

# iOS View Controller カタログ

# 目次

## **View Controllerの概要** 7

はじめに 7

Navigation Controllerは他のView Controllerのスタックを管理する 8

Tab Bar Controllerは相互に依存しない一連のView Controllerを管理する 8

Page View ControllerはView Controllerのページング表示を管理する 8

Split View Controllerは2つのペインの情報を管理する 9

Popoverはコンテンツをフローティングビューで表示する 9

View Controllerを組み合わせて、より高度なインターフェイスを構築する 9

必要事項 9

関連項目 10

## **Navigation Controller** 11

ナビゲーションインターフェイスの構造 11

ナビゲーションインターフェイスのオブジェクト 13

ナビゲーションインターフェイスの作成 15

ナビゲーションインターフェイス用のコンテンツView Controllerの定義 16

ストーリーボードを使用したナビゲーションインターフェイスの作成 17

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 17

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 18

ナビゲーションスタックの変更 20

ナビゲーションスタックの変化の監視 22

Navigation Barの外観のカスタマイズ 23

Navigation Itemオブジェクトの設定 23

Navigation Barの表示と非表示 25

Navigation Barオブジェクトを直接変更する 26

「編集 (Edit)」 / 「完了 (Done)」 ボタンの使用 29

ナビゲーションツールバーの表示 30

ツールバー項目の指定 31

ツールバーの表示と非表示 32

## **Tab Bar Controller** 33

Tab Barインターフェイスの構造 33

Tab Barインターフェイスのオブジェクト 35

Tab Barインターフェイスの作成 37

Tab Barインターフェイス用のコンテンツView Controllerの定義	37
ストーリーボードを使用したTab Barインターフェイスの作成	38
プログラムによるTab Barインターフェイスの作成	39
プログラム内でのTab Bar Itemの作成	40
実行時のタブの管理	40
タブの追加と削除	41
タブの選択の禁止	41
ユーザによるタブの変更の監視	42
タブのカスタマイズの禁止	42
タブのバッジの変更	44
Tab Bar Controllerとビューの回転	45
Tab Barとフルスクリーンレイアウト	45
 <b>Page View Controller</b>	46
Page View Controllerインターフェイスの構造	46
Page View Controllerインターフェイスのオブジェクト	47
Page View Controllerインターフェイスの作成	48
ストーリーボードを使用したPage View Controllerインターフェイスの作成	48
プログラムによるPage View Controllerインターフェイスの作成	49
初期View Controllerの設定	49
初期化時の動作のカスタマイズ	49
デリゲートによる実行時の動作のカスタマイズ	50
データソースの提供によるコンテンツの指定	50
現在のView Controllerの設定によるコンテンツの指定	51
Right-to-LeftとBottom-to-Topコンテンツの特別な考慮事項	51
 <b>Split View Controller</b>	53
ストーリーボードを使用したSplit View Controllerの作成	54
プログラムによるSplit View Controllerの作成	54
Split Viewでの向きの変更のサポート	55
 <b>Popover</b>	56
Popoverの作成と表示	59
Popover用のデリゲートの実装	61
アプリケーションでPopoverを管理する際のヒント	61
 <b>View Controllerインターフェイスの組み合わせ</b>	62
Tab BarインターフェイスへのNavigation Controllerの追加	62
ストーリーボードを使用したオブジェクトの作成	63
プログラムによるオブジェクトの作成	63

モーダルモードでのNavigation Controllerの表示 65  
モーダルモードでのTab Bar Controllerの表示 66  
ナビゲーションインターフェイスでのTable View Controllerの使用 67

書類の改訂履歴 69

# 図、表、リスト

## **Navigation Controller** 11

- 図 1-1 ナビゲーションインターフェイスのビュー 12
- 図 1-2 Navigation Controllerが管理するオブジェクト 13
- 図 1-3 ナビゲーションスタック 14
- 図 1-4 データのレベルごとにView Controllerを定義する 16
- 図 1-5 スタックが変化する間に送信されるメッセージ 22
- 図 1-6 Navigation Barに関連するオブジェクト 23
- 図 1-7 Navigation Barの構造 25
- 図 1-8 Navigation Barのスタイル 26
- 図 1-9 Navigation Barのカスタムボタン 28
- 図 1-10 ナビゲーションインターフェイスのツールバー項目 30
- 図 1-11 ツールバーの中央に配置されたSegmented Control 31
- 表 1-1 ナビゲーションスタックを管理する際の選択肢 20
- 表 1-2 Navigation Bar上の項目の位置 24
- リスト 1-1 プログラム内でのNavigation Controllerの作成 18
- リスト 1-2 カスタムBar Button Itemの作成 28
- リスト 1-3 中央にSegmented Controlを持つツールバーの設定 31

## **Tab Bar Controller** 33

- 図 2-1 Tab Barインターフェイスのビュー 34
- 図 2-2 Tab Bar Controllerとそれに関連付けられたView Controller 35
- 図 2-3 「iPod」アプリケーションのTab Bar Item 36
- 図 2-4 「時計 (Clock)」アプリケーションのタブ 38
- 図 2-5 「iPod」アプリケーションのTab Barの設定 43
- 図 2-6 Tab Bar Itemのバッジ 44
- リスト 2-1 Tab Bar Controllerをゼロから作成する 39
- リスト 2-2 View ControllerのTab Bar Itemの作成 40
- リスト 2-3 現在のタブの削除 41
- リスト 2-4 タブの選択の禁止 42
- リスト 2-5 タブのバッジの設定 44

## **Page View Controller** 46

- 図 3-1 Page Viewインターフェイスのビュー 47
- 図 3-2 Page View Controllerとその関連オブジェクト 48

リスト 3-1 Page View Controllerのカスタマイズ 50

**Split View Controller** 53

図 4-1 Split Viewインターフェイス 53

リスト 4-1 プログラムによるSplit View Controllerの作成 55

**Popover** 56

図 5-1 Popoverを使用したマスタペインの表示 58

リスト 5-1 プログラムによるPopoverの表示 60

**ViewControllerインターフェイスの組み合わせ** 62

リスト 6-1 プログラムによるTab Bar Controllerの作成 64

リスト 6-2 モーダルモードでのNavigation Controllerの表示 65

リスト 6-3 テーブルビューを使用したデータのナビゲーション 67

# View Controllerの概要

View Controllerは、一連のビューの表示および管理を行います。UIKitフレームワークにはView Controller用のクラスが含まれており、これを使用すると、iOSで一般的な対話機能の多くを設定できます。これらのView Controllerは、アプリケーションのユーザインターフェイスの構築に必要となるすべてのカスタムView Controllerと併用します。このドキュメントでは、UIKitフレームワークから提供されるView Controllerの使用方法について説明します。



## はじめに

アプリケーションで使用するView Controllerに慣れてください。カスタムView Controllerを使ってアプリケーションを最初から構築することもできますが、提供されるView Controllerを使用すると、コードの記述量が少なくなり、ユーザ体験を一定の水準に維持するのにも役立ちます。

---

**注意:** このドキュメントでは、iOS 5以降のアプリケーションを開発していること、またストーリーボードとARCを使用していることを前提としています。

---

## Navigation Controllerは他のView Controllerのスタックを管理する

Navigation Controllerは、開発するアプリケーション内でそのまま使用するUINavigationControllerクラスのインスタンスです。構造化されたコンテンツが含まれるアプリケーションは、Navigation Controllerを使用して各レベルのコンテンツ間を行き来できます。Navigation Controller自体は、1つ以上のカスタムView Controllerの表示を管理し、各カスタムView Controllerは、データ階層の特定のレベルにあるデータを管理します。また、Navigation Controllerには、このデータ階層内の現在の位置を特定し、階層内で元の位置に戻るためのコントロールも用意されています。

---

関連する章: “[Navigation Controller](#)” (11 ページ)

---

## Tab Bar Controllerは相互に依存しない一連のView Controllerを管理する

Tab Bar Controllerは、アプリケーションで今まで使用するUITabBarControllerクラスのインスタンスです。アプリケーションはTab Bar Controllerを使用して、任意の数のカスタムビューとView Controllerから成る独立した複数のインターフェイスを管理します。また、Tab Bar Controllerは、ユーザがタップすると、現在選択されているインターフェイスが変更される、Tab Barビューとのやり取りも管理します。たとえば、iPhoneまたはiPod touch用のiPodアプリケーションでは、各タブがユーザのミュージックやメディアのさまざまな表示方法を表すTab Barインターフェイスを使用しています。

---

関連する章: “[Tab Bar Controller](#)” (33 ページ)

---

## Page View ControllerはView Controllerのページング表示を管理する

Page View Controllerは、アプリケーション内でそのまま使用するUIPageViewControllerクラスのインスタンスです。アプリケーションは、Page View Controllerを使用してコンテンツのページングビューを表示します。Page View Controller自体は、各ページにコンテンツを表示する1つまたは複数のコンテンツView Controllerの表示を管理します。Page View Controllerはまた、コンテンツ内のナビゲートに使用されるGestureRecognizerとしても機能します。

---

関連する章: “[Page View Controller](#)” (46 ページ)

---

## [Split View Controller](#)は2つのペインの情報を管理する

[Split View Controller](#)は、アプリケーション内でそのまま使用する `UISplitViewController` クラスのインスタンスです。アプリケーションは、[Split View Controller](#)を使用して2つのペインの情報を管理します。すなわちインターフェイスの2つの部分が、[View Controller](#)により管理されます。これは[Navigation Controller](#)に似たインターフェイスですが、iPadの大型の画面を活用して、一度に多くのコンテンツを表示できます。

---

関連する章: “[Split View Controller](#)” (53 ページ)

---

## [Popover](#)はコンテンツをフローティングビューで表示する

`UIPopoverController` クラスは、アプリケーションの [View Controller](#) と連動してコンテンツをフローティングビューで表示します。このインターフェイスは iPad の大型の画面を活用して、大容量の本文のコンテキスト内で少量のコンテンツを一時的に表示します。

---

関連する章: “[Popover](#)” (56 ページ)

---

## [View Controller](#)を組み合わせて、より高度なインターフェイスを構築する

もっともシンプルなアプリケーションを除いて、複数の [View Controller](#) を連携させるのが一般的な方法です。[Navigation Controller](#)、[Tab Bar Controller](#)、[Split View Controller](#) は、常にほかの [View Controller](#) と連携し、カスタム [View Controller](#) でさえも、場合によってはほかの [View Controller](#) を表示する必要が生じます。ただし、[View Controller](#) の組み合わせかたによっては、ほかの組み合わせよりもうまく動作することがあります。簡単でナビゲートしやすいユーザインターフェイスを作成するには、[View Controller](#) を意味のある方法で組み合わせることが重要です。

---

関連する章: “[View Controller](#)インターフェイスの組み合わせ” (62 ページ)

---

## 必要事項

『[View Controller Programming Guide for iOS](#)』には、iOSでの [View Controller](#) のデザインパターンと一般的な用法の解説があります。

『*iOS App Programming Guide*』では、開発プロセスを紹介し、主要なアーキテクチャを解説しています。

『*App Distribution Guide*』では、アプリケーションを設定、ビルト、デバッグ、調整する手順とアプリケーションをApp Storeに登録する手順を解説しています。

## 関連項目

iOSアプリケーションの設計方法については、『*iOS Human Interface Guidelines*』を参照してください。

このドキュメントで取り上げるViewControllerクラスの詳細については、『*UIKit Framework Reference*』を参照してください。

# Navigation Controller

Navigation Controllerは、View Controllerのスタックを管理し、階層コンテンツ用のドリルダウン型のインターフェイスを提供します。Navigation Controllerのビュー階層は、自己完結型です。この階層は、Navigation Controllerが直接管理するビューと、作成中に指定した各コンテンツView Controllerで管理されるビューから構成されます。コンテンツView Controllerはそれぞれ別個のビュー階層を管理し、Navigation Controllerは各ビュー階層の間のナビゲーションを調整します。

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（画面の一番上にあるビューであり、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。

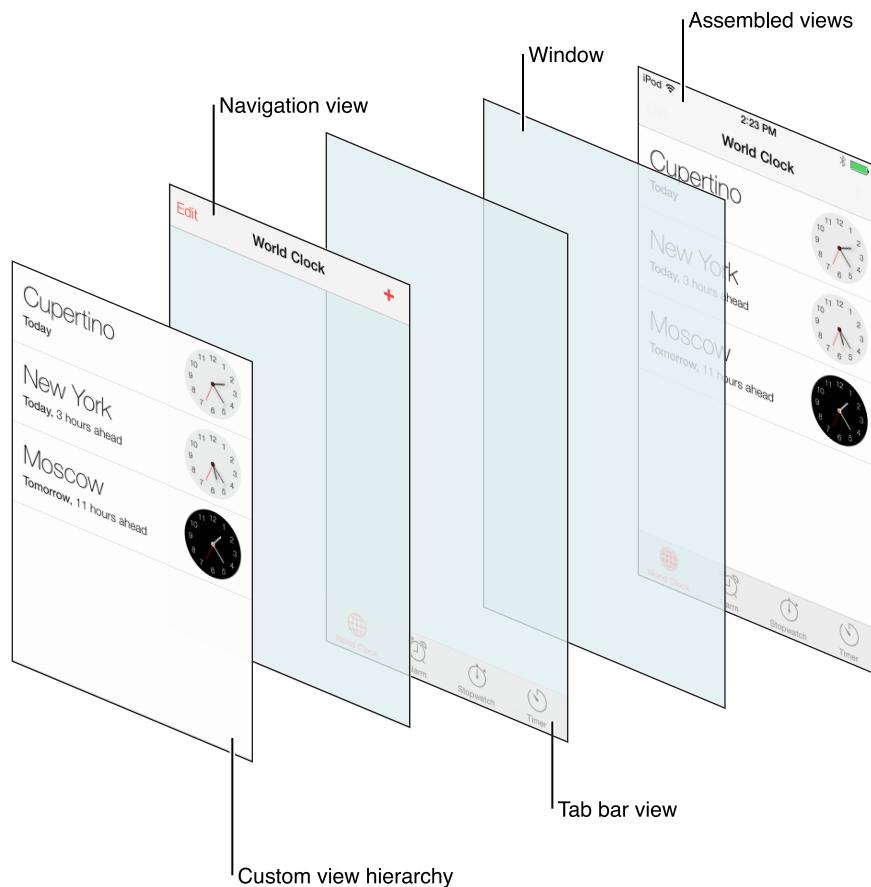
この章では、アプリケーションでNavigation Controllerを設定し、使用する方法について説明します。Navigation Controllerをほかの種類のView Controllerオブジェクトと組み合わせて使用する方法については、“[View Controllerインターフェイスの組み合わせ](#)”（62 ページ）を参照してください。

## ナビゲーションインターフェイスの構造

Navigation Controllerの主な仕事はコンテンツView Controllerの表示を管理することですが、Navigation Controller独自の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できる複数のボタンを含むNavigation Barを表示します。Navigation Controllerは、必要に応じてナビゲーションツールバーのビューを表示し、ここにカスタムボタンを含めることもできます。

図1-1に、ナビゲーションインターフェイスを示します。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。このインターフェイス内のほかのすべてのビューは、Navigation Controllerで管理される不透明なビュー階層に含まれます。

図 1-1 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、

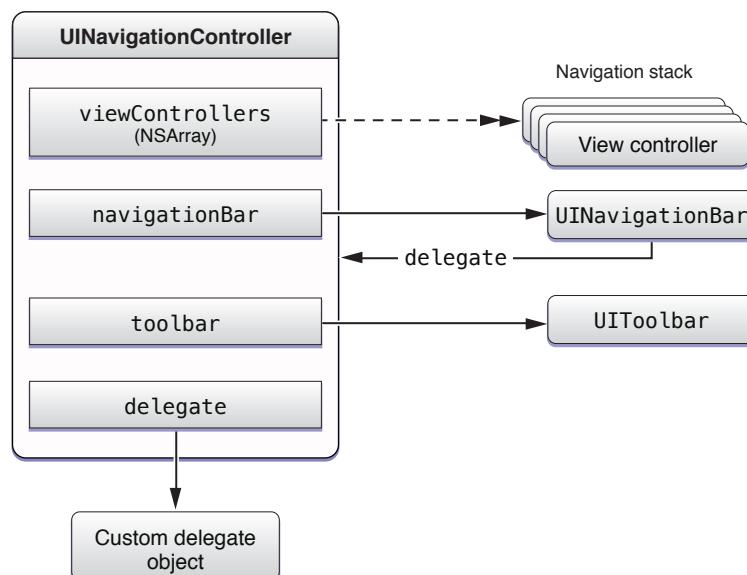
`UINavigationController`クラスと`UIViewController`クラスのメソッドを利用する方法です。

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](#)”（23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](#)”（30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一部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自体で作成されます。具体的に言うと、表示するコンテンツを View Controller に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、NavigationBarやツールバーなどのビューを作成し、それらのビューの管理に責任を負います。図 1-2に、Navigation Controllerと主要なオブジェクトとの関係を示します。

図 1-2 Navigation Controllerが管理するオブジェクト

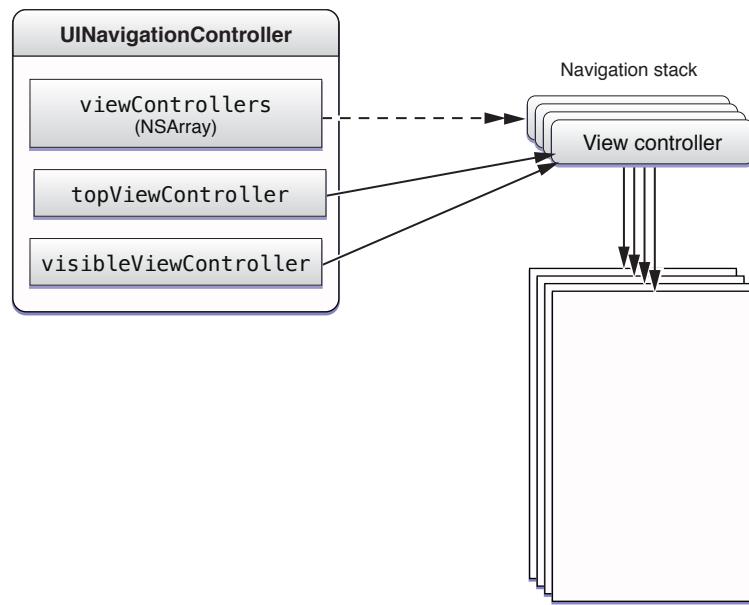


Navigation Controllerに関連したナビゲーションバーおよびツールバーのオブジェクトでは、外観と動作を一部だけカスタマイズできます。これらの設定や表示を担当するのは、Navigation Controllerだけです。さらに、Navigation Controllerオブジェクトは、自動的にそれ自体を UINavigationBarオブジェクトのデリゲートとして割り当て、ほかのオブジェクトがその関係を変更できないようにします。

ナビゲーションスタックでは、デリゲートやほかのView Controllerを修正できます。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 Controller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が ルート View Controller になり、スタックからポップされることはありません。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 1-3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（最上位のView Controllerと表示されるView Controllerは必ずしも一致しません。たとえば、View Controllerをモーダルモードで表示する場合、visibleViewControllerプロパティの値は、表示されたモーダルView Controllerを反映して変更されますが、topViewControllerプロパティは変更されません。）

図 1-3 ナビゲーションスタック



Navigation Controllerの主な役割は、新しいコンテンツView Controllerをスタックにpuschするか、コンテンツView Controllerをスタックからpopして、ユーザアクションに応答することです。ナビゲーションスタックにpusch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、ユーザが表示中のビューのアイテムを選択する場合、そのアイテムの詳細を含むView Controllerを設定し、ナビゲーションスタックにpuschします。たとえば、ユーザが「写真 (Photos)」アプリケーションからフォトアルバムを選択すると、そのアルバム内の写真を表示するView Controllerがpuschされます。

このプロセスは、スタック内の各コンテンツView Controllerがそのスタック内で最上位のコンテンツView Controllerを設定してpuschするという単純なデザインパターンに準拠しています。特定のクラスのインスタンスを使ってスタックにpuschされるというパターンに、View Controllerが陥るのを防ぐ必要があります。この場合、View Controllerをpopする間に下位のスタックにデータを渡すために、スタックの下位にあるView Controllerを、その上にあるView Controllerのデリゲートとして設定します。

ほとんどの場合、プログラムによってView Controllerをスタックからpopする必要はありません。代わりに、Navigation Controllerは戻るボタンをNavigation Barに表示し、ユーザがこのボタンをタップすると、自動的に最上位のView Controllerがpopされます。

Navigation Barをカスタマイズする方法の詳細については、“[Navigation Barの外観のカスタマイズ](#)”（23 ページ）を参照してください。View Controllerをナビゲーションスタックにプッシュする方法（および後から削除する方法）の詳細については、“[ナビゲーションスタックの変更](#)”（20 ページ）を参照してください。ツールバーの内容をカスタマイズする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](#)”（30 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

ナビゲーションインターフェイスを作成する前に、ナビゲーションインターフェイスをどのような目的で使用するかを決めておく必要があります。ナビゲーションインターフェイスはデータに対して支配的な構成となるため、用途は以下に限定されます。

- ウィンドウのルートView Controllerとして追加する。
- Tab BarインターフェイスのタブのView Controllerとして追加する。
- Split Viewインターフェイスの2つのルートView Controllerの1つとして追加する。（iPadのみ）
- 別のView Controllerからモーダルモードで表示する。
- Popoverから表示する。（iPadのみ）

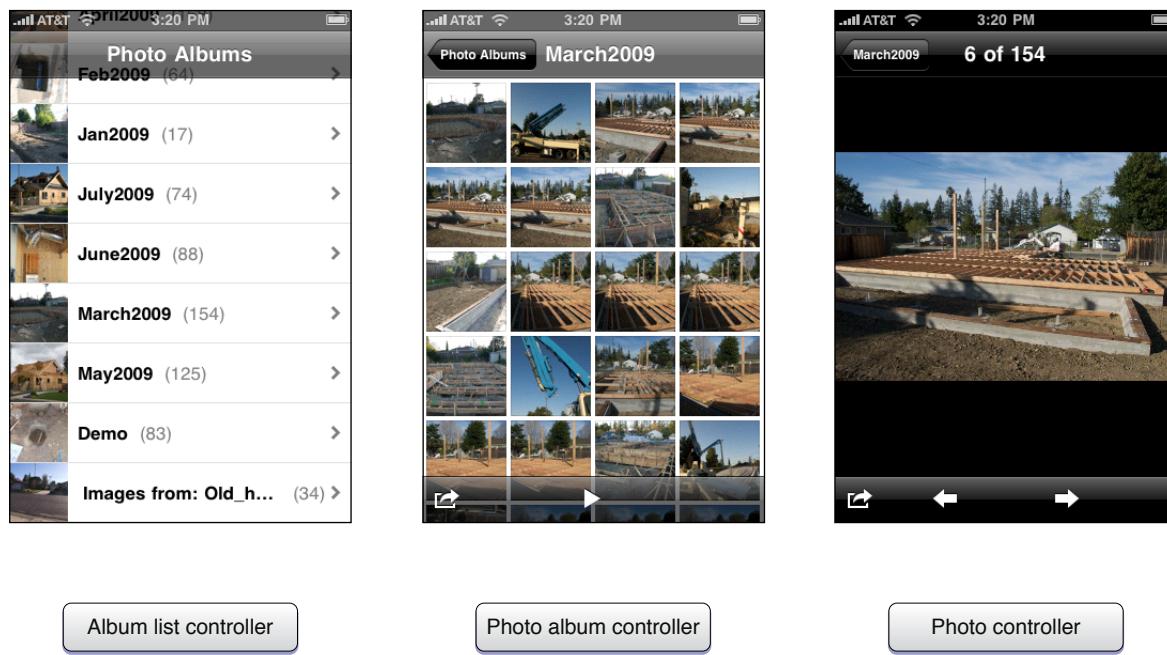
最初の3つのシナリオでは、Navigation Controllerは基本的なインターフェイスのきわめて重要な部分となり、アプリケーションが終了するまで存続します。最後の2つのシナリオは、Navigation Controllerの一時的な用法を表しています。この場合、Navigation Controllerを使用するための手順は、ほかのカスタムView Controllerのための手順と同じです。唯一の違いは、Navigation Controllerは単一のコンテンツView Controllerでは利用できない追加のナビゲーション機能を提供する点です。以降の各セクションでは、より永続的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の使い方に関わらず、すべてのNavigation Controllerに当てはまります。

**注意:** Navigation Controllerをモーダルモードで表示する方法の具体例については、“[モーダルモードでのNavigation Controllerの表示](#)”（65 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コンテンツView Controllerの定義

ナビゲーションインターフェイスごとに、ルートレベルを表す1つのデータレベルが存在します。このレベルはインターフェイスの起点となります。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、用意されているフォトアルバムが一覧表示されます。フォトアルバムを選択すると、アルバム内の写真が表示され、さらに写真を選択すると、より大きいバージョンの写真が表示されます。

図 1-4 データのレベルごとにView Controllerを定義する



ナビゲーションインターフェイスを実装するために、データ階層の各レベルに配置するデータを決定しなければなりません。レベル別に、そのレベルのデータを管理および提示するコンテンツView Controllerを指定します。複数のレベルで表示が共通する場合、同じView Controllerクラスのインスタンスを複数作成し、各インスタンスがそれに固有のデータセットを管理するように設定できます。たとえば、図 1-4に示すように、「写真（Photos）」アプリケーションには3種類の表示形式があるため、3つの異なるView Controllerクラスを使用する必要があります。

リーフデータを管理するView Controllerを除いて、各コンテンツView Controllerは、ユーザがデータ階層の次のレベルにナビゲートするための手段を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しいフォトアルバム用の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。

カスタムView Controllerの定義方法に関する一般的な情報と指示については、『*View Controller Programming Guide for iOS*』の“Creating Custom Content View Controllers”を参照してください。

## ストーリーボードを使用したナビゲーションインターフェイスの作成

新しいXcodeプロジェクトを作成する場合、Master-Detail Applicationテンプレートにより、Navigation Controllerがストーリーボードに作成され、最初のシーンとして設定されます。

ストーリーボードでNavigation Controllerを作成する場合、以下の手順に従います。

1. ライブラリからNavigation Controllerをドラッグします。
2. Interface BuilderからNavigation ControllerとView Controllerが作成され、これらのController間の関係が確立されます。この関係により、新規に作成されたView ControllerがNavigation ControllerのルートView Controllerとして特定されます。
3. 「Attributes」インスペクタで「Is Initial View Controller」をオンにすることにより、最初のView Controllerとして表示（またはView Controllerを別の方法でユーザインターフェイス上に表示）します。

## 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。

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のView Controllerになります。Navigation Barには、そのビューが表示されている間は、戻るボタンが表示されません。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をウインドウのルートView Controllerとして設定します（または、インターフェイスに表示します）。

リスト 1-1に、**Navigation Controller**を作成して、アプリケーションのメインウインドウのルート**View Controller**として設定する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 1-1 プログラム内でのNavigation Controllerの作成

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

    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    window.rootViewController = navigationController;
    [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では、Navigation Barの一番下とツールバーまたはTab Barの一番上との間にカスタムコンテンツが表示されます（図 1-1（12 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。この配置によって、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。上記のリスト内の順序は、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ができません。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。

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。または、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指示できます。

2. Navigation Barの背後に重ねるには、Navigation Controllerの`translucent`プロパティをYESに設定します。
3. 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。
4. 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

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`プロパティと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、『View Controller Programming Guide for iOS』の“Creating Custom Content View Controllers”を参照してください。

## ナビゲーションスタックの変更

ナビゲーションスタックに存在するオブジェクトは、各自が責任を持って作成します。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コンテンツView Controllerを提供する必要があります。プログラムの中で、またはユーザとのやり取りに応じて、View Controllerを追加したり削除したりすることができます。Navigation Controllerクラスでは、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に対応しています。表 1-1に、これらのシナリオとその対応の仕方を示します。

表 1-1 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、セグエまたは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、スタックから最上位の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、最上位の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。	<p>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。</p> <p>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。次回の起動時に、この状態情報を読み込み、setViewControllers:animated:メソッドを呼び出す前に、この情報を使用して必要なView Controllerを再作成します。状態の保存と復元の詳細については、『iOS App Programming Guide』の“App States and Multitasking”を参照してください。</p> <p>このメソッドは、データ階層の任意の場所にジャンプする場合にも使用できます。ただし、任意の場所にジャンプする方法は混乱を生じやすいため、その結果を明確にユーザに伝えるなど特別な注意を払って使用してください。</p>

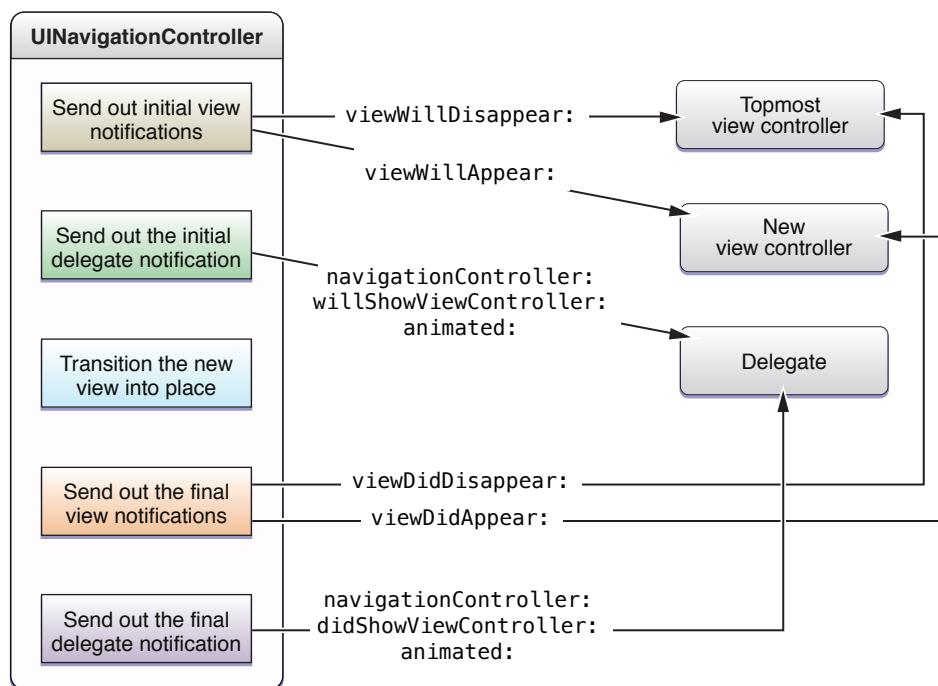
シナリオ	説明
ユーザをルート View Controllerに戻す。	ナビゲーションインターフェイスの最上位に戻るには、 <code>popToRootViewControllerAnimated:</code> メソッドを使用します。このメソッドは、ルートView Controller以外のすべてをナビゲーションスタックから削除します。
階層内を任意の数のレベルだけ戻る。	一度に複数レベル戻るには、 <code>popToViewController:animated:</code> メソッドを使用します。このメソッドは、 <b>Navigation Controller</b> を使用して、（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、一度に1つずつではなく、すべての編集画面をまとめて削除できます。

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**が自動的にもっと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**は最上位のView Controllerに対してだけアニメーションを使用します。中間にある他のView Controllerはすべて、アニメーションなしで削除されます。アニメーションを使ってView Controllerをプッシュまたはポップする場合は、ほかのView Controllerをプッシュまたはポップする前に、アニメーションが完了するまで待機しなければなりません。

## ナビゲーションスタックの変化の監視

View Controllerがプッシュまたはポップされると必ず、Navigation Controllerは影響を受けるView Controllerにメッセージを送信します。Navigation Controllerはまた、スタックの変更時にデリゲートにもメッセージを送信します。図 1-5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、スタックの最上位のView Controllerになる予定のView Controllerを表します。

図 1-5 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すると、コンテンツView Controllerを調整することができます（コンテンツView Controller間で共有する状態を更新する場合など）。複数のView Controllerを一度にプッシュまたはポップした場合、表示されているView Controllerと表示される予定のView Controllerでのみメソッドが呼び出されます。中間のView Controllerでは、コールバックの内部でチェーンの作成が発生する場合を除いて、メソッドが呼び出されません（viewWillAppear:の実装によりpushViewController:animated:が呼び出される場合など）。

UIViewControllerのisMovingToParentViewControllerメソッドと

isMovingFromParentViewControllerメソッドを使用すると、プッシュまたはポップの結果、View Controllerが表示されるか表示されないかを判断できます。

## 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内のコントロールを管理するビューであり、Navigation Controllerオブジェクトで管理される場合に特別な役割を担います。整合性を保証したり、ナビゲーションインターフェイスの作成に必要な作業量を削減したりするために、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の管理における大部分の責任を果たします。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（コンテンツView Controllerなど）とやり取りをします。

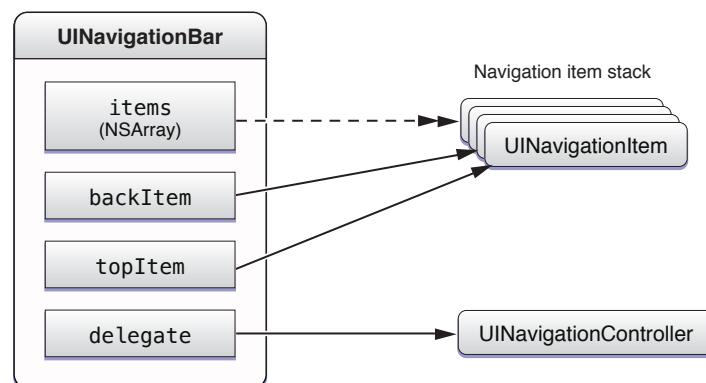
**注意:** Navigation Barをスタンドアロンのビューとして作成し、必要に応じて使用することも可能です。UINavigationBarクラスのメソッドとプロパティの使い方については、『*UINavigationBar Class Reference*』を参照してください。

## 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に表示される一連のビューとコンテンツを提供します。

図 1-6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pushしたり、スタックからpop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 Bar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。Navigation Barのコンテンツのほとんどは最上位のNavigation Itemから取得されますが、（前の項目のタイトルが付けられる）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されます。

図 1-6 Navigation Barに関連するオブジェクト



**Important:** Navigation BarをNavigation Controllerと組み合わせて使用する場合、Navigation Barのデリゲートは常に、それを所有するNavigation Controllerオブジェクトに設定されます。デリゲートの変更を試みると例外が発生します。

ナビゲーションスタック内の各コンテンツView Controllerは、ナビゲーションインターフェイスで使用された場合に、`navigationItem`プロパティの値としてNavigation Itemを提供します。ナビゲーションスタックとNavigation Itemのスタックは常に同等です。ナビゲーションスタック内の各コンテンツView ControllerのNavigation Itemは、Navigation Itemのスタックでも同じ位置にあります。

Navigation Barには、項目を配置するための3つの主要な位置があります（左、右、中央）。表1-2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と組み合わせて使用するようにNavigation Itemを設定している場合は、想定されるコントロールの表示を優先するために、任意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がどのように使われているかについての情報も含まれています。

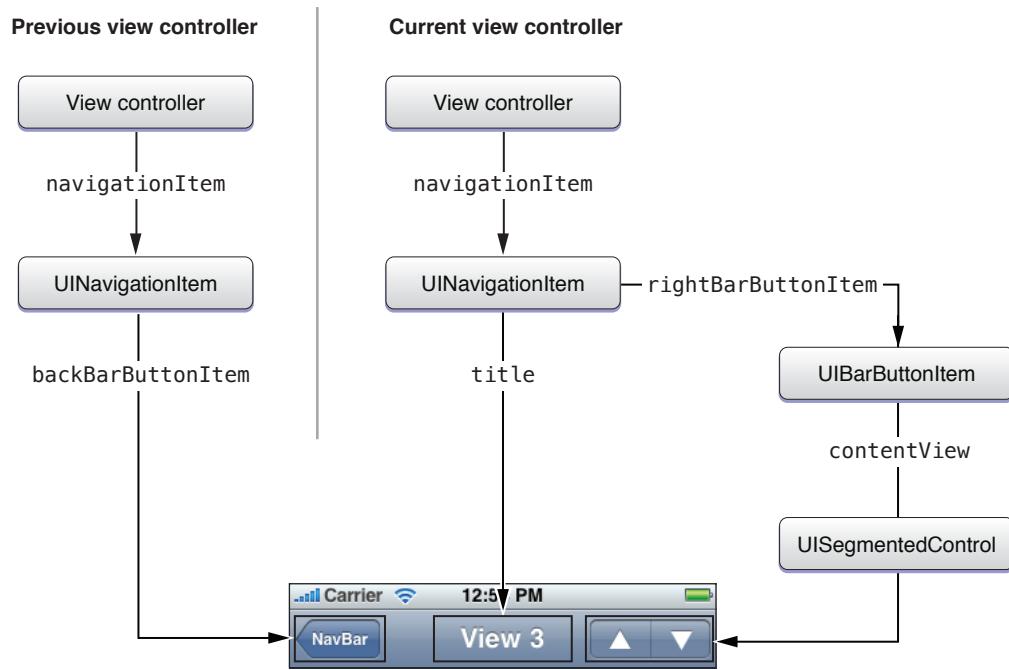
表 1-2 Navigation Bar上の項目の位置

位置	プロパティ	解説
左	<code>backBarButtonItem</code> <code>leftBarButtonItem</code>	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、 <code>backBarButtonItem</code> 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またはビューを割り当て、それでデフォルトの「戻る」ボタンを置換するには、 <code>leftBarButtonItem</code> プロパティにUIBarButtonItemオブジェクトを割り当てます。
中央	<code>titleView</code>	ナビゲーションインターフェイスでは、Navigation Controllerは、デフォルトでコンテンツView Controllerのタイトルを付けてカスタムビューを表示します。必要に応じて、このビューを独自に選択したカスタムビューに置き換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Navigation Itemのタイトル文字列を付けてカスタムビューを表示します。また、Navigation Itemにタイトルが付いていないと、Navigation BarはView Controllerのタイトルを使用します。

位置	プロパティ	解説
右	rightBarButtonItem	この位置はデフォルトで空いています。通常、この位置には、現在の画面を編集したり変更したりするためのボタンが配置されます。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できます。

図 1-7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と、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 1-7 Navigation Barの構造



## Navigation Barの表示と非表示

Navigation BarをNavigation Controllerと組み合わせて使用する場合、Navigation Barを表示したり非表示にしたりするには、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を非表示にすることはできません。Navigation Barの表示/非表示のほかにも、Navigation Controller

のメソッドを使用すると、さらに高度な動作を自動的に実現できます。具体的に言うと、ViewControllerのviewWillAppear:メソッド内でNavigation Barを表示/非表示にすると、Navigation Controllerは、新規のView Controllerの外観に合わせてNavigation Barの表示/非表示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もっとも一般的な方法は、タッチイベントをインターフェースにし、それを使用してNavigation Barの表示/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、このような動作は、Navigation Barの表示/非表示を単に切り替える操作よりも検出が難しい動作です。

## 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はそれ自体のUINavigationBarオブジェクトを所有し、責任を持って管理します。Navigation Barオブジェクトを変更することも、その境界、フレーム、またはアルファ値を直接変更することも許可されていません。ただし、変更可能な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1-8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 1-8 Navigation Barのスタイル



Navigation Bar全体を表示/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:`メソッドを使用します。Navigation Barを表示/非表示にする方法の詳細については、“[Navigation Barの表示と非表示](#)”（25 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。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の`backBarButtonItem`プロパティに有効な項目が設定されていると、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。
- カスタムタイトルのビューが設定されていない場合、Navigation Barは、そのView Controllerのタイトルを付けて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、View Controller自体の`title`プロパティの文字列を使います。

## Navigation Controller

Navigation Barの外観のカスタマイズ

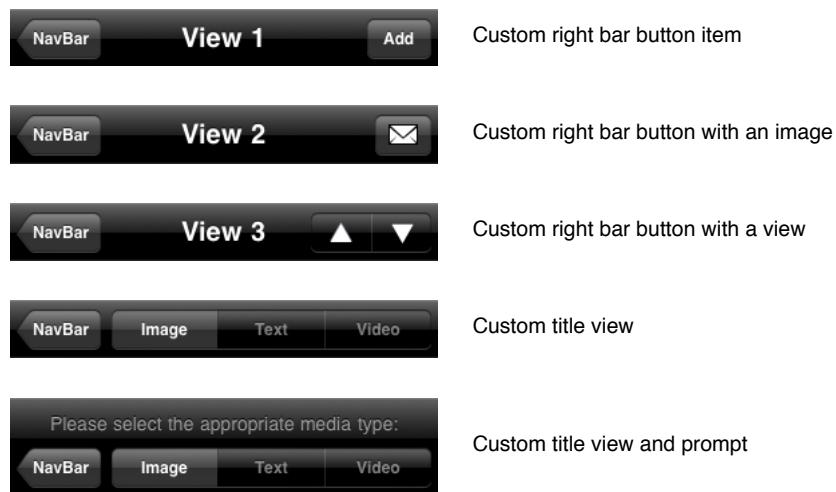
最上位のView Controllerに関して、Navigation Barの右側に表示される項目は次の規則に従って決定されます。

- 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 1-9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 1-9 Navigation Barのカスタムボタン



リスト 1-2に、図 1-9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 1-2 カスタムBar Button Itemの作成

```
// View 3 - Custom right bar button with a view
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; // Keep track if this if you need  
it later.  
UIBarButtonItem *segmentBarItem = [[UIBarButtonItem alloc]  
initWithCustomView:segmentedControl];  
self.navigationItem.rightBarButtonItem = segmentBarItem;
```

プログラムの中で**ViewController**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、**ViewController**の**viewDidLoad**メソッド内に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**ViewController**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**ViewController**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、**ViewController**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方法の詳細については、『*ViewController Programming Guide for iOS*』の“Creating Custom Content View Controllers”を参照してください。

## ナビゲーションツールバーの表示

iOS 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、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。このツールバー自体は、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、Navigation Controllerは、それぞれのツールバー項目セット間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。

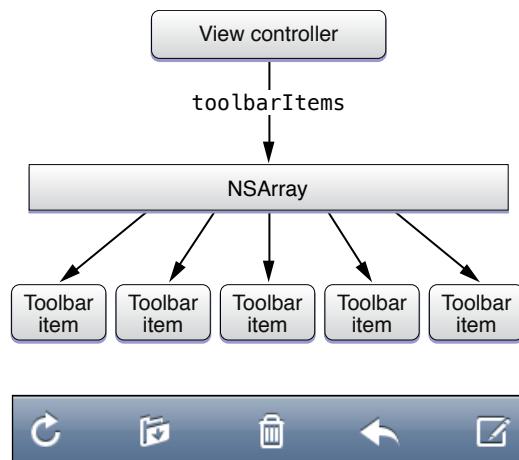
ナビゲーションインターフェイスにツールバーを設定するには、次の手順を実行する必要があります。

- 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各コンテンツView ControllerのtoolbarItemsプロパティに割り当てます（“ツールバー項目の指定”（31 ページ）を参照）。

特定のコンテンツView Controllerにツールバーを表示しない場合は、ツールバーを非表示にすることができます（“ツールバーの表示と非表示”（32 ページ）を参照）。

図 1-10に、コンテンツ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 1-10 ナビゲーションインターフェイスのツールバー項目



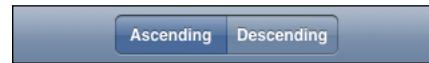
## ツールバー項目の指定

Bar Button Itemを設定する場合は、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自体です。これは、View Controllerがツールバー項目の提供を担当しているからです。図1-11に、Segmented Controlを中央に配置したツールバーの例を示します。

ストーリーボードを使ってツールバー項目を指定する手順を以下に示します。

1. ライブラリからツールバーをドラッグします。
2. ライブラリからドラッグして、ツールバーに可変サイズのBar Button Itemを2つ追加します。
3. ライブラリからドラッグして、可変サイズのBar Buttonの間にSegmented Controlを追加します。  
インスペクタを使ってSegmented Controlの動作を設定します。

図 1-11 ツールバーの中央に配置されたSegmented Control



リスト1-3に、ツールバー項目をプログラムで指定するコード例を示します。このメソッドをView Controllerに実装して、初期化の際に呼び出します。

リスト 1-3 中央にSegmented Controlを持つツールバーの設定

```
- (void)configureToolbarItems
{
    UIBarButtonItem *flexibleSpaceItem = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace
                           target:nil action:nil];

    // Create and configure the segmented control
    UISegmentedControl *sortToggle = [[UISegmentedControl alloc]
initWithItems:[NSArray arrayWithObjects:@"Ascending",
              @"Descending", nil]];
    sortToggle.segmentedControlStyle = UISegmentedControlStyleBar;
    sortToggle.selectedSegmentIndex = 0;
    [sortToggle addTarget:self action:@selector(toggleSorting:)
forControlEvents:UIControlEventValueChanged];
```

```
// Create the bar button item for the segmented control
UIBarButtonItem *sortToggleButtonItem = [[UIBarButtonItem alloc]
                                         initWithCustomView:sortToggle];

// Set our toolbar items
self.toolbarItems = [NSArray arrayWithObjects:
                     flexibleSpaceButtonItem,
                     sortToggleButtonItem,
                     flexibleSpaceButtonItem,
                     nil];
}
```

初期化中にツールバー項目を設定するだけでなく、`ViewController`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`ViewController`用のツールバーを非表示にするには、その`ViewController`の`hidesBottomBarWhenPushed`プロパティを`YES`に設定します。`NavigationController`は、このプロパティが`YES`に設定された`ViewController`を検出すると、この`ViewController`がナビゲーションスタックにpushされる（またはナビゲーションスタックからpop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。

（常時ではなく）時々ツールバーを非表示にする場合は、必要に応じて`NavigationController`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 (Photos)」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。Tab Bar Controllerのビュー階層は自己完結型です。この階層は、Tab Bar Controllerが直接管理するビューと、作成中に指定した各コンテンツView Controllerで管理されるビューから構成されます。コンテンツView Controllerはそれぞれビュー階層を管理し、Tab Bar Controllerは各ビュー階層の間のナビゲーションを調整します。

この章では、アプリケーションでTab Bar Controllerを設定し、使用する方法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](#)”（62 ページ）を参照してください。

## Tab Barインターフェイスの構造

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、画面の一番下に表示される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View Controllerとして指定されています。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、それに対応するコンテンツView Controllerに関連付けられたビューを表示します。

図 2-1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。Tab Bar Controllerには、固有のコンテナビューがあります。このビューは、ほかのすべてのビュー (Tab Bar ビューを含む) で構成されています。カスタムコンテンツは、選択されたタブのViewControllerによって提供されます。

図 2-1 Tab Barインターフェイスのビュー



Tab Bar ビューが Tab Bar インターフェイスに組み込まれている場合は、このビューを変更することができません。Tab Bar インターフェイスでは、Tab Bar ビューは、Tab Bar Controller オブジェクトが所有しているプライベート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 自体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”](#)（40 ページ）を参照してください。

**注意:** Tab Barをスタンドアロンのビューとして作成し、必要に応じて使用することも可能です。UITabBarクラスのメソッドとプロパティの使い方については、『UITabBar Class Reference』を参照してください。

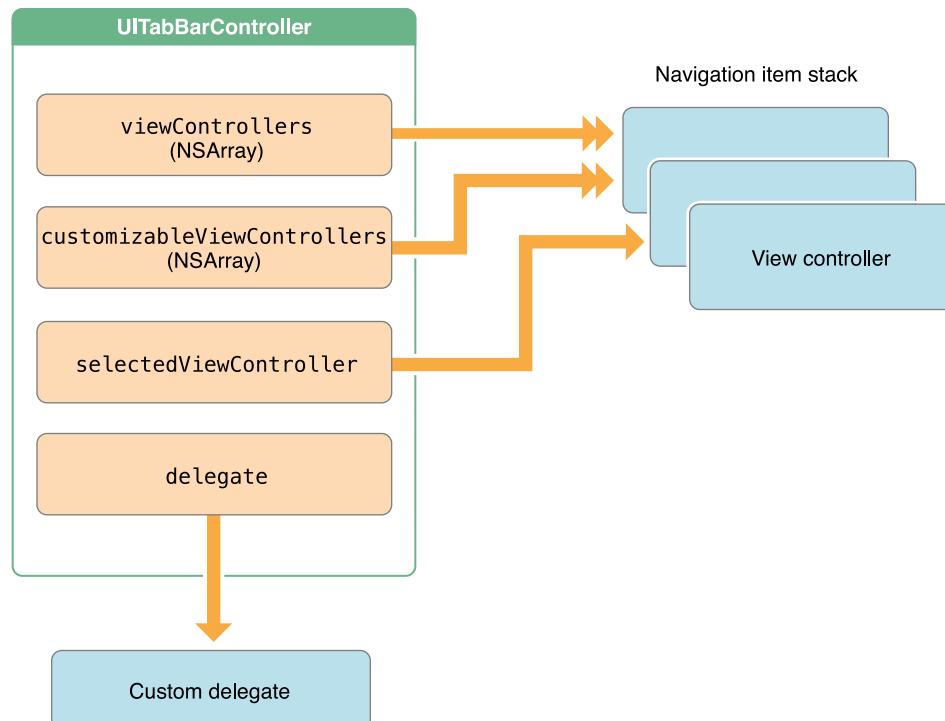
## Tab Barインターフェイスのオブジェクト

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つ存在するコンテンツ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2-2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View Controllerです。

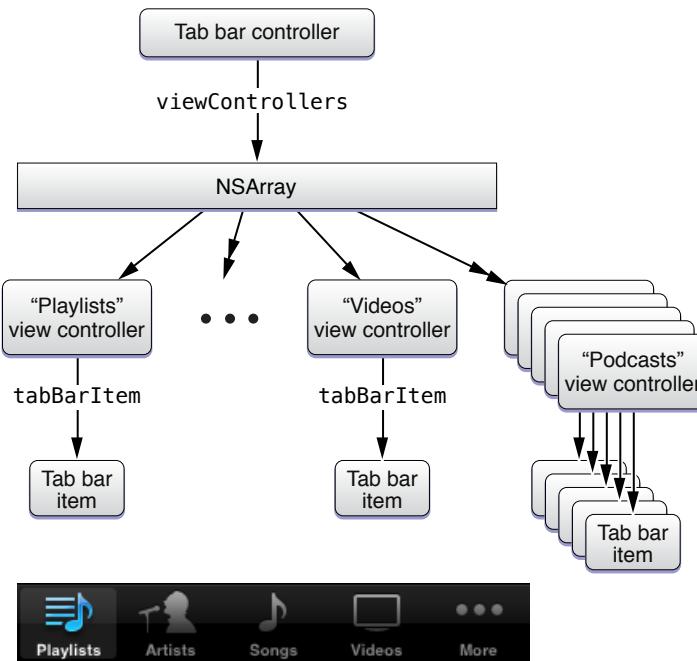
図 2-2 Tab Bar Controllerとそれに関連付けられたView Controller



viewControllersプロパティに項目を6つ以上追加すると、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。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ビューを直接変更しないでください。Tab Bar Controllerオブジェクトは、コンテンツView Controllerが提供するUITabBarItemオブジェクトからTab Barのコンテンツを構成します。図 2-3に、「iPod」 アプリケーションに含まれるTab Bar Controller、View Controller、Tab Bar Itemオブジェクト間の関係を示します。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 2-3 「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インターフェイスを表示する前に、各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（[“ストーリーボードを使用したTab Barインターフェイスの作成”](#)（38 ページ）を参照）。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、コンテンツView Controllerごとに1つのUITabBarItemオブジェクトを新規に作成しなければなりません（“[プログラムによるTab Barインターフェイスの作成](#)”（39 ページ）を参照）。

Tab Bar Controllerは、オプションのデリゲートオブジェクトもサポートしています。このオブジェクトを使用すると、Tab Barの選択やカスタマイズに対応できます。デリゲート関連のメッセージに対応する方法については、[“実行時のタブの管理”](#)（40 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、その使い方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に対して支配的な構成となるため、用途は以下に限定されます。

- ウィンドウのルートView Controllerとして追加する。
- Split Viewインターフェイスの2つのView Controllerの1つとして追加する。（iPadのみ）
- 別のView Controllerからモーダルモードで表示する。
- Popoverから表示する。（iPadのみ）

アプリケーションのメインウィンドウに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 Controllerを単独で使用することができますが、ほかのView Controllerと組み合わせて使用すると、より高度なインターフェイスを作成できます。詳細については、[“Combined View Controller Interfaces”](#)（62 ページ）を参照してください。

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（デバイスに応じて）画面の全体または大部分を占有するため、Tab Barの表示は、モーダルモードで表示されているデータを閲覧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の使い方は避けるべきです。

## Tab Barインターフェイス用のコンテンツ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から独立しているため、各タブの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か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コンテンツView Controllerをインストールします。

図 2-4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、主に時計リストの編集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で1つの画面しか必要でないため、単一のView Controllerを使用しています。「タイマー (Timer)」タブは、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 2-4 「時計 (Clock)」アプリケーションのタブ



Tab Bar Controllerは、コンテンツ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。コンテンツ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。

カスタムView Controllerの定義方法に関する一般的な情報と指示については、『*View Controller Programming Guide for iOS*』の“Creating Custom Content View Controllers”を参照してください。

## ストーリーボードを使用したTab Barインターフェイスの作成

新しいXcodeプロジェクトを作成する場合、Tabbed Applicationテンプレートにより、Tab Bar Controllerがストーリーボードに作成され、最初のシーンとして設定されます。

ストーリーボードでTab Bar Controllerを作成する場合、以下の手順に従います。

1. ライブドリからTab Bar Controllerをドラッグします。
2. Interface Builderから1つのTab Bar Controllerと2つのView Controllerが作成され、これらの間の関係が確立されます。この関係により、Tab Bar Controllerの1つのタブに対応するView Controllerとして、新規に作成された各View Controllerが特定されます。
3. 「Attributes」インスペクタで「Is Initial View Controller」をオンにすることにより、最初のView Controllerとして表示（またはView Controllerを別の方でユーザインターフェイス上に表示）します。

## 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になります。

1. 新規のUITabBarControllerオブジェクトを作成します。
2. 各タブにコンテンツView Controllerを作成します。
3. これらの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をウインドウのルートView Controllerとして設定します（または、インターフェイスに表示します）。

リスト 2-1に、Tab Bar Controllerのインターフェイスを作成してアプリケーションのメインウインドウにインストールするのに必要な基本コードを示します。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。このためには、さらに多くのView Controllerオブジェクトを作成してcontrollers配列に追加します。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換える必要があります。

### リスト 2-1 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;

window.rootViewController = tabBarController;
}
```

## プログラム内のTab Bar Itemの作成

Tab Barインターフェイスの各コンテンツ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。このためには、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の作成に推奨するタイミングは、View Controller自体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。またView Controllerオブジェクト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 2-2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に関連付けられるため、そのView Controllerが初期化処理の一環としてTab Bar Item自体を作成します。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 2-2 View ControllerのTab Bar Itemの作成

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

## 実行時のタブの管理

Tab Barインターフェイスを作成したら、いくつかの方法を使用して、インターフェイスを変更し、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、動的な条件に基づく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。

## 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加えることができます。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:` メソッド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 2-3 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 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 2-3 現在のタブの削除

```
- (IBAction)processUserInformation:(id)sender
{
    // Call some app-specific method to validate the user data.
    // If the custom method returns YES, remove the tab.
    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、タブ内に何もコンテンツがない場合など、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 2-4 に、このようなメソッドの例を示します。`isValidLogin` 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 2-4 タブの選択の禁止

```
- (BOOL)tabBarController:(UITabBarController *)aTabBar  
    shouldSelectViewController:(UIViewController *)viewController  
{  
    if (![self hasValidLogin] && (viewController != [aTabBar.viewControllers  
objectAtIndex:0]))  
    {  
        // Disable all but the first tab.  
        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したり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、コンテンツView Controllerで処理するのが最善です。

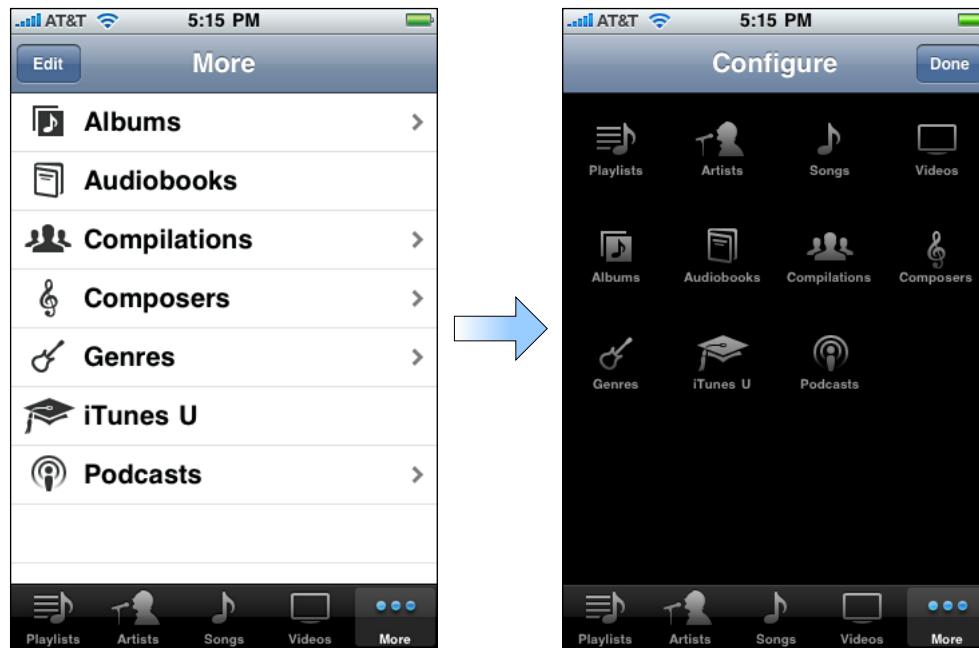
UITabBarControllerDelegateプロトコルのメソッドの詳細とその使用方法については、『*UITabBarControllerDelegate Protocol Reference*』を参照してください。

#### タブのカスタマイズの禁止

「More」 View Controllerには、Tab Barに表示された項目をユーザが修正できるようにする機能が組み込まれています。多数のタブを含むアプリケーションの場合、ユーザはこの機能を利用して、すぐにアクセスできる画面と、アクセスするのに余分なナビゲーションが必要な画面を選択できます。図 2-5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」ViewController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 2-5 「iPod」アプリケーションのTab Barの設定



ほとんど常時、ユーザがタブを配置変更できるようにしておいても構いませんが、特定のタブをTab Barから削除したり、特定のタブをTab Barに配置したり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、ViewController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ViewControllerのサブセットを含めます。この配列に含まれていないViewController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ViewControllerをTab Barから削除することはできません。

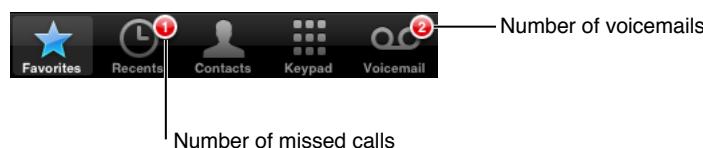
**Important:** Tab BarインターフェイスでView Controllerを追加したり削除したりすると、カスタマイズ可能なView Controllerの配列もデフォルト値にリセットされて、すべてのView Controllerがカスタマイズ可能な状態に戻ります。したがって、viewControllersプロパティを（直接、またはsetViewControllers:animated:メソッドを呼び出して）変更したときに、引き続きカスタマイズ可能なView Controllerを制限する場合は、customizableViewControllersプロパティ内のオブジェクトの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。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）、バッジを使う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2-6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 2-6 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。リスト2-5に、バッジ内の新しい項目の数を表示するView Controllerでバッジの値が設定されるプロセスの例を示します。

### リスト 2-5 タブのバッジの設定

```
if (numberOfNewItems == 0)
    self.tabBarItem.badgeValue = nil;
else
    self.tabBarItem.badgeValue = [NSString stringWithFormat:@"%d", 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にこのような値のプロパティが含まれている場合、キー値監視（KVO）通知を使用して、値の変更を検出し、変更に応じてバッジを更新します。KVO通知の設定と処理の詳細については、『*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、デフォルトで縦向きに対応しています。内部のView Controllerの一部が横向きに対応していない限り、横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つでもその向きに対応していないものがあれば、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はフルスクリーンレイアウトに対応していますが、ほかのほとんどのView Controllerとは方法が異なります。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、コンテンツ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、『*View Controller Programming Guide for iOS*』の“Creating Custom Content View Controllers”を参照してください。

# Page View Controller

Page View Controllerは、コンテンツをページごとに表示する場合に使用します。Page View Controllerは、自己完結型のビュー階層を管理します。この階層の親ビューはPage View Controllerで管理され、子ビューは指定したコンテンツView Controllerで管理されます。

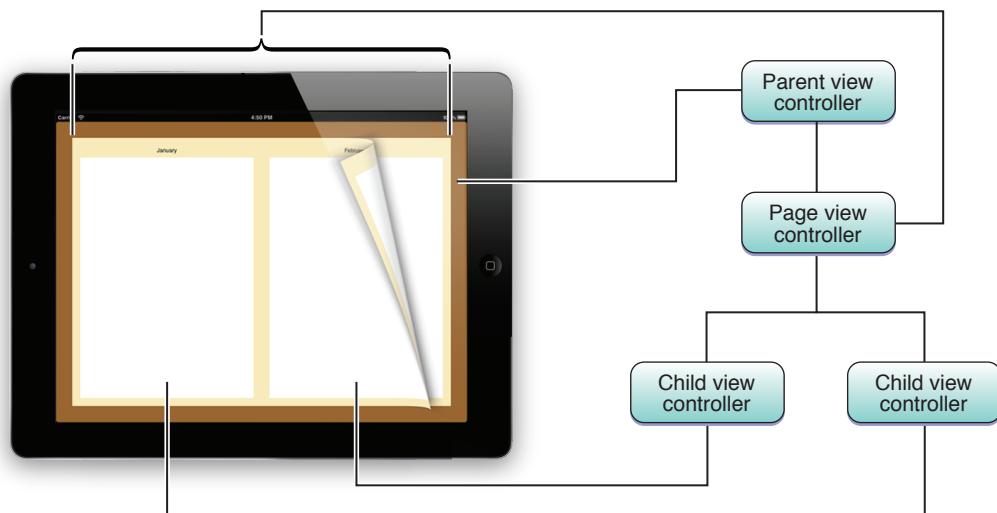
## Page View Controllerインターフェイスの構造

Page View Controllerは、デベロッパのコンテンツをホスティングするためのビューを1つ保有します。Page View Controllerに用意されているUIは、ユーザがページをめくると表示されます。Page View Controllerは、View Controllerのビューにページの折り返しを適用し、ページがめくられるときの外観を付加します。

Page View Controllerにおけるナビゲーションは、一列に並んだページが表示されることです。小説のテキストのように直線的にアクセスされるコンテンツの表示や、カレンダーのような改ページが自然に行われるコンテンツの表示に非常に適しています。リファレンスブックなど、ユーザが非線形的にアクセスする必要のあるコンテンツの場合、ナビゲーションロジックとUIの指定はデベロッパが行います。

図3-1に、アプリケーション例に実装されているPage Viewインターフェイスを示します。一番外側の茶色のビューは、Page View Controller自体ではなく、親View Controllerに対応づけられています。Page View Controller独自のUIはありませんが、ユーザがページをめくる際、子にめくり効果を加えます。カスタムコンテンツは、Page View Controllerの子に当たるView Controllerによって提供されます。

図3-1 Page Viewインターフェイスのビュー



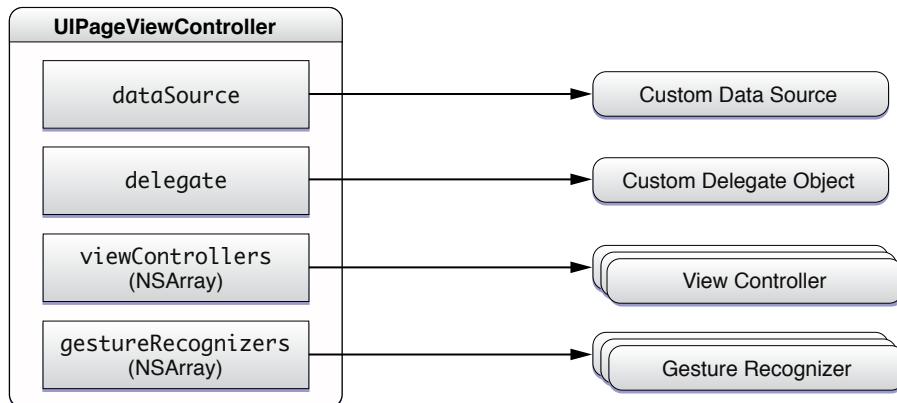
## Page View Controllerインターフェイスのオブジェクト

Page View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。

- オプションのデリゲート
- オプションのデータソース
- 現在のView Controllerの配列

- Gesture Recognizerの配列

図 3-2 Page View Controllerとその関連オブジェクト



データソースは、オンデマンドでView Controllerを提供します。

デリゲートは、ジェスチャベースのナビゲーションと向きの変更に応じて呼び出されるメソッドを提供します。

View Controllerの配列は、現在表示されているコンテンツView Controllerを含みます。この配列の項目の数は、Page View Controllerに渡されるオプションに応じて異なります。

Gesture Recognizerの配列が読み込まれるのは、データソースが設定されている場合のみです。これらのGesture Recognizerにより、ユーザはタップ、フリック、またはドラッグ操作でページ間を移動できます。

## Page View Controllerインターフェイスの作成

Page View Controllerのビューのサイズを変更し、ビュー階層に埋め込むことができます。すなわち、Navigation ControllerやTab Bar Controllerと異なり、Page View Controllerを特定の場合にとどまらず、幅広い設定で使用できるということです。

## ストーリーボードを使用したPage View Controllerインターフェイスの作成

「Page-Based Application」Xcodeテンプレートでは、Page View Controllerを初期シーンに使用して新しいプロジェクトを作成します。

Page View Controllerを既存のストーリーボードに追加する場合、以下の手順に従います。

1. ライブラリからPage View Controllerをドラッグします。ストーリーボードに、Page View Controllerのシーンを追加します。
2. 「Attributes」インスペクタで、適切なオプションを設定します。
3. 必要に応じて、対応するアウトレットの接続を行い、デリゲートまたはデータソース（あるいはその両方）を設定します。
4. 「Attributes」インスペクタで「Is Initial View Controller」をオンにすることにより、最初のView Controllerとして表示（またはView Controllerを別の方法でユーザインターフェイス上に表示）します。

## プログラムによるPage View Controllerインターフェイスの作成

Page View Controllerをプログラムで作成する方法を以下に示します。

1. `initWithTransitionStyle:navigationOrientation:options:`メソッドを使用して、Page View Controllerを割り当て、初期化します。初期化時のカスタマイズについては、“[初期化時の動作のカスタマイズ](#)”（49 ページ）を参照してください。
2. 必要に応じて、デリゲートまたはデータソース（あるいはその両方）を設定します。
3. 初期コンテンツView Controllerを設定します。
4. Page View Controllerのビューを画面に表示します。

## 初期View Controllerの設定

Page View ControllerをInterface Builderとプログラムのどちらで作成するかに関わらず、画面に表示する前に初期View Controllerを設定する必要があります。

初期View Controllerを設定する場合、適切な数のView Controllerを含む配列で `setViewControllers:direction:animated:completion:`メソッドを呼び出します。

**デバッグのヒント:** Page View Controllerに初期View Controllerが設定されていないと、どの向きでも`shouldAutorotateToInterfaceOrientation:`からNOが返され、例外が発生します。

## 初期化時の動作のカスタマイズ

パラメータ値とオプションを`initWithTransitionStyle:navigationOrientation:options:`メソッドに渡し、初期化時のPage View Controllerをカスタマイズします。これらのPage View Controllerは、初期化の後、読み取り専用のプロパティでアクセスできます。以下の動作をカスタマイズできます。

- ナビゲーションの方向（横方向または縦方向）

- スパインの位置（両端または中央）
- トランジションスタイル（ページめくりまたはスクロール）

たとえば、リスト 3-1では、ナビゲーションの方向を横方向、スパインの位置を中心、トランジションスタイルをページめくりに設定して、Page View Controllerを初期化しています。

### リスト 3-1 Page View Controllerのカスタマイズ

```
NSDictionary * options = [NSDictionary dictionaryWithObject:  
    [NSNumber numberWithInt:UIPageViewControllerSpineLocationMid]  
    forKey:UIPageViewControllerOptionSpineLocationKey];  
  
UIPageViewController *pageViewController = [[UIPageViewController alloc]  
    initWithTransitionStyle:UIPageViewControllerTransitionStylePageCurl  
    navigationOrientation:UIPageViewControllerNavigationOrientationHorizontal  
    options:options];
```

## デリゲートによる実行時の動作のカスタマイズ

Page View Controllerのデリゲートは、UIPageViewControllerDelegateプロトコルを実装します。このプロトコルは、デバイスの向きが変更されたとき、およびユーザが新しいページにナビゲートしたときにアクションを実行し、インターフェイスの方向の変更に応じてスパインの位置を更新します。

## データソースの提供によるコンテンツの指定

データソースの指定により、ジェスチャ指向のナビゲーションが可能になります。データソースを指定しない場合は、ナビゲーション用のカスタムUIを用意し、“[現在のView Controllerの設定によるコンテンツの指定](#)”（51 ページ）の説明に従ってコンテンツを指定する必要があります。提供したデータソースにより、UIPageViewControllerDataSourceプロトコルが実装されなければなりません。

データソースのメソッドは、現在表示されているView Controllerで呼び出され、表示中のView Controllerの前または後に表示されるView Controllerを返します。直前または直後のView Controllerを検出するプロセスを簡素化するために、ページ番号などの追加情報をView Controllerに格納しておくことができます。

データソースが提供されると、Page View ControllerはGesture Recognizerをビューに関連付けます。これらのGesture Recognizerを使って、ユーザはタップ、フリック、ドラッグ操作でページを移動します。ページにはgestureRecognizersプロパティとしてアクセスします。

Gesture Recognizerを別のビューに移動するには、gestureRecognizersプロパティの値をターゲットビューのaddGestureRecognizer:メソッドに渡します。Gesture Recognizerを他のビューに移動する方式が特に有用になるのは、Page View Controllerを大規模なビュー階層に埋め込む場合です。

たとえば、Page View Controllerが画面の一部にしか描画しない場合、より大きなスーパービュー上にGesture Recognizerを配置することにより、ユーザのページめくりが容易になります。Gesture Recognizerを移動すると、Page View Controllerのビュー境界内だけでなく、スーパービューの境界内のどこからでも、ジェスチャを始めることができます。

## 現在のView Controllerの設定によるコンテンツの指定

表示されるコンテンツを直接制御する場合は、

setViewControllers:direction:animated:completion:メソッドを呼び出し、コンテンツView Controllerの配列を表示に渡します。渡されるView Controllerの数は一定ではありません。詳細については、メソッドのリファレンスドキュメントを参照してください。

この方法は、ユーザがコンテンツ内の特定の場所（最初のページや概要など）に移動する方法を指定するものです。ユーザとUIとのやり取りに応じて、View Controllerを直接設定します。

データソースを指定しない場合、進むボタンや戻るボタンなど、ページ間を移動するためのUIを提供する必要があります。ジェスチャ指向のナビゲーションは、データソースを提供する場合にのみ利用できます。

## Right-to-LeftとBottom-to-Topコンテンツの特別な考慮事項

Page View Controllerは左側および上部のコンテンツが現在のページの前のコンテンツであると認識し、右側および下部のコンテンツが現在のページの後のコンテンツであると認識します。これはLeft-to-RightおよびTop-to-Bottomコンテンツの利用法と一致しています。

データソースからコンテンツをRight-to-LeftまたはBottom-to-Topに表示するPage View Controllerを使用する場合、2つのメソッドの動作を逆にします。

- データソースで、pageViewController:viewControllerBeforeViewController:を実装し、所定のView Controllerの後にView Controllerを返します。
- データソースで、pageViewController:viewControllerAfterViewController:を実装し、所定のView Controllerの前にView Controllerを返します。

Right-to-LeftまたはTop-to-Bottomコンテンツの場合、通常はスパインの位置を  
UIPageViewControllerSpineLocationMaxに設定します。

# Split View Controller

UISplitViewControllerクラスは、2つのペインからなる情報を管理するコンテナView Controllerです。第1のペインは、幅が320ポイントに固定され、高さは表示中のウインドウの高さに合わせられます。第2のペインは、残りのスペースを埋めます。図4-1に、Split View Controllerのインターフェイスを示します。

図4-1 Split Viewインターフェイス



Split Viewインターフェイスのペインに含まれるコンテンツは、デベロッパが提供するView Controllerで管理されます。ペインにはアプリケーション固有のコンテンツが含まれるため、2つのView Controller間のやり取りはデベロッパが管理します。ただし、回転などのシステム関連の動作は、Split View Controller自体が管理します。

Split View Controllerは常に、デベロッパが作成するインターフェイスのルートでなければなりません。つまり、UISplitViewControllerオブジェクトからのビューは、必ずアプリケーションのウインドウのルートビューとして組み込まなければなりません。Split Viewインターフェイスのペインには、Navigation ControllerやTab Bar Controllerなど、インターフェイスの実装に必要なあらゆる種類のView Controllerを含めることができます。Split View Controllerはモーダルモードで表示できません。

Split View Controllerをアプリケーションに組み込むもっとも簡単な方法は、新規プロジェクトから始めることです。XcodeのSplit View-based Applicationテンプレートは、Split View Controllerを組み込んだインターフェイスを作成する場合の適切な起点になります。Split Viewインターフェイスの実装に必要なものがすべて用意されています。デベロッパがしなければならないのは、コンテンツを表示するた

めに、View Controllerの配列を変更することだけです。これらのView Controllerを変更する手順は、iPhoneアプリケーションで用いる手順とほとんど同じです。唯一の違いは、より広い画面領域を使って詳細関連のコンテンツを表示できる点です。ただし、Split View Controllerを既存のインターフェイスに統合することもできます。

## ストーリーボードを使用したSplit View Controllerの作成

新しいXcodeプロジェクトを作成する場合、Master-Detail Applicationテンプレートにより、Split Viewがストーリーボードに作成され、最初のシーンとして設定されます。

既存のアプリケーションにSplit View Controllerを追加する手順を以下に示します。

1. アプリケーションのメインストーリーボードを開きます。
2. ライブラリからSplit View Controllerをドラッグします。

Interface BuilderからSplit View Controller、Navigation Controller、ViewControllerが作成され、これらのController間の関係が確立されます。この関係により、新規に作成されたViewControllerがSplit View Controllerの左右のペインとして特定されます。

3. 「Attributes」インスペクタで「Is Initial View Controller」をオンにすることにより、最初のView Controllerとして表示（またはViewControllerを別の方法でユーザインターフェイス上に表示）します。

Split Viewに埋め込まれた2つのViewControllerのコンテンツは、デベロッパが管理します。これらのViewControllerの設定は、アプリケーション内のその他のViewControllerを設定する場合と同様です。たとえば、埋め込まれたNavigation ControllerやTab Bar Controllerの場合は、必要に応じて追加のViewController情報を指定します。

## プログラムによるSplit View Controllerの作成

プログラムでSplit View Controllerを作成するには、UISplitViewControllerクラスの新しいインスタンスを作成して、2つのプロパティにViewControllerを割り当てます。Split View Controllerのコンテンツは、その場で作成されるため、Split View Controllerを作成するときにnibファイルを指定する必要はありません。したがって、単にinitメソッドを使用して初期化できます。リスト4-1に、起動時にSplit Viewインターフェイスを作成して設定する方法の例を示します。実践においては、1番目と2番目のViewControllerを、自分のアプリケーションのコンテンツを表示するViewControllerオブジェクトに置き換えます。window変数は、アプリケーションのメインnibファイルからロードされたウインドウを指すアウトレットであると想定します。

**リスト 4-1** プログラムによるSplit View Controllerの作成

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions  
{  
  
    MyFirstViewController* firstVC = [[MyFirstViewController alloc] init];  
    MySecondViewController* secondVC = [[MySecondViewController alloc] init];  
  
    UISplitViewController* splitVC = [[UISplitViewController alloc] init];  
    splitVC.viewControllers = [NSArray arrayWithObjects:firstVC, secondVC, nil];  
  
    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];  
    window.rootViewController = splitVC;  
    [window makeKeyAndVisible];  
  
    return YES;  
}
```

## Split Viewでの向きの変更のサポート

Split View Controllerでは、どの向きをサポートしているかは、これに含まれる2つのView Controllerによって決まります。どちらのView Controllerもサポートしている向きにしか対応できません。これは一方のView Controllerが現在非表示になっていても同じです。向きの変更が生じた場合は、Split View Controllerは回転動作のほとんどを自動的に処理します。

横向きの場合、Split View Controllerは2つのペインを小さな分割線で区切って並べて表示します。縦向きの場合、Split View ControllerはsplitViewController:shouldHideViewController:inOrientation:デリゲートメソッドから返された値に応じて、両方のペインを表示するか、より大きい第2のペインだけを表示し、第1のペインをPopoverで表示するためのツールバーボタンを提供します。

# Popover

UIPopoverControllerクラスは、それ自体ViewControllerではありませんが、ViewControllerの表示を管理します。Popover Controllerオブジェクトを使用して、**Popover**、つまりアプリケーションのウィンドウの手前に浮かぶ視覚的なレイヤにコンテンツを表示します。Popoverは、情報を表示したりユーザから情報を収集したりするための手軽な手段であり、次のような状況でよく使われます。

- 画面上のオブジェクトについての情報を表示する
- 頻繁にアクセスするツールや設定オプションを管理する
- いずれかのビュー内のオブジェクトに対して実行するアクションのリストを表示する
- デバイスが縦向きの場合にSplit View Controllerの一方のペインを表示する

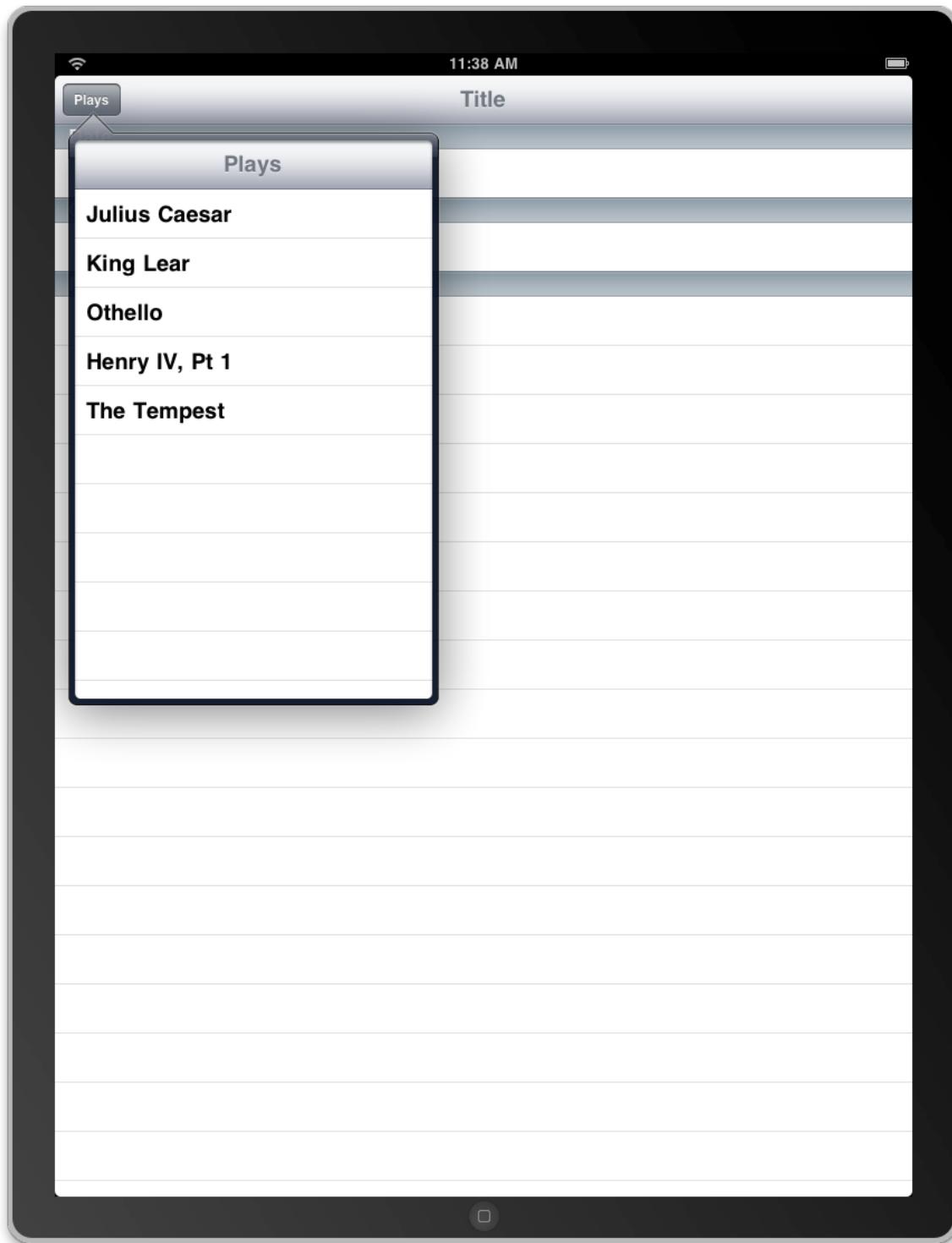
上記のアクションに対してPopoverを使用するほうが、モーダルビューよりも押し付けがましさや煩雑さが抑えられます。iPadアプリケーションでは、モーダルビューは、ユーザに何らかのアクションや情報を明示的に承諾させたりキャンセルさせたりする必要があるときに限って使用すべきです。たとえば、アプリケーションのほかの部分へのアクセスを許可するときに、ユーザにパスワードを尋ねる場合などにモーダルビューを使用するのがよいでしょう。それ以外のほとんどの状況に対しては、代わりにPopoverを使用します。Popoverの利点は、画面全体を覆わず、Popoverビューの外側をタップするだけで消去できる点です。このため、Popoverは、ユーザとの対話なしで、ユーザに情報や追加機能を提供するような状況に非常に適しています。

**注意:** ほかのView Controllerをモーダルモードで表示する際には、表示されているPopoverがあれば、あらかじめそれらを消去しておく必要があります。アプリケーションでPopoverを使用する方法と使用すべき状況に関する指示については、『*iOS Human Interface Guidelines*』を参照してください。

---

図5-1に、Split Viewインターフェイスのペインを表示する場合のPopoverの使用例を示します。Popoverから「再生 (Play)」を選択すると、アプリケーションのメインビューに再生に関する情報が表示されます。（Split Viewインターフェイスを作成する方法について詳しくは、“[Split View Controller](#)”（53ページ）を参照してください。）

図 5-1 Popoverを使用したマスタペインの表示



## Popoverの作成と表示

Popoverのコンテンツは、デベロッパが提供するView Controllerオブジェクトから取得されます。Popoverにはほとんどの種類のView Controllerを表示できます。View ControllerをPopoverに表示する準備ができたら、次の手順を実行します。

1. UIPopoverControllerクラスのインスタンスを作成してView Controllerオブジェクトで初期化します。
2. 次のどちらかの方法で、Popoverのサイズを指定します。
  - Popoverに表示するView ControllerのcontentSizeForViewInPopoverプロパティに値を割り当てる。
  - Popover Controller自体のpopoverContentSizeプロパティに値を割り当てる。
3. (省略可能) Popoverにデリゲートを割り当てる。デリゲートの役割の詳細については、“[Popover用のデリゲートの実装](#)” (61 ページ) を参照してください。
4. Popoverを表示します。

Popoverを表示するときは、ユーザインターフェイスの特定の部分にPopoverを関連付けます。たいていは、Popoverをツールバーのボタンに関連付けます。その場合は、

presentPopoverFromBarButtonItem:permittedArrowDirections:animated:メソッドが、アプリケーションのツールバーからPopoverを表示するための便利な手段になります。

presentPopoverFromRect:inView:permittedArrowDirections:animated:メソッドを使用して、Popoverをビューの特定の部分に関連付けることもできます。

Popoverは通常、表示されているView ControllerのcontentSizeForViewInPopoverプロパティから初期のサイズを取得します。このプロパティに保存されるデフォルトのサイズは、幅が320ピクセル、高さが1100ピクセルです。contentSizeForViewInPopoverプロパティに新しい値を割り当てることによって、デフォルト値をカスタマイズできます。また、Popover Controller自体のpopoverContentSizeプロパティに値を割り当てることもできます。Popoverによって表示されるView Controllerを変更すると、popoverContentSizeプロパティに保存したすべてのカスタムサイズ情報が、新しいView Controllerのサイズに置き換わります。Popoverが表示されているときに、コンテンツView Controllerまたはそのサイズを変更すると、その変更は自動的にアニメーション化されます。また、setPopoverContentSize:animated:メソッドを使用して、(アニメーションの有無を指定して) サイズを変更することもできます。

**注意:** 画面上でのPopoverの実際の配置は、いくつかの要因（ViewControllerのコンテンツのサイズ、Popoverの発生元として使われたボタンやビューの位置、許可される矢印の方向）に基づいてPopover Controller自体で決定されます。

リスト5-1に、ツールバーのタップに応答してPopoverを表示する簡単なアクションメソッドを示します。Popoverは、Popoverオブジェクトへの強固な参照を保持するプロパティ（Popoverを所有するクラスで定義されている）に格納されます。Popoverのサイズは、ViewControllerのビューのサイズに設定されています。ただし、この2つのサイズは必ずしも同じである必要はありません。当然、2つのサイズが同じでない場合は、Popoverのすべてのコンテンツを見る能够ないように、スクロールビューを使用しなければなりません。

#### リスト 5-1 プログラムによるPopoverの表示

```
- (IBAction)toolbarItemTapped:(id)sender
{
    MyViewController* content = [[MyViewController alloc] init];
    UIPopoverController* aPopover = [[UIPopoverController alloc]
        initWithContentViewController:content];
    aPopover.delegate = self;

    // Store the popover in a custom property for later use.
    self.popoverController = aPopover;

    [self.popoverController presentPopoverFromBarButtonItem:sender
        permittedArrowDirections:UIPopoverArrowDirectionAny animated:YES];
}
```

Popoverは、ユーザがPopoverビューの外側をタップすると、自動的に画面から消去されます。Popoverの内側をタップしてもPopoverは自動的に消去されませんが、dismissPopoverAnimated:メソッドを使用すると、プログラムで消去することができます。ユーザがViewControllerのコンテンツ内の項目を選択したり、画面からPopoverを消去する何らかのアクションを実行したりする場合に、この処理が必要になることがあります。Popoverをプログラムで消去する場合は、Popover Controllerオブジェクトへの参照を、ViewControllerからアクセスできる場所に保存しておく必要があります。システムは、現在アクティブなPopover Controllerへの参照を提供しません。

## Popover用のデリゲートの実装

Popoverビューの外側をユーザがタップしたことによってPopoverが画面から消去されると、Popoverはそのアクションを自動的にデリゲートに通知します。デリゲートを用意している場合は、これを使ってPopoverの消去を防いだり、消去のアクションに応えて追加のアクションを実行したりできます。`popoverControllerShouldDismissPopover:`デリゲートメソッドを使用して、Popoverを実際に消去すべきかどうかをコントロールします。デリゲートがこのメソッドを実装していない場合、またはデリゲートのこのメソッドの実装がYESを返した場合、コントローラはPopoverを消去して`popoverControllerDidDismissPopover:`メッセージをデリゲートに送信します。

ほとんどの場合、`popoverControllerShouldDismissPopover:`メソッドをオーバーライドする必要はまったくありません。このメソッドは、Popoverの消去によってアプリケーションに問題が生じる場合に備えるためのものです。しかしながら、このメソッドからNOを返すよりも、Popoverの表示を維持しなければならないような設計は避けたほうがよいでしょう。たとえば、コンテンツをモーダルモードで表示して、ユーザに必須情報の入力や、変更の承諾またはキャンセルを求めるのがよいかも知れません。

デリゲートの`popoverControllerDidDismissPopover:`メソッドが呼び出されるまでには、Popover自体は画面から消去されています。改めて使用する予定がない場合は、この時点でPopover Controllerへの既存の参照を削除しておくことが安全な方法です。また、このメソッドを使用すると、ユーザインターフェイスを更新したり、アプリケーションの状態を更新したりすることもできます。

## アプリケーションでPopoverを管理する際のヒント

アプリケーションでPopover関連のコードを記述する場合は、次の点を考慮します。

- プログラムでPopoverを消去するには、Popover Controllerへのポインタが必要になります。そのようなポインタを取得する唯一の方法は、自分でポインタをコンテンツViewControllerに格納することです。これによって、コンテンツViewControllerは、適切なユーザアクションに応答してPopoverを消去できるようになります。
- Popover Controllerは、毎回最初から作成するのではなく、キャッシュしておき再利用することができます。Popover Controllerは適応性が高いため、使用するごとに異なるView Controllerと設定オプションを指定することができます。
- Popoverを表示する際には、できる限り、許容する矢印の向きにUIPopoverArrowDirectionAny定数を指定します。この定数を指定することで、UIKitがPopoverの位置とサイズを決定する際の柔軟性が最大になります。許容する矢印の向きを制限すると、Popoverを表示する前に、Popover Controllerがそのサイズを縮小しなければならない場合があります。

# View Controllerインターフェイスの組み合わせ

UIKitフレームワークから提供される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、より高度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、包含関係の順序が重要になります。特定の配列のみ有効です。子から親への包含関係の順序は、以下の通りです。

- コンテンツView Controller、大きさ可変のコンテナView Controller（Page View Controllerなど）
- Navigation View Controller
- Tab Bar Controller
- Split View Controller

モーダルView Controllerは割り込みを表し、多少異なる規則に従います。ほとんどのすべての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。

以降の各セクションでは、iOSアプリケーションでTable View Controller、Navigation Controller、Tab Bar Controllerを組み合わせる方法を示します。iPadアプリケーションでのSplit View Controllerの使用方法について詳しくは、“[Split View Controller](#)”（53 ページ）を参照してください。テーブルビューについて詳しくは、『[Table View Programming Guide for iOS](#)』を参照してください。

## 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を使用することもできます。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、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のもっとも一般的な使い方は、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、プログラムによる方法と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できません。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、“[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」(18ページ)に記載されているようにフルスクリーンレイアウトを採用していると、コンテンツがナビゲーションバーの背後に重なることがあります、Tab Barの背後に重なることはありません。

Tab BarインターフェイスにNavigation Controllerを埋め込む場合は、UINavigationControllerクラスのインスタンスのみを埋め込むようにし、UINavigationControllerクラスのサブクラスであるシステム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。

## ストーリーボードを使用したオブジェクトの作成

カスタムView Controllerを含む3つのタブとNavigation Controllerを含む1つのタブで構成された複合インターフェイスを作成する方法を以下に示します。

1. 3つのカスタムView Controller（各タブに1つずつ）と1つのNavigation Controllerを作成します。
2. 3つのカスタムView Controllerと、Navigation Controller（ルートView Controller以外のNavigation Controllerシーンのみ）を選択します。
3. 「Editor」>「Embed In」>「Tab Bar Controller」を選択します。
4. 「Attributes」インスペクタで「Is Initial View Controller」ボックスをオンにすることにより、Tab Bar Controllerを最初のView Controllerとして表示（またはView Controllerを別の方法でユーザインターフェイス上に表示）します。

## 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、applicationDidFinishLaunching:アプリケーションデリゲートのメソッド内です。カスタムView Controllerを含む3つのタブとNavigation Controllerを含む1つのタブがある複合インターフェイスを作成する手順を以下に示します。

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オブジェクトを作成します。
3. ナビゲーションインターフェイスのルート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。

リスト 6-1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の作成に必要なテンプレートコードが表示されます。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、アプリケーションデリゲートクラスの宣言済みプロパティであり、これらのオブジェクトへの強固な参照が保持されます。

#### リスト 6-1 プログラムによるTab Bar Controllerの作成

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

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

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

    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    window.rootViewController = tabBarController;
    [window makeKeyAndVisible];
}
```

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、『“Creating Custom Content View Controllers”「カスタムコンテンツView Controllerの作成」View Controller Programming Guide for iOS』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ABPeoplePickerController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 presentModalViewControllerAnimated: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）追加しなければなりません。

リスト 6-2には、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が示されています。この例では、ナビゲーションスタックにpusch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への参照であり、これも提供する必要があります。

### リスト 6-2 モーダルモードでのNavigation Controllerの表示

```
MyViewController1* rootVC = [[MyViewController1 alloc] init];
MyViewController2* nextVC = [[MyViewController2 alloc] init];
NSArray * viewControllers = [NSArray arrayWithObjects:rootVC, nextVC, nil];

// Create the nav controller and set the view controllers.
UINavigationController* theNavController = [[UINavigationController alloc]
                                             initWithRootViewController:rootVC];
[theNavController setViewControllers:viewControllers animated:NO];

// Display the nav controller modally.
[currentViewController presentModalViewControllerAnimated:theNavController animated:YES];
```

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 Controllerを消去するのは、表示中の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はナビゲーションスタックから削除されるだけですが、一番上のView Controllerは、通常のviewWillDisappear:メッセージを受信します。

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、『*View Controller Programming Guide for iOS*』の“Presenting View Controllers from Other View Controllers”を参照してください。アプリケーションで使用するためのNavigation Controllerを設定する方法の詳細については、“[Navigation Controller](#)”（11 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 Controller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。したがって、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じ方法で、ルートView Controllerを作成して設定し、それらをTab Bar Controllerに追加する必要があります。

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 Controller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は削除されるだけですが、現在表示されているタブのView Controllerは、通常のviewWillDisappear:メッセージを受信します。

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、『*View Controller Programming Guide for iOS*』の“Presenting View Controllers from Other View Controllers”を参照してください。アプリケーションで使用するためのTab Bar Controllerを設定する方法の詳細については、“[Tab Bar Controller](#)”（33 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ています。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使用されることがよくあります。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。

テーブルを管理する方法の1つに、UITableViewControllerオブジェクトを使用する方法があります。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があります。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pushする必要があります。

リスト 6-3には、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が示されています。ユーザが現在のテーブル内の特定の行をタップしたら、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push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。その作業は、行に対応するデータオブジェクトを取得し、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 6-3 テーブルビューを使用したデータのナビゲーション

```
// Implement something like this in your UITableViewController subclass
// or in the delegate object you use to manage your table.
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Create a view controller with the title as its
    // navigation title and push it.
    NSUInteger row = indexPath.row;
    if (row != NSNotFound)
    {
        // Create the view controller and initialize it with the
        // next level of data.
        MyViewController *viewController = [[MyViewController alloc]
            initWithTable:tableView andDataAtIndexPath:indexPath];
        [[self navigationController] pushViewController:viewController
```

```
    animated:YES];  
}  
}
```

テーブルの管理方法とテーブルView Controllerの使い方の詳細については、『*Table View Programming Guide for iOS*』を参照してください。

# 書類の改訂履歴

この表は「*iOS View Controller*カタログ」の改訂履歴です。

日付	メモ
2014-11-15	Tab Bar Controllerを更新しました。
2012-02-16	全体を通して細かい修正を施しました。
2012-01-09	iOSで利用できるView Controllerと、その使い方について解説した新規ドキュメント。
この内容は、従来『 <i>View Controller Programming Guide for iOS</i> 』の一部であったものです。	



Apple Inc.

Copyright © 2015 Apple Inc.

All rights reserved.

の法的権利を与え、地域によってはその他の権利が  
お客様に与えられる場合もあります。

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

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

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

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

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

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

Apple Japan  
〒106-6140 東京都港区六本木  
6丁目10番1号 六本木ヒルズ  
<http://www.apple.com/jp>

Offline copy. Trademarks go here.

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

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

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

一部の国や地域では、默示あるいは偶発的または結果的損害に対する賠償の免責または制限が認められないため、上記の制限や免責がお客様に適用されない場合があります。この保証はお客様に特定