidUnload`メソッドで解放しなかったビュー関連のデータ構造を解放するために使用します（ビューオブジェクトそのものは、必ず`viewDidUnload`メソッドで解放しなければなりません）。
View Controllerで必要な重要なデータ構造を解放する	`dealloc`	View Controllerに関連付けられているすべてのデータ構造を解放するために、このメソッドを使用します。View Controllerに`nil`以外の値を持つアウトレットや変数が残っている場合は、ここでそれらを解放しなければなりません。

実行時のカスタムView Controllerオブジェクトの作成

カスタムView Controllerオブジェクトを作成するには、プログラムによる方法とnibファイルを使用する方法の2つの方法があります。どちらの方法を使用するかは、ユーザインターフェイスの構成によって決まります。Tab Bar ControllerやNavigation Controllerを組み込んだ複雑なインターフェイスの場合、通常は、少なくともいくつかのカスタムView Controllerをアプリケーションのメインnibファイルに保存し、残りをプログラムで作成します。それ以外のほとんどの場合は、必要な場合にのみView Controllerをプログラムで作成するべきです。

プログラムでカスタムView Controllerを作成するには、次のようなコードを使用します。

```
MyViewController* vc = [[MyViewController alloc]
initWithNibName:@"MyViewController"
                                bundle:nil];
```

この例では、View Controllerが重要な初期化を実行しないことを前提としています。重要な初期化を実行する場合は、オーバーライドバージョンの`initWithNibName:bundle:`メソッドで初期化します。しかし、カスタムView Controllerを設計する場合によく使うアプローチは、1つ以上のカスタム初期化メソッドを定義する方法です。これによって、View Controllerの初期化の不変部分（nibファイル名とバンドル名の指定など）を隠して、View Controllerの初期化に使用するデータに集中できます。たとえば、オブジェクトの配列を引数にとるカスタム初期化メソッドは、次のようになります。

```
- (id)initWithData:(NSArray*)data {
    if ((self = [super initWithNibName:@"MyViewController" bundle:nil])) {
        // View Controllerを初期データで初期化する
    }
    return self;
}
```

このような初期化メソッドを使用して、View Controllerを作成および初期化し、すぐに使える状態にします。次に、View Controllerを表示したり、それをナビゲーションインターフェイスに追加します。たとえば、モーダルモードで新規のView Controllerを表示するには、現在のView Controllerに次のようなメソッドを定義します。

```
- (void)presentModalViewControllerWithData:(NSArray*)data {
   MyViewController* vc = [[MyViewController alloc] initWithData:data];
   [self presentModalViewController:vc animated:YES];
}
```

nibファイルを使用してView Controllerを作成する処理はもう少し手間がかかりますが、これについては、「ビューとView Controllerを同じnibファイルに保存する」（31 ページ）を参照してください。実行時にView Controllerのビューを表示する方法については、「View Controllerのビューの表示」（48 ページ）を参照してください。

View Controllerのビューの表示

View Controllerに対応するビューを表示するには、主に次の2つの選択肢があります。

- `addSubview:`メソッドを使用してビューをウインドウに追加することによって、直接ビューを表示する。
- 次のいずれかの方法を使用して、間接的にビューを表示する。
  - `presentModalViewController:animated:`メソッドを使用して、ビューを所有するView Controllerをモーダルモードで表示する。
  - ビューを所有するView ControllerをNavigation Controllerオブジェクトのナビゲーションスタックにプッシュする。
  - ビューを所有するView ControllerをTab Barインターフェイス内のタブのルートView Controllerにする。

リスト 2-6は、アプリケーションのメインウインドウに直接ビューを表示する方法の例です。この例では、`viewController`変数は、メインnibファイルからロードされたView Controllerへのポインタを格納するアウトレットです。同様に、`window`変数には、アプリケーションウインドウへのポインタが格納されます（View-based Applicationプロジェクトテンプレートを使用すると、アプリケーショ

ンデリゲート内にこれとまったく同じコードが作成されます）。ビューをウインドウに追加すると、そのビューがロードされて、次にそのウインドウが表示されたときに、そのビューが表示されます。

リスト 2-6　View Controllerのビューをウインドウに追加する

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // この時点で、メインnibファイルがロードされる。
    // ビューをウインドウに追加する前に、そのビューのフレームを設定しておくとよい。
    [viewController.view setFrame:[[UIScreen mainScreen] applicationFrame]];

    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}
```

注： `addSubview:`メソッドを使用して、ビューをウインドウに追加する場合は、nibファイルからビューをロードした後に、そのビューのフレームを明示的に設定することをお勧めします。nibファイルにはルートビューの最適なサイズが保存されていますが、フレームを明示的に設定することによって、そのビューがウインドウ内で適切なサイズと位置になることが保証されます。モーダルモードでビューを表示する場合や、コンテナView Controllerと組み合わせてビューを使用する場合は、明示的にフレームを設定する必要はありません。

View Controllerのビューを表示する場合は、ここで提案した方法のみを使用することをお勧めします。ビューを適切に表示したり管理するために、システムは、直接または間接的に表示するそれぞれのビュー（およびそれに対応するView Controller）を記録します。システムは、後でこの情報を使用してView Controller関連のイベントをアプリケーションに報告します。たとえば、デバイスの向きが変化した場合、ウインドウはこの情報を使用して、最前面のView Controllerを識別し、それに変化を通知します。その他の方法（ほかのビューのサブビューとしてビューを追加するなど）によってView Controllerのビューをビュー階層に組み込むと、そのビューはデベロッパが自分で管理すると見なされて、システムは、対応するView Controllerオブジェクトにメッセージを送信しません。

アプリケーションの初期のインターフェイスをセットアップするとき以外は、ほとんどのビューはView Controllerオブジェクトによって間接的に表示されます。ビューを間接的に表示する方法については、以下のセクションを参照してください。

- モーダルモードでビューを表示する方法については、「モーダルモードでのView Controllerの表示」（96 ページ）を参照してください。
- ナビゲーションインターフェイスにビューを表示する方法については、「ナビゲーションスタックの変更」（67 ページ）を参照してください。
- タブにビューを表示する方法については、「Tab Barインターフェイスの作成」（82 ページ）を参照してください。

表示関連の通知への応答

View Controllerのビューの外観が変化すると、View Controllerはいくつかの組み込みメソッドを呼び出して、その変化をサブクラスに通知します。これらの組み込みメソッドを使用して、外観の変化に適切に対応できます。たとえば、これらの通知を使用して、これから表示されるビューの表示ス

タイルに合うようにステータスバーの色と向きを変更できます。ビューが画面に表示されようとしているか、画面から消去されようとしているかに応じて、View Controllerは異なるメソッドを呼び出します。

図 2-9に、View Controllerのビューがウインドウに追加されるときに起きる一連のことの基本的な順番を示します（そのビューがすでにウインドウ内に存在するが、現在は別のビューによって隠れている場合は、これらの障害が取り除かれてビューが再び表示されたときに、これと同じ順序で一連のことが起きます）。`viewWillAppear:`メソッドと`viewDidAppear:`メソッドは、サブクラスに、ビューの表示に関連する追加アクションを実行する機会を与えます。

図 2-9　ビューの表示に応答する

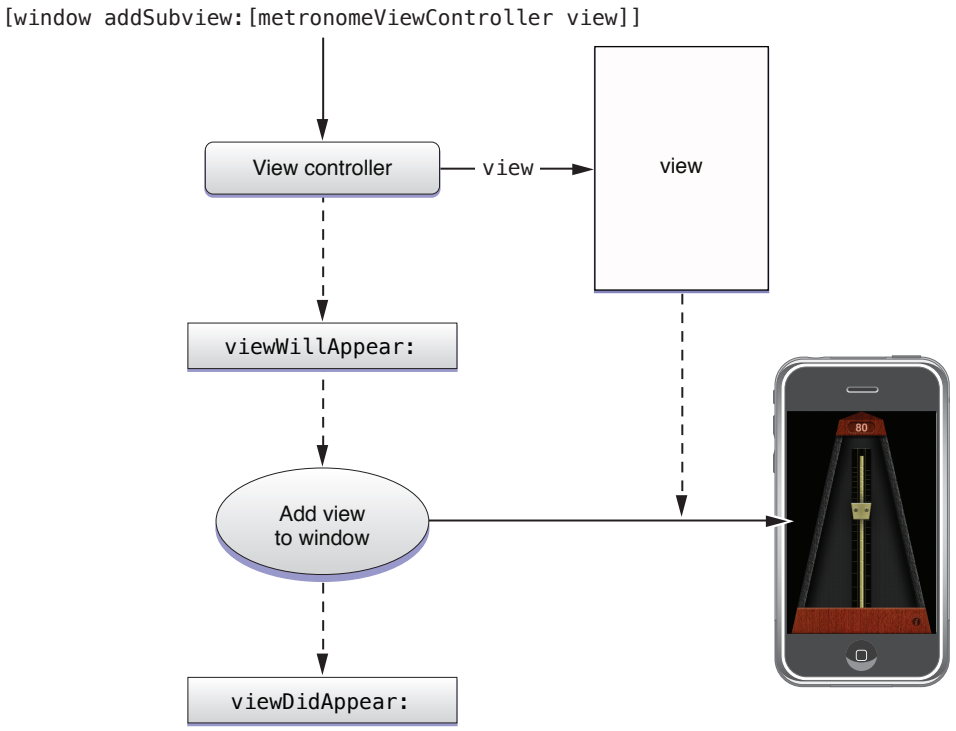
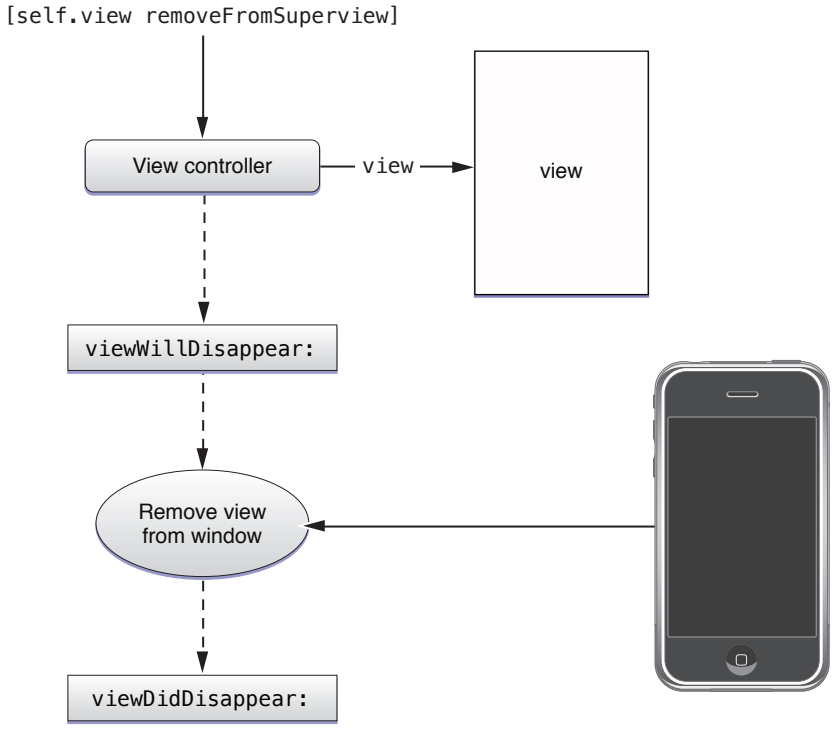


図 2-10に、ビューがウインドウから消去されるときに起きる一連のことの基本的な順序を示します（ビューが別のビューによって完全に隠れてしまう場合も（既存のView Controllerの上に新規のView Controllerが表示される場合など）、これと同じ一連のことが起きます）。View Controllerは、ビューが消去または隠されようとしていることを検出すると、`viewWillDisappear:`メソッドと`viewDidDisappear:`を呼び出して、サブクラスに、関連するタスクを実行する機会を与えます。

図 2-10 ビューの消去に応答する



カスタムビューでのフルスクリーンレイアウトの採用

アプリケーションでステータスバーを表示すると、`UIViewController`クラスは、ビューがステータスバーの下に隠れないように自動的にビューを縮小します。つまり、ステータスバーが不透明な場合は、その下に重なるようにコンテンツを表示したり、そのようなコンテンツとやり取りをする方法はありません。ただし、アプリケーションが半透明なステータスバーを表示する場合は、View Controllerの`wantsFullScreenLayout`プロパティを`YES`に設定すれば、ステータスバーの下にビューを重ねて表示できます。

ステータスバーの背後に重ねて表示する方法は、コンテンツを表示するために利用できるスペースを最大にしたい場合に役立ちます。ステータスバーの背後にコンテンツを表示する場合は、ユーザがスクロールバーの下にあるコンテンツをスクロールして見ることができるように、必ずそのコンテンツをスクロールビュー内に置かなければなりません。ユーザはステータスバーやその他の半透明なビューの背後にあるコンテンツとやり取りをすることはできないため、コンテンツをスクロールできるようにすることは重要です。Navigation Barは、Navigation Barの高さを考慮して、自動的にスクロールビュー（View Controllerのルートビューと仮定する）にスクロールコンテンツ挿入枠を追加します。それ以外の場合は、スクロールビューの`contentInset`を手動で変更しなければなりません。

Navigation Controllerと組み合わせて使うView Controllerで、フルスクリーンレイアウトを採用する方法の詳細については、「ナビゲーションビューでのフルスクリーンレイアウトの採用」（66 ページ）を参照してください。

ビューの編集モードの有効化

同じView Controllerを使用して、コンテンツの表示と編集を行う場合は、`setEditing:animated:`メソッドをオーバーライドし、このメソッドを使用してView Controllerのビューの表示モードと編集モードを切り替えます。このメソッドの実装では、メソッドが呼び出されたときに、指定のモードに合うようにView Controllerのビューを追加、削除、および調整しなければなりません。たとえば、ビューが現在編集可能であることを伝えるために、コンテンツを変更したり、ビューの外観を変更するこができます。View Controllerがテーブルを管理している場合は、そのテーブル自身の`setEditing:animated:`メソッドを呼び出して、そのテーブルを適切なモードに設定することもできます。

注： 通常は、表示モードと編集モードを切り替えるときに、ビュー階層全体を入れ替えるようなことはしません。実際、`setEditing:animated:`メソッドを使用することのねらいは、既存のビューを少しだけ変更できる点にあります。編集用に新たなビューセットを表示する場合は、新規のView Controllerをモーダルモードで表示するか、Navigation Controllerを使用して新規のビューを表示するかのいずれかにするべきです。

図 2-11に、その場での編集をサポートする「連絡先(Contacts)」アプリケーションのビューを示します。右上隅の「編集(Edit)」ボタンをタップすると、View Controllerが編集モードに切り替わります。「完了(Done)」ボタンをタップすると、表示モードに戻ります。テーブルの変更だけでなく、このビューでは、画像ビューや、ユーザ名を表示するビューのコンテンツを変更することもできます。また、関連するビューやセルをタップすると、ほかのアクションを実行するのではなく、そのコンテンツの編集ができるように設定されています。

図 2-11 ビューの表示モードと編集モード

Display mode **Edit mode**

独自の`setEditing:animated:`メソッドの実装は比較的簡単です。View Controllerがどちらのモードに入ろうとしているかを確認し、それに応じてビューのコンテンツを調整するだけです。

```
- (void)setEditing:(BOOL)flag animated:(BOOL)animated
{
    [super setEditing:flag animated:animated];
    if (flag == YES){
        // ビューを編集モードに変更する
    }
    else {
        // 必要であれば変更を保存し、ビューを編集不可に変更する
    }
}
```

編集可能なビューをよく使用する場所は、ナビゲーションインターフェイス内です。ナビゲーションインターフェイスを実装する場合は、編集可能なView Controllerを表示するときに、Navigation Barに特殊な「編集(Edit)」ボタンを含めることができます（View Controllerの`editButtonItem`メソッド

を呼び出すことによって、このボタンを取得できます）。このボタンは、タップされると、自動的に「編集(Edit)」ボタンと「完了(Done)」ボタンの間で切り替わり、View Controllerの`setEditing:animated:`メソッドを適切な値で呼び出します。また、独自のコードからこのメソッドを呼び出して（または、View Controllerの`editing`プロパティの値を変更して）、モードを切り替えることもできます。

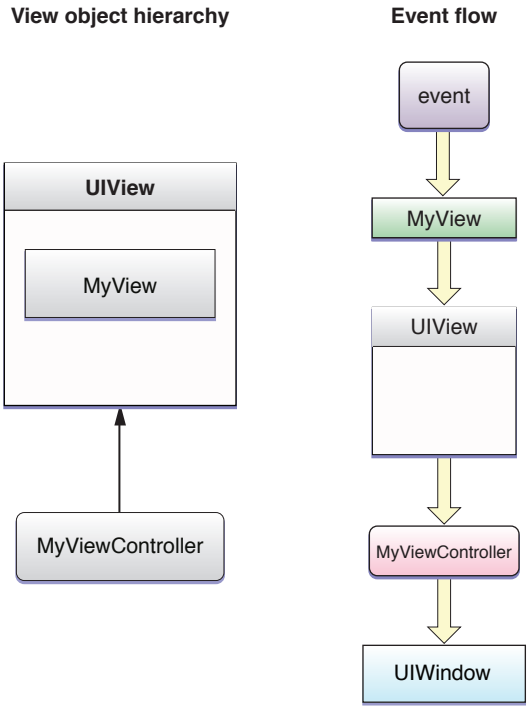
Navigation Barに「編集(Edit)」ボタンを追加する方法の詳細については、「「編集(Edit)」／「完了(Done)」ボタンの使用」（76 ページ）を参照してください。Table Viewの編集をサポートする方法の詳細については、『*Table View Programming Guide for iPhone OS*』を参照してください。

イベント処理

View Controller自身は、`UIResponder`クラスの下位クラスです。したがって、すべての種類のイベントを処理することができます。通常のビューは、自身が受け取ったイベントに応答しない場合、そのイベントをスーパービューに渡します。一方、View Controllerの管理下にあるビューは、イベントをまずView Controllerオブジェクトに渡します。これによって、View Controllerは、ビューで処理されないすべてのイベントを吸収します。View Controllerがイベントを処理しない場合は、通常どおり、そのイベントはそのビューのスーパービューに渡されます。

図 2-12に、ビュー階層内のイベントの流れを示します。この図では、画面と同じサイズの汎用のビューオブジェクトの中にカスタムビューが埋め込まれています。この汎用ビューはカスタムView Controllerで管理されています。このカスタムビューのフレームに届いたタッチイベントは、処理のためにそのビューに配信されます。そのビューがイベントを処理しない場合、そのイベントは親ビューに渡されます。この汎用ビューはイベントを処理しないため、これらのイベントは、まずView Controllerに渡されます。このView Controllerがイベントを処理しない場合は、そのイベントはさらに、汎用の`UIView`のスーパービュー（この場合は、ウインドウオブジェクト）に渡されます。

図 2-12 View Controllerのレスポンダチェーン



注： View Controllerとそのビューの間のメッセージ送信関係は、View Controllerによってプライベートに管理されているため、アプリケーションのプログラムで変更することはできません。

カスタムView Contorollerでわざわざタッチイベントを処理しようとは思わないかもしれませんが、それを使用して動きに基づくイベントを処理することもできます。また、カスタムView Controllerを使用して、ファーストレスポンダの設定や変更を行うこともできます。iPhoneアプリケーションでイベントを配信したり処理したりする方法の詳細については、「Event Handling」を参照してください。

関連するView Controllerオブジェクトへのアクセス

View Controllerの主な仕事はビュー階層の提供と管理ですが、View Controllerがほかの View Controllerと連携して、より複雑なインターフェイスを表示する場合もよくあります。たとえば、Navigation ControllerはNavibagion Bar用のビューを管理します。このビューには、戻るボタンや現在のView Controllerに固有の情報が表示されます。特定のコンテンツについては、Navigation Controllerは、カスタムView Controllerにそのコンテンツの提供を依頼します。同様に、Tab Bar Controllerは、カスタムView Controllerに、タブに関連する情報の提供を依頼します。

カスタムView Controllerには、上位レベルのView Controllerで必要な特定のオブジェクトを提供する役割を担います。表 2-3は、カスタムView Controllerで提供できるようにしておくべきオブジェクトのタイプと、そのオブジェクトの提供が必要となる状況のリストです。これらのプロパティにカスタムオブジェクトを指定しないと、View Controllerは適切なデフォルトのオブジェクトを提供します。

表 2-3 カスタムView Controllerによって管理されるオブジェクトのサポート

オブジェクト	プロパティ	説明
Navigation Toolbar Item	`toolbarItems`	カスタムツールバーを提供するNavigation Controllerと組み合わせて使用します。このプロパティを使用して、そのツールバーに表示する項目を指定できます。ユーザが、ある画面から次の画面に遷移すると、新しいView Controllerに対応するツールバー項目がアニメーションによって現れます。 デフォルトの動作では、カスタムツールバー項目は提供されません。 iPhone OS 3.0以降で利用できます。ナビゲーションツールバーの設定については、「ナビゲーションツールバーの表示」（76ページ）を参照してください。

オブジェクト	プロパティ	説明
Navigation Barのコンテンツ	`navigationItem`	Navigation Controllerと組み合わせて使用します。Navigation Itemオブジェクトは、View Controllerが表示されるときにNavigation Barに表示されるオブジェクトを提供します。このオブジェクトを使用して、ビューにカスタムタイトルや追加のコントロールを提供できます。 デフォルトのNavigation Itemは、Navigation Barのタイトルとして View Controllerのタイトルを使用します。カスタムボタンは提供しません。Navigation Controllerは、戻るボタンを自動的に追加します。 Navigation Barのコンテンツを指定する方法については、「Navigation Barの外観のカスタマイズ」（70 ページ）を参照してください。
Tab Bar Item	`tabBarItem`	Tab Bar Controllerと組み合わせて使用します。Tab Bar Itemは、View Controllerに関連付けられたタブに表示される画像やテキストを提供します。 デフォルトのTab Bar ItemにはView Controllerのタイトルが含まれています。画像は含まれていません。 View Controllerにタブのコンテンツを指定する方法については、「Tab Barインターフェイスの作成」（82 ページ）を参照してください。

これらのプロパティの詳細については、『*UIViewController Class Reference*』を参照してください。

Navigation Controller

アプリケーションで階層的なデータの表示を管理するには、Navigation Controllerを使用します。Navigation Controllerは、自己完結型のビュー階層（ナビゲーションインターフェイスと呼ばれる）を管理します。このビュー階層のコンテンツは、Navigation Controllerが直接管理するビューと、デベロッパが提供するカスタムView Controllerが管理するビューから構成されます。カスタムView Controllerは各画面に相当するデータを提供し、Navigation Controllerはこれらの画面間のナビゲーションを管理します。

ナビゲーションインターフェイスのほとんどはカスタムコンテンツで構成されますが、それでもコード内でNavigation Controllerオブジェクトと直接やり取りしなければならない部分は存在します。新しいビューを表示するタイミングをNavigation Controllerに伝えることのほかに、Navigation Bar（画面の上端にあるビューで、ナビゲーション階層におけるユーザの現在位置に関する情報を提供する）を設定することもデベロッパの責任です。デベロッパは、Navigation Controllerが管理するツールバーの項目を提供することもできます。

この章では、アプリケーションでNavigation Controllerを設定したり使用したりする方法の概要について説明します。Navigation Controllerをほかの種類のView Controllerオブジェクトと組み合わせて使用する方法については、「View Controllerインターフェイスの組み合わせ」（101 ページ）を参照してください。

ナビゲーションインターフェイスの構造

Navigation Controllerの主な仕事はカスタムView Controllerの表示を管理することですが、自身が所有するカスタムビューの表示も行います。具体的に言うと、前の画面に戻るためのボタンと、デベロッパがカスタマイズ可能ないくつかのボタンを含むNavigation Barを表示します。iPhone OS 3.0以降では、Navigation Controllerでナビゲーションツールバーのビューを表示したり、このツールバーにカスタムボタンを含めたりすることもできます。ただし、このツールバーの表示は任意です。

図 3-1に、ナビゲーションインターフェイスの主なビューを示します。この図のナビゲーションビューは、Navigation Controllerの`view`プロパティに格納されているビューです。このビューは、ウインドウに埋め込んだり、別のView Controllerを使用して表示します。このインターフェイス内のほかのすべてのビューは、本質的には、Navigation Controllerが管理する不透明なビュー階層に含まれます。

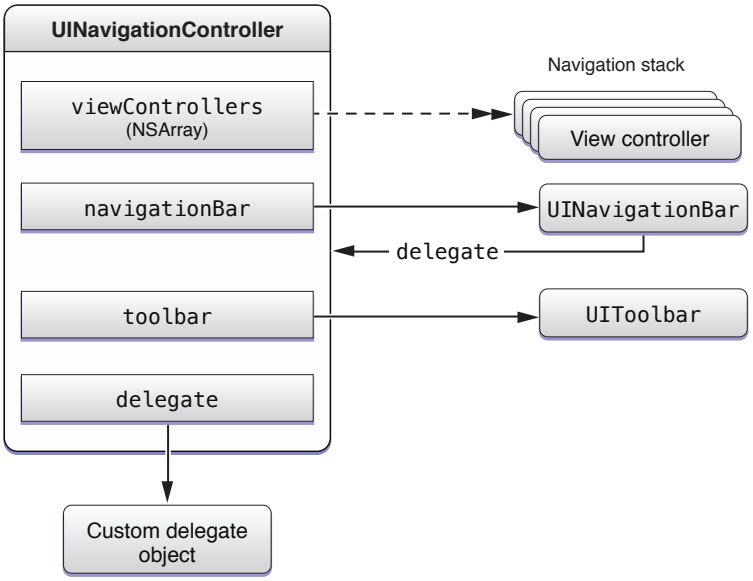
図 3-1 ナビゲーションインターフェイスのビュー

Navigation Barとツールバーはカスタマイズ可能なビューですが、ナビゲーション階層内のビューを直接変更してはいけません。これらのビューをカスタマイズする唯一の方法は、`UINavigationController`クラスと`UIViewController`クラスの特定のインターフェイスを利用する方法です。Navigation Barのコンテンツのカスタマイズ方法については、「Navigation Barの外観のカスタマイズ」（70 ページ）を参照してください。ナビゲーションインターフェイスでカスタムツールバー項目を表示したり設定したりする方法については、「ナビゲーションツールバーの表示」（76 ページ）を参照してください。

ナビゲーションインターフェイスのオブジェクト

Navigation Controllerは、いくつかのオブジェクトを使用してナビゲーションインターフェイスを実装します。これらのオブジェクトのいくつかはデベロッパが提供しなければなりませんが、それ以外はNavigation Controller自身が作成します。具体的に言うと、表示するカスタムコンテンツを含むView Controllerを提供するのはデベロッパの責任です。Navigation Controllerからの通知に応答するために、デリゲートオブジェクトを提供することもできます。Navigation Controllerは、ナビゲーションインターフェイスに使われる追加のビュー（Navigation Bar、ツールバーなど）を作成し、これらのビューを管理します。図 3-2に、Navigation Controllerと主要なオブジェクトとの関係を示します。

図 3-2 Navigation Controllerが管理するオブジェクト

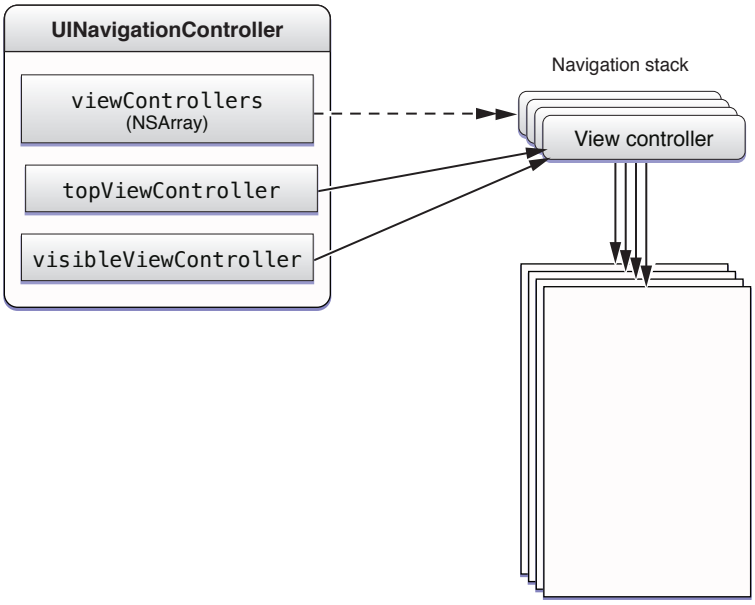


外観のいくつかの部分を変更する以外は、Navigation Controllerに関連付けられたNavigation Barオブジェクトやツールバーオブジェクトを変更するべきではありません。これらの設定や表示を担当するのは、Navigation Controllerだけです。さらに、Navigation Controllerオブジェクトは、自動的に自身を`UINavigationBar`のデリゲートとして割り当て、ほかのオブジェクトがその関係を変更できないようにします。

これらのオブジェクトの中でデベロッパが変更できるのは、デリゲートと、ナビゲーションスタック上のほかのView Controllerです。**ナビゲーションスタック**は、Navigation Controllerが管理する、後入れ先出し方式のカスタムView Contollerオブジェクトのコレクションです。このスタックに最初に追加された要素は**ルートView Controller**になり、決して削除できません。それ以外の要素は、`UINavigationController`クラスのメソッドを使用してスタックに追加できます。

図 3-3に、Navigation Controllerとナビゲーションスタック上のオブジェクトとの関係を示します。`topViewController`プロパティと`visibleViewController`プロパティ内のオブジェクトは同じである必要はないことに注意してください。スタックの一番上のオブジェクトからView Controllerをモーダルモードで表示すると、`topViewController`プロパティは変化しませんが、`visibleViewController`プロパティのオブジェクトは変化します。特に、`visibleViewController`プロパティの値は、表示されたモーダルView Controllerを表すように変化します。

図 3-3 ナビゲーションスタック



デベロッパの主な仕事は、ユーザのアクションに応答して、新しいView Controllerをスタックにプッシュすることです。ナビゲーションスタックにプッシュされた各View Controllerは、アプリケーションのデータの何らかの部分を表示する役割を担います。通常は、現在表示されているビューでユーザが項目を選択したときに、新しいView Controllerオブジェクトを作成し、選択された項目に対応するデータをそれに割り当てて、その新しいView Controllerをスタックにプッシュします。これが、選択されたデータをユーザに表示する方法です。たとえば、ユーザがフォトアルバムを選択すると、「写真(Photos)」アプリケーションは、そのアルバム内の写真を表示するView Controllerをプッシュします。ほとんどの場合、プログラムによってView Controllerをポップする必要はありません。代わりに、Navigation ControllerがNavigation Barに戻るボタンを表示します。それがタップされると、一番上のView Controllerが自動的にポップされます。

Navigation Barのカスタマイズ方法の詳細については、「Navigation Barの外観のカスタマイズ」（70 ページ）を参照してください。ナビゲーションスタックにView Controllerをプッシュしたり、後からそれを削除したりする方法の詳細については、「ナビゲーションスタックの変更」（67 ページ）を参照してください。ツールバーの内容のカスタマイズ方法の詳細については、「ナビゲーションツールバーの表示」（76 ページ）を参照してください。

ナビゲーションインターフェイスの作成

表示する情報が階層的に構成されている場合は、ナビゲーションインターフェイスを使用します。通常、Navigation Controllerは階層的なデータの表示を管理するために使用しますが、複数のレベルの編集を管理したり、連続する複数の画面が必要なその他のインターフェイスを管理するために使用することもできます。

ナビゲーションインターフェイスを作成する前に、その使いかたを決定する必要があります。アプリケーションにナビゲーションインターフェイスをインストールするには、主に次の3つのケースがあります。

- アプリケーションのメインウインドウに直接インストールする。

- Tab Barインターフェイスのタブのルートビューとしてインストールする。
- モーダルモードで表示する、または必要に応じて表示したり閉じたりできるスタンドアロンのView Controllerとして使用する。

最初の2つのシナリオでは、Navigation Controllerは基本的なインターフェイスの極めて重要な部分を提供し、アプリケーションが終了するまで存続します。一方、最後のシナリオはNavigation Controllerの一時的な使用を表しています。この場合、Navigation Controllerを使用するための手順は、ほかのモーダルView ControllerやカスタムView Controllerのための手順と同じです。唯一の違いは、Navigation Controllerが1つ以上の画面に相当するコンテンツを提供する点です。以降の各セクションでは、より永続的なタイプのナビゲーションインターフェイスを作成する方法に焦点を当てますが、カスタマイズの手順と一般的な情報のほとんどは、Navigation Controllerの使いかたに関わらず、すべてのNavigation Controllerに当てはまります。

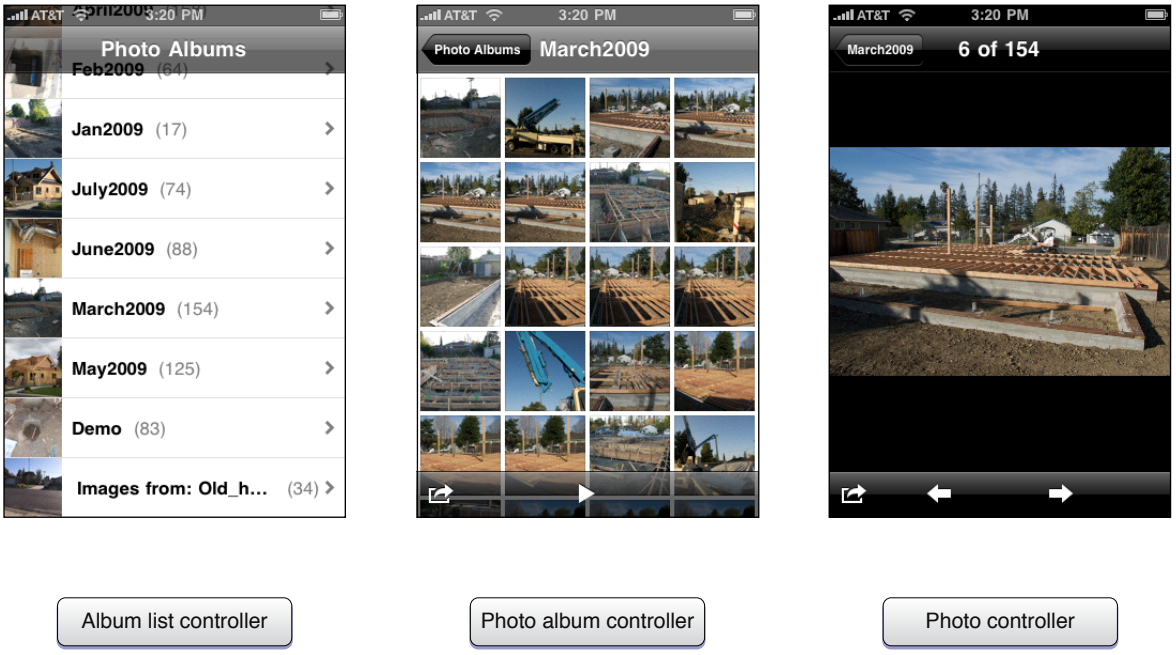
注： Navigation Controllerをモーダルモードで表示する方法に特化した例については、「モーダルモードでのNavigation Controllerの表示」（106 ページ）を参照してください。

ナビゲーションインターフェイス用のカスタムView Controllerの定義

ナビゲーションインターフェイスを実装するためにしなければならない重要な点の1つは、各ステージで表示するデータを決定することです。すべてのナビゲーションインターフェイスは、ルートレベルのデータを表すデータレベルを少なくとも1つ持っています。これが、インターフェイスの開始点になります。たとえば、「写真(Photos)」アプリケーションでは、データ階層のルートレベルに、利用可能なフォトアルバムのリストを表示します。フォトアルバムを選択するとアルバム内の写真が一覧表示され、さらに写真を選択するとより大きなバージョンの写真が表示されます。

データ階層のレベルごとに、そのレベルのデータを管理したり表示したりするカスタムView Controllerオブジェクトを1つ提供する必要があります。複数のレベルでの表示が本質的に同じ場合は、同じView Controllerクラスを再利用することも可能です。ただし、実行時には、そのクラスの別々のインスタンスを作成して、それぞれ固有のデータセットを管理するように設定する必要があります。たとえば、「写真(Photos)」アプリケーションには3つの異なる表示タイプがあります。したがって、図 3-4に示すような3つの別個のView Controllerオブジェクトが必要になります。

図 3-4 データのレベルごとにView Controllerを定義する



リーフデータを管理するView Controllerを除いて、各カスタムView Controllerは、データ階層の次のレベルに移動するための手段をユーザに提供する必要があります。項目のリストを表示するView Controllerは、特定のテーブルセル内でのタップを使用して、次のレベルのデータを表示できます。たとえば、ユーザが最上位レベルのリストからフォトアルバムを選択した場合、「写真(Photos)」アプリケーションは新規にフォトアルバム用View Controllerを作成します。この新規のView Controllerは、関連する写真を表示するのに十分なアルバム情報で初期化されます。

アプリケーションが、あるレベルから次のレベルへ既知の順番で移動するような高度に構造化されたデータを持っている場合、各レベルのView Controller（およびそれらの関係）を定義することは比較的簡単です（「写真(Photos)」アプリケーションの場合は、必ずアルバムのリスト、1つのアルバムのコンテンツ、1つの写真の順番で表示されます）。一方、現在のレベルで選択されている項目によって異なる方法でデータを表示するアプリケーションの場合は、そのデータの表示方法を決定するのにデータモデル内の追加情報を使用できます。その情報を使用して、別のView Controllerクラスを使用するか、1つのView Controllerクラスの表示スタイルをカスタマイズするかを決定します。たとえば、「iPod」アプリケーションで「作曲者(Composer)」別に曲を表示すると、その作曲者の曲が同じアルバムのものか、複数の別々のアルバムのものかによって、アルバムのリスト、または曲のリストが表示されます。ただし、どちらの場合もデータはテーブルを使用してリスト形式で表示されるため、1つのView Controllerを用意し、データのタイプごとに異なるテーブルセルを提供するだけで済みます。

カスタムView Controllerに関する一般的な情報およびガイダンスについては、「カスタムView Controller」（23 ページ）を参照してください。

nibファイルからのナビゲーションインターフェイスのロード

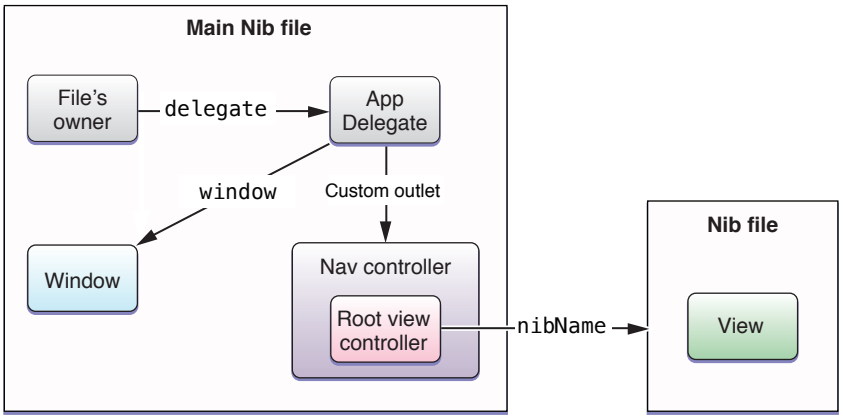
Interface Builderには、Navigation Controllerオブジェクトをnibファイルに含める機能がありますが、そのセマンティクスは、カスタムView Controllerの場合とは少し異なります。カスタムView Controllerでは、View Controllerに関連付けられているビューをnibファイルに格納しますが、一般に、View

Controllerそのものは、プログラムの中で作成する場合でも、ほかのnibファイルからロードする場合でも、別個に作成します。一方、Navigation Controllerでは、ビューは必ずプログラムの中で作成するため、独立のnibファイルに格納するビューは存在しません。この結果、Navigation Controllerがnibファイルを管理することはありません。つまり、Navigation ControllerをFile's Ownerプレースホルダに割り当てることはありません。代わりに、Navigation Controllerとnibファイルが関わりを持つのは、Navigation Controllerそのものがnibファイルに保存される場合だけです。このような理由から、Navigation Controllerはほとんどの場合、ほかのオブジェクトが管理するnibファイルに保存されます。

Navigation Controllerをアプリケーションのメインnibファイルに含めることが最も理にかなっています。Navigation Controller自身がアプリケーションのウインドウのメインビューを提供する場合や、Navigation ControllerがTab Barインターフェイスのルートビューを提供する場合は、このようにします。スタンドアロンまたはモーダルモードで表示されるNavigation Controllerをメインnibファイル（またはその他のnibファイル）からロードすることもできますが、最適ではありません。そのような場合、通常、Navigation Controllerを必要になった時点でプログラムの中で作成する方が簡単です。

図 3-5に、メインウインドウにナビゲーションインターフェイスを表示するアプリケーションのメインnibファイルの構成を示します。このnibファイルにはウインドウオブジェクト、Navigation Controller、このナビゲーションインターフェイスのルートView Controllerが含まれています。このルートView ControllerはNavigation Controllerそのものに含まれています（ルートView Controllerが管理するビューは、通常、同じnibファイルに含める必要はありません。この例でも、別のnibファイル内にあります）。

図 3-5　ナビゲーションインターフェイスを含むnibファイル



Navigation Controllerはプログラムの中でビューを作成するため、Interface Builderを使用してそのビューをウインドウにインストールすることはできません。代わりに、プログラムの中でビューをウインドウに追加しなければなりません。ウインドウとNavigation Controllerが1つのnibファイル内にある場合は、これらのオブジェクトへのポインタを必ず保存して、後からそれにアクセスできるようにする必要があります。

Navigation Controllerをnibファイル内に設定するには、次の手順を実行します。

1. ライブラリからNavigation Controllerオブジェクトを Interface Builderのドキュメントウインドウにドラッグします。

Navigation Controllerを追加すると、Interface Builderによって、Navigation Bar、Navigation Controllerに含まれるRoot View Controllerオブジェクト、およびRoot View Controller用のNavigation Itemが追加されます。これらのオブジェクトには、Navigation Controllerの編集サーフェスでオブジェクトを選択してアクセスできます。アウトラインモードやブラウザモードのときには、ドキュメントウインドウから任意のオブジェクトを選択することもできます。

2. アウトレットを使用してNavigation Controllerへの参照を保存します。

   実行時にこのNavigation Controllerにアクセスするには、アウトレットを使用するか、nibファイルをロードするときにそのnibファイルの最上位レベルのオブジェクトを明示的に取得する必要があります。一般には、アウトレットを使用する方がはるかに簡単です。アウトレットを追加するには、次のようにアプリケーションデリゲートクラスの宣言に変数を追加します。

   ```
   @interface MyAppDelegate :NSObject <UIApplicationDelegate> {
      IBOutlet UINavigationController*  myNavigationController;
   }
   @end
   ```

   アウトレットの定義を追加したら、Interface Builderで、そのアウトレットからNavigation Controllerへの接続を作成します。

3. ルートView Contrllerのクラスを、Xcodeプロジェクトのカスタムクラスに設定します。

   ここで選択するクラスは、ナビゲーション階層の最上位レベルのデータを表示するクラスでなければなりません。複数のレベルのデータがある場合は、View Controllerクラスが表示するビューに、次のレベルのデータに移動するためのコントロールを含める必要があります。View Controllerの設計方法については、「ナビゲーションインターフェイス用のカスタムView Controllerの定義」（61 ページ）を参照してください。

4. ルートView Controller用のビューを作成します。

   このビューは同じnibファイルに含めることもできますし、別のnibファイルに保存することもできます。メモリ不足状態のときにビューをメモリから削除できるようにするために、別のnibファイルを使用することをお勧めします。

   別のnibファイルを指定するには、ルートView ControllerのNIB Name属性をそのnibファイルの名前に設定します。ビューをNavigation Controllerと同じnibファイルに含めるには、ビューオブジェクトをNavigation Controllerの編集サーフェスにドラッグします。

5. アプリケーションデリゲートの`applicationDidFinishLaunching:`メソッド内で、Navigation Controllerのビューをメインウインドウに追加します。

   Navigation Controllerが、自身を自動的にアプリケーションのウインドウにインストールすることはありません。次のようなコードによって、プログラムでインストールする必要があります。

   ```
   - (void)applicationDidFinishLaunching:(UIApplication *)application {
       [window addSubview:myNavigationController.view];
   }
   ```

6. nibファイルを保存します。

ルートのView Controllerオブジェクトを設定することに加えて、インスペクタを使用して、Interface Builderで作成したNavigation ItemやNavigation Barの属性を変更することもできます。Navigation Barについては、基本的なスタイル属性を変更できます。Navigation Itemについては、カスタムタイトルとプロンプトテキストを指定できます。Navigation Itemにカスタムの文字列を指定しない場合、Interface BuilderはカスタムView Controllerのtitleプロパティの文字列を使用します。

Navigation Controllerの編集サーフェイスには、Navigation Barの設定を編集するための便利な手段があります。Navigation Barにボタンやビューを追加するには、適切なオブジェクトをライブラリから編集サーフェイス内のNavigation Barにドラッグします。これによって、対応するオブジェクトがルートView ControllerのNavigation Itemに追加されます。これで、ほかのInterface Builderオブジェクトと同様に、いまドラッグした項目を設定できます。たとえば、実行時のユーザのタップを処理できるようにNavigation ItemのボタンをView Controllerのアクションメソッドに接続できます。

Interface Builderを使用してnibファイルの内容を設定する方法（ビューやその他のコントロールをアクションメソッドに接続する方法も含む）については、『*Interface Builder User Guide*』を参照してください。

プログラムによるナビゲーションインターフェイスの作成

プログラムの中でNavigation Controllerを作成するには、コードの適切な位置でそれを行います。たとえば、Navigation Controllerがアプリケーションウインドウのルートビューを提供する場合は、アプリケーションデリゲートの`applicationDidFinishLaunching:`メソッド内でNavigation Controllerを作成します。

それ以外にも、プログラムの中でView Controllerを作成する方が理にかなっている場合があります。たとえば、ナビゲーションインターフェイスをモーダルモードで表示する場合は、必要になった時点でNavigation Controllerオブジェクトを作成して表示し、不要になった時点で解放する方が通常は簡単です。このようなView Controllerをnibファイルからロードすると、nibファイルをロードするために余分なオーバーヘッドがかかり、不要になった後もNavigation Controllerオブジェクトへのポインタを保存しておく必要があります。

Navigation Controllerを作成するには、次の手順を実行する必要があります。

1. ナビゲーションインターフェイス用のルートView Controllerを作成します。

   このオブジェクトは、ナビゲーションスタックの最上位レベルのView Controllerになります。そのビューが表示されるときは、Navigation Barに戻るボタンは表示されません。また、このView Controllerをナビゲーションスタックからポップすることはできません。

2. Navigation Controllerを作成し、`initWithRootViewController:`メソッドを使用して初期化します。

3. Navigation Controllerのビューをウインドウに追加します（または、インターフェイスに表示します）。

Navigation Controllerを作成したら、必要に応じて使用できます。たとえば、そのビューをウインドウに追加したり、そのビューを使用して別のView Controllerを初期化したり、ビューをモーダルモードで表示したりできます。

リスト3-1に、Navigation Controllerを作成してそれをアプリケーションのメインウインドウに追加する`applicationDidFinishLaunching:`メソッドの簡単な実装を示します。`navigationController`変数と`window`変数はアプリケーションデリゲートのメンバ変数です。また、`MyRootViewController`

クラスはカスタムView Controllerクラスです。この例のウインドウが表示されると、ナビゲーションインターフェイスは、このナビゲーションインターフェイスのルートView Controller用のビューを表示します。

リスト 3-1　プログラム内でのNavigation Controllerの作成

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
{
    UIViewController *rootController = [[MyRootViewController alloc] init];
    navigationController = [[UINavigationController alloc]
                                initWithRootViewController:rootController];
    [rootController release];

    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    [window addSubview:navigationController.view];
    [window makeKeyAndVisible];
}
```

ナビゲーションビューでのフルスクリーンレイアウトの採用

通常、ナビゲーションインターフェイスは、Navigation Barの下端と、ツールバーまたはTab Barの上端との間にカスタムコンテンツを表示します（図 3-1 （58 ページ）を参照）。ただし、View Controllerでは、代わりにそのビューをフルスクリーンレイアウトで表示するように指定できます。フルスクリーンレイアウトでは、コンテンツビューはNavigation Bar、ステータスバー、およびツールバーの背後に重なるように設定されます。これによって、ユーザ向けのコンテンツの表示領域を最大にできるため、写真の表示やスペースを広くとりたい場合に役立ちます。

ビューを、画面全体か、またはほぼ全体のサイズまで広げるかどうかを判断する場合、Navigation Controllerは次のような点を考慮します。

- ベースとなるウインドウ（親ビュー）のサイズが画面の境界一杯まで広がっているか？
- Navigation Barが半透明に設定されているか？
- ナビゲーションツールバー（使われている場合）が半透明に設定されているか？
- ベースとなるView Controllerの`wantsFullScreenLayout`プロパティが`YES`に設定されているか？

これらの各要素は、カスタムビューの最終的なサイズを決定するために使われます。上記のリストの順序は、各要素が考慮される優先順位を表しています。ウインドウサイズは、最優先の制限要素です。アプリケーションのメインウインドウ（モーダルモードで表示されたView Controllerの場合は、それを含む親ビュー）が画面一杯に広がっていない場合は、それに含まれるビューも画面一杯に広げることはできません。同様に、Navigation Barやツールバーが表示されていても半透明でない場合は、View Controllerがビューをフルスクリーンレイアウトで表示しようとしても無効です。Navigation Controllerは、不透明なNavigation Barの背後にコンテンツを表示することはありません。

注： Tab Barのビューは半透明をサポートしないので、Tab Bar Controllerがそれに対応するTab Barの下にコンテンツを表示することはありません。したがって、ナビゲーションインターフェイスがTab Bar Controllerのタブに埋め込まれている場合は、Navigation Barの背後にはコンテンツを重ねることができますが、Tab Barの背後に重ねることはできません。

ナビゲーションインターフェイスを作成していて、カスタムコンテンツを画面全体またはほぼ全体に広げたい場合は、次の手順を実行します。

1. カスタムビューのフレームを画面の境界一杯に設定します。

   ビューの自動サイズ変更属性も忘れずに設定します。自動サイズ変更属性を設定すると、ビューのサイズ変更が必要な場合に、それに応じてコンテンツが調整されます。あるいは、ビューのサイズが変更されたら、そのビューの`setNeedsLayout`メソッドを呼び出してサブビューの位置の調整が必要であることを表明できます。

2. Navigation Controllerの`translucent`プロパティを`YES`に設定します。これによって、コンテンツをNavigation Barの背後に重ねることができます。

3. ステータスバーの背後に重ねるには、View Controllerの`wantsFullScreenLayout`プロパティを`YES`に設定します（この属性が認識されるようにするには、Navigation Barを半透明にしなければなりません）。

4. オプションのツールバーの背後に重ねるには、ツールバーの`translucent`プロパティを`YES`に設定します。

ナビゲーションインターフェイスを表示するときは、ナビゲーションビューの追加先となるウインドウまたはビューも適切なサイズに設定する必要があります。アプリケーションが主たるインターフェイスとしてNavigation Controllerを使用する場合は、メインウインドウのサイズを画面のサイズに一致させる必要があります。つまり、メインウインドウのサイズを、（`applicationFrame`プロパティではなく）`UIScreen`クラスの`bounds`プロパティに一致させなければなりません。実際には、ナビゲーションインターフェイスについては、どのような状況でもフルスクリーンの境界に合わせてウインドウを作成する方が優れています。Navigation Controllerは、いずれにしてもそのビューのサイズをステータスバーに合わせて自動的に調節するからです。

Navigation Controllerをモーダルモードで表示している場合、そのNavigation Controllerが表示するコンテンツは、Navigation Controllerの表示を実行しているView Controllerによって制限されます。そのView Controllerがステータスバーの背後に重なるようになっていない場合は、モーダルモードで表示されたNavigation Controllerもステータスバーの背後に重ねることはできません。つまり、親のビューは、モーダルモードで表示されたビューの表示の仕方に必ず何らかの影響を与えます。

フルスクリーンレイアウトをサポートするインターフェイスを設定する方法の詳細については、「カスタムビューでのフルスクリーンレイアウトの採用」（51 ページ）を参照してください。

ナビゲーションスタックの変更

Navigation Controllerはナビゲーションスタックを管理しますが、そのスタックに置くオブジェクトを作成するのはデベロッパの責任です。Navigation Controllerオブジェクトを初期化するときには、データ階層のルートコンテンツを表示するためのカスタムView Controllerを提供する必要があります。このルートView Controllerは、必ずナビゲーションスタックの一番下に置かれ、削除することは

できません。その後は、プログラムの中で、あるいはユーザのやり取りに応じて、ほかのView Controllerを追加したり削除したりできます。これらのView Controllerを追加したり削除したりするには、`UINavigationController`クラスのメソッドを使用します。

Navigation Controllerでは、ナビゲーションスタックのコンテンツを管理する際にはいくつかの選択肢があります。これらの選択肢は、アプリケーションでよく見られるさまざまなシナリオをカバーしています。表 3-1に、これらのシナリオとそれへの対応の仕方を示します。

表 3-1　ナビゲーションスタックを管理する際の選択肢

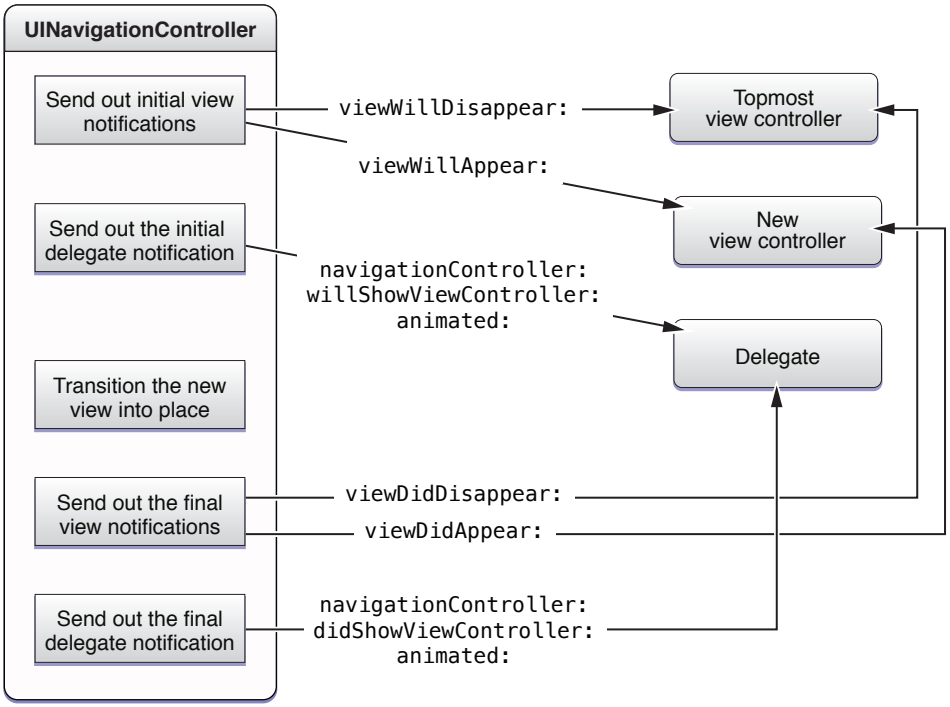
シナリオ	説明
階層データの次のレベルを表示する。	最上位のView Controllerに表示された項目をユーザが選択したときには、`pushViewController: animated:`メソッドを使用して新規のView Controllerをナビゲーションスタックにプッシュします。この新規のView Controllerが、選択された項目のコンテンツの表示を担当します。
階層内を1レベル戻る。	通常、Navigation Controllerには、スタックから一番上のView Controllerを削除して前の画面に戻るための戻るボタンがあります。プログラムの中で`popViewControllerAnimated:`メソッドを使用して、一番上のView Controllerを削除することもできます。
ナビゲーションスタックを前の状態に復元する。	アプリケーションを起動するときには、`setViewControllers: animated:`メソッドを使用してNavigation Controllerを前の状態に復元できます。たとえば、ユーザが前回アプリケーションを終了したときと同じ画面に戻すには、このメソッドを使用します。 アプリケーションを前の状態に復元するには、まず、必要なView Controllerを再作成するために十分な状態情報を保存する必要があります。ユーザがアプリケーションを終了するときに、データ階層内でのユーザの位置を示すいくつかのマーカーやその他の情報を保存する必要があります。次回起動したときには、この状態情報を読み込んで、それを使用して、`setViewControllers: animated:`メソッドを呼び出す前に、必要なView Controllerを再作成します。
ユーザをルートView Controllerに戻す。	ナビゲーションインターフェイスのトップに戻るには、`popToRootViewControllerAnimated:`メソッドを使用します。このメソッドは、ルートView Controller以外のすべてのビューをナビゲーションスタックから削除します。
階層内を任意の数のレベルだけ戻る。	一度に複数レベル戻るには、`popToViewController: animated:`を使用します。このメソッドは、Navigation Controllerを使用して、（モーダルモードでコンテンツを表示するのではなく）カスタムコンテンツの編集を管理する場合に使用できます。複数の編集画面をプッシュした後で、ユーザが操作をキャンセルした場合、このメソッドを使用して、1つずつではなく、一度にすべての編集画面を削除できます。
データ階層内の任意の場所にジャンプする。	`setViewControllers: animated:`を使用すると、データ階層内の任意の場所にシームレスにジャンプできます。データ階層内をジャンプして移動することは、（混乱を招くおそれがあるため）あまり良い考えではありませんが、このメソッドを使用してルートView Controllerに戻るまでの流れを示すのは最良の方法です。

View Controllerのプッシュやポップをアニメーション化する場合は、Navigation Controllerが自動的に最も分かりやすいアニメーションを作成します。たとえば、`popToViewController:animated:`メソッドを使用して、複数のView Controllerをスタックからポップした場合、Navigation Controllerは一番上のView Controllerに対してだけアニメーションを使用します。中間にあるその他のView Controllerは、すべてアニメーションなしで削除されます。

ナビゲーションスタックの変化の監視

ナビゲーションスタックが変化すると、Navigation Controllerはデリゲートに適切なメッセージを送信します。つまり、View Controllerがプッシュまたはポップされるたびに、Navigation Controllerは影響を受けるView Controllerにメッセージを送信します。図 3-6に、プッシュ操作またはポップ操作の間に発生するイベントの順序と、イベントに対応して各ステージでカスタムオブジェクトに送信されるメッセージを示します。新規のView Controllerは、スタックの一番上のView ControllerになろうとしているView Controllerを表します。

図 3-6 スタックが変化する間に送信されるメッセージ



Navigation Controllerのデリゲートのメソッドを使用して、ナビゲーションインターフェイスの状態を更新したり、アプリケーションのデータモデルを変更したりできます。たとえば、コンテンツの編集をサポートするためにナビゲーションインターフェイスを使用している場合は、`navigationController:willShowViewController:animated:`メソッドを使用して、対応するView Controllerが消去される前に変更内容を保存できます。ただし、ビューやビュー階層に変更を加えるために、これらのメソッドを使用することはしません。この種の変更は、カスタムView Controllerで処理するべきです。

Navigation Barの外観のカスタマイズ

Navigation Barは、ナビゲーションインターフェイスでよく使われるコントロールを管理するためのコンテナビューです。これは本質的には単なるビューオブジェクトですが、Navigation Barは、対応するNavigation Controllerオブジェクトが管理しているときは特殊な役割を果たします。整合性を保証したり、ナビゲーションインターフェイスを作成するために必要な作業量を削減するために、Navigation Controllerオブジェクトはそれぞれ固有のNavigation Barを作成し、そのバーのコンテンツ管理のほとんどの仕事を担当します。必要に応じて、Navigation Controllerは、この仕事に役立てるために、ほかのオブジェクト（カスタムView Controllerなど）とやり取りをします。

Navigation Barの管理はNavigation Controllerの仕事であるため、Navigation Barそのものを直接変更することは、たいていは禁止されています。それでも、独自のニーズに合うようにNavigation Barをカスタマイズする方法はいくつか存在します。以降の各セクションでは、Navigation Barの構造とアプリケーション用にそれらをカスタマイズする方法について説明します。

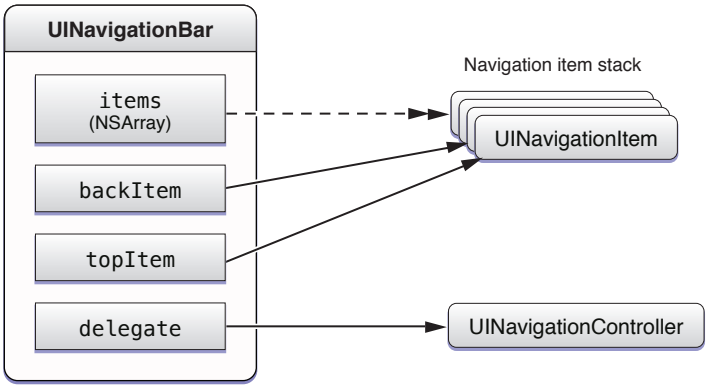
注： 以降の各セクションでは、Navigation BarをNavigation Controllerと一緒に使用する方法に焦点を当てていますが、Navigation Barをスタンドアロンのビューとして作成して使用することもできます。`UINavigationBar`クラスのメソッドとプロパティの使用方法の詳細については、*UINavigationBar Class Reference*を参照してください。

Navigation Itemオブジェクトの設定

Navigation Barの構造は、多くの点でNavigation Controllerの構造に似ています。Navigation Controllerと同様に、Navigation Barはほかのオブジェクトが提供するコンテンツのためのコンテナです。Navigation Barの場合、そのコンテンツは1つ以上の`UINavigationItem`オブジェクトによって提供されます。このオブジェクトは**Navigation Itemスタック**と呼ばれるスタックデータ構造を使用して格納されます。各Navigation Itemは、Navigation Barに表示するビュー一式とコンテンツを提供します。Navigation Controllerと異なり、Navigation Barは、ほかのビュー内に埋め込むことができる実際のビューオブジェクトです。

図3-7に、実行時にNavigation Barに関連する主要なオブジェクトをいくつか示します。Navigation Barの所有者は（それがNavigation Controllerかカスタムコードかに関わらず）、必要に応じてNavigation Itemをスタックにプッシュしたり、スタックからポップしたりする仕事を担当します。適切なナビゲーションを提供するために、Navigation Itemは、スタック内のオブジェクトを選択するためのポインタを保持しています。ほとんどのコンテンツは一番上のNavigation Itemから取得しますが、（前の項目のタイトルを持つ）戻るボタンを作成できるように、前の項目へのポインタも保持しています。

図 3-7 Navigation Barに関連するオブジェクト



重要： Navigation Controllerと組み合わせて使用する場合は、必ずNavigation Barのデリゲートは、それを所有するNavigation Controllerオブジェクトに設定されます。このデリゲートを変更しようとすると、例外が発生します。

ナビゲーションインターフェイスで使用する場合は、Navigation Barのスタックのコンテンツは、必ず親のNavigation Controllerのスタックのコンテンツに対応しています。つまり、ナビゲーションスタック内の各View Controllerに対して、それに対応するNavigation Itemが、Navigation BarのNavigation Itemスタック内の同じ位置に存在します。このように1対1に対応している理由は、各View Controllerが固有のNavigation Itemを実際に提供するからです。

Navigation Barには、項目を配置するために左、右、中央の3つの位置があります。表 3-2に、この位置を設定するために使われる`UINavigationItem`クラスのプロパティのリストを示します。Navigation Controllerと一緒に使用するためにNavigation Itemを作成した場合は、想定されているコントロールの表示を優先するために、どこかの位置のカスタムコントロールが無視されることがあります。各位置の説明には、カスタムオブジェクトの使われかたについての情報も含まれています。

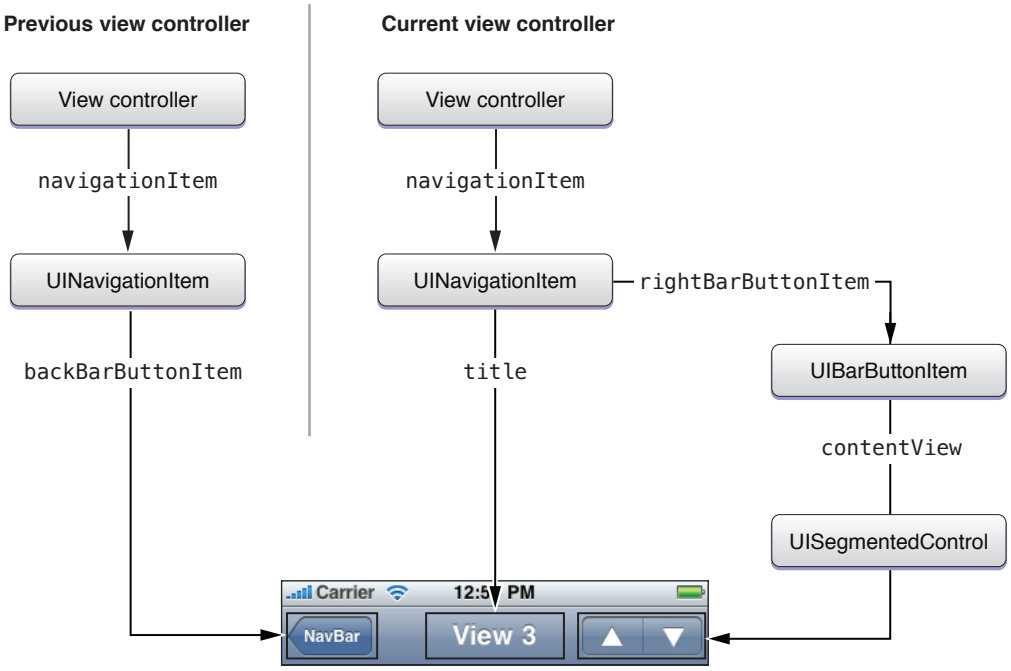
表 3-2 Navigation Bar上の項目の位置

位置	プロパティ	説明
左	`backBarButtonItem` `leftBarButtonItem`	ナビゲーションインターフェイスでは、Navigation Controllerはデフォルトで「戻る」ボタンを左の位置に割り当てます。Navigation Controllerが提供するデフォルトの「戻る」ボタンを取得するには、`backBarButtonItem`プロパティの値を取得します。 左の位置にカスタムボタンやビューを割り当て、それらでデフォルトの「戻る」ボタンを置きかえるには、`leftBarButtonItem`プロパティに`UIBarButtonItem`オブジェクトを割り当てます。

位置	プロパティ	説明
中央	`titleView`	ナビゲーションインターフェイスでは、Navigation Controllerはデフォルトで View Controllerのタイトルを持つカスタムビューを表示します。必要であれば、このビューを独自に選択したカスタムビューに置きかえることができます。 カスタムタイトルのビューを提供しないと、Navigation Barは適切なタイトル文字列を持つカスタムビューを表示します。タイトル文字列は、デフォルトではNavigation Itemから取得されます。Navigation Itemが適切なタイトルを提供しない場合は、View Controllerから取得されます。
右	`rightBarButtonItem`	デフォルトではこの位置は空いています。この位置は、通常、現在の画面を編集したり変更したりするためのボタンを配置するために使われます。ビューを`UIBarButtonItem`オブジェクトでラップすることによって、ここにカスタムビューを配置することもできます。

図 3-8に、Navigation Barのコンテンツがナビゲーションインターフェイスを構成する仕組みを示します。現在のView Controllerに対応するNavigation Itemは、Navigation Barの中央と右の位置にコンテンツを提供しています。前のView Controllerに対応するNavigation Itemは、左の位置にコンテンツを提供しています。左と右の項目には`UIBarButtonItem`オブジェクトを指定する必要がありますが、この図に示すようにBar Button Item内のビューをラップすることもできます。カスタムタイトルのビューを提供しない場合は、Navigation Itemは、現在のView Controllerのタイトルを使用してそのタイトルを作成します。

図 3-8 Navigation Barの構造



Navigation Barの表示と非表示

Navigation Controllerと組み合わせて使用するNavigation Barを表示したり非表示にするには、必ず`UINavigationController`の`setNavigationBarHidden:animated:`メソッドを使用します。`UINavigationBar`オブジェクトの`hidden`プロパティを直接変更して、Navigation Barを非表示にしてはいけません。Navigation Barの表示や非表示のほかにも、Navigation Controllerのメソッドを使用すると、さらに複雑な動作を自動的に実現できます。具体的に言うと、View Controllerの`viewWillAppear:`メソッド内でNavigation Barの表示や非表示を行うと、Navigation Controllerは、新規のView Controllerの出現と連動させてNavigation Barの出現や消去をアニメーション化します。

ユーザは前の画面に戻るためにNavigation Bar上の戻るボタンを必要とします。このため、前の画面に戻るための何らかの手段をユーザに提供せずに、Navigation Barを非表示にするべきではありません。ナビゲーションをサポートするための最も一般的な方法は、タッチイベントをインターセプトし、それを使用してNavigation Barの表示と非表示を切り替える方法です。たとえば、「写真(Photos)」アプリケーションでは、1つの画像をフルスクリーンで表示するときにこのような方法をとっています。また、スワイプジェスチャを検出し、それを使用して現在のView Controllerをスタックからポップすることもできます。ただし、このようなジェスチャは、Navigation Barの表示を単純に切り替える操作よりも検出が難しいジェスチャです。

Navigation Barオブジェクトを直接変更する

ナビゲーションインターフェイスでは、Navigation Controllerは自身の`UINavigationBar`オブジェクトを所有してその管理を担当します。Navigation Barオブジェクトを変更したり、その境界、フレーム、またはアルファの値を直接変更することは許されていません。ただし、変更が許されているプロパティがいくつかあります。その中には、次のプロパティが含まれています。

- `barStyle`プロパティ
- `translucent`プロパティ
- `tintColor`プロパティ

図 3-9に、`barStyle`プロパティと`translucent`プロパティがNavigation Barの外観に与える影響を示します。半透明スタイルの場合は、ベースになるView Controllerのメインビューがスクロールビューならば、Navigation Barの背後からコンテンツをスクロールして出せるように、Navigation Barが自動的にコンテンツ挿入枠の値を調節することは注目に値します。その他のタイプのビューの場合は、このような調節は行われません。

図 3-9 Navigation Barのスタイル

Navigation Bar全体を表示または非表示にするには、同様に、Navigation Barを直接変更するのではなく、Navigation Controllerの`setNavigationBarHidden:animated:`を使用します。Navigation Barの表示と非表示の詳細については、「Navigation Barの表示と非表示」（73 ページ）を参照してください。

Navigation Itemとしてカスタムボタンやカスタムビューを使用する

特定のView Controller用にNavigation Barの外観をカスタマイズするには、それに対応する`UINavigationItem`オブジェクトの属性を変更します。View Controller用のNavigation Itemは、`navigationItem`プロパティから取得できます。View Controllerは、要求されるまではNavigation Itemを作成しません。そのためナビゲーションインターフェイスにView Controllerをインストールする予定がある場合に限り、このオブジェクトを要求するべきです。

View Controller用のNavigation Itemを変更しなくても、Navigation Itemは、ほとんどの場合に十分対応できるデフォルトのオブジェクトセットを提供します。もちろん、カスタマイズを行えばそれらはすべてデフォルトのオブジェクトよりも優先されます。

最上位のView Controllerに関して、Navigation Barの左側に表示される項目は次の規則に従って決定されます。

- 最上位のView ControllerのNavigation Itemの`leftBarButtonItem`プロパティにカスタムBar Button Itemを割り当てた場合は、その項目の優先度が最も高くなります。
- カスタムBar Button Itemを割り当てていない場合は、ナビゲーションスタックの1レベル下のView ControllerのNavigation Itemが`backBarButtonItem`プロパティに有効な項目をもっていれば、Navigation Barはその項目を表示します。
- どちらのView ControllerにもBar Button Itemが指定されていない場合は、デフォルトの戻るボタンが使用され、そのタイトルには、前のView Controller（ナビゲーションスタックの1レベル下のView Controller）のtitleプロパティの値が設定されます（最上位のView ControllerがルートView Controllerの場合は、デフォルトの戻るボタンは表示されません）。

最上位のView Controllerに関して、Navigation Barの中央に表示される項目は次の規則に従って決定されます。

- 最上位のView ControllerのNavigation Itemの`titleView`プロパティにカスタムビューを割り当てた場合、Navigation Barはそのビューを表示します。
- カスタムタイトルビューが設定されていない場合、Navigation BarはそのView Controllerのタイトルを持つカスタムビューを表示します。このビューの文字列は、View ControllerのNavigation Itemの`title`プロパティから取得します。このプロパティの値が`nil`の場合は、このView Controller自身の`title`プロパティの文字列を使います。

最上位のView Controllerに関して、Navigation Barの右側に表示される項目は次の規則に従って決定されます。

- 新たに最上位になったView Controllerがカスタムの右Bar Button Itemを持つ場合は、その項目を表示します。カスタムの右Bar Button Itemを指定するには、Navigation Itemの`rightBarButtonItem`プロパティを設定します。
- カスタムの右Bar Button Itemが指定されていない場合、Navigation Barはバーの右側に何も表示しません。

Navigation Barコントロールの上に何らかのカスタムプロンプトテキストを追加するには、Navigation Itemの`prompt`プロパティに値を設定します。

図 3-10に、Navigation Barのさまざまな構成を示します。この中には、カスタムビューやプロンプトを使用したものも含まれています。この図のNavigation Barは、*NavBar*サンプルプロジェクトから引用しています。

図 3-10 Navigation Barのカスタムボタン

リスト 3-2に、図 3-10の3番目のNavigation Bar（カスタムビューを持つ右Bar Button Itemを含むNavigation Bar）を作成するために必要なNavBarアプリケーションのコードを示します。Navigation Barの右の位置にあるため、カスタムビューを`UIBarButtonItem`オブジェクトでラップしてから、`rightBarButtonItem`プロパティに割り当てなければなりません。

リスト 3-2 カスタムBar Button Itemの作成

```
// View 3 -ビューを持つカスタムの右バーボタン
UISegmentedControl *segmentedControl = [[UISegmentedControl alloc] initWithItems:
                                        [NSArray arrayWithObjects:
                                            [UIImage imageNamed:@"up.png"],
                                            [UIImage imageNamed:@"down.png"],
                                            nil]];

[segmentedControl addTarget:self
action:@selector(segmentAction:)forControlEvents:UIControlEventValueChanged];
segmentedControl.frame = CGRectMake(0, 0, 90, kCustomButtonHeight);
segmentedControl.segmentedControlStyle = UISegmentedControlStyleBar;
segmentedControl.momentary = YES;

defaultTintColor = [segmentedControl.tintColor retain];    // 後で使用するためにこ
れを保存する

UIBarButtonItem *segmentBarItem = [[UIBarButtonItem alloc]
initWithCustomView:segmentedControl];
[segmentedControl release];

self.navigationItem.rightBarButtonItem = segmentBarItem;
[segmentBarItem release];
```

プログラムの中でView ControllerのNavigation Itemを作成する方法は、ほとんどのアプリケーションで最もよく使われるアプローチです。Interface Builderを使用してBar Button Itemを作成することもできますが、通常は、プログラムの中で作成した方がはるかに簡単です。これらの項目は、View Controllerの`viewDidLoad`メソッド内で作成しなければなりません。

「編集(Edit)」／「完了(Done)」ボタンの使用

その場での編集をサポートするビューでは、ユーザが表示モードと編集モードを行き来できるようにする特別なタイプのボタンをNavigation Barに含めることができます。`UIViewController`の`editButtonItem`メソッドは、押されたときに「編集(Edit)」ボタンと「完了(Done)」ボタンの切り替えを行い、適切な値を引数としてView Controllerの`setEditing:animated:`メソッドを呼び出す、設定済みのボタンを返します。このボタンをView ControllerのNavigation Barに追加するには、次のようなコードを使用します。

```
myViewController.navigationItem.rightBarButtonItem = [myViewController
editButtonItem];
```

Navigation Barにこのボタンを含める場合は、View Controllerの`setEditing:animated:`メソッドをオーバーライドし、それを使用してビュー階層を調節しなければなりません。このメソッドの実装の詳細については、「ビューの編集モードの有効化」（52 ページ）を参照してください。

ナビゲーションツールバーの表示

iPhone OS 3.0以降では、ナビゲーションインターフェイスにツールバーを表示して、そこに、現在表示されているView Controllerが提供する項目を組み込むことができます。このツールバー自身は、Navigation Controllerオブジェクトが管理します。このレベルでツールバーをサポートすることは、画面間のスムーズな遷移を実現するために必要です。ナビゲーションスタックの最上位のView Controllerが変わるときには、Navigation Controllerは、別のツールバー項目セットへの変化をアニメーション化します。特定のView Controller用のツールバーの表示／非表示を切り替える場合にも、スムーズなアニメーションが作成されます。

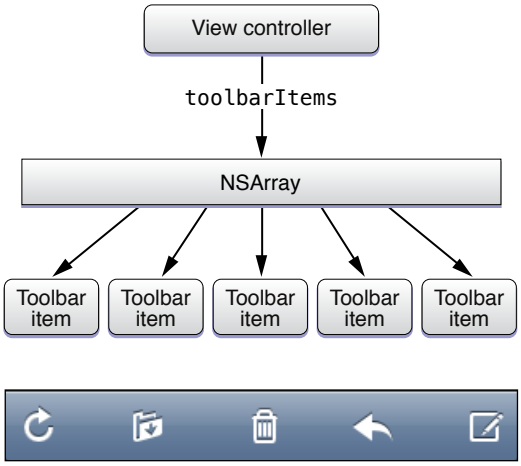
ナビゲーションインターフェイスにツールバーを設定するには、次の手順を実行する必要があります。

- Navigation Controllerオブジェクトの`toolbarHidden`プロパティを`NO`に設定して、ツールバーを表示します。
- `UIBarButtonItem`オブジェクトの配列をカスタムView Controllerの`toolbarItems`プロパティに割り当てます。これについては、「ツールバー項目の指定」（77 ページ）で説明します。

  特定のView Controllerにツールバーを表示したくない場合は、ツールバーを非表示にすることができます。これについては、「ツールバーの表示と非表示」（78 ページ）で説明します。

図 3-11に、カスタムView Controllerに関連付けられたオブジェクトがツールバーに表示されている例を示します。これらの項目は、配列内の配置と同じ順番でツールバーに表示されます。この配列には、すべてのタイプのBar Button Item（固定サイズや可変サイズの項目、システムボタン項目、任意のカスタムボタン項目など）を含めることができます。この例の5つの項目は、すべて「メール(Mail)」アプリケーションのボタン項目です。

図 3-11 ナビゲーションインターフェイスのツールバー項目



ツールバー項目の指定

Bar Button Itemを設定する場合は、必ず適切なターゲットとアクションをそのボタンに関連付ける必要があります。ターゲットとアクションの情報は、ツールバーでのタップに応答するために使われます。ほとんどの場合、ターゲットはView Controller自身です。これは、View Controllerがツールバー項目の提供を担当するからです。

図 3-12に、ツールバーの中央にSegmented Controlが配置されたツールバーの例を示します。リスト3-3に、このようなツールバーを設定するために必要なコードを示します。独自のView Controllerにこのメソッドを実装して、初期化のときに呼び出します。

図 3-12 ツールバーの中央に配置されたSegmented Control

リスト 3-3 中央にSegmented Controlを持つツールバーの設定

```
- (void)configureToolbarItems
{
   UIBarButtonItem *flexibleSpaceButtonItem = [[UIBarButtonItem alloc]

initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace
                        target:nil action:nil];

   // Segmented Controlを作成して設定する
   UISegmentedControl *sortToggle = [[UISegmentedControl alloc]
                        initWithItems:[NSArray arrayWithObjects:@"Ascending",
                                            @"Descending", nil]];
   sortToggle.segmentedControlStyle = UISegmentedControlStyleBar;
   sortToggle.selectedSegmentIndex = 0;
   [sortToggle addTarget:self action:@selector(toggleSorting:)
               forControlEvents:UIControlEventValueChanged];

   // Segmented Control用のBar Button Itemを作成する
```

```
    UIBarButtonItem *sortToggleButtonItem = [[UIBarButtonItem alloc]
                                        initWithCustomView:sortToggle];
    [sortToggle release];

    // ツールバー項目を設定する
    self.toolbarItems = [NSArray arrayWithObjects:
                         flexibleSpaceButtonItem,
                         sortToggleButtonItem,
                         flexibleSpaceButtonItem,
                         nil];

    [sortToggleButtonItem release];
}
```

初期化中にツールバー項目を設定するだけでなく、View Controllerでは、`setToolbarItems:animated:`メソッドを使用して動的に既存のツールバー項目セットを変更することもできます。このメソッドは、何らかのユーザアクションを反映してツールバーのコマンドを更新したい場合に便利です。たとえば、これを使用して、ツールバー上のボタンをタップすると、それに関連する子のボタンセットが表示される階層的なツールバー項目を実装できます。

ツールバーの表示と非表示

特定のView Controller用のツールバーを非表示にするには、そのView Controlerの`hidesBottomBarWhenPushed`プロパティを`YES`に設定します。Navigation Controllerは、このプロパティが`YES`に設定されたView Controllerを検出すると、このView Controllerがナビゲーションスタックにプッシュされる（またはナビゲーションスタックからポップされる）たびに、適切なトランジションアニメーションを生成します。

（常にではなく）時々ツールバーを非表示にする場合は、必要なときにそのNavigation Controllerの`setToolbarHidden:animated:`メソッドを呼び出します。このメソッドがよく使用されるのは、`setNavigationBarHidden:animated:`メソッドの呼び出しと組み合わせて、一時的にフルスクリーンビューを作成する場合です。たとえば、「写真(Photo)」アプリケーションでは、1つの写真を表示しているときにユーザが画面をタップすると、両方のバーの表示／非表示が切り替わります。

Tab Bar Controller

アプリケーションを1つ以上の別々の操作モードに分割整理するには、Tab Bar Controllerを使用します。Ta Bar Controllerは、自己完結型のビュー階層（Tab Barインターフェイスと呼ばれる）を管理します。このビュー階層のコンテンツは、Tab Bar Controllerが管理するビューと、カスタムView Controllerが管理するビューから構成されます。

この章では、アプリケーションでTab Bar Controllerを設定して使用する方法の概要について説明します。Tab Bar Controllerをほかの種類のView Controllerオブジェクトと組み合わせて使用する方法については、「View Controllerインターフェイスの組み合わせ」（101 ページ）を参照してください。

Tab Barインターフェイス

Tab Barインターフェイスは、同じデータセットに対して異なる見方を提供する場合や、機能系統に沿ってアプリケーションを編成したい場合に役立ちます。Tab Barインターフェイスの主要なコンポーネントは、画面の下端に表示されるTab Barビューです。このビューは、アプリケーションのさまざまなモード間の切り替えを行うために使われます。また、各モードの状態に関する情報を伝えることもできます。

Tab Barインターフェイス用のマネージャは、Tab Bar Controllerオブジェクトです。Tab Bar Controllerは、Tab Barビューを作成して管理します。また、各モードのコンテンツビューを提供するカスタムView Controllerも管理します。各カスタムView Controllerは、Tab Barビュー内のタブの1つに対応するルートView Controllerに指定されています。ユーザがタブをタップすると、Tab Bar Controllerオブジェクトは、そのタブを選択して、それに対応するルートView Controllerに関連付けられたビューを表示します。

図4-1に、「時計(Clock)」アプリケーションで実装されているTab Barインターフェイスを示します。このTab Bar Controllerは、固有のコンテナビューを持っています。このビューは、ほかのすべてのビュー（Tab Barビューを含む）を含んでいます。カスタムコンテンツは、選択されたタブのルートView Controllerによって提供されます。

図 4-1 Tab Barインターフェイスのビュー

Tab Barビューは通常のカスタマイズ可能なオブジェクトですが、Tab Barインターフェイスに含まれている場合は、変更してはいけません。Tab Barインターフェイスでは、Tab Barビューは、Tab Bar Controllerオブジェクトに所有されるプライベートなビュー階層の一部と見なされます。アクティブなタブのリストを変更する必要がある場合は、必ずTab Bar Controller自身のメソッドを使用しなければなりません。実行時にTab Barインターフェイスを変更する方法については、「実行時のタブの管理」（88 ページ）を参照してください。

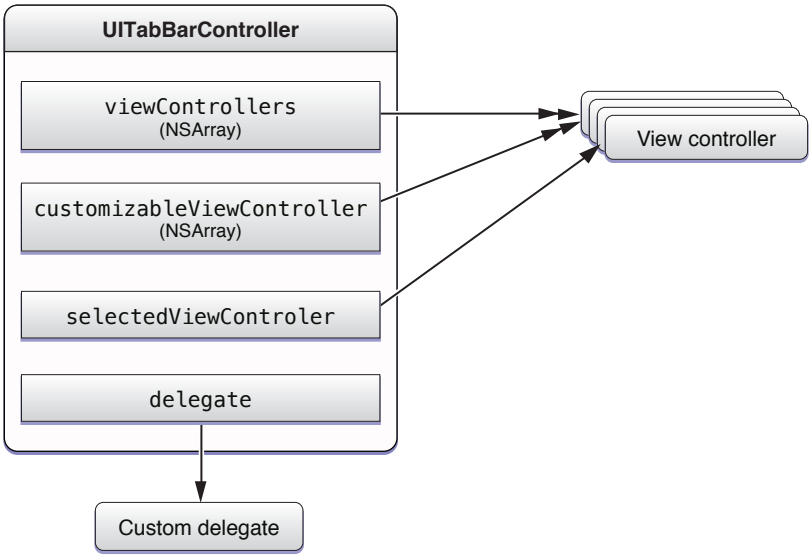
Tab Barインターフェイスのオブジェクト

標準的なTab Barインターフェイスは、次のオブジェクトから構成されます。

- `UITabBarController`オブジェクト
- タブごとに1つ存在するカスタムView Controllerオブジェクト
- オプションのデリゲートオブジェクト

図4-2に、Tab Bar Controllerとそれに関連付けられたView Controllerの関係を示します。Tab Bar Controllerの`viewControllers`プロパティ内の各View Controllerは、Tab Bar内の対応するタブのルートView Controllerです。

図 4-2 Tab Bar Controllerとそれに関連付けられたView Controller

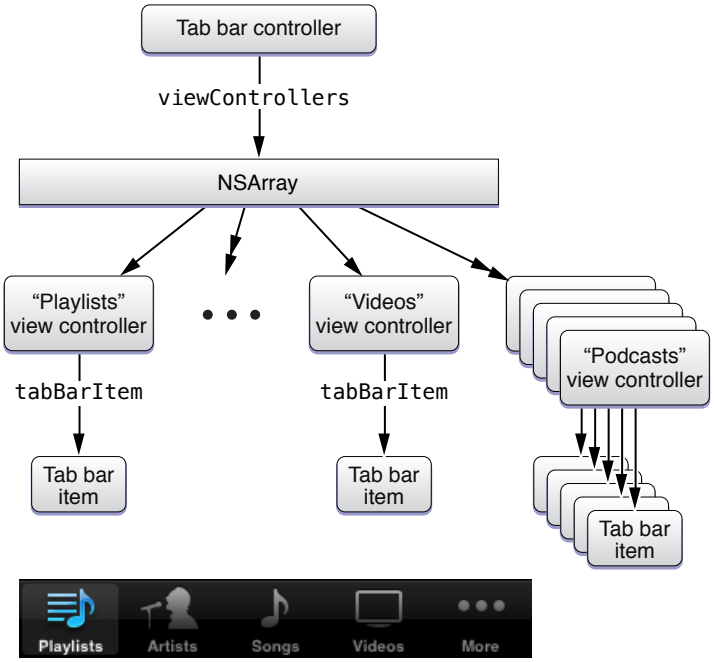


これらのカスタムView Controllerは、Tab Barインターフェイスの中で最も重要な要素です。各View Controllerには、対応するタブが選択されたときに表示されるコンテンツが定義されています。単一のビューを表示するカスタムView Controllerを使用できるほか、Navigation Controllerを使用して、タブ内でより複雑なナビゲーションを実現できます。ただし、タブ内に別のTab Bar Controllerをインストールすることはしません。

`viewControllers`プロパティに5つを超える項目を追加すると、Tab Bar Controllerは、Tab Barに入りきらなかった項目の表示を処理するために特殊なView Controller（**「More」View Controller**と呼ぶ）を自動的に挿入します。「More」View Controllerは、Tab Barに入りきらなかったView Controllerをテーブル形式で表示するカスタムインターフェイスです。このテーブルにはいくつでもView Controllerを表示できます。「More」View Controllerをカスタマイズしたり選択したりすることはできません。また、「More」View Controllerは、Tab Bar Controllerが管理するView Controllerリストの中には含まれません。たいていは、必要なときに自動的に表示されて、カスタムコンテンツからは独立しています。ただし、`UITabBarController`の`moreNavigationController`プロパティにアクセスすれば、「More」View Controllerへの参照を取得できます。

Tab Barビューは、Tab Barインターフェイスの主要部分ですが、このビューを直接変更することはありません。Tab Bar Controllerオブジェクトは、カスタムView Controllerが提供する`UITabBarItem`オブジェクトによってTab Barのコンテンツを構成します。図4-3に、「iPod」アプリケーションのTab Bar Controller、View Controller、Tab Bar Itemの各オブジェクトの関係を示します。一度に表示できる数よりも多くのView Controllerが存在するため、最初の4つのView ControllerのTab Bar Itemだけが表示されます。最後のTab Bar Itemは、「More」View Controllerによって提供されます。

図 4-3 「iPod」アプリケーションのTab Bar Item



Tab Bar ItemはTab Barを構成するために使われるため、Tab Barインターフェイスが表示する前に、各ルートView ControllerのTab Bar Itemを設定しなければなりません。Interface Builderを使用してインターフェイスを作成している場合は、そのタイトルと画像を指定できます。これについては「nibファイルを使用したTab Barインターフェイスの作成」（84 ページ）で説明します。プログラムによってTab Barインターフェイスを作成している場合は、View Controllerごとに1つの新規の`UITabBarItem`オブジェクトを作成しなければなりません。これについては、「プログラムによるTab Barインターフェイスの作成」（87 ページ）で説明します。

Tab Bar Controllerは、オプションでデリゲートオブジェクトをサポートします。このオブジェクトは、Tab Barの選択やカスタマイズに応答するために使用できます。デリゲート関連のメッセージへの応答については、「実行時のタブの管理」（88 ページ）を参照してください。

Tab Barインターフェイスの作成

Tab Barインターフェイスを作成する前に、その使いかたを決定する必要があります。Tab Barインターフェイスはデータの構成を支配するため、Tab Barインターフェイスを使用する適切な方法は次の2つしかありません。

- アプリケーションのメインウインドウに直接インストールする。
- モードに固有な構成を必要とするデータを表示するために、モーダルモードで表示する。

アプリケーションのメインウインドウにTab Barインターフェイスをインストールする方法が、はるかに一般的な使いかたです。このようなシナリオでは、Tab Barインターフェイスがアプリケーションのデータの基本的な構成方針を提供します。また、各タブはアプリケーションの個別の部分にユーザを誘導します。Tab Barインターフェイスはアプリケーション全体へのアクセスを提供します。このため、ウインドウのルート部分でなければなりません。

注： Navigation Controllerをタブ内に埋め込むことはできますが、その逆はできません。ナビゲーションインターフェイス内にTab Barインターフェイスを表示すると、ユーザを混乱させる恐れがあります。ナビゲーションインターフェイスは、1つ以上のカスタムView Controllerを使用して、1つの目的（通常は、特定のタイプのデータの管理）に焦点を当てたインターフェイスを表示します。それとは対照的に、Tab Barインターフェイスのタブはアプリケーション内のまったく別々の目的を表し、それらの間に何の関係も必要ありません。また、Tab Bar Controllerをナビゲーションスタックにプッシュすると、その画面にだけタブが表示され、その他の画面には表示されません。

もちろん、非常に特殊なニーズのためにそうする価値があれば、Tab Bar Controllerをモーダルモードで表示することも可能です。たとえば、異なるオプションセットを複数持つ複雑なデータセットを編集するために、Tab Bar Controllerをモーダルモードで表示することができます。モーダルビューは画面全体に表示されるため、このTab Barの存在は、モーダルモードで表示されたデータを表示したり編集したりするために利用できる選択肢を表しているだけです。もっと簡単な設計方法が利用できる場合は、このようなTab Barの使いかたは避けるべきです。

Tab Barインターフェイス用のカスタムView Controllerの定義

Tab Barインターフェイスの各モードは、ほかのすべてのモードとは独立です。このため、各タブのルートView Controllerが本質的にそのタブのコンテンツを定義します。したがって、各タブに割り当てられたView Controllerは、特定の操作モードのニーズを反映していなければなりません。比較的複雑なデータセットを表示する必要がある場合は、Navigation Controllerをインストールして、そのデータ内のナビゲーションを管理します。表示するデータが単純な場合は、単一のビューを持つカスタムView Controllerをインストールします。

図4-4に、「時計(Clock)」アプリケーションのいくつかの画面を示します。「世界時計(World Clock)」タブでは、主に時計のリストを編集するために必要なボタンを表示するために、Navigation Controllerを使用しています。「ストップウォッチ(Stopwatch)」タブはインターフェイス全体に1つの画面しか必要ありません。したがって、単一のView Controllerを使用しています。「タイマー(Timer)」タブでは、メイン画面用にカスタムView Controllerを1つ使用し、「タイマー終了時(When Timer Ends)」ボタンがタップされたときに別のView Controllerをモーダルモードで表示します。

図 4-4 「時計(Clock)」アプリケーションのタブ

Tab Bar Controllerは、ルートView Controllerの表示に関するやり取りをすべて処理します。このため、タブやタブ内のView Controllerに関して、デベロッパがしなければならないことはほとんどありません。カスタムView Controllerは、いったん表示されたら、コンテンツを表示することに集中するべきです。

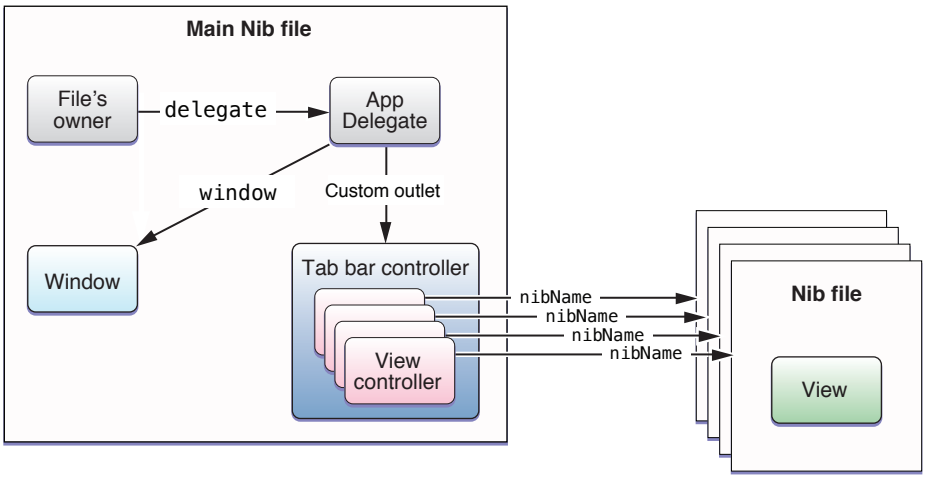
nibファイルを使用したTab Barインターフェイスの作成

Tab Bar ControllerをnibファイルからロードするセマンティクスはカスタムView Controllerの場合とは少し異なります。カスタムView Controllerでは、View Controllerに関連付けられているビューをnibファイルに格納しますが、一般に、View Controllerそのものは、プログラムの中で作成する場合でも、ほかのnibファイルからロードする場合でも、別個に作成します。一方、Tab Bar Controllerの場合は、ビューは必ずプログラムの中で作成するため、独立のnibファイルに格納するビューは存在しません。この結果、Tab Bar Controllerがnibファイルを管理することはありません。つまり、Tab Bar ControllerをFile's Ownerプレースホルダに割り当てることはありません。代わりに、Tab Bar Controllerとnibファイルが関わりを持つのは、Tab Bar Controllerそのものがnibファイルに保存される場合だけです。このような理由から、Tab Bar Controllerはほとんどの場合、ほかのオブジェクトが管理するnibファイルに保存されます。

Tab Bar Controllerをアプリケーションのメインnibファイルに含めることが最も理にかなっています。Tab Bar Controller自身がアプリケーションのウインドウのメインビューを提供する場合は、このようにします。モーダルモードで表示されるTab Bar Controllerをメインnibファイル（またはその他のnibファイル）からロードすることもできますが、それは最適ではありません。実際には、Tab Bar Controllerを必要な時にプログラムの中で作成する方が通常は簡単です。

図4-5に、アプリケーションのメインnibファイル内のTab Bar Controllerとそれに関連するオブジェクトの典型的な構成を示します。各カスタムView Controllerは、1つのタブに関連付けられたルートView Controllerを表します。これらはすべてカスタムView Controllerであるため（Navigation Controllerではない）、各View Controllerは、そのビューを含む別のnibファイルへの参照を持っています。メインnibファイルにこれらのカスタムビューを含めることもできますが、それはお勧めできません。ビューを別のnibファイルに保存することによって、システムは必要に応じてそれらをメモリから完全に破棄できます。

図 4-5　Tab Barインターフェイスを含むnibファイル



Xcodeプロジェクトをゼロから作成する場合は、Tab Bar Applicationテンプレートを使用してプロジェクトを作成します。このプロジェクトに付属のメインnibファイルには、プロジェクト用に最小限の設定が行われたTab Bar Controllerが含まれています。

次の手順は、アプリケーションのメインnibファイルにゼロからTab Bar Controllerを追加する方法を示しています。Tab Bar Applicationテンプレートを使用する場合は、最初の2つのステップを単純にスキップできます。

1. Tab Bar Controllerオブジェクトをライブラリから Interface Builderのドキュメントウインドウにドラッグします。

   Tab Bar Controllerをnibファイルに追加すると、Interface Builderによって、1つのTab Barビュー、2つのルートView Controller、および2つのTab Bar Item（View Controllerごとに1つ）も追加されます。これらのオブジェクトには、Tab Bar Controllerの編集サーフェスでオブジェクトを選択してアクセスできます。アウトラインモードやブラウザモードのときには、ドキュメントウインドウから任意のオブジェクトを選択することもできます。

   Tab Bar Applicationテンプレートを使用してプロジェクトを作成した場合は、このステップはすでに完了しています。

2. アウトレットを使用して、Tab Bar Controllerへの参照を保存します。

   実行時にこのTab Bar Controllerにアクセスするには、アウトレットを使用するか、nibファイルをロードするときにそのnibファイルの最上位レベルのオブジェクトを明示的に取得する必要があります。一般には、アウトレットを使用する方がはるかに簡単です。アウトレットを追加するには、次のようにアプリケーションデリゲートクラスの宣言に変数を追加します。

   ```
   @interface MyAppDelegate :NSObject <UIApplicationDelegate> {
      IBOutlet UITabBarController*  myTabBarController;
   }
   @end
   ```

   アウトレットの定義を追加したら、そのアウトレットからTab Bar Controllerオブジェクトへの接続を作成します。

   Tab Bar Applicationテンプレートを使用してプロジェクトを作成した場合は、このステップはすでに完了しています。

3. インターフェイスに表示したいタブの数に合わせて、View Controllerを追加または削除します。

   Tab Bar Controllerオブジェクトに含まれるView Controllerの数によって、Tab Barインターフェイスに表示されるタブの数が決まります。最終的なTab Bar Controllerには、少なくとも2つのView Controllerが含まれていなければなりません。そうならない場合は、Tab Bar Controllerの必要性が疑われます。Interface Builderのライブラリから、View Controllerオブジェクト(`UIViewController`)、Navigation Controllerオブジェクト(`UINavigationController`)、またはTable View Controllerオブジェクト(`UITableViewController`)をドラッグし、それをタブを関連付けることができます。

   View Controllerを追加するには、次のいずれかを実行します。

   - 適切なオブジェクトをライブラリから編集サーフェイス内のTab Barにドラッグします。
   - オブジェクトをライブラリから、Interface Builderのドキュメントウインドウ内のTab Bar Controllerにドラッグします。このウインドウはアウトラインモードになっていなければなりません。

Navigation ControllerまたはTable View Controllerをインターフェイスに追加する場合は、適切なオブジェクトをライブラリからドラッグするか、Tab Bar Controllerを選択し、「Attributes」インスペクタを使用してView Controllerの種類を設定します。どちらを選択しても、適切な種類のView Controllerオブジェクトがnibファイルに追加されます。汎用のView Controllerオブジェクトをnibファイルにドラッグして、そのクラス名を希望のクラスタイプに変更することによって、Navigation ControllerまたはTable View Controllerを追加するべきではありません。

View Controllerを削除するには、編集サーフェスまたはドキュメントウインドウでView Controllerオブジェクトを選択してDeleteキーを押します。

4. 画面に表示する順番にView Controllerを並べます。

   Tab Bar Controllerの編集サーフェスに表示されているタブをドラッグするか、Interface Builderのドキュメントウインドウで View Controllerをドラッグすることによって（アウトラインモードの場合のみ）、View Controller（およびそれに対応するタブ）を並べ替えることができます。編集サーフェスにはすべてのタブが表示されますが、実行時には5つしか表示されません。Tab Bar Controllerに6つ以上のView Controllerが含まれる場合は、初期状態ではTab Barには最初の4つしか表示されません。Tab Barの最後の場所は、残りのView Controllerを表示する「More」View Controller用に予約されています。

5. 各タブにルートView Controllerを設定します。

   各ルートView Controllerには、次のような属性を設定しなければなりません。

   - 「Identity」インスペクタを使用して、任意のView Controllerオブジェクトのクラスを設定します。ルートView Controllerが汎用のView ControllerオブジェクトまたはTable View Controllerオブジェクトの場合は、クラス名をカスタムサブクラスに変更できます。ルートView ControllerがNavigation Controllerオブジェクトの場合は、クラス名を変更してはいけません。
   - このView Controllerにビューを設定します。そのためによく使われるのは、View ControllerのNIB Name属性を、ビューを含むnibファイルの名前に設定する方法です。ビューをメインnibファイルに含めることもできますが、それはお勧めできません。
   - 必要に応じて、View Controllerのスタイルや外観情報を設定します。

   ルートView Controllerの1つにNavigation Controllerを使用している場合は、「nibファイルからのナビゲーションインターフェイスのロード」（62 ページ）の説明に従って、それを設定します。Navigation ControllerをTab Bar Controllerに埋め込む方法の詳細および例については、「Tab BarインターフェイスへのNavigation Controllerの追加」（101 ページ）を参照してください。

6. 各View ControllerにTab Bar Itemを設定します。

   Tab Bar Controllerの編集サーフェス、またはInterface Builderのドキュメントウインドウ（アウトラインモードまたはブラウザモードの場合）からTab Bar Itemを選択できます。Interface Builderを使用して、Tab Bar Itemのタイトル、画像、およびバッジを指定できます。あるいは、Tab Bar Itemを標準的なシステムタブの1つに設定することもできます。それには、「Attributes」インスペクタでIdentifierプロパティに値を割り当てます。

7. アプリケーションデリゲートの`applicationDidFinishLaunching:`メソッド内で、Tab Bar Controllerのビューをメインウインドウに追加します。

   Tab Bar Controllerが、それ自身を自動的にアプリケーションのウインドウにインストールすることはありません。次のようなコードによって、プログラムでインストールする必要があります。

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    [window addSubview:myTabBarController.view];
```

```
}
```

8. nibファイルを保存します。

Tab Bar Controllerと一緒にnibファイルに追加されたTab Barビューを設定する必要はありません。このバーには、Tab Bar Controllerオブジェクトが管理するスタイルオプションおよびその他のオプションは含まれていません。

Interface Builderを使用してnibファイルを作成する方法の詳細については、『*Interface Builder User Guide*』を参照してください。

プログラムによるTab Barインターフェイスの作成

Tab Bar Controllerをプログラムの中で作成したい場合、それに最も適した場所はアプリケーションデリゲートの`applicationDidFinishLaunching:`メソッド内です。通常、Tab Bar Controllerはアプリケーションのウインドウのルートビューを提供するので、アプリケーションを起動した直後で、ウインドウを表示する前に、Tab Bar Controllerを作成する必要があります。Tab Barインターフェイスの作成手順は次のようになります。

1. 新規の`UITabBarController`オブジェクトを作成します。
2. タブごとに1つカスタムルートView Controllerを作成します。
3. これらのルートView Controllerを配列に追加し、その配列をTab Bar Controllerの`viewControllers`プロパティに割り当てます。
4. Tab Bar Controllerのビューをアプリケーションのメインウインドウに追加します。

リスト 4-1に、アプリケーションのメインウインドウにTab Bar Controllerのインターフェイスを作成してインストールするために必要な基本的なコードを示します。この例では、2つのタブしか作成していませんが、必要に応じて、いくつでもタブを作成できます。それには、さらに多くのView Controllerオブジェクトを作成して、それを`controllers`に追加します。カスタムView Controller名の`MyViewController`と`MyOtherViewController`は、独自のアプリケーションのクラスで置きかえる必要があります。

リスト 4-1　Tab Bar Controllerをゼロから作成する

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
   tabBarController = [[UITabBarController alloc] init];

   MyViewController* vc1 = [[MyViewController alloc] init];
   MyOtherViewController* vc2 = [[MyOtherViewController alloc] init];

   NSArray* controllers = [NSArray arrayWithObjects:vc1, vc2, nil];
   tabBarController.viewControllers = controllers;

   // Tab Bar Controllerの現在のビューをウインドウのサブビューとして追加する
   [window addSubview:tabBarController.view];
}
```

プログラムによるTab Bar Itemの作成

Tab Barインターフェイスの各ルートView Controllerには、それに対応するタブに表示する画像とテキストを持つ`UITabBarItem`オブジェクトを提供しなければなりません。Tab Barインターフェイスが表示される前であれば、いつでもTab Bar ItemをView Controllerに関連付けることができます。それには、Tab Bar Itemをそれに対応するView Controllerの`tabBarItem`プロパティに割り当てます。Tab Bar Itemを作成する理想的なタイミングは、View Controller自身の初期化の間ですが、通常これができるのはカスタムView Controllerの場合だけです。非常に簡単に、View Controllerを作成して初期化し、Tab Bar Itemを作成してそれらを関連付けることができます。

リスト 4-2に、カスタムView Controller用にTab Bar Itemを作成する方法の例を示します。カスタムView Controllerであるため、このView Controllerが初期化処理の一部としてTab Bar Itemそのものを作成します。この例では、Tab Bar Itemにカスタム画像（アプリケーションのバンドルに保存されている）とカスタムタイトル文字列の両方が含まれます。

リスト 4-2　View ControllerのTab Bar Itemの作成

```
- (id)init {
   if (self = [super initWithNibName:@"MyViewController" bundle:nil]) {
      self.title = @"My View Controller";

      UIImage* anImage = [UIImage imageNamed:@"MyViewControllerImage.png"];
      UITabBarItem* theItem = [[UITabBarItem alloc] initWithTitle:@"Home"
image:anImage tag:0];
      self.tabBarItem = theItem;
      [theItem release];
   }
   return self;
}
```

Tab Barインターフェイスをnibファイルからロードする場合は、Interface Builderを使用してTab Bar Itemを作成することもできます。Tab Bar Controller（およびTab Bar Item）をnibファイルに保存する方法の詳細については、「nibファイルを使用したTab Barインターフェイスの作成」（84 ページ）を参照してください。

実行時のタブの管理

Tab Barインターフェイスを作成した後は、いくつかの方法を使用してインターフェイスを変更したりアプリケーション内の変化に対応できます。タブの追加や削除ができます。また、デリゲートオブジェクトを使用して、動的な条件に基づいてタブの選択を禁止することもできます。個々のタブにバッジを付加して、そのタブに対してユーザの注意を引き付けることもできます。以降の各セクションでは、アプリケーションでのこれらの機能の利用方法を示します。

タブの追加と削除

Tab Barインターフェイス内のタブの数が動的に変化する場合は、必要に応じて実行時に適切な変更を行うことができます。作成時にタブを設定する方法と同様に、適切なView ControllerセットをTab Bar Controllerに割り当てることによって、実行時にタブを変更します。ユーザに見えるような形でタブの追加や削除を行う場合は、`setViewControllers:animated:`を使用してタブの変更をアニメーション化することもできます。

リスト 4-3に、現在選択されているタブ内の特定のボタンのタップに応答して、そのタブを削除するメソッドを示します。このメソッドは、対象タブのルートView Controllerで実装されています。不要になったタブを削除する場合も、これと同様のコードを使用できます。たとえば、これを使用して、一度しか入力する必要がないユーザ固有のデータを含むタブを削除できます。

リスト 4-3　現在のタブの削除

```
- (IBAction)processUserInformation:(id)sender
{
   // ユーザデータを検証するために、アプリケーション固有の何らかのメソッドを呼び出す。
   // このカスタムメソッドがYESを返した場合は、タブを削除する。
   if ([self userDataIsValid])
   {
      NSMutableArray* newArray = [NSMutableArray
arrayWithArray:self.tabBarController.viewControllers];
      [newArray removeObject:self];

      [self.tabBarController setViewControllers:newArray animated:YES];
   }
}
```

タブの選択の禁止

ユーザがタブを選択できないようにするには、デリゲートオブジェクトを用意して、そのオブジェクトにtabBarController:shouldSelectViewController:メソッドを実装します。タブの選択を禁止するのは、タブがまったくコンテンツを持たない場合など、一時的な場合だけにするべきです。たとえば、アプリケーションでユーザに特定の情報（ログイン名とパスワードなど）の入力を求める場合、必要な情報をユーザに要求するタブ以外のすべてのタブを無効にすることができます。リスト 4-4に、このようなメソッドの例を示します。hasValidLoginメソッドは、入力された情報を検証するために実装したカスタムメソッドです。

リスト 4-4　タブの選択の禁止

```
- (BOOL)tabBarController:(UITabBarController *)aTabBar
         shouldSelectViewController:(UIViewController *)viewController
{
   if (![self hasValidLogin] && (viewController != [aTabBar.viewControllers
objectAtIndex:0]) )
   {
      // 最初のタブ以外はすべて無効にする。
      return NO;
   }

   return YES;
}
```

ユーザによるタブの変更の監視

Tab Barで発生するユーザによる変更には、次の2種類があります。

- ユーザがタブを選択する。
- ユーザがタブの配置を変更する。

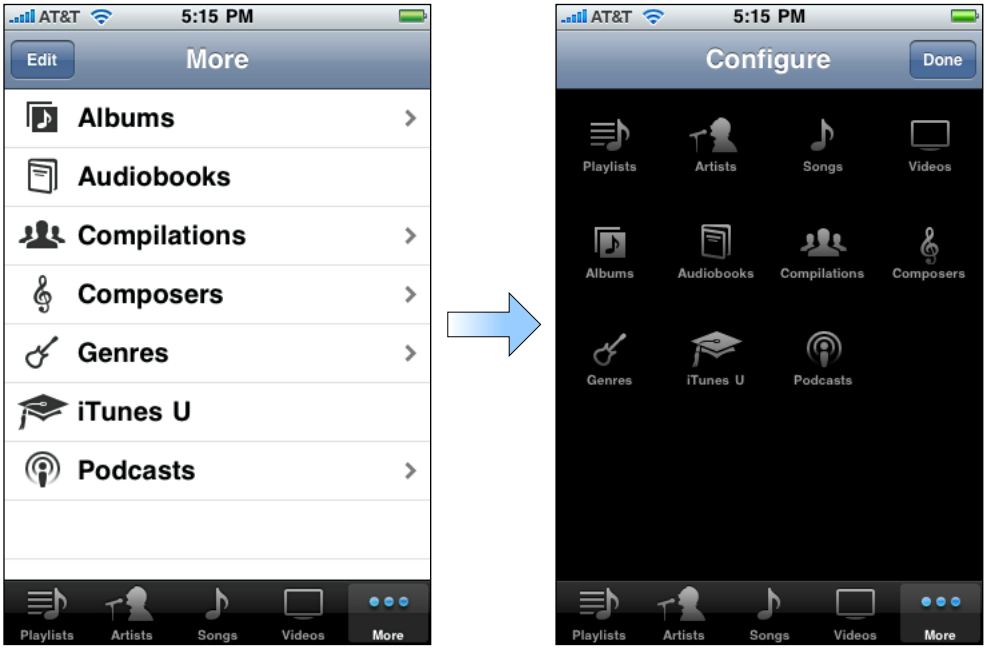
どちらの変更もTab Bar Controllerのデリゲートに報告されます。デリゲートは`UITabBarControllerDelegate`プロトコルに準拠したオブジェクトです。デリゲートを用意して、ユーザによる変更を追跡したり、それに応じてアプリケーションの状態情報を更新することができます。ただし、(表示中または非表示中の）View Controllerが標準的に処理する作業を実行するために、これらの通知を使用するべきではありません。たとえば、現在選択されているビューのスタイルに合うようにステータスバーの外観を変更するために、Tab Bar Controllerのデリゲートを使用することはしません。この種の外観の変更は、カスタムView Controllerで処理するのが最善です。

`UITabBarControllerDelegate`プロトコルのメソッドとその使いかたの詳細については、『*UITabBarControllerDelegate Protocol Reference*』を参照してください。

タブのカスタマイズの禁止

「More」View Controllerには、Tab Barに表示する項目をユーザが修正できるようにする機能が内蔵されています。たくさんのタブを持つアプリケーションの場合、ユーザーはこの機能を利用して、すぐにアクセスできる画面と、アクセスするために余分なナビゲーションが必要となる画面を選択できます。図 4-6の左側に、「iPod」アプリケーションで表示される「その他(More)」画面を示します。ユーザがこの画面の左上隅にある「編集(Edit)」ボタンをタップすると、「More」View Controllerがこの図の右側に示すような設定画面を自動的に表示します。この画面から、新しい項目をTab Barにドラッグすることによって、Tab Barの内容を置き換えることができます。

図 4-6　「iPod」アプリケーションのTab Barの設定



たいていの場合は、タブの配置変更をユーザに許可するのは良い考えです。しかし、特定のタブをTab Barから削除したり、特定のタブをTab Barに配置する操作をユーザに許可したくない場合もあります。このような場合は、View Controllerオブジェクトの配列を`customizableViewControllers`プロパティに割り当てます。この配列には、配置変更してもよいView Controllerのサブセットを含めます。この配列に含まれていないView Controllerは、配置変更の画面に表示されません。したがって、すでにTab Barに表示されているView ControllerをTab Barから削除することはできません。

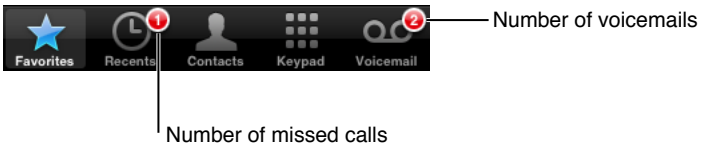
重要： Tab Barインターフェイスで View Controllerを追加したり削除したりすると、カスタマイズ可能なView Controllerの配列もデフォルト値にリセットされて、すべてのView Controllerがカスタマイズ可能な状態に戻ります。したがって、`viewControllers`プロパティを（直接、または`setViewControllers:animated:`メソッドを呼び出して）変更したときに、引き続きカスタマイズ可能なView Controllerを制限する場合は、`customizableViewControllers`プロパティ内のオブジェクトの配列も更新する必要があります。

タブのバッジの変更

通常、Tab Barインターフェイスのタブの外観は、そのタブが選択されているとき以外は変化しません。ただし、特定のタブへの注意を引き付けたい場合は（そのタブにユーザに見てほしい新しいコンテンツがある場合など）、バッジを付加することでそれができます。

バッジは、タブの隅に表示される赤い小さな印です。バッジの内側には、デベロッパが提供する何らかのカスタムテキストが表示されます。通常、バッジには、そのタブで利用可能な新規項目の数を表す数字が含まれますが、非常に短い文字列を指定することもできます。図4-7に、「電話(Phone)」アプリケーションのタブに付加されたバッジを示します。

図 4-7 Tab Bar Itemのバッジ



タブにバッジを割り当てるには、該当するTab Bar Itemの`badgeValue`プロパティにnil以外の値を割り当てます。たとえば、バッジ内に新規項目の数を表示するView Controllerでは、次のようなコードを使用してバッジの値を作成します（この例の`numberOfNewItems`プロパティは、新規項目の数を記録するためにView Controllerに実装された架空のプロパティです。この例を実際のコードに実装するには、このプロパティへの参照を、独自のView Controllerの適切な値に置き換えてください）。

```
if (self.numberOfNewItems == 0)
   self.tabBarItem.badgeValue = nil;
else
   self.tabBarItem.badgeValue = [NSString stringWithFormat:@"%d",
self.numberOfNewItems];
```

バッジの値を表示したり更新したりするタイミングを決定するのは、デベロッパの責任です。ただし、View Controllerにバッジ用の値を持つプロパティ（上の例の架空の`numberOfNewItems`プロパティなど）が含まれている場合は、KVO通知を使用して値の変化を検出し、それに応じてバッジを更新できます。KVO通知のセットアップと処理については、『*Key-Value Observing Programming Guide*』を参照してください。

Tab Bar Controllerとビューの回転

Tab Bar Controllerは、デフォルトでは縦長の向きをサポートします。すべてのルートView Controllerが横長の向きをサポートしていない限り、横長の向きには回転しません。デバイスの向きが変化すると、Tab Bar ControllerはView Controllerの配列に問い合わせをします。1つでもその向きをサポートしないものがあれば、Tab Bar Controllerは向きを変更しません。

Tab Barとフルスクリーンレイアウト

Tab Bar Controllerは、ほかのほとんどのView Controllerとは異なる方法でフルスクリーンレイアウトをサポートします。ステータスバーやNavigation Bar（存在する場合）の背後にビューを重ねたい場合は、カスタムView Controllerの`wantsFullScreenLayout`プロパティを`YES`に設定します。ただし、このプロパティを`YES`に設定しても、Tab Barビューの背後にビューを重ねることはできません。Tab Bar Controllerは、Tab Barの背後にビューが重ならないように常にビューのサイズを変更します。

カスタムビューのフルスクリーンレイアウトの詳細については、「カスタムビューでのフルスクリーンレイアウトの採用」（51 ページ）を参照してください。

モーダルView Controller

モーダルView Controllerは、アプリケーションのフローを管理するための興味深い方法を提供します。最も一般的なのは、ユーザから重要な情報を得るために、モーダルView Controllerを一時的な割り込みとして使用するケースです。一方、特定のタイミングでアプリケーションに代替のインターフェイスを実装するために、View Controllerをモーダルモードで表示して使用することもできます。

この章では、アプリケーションでのモーダルビューの用途について説明し、それを表示するタイミングと方法を示します。

モーダルView Controllerについて

モーダルView Controllerは、新規に1画面分のコンテンツを表示するときに自由に使えるツールです。モーダルView Controllerは、`UITabBarController`や`UINavigationController`のように、`UIViewController`の特定のサブクラスではありません。むしろ、任意のView Controllerをアプリケーションからモーダルモードで表示できます。ただし、Tab Bar ControllerやNavigation Controllerのように、前の画面と新たに表示する画面の関係に特定の意味を持たせる場合に、View Controllerをモーダルモードで表示します。

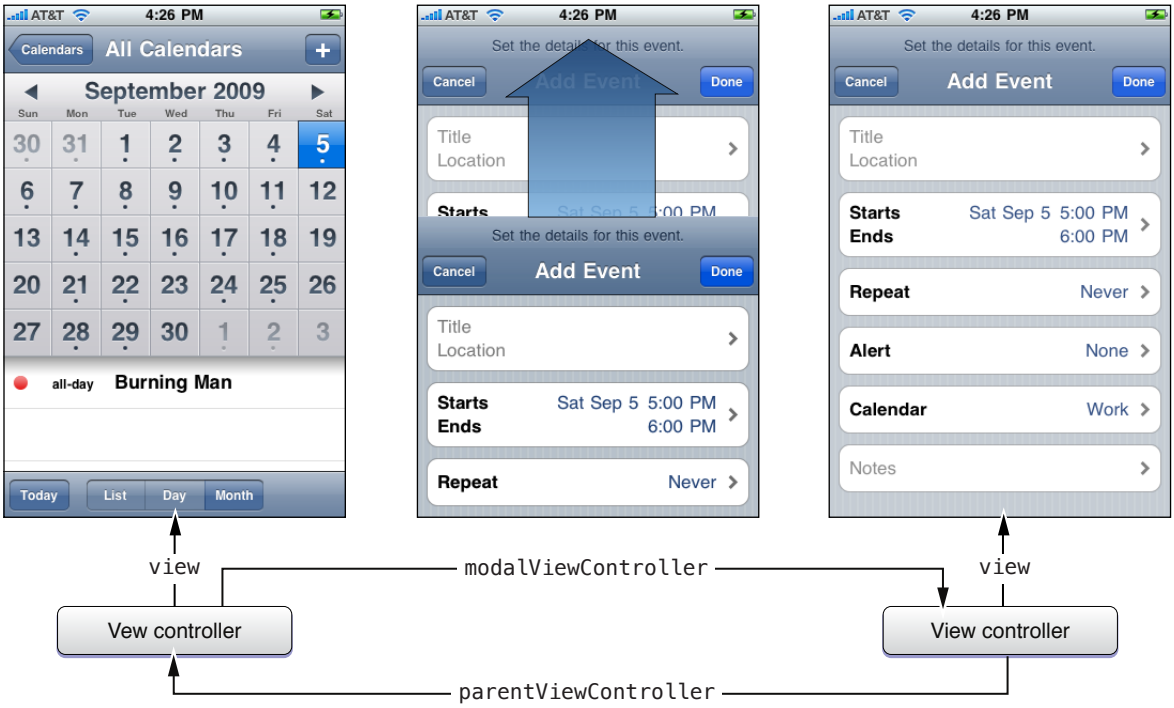
アプリケーションでモーダルView Controllerを使用する理由としては、次のような理由が考えられます。

- ユーザから直ちに情報を収集するため。
- 何らかのコンテンツを一時的に表示するため。
- 作業モードを一時的に変更するため。
- デバイスの向きに応じて代替のインターフェイスを実装するため。

これらの理由のほとんどでは、何らかの情報を収集したり表示したりするために、アプリケーションのワークフローに一時的に割り込みを行うことを意味しています。必要な情報を入手したら（または、適切な情報をユーザに表示したら）、モーダルView Controllerを消去してアプリケーションを以前の状態に戻します。最後の用途（代替インターフェイスの実装）も一時的な割り込みと見なすべきです。

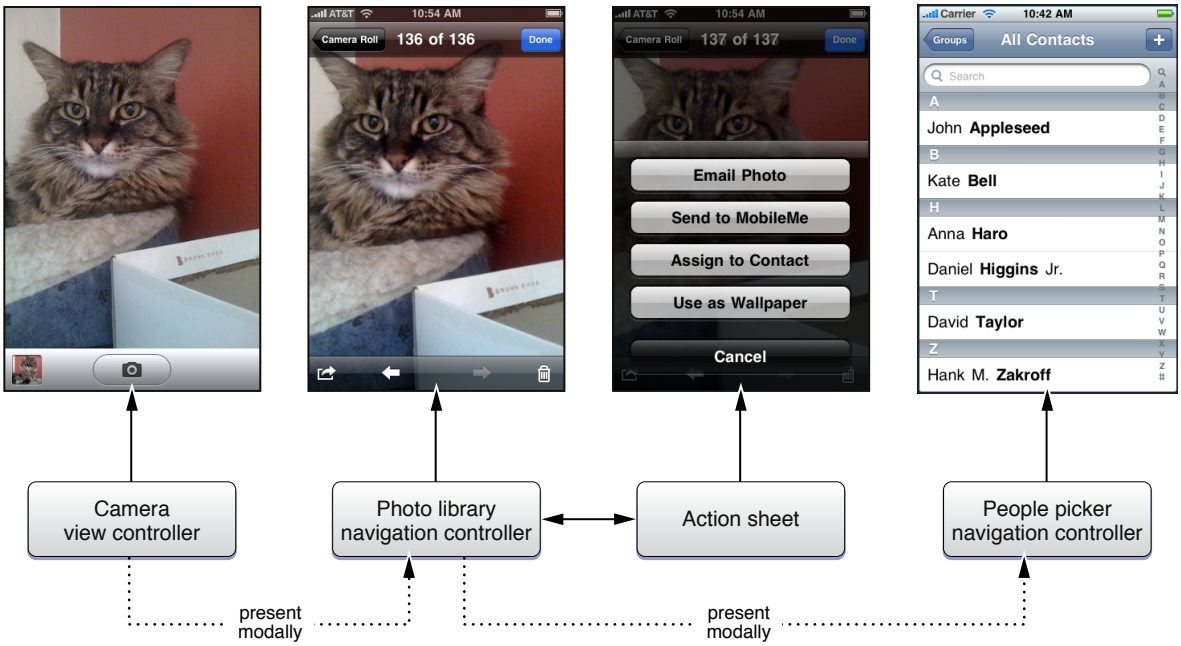
モーダルView Controllerを表示すると、システムによって表示元のView Controllerと表示されたView Controllerの間に親子関係が作成されます。具体的に言うと、表示元のView Controllerの`modalViewController`プロパティが、表示した（子の）View Controllerを指すように更新されます。同様に、表示されたView Controllerの`parentViewController`プロパティが、表示元のView Controllerを指すように更新されます。図 5-1に、「カレンダー(Calendar)」アプリケーションのメイン画面を管理するView Controlerと、新規イベントを作成するために使われるモーダルView Controllerの関係を示します。

図 5-1　「カレンダー(Calendar)」アプリケーションのモーダルビュー



どのView Controllerオブジェクトも、ほかの1つのView Controllerオブジェクトをモーダルモードで表示できます。これは、モーダルモードで表示されたView Controller自身にも当てはまります。つまり、必要に応じて、モーダルView Controllerの手前に新規のモーダルView Controllerを表示して、モーダルView Controllerを連鎖させることができます。図 5-2に、この連鎖のプロセスとそれを開始するアクションを図示します。この例では、ユーザがカメラビューのアイコンをタップすると、アプリケーションは、ユーザの写真を含むモーダルView Controllerを表示します。フォトライブラリのツールバーのアクションボタンをタップすると、適切なアクションを選択する画面がユーザに表示されます。次に、ユーザが選択したアクションに対応する別のモーダルView Controller（Peopleピッカー）が表示されます。連絡先を選択するか、またはPeopleピッカーをキャンセルすると、そのインターフェイスが閉じてフォトライブラリに戻ります。「完了(Done)」ボタンをタップすると、フォトライブラリが閉じてカメラインターフェイスに戻ります。

図 5-2 モーダルView Controllerチェーンの作成

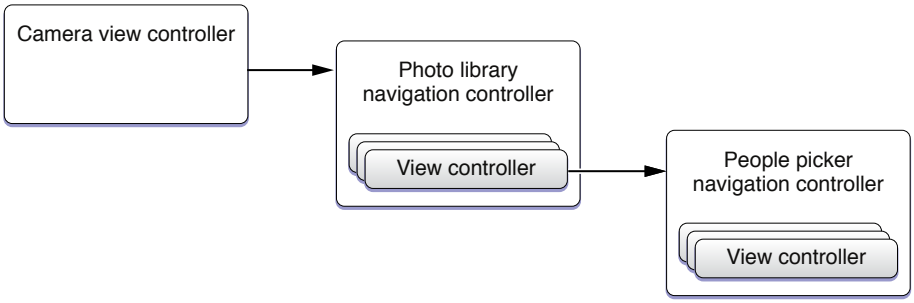


モーダルモードで表示されたView Controllerのチェーン内の各View Controllerは、同じチェーン内の周囲のほかのオブジェクトへのポインタを持っています。つまり、ほかのモーダルView Controllerを表示するモーダルView Controllerは、`parentViewController`プロパティと`modalViewController`プロパティの両方に有効なオブジェクトを持っています。これらの関係を使用して、必要に応じてView Controllerのチェーンをたどることができます。たとえば、ユーザが現在の操作をキャンセルした場合は、最初にモーダルモードで表示されたView Controllerを閉じることによって、チェーン内のすべてのオブジェクトを閉じることができます。つまり、1つのモーダルView Controllerを閉じると、そのView Controllerだけでなく、そのView Controllerがモーダルモードで表示したすべてのView Controllerを閉じることができます。

図 5-2 （95 ページ）では、モーダルモードで表示したView Controllerが両方ともNavigation Controllerであることに注目してください。`UINavigationController`オブジェクトも、カスタムView Controllerを表示するのと同じ方法でモーダルモードで表示できます（まれですが、Tab Bar Controllerを表示することもできます）。

Navigation Controllerをモーダルモードで表示するときは、ナビゲーションスタック上の任意のView Controllerを表示するのではなく、必ず`UINavigationController`オブジェクトそのものを表示します。ただし、ナビゲーションスタック上の個々のView Controllerが、ほかのView Controller（ほかのNavigation Controllerも含む）をモーダルモードで表示することはできます。図 5-3に、上記の例に関連するオブジェクトの詳細を示します。この図からわかるとおり、Peopleピッカーは「Photo Library」Navigation Controllerによって表示されるのではなく、そのナビゲーションスタック上のカスタムView Controllerの1つによって表示されます。

図 5-3 Navigation Controllerのモーダルモードでの表示



モーダルモードでのView Controllerの表示

View Controllerをモーダルモードで表示するには、次の手順を実行する必要があります。

1. 表示するView Controllerを作成します。
2. そのView Controllerの`modalTransitionStyle`プロパティを希望の値に設定します。
3. 必要に応じてデリゲートオブジェクトを割り当てます（このデリゲートは、View Controllerを閉じる準備が整ったことを通知するために、主にシステムView Controllerによって使用されます。詳細については、「標準のシステムモーダルView Controllerの表示」（99 ページ）を参照してください）。
4. 現在のView Controllerの`presentModalViewController:animated:`メソッドを呼び出して、モーダルモードで表示するView Controllerを渡します。

`presentModalViewController:animated:`メソッドは、指定されたView Controllerオブジェクトに対応するビューを表示して、このモーダルView Controllerと現在のView Controllerとの間に親子関係を作成します。アプリケーションを以前の状態に復元する場合を除いて、通常は、モーダルView Controllerの表示をアニメーション化します。使用するトランジションスタイルは、表示するView Controllerの使いかたによって異なります。表 5-1に、表示するView Controllerの`modalTransitionStyle`プロパティに割り当てることができるトランジションスタイルとその使用法を示します。

表 5-1 モーダルView Controllerのトランジションスタイル

トランジションスタイル	使用法
`UIModalTransition-StyleCoverVertical`	このスタイルは、ユーザから情報を収集するために現在のワークフローに割り込みを行う場合に使用します。ユーザが変更を加える可能性のあるコンテンツの表示にも使用できます。 このトランジションスタイルでは、カスタムView Controllerが、そのView Controllerを明示的に閉じるためのボタンを提供しなければなりません。通常、これは「完了(Done)」ボタンとオプションの「キャンセル(Cancel)」ボタンになります。 トランジションスタイルを明示的に設定しない場合は、デフォルトでこのスタイルが使われます。

トランジションスタイル	使用法
UIModalTransitionStyleFlipHorizontal	このスタイルは、アプリケーションの作業モードを一時的に変更するために使用します。このスタイルが最もよく使われるのは、「株価(Stocks)」アプリケーションや「天気(Weather)」アプリケーションなどで、頻繁に変更される可能性がある設定を表示する場合です。これらの設定はアプリケーション全体に影響を与えるものもあれば、現在の画面に特有のものもあります。 このトランジションスタイルでは、一般に、アプリケーションの通常の動作モードに戻るための何らかのボタンを提供します。
UIModalTransitionStyleCrossDissolve	このスタイルは、デバイスの向きが変化したときに代替のインターフェイスを表示するために使用します。このような場合に、向きの変化の通知に応答して代替のインターフェースを表示したり閉じたりするのはアプリケーションの仕事です。 メディアベースのアプリケーションでは、このスタイルを使用してメディアコンテンツを表示する画面をフェードインすることもできます。 デバイスの向きの変化に応答して代替のインターフェイスを実装する方法の例は、「代替の横長インターフェイスの作成」（45 ページ）を参照してください。

リスト 5-1に、レシピアプリケーションでモーダルView Controllerを表示する例を示します。ユーザが新しいレシピを追加すると、アプリケーションは、Navigation Ctontrollerをモーダルモードで表示して、そのレシピに関する基本情報の入力をユーザに要求します。「キャンセル(Cancel)」ボタンと「完了(Done)」ボタンを標準で置く場所を確保するために、Navigation Controllerを選択しました。また、Navigation Controllerを使用すると、新規レシピのインターフェイスを拡張することも簡単になります。デベロッパがしなければならないのは、新しいView Controllerをナビゲーションスタックにプッシュすることだけです。

リスト 5-1　View Controllerをモーダルモードで表示する

```
- (void)add:(id)sender {
    // Navigation Controller用のルートView Controllerを作成する。
    // この新しいView Controllerに、ナビゲーションバー用の
    // 「キャンセル(Cancel)」ボタンと「完了(Done)」ボタンを設定する。
    RecipeAddViewController *addController = [[RecipeAddViewController alloc]
                        initWithNibName:@"RecipeAddView" bundle:nil];
    addController.delegate = self;

    // Navigation Controllerを作成しモーダルモードで表示する。
   UINavigationController *navigationController = [[UINavigationController alloc]
                               initWithRootViewController:addController];
    [self presentModalViewController:navigationController animated:YES];

    // Navigation Controllerは現在のView Controllerに所有されている。
    // また、ルートView ControllerはNavigation Controllerに所有されている。
    // しがたって、重複保持を避けるために両方のオブジェクトを解放しなければならない。
    [navigationController release];
    [addController release];
}
```

新規レシピ入力用のインターフェイスでユーザが「完了(Done)」ボタンまたは「キャンセル(Cancel)」のいずれかをタップすると、アプリケーションはこのView Controllerを閉じてメインビューに戻ります。いずれかのボタンに関連付けられているアクションメソッドから直接`dismissModalViewControllerAnimated:`メソッドを呼び出すこともできますが、より堅牢なアプローチはデリゲートオブジェクトを使用する方法です。これについては、「モーダルView Controllerを閉じる」（98 ページ）で説明します。

モーダルView Controllerを閉じる

モーダルView Controllerを閉じるときによく使われるアプローチは、親のView Controllerに閉じさせる方法です。つまり、できる限り、そのモーダルView Controllerを表示したのと同じView Controllerが閉じるべきなのです。親のView Controllerに、モーダルモードで表示した子を閉じるように通知するにはいくつかの方法がありますが、よく使われる方法はデリゲーションです。

デリゲーションを使用するモデルでは、モーダルモードで表示されるView Controllerに、デリゲートで実装するプロトコルが定義されていなければなりません。このプロトコルには、特定のアクション（「完了(Done)」ボタンのタップなど）に応じてモーダルView Controllerから呼び出されるメソッドを定義します。これらのメソッドを実装して適切な応答を返すのは、デリゲートの仕事です。親のView Controllerがモーダルな子に対するデリゲートとしての役割を果たす場合は、必要に応じて子View Controllerを閉じることもその応答に含まれます。

このようにデリゲーションを使用してモーダルView Controllerとのやり取りを管理する方法は、次のような点でほかの方法よりも優れています。

- モーダルView Controllerを閉じる前に、デリゲートオブジェクトで、そのView Controllerでの変更を検証したり反映したりすることができます。
- モーダルView Controllerは、それを表示した親オブジェクトについて何も知る必要がないため、デリゲートを使用することによってカプセル化が促進されます。これによって、そのモーダルView Controllerをアプリケーションの別の部分で再利用できます。

デリゲートプロトコルの実装を示すために、「モーダルモードでのView Controllerの表示」（96 ページ）で使用したレシピ用のView Controllerの例を考えます。この例では、レシピアプリケーションが、ユーザからの新規レシピ追加要求に応答して、モーダルView Controllerを表示しました。このモーダルView Controllerを表示する前に、現在のView Controllerが自身を`RecipeAddViewController`オブジェクトのデリゲートにします。リスト 5-2に、`RecipeAddViewController`オブジェクトのデリゲートプロトコルの定義を示します。

リスト 5-2　モーダルView Controllerを閉じるためのデリゲートプロトコル

```
@protocol RecipeAddDelegate <NSObject>
// キャンセルの場合はrecipe == nil
- (void)recipeAddViewController:(RecipeAddViewController *)recipeAddViewController
                    didAddRecipe:(MyRecipe *)recipe;
@end
```

新規レシピ用のインターフェイスで、ユーザが「キャンセル(Cancel)」ボタンまたは「完了(Done)」ボタンをタップすると、`RecipeAddViewController`オブジェクトは、デリゲートオブジェクトの上記のメソッドを呼び出します。次に、デリゲートはどのようなアクションを取るべきかを決定します。

リスト 5-3に、新規レシピの追加を処理するデリゲートメソッドの実装を示します。このメソッドは、RecipeAddViewControllerオブジェクトをモーダルモードで表示したView Controllerで実装します。ユーザが新規レシピを受け入れた場合（レシピオブジェクトがnilでない場合）、このメソッドはそのレシピを内部のデータ構造に追加し、テーブルビューの更新を指示します（それを受けて、テーブルビューは、ここに示したものと同じrecipesControllerオブジェクトからレシピデータを再ロードします）。最後に、デリゲートメソッドはモーダルView Controllerを閉じます。

リスト 5-3　デリゲートを使用してモーダルView Controllerを閉じる

```
- (void)recipeAddViewController:(RecipeAddViewController *)recipeAddViewController
                    didAddRecipe:(Recipe *)recipe {
    if (recipe) {
        // レシピをレシピコントローラに追加する。
        int recipeCount = [recipesController countOfRecipes];
        UITableView *tableView = [self tableView];
        [recipesController insertObject:recipe inRecipesAtIndex:recipeCount];

        [tableView reloadData];
    }
    [self dismissModalViewControllerAnimated:YES];
}
```

標準のシステムモーダルView Controllerの表示

iPhone OSでは、アプリケーションからモーダルモードで表示するように設計された標準のシステムView Controllerがいくつかあります。これらのView Controllerを表示する際の基本規則は、カスタムView Controllerの場合と同じです。ただし、アプリケーションは、システムView Controllerが管理するビュー階層にはアクセスできないため、そのビュー内のコントロールに対応するアクションを単純に実装することはできません。システムView Controllerとのやり取りはすべて、デリゲートオブジェクトを介して実行しなければなりません。

各システムView Controllerには、それに対応するプロトコルが定義されています。それらのメソッドをデリゲートオブジェクトに実装します。通常、各デリゲートには、選択された項目を受け付けたり、操作をキャンセルしたりするためのメソッドを実装します。デリゲートオブジェクトでは、必ず両方のケースを処理できるようにしておくべきです。デリゲートオブジェクトで実行しなければならない最も重要な処理の1つは、表示元のView Controller（モーダルView Controllerの親）のdismissModalViewControllerAnimated:メソッドを呼び出して、表示したView Controllerを閉じることです。

表 5-2に、iPhone OSで利用できる標準のシステムView Controllerをいくつか示します。これらのクラスの詳細（その機能も含む）については、それぞれのクラスのリファレンスドキュメントを参照してください。

表 5-2　標準のシステムView Controller

フレームワーク	View Controller
UIKit	UIImagePickerController UIVideoEditorController

フレームワーク	View Controller

Address Book UI	`ABNewPersonViewController`
	`ABPeoplePickerNavigationController`
	`ABPersonViewController`
	`ABUnknownPersonViewController`

Game Kit	`GKPeerPickerController`
Message UI	`MFMailComposeViewController`
Media Player	`MPMediaPickerController`

注： Media Playerフレームワークの`MPMoviePlayerController`クラスは、技術的にはモーダルView Controllerと見なされますが、それを使用する際のセマンティクスは少し異なります。このView Controllerを表示するのではなく、View Controllerを初期化したら、そのメディアファイルを再生するように指示します。その後は、このView Controllerが、ビューの表示と消去のすべての側面を処理します。このクラスの詳細については、『*MPMoviePlayerController Class Reference*』を参照してください。

View Controllerインターフェイスの組み合わせ

UIKitフレームワークには、アプリケーションのインターフェイスを実装するための標準のView Controllerはほんのわずかしかありません。

- 1つのカスタムView Controllerは、1画面に相当するコンテンツを提供します。
- Navigation Controllerは、複数のView Controllerを階層的に表示します。
- Tab Bar Controllerは、複数のView Controllerをアプリケーションのさまざまな操作モードに応じて表示します。

これらのView Controllerを単独で使用することもできますが、ほかのView Controllerと組み合わせて使用することによって、より複雑なインターフェイスを作成できます。ただし、View Controllerを組み合わせる場合は、上記のリストでの順番が重要になります。一般的な規則は、各View Controllerは、リストの中でそのView Controllerよりも手前にあるView Controllerを組み込めるということです。したがって、Navigation ControllerはカスタムView Controllerを組み込むことができ、Tab Bar Controllerは、Navigation ControllerとカスタムView Controllerのどちらも組み込めます。一方、Navigation Controllerには、ナビゲーションインターフェイスの一部としてTab Bar Controllerを組み込むべきではありません。そのようなインターフェイスは、Tab Barの表示に一貫性がなくなるため、ユーザを混乱させます。

モーダルView Controllerは、いずれにしても何らかの割り込みを表します。このため、少し異なる規則に従います。ほとんどすべてのView Controllerを、いつでもモーダルモードで表示できます。カスタムView ControllerからTab Bar ControllerやNavigation Controllerをモーダルモードで表示しても、ほとんど混乱は生じません。たとえ一時的であるにしろ、新しいインターフェイススタイルが親のスタイルに置き換わります。

以降の各セクションでは、iPhoneアプリケーションで推奨されるタイプの複合インタフェースの作成方法を示します。

Tab Barインターフェイスへの Navigation Controllerの追加

Tab Bar Controllerを使用するアプリケーションでは、1つ以上のタブでNavigation Controllerを使用できます。同じユーザインターフェイス内でこの2種類のView Controllerを組み合わせる場合は、Tab Bar Controllerは常にNavigation Controllerのラッパーとしての役割を果たします。Tab Bar ControllerをNavigation Controllerのナビゲーションスタックにプッシュするべきではありません。そのようなことを行うと、特定のView Controllerがナビゲーションスタックの一番上にあるときだけそのTab Barが表示されるという異常な状況が生じます。Tab Barは、永続的に表示されるように設計されています。したがって、このような移り変わりのあるアプローチはユーザを混乱させる恐れがあります。

Tab Bar Controllerの最も一般的な使いかたは、アプリケーションのメインウインドウにそのビューを埋め込むことです。以降の各セクションでは、1つのTab Bar Controllerと1つ以上のNavigation Controllerを含むように、アプリケーションのメインウインドウを設定する方法を示します。ここでは、プロ

グラムよる方法とInterface Builderを使用する方法の両方の例を示します。Tab Bar Controllerをモーダルモードで表示する必要がある場合は、一般に、それに関連するオブジェクトをプログラムで作成することをお勧めします。

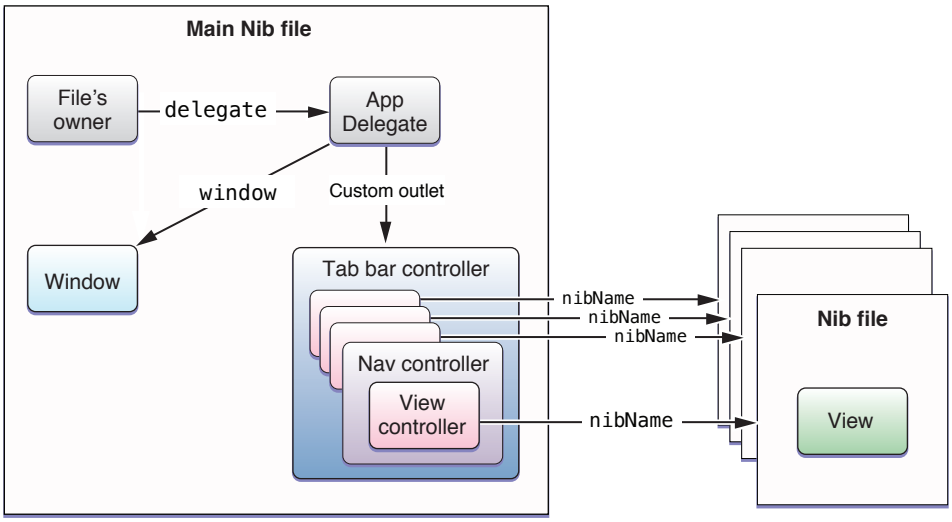
注： Tab BarインターフェイスにNavigation Controllerを埋め込む場合は、汎用の`UINavigationController`オブジェクトのみを埋め込むようにし、`UINavigationController`クラスから派生したシステムView Controllerを埋め込むべきではありません。システムは、連絡先の選択、画像の選択、およびその他の動作を実装するためのカスタムView Controllerを提供していますが、これらのView Controllerは、ほとんどの場合、モーダルモードで表示されるように設計されています。特定のView Controllerの使用法については、そのクラスのリファレンスドキュメントを参照してください。

Interface Builderでのオブジェクトの作成

Tab Bar ControllerとNavigation Controllerを組み合わせて1つのnibファイルに保存する手順は、比較的簡単です。唯一難しいのは、Tab Bar ControllerとNavigation Controllerの関係をどのように作成するかです。これらのオブジェクトを単独で使用する場合は、それぞれがアプリケーションのウインドウのルートビューの役割を果たします。一方、これらのオブジェクトを組み合わせる場合は、Tab Bar Controllerのみがこの役割を果たします。Navigation Controllerは、ウインドウのルートビューを提供するのではなく、Tab Barインターフェイス内の1つのタブのルートビューとしての役割を果たします。

図 6-1に、nibファイル内に作成するオブジェクトの構成を示します。この例では、Tab Barインターフェイスの最初の3つのタブでカスタムView Controllerを使用し、最後のタブでNavigation Controllerを使用します。その後、このNavigation Controllerのルートビューとしての役割を果たすもう1つView Controllerを追加しています。メモリの管理をしやすくするために、各カスタムView Controller（Navigation Controllerのルート View Controllerを含む）は、それに対応するビューを別のnibファイルに保存しています。

図 6-1 Navigation ControllerとTab Bar Controllerを1つのnibファイルにまとめる



汎用のメインnibファイル（Tab Bar Controllerを含んでいないファイル）をもとに作業を開始した場合は、Interface Builderで図 6-1 （102 ページ）のオブジェクトを作成するために次の手順を実行します。

1. Tab Bar Controllerオブジェクトをライブラリから Interface Builderのドキュメントウインドウにドラッグします。

   Tab Bar Controllerをnibファイルに追加すると、Interface Builderによって、1つのTab Barビュー、2つのルートView Controller、および2つのTab Bar Item（View Controllerごとに1つ）も追加されます。

2. アウトレットを使用して、Tab Bar Controllerへの参照を保存します。

   実行時にこのTab Bar Controllerにアクセスするには、アウトレットを使用するか、nibファイルをロードするときにそのnibファイルの最上位レベルのオブジェクトを明示的に取得する必要があります。一般には、アウトレットを使用する方がはるかに簡単です。Tab Bar Controller用とウインドウ用の両方のアウトレットを追加するには、アプリケーションデリゲートのヘッダファイルに次のようなコードを含める必要があります。

   ```
   @interface MyAppDelegate :NSObject <UIApplicationDelegate> {
      UITabBarController*  myTabBarController;
      UIWindow *window;
   }
   @property (nonatomic, retain) IBOutlet UIWindow *window;
   @property (nonatomic, retain) IBOutlet UITabBarController *tabBarController;
   @end
   ```

   アウトレットの定義を追加したら、そのアウトレットからTab Bar Controllerオブジェクトへの接続を作成します。

3. アプリケーションデリゲートクラスの実装ファイルに次のコードを追加し、前のステップをもとにプロパティを合成します。

   ```
   @synthesize window;
   @synthesize tabBarController;
   ```

4. 1つのView Controllerオブジェクトと1つのNavigation ControllerオブジェクトをTab Bar Controllerに追加します。

   Tab Bar Controllerオブジェクトに含まれるView Controllerの数によって、Tab Barインターフェイスに表示されるタブの数が決まります。初期のTab Bar Controllerにはすでに2つの汎用View Controllerが含まれているため、もう1つのView Controllerオブジェクト(`UIViewController`)と 1つのNavigation Controllerオブジェクト(`UINavigationController`)を追加する必要があります。

   View Controllerを追加するには、次のいずれかを実行します。

   - 適切なオブジェクトをライブラリから編集サーフェイス内のTab Barにドラッグします。
   - オブジェクトをライブラリから、Interface Builderのドキュメントウインドウ内のTab Bar Controllerにドラッグします。このウインドウはアウトラインモードになっていなければなりません。

Navigation Controllerを追加する場合は、適切なオブジェクトをライブラリからドラッグするか、Tab Bar Controllerを選択し、「Attributes」インスペクタを使用してView Controllerの種類を設定します。どちらを選択しても、適切な種類のView Controllerオブジェクトがnibファイルに追加されます。汎用のView Controllerオブジェクトをnibファイルにドラッグして、そのクラス名を希望のクラスタイプに変更することによって、Navigation Controllerを追加するべきではありません。

View Controllerを削除するには、編集サーフェスまたはドキュメントウインドウでView Controllerオブジェクトを選択してDeleteキーを押します。

5. Tab Barインターフェイスに表示する順番にView Controllerを並べます。

   Tab Bar Controllerの編集サーフェスに表示されているタブをドラッグするか、Interface Builderのドキュメントウインドウで View Controllerをドラッグすることによって（アウトラインモードの場合のみ）、View Controller（およびそれに対応するタブ）を並べ替えることができます。編集サーフェスにはすべてのタブが表示されますが、実行時には5つしか表示されません。Tab Bar Controllerに6つ以上のView Controllerが含まれる場合は、初期状態ではTab Barには最初の4つしか表示されません。Tab Barの最後の場所は、残りのView Controllerを表示する「More」View Controller用に予約されています。

6. View Controllerを設定します。

   各ルートView Controllerには、次のような属性を設定しなければなりません。

   - 「Identity」インスペクタを使用して、各カスタムView Controllerオブジェクトのクラスを設定します。汎用のView Controllerオブジェクトの場合は、クラス名をそのタブのコンテンツを表示するために使用するカスタムサブクラスに変更します。Navigation Controllerオブジェクト自身のクラスは変更せずに、そのNavigation Controllerに埋め込まれているカスタムView Controllerのクラスを設定します。
   - 各カスタムView Controllerにビューを設定します。そのためによく使われるのは、各カスタムView ControllerのNIB Name属性を、ビューを含むnibファイルの名前に設定する方法です。各ビューをView Controllerと同じnibファイルに含めることもできますが、それはお勧めできません。カスタムView Controller用のnibファイルの作成方法については、「分離nibファイルにビューを保存する」（29 ページ）を参照してください。
   - 必要に応じて、View Controllerのスタイルや外観情報を設定します。

7. 各View ControllerにTab Bar Itemを設定します。

   Tab Bar Controllerの編集サーフェス、またはInterface Builderのドキュメントウインドウ（アウトラインモードまたはブラウザモードの場合）からTab Bar Itemを選択できます。Interface Builderを使用して、Tab Bar Itemのタイトル、画像、およびバッジを指定できます。あるいは、Tab Bar Itemを標準的なシステムタブの1つに設定することもできます。それには、「Attributes」インスペクタでIdentifierプロパティに値を割り当てます。

8. nibファイルを保存します。

上記の手順でTab Barインターフェイスの設定はできますが、アプリケーションのメインウインドウへのインストールはできません。それには、アプリケーションデリゲートの`applicationDidFinishLaunching:`メソッドに、リスト 6-1に示すようなコードを追加する必要があります。ここで、アプリケーションデリゲートに作成したTab Bar Controller用のアウトレットを使用します。

リスト 6-1　アプリケーションのウインドウへの複合インターフェイスのインストール

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    [window addSubview:myTabBarController.view];
}
```

プログラムによるオブジェクトの作成

Tab Barとナビゲーションを組み合わせたインターフェイスをプログラムによってアプリケーションのメインウインドウに作成する場合、そのために最適な場所は、アプリケーションデリゲートの`applicationDidFinishLaunching:`メソッド内です。以下の手順に、カスタムView Controllerを含むタブが3つと、Navigation Controllerを含むタブが1つ存在する複合インターフェイスを作成する方法を示します。

1. `UITabBarController`オブジェクトを作成します。
2. タブごとに1つ、計3つのカスタムルートView Controllerを作成します。
3. ナビゲーションインタフェースのルートView Controllerとしての役割を果たすカスタムView Controllerをもう1つ作成します。
4. `UINavigationController`オブジェクトを作成し、ルートView Controllerで初期化します。
5. このNavigation Controllerと3つのカスタムView ControllerをTab Bar Controllerの`viewControllers`プロパティに追加します。
6. Tab Bar Controllerのビューをアプリケーションのメインウインドウに追加します。

リスト4-1に、Tab Barインターフェイスのタブとして、3つのカスタムView Controllerと1つのNavigation Controllerを作成するテンプレートコードを示します。このカスタムView Controllerのクラス名は、独自に作成するクラス用のプレースホルダです。各カスタムView Controllerの`init`メソッドは、そのView Controllerを初期化するために用意するメソッドです。`tabBarController`変数と`window`変数は、これらの値を保持するクラスの宣言済みプロパティです。

リスト 6-2　Tab Bar Controllerをゼロから作成する

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
   tabBarController = [[[UITabBarController alloc] init] autorelease];

   MyViewController1* vc1 = [[[MyViewController1 alloc] init] autorelease];
   MyViewController2* vc2 = [[[MyViewController2 alloc] init] autorelease];
   MyViewController3* vc3 = [[[MyViewController3 alloc] init] autorelease];
   MyNavRootViewController* vc4 = [[[MyNavRootViewController alloc] init]
autorelease];
   UINavigationController* navController = [[[UINavigationController alloc]
                           initWithRootViewController:vc4] autorelease];

  NSArray* controllers = [NSArray arrayWithObjects:vc1, vc2, vc3, navController,
 nil];
   tabBarController.viewControllers = controllers;

   // Tab Bar Controllerの現在のビューをウインドウのサブビューとして追加する
   window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]]
autorelease];
```

```
    [window addSubview:tabBarController.view];
}
```

注： 上の例では、便宜上、オブジェクトを直接解放せずに自動解放にしています。`applicationDidFinishLaunching:`メソッドはアプリケーションでは一度しか呼び出されないため、この時点でオブジェクトを自動解放しても、パフォーマンスに大きな影響を与えることはありません。しかし、アプリケーションのパフォーマンスへの影響が顕著な場合には、このメソッドの最後で各オブジェクトを直接解放します。

プログラムによってカスタムView Controllerを作成する場合でも、各View Controller用のビューの作成方法に制限はありません。ビューの作成方法に関わらず、View Controller用のビューの管理サイクルは同じです。したがって、プログラムによる場合も、Interface Builderを使用する場合も「View Controller用のビューの作成」（29 ページ）での説明に従ってビューを作成できます。

モーダルモードでのNavigation Controllerの表示

アプリケーションからNavigation Controllerをモーダルモードで表示することは、まったく合理的です（また、比較的よく行うことです）。実際に、標準のシステムView Controllerの多くは（`UIImagePickerController`と`ABPeoplePickerNavigationController`も含む）、モーダルモードで表示するために特別に設計されたNavigation Controllerです。

独自のナビゲーションインターフェイスをモーダルモードで表示する場合は、必ず`presentModalViewController:animated:`メソッドの第1パラメータにNavigation Controllerオブジェクトを渡します。View Controllerを表示する前に、必ずそれを適切に設定しなければなりません。最低限、Navigation ControllerはルートView Controllerを持っていなければなりません。また、ナビゲーション階層の別の位置からユーザをスタートさせたい場合は、Navigation Controllerを表示する前に、これらのView Controllerをナビゲーションスタックに（アニメーションなしで）追加しなければなりません。

リスト6-3に、Navigation Controllerを作成して設定し、モーダルモードで表示する方法の例を示します。この例では、ナビゲーションスタックにプッシュされるView Controllerはカスタムオブジェクトです。このオブジェクトを定義してビューを設定する必要があります。`currentViewController`オブジェクトは、現在表示されているView Controllerへの参照です。これも提供する必要があります。

リスト 6-3　モーダルモードでのNavigation Controllerの表示

```
MyViewController1*  rootVC = [[MyViewController1 alloc] init];
MyViewController2*  nextVC = [[MyViewController2 alloc] init];

// Navigation Controllerを作成してView Controllerを追加する。
UINavigationController*  theNavController = [[UINavigationController alloc]
                         initWithRootViewController:rootVC];
[theNavController pushViewController:nextVC animated:NO];

// Navigation Controllerをモーダルモードで表示する。
[currentViewController presentModalViewController:theNavController animated:YES];

// 重複保持を避けるためにView Controllerを解放する。
[rootVC release];
[nextVC release];
[theNavController release];
```

モーダルモードで表示されるほかのすべてのView Controllerと同様に、適切なユーザアクションに応答して、モーダルモードで表示された子のView Conctrollerを消去するのは、親のView Controllerの仕事です。ただし、Navigation Controllerを消去すると、そのNavigation Controllerオブジェクトだけでなく、現在ナビゲーションスタック上に存在するView Controllerも削除されることを忘れないでください。表示されていないView Controllerは単純に解放されますが、一番上のView Controllerは、通常どおり`viewWillDisappear:`メッセージを受信します。

注： Navigation Controllerをモーダルモードで表示する場合は、通常、プログラムによってNavigation Controllerの作成と設定を行う方が簡単です。Interface Builderを使用してNavigation Controllerの作成と設定を行うこともできますが、一般に、これはお勧めできません。

モーダルモードでView Controller（Navigation Controllerを含む）を表示する方法については、「モーダルモードでのView Controllerの表示」（96 ページ）を参照してください。アプリケーションで使用するためにNavigation Controllerを設定する方法の詳細については、「ナビゲーションインターフェイスの作成」（60 ページ）を参照してください。

モーダルモードでのTab Bar Controllerの表示

（一般的ではありませんが）アプリケーションでTab Bar Controllerをモーダルモードで表示することもできます。Tab Barインターフェイスは、通常はアプリケーションのメインウインドウにインストールされ、必要な場合にだけ更新されます。ただし、インターフェイスの設計において必要と認められる場合には、Tab Bar Controllerをモーダルモードで表示することも可能です。たとえば、アプリケーションの主たる操作モードを、Tab Barインターフェイスを使用したまったく別のモードに切り替えるには、クロスフェードトランジションを使用して、第2のTab Bar Controllerをモーダルモードで表示します。

Tab Bar Controllerをモーダルモードで表示する場合は、必ず`presentModalViewController:animated:`メソッドの第1パラメータにTab Bar Contrllerオブジェクトを渡します。このTab Bar Controllerは、表示する前にすでに設定されていなければなりません。つまり、Tab Barインターフェイスをメインウインドウにインストールする場合とまったく同様に、ルートView Controllerを作成して設定し、それらをTab Bar Controllerに追加しなければなりません。

モーダルモードで表示されるほかのすべてのView Controllerと同様に、適切なユーザアクションに応答して、モーダルモードで表示された子のView Conctrollerを消去するのは、親のView Controllerの仕事です。ただし、Tab Bar Controllerを消去すると、そのTab Bar Controllerオブジェクトだけでなく、各タブに関連付けられたView Controllerも削除されることを忘れないでください。表示されていないView Controllerは単純に解放されますが、現在表示されているタブのView Controllerは、通常どおり`viewWillDisappear:`メッセージを受信します。

注： Navigation Controllerをモーダルモードで表示する場合は、通常、プログラムによってNavigation Controllerの作成と設定を行う方が簡単です。Interface Builderを使用してNavigation Controllerの作成と設定を行うこともできますが、一般に、これはお勧めできません。

モーダルモードでView Controller（Navigation Controllerを含む）を表示する方法については、「モーダルモードでのView Controllerの表示」（96 ページ）を参照してください。Tab Bar Controllerをプログラムによって設定する方法については、「プログラムによるTab Barインターフェイスの作成」（87 ページ）を参照してください。

ナビゲーションインターフェイスでのTable View Controllerの使用

テーブルビューとNavigation Controllerを組み合わせてナビゲーションインターフェイスを作成することは非常に一般的です。Navigation Controllerはデータ階層のナビゲーションをサポートします。このため、テーブルは、次の移動先の選択肢をユーザに提示するためにしばしば使われます。テーブル内の行をタップすると、その行に対応するデータを表示する新しい画面に切り替わります。たとえば、「iPod」アプリケーションのプレイリストを選択すると、そのプレイリストに含まれる曲のリストがユーザに表示されます。

テーブルの管理と言えば、その方法の1つに`UITableViewController`オブジェクトの利用があります。このクラスを利用すると、テーブル形式のデータの管理がかなり簡単になりますが、それでもナビゲーションをサポートするためのカスタムコードを実装する必要はあります。具体的に言うと、ユーザがテーブル内の行をタップしたときには、適切な新規のView Controllerオブジェクトをナビゲーションスタックにプッシュする必要があります。

リスト 6-4に、ナビゲーションインターフェイスで次のレベルのデータに移動する方法の例を示します。ユーザが現在のテーブル内の特定の行をタップすると、その行に対応する情報を使用して新規のView Controllerを初期化します。次に、このView Controllerをナビゲーションスタックにプッシュします。`initWithTable:andDataAtIndexPath:`メソッドは、デベロッパが自分で実装しなければならないカスタムメソッドです。その仕事は、行に対応するデータオブジェクトを取得して、それを使用して次のレベルのView Controllerを初期化することです。

リスト 6-4　テーブルビューを使用したデータのナビゲーション

```
// UITableViewControllerのサブクラス、またはテーブルの管理に使用するデリゲートオブジェクト
に
// 次のようなコードを実装する。
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
 *)indexPath
{
   // ナビゲーションタイトルと同じタイトルを持つ
   // View Controllerを作成して、それをプッシュする。
   NSUInteger row = indexPath.row;
   if (row != NSNotFound)
   {
      // View Controllerを作成して、
      // それを次のレベルのデータで初期化する。
      MyViewController *viewController = [[MyViewController alloc]
                  initWithTable:tableView andDataAtIndexPath:indexPath];
      [[self navigationController] pushViewController:viewController
                  animated:YES];
   }
}
```

テーブルの管理およびTable View Controllerの使いかたの詳細については、『*Table View Programming Guide for iPhone OS*』を参照してください。

書類の改訂履歴

この表は「*iPhone OS View Controller プログラミングガイド*」の改訂履歴です。

日付	メモ
2009-10-19	iPhone OS 3.0の変更に対応するため、文書の書き直しと拡張を行いました。
2009-05-28	iPhone OS 3.0をサポートしていないことについての注記を追加しました。
2010-03-04	回転するアプリケーションの詳細情報を拡張しました。
2008-10-15	『iPhone OS Programming Guide』への古くなった参照を更新しました。
2008-09-09	誤植を修正しました。
2008-06-23	View Controllerを使用してラジオインターフェイス、ナビゲーションインターフェイス、およびモーダルインターフェイスを実装する方法を説明する新規文書。

書類の改訂履歴

用語解説

コンテナView Controller (container view controller) 特定のタイプのユーザインターフェイスを表示するために、ほかのView Controllerとのやり取りを調整するView Controller。

カスタムView Controller (custom view controller) 何らかのコンテンツを画面に表示するために特別に定義したView Controller。

分離nibファイル(detached nib file) 特定のView Controller用のビューを含むが、そのView Controllerオブジェクトそのものは含まないnibファイル。代わりに、View Controllerは別のnibファイルからロードするか、プログラムによって作成します。

統合nibファイル(integrated nib file) View Controllerとそれに対応するビューの両方を含むnibファイル。

モーダルView Controller (modal view controller) 特殊なタイプのトランジションを使用して、現在のView Controllerの前面に表示されるView Controller。モーダルView Controllerは、通常、ユーザから情報を収集するために使いますが、それ以外の目的で使うこともできます。

「More」View Controller (More view controller) システムが提供するView Controllerの1つで、Tab Barインターフェイスに入りきらないタブの表示を管理する。このView Controllerは、タブの数が一度に画面に表示できる数を超えたときにのみ表示されます。

ナビゲーションインターフェイス(navigation interface) Navigation Controllerのビューによって表示されるインターフェイスのスタイル。ナビゲーションインターフェイスには、異なる画面間の移動をサポートするナビゲーションバーが画面の上端に沿って表示されます。

ナビゲーションスタック(navigation stack) Navigation Controllerによって現在管理されているView Controllerのリスト。スタック上のView Controllerは、ナビゲーションインターフェイスに現在表示されているコンテンツを表します。

ルートView Controller (root view controller) ナビゲーションインターフェイスやTab Barインターフェイスに最初に表示されるView Controller。ルートView Controllerは、インターフェイスのアンカーとして役割を果たします。ナビゲーションインターフェイスやTab Barインターフェイス内のほとんどのView Controllerは動的に追加したり削除したりできますが、インターフェイスのルートView Controllerは、初期化時に追加した後は削除できません。

Tab Barインターフェイス(tab bar interface) Tab Bar Controllerのビューによって表示されるインターフェイスのスタイル。Tab Barインターフェイスには、画面の下端に1つ以上のタブが表示されます。タブをタップすると、現在表示されている画面のコンテンツが変化します。

View Controller `UIViewController`クラスから派生したオブジェクト。View Controllerは、一連のビューとそれらのビューによって表示されるカスタムデータとのやり取りを調整します。