iPadプログラミングガイド

2010-03-24

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

序章 **はじめに 9**

お読みになる前に 9
この書類の構成 9
関連項目 10

第 1 章 **iPad開発について 11**

iPadとは 11
開発の基礎 11
　コアアーキテクチャ 12
　View Controller 12
　グラフィックスおよびマルチメディア 13
　イベント処理 14
　デバイス統合のサポート 14
iPadデバイスの新機能 14
　より広いスペース 15
　ユーザインターフェイスを際立たせる新しい要素 15
　テキスト入力および表示のサポート強化 16
　外部ディスプレイおよびプロジェクタのサポート 17
　文書およびファイル処理の正式サポート 17
　PDFの生成 18

第 2 章 **プロジェクトの開始 19**

ユニバーサルアプリケーションの作成 19
　Xcodeプロジェクトの設定 20
　Info.plist設定の更新 20
　ビューおよびView Controllerの更新 21
　新しいシンボル用のランタイムチェックの追加 22
　ランタイムチェックを使用して条件付きコードパスを作成 23
　リソースファイルの更新 23
1つのXcodeプロジェクトを使用して、2つのアプリケーションを作成 23
一から始める 25
Media Playerフレームワーク使用に関する移植の重要なヒント 25

第 3 章 **アプリケーションのコア設計 27**

iPadアプリケーションのアーキテクチャ 27
アプリケーションバンドル 30
　アプリケーションのInfo.plistファイルに新たに追加されたキー 30
　さまざまな向き用の起動画像の提供 32

iPadデバイスのドキュメントサポート　33
ファイルのプレビューとオープン　33
アプリケーションでサポートするファイルタイプの登録　35
サポート可能なファイルタイプのオープン　36

第 4 章　ビューとView Controller　39

複数の向きに対応した設計　39
Split Viewインターフェイスの作成　40
Interface BuilderでのSplit View Controllerの追加　42
プログラムによるSplit View Controllerの作成　43
Split Viewでの向きの変更のサポート　43
Popoverを使用したコンテンツの表示　44
Popoverの作成と表示　46
Popover用のデリゲートの実装　47
アプリケーションでPopoverを管理する際のヒント　47
モーダルビューの表示スタイルの設定　48
ツールバーの有効利用　50

第 5 章　Gesture Recognizer　51

イベント処理を簡単にするGesture Recognizer　51
認識可能なジェスチャ　51
Gesture Recognizerをビューに添付する　52
ジェスチャによってアクションメッセージが発行される　53
単発のジェスチャと連続的なジェスチャ　53
ジェスチャ認識の実装　54
Gesture Recognizerの準備　55
ジェスチャへの応答　56
ほかのGesture Recognizerとのやり取り　57
Gesture Recognizerの失敗を条件とする　57
Gesture Recognizerのタッチ解析を禁止する　58
同時のジェスチャ認識を許可する　58
ビューへのタッチ配信の制御　59
デフォルトのタッチイベント配信　59
ビューへのタッチ配信の操作　60
カスタムGesture Recognizerの作成　61
状態遷移　61
カスタムGesture Recognizerの実装　62

第 6 章　グラフィックスと描画　65

ベジエパスを使用した図形の描画　65
ベジエパスの基礎　65
パスへの直線と多角形の追加　66
パスへの弧の追加　66

パスへの曲線の追加　67
楕円パスと矩形パスの作成　68
Core Graphicsの関数を使用したパスの変更　69
ベジエパスオブジェクトのコンテンツのレンダリング　70
パス上でのヒット検出　71
PDFコンテンツの生成　72
PDFコンテキストの作成と設定　73
PDFページの描画　75
PDFコンテンツ内でのリンクの作成　77

第 7 章　テキストの独自処理と入力　79

入力ビューと入力アクセサリビュー　79
単純なテキスト入力　80
テキスト入力システムとのやり取り　82
クライアント側のテキスト入力の概要　82
テキスト位置とテキスト範囲　83
UITextInputオブジェクトの仕事　83
トークナイザ　84
テキスト描画とテキスト処理の機能　85
Core Text　85
UIStringDrawingとCATextLayer　87
Core Graphicsのテキスト描画　87
Foundationレベルの正規表現　88
ICU正規表現のサポート　89
スペルチェックと単語補完　89
カスタム編集メニュー項目　91

改訂履歴　書類の改訂履歴　93

図、表、リスト

第 3 章　アプリケーションのコア設計　27

図 3-1　iPadアプリケーションの主要オブジェクト　28
表 3-1　アプリケーションでの各オブジェクトの役割　28
表 3-2　iPhone OS 3.2で新たに追加されたInfo.plistのキー　31
表 3-3　アプリケーションのデフォルトの起動画像ファイル　32
リスト 3-1　カスタムファイル形式用のドキュメントタイプ情報　36

第 4 章　ビューとView Controller　39

図 4-1　Split Viewインターフェイス　41
図 4-2　Popoverを使用したマスタペインの表示　45
図 4-3　モーダルモードの表示スタイル　49
リスト 4-1　プログラムによるSplit View Controllerの作成　43
リスト 4-2　Popoverの表示　46

第 5 章　Gesture Recognizer　51

図 5-1　Gesture Recognizerがビューにアタッチされている場合のタッチオブジェクトの経路　52
図 5-2　単発のジェスチャと連続的なジェスチャ　54
図 5-3　Gesture Recognizerで起こり得る状態遷移　62
表 5-1　UIKitフレームワークのGesture Recognizerクラスで認識可能なジェスチャ　52
リスト 5-1　単発ジェスチャおよび連続ジェスチャに対応するGesture Recognizerの作成と初期化　55
リスト 5-2　ピンチ、パニング、ダブルタップの各ジェスチャの処理　56
リスト 5-3　「チェックマーク」Gesture Recognizerの実装　63
リスト 5-4　Gesture Recognizerのリセット　64

第 6 章　グラフィックスと描画　65

図 6-1　デフォルトの座標系での弧　67
図 6-2　パス内の曲線セグメント　68
図 6-3　PDF文書を作成するためのワークフロー　73
図 6-4　リンク先とジャンプ元の作成　77
リスト 6-1　五角形の作成　66
リスト 6-2　弧パスの新規作成　67
リスト 6-3　新規のCGPathRefをUIBezierPathオブジェクトに割り当てる　69
リスト 6-4　Core Graphics呼び出しとUIBezierPath呼び出しの併用　69
リスト 6-5　ビューにパスを描画する　70
リスト 6-6　パスオブジェクトに対する点のテスト　71
リスト 6-7　PDFファイルの新規作成　74

リスト 6-8　ページ単位のコンテンツの描画　75

第 7 章　テキストの独自処理と入力　79

図 7-1　テキスト入力システムとのコミュニケーション経路　82
図 7-2　Core Textのレイアウトエンジンのアーキテクチャ　86
図 7-3　カスタムメニュー項目を含む編集メニュー　91
表 7-1　iPhone OS 3.2に含まれているICUファイル　89
リスト 7-1　プログラムによる入力アクセサリビューの作成　80
リスト 7-2　単純なテキスト入力の実装　81
リスト 7-3　正規表現を使用した部分文字列の検索　88
リスト 7-4　文書のスペルチェック　90
リスト 7-5　現在の不完全な文字列を補完した単語のリスト　91
リスト 7-6　「Change Color」メニュー項目の実装　92

はじめに

iPadの登場は、iPhone OSを使用したアプリケーション開発に新たな機会をもたらします。iPadはiPhone OSを実行するため、すでにiPhoneやiPod touch向けに作成されているアプリケーションをすべてiPadで実行できます。しかし、iPadのより大きな画面サイズは、これまで以上のことを行えるアプリケーションを作成する新たな機会がもたらされたということも意味しているのです。

この文書では、iPadで利用できる新しい機能について解説し、これらの機能をアプリケーションの中で使う方法を示します。しかし、機能が利用できるからといって、その機能を使用しなければならないというわけではありません。そのため、この文書では、ユーザにとって魅力的なアプリケーションを作成するために、いつどのように新しい機能を使用すればよいかについてのガイダンスも示します。

お読みになる前に

この文書を読むための前提として、iPhoneアプリケーションの開発プロセスをよく理解している必要があります。iPadアプリケーションとiPhoneアプリケーションの開発プロセスはとてもよく似ているため、iPhoneアプリケーションの開発プロセスを出発点として考えるべきです。iPhoneアプリケーションの（広くとらえるとiPadアプリケーションの）アーキテクチャや開発プロセスに関する情報が必要な場合は、『*iPhone Application Programming Guide*』を参照してください。

この書類の構成

この文書は、次の章で構成されています。

- 「iPad開発について」（11 ページ）では、iPadアプリケーションに含めることのできる新機能の情報を含め、プラットフォームについて概説します。
- 「プロジェクトの開始」（19ページ）では、iPhoneアプリケーションを移植するための選択肢について説明し、iPad開発を支援するXcodeプロジェクトの設定方法を示します。
- 「アプリケーションのコア設計」（27 ページ）では、いくつかの核となる新しい機能の使いかたとともに、iPadの基本アプリケーションアーキテクチャについて説明します。
- 「ビューとView Controller」（39 ページ）では、プラットフォームの新しいインターフェイス要素について説明し、その使用例を示します。
- 「Gesture Recognizer」（51 ページ）では、タッチイベントを処理し、アクションを開始する新しいジェスチャ認識テクノロジーの使いかたを説明します。
- 「グラフィックスと描画」（65 ページ）では、新しい描画関連テクノロジーの使いかたを説明します。
- 「テキストの独自処理と入力」（79 ページ）では、新しいテキスト関連機能と、アプリケーションにテキストをよりうまく組み込むための方法について説明します。

関連項目

iPadアプリケーションの開発には、iPhoneアプリケーションの開発に使用するのと同じ手法やプロセスの多くを使用します。iPhoneアプリケーションの設計プロセスをよく知らない場合は、詳細について以下の文書を参照してください。

- iPadアプリケーションのアーキテクチャの概要については、『*iPhone Application Programming Guide*』を参照してください。
- View Controllerについて、およびアプリケーションインフラストラクチャの実装でView Controllerが果たす重要な役割については、『*View Controller Programming Guide for iPhone OS*』を参照してください。
- iPadアプリケーションの実装時に従うべきヒューマンインターフェイスのガイドラインについては、『*iPad Human Interface Guidelines*』を参照してください。

iPad開発について

この章ではiPadファミリのデバイスについて紹介し、これらのデバイスで利用できる基本機能および、これらのデバイス向けにアプリケーションを開発するための要件について説明します。すでにiPhoneアプリケーションを作成したことがあれば、iPadアプリケーションの作成はとても馴染みがあるように感じるでしょう。基本的な機能および動作の大部分は同じです。ただしiPhone OS 3.2には、アプリケーションで使いたくなるようなiPadデバイス固有の機能が備わっています。

iPadとは

iPadデバイスでは、これまでよりも大きな画面でMulti-Touchアプリケーションを作成することができます。1024×768ピクセルの画面では、コンテンツを表示するスペースがはるかに広くなります。すなわち、既存のコンテンツをより詳細に表示できます。そして、iPhone OS 3.2で追加された新しいインターフェイス要素によって、まったく新しいタイプのアプリケーションが実現できます。

iPadのサイズと機能は、ポータブルデバイス向けに新しい種類のアプリケーションが作れるようになったことを意味します。画面サイズが大きくなったことで、ほぼどのようなコンテンツでも表示できるだけのスペースが提供されます。また、Multi-Touchインターフェイスと物理的なキーボードのサポートによって、単純なジェスチャ駆動型のやり取りから、コンテンツ作成および多量のテキスト入力に至るまで、さまざまなモードの対話操作が可能になります。

大きくなった画面サイズによって、実世界のモノをデジタルな形態で再現する新しい分類の没入型アプリケーションの作成も可能になりました。たとえば、iPadの「連絡先(Contacts)」や「カレンダー(Calendar)」アプリケーションは、家の机の上にありそうな紙でできたアドレス帳やカレンダーのように見えます。実在するモノのこのようなデジタルメタファによって、より自然で馴染みのある体験がユーザに提供され、そのアプリケーションを使いたいという気持ちにさせます。しかし、デジタルであるがゆえに、物理的なオブジェクトそのものの制約を超え、生産性や利便性をより高められるアプリケーションを作成することが可能です。

開発の基礎

すでにiPhoneアプリケーションの作成プロセスに精通していれば、iPadアプリケーションの作成プロセスはとても馴染みがあるように感じるでしょう。大部分については、上位レベルのプロセスは同じです。すべてのiPhoneおよびiPadデバイスはiPhone OSを実行し、同じ基本テクノロジーと設計手法を使用します。2つのデバイスのもっとも大きな違いは画面サイズであり、これは、それぞれのデバイス向けに作成するインターフェイスのタイプに影響する可能性があります。もちろん2つのデバイスにはほかにも細かな違いがあるため、以降の各セクションでは、iPadデバイス用の主要なシステム機能の概要および、これらの機能のどこがiPhoneデバイスと異なるのかについて説明します。

コアアーキテクチャ

わずかな例外を除き、iPadアプリケーションのコアアーキテクチャはiPhoneアプリケーションのコアアーキテクチャと同じです。システムレベルでは次のとおりです。

- 一度に1つのアプリケーションだけ実行でき、そのアプリケーションのウインドウは画面全体を覆う。
- アプリケーションは、すばやく起動、終了することが期待されている。
- セキュリティのために、それぞれのアプリケーションはサンドボックス環境内で実行される。サンドボックスには、アプリケーション固有のファイルや環境設定のためのスペースが含まれます。これらはユーザのコンピュータにバックアップされます。デバイスのほかのアプリケーションとのやり取りは、システムに用意されているインターフェイスを通じてのみ行われます。
- 各アプリケーションはそれぞれの仮想メモリ空間で実行されるが、使用可能な仮想メモリ容量は物理メモリ容量の制約を受ける。つまり、メモリとディスクとの間でページングは行われません。
- カスタムプラグインおよびフレームワークはサポートされない。

アプリケーション内では、次の動作が適用されます。

- （**新**）アプリケーションのインターフェイスは、横向き、縦向き、すべての方向をサポートしなければならない。これは、縦向き、横向き両方での実行が必須ではないiPhoneとは、若干異なる動作です。詳細については、「複数の向きに対応した設計」（39 ページ）を参照してください。
- アプリケーションは主にObjective-Cで書かれるが、CやC++も同様に使用できる。
- iPhoneアプリケーションで利用できるクラスはすべて、iPadアプリケーションでも利用できる（iPhone OS 3.2で導入されたクラスは、iPhoneアプリケーションでは利用できない）。
- 保持／解放モデルを使用してメモリを管理する。
- アプリケーションは必要に応じて追加のスレッドを生成することがある。ただし、ビューベースの操作や多くのグラフィックス操作は、必ずアプリケーションのメインスレッドで実行しなければなりません。

すでにiPhoneアプリケーションで馴染みのある基本的なデザインパターンはすべて、iPadアプリケーションにも適用されます。デリゲーションおよびプロトコル、Model-View-Controller、ターゲット／アクション、通知、および宣言済みプロパティなどのパターンは、すべてiPadアプリケーションで一般的に使用されます。

iPhoneアプリケーション開発の基礎をよく知らない場合は、この文書を読み進める前に『*iPhone Application Programming Guide*』を読むことをお勧めします。すべてのCocoa Touchアプリケーションで使用される基本的なデザインパターンの詳細については、『*Cocoa Fundamentals Guide*』を参照してください。

View Controller

iPhoneアプリケーションの場合と同様に、View ControllerはiPadアプリケーションのユーザインターフェイスの管理と表示に欠かせないインフラストラクチャの1つです。View Controllerは1つのビューを対象とします。ほとんどの場合、View Controllerのビューはアプリケーションウインドウの全体を埋めるものと想定されています。しかし時には、View Controllerが別のView Controller（コンテナView

Controllerと呼びます）の内部に組み込まれ、ほかのコンテンツと一緒に表示されることもあります。Navigation ControllerとTab Bar Controllerは、コンテナView Controllerの例です。これらは、複雑なナビゲーションインターフェイスを実装するために、カスタムビューと、埋め込まれているView Controllerからのビューを融合したビューを示します。

iPadアプリケーションでは、Navigation ControllerとTab Bar Controllerが引き続きサポートされており、これらを使用することについてはまったく問題ありません。しかし、洗練されたインターフェイスの作成におけるNavigation ControllerとTab Bar Controllerの重要性はいくぶん低下しています。より単純なデータセットでは、Split View Controllerと呼ばれる新しいタイプのView Controllerで、Navigation ControllerやTab Bar Controllerを置き換えることができるかもしれません。より複雑なデータセットでも、Navigation ControllerやTab Bar Controllerは、下位レベルの移動をサポートするだけで、ユーザインターフェイスにおいては2次的な役割を果たすだけということがよくあります。

iPhone OS 3.2の新しいView Controller関連の動作の詳細については、「ビューとView Controller」（39 ページ）を参照してください。

グラフィックスおよびマルチメディア

iPhoneアプリケーションで使用しているグラフィックスおよびメディアテクノロジーはすべて、iPadアプリケーションでも利用できます。これには、Core Graphics、UIKit、Core Animationなどのネイティブ2D描画テクノロジーが含まれます。2Dおよび3Dコンテンツの描画には、OpenGL ES 2.0またはOpenGL ES 1.1を使用することもできます。

iPadでのOpenGL ESの使用は、ほかのiPhone OSデバイスでのOpenGL ESの使用と同じです。iPadはPowerVR SGXデバイスであり、ほかのSGXデバイスと同じ基本機能をサポートします。ただしiPadの場合、プロセッサ、メモリアーキテクチャ、画面サイズが異なるため、コードは出荷前に必ずiPadデバイスでテストし、パフォーマンスが要件を満たしていることを確認するべきです。

これまでにiPhone OSで使用していたオーディオテクノロジーはすべて、iPadアプリケーションでも利用できます。Core Audio、AV Foundation、OpenALなどのテクノロジーを使用して、内蔵スピーカーやヘッドフォンジャックを通して高品質オーディオを再生できます。また、Media Playerフレームワークのクラスを使用して、ユーザのiPodライブラリからトラックを再生することもできます。

アプリケーションにビデオ再生を組み込みたい場合は、Media Playerフレームワークのクラスを使用します。iPhone OS 3.2では、ビデオを再生するためのインターフェイスが大幅に変更され、柔軟性が大幅に高まりました。常にフルスクリーンモードで再生するのではなく、ビューを任意のサイズでユーザインターフェイスに組み込めるようになりました。また、トラックでの早送りおよび巻き戻しや、トラックの開始点および停止点の設定、さらにビデオフレームのサムネイル画像の生成機能など、プログラムの中からより直接的に再生を制御できるようになりました。

新しいインターフェイスを使うために既存のMedia Playerコードを移植する方法については、「Media Playerフレームワーク使用に関する移植の重要なヒント」（25 ページ）を参照してください。OpenGL ESのハードウェア機能の詳細、およびiPhone OSアプリケーションでのOpenGL ESの使いかたについては、『*OpenGL ES Programming Guide for iPhone OS*』を参照してください。

イベント処理

Multi-Touchテクノロジーは、iPhoneアプリケーションおよびiPadアプリケーションのどちらにとっても不可欠です。iPhoneアプリケーションと同様に、iPadアプリケーションのイベント処理モデルは、アプリケーションのビューで1つまたは複数のタッチイベントを受け取ることを基本としています。そしてビューは、そのタッチイベントを、アプリケーションのコンテンツを変更したり操作したりするアクションに変換するという役割を担います。

タッチイベントの受け取りや処理のプロセスはiPadアプリケーションでも変わっていませんが、iPhone OS 3.2は、ジェスチャを統一的な方法で検出するためのサポートを提供するようになりました。Gesture Recognizerによって、特にスワイプ、ピンチ、回転ジェスチャの検出のためのインターフェイスが、そしてこれらのジェスチャを使用してさらなる動作をトリガするためのインターフェイスが簡素化されます。また、Gesture Recognizerクラスの基本セットを拡張し、アプリケーションで使用するカスタムジェスチャへのサポートを追加することもできます。

Gesture Recognizerの使いかたの詳細については、「Gesture Recognizer」（51 ページ）を参照してください。

デバイス統合のサポート

iPhoneの特徴的な機能の多くはiPadでも利用できます。具体的には、次の機能のサポートをiPadアプリケーションに組み込むことができます。

- 加速度センサー
- Core Location
- マップ(Maps)（MapKitフレームワークを使用）
- 環境設定（アプリケーション内または、「設定(Settings)」アプリケーションからの提示）
- アドレスブックの連絡先
- 外部ハードウェアアクセサリ
- ピアツーピアBluetooth接続（Game Kitフレームワークを使用）

iPadデバイスはカメラを備えていませんが、それでもiPadデバイスを使ってユーザの写真にアクセスできます。画像ピッカーインターフェイスが、すでにデバイスにあるフォトライブラリからの画像選択をサポートします。

iPadデバイスの新機能

iPhoneアプリケーションとiPadアプリケーションにはたくさんの類似点がありますが、iPadデバイスでは、大きく異なるタイプのアプリケーションの作成を可能にする新機能も利用できます。これらの新機能があるので、移植の過程で既存のiPhoneアプリケーションの見直しを行う価値があるかもしれません。新機能を使う利点は、iPadデバイスに対するアプリケーションの親和性が高まることです。

より広いスペース

iPhoneアプリケーションとiPadアプリケーションの最大の違いは、コンテンツ表示に利用できる画面スペースの広さです。iPadデバイスの画面サイズは、1024×768ピクセルです。アプリケーションをこの大きな画面に対応するように修正する方法は、既存のiPhoneアプリケーションの現在の実装に大きく依存します。

アプリケーションのコンテンツがすでに画面一杯に表示されるゲームのような没入型のアプリケーションの場合、アプリケーションを拡大するのはよい戦略です。ゲームを拡大するときに、増えたピクセルを利用して、ゲーム環境およびその中のモノの詳細さを高めることができます。利用可能なスペースが増えたことを受けて、ゲーム環境に新しいコントロールやステータスを加えることも検討するべきです。コードを適切に切り分ければ、両方のタイプのデバイスに対して同じコードを使い、iPadでレンダリングするときには単に詳細度を高めるだけということができるかもしれません。

情報を提示するために標準のシステムコントロールを使う生産性型アプリケーションについては、iPadデバイスを活用できるように設計されている新しいビューで既存のビューを置き換えたい、とほぼ確実に思うことでしょう。この機会に設計を見直してください。たとえば、ユーザが大きなデータセット内を移動するためにアプリケーションでNavigation Controllerを使用している場合、新しいユーザインターフェイス要素の一部を活用してそのデータをより効率的に提示できるかもしれません。

ユーザインターフェイスを際立たせる新しい要素

広くなった画面スペースやiPadが提供する新機能をサポートするために、iPhone OS 3.2は以下の新しいクラスとインターフェイスを備えています。

- **Split View**は2つのカスタムビューを横に並べて表示する方法です。Split Viewは、ナビゲーションベースのインターフェイスや、マスタと詳細が対になったその他のタイプのインターフェイスを補完するのに適しています。
- **Popover**は、既存のビューの手前に一時的にコンテンツを重ねます。Popoverを使用して、ツールパレットやオプションメニューを実装したり、アプリケーションのメインコンテンツからユーザの注意をそらすことなく異なる種類の情報を示したりできます。
- モーダルモードで表示されたコントローラは、設定可能な**表示スタイル**をサポートするようになりました。これは、モーダルビューが画面全体を覆うのか、画面の一部のみを覆うのかを決定します。
- Toolbarは、ビューの上部と下部に配置できるようになりました。画面サイズが大きくなったことによって、Toolbarにより多くの項目を含めることも可能になります。
- レスポンダオブジェクトは、**カスタム入力ビュー**をサポートする。カスタム入力ビューは、オブジェクトがファーストレスポンダになったときに画面の下部から上にスライドするビューです。これまでは、Text FieldとText Viewのみが入力ビュー（キーボード）をサポートしており、そのビューは変更できませんでした。これからは、作成したどのカスタムビューとでも入力ビューを関連付けることができます。カスタム入力ビューの指定については、「入力ビューと入力アクセサリビュー」（79 ページ）を参照してください。
- レスポンダもカスタムの**入力アクセサリビュー**を持つことができます。入力アクセサリビューは、レスポンダの入力ビューの上部に配置された状態で、オブジェクトがファーストレスポンダになったときに入力ビューと一緒にスライドして出てきます。この機能のもっとも一般的な

用法は、カスタムToolbarまたはその他のビューをキーボードの上部に配置するというものです。カスタム入力ビューの指定については、「入力ビューと入力アクセサリビュー」（79 ページ）を参照してください。

iPadアプリケーションのインターフェイスについて考える際には、新しい要素の組み込みを適宜検討してください。新しい要素の中には、コンテンツをより自然に表示してくれるものがあります。たとえばSplit Viewは、ナビゲーションインターフェイスの代替（補完）として適していることがよくあります。また、ほかの要素を通じて新しい機能を活用してアプリケーションの機能を拡張できます。

Split View、Popover、新しいモーダル表示スタイルの使いかたの詳細については、「ビューとView Controller」（39 ページ）を参照してください。入力ビューおよび入力アクセサリビューについては、「テキストの独自処理と入力」（79 ページ）を参照してください。ユーザインターフェイス全般の設計方法に関するガイダンスについては、『*iPad Human Interface Guidelines*』を参照してください。

テキスト入力および表示のサポート強化

これまでのバージョンのiPhone OSでは、テキストサポートは、単純なテキスト入力および表示のために最適化されていました。これからは、より大きなiPad画面のおかげで、より洗練されたテキスト編集や表示が可能になっています。さらに、iPadデバイスには物理的なキーボードを接続できるので、集中的なテキスト入力も可能です。強化されたテキスト入力および表示をサポートするために、アプリケーションで使用できる次のような新しい機能もiPhone OS 3.2に含まれています。

- Core Textフレームワークは、洗練されたテキストレンダリングやレイアウトをサポートする。
- UIKitフレームワークには、テキストをサポートする次のような機能強化が含まれる。
  - カスタムビューでシステムキーボードからの入力を受け取れるようにする新しいプロトコル
  - スペルチェックを管理する新しい`UITextChecker`クラス
  - `UIMenuController`クラスによって管理される編集メニューにカスタムコマンドを追加するためのサポート
- Core Animationには今度から`CATextLayer`クラスが含まれており、これを使用してレイヤ内にテキストを表示できる。

これらの機能により、単純なテキスト入力コントロールから洗練されたテキスト編集アプリケーションまで、何でも作れるようになります。たとえば、システムキーボードとのやり取りができることによって、基本的な入力から複雑なテキスト選択および編集動作に至るまで何でも処理できるカスタムText Viewを作ることが可能になります。そしてそのテキストを描画するために、Core Textフレームワークにアクセスできるようになりました。Core Textフレームワークを使って、カスタムレイアウトや、複数のフォント、複数の色、その他のスタイル属性を使用したテキストを表示できます。

アプリケーションでテキストを処理するこれらのテクノロジーの使いかたの詳細については、「テキストの独自処理と入力」（79 ページ）を参照してください。

外部ディスプレイおよびプロジェクタのサポート

iPadは、サポートされているケーブルで外部ディスプレイに接続できます。この接続を使用して、アプリケーションでは、デバイスのメイン画面上のコンテンツ以外にもコンテンツを表示できます。ケーブルに応じて、最大720p (1280×720)の解像度でコンテンツを出力できます。1024×768の解像度も、このアスペクト比の方が望ましければ利用できます。

外部ディスプレイにコンテンツを表示するには、次の手順を実行します。

1. `UIScreen`クラスの`screens`クラスメソッドを使用して、外部ディスプレイが利用できるか確認する。
2. 外部画面が利用できる場合は画面オブジェクトを取得し、画面オブジェクトの`availableModes`プロパティの値を確認する。画面がサポートしている設定が、このプロパティに含まれています。
3. 希望の解像度に対応する`UIScreenMode`オブジェクトを選択し、画面オブジェクトの`currentMode`プロパティに割り当てる。
4. コンテンツを表示するために新しいウインドウオブジェクト(`UIWindow`)を作成する。
5. 画面オブジェクトを新しいウインドウの`screen`プロパティに割り当てる。
6. ウインドウを設定する（ビューを追加、またはOpenGL ESレンダリングコンテキストを設定）。
7. ウインドウを表示する。

重要： ウインドウを表示する前に、必ず、ウインドウに画面オブジェクトを割り当てる必要があります。ウインドウがすでに表示されている状態で画面を変更することはできますが、これは負荷の高い処理のため推奨しません。

画面モードオブジェクトは、画面がサポートしている具体的な解像度を示します。多くの画面は複数の解像度をサポートしますが、その中には異なるピクセルアスペクト比の解像度が含まれることがあります。どの画面モードを使用すべきかの決定は、パフォーマンスおよび、どの解像度がユーザインターフェイスの必要性にもっとも適しているかということに基づいてなされるべきです。描画を開始する準備が整ったら、`UIScreen`オブジェクトによって与えられる境界範囲を使用して、コンテンツをレンダリングするための適切なサイズを取得します。画面の境界範囲は、デベロッパがコンテンツの描画に集中できるように、あらゆるアスペクト比のデータも考慮します。

画面が接続されたり、切り離されたりしたことを検出したければ、画面接続および切り離しの通知を受け取るように登録します。画面および画面通知の詳細については、『*UIScreen Class Reference*』を参照してください。画面モードの詳細については、『*UIScreenMode Class Reference*』を参照してください。

文書およびファイル処理の正式サポート

生産性型アプリケーションを作成する機能をサポートするために、iPhone OS 3.2は、文書やファイルの作成および処理のサポートを目的としたいくつかの新しい機能を備えています。

- アプリケーションは、特定のファイル形式を開くことができるアプリケーションとして自身を登録できるようになりました。このサポートによって、ファイルを扱う必要のあるアプリケーション（電子メールプログラムなど）は、ファイルをほかのアプリケーションに渡すことができます。
- UIKitフレームワークは、形式が不明のファイルとやり取りするための`UIDocumentInteractionController`クラスを提供するようになりました。このクラスを使用して、ファイルをプレビューしたり、内容をペーストボードにコピーしたり、ファイルを開くためにほかのアプリケーションにファイルを渡したりできます。

もちろん、iPadアプリケーションでファイルを操作できるとしても、ファイルを決してアプリケーションの中心的な対象にすべきではないと覚えておくことが大切です。iPhone OSにはファイルを開いたり保存したりするためのパネルがありませんが、これには相応の理由があります。特にファイル保存パネルは、すべてのデータを保存するのはユーザの責任であるということを示唆します。しかしこれはiPhoneアプリケーションが使用すべきモデルではありません。その代わりにアプリケーションは、アプリケーションが終了するときやシステムによって中断されたときに、データの喪失を防ぐために、データをインクリメンタルに保存すべきです。これを行うために、アプリケーションは、ユーザのコンテンツの作成および適切なタイミングでの保存の管理に責任を持たなければなりません。

文書およびファイルとのやり取りを行う方法の詳細については、「アプリケーションのコア設計」（27 ページ）を参照してください。

PDFの生成

iPhone OS 3.2では、UIKitによって、アプリケーションからPDFコンテンツを作成するためのサポートが導入されています。このサポート機能を使用して、アプリケーションのホームディレクトリまたは、アプリケーションのコンテンツに組み込めるデータオブジェクトで、PDFファイルを作成できます。以前から利用可能なネイティブ描画テクノロジーを活用しているため、PDFコンテンツの作成は簡単です。PDFキャンバスを準備したあと、UIKit、Core Graphics、Core Textを使用して、必要なテキストおよびグラフィックスを描画できます。また、PDF作成機能を使用して、PDFコンテンツにリンクを埋め込むこともできます。

新しいPDF作成機能の使いかたの詳細については、「PDFコンテンツの生成」（72 ページ）を参照してください。

プロジェクトの開始

iPadアプリケーション作成のプロセスは、新しいアプリケーションを作成するのか、既存のiPhoneアプリケーションを移植するのかによって異なります。新しいアプリケーションを作成する場合は、Xcodeのテンプレートを使って作成を開始できます。しかし、iPadはiPhone OSを実行するため、既存のiPhoneアプリケーションを、iPadで（互換モードではなく）ネイティブに実行するように移植することが可能です。既存のアプリケーションを移植するには、iPadデバイスをサポートするようにコードとリソースに少し変更を加える必要がありますが、よく整理されたアプリケーションでは、この作業は比較的容易です。また、Xcodeによってもプロジェクトの設定プロセスのほとんどが自動化され、移植のプロセスが簡単になります。

既存のiPhoneアプリケーションを移植することにした場合は、できあがったアプリケーションをどのように配布したいのか、そしてどのような開発プロセスが最適なのかの両方を検討するべきです。移植方法の選択肢と、それぞれの内容を以下に示します。

- すべてのデバイスタイプ向けに最適化されたユニバーサルアプリケーションを作成。
- 1つのXcodeプロジェクトを使用し、2つの別のアプリケーションを作成（1つはiPhoneおよびiPod touchデバイス用、もう1つはiPadデバイス用）。
- 別々のXcodeプロジェクトを使用し、デバイスタイプごとにアプリケーションを作成。

Appleでは、ユニバーサルアプリケーションまたは1つのXcodeプロジェクトの作成を強くお勧めします。どちらの手法でも既存のiPhoneアプリケーションのコードを再利用できます。ユニバーサルアプリケーションを作成すれば、すべてのデバイスタイプをサポートする1つのアプリケーションを販売できます。これは、ユーザにとって非常にシンプルな体験になります。もちろん、2つの別々のアプリケーションを作成するほうが、ユニバーサルアプリケーションの作成よりも開発やテストの時間がかからないという可能性はあります。

ユニバーサルアプリケーションの作成

ユニバーサルアプリケーションとは、iPhone、iPod touch、およびiPadデバイス向けに最適化され、実行される単一のアプリケーションです。このようなバイナリを作成することが、ユーザが持っているどのデバイスでもアプリケーションを実行できるということを保証し、ユーザ体験を大幅に単純化します。しかしこのようなバイナリの作成は、デベロッパにとっては、多少の追加作業を伴うものです。よく整理されたアプリケーションでも、両方のタイプのデバイスで問題なく動作させるためには何らかの作業が必要です。

以降の各セクションでは、確実にどのタイプのデバイス上でもネイティブに実行されるようにするために、既存のアプリケーションに加えなければならない主な変更を取り上げます。

Xcodeプロジェクトの設定

ユニバーサルアプリケーション作成の最初のステップは、Xcodeプロジェクトの設定です。新しいプロジェクトを作成する場合は、Window-Based Applicationテンプレートを使ってユニバーサルアプリケーションを作成できます。既存のプロジェクトを更新する場合は、次のようにXcodeの「現在のターゲットをiPad用にアップグレード(Upgrade Current Target for iPad)」コマンドを使ってプロジェクトを更新できます。

1. Xcodeプロジェクトを開く。
2. 「ターゲット(Targets)」セクションで、ユニバーサルアプリケーションに更新したいターゲットを選択する。
3. 「プロジェクト(Project)」＞「現在のターゲットをiPad用にアップグレード(Upgrade Current Target for iPad)」を選択し、指示に従って1つのユニバーサルアプリケーションを作成する。

   Xcodeは、いくつかのビルド設定をiPhoneとiPadの両方をサポートするように変更し、プロジェクトを更新します。

重要： 既存のプロジェクトを移行する場合は、必ず「現在のターゲットをiPad用にアップグレード(Upgrade Current Target for iPad)」コマンドを使ってください。ファイルを手作業で移行しないでください。

プロジェクトに対する主な変更点は、「Targeted Device Family」ビルド設定を「iPhone/Pad」に設定することです。プロジェクトの「ベースSDK (Base SDK)」も、まだ変更されていなければ、通常「iPhoneデバイス3.2 (iPhone Device 3.2)」に変更されます（iPadをターゲットとするには、3.2 SDKを使って開発しなければなりません）。アプリケーションをiPhoneおよびiPod touchデバイスで実行できるようにするため、プロジェクトの配置ターゲットは変更すべきではなく、以前のバージョンのSDK（3.1.3など）であるべきです。

ビルド設定の更新に加え、Xcodeは、iPadをサポートするための新しいメインnibファイルも作成します。移行されるのはそのメインnibファイルだけです。アプリケーションの既存のView Controller用の新しいnibファイルは、デベロッパが作成しなければなりません。アプリケーションの`Info.plist`も、iPadでの実行時に新しいメインnibファイルのロードをサポートするように、更新されます。

iPhone OS 3.1.3またはそれ以前で実行するときは、iPhone OS 3.2で導入されたシンボルをアプリケーションで使用してはいけません。たとえば、iPhone OS 3.1で実行中にアプリケーションが`UISplitViewController`クラスを使おうとすると、シンボルが利用できないためアプリケーションはクラッシュします。この問題を避けるためには、シンボルを使用する前にコードでランタイムチェックを実行し、その特定のシンボルが利用できることを確認しなければなりません。必要なランタイムチェックの実行方法については、「新しいシンボル用のランタイムチェックの追加」（22 ページ）を参照してください。

Info.plist設定の更新

iPhoneおよびiPod touchデバイスでアプリケーションが適切に動作することを確実にするために、`Info.plist`の既存キーのほとんどは変更せずに置いておくべきです。ただし、iPadデバイスをサポートするために、`Info.plist`に`UISupportedInterfaceOrientations`キーを追加するべきです。アプリケーションの機能によっては、iPhone OS 3.2で導入されたほかの新しいキーも追加するとよいでしょう。

iPadアプリケーションをiPhoneアプリケーションとは異った設定にする必要がある場合は、iPhone OS 3.2以降で`Info.plist`キーにデバイス固有の値を指定できます。`Info.plist`ファイルのキーを読むとき、システムは次のパターンを使用してそれぞれのキーを解釈します。

key_root~<*platform*>~<*device*>

このパターンでは、*key_root*の部分はキーの元の名前を表しています。<*platform*>と<*device*>の部分はどちらも、任意で指定できるキー末尾であり、キーを特定のプラットフォームやデバイスに適用するために使用できます。platformとして`iphoneos`という文字列を指定すると、そのキーはすべてのiPhone OSアプリケーションに適用されるということを意味します（もちろん、そもそもアプリケーションがiPhone OSのみに配布される場合は、platformの部分をすべて省略できます）。キーを特定のデバイスに適用するには、次の値のいずれかを使用できます。

- `iphone` - iPhoneデバイスに適用されるキー
- `ipod` - iPod touchデバイスに適用されるキー
- `ipad` - iPadデバイスに適用されるキー

たとえば、iPhoneおよびiPod touchデバイスではアプリケーションを縦向き(portrait)で起動したいが、iPadでは右ホームの横向き(landscape-right)で起動したいことを示す場合は、次のキーで`Info.plist`を設定します。

```
<key>UIInterfaceOrientation</key>
<string>UIInterfaceOrientationPortrait</string>
<key>UIInterfaceOrientation~ipad</key>
<string>UIInterfaceOrientationLandscapeRight</string>
```

iPadアプリケーション用にサポートされているキーの詳細については、「アプリケーションのInfo.plistファイルに新たに追加されたキー」（30 ページ）を参照してください。

ビューおよびView Controllerの更新

iPadおよびiPhoneデバイスの両方をサポートするために必要なすべての変更の中でもっとも重要なのは、ビューおよびView Controllerの更新です。画面サイズが異なるということは、両方のタイプのデバイスをサポートするために、既存のインターフェイスを完全に設計し直す必要があるかもしれないということです。これは、異なるビューサイズをサポートするために別々のView Controller一式を作成（または、既存のView Controllerを変更）しなければならないということでもあります。

ビューに関して必要な主要な変更は、より大きな画面をサポートするようにビューのレイアウトを設計し直すことです。既存のビューを単純に拡大しても機能するでしょうが、通常、最良の結果は生み出しません。新しいインターフェイスでは、利用可能なスペースを活用し、必要に応じて新しいインターフェイス要素をうまく利用するべきです。そうすることによって、画面がより大きくなったiPhoneアプリケーションというのではなく、ユーザにとってより自然なインターフェイスになる可能性が高まります。

その他、ビューおよびView Controllerクラスを更新するときには、次のことを考慮しなければなりません。

- **View Controllerについて**
  - View Controllerでnibファイルを使用する場合は、View Controller作成時にデバイスタイプごとに異なるnibファイルを指定しなければならない。

- ビューをプログラムによって作成する場合は、両方のデバイスタイプをサポートするようにビュー作成コードを変更しなければならない。

- **ビューについて**
  - ビューに`drawRect:`メソッドを実装する場合、描画コードで異なるビューサイズの描画ができる必要がある。
  - ビューの`layoutSubviews`メソッドを実装する場合、レイアウトコードは異なるビューサイズに対応できなければならない。

iPhone OS 3.2で導入されたビューおよびView Controllerの一部の統合については、「ビューとView Controller」（39 ページ）を参照してください。

新しいシンボル用のランタイムチェックの追加

iPhone OS 3.2で導入されたシンボルを使用するコードはすべて、それらのシンボルが利用できることを検証するランタイムチェックによって保護しなければなりません。これらのチェックによって、システムで新しい機能が利用できるかどうかを判断でき、利用できない場合は代替コードパスをたどるようにすることができます。アプリケーションがiPhone OS 3.1またはそれ以前で実行されるときは、このようなチェックを行わなければ、アプリケーションのクラッシュを招きます。

実行できるチェックには、次のようにいくつかのタイプがあります。

- iPhone OS 3.2で導入されたクラスについては、`NSClassFromString`関数を使用して、そのクラスが定義されているかどうかを確認できます。関数が`nil`以外の値を返した場合、そのクラスは使用できます。以下に例を示します。

```
Class splitVCClass = NSClassFromString(@"UISplitViewController");
if (splitVC)
{
   UISplitViewController* mySplitViewController = [[splitVCClass alloc] init];
   // Split View Controllerを設定する
}
```

- 既存のクラスでメソッドが利用可能かどうかを判断するためには、`instancesRespondToSelector:`クラスメソッドを使用します。
- 関数が利用可能かどうかを判断するためには、関数名と`NULL`のブール比較を実行します。結果が`YES`であれば、その関数は使用できます。以下に例を示します。

```
if (UIGraphicsBeginPDFPage != NULL)
{
    UIGraphicsBeginPDFPage();
}
```

複数の配置ターゲットをサポートするコードの書きかたの例については、『*SDK Compatibility Guide*』を参照してください。

ランタイムチェックを使用して条件付きコードパスを作成

基盤のデバイスタイプに応じてコードが異なるパスをたどらなければならない場合、UIDeviceのuserInterfaceIdiomプロパティを使用してどのパスをたどるべきかを決定できます。このプロパティは、作成するインターフェイスのスタイルを示します（iPadまたはiPhone）。このプロパティはiPhone OS 3.2以降でのみ利用可能なため、呼び出しの前に、プロパティが利用できることを確認しなければなりません。そのためのもっとも簡単な方法は、以下に示すUI_USER_INTERFACE_IDIOMマクロを使うことです。

```
if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
{
     // デバイスは、iPhone 3.2またはそれ以降を実行するiPad
}
else
{
     // デバイスはiPhoneまたはiPod touch
}
```

リソースファイルの更新

一般にリソースファイルはアプリケーションのユーザインターフェイスを実装するために使用されるため、次の変更を加える必要があります。

- アプリケーションがiPhoneデバイスで起動されるときに表示されるDefault.pngファイルに加え、「さまざまな向き用の起動画像の提供」（32 ページ）に記載されているようにiPadデバイス用の新しい起動画像を追加しなければならない。
- 画像を使用する場合、iPadデバイスをサポートするためにより大きな（より解像度の高い）画像の追加が必要になる可能性がある。
- nibファイルを使用する場合、iPadデバイス用に新しいnibファイル一式を準備する必要がある。
- iPadのアプリケーションアイコンの大きさは、72×72ピクセルでなければならない。

各プラットフォームに異なるリソースファイルを使用するときは、条件付きでコードを実行するのと同様に、条件付きでリソースをロードできます。ランタイムチェックの使いかたの詳細については、「ランタイムチェックを使用して条件付きコードパスを作成」（23 ページ）を参照してください。

1つのXcodeプロジェクトを使用して、2つのアプリケーションを作成

iPhone開発およびiPad開発の両方に1つのXcodeプロジェクトを使用することによって、2つの別のアプリケーション間でコードを共有できるようになり、開発プロセスが非常に簡素化されます。Xcodeの「プロジェクト(Project)」メニューには新しい「現在のターゲットをiPad用にアップグレード(Upgrade Current Target for iPad)」コマンドがあり、このコマンドを使って、既存のiPhoneプロジェクトにiPad用のターゲットを簡単に追加できます。このコマンドを使用する際は、次の手順に従ってください。

1. 既存のiPhoneアプリケーションのXcodeプロジェクトを開く。

2. iPhoneアプリケーションのターゲットを選択する。

3. 「プロジェクト(Project)」＞「現在のターゲットをiPad用にアップグレード(Upgrade Current Target for iPad)」を選択し、指示に従って2つのデバイス固有アプリケーションを作成する。

重要： 必ず「現在のターゲットをiPad用にアップグレード(Upgrade Current Target for iPad)」コマンドを使って既存のプロジェクトを移行してください。ファイルを手作業で移行しないでください。

「現在のターゲットをiPad用にアップグレード(Upgrade Current Target for iPad)」コマンドによって、新しいiPadターゲットが作成され、iPadプロジェクト用に新しいnibファイルが作成されます。このnibファイルはプロジェクトの既存のnibファイルに基づいたものですが、nibファイルのウインドウおよび最上位ビューはiPad画面用にサイズが調整されます。最上位のビューはサイズが変更されますが、このコマンドでは、組み込まれたサブビューのサイズや位置は変更されず、ビューのレイアウトは基本的に元のままです。これらの組み込まれたビューのレイアウトを調整するかどうかはデベロッパの自由です。

新しいターゲットの作成も、プロジェクト更新の最初の一歩にすぎません。新しいnibファイルのレイアウト調整に加え、これらのnibファイルを管理するためにView Controllerコードを更新しなければなりません。ほぼすべての場合において、iPad版のアプリケーションインターフェイスを管理するために（特にそのインターフェイスがiPhoneインターフェイスとまったく異なる場合は）新しいView Controllerクラスを定義したいと思うことでしょう。条件付きのコンパイルを使用して（以下に示すように）、異なるView Controllerの作成を調整できます。ビュー階層をほとんど、あるいはまったく変更しない場合は、既存のView Controllerクラスを再利用することもできます。そのような場合、同様に条件付きのコンパイルを使用して、基盤のデバイスタイプに適したnibファイルでView Controllerを初期化します。

次の例では、ターゲットの「ベースSDK (Base SDK)」が「iPhoneデバイス3.2 (iPhone Device 3.2)」またはそれ以降に設定されていれば、iPad View Controllerコードが使われます。iPhoneアプリケーションターゲットの「ベースSDK (Base SDK)」では、以前のバージョンのオペレーティングシステムに設定されるので、コードの`#else`部分が使われます。

```
#if __IPHONE_OS_VERSION_MAX_ALLOWED >= 30200
  MyIPadViewController* vc;
  // iPad View Controllerの作成
#else
  MyIPhoneViewController* vc;
  // iPhone View Controllerの作成
#endif
```

View Controllerに加え、iPhoneおよびiPadデバイスで共有されているクラスにはすべて、デバイス固有コードを分離するための条件付きコンパイルマクロを含める必要があります。特定のクラスやメソッドが利用できるかどうかを確認するためにランタイムチェックを使用することもできますが、そうした場合、どちらか一方のデバイスでしかたどることのないコードパスが追加され、実行可能ファイルの容量が増加するだけです。コンパイラにこのコードを削除させることで、コードを簡潔に保つことができます。

デバイスタイプごとにコードを条件付きでコンパイルすることに加えて、適切だと思うデバイス固有の機能を何でも自由に組み込めます。この文書の残りの章では、iPadデバイスでのみサポートされる機能について説明します。これらの機能を使用して作成したコードは、iPadデバイスでのみ実行されなければなりません。

条件付きコンパイルおよび使用可能マクロの詳細については、『*SDK Compatibility Guide*』を参照してください。

一から始める

iPadアプリケーションを一から作成するには、iPhoneアプリケーションを一から作成する場合と同じプロセスに従います。もっとも目に付く違いは、ユーザインターフェイスを表示するために作成するビューのサイズです。新しいアプリケーションの良いアイデアがあるのなら、一から始めるという決定は当然のことです。しかし、既存のiPhoneアプリケーションがあり、そのアプリケーションの既存のXcodeプロジェクトやリソースを活用して2つの別々のアプリケーションを作るべきか、全デバイスタイプをサポートするユニバーサルアプリケーションを作るべきかどうかを迷っているだけならば、次の質問を自分自身に問いかけてみてください。

- アプリケーションのデータモデルオブジェクトは、それを描画するビューと密接に統合されているか。
- iPad版のアプリケーションにかなり多くの機能を追加する予定があるか。
- アプリケーションにデバイス固有の部分があまりにも多く、移植のためにコードの大部分を変更する必要があるか。

上記の質問への答えがいずれか1つでも「はい」であれば、iPadデバイス用に別のXcodeプロジェクトを作成することを検討すべきです。いずれにせよ大部分のコードを書き直さなければならない場合は、一般に、別のXcodeプロジェクトを作成する方が簡単です。別のプロジェクトを作成すると、コードがその他のデバイスで動作するかどうかを心配することなく、iPadデバイス用のコードを自由に作成できます。

Media Playerフレームワーク使用に関する移植の重要なヒント

Media Playerフレームワークの`MPMoviePlayerController`クラスを使用するアプリケーションを移植する場合は、そのアプリケーションをiPhone OS 3.2で実行させたければ、コードを変更しなければなりません。このクラスの古いバージョンでは、単純なインターフェイスを使用したフルスクリーン再生しかサポートしていません。新しいバージョンでは、フルスクリーン再生と部分スクリーン再生の両方をサポートしており、再生のさまざまな側面をより細かく制御できます。しかし、新しい動作をサポートするために、古いメソッドやプロパティの多くが廃止されたり、動作が大幅に変更されたりしました。このため、iPhone OS 3.2では、古いコードは期待通りには動作しません。

既存のコードに影響する可能性がもっとも高い主要な変更は次の通りです。

- ムービープレーヤーコントローラは、ムービーの表示を管理しなくなりました。その代わりに、ムービープレーヤーコントローラは、ムービーコンテンツの再生サーフェスとして機能するViewオブジェクトを提供します。
- `play`メソッドの呼び出しは、これまで通りムービーの再生を開始しますが、ムービーが表示されていることを保証はしません。

ムービーを表示するためには、`MPMoviePlayerController`オブジェクトからビューを取得し、そのビューをビュー階層に追加しなければなりません。通常、これを、View Controllerの1つから行います。たとえば、ビューをプログラムでロードする場合は、これを`loadView`メソッドで行うことができます。そうでなければ、`viewDidLoad`メソッドで行うことができます。View Controllerを表示した後、ムービーの再生を始められます。あるいはムービープレーヤーの組み込みコントロールを表示することによってユーザに再生を開始させることができます。

ムービーをフルスクリーンモードで表示したい場合は、2つの方法があります。もっとも簡単な方法は、`MPMoviePlayerViewController`クラスのインスタンス（iPhone OS 3.2で新しく登場）を使用してムービーを表示することです。このクラスは、`UIViewController`を継承するため、ほかのView Controllerと同じようにアプリケーションで表示できます。`presentMoviePlayerViewControllerAnimated:`メソッドを使用してモーダルモードで表示される場合、表示に使用されるトランジションアニメーションを含め、ムービーの表示はこれまで`MPMoviePlayerController`クラスによって提供されていた体験を再現します。View Controllerを閉じるには、`dismissMoviePlayerViewControllerAnimated`メソッドを使用します。

ムービーをフルスクリーンで表示するもう1つの方法は、ビューを`MPMoviePlayerController`オブジェクトからビュー階層に組み込み、その`setFullscreen:animated:`メソッドを呼び出すことです。このメソッドはムービーの表示を、フルスクリーンモードと、ビュー内だけのムービーコンテンツ表示との間で切り替えます。

既存のコードに加えなければならない変更のほかに、iPhone OS 3.2で実行されるアプリケーションで使用できる次のようないくつかの新しい機能があります。

- 新しいムービープレーヤーコントローラを作成せずに、プログラムで再生されているムービーを変更できる。
- 現在のムービーを通して、プログラムで開始、停止、一時停止、スクラブ（早送り、巻き戻し）を行うことができる。
- ビデオコンテンツの手前に追加のビューを組み込むことができる。
- ムービープレーヤーコントローラによって、カスタムの背景コンテンツを組み込める背景ビューが提供される。
- ムービーの開始、停止時間の両方を設定できる。またムービーを繰り返し再生したり、自動的に開始したりできる（これまでは、開始時間の設定のみが可能）。
- ムービーのフレームからサムネイル画像を生成できる。
- ムービーの再生時間、現在の再生位置、現在の再生速度など、現在のムービーの状態に関する一般的な情報を取得できる。
- ムービープレーヤーコントローラは、大部分の状態変更について通知を生成する。

アプリケーションのコア設計

iPadアプリケーションは、iPhone OSで実行されるため、既存のiPhoneアプリケーションと同じオブジェクトとインターフェイスをすべて使用できます。その結果として、この2つのアプリケーションタイプのコアアーキテクチャはまったく同じです。ただし、iPhone OS 3.2では、iPadアプリケーションでは利用できて、iPhoneアプリケーションでは利用できない新機能がいくつか導入されています。この章では、これらの機能について説明し、これらの機能をアプリケーションで使用する方法と状況を示します。

iPadアプリケーションのアーキテクチャ

iPhoneアプリケーションとiPadアプリケーションのアーキテクチャはまったく同じですが、それぞれのデバイスタイプをサポートするために、コードやリソースファイルを調整しなければならない場合があります。図3-1は、iPhoneアプリケーションの基本アーキテクチャをまとめた図です。この図には、最もよく使われる主要なオブジェクトが示されています。また、表3-1は、これらの各オブジェクトタイプの役割を示しています（iPhone（つまりiPad）アプリケーションのコアアーキテクチャの詳細については、『*iPhone Application Programming Guide*』を参照してください）。

図 3-1 iPadアプリケーションの主要オブジェクト

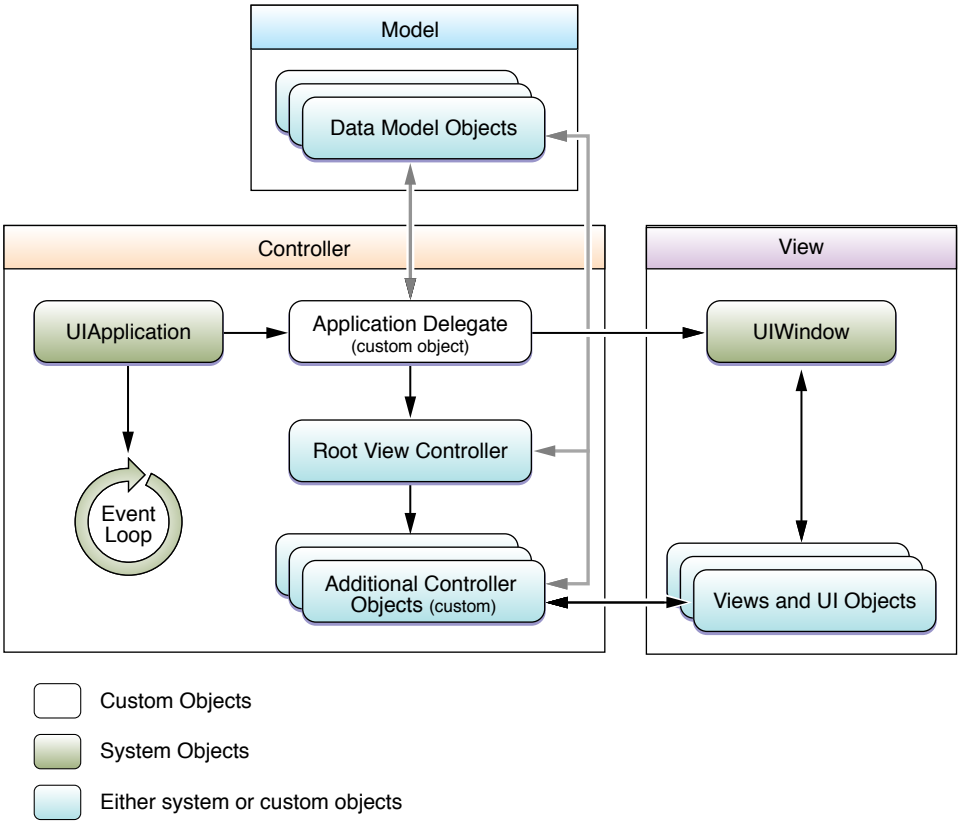


表 3-1 アプリケーションでの各オブジェクトの役割

オブジェクト	説明
`UIApplication` オブジェクト	`UIApplication`オブジェクトは、アプリケーションのイベントループを管理して、アプリケーションのその他の上位レベル動作を調整します。このオブジェクトはそのまま使用します。これを使用して、主にアプリケーションの外観のさまざまな側面を設定します。アプリケーションレベルのカスタムコードは、このオブジェクトと連携して動作するアプリケーションデリゲートオブジェクトに含まれます。

オブジェクト	説明
アプリケーションデリゲートオブジェクト	アプリケーションデリゲートは、アプリケーションの起動時にデベロッパによって提供されるカスタムオブジェクトです。通常、このオブジェクトは、アプリケーションのメインnibファイルに埋め込まれています。このオブジェクトの主な仕事は、アプリケーションを初期化してウインドウを画面に表示することです。`UIApplication`オブジェクトは、（着信メッセージのために）アプリケーションに割り込みを行う必要があるときや、（ユーザがホーム(Home)ボタンをタップしたために）アプリケーションを終了する必要があるときなど、アプリケーションレベルの特定のイベントが発生したときに、このオブジェクトに通知します。 iPadアプリケーションでも、アプリケーションの起動時と終了時の動作を調整するために、デリゲートオブジェクトを引き続き使用します。ただし、デバイスタイプごとに固有のサポートを行うには、デリゲートのメソッド内に条件チェックを含めなければならない場合もあります。具体的に言うと、一般的に起動時には、初期インターフェイス用に個別のnibファイルをロードする必要があります。同様に、初期化コードや終了コードがデバイスタイプに応じて異なる場合もあります。
データモデルオブジェクト	データモデルオブジェクトはアプリケーションのコンテンツを格納します。したがって、アプリケーションに固有のオブジェクトです。 データモデルオブジェクトでは、デバイスによる違いがほとんどないようにするのが理想的です。違いが生じる可能性があるのは、iPad固有の機能をサポートするためにデータモデルオブジェクトを追加または修正する場合だけです。
View Controllerオブジェクト	View Controllerオブジェクトは、アプリケーションのユーザインターフェイスの表示を管理します。また、データモデルオブジェクトとそのデータを表示するために使われるビューとの間のやり取りを調整します。`UIViewController`クラスは、すべてのView Controllerオブジェクトの基底クラスで、デベロッパがしなければならない作業を最小限に抑えられるように、かなりの量のデフォルトの動作を提供しています。 iPhoneアプリケーションを移植する場合、ほとんどの変更はビューとView Controllerで生じます。View Controllerをどの程度変更する必要があるかは、ビューをどの程度変更するかによって完全に決まります。変更が少ない場合は、既存のView Controllerを再利用して、少しの変更を加えれば各デバイスをサポートできます。変更が多い場合は、iPadアプリケーション用とiPhoneアプリケーション用に別々のView Controllerクラスを定義する必要があるかもしれません。
`UIWindow` オブジェクト	`UIWindow`オブジェクトは、アプリケーションの描画サーフェスを管理します。 iPadとiPhoneのどちらのアプリケーションでも、ウインドウの使いかたは本質的に同じです。ウインドウを作成してルートビューをインストールした後は、基本的にはウインドウを無視します。ユーザインターフェイスのあらゆる変更は、ウインドウオブジェクトではなくView Controllerを操作することによって行います。

オブジェクト	説明
ビューとUIオブジェクト	ビューとコントロールは、アプリケーションのコンテンツのビジュアル表現を提供します。UIKitフレームワークでは、テーブル、ボタン、ピッカーコントロール、テキストラベル、入力フィールドなどを実装するための標準的なビューを提供しています。また、`UIView`（または、その下位クラス）を直接サブクラス化して、カスタムビューを定義することもできます。 iPadアプリケーションでは、広くなったデバイス画面に合うようにビューを調整する必要があります。このような「調整」には、既存のビューのサイズを拡大する方法から、既存のビューの一部または全部を置き換える方法まであります。ビューを置き換えるのは極端なように思えますが、その方が良い結果になる場合もあります。特に、新しいビューにした方が、広がった分の画面スペースを有効に使用できる場合です。

既存のiPhoneアプリケーションをiPadに移植する場合、最も大きな変更部分は、アプリケーションのカスタムビューとViewControllerになります。その他の変更が必要になる場合もありますが、ビューとView Controllerは、ほぼ確実に変更しなければならない部分です。

iPhone OS 3.2で新規のビューとView Controllerを使用する方法の例については、「ビューとView Controller」（39 ページ）を参照してください。ユーザインターフェイスを構成する際に考慮すべき設計ガイドラインのリストについては、『*iPad Human Interface Guidelines*』を参照してください。

アプリケーションバンドル

iPadアプリケーションでは、iPhoneアプリケーションと同じバンドル構造を使用します。つまり、アプリケーションのコードとリソースのほとんどは、バンドルの最上位レベルのディレクトリに存在します。バンドルの内容も非常に類似しています。ただし、iPadアプリケーションでしか利用できない機能がいくつかあります。

アプリケーションのInfo.plistファイルに新たに追加されたキー

情報プロパティリストファイル（`Info.plist`ファイル）に、iPadアプリケーションに固有の機能をサポートするために使用するキーが追加されました。これらのキーのほとんどは省略可能ですが、必須のキーが1つと、強くお勧めするキーが1つあります。表 3-2に、新たに追加されたキーと、どのようなときにそれらをアプリケーションの`Info.plist`ファイルに含めべきかを示します。いつでも可能な限り、Xcodeで適切なビルド設定を変更することによって、`Info.plist`キーを変更するべきです。ただし、いくつかのキーの追加については、手動でファイルを編集しなければならない場合もあります。

表 3-2 iPhone OS 3.2で新たに追加されたInfo.plistのキー

キー	説明
UIDeviceFamily	（必須）アプリケーションがサポートするデバイスを識別します。Xcodeプロジェクトの「Targeted Device Family」ビルド設定で値を変更することによって、このキーの値を設定します。 明確にiPad用として作成するすべての新規アプリケーションには、自動的にこのキーを含めなければなりません。同様に、iPadをサポートするために移行するすべてのプロジェクトには、自動的にこのキーを含めなければなりません。
UILaunchImageFile	使用する起動画像の名前を識別する文字列を含みます。このキーが提供されない場合、アプリケーションはDefault.pngという名前の画像を探します。ユニバーサルアプリケーションでは、このキーを使用して、iPadアプリケーション用とiPhoneアプリケーション用に別々のデフォルト画像を指定できます。起動画像の詳細については、「さまざまな向き用の起動画像の提供」（32 ページ）を参照してください。
UISupportedInterface-Orientations	（推奨）アプリケーションが起動時にサポートする向きを指定する文字列の配列を含みます。指定可能な値は、UIInterfaceOrientation型で指定された定数です。 システムは、この情報を使用してアプリケーション用の適切な起動画像を選びます。詳細については、「さまざまな向き用の起動画像の提供」（32 ページ）を参照してください。同様に、アプリケーションは、指定の向きのいずれであっても初期ユーザインターフェイスを構成できるように準備しておかなければなりません。
CFBundleDocument-Types	アプリケーションで開くことができるドキュメントタイプを指定する辞書の配列を含みます。このキーを使用して、アプリケーションで開くことができる特定のファイルタイプをシステムに知らせます。 ドキュメントタイプ情報を指定するには、アプリケーションターゲットを選択して、インスペクタウインドウを開きます。「プロパティ(Properties)」ペインで、「書類のタイプ(Document Types)」セクションを使用して、ドキュメントタイプ情報を入力します。必須のフィールドは、「名前(Name)」フィールドと「UTI」フィールドだけです。その他のほとんどのフィールドは無視されます。
CFBundleIconFiles	アプリケーションのアイコンに使用する画像リソースを識別するファイル名の配列を指定します。アプリケーションがiPhoneとiPadの両方のデバイスをサポートする場合は、デバイスごとに別々の画像リソースを指定できます。システムは、それぞれのシステムに最適なサイズの画像を自動的に使用します。

アプリケーションのInfo.plistファイルに含めることができる全キーのリストについては、『*Information Property List Key Reference*』を参照してください。

さまざまな向き用の起動画像の提供

起動画像は、アプリケーションによって提供される静的な画像ファイルで、アプリケーションが最初に起動されたときにシステムによって表示されます。システムは、アプリケーションが起動したことを直ちにユーザにフィードバックするために起動画像を表示します。そして、アプリケーションが初期化されて最初のビューが表示されるまでの時間稼ぎをします。iPhoneアプリケーションは常に同じ初期インターフェイスの向きで起動されるため、`Default.png`リソースファイルに保存する起動画像は1つだけです。それとは対照的に、iPadアプリケーションは任意の向きのインターフェイスで起動できなければなりません。したがって、起動時の向きごとに別々の起動画像を持つ必要があります。

表3-3は、iPadアプリケーションバンドルの最上位レベルのディレクトリに含めることができる起動画像の一覧です。起動画像はすべてPNG形式でなければなりません。また、サイズは、指定の向きの画面サイズ（ステータスバーの高さを差し引いたサイズ）に一致していなければなりません。これらすべてのファイルをバンドルに含めることができますが、システムは、常に、汎用性の高い起動画像よりも、より限定された起動画像を選びます。

表 3-3　アプリケーションのデフォルトの起動画像ファイル

ファイル名	説明
`Default-PortraitUpsideDown.png`	縦長逆さまバージョンの起動画像を指定します。この画像の高さは1004ピクセル、幅は768ピクセルでなければなりません。縦長逆さまの向きの場合は、このファイルが`Default-Portrait.png`画像ファイルよりも優先されます。
`Default-LandscapeLeft.png`	横長左向きバージョンの起動画像を指定します。この画像の高さは748ピクセル、幅は1024ピクセルでなければなりません。横長左向きの場合は、このファイルが`Default-Landscape.png`画像ファイルよりも優先されます。
`Default-LandscapeRight.png`	横長右向きバージョンの起動画像を指定します。この画像の高さは748ピクセル、幅は1024ピクセルでなければなりません。横長右向きの場合は、このファイルが`Default-Landscape.png`画像ファイルよりも優先されます。
`Default-Portrait.png`	汎用の縦長バージョンの起動画像を指定します。この画像の高さは1004ピクセル、幅は768ピクセルでなければなりません。この画像は、通常の（逆さまではない）縦長の向きの場合に使用され、`Default.png`画像ファイルよりも優先されます。`Default-PortraitUpsideDown.png`画像ファイルが指定されていない場合は、縦長逆さまの向きの場合もこのファイルが使用されます。
`Default-Landscape.png`	汎用の横長バージョンの起動画像を指定します。この画像の高さは748ピクセル、幅は1024ピクセルでなければなりません。`Default-LandscapeLeft.png`または`Default-LandscapeRight.png`の画像ファイルが指定されていない場合は、この画像が代わりに使用されます。この画像は、`Default.png`画像ファイルよりも優先されます。

ファイル名	説明
`Default.png`	縦長のデフォルトの起動画像を指定します。この画像は、より限定された向きの画像が利用できない場合に使用されます。 iPadアプリケーションでは、この起動画像は使用せずに、より限定された向きの起動画像を使用することをお勧めします。iPhoneおよびiPod touchデバイスで動作するバージョンのアプリケーションでは、この画像ファイルを引き続き使用できます。

ユニバーサルアプリケーションを作成するときに、iPad用とiPhone用に別々の縦長画像と横長画像を含めたい場合は、Info.plistファイルで`UILaunchImageFile`キーを使用して、それぞれに別々のベース名を指定できます。このキーの値は、起動画像のベースファイル名を指定する文字列です。この新しい名前を、それぞれのプラットフォーム用の縦長および横長バージョンの起動画像を作成するために使用します。たとえば、`MyiPadImage.png`という文字列を`UILaunchImageFile~ipad`キーに割り当てた場合は、`MyiPadImage-Portrait.png`ファイルと`MyiPadImage-Landscape.png`ファイルをバンドルに含めれば、iPad専用の縦長画像と横長画像を指定できます。デバイス固有の`Info.plist`キーの作成方法の詳細については、「Info.plist設定の更新」（20 ページ）を参照してください。

起動時の画像が異なる場合でも、アプリケーションの設定手順は、iPhoneおよびiPod touchデバイスとほとんど同じです。`application:didFinishLaunchingWithOptions:`メソッドでは、だた1つの望ましい向きを使用して、ウインドウとビューをセットアップするべきです。つまり、ウインドウとビューの最初の向きを、デバイスの現在の向きに合わせようとするべきではありません。`application:didFinishLaunchingWithOptions:`メソッドが戻った直後に、システムは正しい起動時の向きをウインドウに通知し、標準的な処理を使用してコンテンツの向きを変更する機会を与えます。

iPadデバイスのドキュメントサポート

iPadデバイスで実行中のアプリケーションは、ドキュメントやファイルを扱ったり管理したりするための拡張サポート機能にアクセスできます。このサポート機能の目的は、アプリケーションが舞台裏でファイルを扱いやすいようにすることです。アプリケーションは、未知のタイプのファイルを検出した場合に、そのファイルの内容を表示したり、そのファイルを表示可能なアプリケーションを検索したりするための支援をシステムに要請できます。アプリケーションが特定のファイル形式を表示できる場合は、そのファイルを表示できるアプリケーションとしてシステムに登録することもできます。

ファイルのプレビューとオープン

アプリケーションが未知のタイプのファイルとやり取りをする必要がある場合は、`UIDocumentInteractionController`オブジェクトを使用して、それらのやり取りを行うことができます。Document Interaction Controllerは、システムと連携して、ファイルをその場でプレビューできるかどうかや、別のアプリケーションでオープンできるかどうかを判断します。アプリケーションはDocument Interaction Controllerと連携して、利用可能な選択肢を適切なタイミングでユーザに提示します。

アプリケーションでDocument Interaction Controllerを使用するには、次の手順を実行します。

1. 管理したいファイルごとにUIDocumentInteractionControllerクラスのインスタンスを作成します。
2. アプリケーションのユーザインターフェイスにそのファイルを表示します（通常は、ファイル名または何らかのアイコンをインターフェイスに表示します）。
3. ユーザがそのファイルを操作するときには、Document Interaction Controllerに次のいずれかのインターフェイスの表示を依頼します。
    - ファイルの内容を表示するファイルプレビュー用のビュー
    - ファイルのプレビュー、ファイル内容のコピー、別のアプリケーションを使用したファイルのオープンのための選択肢を含むメニュー
    - 別のアプリケーションを使用してファイルをオープンするようにユーザに指示するメニュー

ファイルとやり取りをする任意のアプリケーションで、Document Interaction Controllerを使用できます。ネットワークからファイルをダウンロードするプログラムは、この機能を最も必要としそうなプログラムです。たとえば、電子メールプログラムでは、Document Interaction Controllerを使用して、電子メールに添付されたファイルをプレビューしたりオープンしたりできます。もちろん、この機能を利用するために、ネットワークからファイルをダウンロードする必要はありません。

Document Interaction Controllerの作成と設定

Document Interaction Controllerを新規に作成するには、UIDocumentInteractionControllerクラスの新しいインスタンスを、管理対象のファイルで初期化して、適切なデリゲートオブジェクトを割り当てます。割り当てたデリゲートオブジェクトは、Document Interaction Controllerに、ビューを表示するために必要な情報を提供する役割を果たします。また、これらのビューが表示されているときに、デリゲートを使用して追加のアクションを実行することもできます。次のコードでは、Document Interaction Controllerを新規に作成して、デリゲートを現在のオブジェクトに設定しています。このメソッドの呼び出し元は、返されたオブジェクトを保持する必要がある点に注意してください。

```
- (UIDocumentInteractionController*)docControllerForFile:(NSURL*)fileURL
{
   UIDocumentInteractionController* docController =
       [UIDocumentInteractionController interactionControllerWithURL:fileURL];
   docController.delegate = self;

   return docController;
}
```

Document Interaction Controllerオブジェクトを得たら、そのプロパティを使用して、ファイルについての情報（名前、タイプ情報、パス情報など）を取得できます。Document Interaction Controllerは、ドキュメントのさまざまなサイズのアイコンを表すUIImageオブジェクトを含むiconsプロパティも持っています。ドキュメントをユーザインターフェイスに表示するときは、これらすべての情報を使用できます。

別のアプリケーションでファイルをオープンできるようにする場合は、Document Interaction Controllerのannotationプロパティを使用して、オープン用のアプリケーションにカスタム情報を渡すことができます。オープン用の別のアプリケーションが認識する形式で情報を提供するのは、デベロッパの責任です。たとえば、アプリケーションスイートで、同じスイート内の別のアプリケーションに

ファイルについての補足情報を渡したい場合、通常はこのプロパティが使われます。オープン用のアプリケーションは、起動時に渡されたoptionsディクショナリの`UIApplicationLaunchOptionsAnnotationKey`キー内のannotationデータを参照します。

Document Interaction Controllerの表示

ユーザがファイルとやり取りをする場合は、Document Interaction Controllerを使用して適切なユーザインターフェイスを表示します。次のいずれかのメソッドを使用して、ドキュメントのプレビューを表示するか、ファイルに対する適切なアクションをユーザに選ばせるかを選択できます。

- `presentOptionsMenuFromRect:inView:animated:`メソッドまたは`presentOptionsMenuFromBarButtonItem:animated:`メソッドを使用して、ユーザにさまざまな選択肢を表示する。
- `presentPreviewAnimated:`メソッドを使用して、ドキュメントのプレビューを表示する。
- `presentOpenInMenuFromRect:inView:animated:`メソッドまたは`presentOpenInMenuFromBarButtonItem:animated:`メソッドを使用して、ファイルを開くために使用するアプリケーションのリストをユーザに表示する。

上記の各メソッドは、適切なコンテンツを含むカスタムビューを表示しようとします。これらのメソッドを呼び出したときは、必ず戻り値をチェックして、ビューの表示が本当に成功したかどうかを確認する必要があります。表示されたインターフェイスに何もコンテンツが含まれていない場合、これらのメソッドは`NO`を返すことがあります。たとえば、`presentOpenInMenuFromRect:inView:animated:`メソッドは、ファイルを開くことが可能なアプリケーションが存在しない場合は`NO`を返します。

ファイルのプレビューを表示するメソッドを選んだ場合は、デリゲートオブジェクトで`documentInteractionControllerViewControllerForPreview:`メソッドを実装しなければなりません。ドキュメントのプレビューは、モーダルビューを使用して表示されます。したがって、戻り値となるView Controllerはこのモーダルドキュメントプレビューの親になります。このメソッドを実装していない場合、実装が`nil`を返した場合、または、指定されたView Controllerが別のモーダルView Controllerを表示できない場合は、ドキュメントのプレビューは表示されません。

標準では、Document Interaction Controllerは、自身が表示したビューの破棄を自動的に行います。ただし、必要であれば、`dismissMenuAnimated:`メソッドまたは`dismissPreviewAnimated:`メソッドを呼び出して、プログラムでビューを破棄することもできます。

アプリケーションでサポートするファイルタイプの登録

アプリケーションで特定のタイプのファイルを開ける場合は、そのことをシステムに登録するべきです。ファイルタイプのサポートを宣言するには、アプリケーションの`Info.plist`ファイルに`CFBundleDocumentTypes`キーを含めなければなりません。システムは、アプリケーションからこの情報を収集して、ほかのアプリケーションがDocument Interaction Controllerを介してアクセスできるように登録を管理します。

`CFBundleDocumentTypes`キーには、辞書の配列が含まれており、それぞれの辞書が特定のドキュメントタイプについての情報を示します。通常、1つのドキュメントタイプは、特定のファイルタイプと1対1に対応します。ただし、アプリケーションで複数のファイルタイプを同じように扱う場合は、これらのファイルタイプを1つのドキュメントタイプにまとめることができます。たとえば、アプリケーション固有のドキュメントタイプとして2つの異なるファイル形式がある場合は、古い

タイプと新しいタイプの両方を1つのドキュメントタイプエントリにまとめることができます。これによって、古いファイルも新しいファイルも同じタイプのファイルのように見えるので、同様に扱うことができます。

CFBundleDocumentTypes配列内のそれぞれの辞書には、次のキーを含めることができます。

- CFBundleTypeNameは、ドキュメントタイプの名前を指定します。
- CFBundleTypeIconFilesは、ドキュメントのアイコンとして使用する画像リソース用のファイル名の配列です。
- LSItemContentTypesには、このグループ内のサポート可能なファイルタイプを表すUTI typesについての文字列の配列が含まれます。
- LSHandlerRankは、このアプリケーションがドキュメントタイプを所有しているのか、それとも単にそのドキュメントタイプを開けるだけなのかを表します。

アプリケーション側から見ると、ドキュメントは、そのアプリケーションがサポートし、1つのエンティティとして扱う1つの（または複数の）ファイルタイプです。たとえば、画像処理アプリケーションでは、さまざまな画像ファイル形式を別々のドキュメントタイプとして扱うことによって、それぞれの形式に対応する動作をきめ細かく調節できます。逆に、ワードプロセッサアプリケーションでは、基盤の画像形式には関知しないため、単純にすべての画像形式を1つのドキュメントタイプとして扱うことが考えられます。

リスト 3-1に、カスタムファイルタイプを開けるアプリケーションのInfo.plistのXMLの例を示します。LSItemContentTypesキーは、このファイル形式に関連付けられているUTIを示します。CFBundleTypeIconFilesキーは、それを表示するときに使用するアイコンリソースを指します。

リスト 3-1　カスタムファイル形式用のドキュメントタイプ情報

```
<dict>
   <key>CFBundleTypeName</key>
   <string>My File Format</string>
   <key>CFBundleTypeIconFiles</key>
       <array>
           <string>MySmallIcon.png</string>
           <string>MyLargeIcon.png</string>
       </array>
   <key>LSItemContentTypes</key>
       <array>
           <string>com.example.myformat</string>
       </array>
   <key>LSHandlerRank</key>
   <string>Owner</string>
</dict>
```

CFBundleDocumentTypesキーの内容の詳細については、『*Information Property List Key Reference*』のこのキーの説明を参照してください。

サポート可能なファイルタイプのオープン

アプリケーションの起動時に、システムから、特定のファイルを開いてそれをユーザに表示するように指示される場合があります。通常、このような状況は、別のアプリケーションがそのファイルを検出して、それを処理するためにDocument Interaction Controllerを使用した場合に発生します。開くべきファイルの情報は、アプリケーションデリゲートの

application:didFinishLaunchingWithOptions:メソッドで受け取ります。アプリケーションがカスタムファイルタイプを扱う場合は、（applicationDidFinishLaunching:メソッドではなく）このデリゲートメソッドを実装して、それを使用してアプリケーションを初期化しなければなりません。

application:didFinishLaunchingWithOptions:メソッドに渡されたoptionsディクショナリには、開くべきファイルについての情報が含まれています。具体的に言うと、アプリケーションは、このディクショナリ内で次のキーを探さなければなりません。

- UIApplicationLaunchOptionsURLKeyには、開くべきファイルを指定するNSURLオブジェクトが含まれている。
- UIApplicationLaunchOptionsSourceApplicationKeyには、オープン要求を出したアプリケーションのバンドル識別子を持つNSStringが含まれている。
- UIApplicationLaunchOptionsAnnotationKeyには、ファイルを開くときにそのファイルに関連付けたいと要求元アプリケーションが希望していたプロパティリストオブジェクトが含まれている。

UIApplicationLaunchOptionsURLKeyキーが存在する場合、アプリケーションはそのキーで参照されるファイルを開いて、その内容を直ちに表示しなければなりません。ディクショナリ内のその他のキーを使用すると、ファイルを開くときの環境についての情報を収集することができます。

ビューとView Controller

iPhone OS 3.2では、UIKitフレームワークにiPad上でコンテンツを整理したり表示したりするのに役立つ新機能が含まれています。これらの機能は、新しいView Controllerクラスから、既存のインターフェイス機能の変更にまで及びます。これらの機能をアプリケーションに組み込むのが適切なケースの詳細については、『*iPad Human Interface Guidelines*』を参照してください。

複数の向きに対応した設計

ほとんど例外なく、アプリケーションはiPadデバイスではすべてのインターフェイスの向きをサポートするべきです。向きの変更をサポートする手順は、iPadデバイスでもiPhoneおよびiPod touchデバイスでも同じです。アプリケーションのウインドウとView Controllerは、回転をサポートするために必要な基本的なインフラストラクチャを提供します。既存のインフラストラクチャをそのまま使用することも、アプリケーションの個別の事情に合わせてその動作をカスタマイズすることもできます。

インターフェイスのすべての向きに対して基本的なサポートを実装するには、次の手順を実行しなければなりません。

- 各カスタムView Controllerに`shouldAutorotateToInterfaceOrientation:`メソッドを実装して、すべての向きに対して`YES`を返す。
- ビューがレイアウト変更に適切に対応できるように、ビューの`autoresizingMask`プロパティを設定する（このプロパティは、プログラムの中で、またはInterface Builderを使用して設定できます）。

基本的なサポート以上のことを行うには、ニーズに応じて実行できる次のような追加タスクがあります。

- サブビューの配置をより精密に管理する必要があるカスタムビューの場合は、`layoutSubviews`メソッドをオーバーライドして、そこにカスタムレイアウトコードを置く。
- ビューが実際に回転する直前、回転している最中、あるいは回転した直後にタスクを実行するには、`UIViewController`クラスの1ステップ回転の通知を使用する。

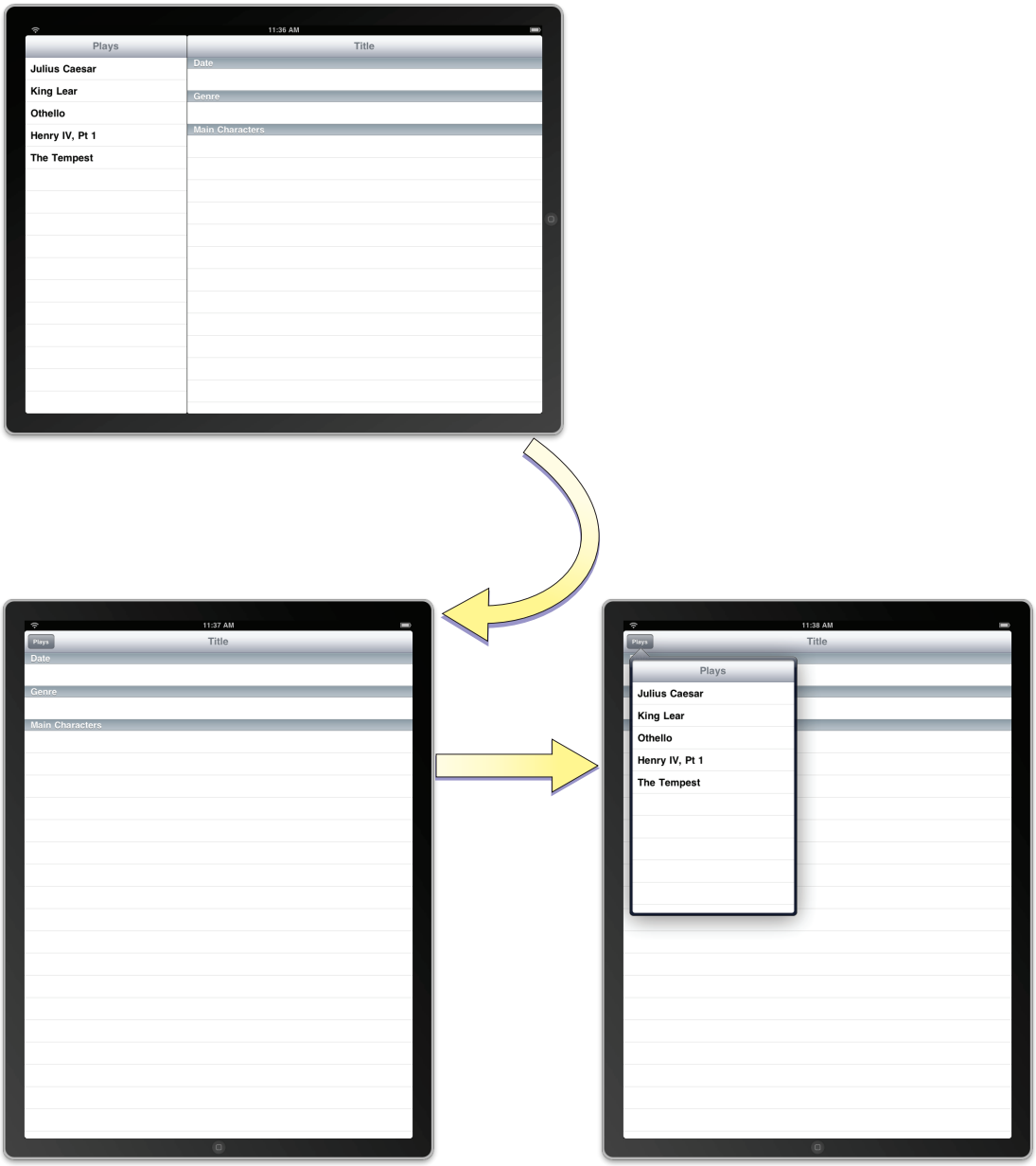
向きの変更が発生すると、ウインドウは最前面のView Controllerと連携して、新しい向きに合うようにコンテンツを調整します。この処理の間に、View Controllerは、追加のタスクを実行する機会となるいくつかの通知を受信します。具体的には、View Controllerの`willRotateToInterfaceOrientation:duration:`、`willAnimateRotationToInterfaceOrientation:duration:`、および`didRotateFromInterfaceOrientation:`の各メソッドは、ビューの回転の前後にタスクを実行する機会として最適なタイミングで呼び出されます。これらのメソッドを使用して、向きの変更に関連する任意のタスクを実行できます。たとえば、これらのメソッドを使用して、ビューの追加や削除、表示中の任意のテーブル内のデータの再ロードを実行できます。また、回転処理の間にコードのパフォーマンスを微調整することもできます。

View Controllerでの向きの変化への対応の詳細については、『*View Controller Programming Guide for iPhone OS*』を参照してください。

Split Viewインターフェイスの作成

Split Viewは、区切り要素で分割された左右に並んだ2つのペインで構成されます。Split View Controllerの第1のペインは、幅が320ポイントに固定されています。また、高さは、表示中のウインドウの高さと同じです。第2のペインは、残りのスペースを埋めます。iPhone OSでは、Split Viewは、マスタと詳細が対になったインターフェイスや、2つの異なるタイプの情報を左右に並べて表示したい場合に使用できます。デバイスが横長の向きのときには、Split Viewには両方のペインが表示されます。一方、縦長の向きのとき、Split Viewは第2のペインのみを表示し、利用可能なスペース全体に広がります。ユーザが第1のペインにアクセスできるようにするには、デベロッパが自分でそのペインを表示しなければなりません。縦長モードで第1のペインを表示するための最も一般的な方法は、第2のペインのツールバーにボタンを1つ追加することです。このボタンを使用して、図 4-1に示すように、第1のペインの内容を含むPopoverを表示します。

図 4-1 Split Viewインターフェイス



`UISplitViewController`クラスは、左右に並ぶペインの表示を管理します。これらのペインは、デベロッパが用意する1つのView Controllerによってそれぞれ管理されます。Split View Controllerは、2つのペインの間の調整を必要とする回転やその他のシステム関連動作を処理します。Split View Controllerのビューは、必ずアプリケーションウインドウのルートビューとしてインストールしなければなりません。Split Viewをナビゲーションインターフェイスやタブバーインターフェイスの内部に表示してはいけません。

Split View Controllerをアプリケーションに組み込む最も簡単な方法は、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ための良い出発点となります。Split Viewインターフェイスを実装するために必要なすべてが、すでに用意されています。デベロッパがしなければならないのは、カスタム

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、詳細に関連するコンテンツを表示するために利用できる画面領域が広くなっているという点です。一方、Split View Controllerを既存のインターフェイスに組み込むこともできます。これについては、「Interface BuilderでのSplit View Controllerの追加」（42 ページ）で説明します。

アプリケーションでのView Controllerの設定の詳細については、『*View Controller Programming Guide for iPhone OS*』を参照してください。

Interface BuilderでのSplit View Controllerの追加

Split View-based Applicationテンプレートのプロジェクトから作業を開始したくない場合でも、Split View Controllerをユーザインターフェイスに追加することはできます。Interface Builderのライブラリには、既存のnibファイルに追加可能なSplit View Controllerが含まれています。Split View Controllerを追加する場合は、通常、それをアプリケーションのメインnibファイルに追加します。それは、一般にSplit Viewがアプリケーションのウインドウの最上位レベルのビューとして挿入されるため、起動時にロードする必要があるからです。

Split View Controllerをアプリケーションのメインnibファイルに追加するには、次の手順を実行します。

1. アプリケーションのメインnibファイルを開く。
2. Split View Controllerオブジェクトをそのnibファイルのウインドウにドラッグする。

   このSplit View Controllerオブジェクトには、2つのペイン用の汎用のView Controllerが含まれています。
3. このSplit View Controller用のアウトレットをアプリケーションのデリゲートオブジェクトに追加し、そのアウトレットとSplit View Controllerオブジェクトを接続する。
4. アプリケーションデリゲートの`application:didFinishLaunchingWithOptions:`メソッドで、次のようにして、このSplit View Controllerのビューをウインドウのメインビューとしてインストールする。

   ```
   [window addSubview:mySplitViewController.view];
   ```
5. Split View Controllerに含まれる各View Contollerに対して、次の手順を実行する。
   - 「Identity」インスペクタを使用して、View Controllerのクラス名を設定します。
   - 「Attributes」インスペクタで、そのView Controllerのビューを含むnibファイルの名前を設定します。

Split Viewに含まれている2つのView Controllerのコンテンツについては、デベロッパに任されています。これらのView Controllerの設定は、アプリケーション内のその他のView Controllerを設定する場合と同様です。アプリケーションのメインnibファイルで行わなければならないのは、クラス名とnib名の設定だけです。それ以外の設定は、View Controllerのタイプによって異なります。たとえば、Navigation ControllerやTab Bar Controllerの場合は、追加のView Controller情報を指定する必要があるかもしれません。Navigation Controller、Tab Bar Controller、およびカスタムView Controllerを設定する手順については、『*View Controller Programming Guide for iPhone OS*』を参照してください。

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、その2つのプロパティにView Controller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 Controller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View Controllerを、自分のアプリケーションのコンテンツを表示するカスタムView Controller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]
                     initWithNibName:@"FirstNib" bundle:nil] autorelease];
   MySecondViewController* secondVC = [[[MySecondViewController alloc]
                     initWithNibName:@"SecondNib" bundle:nil] autorelease];

   UISplitViewController* splitVC = [[UISplitViewController alloc] init];
   splitVC.viewControllers = [NSArray arrayWithObjects:firstVC, secondVC, nil];

   [window addSubview:splitVC.view];
   [window makeKeyAndVisible];

   return YES;
}
```

Split Viewでの向きの変更のサポート

Split View Controllerでは、インターフェイスの向きの変更をすべきかどうかの判断は、2つのView Controllerによって決まります。これらのView Controllerの一方または両方が新しい向きをサポートしていない場合、向きは変更されません。これは、第1のView Controllerが表示されない縦長モードにも当てはまります。したがって、両方のView Controllerの`shouldAutorotateToInterfaceOrientation:`メソッドをオーバーライドして、サポート可能なすべての向きに対して`YES`を返すようにしなければなりません。

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。具体的に言うと、Split View Controllerは、縦長の向きに回転する場合は`viewControllers`配列内の第1のView Controllerを自動的に非表示にし、横長の向きに回転する場合はこのView Controllerを表示します。

縦長の向きのときにPopoverを使用して第1のView Controllerを表示したい場合は、デリゲートオブジェクトを使用します。このView Controllerが非表示または表示になると、Split View Controllerはそれをデリゲートに通知します。このView Controllerが非表示のときは、このView Controllerを表示するために使用するボタンとPopover Controllerがデリゲートに提供されます。デリゲートメソッドで実行しなければならないのは、このView Controllerへのアクセス手段を提供するために、指定されたボタンを表示中のツールバーに追加することだけです。同様に、このView Controllerが再び表示されたときには、このボタンを削除する機会がデリゲートに与えられます。デリゲートメソッドとその使用法の詳細については、『*UISplitViewControllerDelegate Protocol Reference*』を参照してください。

Popoverを使用したコンテンツの表示

Popoverは、現在のビューの前面に一時的に情報を重ねて表示するために使用する特殊なインターフェイス要素です。Popoverは、ユーザのアクションを必要とせずに、情報を表示したり収集したりするための手軽な手段を提供します。たとえば、Popoverは次のような場合に最適です。

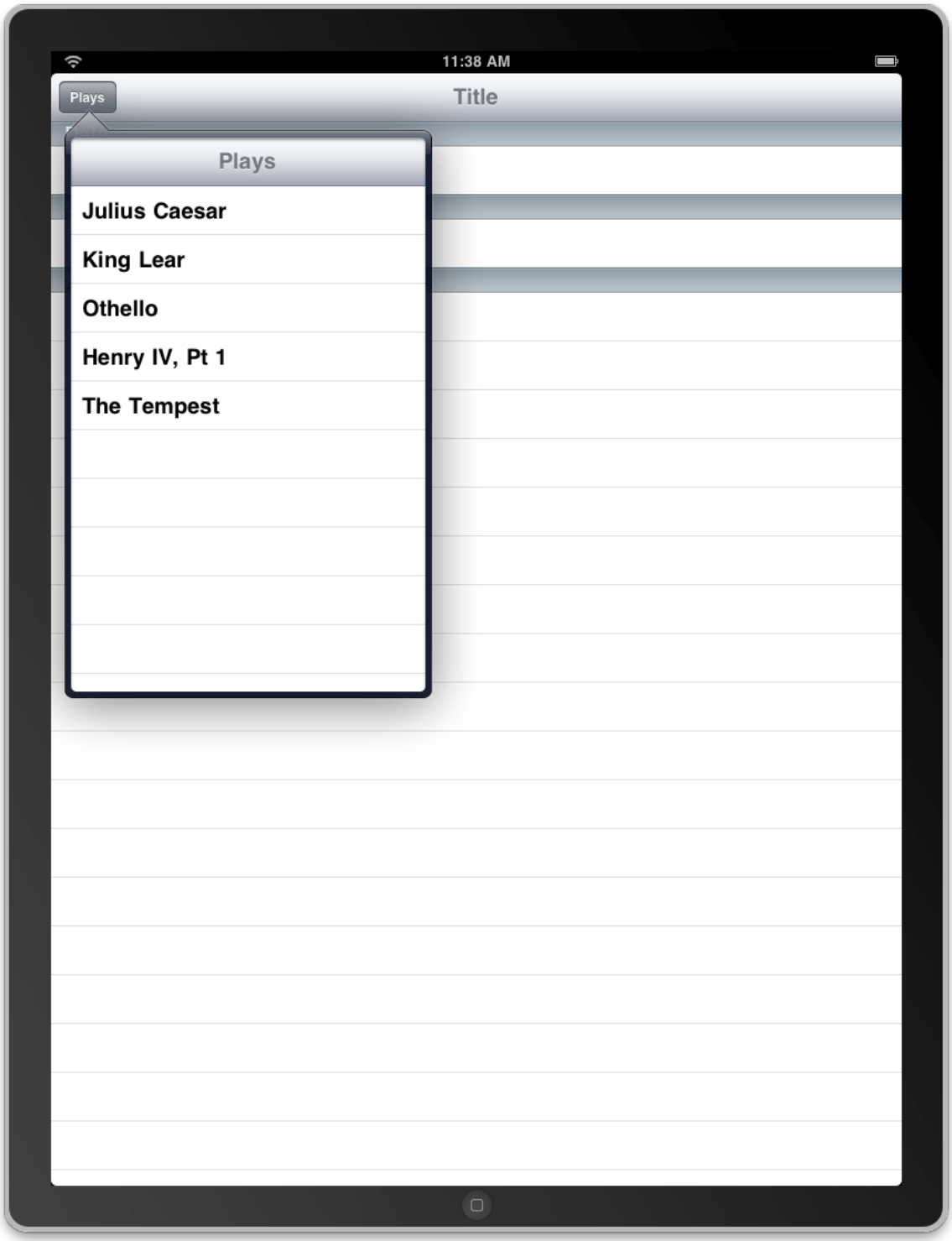
- デバイスが縦長の向きのときに、Split Viewインターフェイスの一部を表示するため
- いずれかのビュー内のオブジェクトに対して実行するアクションのリストを表示するため
- 画面上のオブジェクトについての情報を表示するため
- 頻繁にアクセスするツールや設定オプションを管理するため

iPhoneアプリケーションでは、モーダルビューを使用して上記のアクションのいくつかを実装できます。iPadデバイスでは、Popoverとモーダルビューはまったく別の目的を持っています。iPadアプリケーションでは、現在のワークフローに割り込みをして必要な情報をユーザから収集するために、モーダルビューを使用します。このような割り込みでは、ユーザが明確にアクションを許可またはキャンセルしなければならないため、中断が生じます。それに対し、Popoverによる割り込みは妨げになる度合いがはるかに少なく、ユーザが許可やキャンセルの意思表示をする必要がありません。Popoverは、ユーザのコンテンツの前面に表示されます。また、Popover領域の外をタップすることによって簡単に消去できます。したがって、Popoverから項目を選択する操作は任意です。アプリケーションの状態に影響するのは、ユーザが実際にPopoverのコンテンツとやり取りをする場合だけです。

Popoverは、変更の対象となるコンテンツの横に表示され、通常は、そのコンテンツを指す矢印を含んでいます。Popover自体のサイズは設定可能で、View Controllerのサイズに基づいています。ただし、必要であればサイズを変更できます。また、Popoverをきちんと画面に収めるために、Popover自身が、表示中のコンテンツのサイズを変更する場合もあります。

図 4-2に、Split Viewインターフェイスのマスタペインを表示するために使われるPopoverの例を示します。縦長の向きでは、詳細ペインのツールバーにカスタムボタンが追加されます。このボタンをタップすると、アプリケーションはPopoverを表示します。

図 4-2 Popoverを使用したマスタペインの表示



Popoverを使用するケースの詳細については、『*iPad Human Interface Guidelines*』を参照してください。

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から取得されます。Popoverは、ほとんどのタイプのView Controllerを表示できます。それには、カスタムView Controller、Table View Controller、Navigation Controller、そしてTab Bar Controllerさえも含まれます。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、それをカスタムView Controllerで初期化する。
2. （省略可能）`popoverContentSize`プロパティを使用して、Popoverのサイズをカスタマイズする。
3. （省略可能）Popoverにデリゲートを割り当てる。この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」（47 ページ）を参照してください。
4. Popoverを表示する。

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をPopoverに関連付けます。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。Popoverをいずれかのビューのコンテンツに関連付ける場合は、代わりに`presentPopoverFromRect:inView:permittedArrowDirections:animated:`メソッドを使用します。

Popoverは、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。あるいは、Popover Controller自身の`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コンテンツ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。

リスト 4-2に、ツールバーボタン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、View Controller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。もちろん、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ことができるように、スクロールビューを使用しなければなりません。

リスト 4-2　Popoverの表示

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

```
    // 後から使用するために、Popoverをカスタムプロパティに格納する
    self.popoverController = aPopover;
    [aPopover release];

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

プログラムでPopoverを消去するには、そのPopover Controllerの`dismissPopoverAnimated:`メソッドを呼び出します。Popoverの消去が必要になるのは、Popoverのコンテンツ領域内のタップによって、Popoverが消去されるようにしたい場合だけです。Popoverの外部をタップした場合は、自動的にPopoverは消去されます。一般には、Popoverのコンテンツ領域内のタップに応答してPopoverを消去することをお勧めします。コンテンツ領域内のタップによって、コンテンツの選択やコンテンツへの変更が行われる場合は、特にお勧めします。ただし、このような動作がアプリケーションにとって適切かどうかの判断はデベロッパに任されています。もっとも、Popoverを消去できるように、Popover Controllerへの参照を格納するのもデベロッパの責任であることを忘れないでください。システムは、デフォルトではこの参照を提供しません。

Popover用のデリゲートの実装

Popoverビューの外部をユーザがタップしたことによってPopoverが消去されたときは、Popoverは自動的にそれをデリゲートに通知します。Popoverが消去される前に、Popover Controllerは`popoverControllerShouldDismissPopover:`メッセージをデリゲートに送信します。デリゲートのこのメソッドの実装が`YES`を返した場合、またはデリゲートがこのメソッドをまったく実装していない場合は、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。そのような場合は、このメソッドを実装して`NO`を返すことができます。ただし、より良いアプローチは、アプリケーションがそのような状況に陥るのを避けることです。

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。再び使用する予定がなければ、この時点でPopover Controllerを解放するのが安全です。また、このメッセージを使用して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情報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である。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View Controllerに格納することです。これによって、コンテンツView Controller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。
- 頻繁に使用する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く。同様に、Popoverごとに新しいPopover Controllerを作成するのではなく、Popover Controllerを自由に再利用します。Popover Controllerはかなり柔軟性のあるオブジェクトなので、再利用も簡単です。また、これらはアプリケーションがメモリ不足の警告を受け取った場合に気楽に解放できるオブジェクトです。

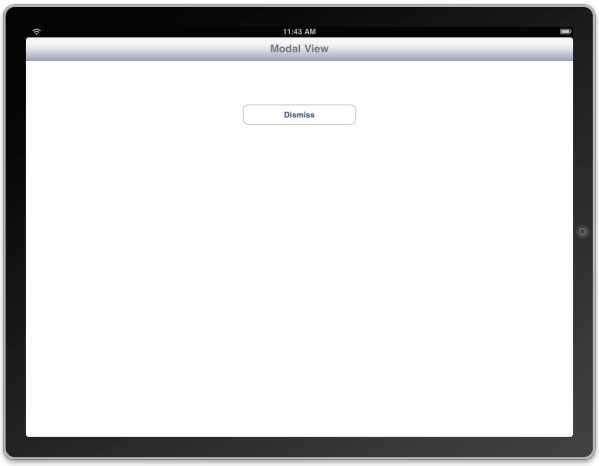
- Popoverを表示するときは、できる限り、許容する矢印の向きに`UIPopoverArrowDirectionAny`定数を指定する。この定数を指定することで、UIKitがPopoverの位置とサイズを決定する際の柔軟性が最大になります。許容する矢印の向きを制限した場合は、Popover Controllerが、Popoverを表示する前に、そのサイズを縮小しなければならない場合もあります。

モーダルビューの表示スタイルの設定

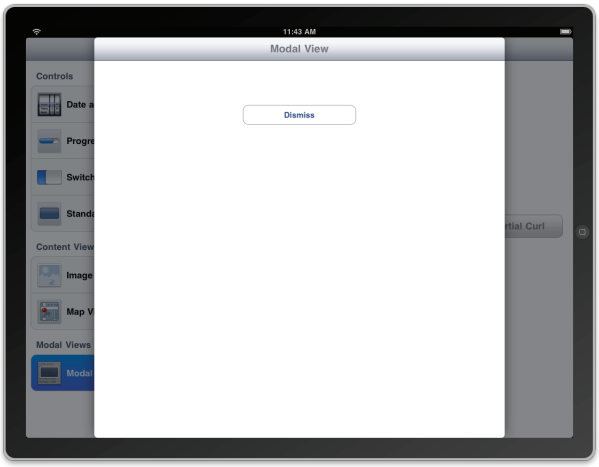
iPhoneOS 3.2では、View Controllerをモーダルモードで表示するための新たなオプションが追加されました。以前は、モーダルモードで表示されたビューは必ず背後にあるウインドウの表示部分を覆い隠していました。これからは、`UIViewController`クラスに、モーダルモードで表示するときのView Controllerの外観を決定する`modalPresentationStyle`プロパティが追加されています。このプロパティにさまざまなオプションを指定することによって、以前のようにView Controllerを画面全体に表示したり、画面の一部に表示することができます。

図4-3に、利用可能な主要な表示スタイルを示します（`UIModalPresentationCurrentContext`スタイルを利用すると、View Controllerは親の表示スタイルを採用します）。各モーダルビューでは、薄暗くなっている領域に背後のコンテンツが表示されますが、そのコンテンツ内をタップすることはできません。したがって、Popoverとは違って、モーダルビューには、従来どおり、ユーザがモーダルビューを消去するためのコントロールが含まれていなければなりません。

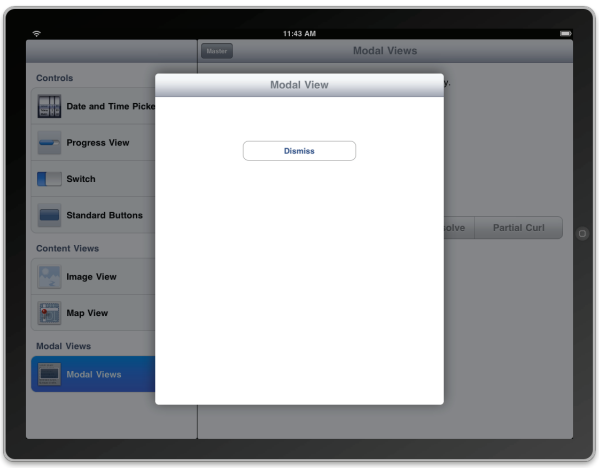
図 4-3 モーダルモードの表示スタイル



`UIModalPresentationFullScreen`



`UIModalPresentationPageSheet`



`UIModalPresentationFormSheet`

各種の表示スタイルをどの局面で使用するかのガイダンスについては、『*iPad Human Interface Guidelines*』を参照してください。

ツールバーの有効利用

ツールバーはiPhone OS 2.0からサポートされていますが、iPadアプリケーションでは、ツールバーが以前よりも重要な役割を果たします。iPhone OS 3.2より前は、ユーザインターフェイスのガイドラインでは、ツールバーをアプリケーションのウインドウの下端に配置することを推奨していました。ウインドウの上端は、ビュー間の一般的なナビゲーションを提供するNavigation Bar用に確保されていました。iPadデバイスでは、利用可能なスペースが広がったため、Navigation Barの代わりに、ツールバーをアプリケーションのウインドウの上端に配置できるようになりました。このように配置することで、アプリケーションでのツールバーのコマンドが目立つようになります。

アプリケーションでのツールバーの設定と使用に関するガイドラインについては、『*iPad Human Interface Guidelines*』を参照してください。

Gesture Recognizer

iPhone OSアプリケーションは、主として、アプリケーションのユーザインターフェイス内のボタン、ツールバー、テーブルビューの行、その他のオブジェクトをユーザがタップしたときに生成されるイベントによって駆動されます。UIKitフレームワークのクラスは、これらのほとんどのオブジェクトに対するデフォルトのイベント処理動作を提供します。ただし、独自のイベント処理を実装しなければならないアプリケーションもあります（主に、カスタムビューを持つアプリケーション）。このようなアプリケーションでは、マルチタッチのシーケンスにおけるタッチオブジェクトの流れを解析して、ユーザの意図を判断しなければなりません。

ほとんどのイベント処理ビューは、ユーザが画面上で行う一般的なジェスチャ（トリプルタップ、タッチアンドホールド（「長押し」とも呼ばれる）、ピンチ、回転ジェスチャなど）を検出しようとします。マルチタッチイベントの流れをそのまま解析して、1つ以上のジェスチャを検出するコードは、通常は複雑です。iPhone OS 3.2より前のバージョンでは、このようなコードを別のプロジェクトにコピーして適宜修正する方法以外に、コードを再利用できません。

iPhone OS 3.2では、アプリケーションでのジェスチャ検出を支援するために、Gesture Recognizerが導入されました。このオブジェクトは、`UIGestureRecognizer`クラスを直接継承します。以降の各セクションでは、このオブジェクトの仕組み、使用法、およびアプリケーション間で再利用可能なカスタムGesture Recognizerの作成方法について説明します。

注： iPhone OSのマルチタッチイベントの概要については、『*iPhone Application Programming Guide*』の「Event Handling」を参照してください。

イベント処理を簡単にするGesture Recognizer

`UIGestureRecognizer`は、Gesture Recognizer具象サブクラス（単に、Gesture Recognizerと呼ぶ）のための抽象基底クラスです。`UIGestureRecognizer`クラスには、プログラムインターフェイスが定義されており、ジェスチャ認識動作の土台が実装されています。UIKitフレームワークでは、最も良く使われるジェスチャに対応する6つのGesture Recognizerを提供しています。それ以外のジェスチャについては、独自のGesture Recognizerを設計して実装できます（詳細については、「カスタムGesture Recognizerの作成」（61 ページ）を参照）。

認識可能なジェスチャ

UIKitフレームワークでは、表 5-1に示すジェスチャの認識をサポートしています。ここに挙げた各クラスは、`UIGestureRecognizer`を直接継承したサブクラスです。

表 5-1 UIKitフレームワークのGesture Recognizerクラスで認識可能なジェスチャ

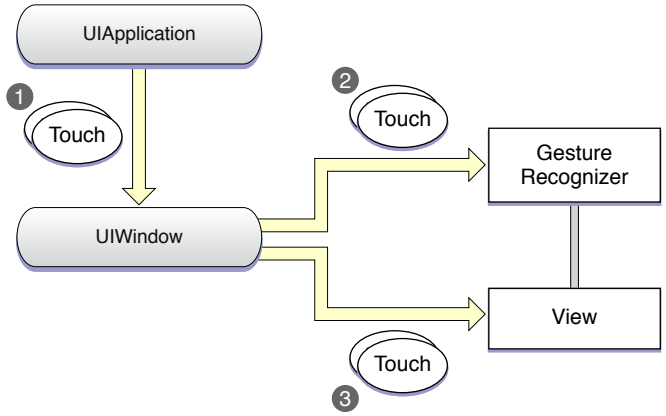
ジェスチャ	UIKitのクラス
タップ（タップ数は任意）	`UITapGestureRecognizer`
ピンチインとピンチアウト（ビューのズーム拡大縮小用）	`UIPinchGestureRecognizer`
パニング（ドラッグ）	`UIPanGestureRecognizer`
スワイプ（任意の方向に対応）	`UISwipeGestureRecognizer`
回転（互いに反対の方向への指の移動）	`UIRotationGestureRecognizer`
長押し（「タッチアンドホールド」とも呼ばれる）	`UILongPressGestureRecognizer`

Gesture Recognizerの使用を決定する前に、その使いかたを検討してください。ユーザの想定に合った方法でジェスチャに応答するようにします。たとえば、ピンチジェスチャは、ビュー内容の拡大縮小に対応させるべきであって、たとえば選択要求として解釈するべきではありません。選択要求にはタップの方が適切です。適切なジェスチャの使用についてのガイドラインは、『*iPad Human Interface Guidelines*』を参照してください。

Gesture Recognizerをビューに添付する

ジェスチャを検出するには、ユーザがタッチするビューにGesture Recognizerをアタッチしなければなりません。このビューをヒットテストビューと呼びます。iPhone OSのイベントは`UIEvent`オブジェクトで表現されることを思い出してください。また、各イベントオブジェクトは、現在のマルチタッチシーケンスを表す`UITouch`オブジェクトをカプセル化しています。これらの`UITouch`オブジェクトの集合は、特定のフェーズのマルチタッチシーケンスに固有のものです。イベントの配信は、最初は、通常の経路（オペレーティングシステムからアプリケーションオブジェクトを経て、タッチが発生しているウインドウを表すウインドウオブジェクトへ）をたどります。しかし、ヒットテストビューへのイベントの送信の前に、そのビューまたはそのビューのいずれかのサブビューにアタッチされているGesture Recognizerにイベントが送信されます。図 5-1に、この一般的な経路を示します。図中の番号は、タッチが受信される順番を表します。

図 5-1 Gesture Recognizerがビューにアタッチされている場合のタッチオブジェクトの経路



したがって、Gesture Recognizerは、対応するビューまたはビュー階層に送信されるタッチオブジェクトを監視する役割を果たします。ただし、Gesture Recognizerはそのビュー階層には含まれないため、レスポンダチェーンには加わりません。Gesture Recognizerは、ジェスチャの認識処理を行っている間、ビューへのタッチオブジェクトの配信を遅らせる場合があります。また、デフォルトでは、ジェスチャを認識すると、ビューへの残りのタッチオブジェクトの配信を取り消します。Gesture Recognizerからビューへのイベント配信で考えられるシナリオの詳細については、「ビューへのタッチ配信の制御」（59 ページ）を参照してください。

ジェスチャによっては、`UIGestureRecognizer`の`locationInView:`メソッドや`locationOfTouch:inView:`メソッドを利用して、クライアントが、対応するビューやサブビューでのジェスチャや特定のタッチの位置を検出できるものもあります。詳細については、「ジェスチャへの応答」（56 ページ）を参照してください。

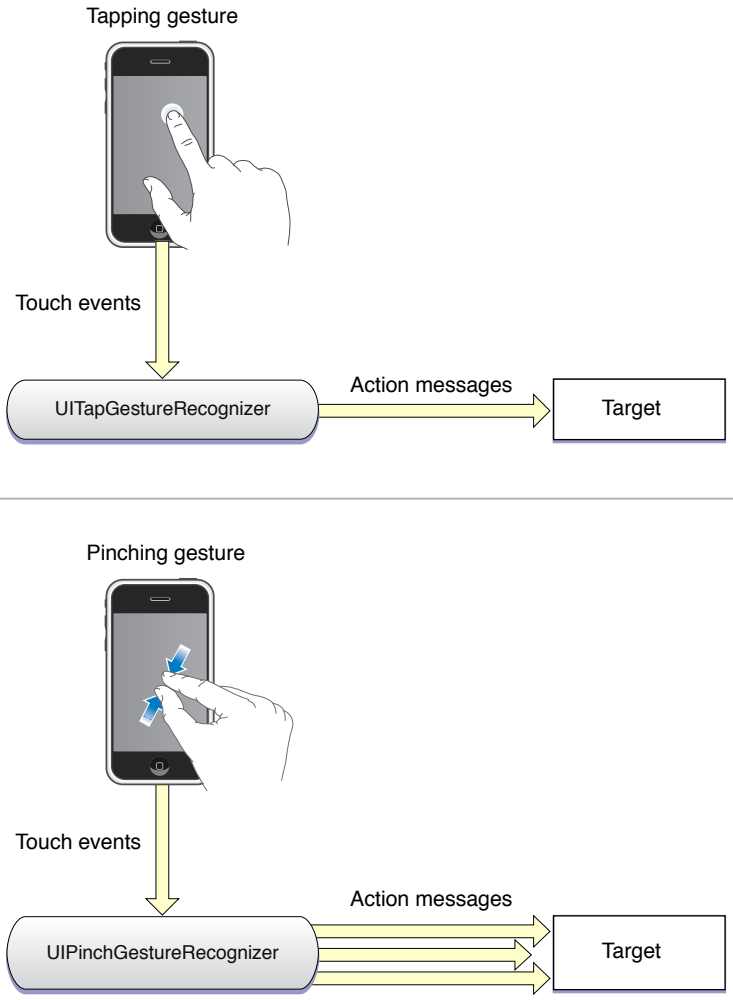
ジェスチャによってアクションメッセージが発行される

Gesture Recognizerは、対応するジェスチャを認識すると、1つ以上のターゲットに1つ以上のアクションメッセージを送信します。Gesture Recognizerを作成するときは、アクションとターゲットで初期化します。以後、ターゲット／アクションのペアを追加していけます。ターゲット／アクションのペアは加算的ではありません。つまり、アクションは、あらかじめ関連付けられていたターゲットにのみ送信され、それ以外のターゲットには送信されません（別のターゲット／アクションのペアで指定されていない限り）。

単発のジェスチャと連続的なジェスチャ

Gesture Recognizerは1つのジェスチャを認識すると、ターゲットに1つだけアクションメッセージを送信するか、そのジェスチャが終了するまで複数のアクションメッセージを送信するかのいずれかの動作をします。この動作は、ジェスチャが単発か連続的かによって決まります。単発のジェスチャ（ダブルタップなど）は1回だけ発生します。Gesture Recognizerは、単発のジェスチャを認識すると、ターゲットに1つだけアクションメッセージを送信します。連続的なジェスチャ（ピンチなど）は、一定時間にわたって発生し、ユーザがマルチタッチシーケンスでの最後の指を離したときに終了します。Gesture Recognizerは、マルチタッチシーケンスが終了するまで、短い間隔で複数のアクションメッセージをターゲットに送信します。

図 5-2 単発のジェスチャと連続的なジェスチャ



各Gesture Recognizerクラスのリファレンスドキュメントには、そのクラスのインスタンスが単発のジェスチャを検出するか、連続的なジェスチャを検出するかが明記されています。

ジェスチャ認識の実装

ジェスチャ認識を実装するには、Gesture Recognizerのインスタンスを作成して、ターゲットとアクションを割り当てます。また、場合によっては、ジェスチャ固有の属性も割り当てます。このGesture Recognizerオブジェクトをビューにアタッチしたら、ジェスチャを処理するターゲットオブジェクトにアクションメソッドを実装します。

Gesture Recognizerの準備

Gesture Recognizerを作成するには、UIGestureRecognizer具象サブクラスのインスタンスを割り当てて初期化しなければなりません。初期化するときには、次のコードのように、ターゲットオブジェクトとアクションセレクタを指定します。

```
UITapGestureRecognizer *doubleFingerDTap = [[UITapGestureRecognizer alloc]
   initWithTarget:self action:@selector(handleDoubleDoubleTap:)];
```

ジェスチャを処理するアクションメソッド（および、それを識別するセレクタ）は、次の2つのシグニチャのいずれかに従うことが前提になります。

```
- (void)handleGesture
- (void)handleGesture:(UIGestureRecognizer *)sender
```

*handleGesture*と*sender*には、任意の名前を使用できます。2番目のシグネチャを有するメソッドを利用すると、ターゲットがGesture Recognizerに補足情報を照会できるようになります。たとえば、UIPinchGestureRecognizerオブジェクトのターゲットは、ピンチジェスチャに関連する現在の倍率をそのGesture Recognizerオブジェクトに問い合わせることができます。

Gesture Recognizerを作成したら、UIViewのaddGestureRecognizer:メソッドを使用して、タッチを受け取るビュー（つまり、ヒットテストビュー）にそれをアタッチしなければなりません。現在ビューにアタッチされているGesture Recognizerは、gestureRecognizersプロパティを通じて知ることができます。また、removeGestureRecognizer:を呼び出して、Gesture Recognizerをビューからデタッチすることもできます。

リスト 5-1に示すメソッド例では、1本指によるダブルタップ、パニングジェスチャ、回転ジェスチャに対応する3つのGesture Recognizerの作成と初期化を行います。その後、それぞれのGesture Recognizerオブジェクトを同じビューにアタッチします。singleFingerDTapオブジェクトでは、そのジェスチャを認識するために2つのタップが必要であることを、コードで指定しています。それぞれのメソッドでは、作成したGesture Recognizerをビューに追加した後は、それを解放します（ビューがGesture Recognizerを保持しているため）。

リスト 5-1　単発ジェスチャおよび連続ジェスチャに対応するGesture Recognizerの作成と初期化

```
- (void)createGestureRecognizers {
    UITapGestureRecognizer *singleFingerDTap = [[UITapGestureRecognizer alloc]
        initWithTarget:self action:@selector(handleSingleDoubleTap:)];
    singleFingerDTap.numberOfTapsRequired = 2;
    [self.theView addGestureRecognizer:singleFingerDTap];
    [singleFingerDTap release];

    UIPanGestureRecognizer *panGesture = [[UIPanGestureRecognizer alloc]
        initWithTarget:self action:@selector(handlePanGesture:)];
    [self.theView addGestureRecognizer:panGesture];
    [panGesture release];

    UIPinchGestureRecognizer *pinchGesture = [[UIPinchGestureRecognizer alloc]
        initWithTarget:self action:@selector(handlePinchGesture:)];
    [self.theView addGestureRecognizer:pinchGesture];
    [pinchGesture release];
}
```

UIGestureRecognizerのaddTarget:action:メソッドを使用して、1つのGesture Recognizerに複数のターゲットとアクションを追加することもできます。各ターゲット／アクションペアに対応するアクションメッセージは、そのペアに限定されることを忘れないでください。複数のターゲット／アクションを持つ場合、それらは加算的ではありません。

ジェスチャへの応答

ジェスチャを処理するには、Gesture Recognizerのターゲットが、そのGesture Recognizerの初期化時に指定されたアクションセレクタに対応するメソッドを実装していなければなりません。タップジェスチャのような単発のジェスチャの場合、Gesture Recognizerは、ジェスチャを認識するたびにこのメソッドを呼び出します。一方、連続的なジェスチャの場合、Gesture Recognizerは、ジェスチャが終了する（つまり、Gesture Recognizerのビューから最後の指が離れる）まで一定間隔で繰り返しこのメソッドを呼び出します。

ジェスチャ処理メソッドでは、しばしばターゲットオブジェクトがGesture Recognizerからジェスチャについての補足情報を取得します。それには、Gesture Recognizerで定義されているプロパティ（scale（拡大縮小の倍率）、velocityなど）の値を取得します。また、（必要な場合は）Gesture Recognizerにジェスチャの位置を問い合わせることもできます。

リスト 5-2に、回転ジェスチャ(handleRotate:)とパニングジェスチャ(handlePanGesture:)の2つの連続的なジェスチャ用の処理を示します。また、単発のジェスチャ用の処理の例も示します。この例では、ユーザが1本の指でビューをダブルタップしたときに、ハンドラ(handleSingleDoubleTap:)はそのビューをダブルタップされた位置にセンタリングします。

リスト 5-2　ピンチ、パニング、ダブルタップの各ジェスチャの処理

```
- (IBAction)handlePinchGesture:(UIGestureRecognizer *)sender {
    CGFloat factor = [(UIPinchGestureRecognizer *)sender scale];
    self.view.transform = CGAffineTransformMakeScale(factor, factor);
}

- (IBAction)handlePanGesture:(UIPanGestureRecognizer *)sender {
    CGPoint translate = sender.translation;

    CGRect newFrame = currentImageFrame;
    newFrame.origin.x += translate.x;
    newFrame.origin.y += translate.y;
    sender.view.frame = newFrame;

    if (sender.state == UIGestureRecognizerStateEnded)
        currentImageFrame = newFrame;
}

- (IBAction)handleSingleDoubleTap:(UIGestureRecognizer *)sender {
    CGPoint tapPoint = [sender locationInView:sender.view.superview];
    [UIView beginAnimations:nil context:NULL];
    sender.view.center = tapPoint;
    [UIView commitAnimations];
}
```

これらのアクションメソッドは、次のような特有の方法でジェスチャを処理します。

- handlePinchGesture:メソッドでは、ターゲットがGesture Recognizer (sender)とやり取りをして拡大縮小の倍率(scale)を取得します。このメソッドは、Core Graphicsの関数にこの倍率値を使用します。この関数は、ビューを拡大縮小して、その計算結果をそのビューのアフィンtransformプロパティに割り当てます。
- handlePanGesture:メソッドは、Gesture Recognizerから取得したtranslationInView:値を対応するビューのキャッシュ済みのフレーム値に適用します。ジェスチャが終了したときには、最新のフレーム値がキャッシュされます。
- handleSingleDoubleTap:メソッドでは、ターゲットがlocationInView:メソッドを呼び出して、Gesture Recognizerからダブルタップジェスチャの位置を取得します。次に、この位置（スーパービューの座標に変換されている）を使用して、ビューの中心をダブルタップの位置にアニメーションで移動します。

handlePinchGesture:メソッドで取得した拡大縮小倍率は、回転角度やその他の連続的なジェスチャのGesture Recognizerに関連する変換値と同様に、ジェスチャが最初に認識された時点のビューの状態に適用されます。これは、特定のジェスチャのハンドラが呼び出されるたびに連結される差分値ではありません。

Gesture Recognizerがアタッチされているヒットテストビューは、タッチイベントが入ってきたときに、受け身である必要はありません。代わりに、gestureRecognizersプロパティを調べて、特定のUITouchオブジェクトに関与するGesture Recognizerを決定できます（対応するGesture Recognizerがあれば）。同様に、UIEventのtouchesForGestureRecognizer:メソッドを呼び出して、特定のGesture Recognizerが特定のイベントに対して解析中のタッチを知ることもできます。

ほかのGesture Recognizerとのやり取り

ビューには複数のGesture Recognizerがアタッチされている場合もあります。デフォルトの動作では、マルチタッチシーケンス内のタッチイベントは、最終的にビューに配信されるまでに、Gesture RecognizerからGesture Recognizerへ非決定的な順番で渡されます。通常は、このデフォルトの動作で十分です。しかし、次のような動作が必要になる場合もあります。

- 最初のGesture Recognizerがタッチイベントの解析に失敗すると、次のGesture Recognizerが解析を開始する。
- ほかのGesture Recognizerが、特定のマルチタッチシーケンスやそのシーケンス内のタッチオブジェクトを解析するのを禁止する。
- 2つのGesture Recognizerが同時に動作できるようにする。

UIGestureRecognizerクラスでは、クライアントのメソッド、デリゲートのメソッド、およびサブクラスでオーバーライドしたメソッドを提供することで、これらの動作を可能にします。

Gesture Recognizerの失敗を条件とする

2つのGesture Recognizerの間に、一方が失敗した場合にだけ、もう一方が動作可能になるような関係を持たせたい場合があります。たとえば、Gesture Recognizer Aは、Gesture Recognizer Bが失敗するまでマルチタッチシーケンスの解析を開始しないようにする場合などです。逆に、Gesture Recognizer Bがジェスチャの認識に成功した場合は、Gesture Recognizer Aはそのマルチタッチシーケンスを解析しません。このような関係を指定する場合の例としては、シングルタップ用のGesture Recognizerと

ダブルタップ用のGesture Recognizerを1つずつ持つ場合が考えられます。シングルタップ用のGesture Recognizerは、ダブルタップ用のGesture Recognizerが失敗するまでは、マルチタッチシーケンスの解析を開始しません。

このような関係を指定するために呼び出すメソッドが、`requireGestureRecognizerToFail:`です。このメッセージを送信すると、それを受信したGesture Recognizerは、指定されたGesture Recognizerが`UIGestureRecognizerStateFailed`に遷移するまでは、`UIGestureRecognizerStatePossible`状態に留まっていなければなりません。指定されたGesture Recognizerが`UIGestureRecognizerStateRecognized`または`UIGestureRecognizerStateBegan`に遷移した場合は、処理を進めることができますが、ジェスチャを認識してもアクションメッセージは送信されません。

ジェスチャ認識の状態と、それらの状態間に起こり得る遷移の詳細については、「状態遷移」（61 ページ）を参照してください。

Gesture Recognizerのタッチ解析を禁止する

Gesture Recognizerが特定のタッチを解析するのを禁止したり、ジェスチャを認識するのを禁止できます。このような「禁止」関係は、デリゲーションのメソッド、または`UIGestureRecognizer`クラスで宣言されているメソッドのオーバーライドを使用して指定できます。

`UIGestureRecognizerDelegate`プロトコルには、ケースバイケースで特定のGesture Recognizerのジェスチャ認識を禁止する、次の2つのオプションメソッドが宣言されています。

- `gestureRecognizerShouldBegin:`—このメソッドは、Gesture Recognizerが`UIGestureRecognizerStatePossible`から出ようとするときに呼び出されます。`UIGestureRecognizerStateFailed`に遷移させるには、`NO`を返します（デフォルト値は`YES`です）。
- `gestureRecognizer:shouldReceiveTouch:`—このメソッドは、1つ以上の新規のタッチが発生したときに、ウインドウオブジェクトがGesture Recognizerの`touchesBegan:withEvent:`を呼び出す前に呼び出されます。Gesture Recognizerがこれらのタッチを表すオブジェクトを解析するのを禁止するには`NO`を返します（デフォルト値は`YES`です）。

さらに、これらのデリゲーションメソッドと同じ振る舞いをする2つの`UIGestureRecognizer`メソッド（`UIGestureRecognizerSubclass.h`で宣言されている）があります。サブクラスでこれらのメソッドをオーバーライドすると、クラスレベルの禁止規則を定義できます。

```
- (BOOL)canPreventGestureRecognizer:(UIGestureRecognizer
*)preventedGestureRecognizer;
- (BOOL)canBePreventedByGestureRecognizer:(UIGestureRecognizer
*)preventingGestureRecognizer;
```

同時のジェスチャ認識を許可する

デフォルトでは、2つのGesture Recognizerが同時にジェスチャ認識を試みることはできません。しかし、`UIGestureRecognizerDelegate`プロトコルのオプションメソッドの`gestureRecognizer:shouldRecognizeSimultaneouslyWithGestureRecognizer:`を実装することによって、この動作を変更できます。このメソッドは、それを受信したGesture Recognizerの認識が、指定されたGesture Recognizerの動作をブロックする（または、その逆の）場合に呼び出されます。両方のGesture Recognizerに同時にジェスチャ認識を許可する場合は、`YES`を返します。

注： YESを返すと、同時認識ができることは保証されますが、NOを返したからといって、同時認識が禁止される保証はありません。ほかのジェスチャのデリゲートがYESを返す場合があるからです。

ビューへのタッチ配信の制御

一般に、ウインドウは、UITouchオブジェクト（UIEventオブジェクトにカプセル化されている）をGesture Recognizerに配信してから、それに対応するヒットテストビューに配信します。ただし、ジェスチャが認識されるかどうかによって、この一般的な経路が回り道をしたり、行き止まりになったりします。アプリケーションのニーズに合うように、この配信経路を変更できます。

デフォルトのタッチイベント配信

デフォルトでは、マルチタッチシーケンス内のウインドウは、ヒットテストビューへの「Ended」フェーズのタッチオブジェクトの配信を遅らせます。そして、ジェスチャが認識された場合は、ビューへの現在のタッチオブジェクトの配信を禁止して、それまでにビューが受信したタッチオブジェクトを取り消します。実際の動作は、タッチオブジェクトのフェーズと、Gesture Recognizerがマルチタッチシーケンス内でそのジェスチャを認識できたかどうかによって決まります。

この動作をわかりやすく説明するために、2つのタッチ（つまり、2本の指）を伴う単発のジェスチャ用の架空のGesture Recognizerを考えます。タッチオブジェクトがシステムに入ると、それは、UIApplicationオブジェクトから、ヒットテストビューに対応するUIWindowオブジェクトに渡されます。ジェスチャが認識されると、次のようなシーケンスが発生します。

1. ウインドウは、「Began」フェーズ(UITouchPhaseBegan)の2つのタッチオブジェクトをGesture Recognizerに送信します。しかし、Gesture Recognizerはこのジェスチャを認識しません。ウインドウは、これと同じタッチを、このGesture Recognizerに対応するビューに送信します。

2. ウインドウは、「Moved」フェーズ(UITouchPhaseMoved)の2つのタッチオブジェクトをGesture Recognizerに送信します。しかし、Gesture Recognizerは依然としてジェスチャを認識しません。次に、ウインドウは、これらのタッチをGesture Recognizerに対応するビューに送信します。

3. ウインドウは、「Ended」フェーズ(UITouchPhaseEnded)の*1*つのタッチオブジェクトをGesture Recognizerに送信します。このタッチオブジェクトはジェスチャに関する十分な情報をもたらしませんが、ウインドウはそのオブジェクトをGesture Recognizerに対応するビューに渡さずに保留します。

4. ウインドウは、「Ended」フェーズのもう1つのタッチオブジェクトをGesture Recognizerに送信します。ここでようやくGesture Recognizerはジェスチャを認識します。その結果、Gesture Recognizerの状態はUIGestureRecognizerStateRecognizedに設定されます。最初の（唯一の）アクションメッセージが送信される直前に、ビューはtouchesCancelled:withEvent:を受信して、それまで（「Began」フェーズと「Moved」フェーズで）送信されたタッチオブジェクトを無効にします。「Ended」フェーズのタッチが取り消されます。

今度は、最後のステップのGesture Recognizerが、それまで解析してきたマルチタッチシーケンスは、自身が担当するジェスチャではないと判断したとします。その場合は、Gesture Recognizerの状態はUIGestureRecognizerStateFailedに設定されます。そして、ウインドウは「Ended」フェーズの2つのタッチオブジェクトを、touchesEnded:withEvent:メッセージとして、Gesture Recognizerに対応するビューに送信します。

連続的なジェスチャ用のGesture Recognizerも同様のシーケンスをたどります。ただし、タッチオブジェクトが「Ended」フェーズに到達する前に、ジェスチャを認識する可能性が高くなります。ジェスチャを認識すると、Gesture Recognizerの状態は`UIGestureRecognizerStateBegan`に設定されます。ウインドウは、マルチタッチシーケンス内のそれ以降のすべてのタッチオブジェクトを、Gesture Recognizerに対応するビューではなく、Gesture Recognizerに送信します。

注： ジェスチャ認識の状態と、それらの状態間に起こり得る遷移の詳細については、「状態遷移」（61 ページ）を参照してください。

ビューへのタッチ配信の操作

`UIGestureRecognizer`の3つのプロパティの値を変更することによって、ビューへのタッチオブジェクトのデフォルトの配信経路をある程度変更できます。これらのプロパティとデフォルトは次のとおりです。

`cancelsTouchesInView`（デフォルト値は`YES`）
`delaysTouchesBegan`（デフォルト値は`NO`）
`delaysTouchesEnded`（デフォルト値は`YES`）

これらのプロパティのデフォルト値を変更すると、動作に次のような違いが生じます。

- `cancelsTouchesInView`を`NO`に設定する—認識されたジェスチャに含まれるタッチについて、`touchesCancelled:withEvent:`がビューに送信*されません*。その結果、対応するビューが「Began」フェーズや「Moved」フェーズでそれまでに受信したタッチオブジェクトは無効になりません。
- `delaysTouchesBegan`を`YES`に設定する—Gesture Recognizerがジェスチャを認識すると、そのジェスチャに含まれていたタッチオブジェクトが、対応するビューに配信されません。この設定は、`UIScrollView`の`delaysContentTouches`プロパティで提供されるものと同様の動作を提供します。スクロールビューの場合は、タッチが始まるとすぐにスクロールが始まるときは、そのスクロールビューオブジェクトのサブビューはタッチを受信しません。したがって、瞬時の視覚的なフィードバックはありません。ユーザインターフェイスの反応が鈍いように感じられやすくなるので、この設定には注意が必要です。
- `delaysTouchesEnded`を`NO`に設定する—タッチが終了した後にジェスチャを認識したGesture Recognizerは、ビューに対してそのタッチを取り消せなくします。たとえば、あるビューに、`numberOfTapsRequired`が2に設定された`UITapGestureRecognizer`オブジェクトがアタッチされている場合に、ユーザがそのビューをダブルタップしたとします。このプロパティが`NO`に設定されていると、このビューは、`touchesBegan:withEvent:`、`touchesEnded:withEvent:`、`touchesBegan:withEvent:`、`touchesCancelled:withEvent:`の順番でメッセージを受け取ります。このプロパティが`YES`に設定されていると、ビューは、`touchesBegan:withEvent:`、`touchesBegan:withEvent:`、`touchesCancelled:withEvent:`、`touchesCancelled:withEvent:`の順番でメッセージを受け取ります。このプロパティの目的は、Gesture Recognizerによって後から取り消される可能性があるタッチのために、ビューがアクションを実行しないようにすることです。

カスタムGesture Recognizerの作成

カスタムGesture Recognizerを作成する場合は、Gesture Recognizerの仕組みを十分に理解しておく必要があります。次のセクションでは、Gesture Recognizerのアーキテクチャ面の背景について説明します。また、その次のセクションでは、実際にGesture Recognizerを作成する際の詳細について説明します。

状態遷移

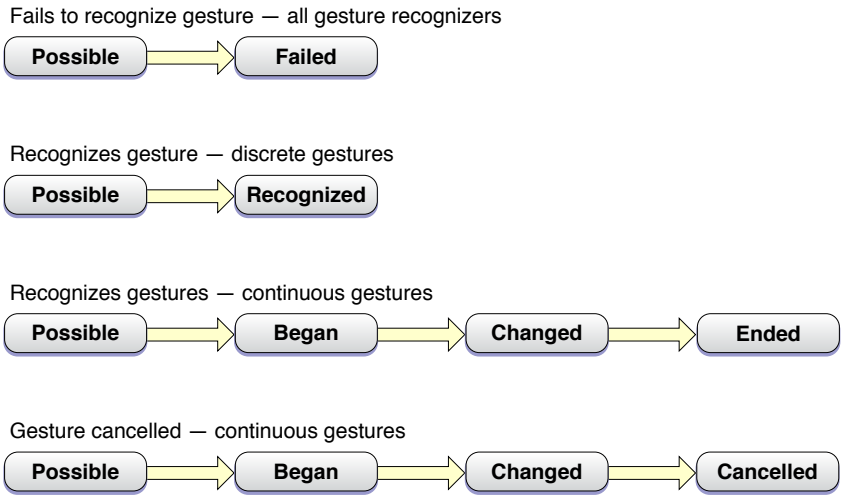
Gesture Recognizerは、定義済みの状態マシンとして動作します。Gesture Recognizerは、特定の条件が満たされるかどうかに応じて、ある状態から別の状態に遷移します。`UIGestureRecognizer.h`の次の`enum`定数は、Gesture Recognizerの状態を定義しています。

```
typedef enum {
    UIGestureRecognizerStatePossible,
    UIGestureRecognizerStateBegan,
    UIGestureRecognizerStateChanged,
    UIGestureRecognizerStateEnded,
    UIGestureRecognizerStateCancelled,
    UIGestureRecognizerStateFailed,
    UIGestureRecognizerStateRecognized = UIGestureRecognizerStateEnded
} UIGestureRecognizerState;
```

Gesture Recognizerが遷移する状態の順番は、認識されるジェスチャが単発ジェスチャか連続的ジェスチャかどうかに応じて異なります。すべてのGesture Recognizerは、Possible状態(`UIGestureRecognizerStatePossible`)から始まります。次に、Gesture Recognizerは、対応するヒットテストビューで発生したマルチタッチシーケンスを解析し、ジェスチャの認識に成功するか、失敗します。Gesture Recognizerがジェスチャを認識しなかった場合は、Failed状態(`UIGestureRecognizerStateFailed`)に遷移します。これは、ジェスチャが単発か連続的かに関係なく、すべてのGesture Recognizerに当てはまります。

一方、ジェスチャが認識された場合は、ジェスチャが単発か連続的によって、状態遷移が異なります。単発のジェスチャ用のGesture Recognizerは、PossibleからRecognized(`UIGestureRecognizerStateRecognized`)に遷移します。一方、連続的なジェスチャ用のGesture Recognizerは、最初にジェスチャを認識したときは、PossibleからBegan(`UIGestureRecognizerStateBegan`)に遷移します。次に、BeganからChanged(`UIGestureRecognizerStateChanged`)に遷移します。その後は、ジェスチャに変化があるたびにChangedからChangedに遷移します。最終的に、マルチタッチシーケンスの最後の指がヒットテストビューから離れたときに、Gesture RecognizerはEnded状態(`UIGestureRecognizerStateEnded`)に遷移します。この状態は、`UIGestureRecognizerStateRecognized`状態の別名です。連続的なジェスチャ用のGesture Recognizerは、認識中のジェスチャが、期待するジェスチャのパターンにこれ以上当てはまらないと判断したら、Changed状態からCancelled状態(`UIGestureRecognizerStateCancelled`)に遷移することもできます。図 5-3 （62 ページ）に、これらの遷移を示します。

図 5-3 Gesture Recognizerで起こり得る状態遷移



注： Began、Changed、Ended、Cancelledの各状態は必ずしも同じタッチフェーズ内の`UITouch`オブジェクトに関連付けられているとは限りません。これらの状態は、認識中のタッチオブジェクトではなく、厳密にジェスチャ自体のフェーズを表します。

ジェスチャが認識されると、それ以降のすべての状態遷移によって、ターゲットにアクションメッセージが送信されます。Gesture RecognizerがRecognized（またはEnded）状態に達すると、新たなジェスチャ認識動作に備えるために、内部状態がリセットされます。そして、`UIGestureRecognizer`クラスはGesture Recognizerの状態をPossibleに戻します。

カスタムGesture Recognizerの実装

カスタムGesture Recognizerを実装するには、まず、Xcodeで`UIGestureRecognizer`のサブクラスを作成します。次に、このサブクラスのヘッダファイルに次のようなimportディレクティブを追加します。

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

そして、次のメソッド宣言を`UIGestureRecognizerSubclass.h`からこのサブクラスのヘッダファイルにコピーします。これらは、サブクラスでオーバーライドするメソッドです。

```
- (void)reset;
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
```

オーバーライドするすべてのメソッド内で、必ずスーパークラスの実装(`super`)を呼び出さなければなりません。

`UIGestureRecognizerSubclass.h`の`state`プロパティの宣言を見てみましょう。（`UIGestureRecognizer.h`では）`readonly`だったものが、`readwrite`オプションになっている点に注目してください。サブクラスでは、このプロパティに`UIGestureRecognizerState`定数を割り当てることによって、状態を変更できるようになっています。

UIGestureRecognizerクラスは、自動的にアクションメッセージを送信して、ヒットテストビューへのタッチオブジェクトの配信を制御します。この作業を自分で実装する必要はありません。

マルチタッチ用のイベント処理メソッドの実装

Gesture Recognizerの実装の中心は、touchesBegan:withEvent:、touchesMoved:withEvent:、touchesEnded:withEvent:、およびtouchesCancelled:withEvent:の4つのメソッドです。これらのメソッドを実装する作業は、カスタムビュー用のメソッドを実装する場合と同様です。

注： マルチタッチシーケンスの間に配信されるイベントの処理の詳細については、『*iPhone Application Programming Guide*』の「Event Handling」の「Handling Multi-Touch Events」を参照してください。

Gesture Recognizer用のメソッドを実装する場合の主な違いは、適切なタイミングで状態を遷移させる点です。それには、stateプロパティの値を適切なUIGestureRecognizerState定数に設定しなければなりません。Gesture Recognizerが単発のジェスチャを認識した場合は、stateプロパティをUIGestureRecognizerStateRecognizedに設定します。ジェスチャが連続的な場合は、まず、stateプロパティをUIGestureRecognizerStateBeganに設定し、次に、ジェスチャの位置が変化するたびにstateプロパティをUIGestureRecognizerStateChangedに設定します（またはリセットします）。ジェスチャが終了したら、stateをUIGestureRecognizerStateEndedに設定します。Gesture Recognizerは、このマルチタッチシーケンスが自分の担当するジェスチャではないと判断した場合は、いつでもstateをUIGestureRecognizerStateFailedに設定できます。

リスト5-3は、単発のシングルタッチの「チェックマーク」ジェスチャ（実際には、任意のV字を描くジェスチャ）用のGesture Recognizerの実装です。このGesture Recognizerは、ジェスチャの中心点（上向きのストロークが始まる位置）の値をクライアントが取得できるようにその中心点を記録します。

リスト 5-3　「チェックマーク」Gesture Recognizerの実装

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesBegan:touches withEvent:event];
    if ([touches count] != 1) {
        self.state = UIGestureRecognizerStateFailed;
        return;
    }
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesMoved:touches withEvent:event];
    if (self.state == UIGestureRecognizerStateFailed) return;
    CGPoint nowPoint = [[touches anyObject] locationInView:self.view];
    CGPoint prevPoint = [[touches anyObject] previousLocationInView:self.view];
    if (!strokeUp) {
        // 下向きのストロークでは、xとyの両方が正の方向に増加する
        if (nowPoint.x >= prevPoint.x && nowPoint.y >= prevPoint.y) {
            self.midPoint = nowPoint;
            // 上向きのストロークでは、xの値は増加するが、yの値は減少する
        } else if (nowPoint.x >= prevPoint.x && nowPoint.y <= prevPoint.y) {
            strokeUp = YES;
        } else {
            self.state = UIGestureRecognizerStateFailed;
        }
    }
}
```

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesEnded:touches withEvent:event];
    if ((self.state == UIGestureRecognizerStatePossible) && strokeUp) {
        self.state = UIGestureRecognizerStateRecognized;
    }
}

- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesCancelled:touches withEvent:event];
    self.midPoint = CGPointZero;
    strokeUp = NO;
    self.state = UIGestureRecognizerStateFailed;
}
```

Gesture Recognizerは、自分の担当するジェスチャには含まれないと判断したタッチ（UITouchオブジェクトで表される）を検出した場合、それを直接ビューに渡すことができます。それには、自身のignoreTouch:forEvent:を呼び出して、そのタッチオブジェクトを渡します。無視されたタッチは、cancelsTouchesInViewプロパティの値がYESに設定されていても、対応するビューに渡されます。

状態のリセット

Gesture RecognizerがUIGestureRecognizerStateRecognized状態またはUIGestureRecognizerStateEnded状態に遷移すると、UIGestureRecognizerクラスは、Gesture Recognizerの状態がUIGestureRecognizerStatePossibleに戻る直前に、そのGesture Recognizerのresetメソッドを呼び出します。Gesture Recognizerクラスは、すべての内部状態をリセットして、新たなジェスチャを認識動作に備えるために、このメソッドを実装しなければなりません。Gesture Recognizerは、このメソッドから戻った後は、すでに始まっているがまだ終了していないタッチに関して更新を受け取らなくなります。

リスト 5-4　Gesture Recognizerのリセット

```
- (void)reset {
    [super reset];
    self.midPoint = CGPointZero;
    strokeUp = NO;
}
```

グラフィックスと描画

描画のために使用する標準的なフレームワークの他に、iPhone OS 3.2では、レンダリングコンテンツを生成するための新機能がいくつか導入されました。UIBezierPathクラスは、ベクトルベースのパスの作成を容易にするCore Graphicsのパスをラップしたobjective-Cのラッパーです。また、PDFコンテンツを使用する場合は、PDFデータを生成したり、それをファイルやデータオブジェクトに保存したりするために使用できる関数がいくつか導入されました。

ベジエパスを使用した図形の描画

iPhone OS 3.2では、UIBezierPathクラスを使用してベクトルベースのパスを作成できるようになりました。このクラスは、Core Graphicsフレームワークのパス関連機能をラップしたObjective-Cのラッパーです。このクラスを使用すると、楕円や矩形などの単純な図形だけでなく、複数の直線や曲線のセグメントを組み込んだ複雑な図形も定義できます。

アプリケーションのユーザインターフェイスに図形を描画するには、パスオブジェクトを使用します。パスの輪郭を描いたり、それで囲まれた領域を塗りつぶしたり、その両方を実行したりできます。また、パスを使用して現在のグラフィックスコンテキストにクリッピング領域を定義し、それを使用してそのコンテキストでのその後の描画操作を変更できます。

ベジエパスの基礎

UIBezierPathオブジェクトは、CGPathRefデータ型のラッパーです。**パス**は、直線や曲線のセグメントを使用して作成されるベクトルベースの図形です。矩形や多角形を作成するには、直線のセグメントを使用します。一方、円弧、円、複雑な曲線を持つ図形を作成するには、曲線のセグメントを使用します。各セグメントは、（現在の座標系での）1つ以上の点と、これらの点の解釈の仕方を定義した描画コマンドから構成されます。直線や曲線のセグメントの終端は、次のセグメントの始点になります。いくつかの直線や曲線のセグメントが接続されると、**サブパス**と呼ばれる図形を形成します。そして1つのUIBezierPathオブジェクトには、パス全体を定義する1つ以上のサブパスが含まれることもあります。

パスオブジェクトを作成する処理と使用する処理は独立です。パスの作成は最初の処理です。これには次の手順が含まれます。

1. パスオブジェクトを作成する。
2. moveToPoint:メソッドを使用して、最初のセグメントの始点を設定する。
3. 直線や曲線のセグメントを追加して、1つ以上のサブパスを定義する。
4. UIBezierPathオブジェクトの関連する描画属性を変更する。たとえば、ストロークパスのlineWidthプロパティやlineJoinStyleプロパティを設定したり、塗りつぶしパスのusesEvenOddFillRuleプロパティを設定したりします。これらの値は、必要に応じて後からいつでも変更できます。

パスを作成する際には、原点(0, 0)を起点としてパスの点を構成するようにします。そうしておくと、後でパスを移動する操作が簡単になります。描画中は、パスの点には、現在のグラフィックスコンテキストの座標系がそのまま適用されます。パスが原点を起点として構成されていれば、そのパスの位置を変更するのに、変換係数を指定したアフィン変換を現在のグラフィックスコンテキストに適用するだけで済みます。（パスオブジェクト自体を変更する代わりに）グラフックコンテキストを変更することの利点は、グラフィックスの状態を保存しておけば、それを復元することで、簡単に変換を元に戻せる点です。

パスオブジェクトを描画するには、`stroke`メソッドと`fill`メソッドを使用します。これらのメソッドは、現在のグラフィックスコンテキストに、パスの直線や曲線のセグメントをレンダリングします。このレンダリング処理には、パスオブジェクトの属性を使用して直線や曲線のセグメントをラスタ化することが含まれます。ラスタ化処理を行っても、パスオブジェクトそのものは変更されません。その結果、同じパスオブジェクトを現在のコンテキストに何度でもレンダリングできます。

パスへの直線と多角形の追加

直線と多角形は、`moveToPoint:`メソッドと`addLineToPoint:`メソッドを使用して、点を並べて作成する単純な図形です。`moveToPoint:`メソッドは、作成する図形の始点を設定します。その点から、`addLineToPoint:`メソッドを使用して図形の直線部分を作成します。直前に指定した点と新たに指定した点の間に直線を形成し、直線を連続的に作成していきます。

リスト6-1に、別々の直線セグメントを使用して五角形を作成するために必要なコードを示します。このコードでは、図形の最初の点を設定した後に、接続された4つの直線セグメントを追加します。5番目のセグメントは、`closePath`メソッドの呼び出しによって追加されます。このメソッドは、最後の点(0, 40)と最初の点(100, 0)を接続します。

リスト 6-1　五角形の作成

```
UIBezierPath*    aPath = [UIBezierPath bezierPath];

// 図形の始点を設定する
[aPath moveToPoint:CGPointMake(100.0, 0.0)];

// 直線を描画する
[aPath addLineToPoint:CGPointMake(200.0, 40.0)];
[aPath addLineToPoint:CGPointMake(160, 140)];
[aPath addLineToPoint:CGPointMake(40.0, 140)];
[aPath addLineToPoint:CGPointMake(0.0, 40.0)];
[aPath closePath];
```

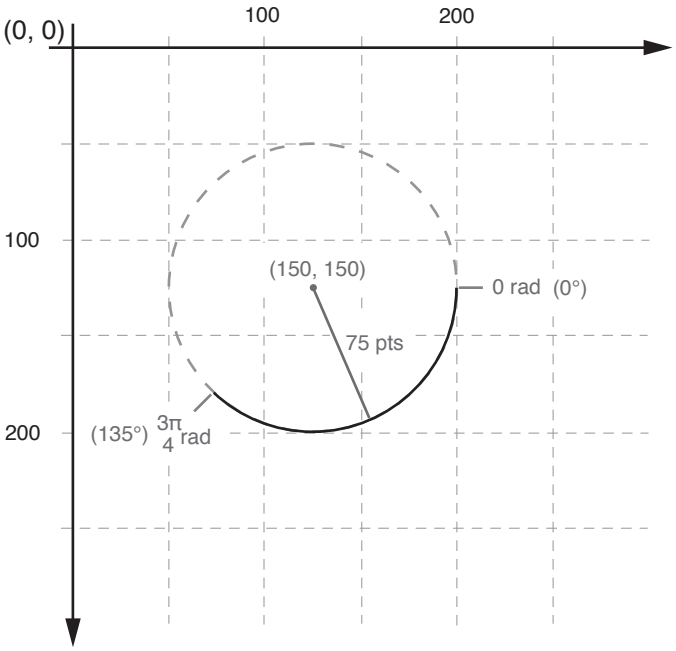
`closePath`メソッドを使用すると、図形を終了させるだけでなく、最初の点と最後の点の間に直線を引くことができます。これは、最後の直線を描画する必要なしに多角形を完成させる便利な方法です。

パスへの弧の追加

`UIBezierPath`クラスでは、新しいパスオブジェクトを弧セグメントで初期化できます。`bezierPathWithArcCenter:radius:startAngle:endAngle:clockwise:`メソッドのパラメータは、希望する弧と、その弧の始点と終点を含む円を定義します。図 6-1に、弧の作成に必要な構成要素を示します。これには、この弧を定義する円と、それを指定するために使用する角度も含まれてい

ます。ここでは、弧を時計回りに作成しています（代わりに、反時計回りに弧を描画すると、この円の破線部分が描画されます）。この弧を作成するためのコードをリスト 6-2 （67 ページ）に示します。

図 6-1 デフォルトの座標系での弧



リスト 6-2 弧パスの新規作成

```
// piは、ほぼ3.14159265359に等しい
#define   DEGREES_TO_RADIANS(degrees)  ((pi * degrees)/ 180)

- (UIBezierPath*)createArcPath
{
   UIBezierPath* aPath = [UIBezierPath bezierPathWithArcCenter:CGPointMake(150,
 150)
                          radius:75
                          startAngle:0
                          endAngle:DEGREES_TO_RADIANS(135)
                          clockwise:YES];
   return aPath;
}
```

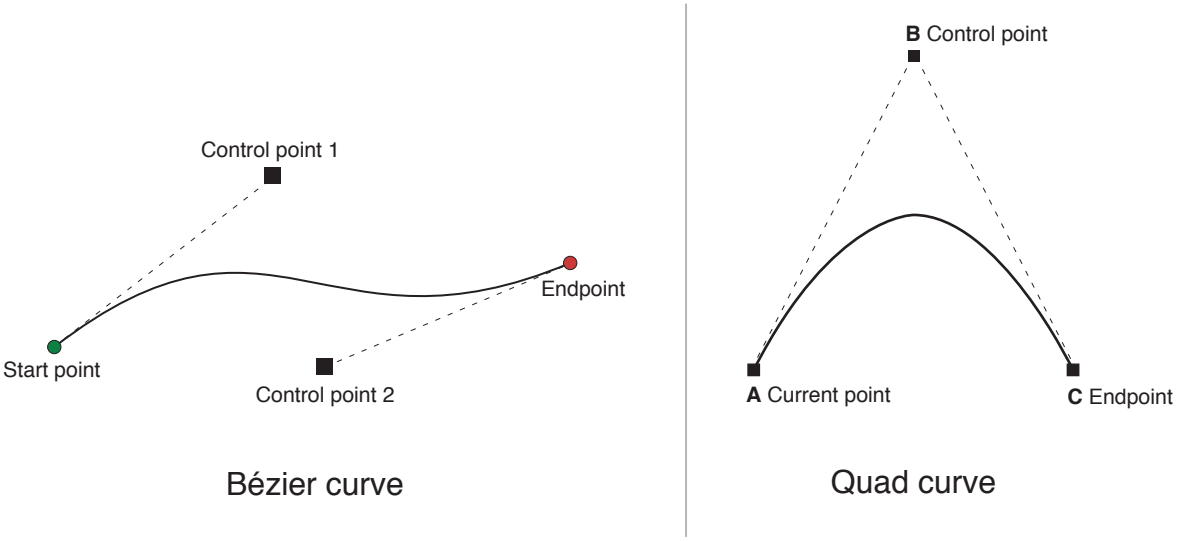
パスの途中に弧セグメントを組み込みたい場合は、そのパスオブジェクトの`CGPathRef`データ型を直接変更しなければなりません。Core Graphicsの関数を使用してパスを変更する方法の詳細については、「Core Graphicsの関数を使用したパスの変更」 （69 ページ）を参照してください。

パスへの曲線の追加

`UIBezierPath`クラスは、2次および3次のベジエ曲線をパスに追加する機能をサポートしています。曲線セグメントは、現在の点から始まり、指定した点で終了します。曲線の形状は、始点と終点と1つ以上の制御点の間の接線を使用して定義されます。図 6-2に、2種類の曲線の近似値と、制御点

と曲線形状の関係を示します。各セグメントの正確な曲率には、すべての点の間の複雑な数学的関係が関与しています。これについては、オンラインドキュメントおよびWikipediaに詳しく記述されています。

図 6-2　パス内の曲線セグメント



曲線をパスに追加するには、次のメソッドを使用します。

- **三次曲線：** `addCurveToPoint:controlPoint1:controlPoint2:`
- **二次曲線：** `addQuadCurveToPoint:controlPoint:`

曲線は、パスの現在の点によって異なるため、上記のいずれかのメソッドを呼び出す前に、現在の点を設定しなければなりません。曲線が完成すると、現在の点は、新たに指定した終点に更新されます。

楕円パスと矩形パスの作成

楕円と矩形は、曲線と直線のセグメントを組み合わせて作成される一般的なタイプのパスです。`UIBezierPath`クラスには、楕円や矩形のパスを作成するための簡易メソッドとして`bezierPathWithRect:`と`bezierPathWithOvalInRect:`があります。これらのメソッドはどちらも、パスオブジェクトを新規に作成して、それを指定された形状で初期化します。メソッドから返されたパスオブジェクトをすぐに使用することも、必要に応じてそれにいくつかの形状を追加することもできます。

既存のパスオブジェクトに矩形を追加するには、ほかの多角形を追加するときと同様に、`moveToPoint:`、`addLineToPoint:`、および`closePath`の各メソッドを使用しなければなりません。矩形の最後の辺に`closePath`メソッドを使用すると、パスの最後の直線を追加するのと同時に、この矩形サブパスの終了を示すことができるので便利です。

既存のパスに楕円を追加する場合は、Core Graphicsを使用する方法が最も簡単です。`addQuadCurveToPoint:controlPoint:`を使用して楕円の曲線を近似することもできますが、`CGPathAddEllipseInRect`関数の方が、はるかに簡単に使用できて正確です。詳細については、「Core Graphicsの関数を使用したパスの変更」（69 ページ）を参照してください。

Core Graphicsの関数を使用したパスの変更

UIBezierPathクラスは、実際にはCGPathRefデータ型とそのパスに関連付けられている描画属性をラップしたラッパーに過ぎません。通常は、UIBezierPathクラスのメソッドを使用して直線や曲線のセグメントを追加しますが、このクラスは、基盤となるパスデータ型を直接変更できるCGPathプロパティも公開しています。Core Graphicsフレームワークの関数を使用してパスを作成したい場合は、このプロパティを使用します。

UIBezierPathオブジェクトに関連付けられているパスを変更するには2つの方法があります。Core Graphicsの関数を使用してパス全体を変更する方法、およびCore Graphicsの関数とUIBezierPathのメソッドを組み合わせて使用する方法の2つです。Core Graphics呼び出しを使用してパス全体を変更する方が、いくらか簡単です。可変のCGPathRefデータ型を作成し、パス情報の変更に必要な関数を呼び出します。それが終了したら、そのパスオブジェクトを、対応するUIBezierPathオブジェクトに割り当てます。そのコード例をリスト 6-3に示します。

リスト 6-3　新規のCGPathRefをUIBezierPathオブジェクトに割り当てる

```
// パスデータを作成する
CGMutablePathRef cgPath = CGPathCreateMutable();
CGPathAddEllipseInRect(cgPath, NULL, CGRectMake(0, 0, 300, 300));
CGPathAddEllipseInRect(cgPath, NULL, CGRectMake(50, 50, 200, 200));

// ここでUIBezierPathオブジェクトを作成する
UIBezierPath* aPath = [UIBezierPath bezierPath];
aPath.CGPath = cgPath;
aPath.usesEvenOddFillRule = YES;

// UIBezierPathオブジェクトに割り当てた後は、
// CGPathRefデータ型を安全に解放できる
CGPathRelease(cgPath);
```

Core Graphicsの関数とUIBezierPathのメソッドを組み合わせる方法を選んだ場合は、これら2つの間でパス情報を注意深くやり取りしなければなりません。UIBezierPathオブジェクトは、基盤となるCGPathRefデータ型を内部に所有しているため、単純にそのデータ型を取得して直接変更を加えることはできません。代わりに、可変のコピーを作成して、そのコピーを変更してから、CGPathプロパティに戻さなければなりません。そのコード例をリスト 6-4に示します。

リスト 6-4　Core Graphics呼び出しとUIBezierPath呼び出しの併用

```
UIBezierPath*    aPath = [UIBezierPath bezierPathWithOvalInRect:CGRectMake(0,
0, 300, 300)];

// CGPathRefを取得し、その可変バージョンを作成する
CGPathRef cgPath = aPath.CGPath;
CGMutablePathRef  mutablePath = CGPathCreateMutableCopy(cgPath);

// パスを変更して、それをUIBezierPathオブジェクトに戻す
CGPathAddEllipseInRect(mutablePath, NULL, CGRectMake(50, 50, 200, 200));
aPath.CGPath = mutablePath;

// パスの可変コピーと不変コピーの両方を解放する
CGPathRelease(cgPath);
CGPathRelease(mutablePath);
```

ベジエパスオブジェクトのコンテンツのレンダリング

UIBezierPathオブジェクトを作成したら、そのオブジェクトのstrokeメソッドやfillメソッドを使用して、パスを現在のグラフィックスコンテキストにレンダリングします。ただし、これらのメソッドを呼び出す前に、パスが正しく描画されるように、通常は次のような作業を行います。

- UIColorクラスのメソッドを使用して希望のストローク色と塗りつぶし色を設定する。
- ターゲットビュー内の希望の位置に図形を配置する。

  原点(0, 0)を起点としてパスを作成した場合は、現在の描画コンテキストに適切なアフィン変換を適用できます。たとえば、点(10, 10)を始点とする図形を描画するには、CGContextTranslateCTM関数を呼び出して、水平方向と垂直方向の両方の変換値として10を指定します。（パスオブジェクトの点を調整するよりも）グラフィックスコンテキストを調整する方が好まれます。以前のグラフィックスの状態を保存しておけば、それを復元することで簡単に変更を元に戻せるからです。
- パスオブジェクトの描画属性を更新する。UIBezierPathインスタンスの描画属性は、パスをレンダリングするときに、グラフィックスコンテキストに関連付けられている値よりも優先されます。

リスト6-5に、カスタムビューに楕円を描画するdrawRect:メソッドの簡単な実装を示します。この楕円に接する矩形の左上隅は、このビューの座標系の点(50, 50)にあります。塗りつぶし操作はパスの境界線上まで塗りつぶしを行うため、このメソッドでは、境界線を描く前に塗りつぶしを実行します。これによって、境界線の半分が塗りつぶし色によって隠れてしまうのを防止できます。

リスト 6-5　ビューにパスを描画する

```
- (void)drawRect:(CGRect)rect
{
    // 描画するための楕円を作成する
    UIBezierPath* aPath = [UIBezierPath bezierPathWithOvalInRect:
                                CGRectMake(0, 0, 200, 100)];

    // レンダリング色を設定する
    [[UIColor blackColor] setStroke];
    [[UIColor redColor] setFill];

    CGContextRef aRef = UIGraphicsGetCurrentContext();

    // 図形の後に描画するコンテンツがある場合は、
    // 変換を行う前に、現在の状態を保存する
    //CGContextSaveGState(aRef);

    // ビューの原点を一時的に調整する。この楕円は
    // 新しい原点に基づいて描画される
    CGContextTranslateCTM(aRef, 50, 50);

    // 必要であれば、描画オプションを調整する
    aPath.lineWidth = 5;

    // 塗りつぶし色によって境界線が隠れてしまわないように
    // 境界線を描く前にパスを塗りつぶす
    [aPath fill];
    [aPath stroke];
```

```
    // その他のコンテンツを描画する前に、グラフィックスの状態を復元する
    //CGContextRestoreGState(aRef);
}
```

パス上でのヒット検出

パスの塗りつぶし領域でタッチイベントが発生したかどうかを判断するには、`UIBezierPath`の`containsPoint:`メソッドを使用します。このメソッドは、パスオブジェクト内のすべての閉じたサブパスに対して、指定された点をテストし、これらのサブパスの上または内部にその点が存在する場合は`YES`を返します。

重要： `containsPoint:`メソッドとCore Graphicsのヒットテスト関数は、閉じたパスを対象とした場合にのみ動作します。これらのメソッドは、開いたサブパスのヒットに対しては、常に`NO`を返します。開いたサブパスを対象にヒット検出を実行したい場合は、そのパスオブジェクトのコピーを作成して、点をテストする前に、開いたサブパスを閉じます。

パスの（塗りつぶし領域ではなく）ストローク部分のヒットテストを実行したい場合は、Core Graphicsを使用しなければなりません。`CGContextPathContainsPoint`関数を利用すると、現在グラフィックスコンテキストに割り当てられているパスの塗りつぶし部分、またはストローク部分のいずれかの領域を対象に点をテストできます。リスト 6-6に、指定した点が、指定したパスと交わるかどうかをテストするメソッドを示します。*inFill*パラメータを使用して、点のテストをパスの塗りつぶし部分に対して行うか、ストローク部分に対して行うかを、呼び出し側で指定できます。ヒット検出が成功するためには、呼び出し側から渡されたパスに、1つ以上の閉じたサブパスが含まれていなければなりません。

リスト 6-6　パスオブジェクトに対する点のテスト

```
- (BOOL)containsPoint:(CGPoint)point onPath:(UIBezierPath*)path
inFillArea:(BOOL)inFill
{
   CGContextRef context = UIGraphicsGetCurrentContext();
   CGPathRef cgPath = path.CGPath;
   BOOL    isHit = NO;

   // 使用する描画モードを決定する。デフォルトでは
   // パスのストローク部分に対するヒットが検出される
   CGPathDrawingMode mode = kCGPathStroke;
   if (inFill)
   {
      // 代わりに、パスの塗りつぶし領域でのヒットを検出する
      if (path.usesEvenOddFillRule)
         mode = kCGPathEOFill;
      else
         mode = kCGPathFill;
   }

   // パスを後で削除できるように
   // グラフィックスの状態を保存する
   CGContextSaveGState(context);
   CGContextAddPath(context, cgPath);

   // ヒット検出を実行する
   isHit = CGContextPathContainsPoint(context, point, mode);
```

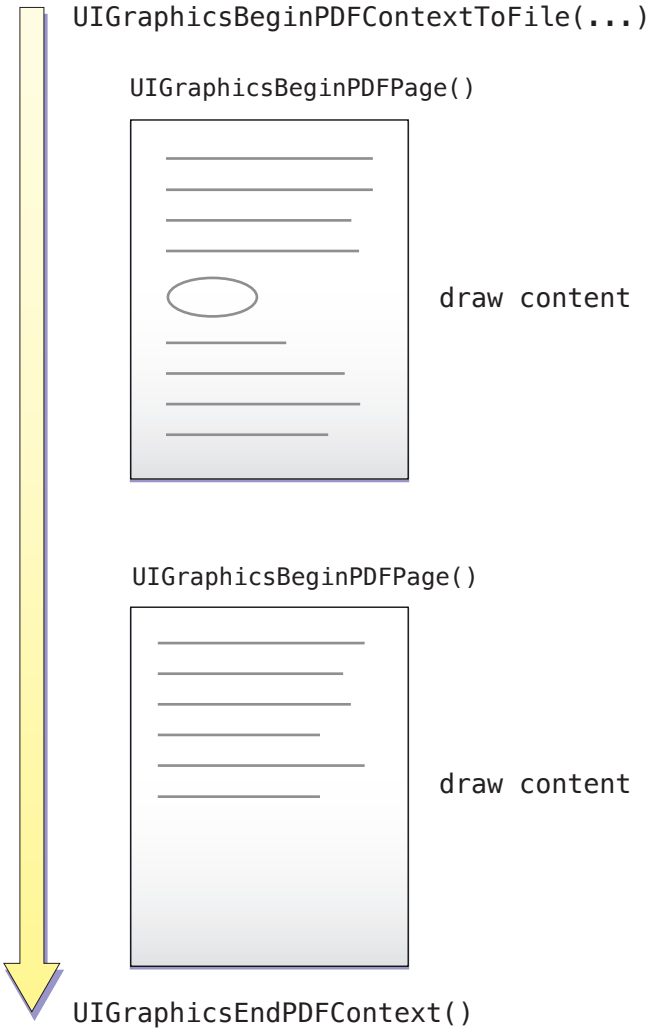
```
    CGContextRestoreGState(context);

    return isHit;
}
```

PDFコンテンツの生成

iPhone OS 3.2では、UIKitフレームワークが、ネイティブの描画コードを使用してPDFコンテンツを生成する関数セットを提供しています。これらの関数を利用すると、PDFファイルやPDFデータオブジェクトをターゲットにしたグラフィックスコンテキストを作成できます。そして、画面に描画する際に使用するのと同じUIKitやCore Graphicsの描画ルーチンを使用して、このグラフィックスコンテキストに描画できます。PDFには任意の数のページを作成できます。描画が終了すると、PDFバージョンの描画結果が残ります。

図6-3に、ローカルのファイルシステムにPDFファイルを作成するためのワークフローを示します。`UIGraphicsBeginPDFContextToFile`関数は、PDFコンテキストを作成して、それをファイル名に関連付けます。コンテキストを作成したら、`UIGraphicsBeginPDFPage`関数を使用して、最初のページを開きます。ページを開いたら、そのコンテンツの描画を開始できます。新しいページを作成するには、単に`UIGraphicsBeginPDFPage`を再度呼び出してから、描画を開始します。描画が終了したら、`UIGraphicsEndPDFContext`関数を呼び出して、グラフィックスコンテキストを閉じ、描画結果データをPDFファイルに書き込みます。

図 6-3 PDF文書を作成するためのワークフロー



以降の各セクションでは、PDF作成の手順について、例を使ってさらに詳しく説明します。PDFコンテンツを作成するために使用する関数の詳細については、『*UIKit Function Reference*』を参照してください。

PDFコンテキストの作成と設定

PDFグラフィックスコンテキストは、`UIGraphicsBeginPDFContextToData`関数または`UIGraphicsBeginPDFContextToFile`関数のいずれかを使用して作成します。これらの関数は、グラフィックスコンテキストを作成して、それをPDFデータの保存先に関連付けます。`UIGraphicsBeginPDFContextToData`関数の場合、保存先はこの関数に渡される`NSMutableData`オブジェクトです。一方、`UIGraphicsBeginPDFContextToFile`関数の場合、保存先はアプリケーションのホームディレクトリ内のファイルです。

PDF文書は、ページ単位の構造を使用してコンテンツを編成します。この構造には、どのような描画を実行する場合にも次の2つの制限があります。

- 任意の描画コマンドを発行する前に、開いているページが1つ存在する必要がある。
- 各ページのサイズを指定する必要がある。

PDFグラフックコンテキストを作成するために使用する関数では、デフォルトのページサイズを指定することはできますが、これらの関数が自動的にページを開くわけではありません。コンテキストを作成したら、`UIGraphicsBeginPDFPage`関数または`UIGraphicsBeginPDFPageWithInfo`関数のいずれかを使用して、明示的に新しいページを開く必要があります。また、新しいページを作成したい場合は、毎回これらの関数のいずれかを再度呼び出して、新しいページの開始を指示しなければなりません。`UIGraphicsBeginPDFPage`関数は、デフォルトのサイズを使用してページを作成します。一方、`UIGraphicsBeginPDFPageWithInfo`関数を利用すると、ページサイズやその他の属性をカスタマイズできます。

描画が終了したら、`UIGraphicsEndPDFContext`を呼び出して、PDFグラフィックスコンテキストを閉じます。この関数は、最後のページを閉じて、コンテキストの作成時に指定されたファイルやデータオブジェクトにPDFコンテンツを書き込みます。また、この関数は、PDFコンテキストをグラフィックスコンテキストのスタックから削除します。

リスト6-7に、テキストビュー内のテキストからPDFファイルを作成するために、アプリケーションが使用する処理ループを示します。PDFコンテキストを設定したり管理したりするための3つの関数呼び出し以外は、ほとんどのコードは、必要なコンテンツの描画に関係するものです。`textView`メンバ変数は、必要なテキストが含まれている`UITextView`オブジェクトを指します。このアプリケーションでは、Core Textフレームワーク（具体的には、`CTFramesetterRef`データ型）を使用して、テキストのレイアウトや連続するページの管理を行います。独自の`renderPageWithTextRange:andFramesetter:`メソッドと`drawPageNumber:`メソッドの実装を、リスト 6-8 （75 ページ）に示します。

リスト 6-7　PDFファイルの新規作成

```
- (IBAction)savePDFFile:(id)sender
{
    // これは、PDFファイルの名前を取得するためのカスタムメソッド
    NSString* pdfFileName = [self getPDFFileName];

    // 612×792のデフォルトのページサイズを使用してPDFコンテキストを作成する
    UIGraphicsBeginPDFContextToFile(pdfFileName, CGRectZero, nil);

    // テキストを準備して、レイアウトを処理するCTFramesetterを作成する
    CFAttributedStringRef currentText = CFAttributedStringCreate(NULL,
(CFStringRef)textView.text, NULL);
    CTFramesetterRef framesetter =
CTFramesetterCreateWithAttributedString(currentText);
    if (!framesetter)
        return;

    // いくつかのローカル変数をセットアップする
    CFRange currentRange = CFRangeMake(0, 0);
    NSInteger currentPage = 0;
    BOOL done = NO;

    // 個々のページを作成するメインループを開始する
    do
    {
        // 新規ページの開始を指定する
        UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 612, 792), nil);
```

```
        // 各ページの下端にページ番号を描画する
        currentPage++;
        [self drawPageNumber:currentPage];

        // 現在のページをレンダリングして、現在の範囲を
        // 次のページの先頭を指すように更新する
        currentRange = [self renderPageWithTextRange:currentRange
andFramesetter:framesetter];

        // テキストの終わりに達したら、ループを終了する
        if (currentRange.location ==
CFAttributedStringGetLength((CFAttributedStringRef)currentText))
            done = YES;
    }
    while (!done);

    // PDFコンテキストを閉じて、コンテンツを書き出す
    UIGraphicsEndPDFContext();

    // 後始末をする
    CFRelease(currentText);
    CFRelease(framesetter);
}
```

PDFページの描画

PDFの描画はすべて、ページのコンテキスト内で実行されなければなりません。どのPDF文書にも少なくとも1つのページがあります。ほとんどのPDF文書には複数のページがあります。新しいページの開始を指定するには、`UIGraphicsBeginPDFPage`関数または`UIGraphicsBeginPDFPageWithInfo`関数を呼び出します。これらの関数は、（既にページが開いている場合は）前のページを閉じて、新しいページを作成し、そこに描画できるようにします。`UIGraphicsBeginPDFPage`は、デフォルトのサイズを使用して新しいページを作成します。一方、`UIGraphicsBeginPDFPageWithInfo`関数を利用すると、ページサイズや、PDFページのその他の属性をカスタマイズできます。

ページを作成したら、その後のすべての描画コマンドは、PDFグラフィックスコンテキストに捕捉されて、PDFコマンドに変換されます。ページには、アプリケーションのカスタムビューに描画するときと同様に、テキスト、ベクトル図形、画像など、何でも描画できます。発行した描画コマンドは、PDFコンテキストに捕捉され、PDFデータに変換されます。ページ上でのコンテンツの配置は、完全にデベロッパに任されていますが、そのページの境界矩形内で行わなければなりません。

リスト 6-8に、PDFページ内にコンテンツを描画するために使われる2つのカスタムメソッドを示します。`renderPageWithTextRange:andFramesetter:`メソッドは、Core Textを使用して、ページに合ったテキストフレームを作成し、そのフレーム内にテキストを配置します。テキストを配置したら、現在のページの終わりと次のページの始まりを表す新しい範囲を返します。`drawPageNumber:`メソッドは、`NSString`の描画機能を使用して、各PDFページの下端にページ番号文字列を描画します。

リスト 6-8　ページ単位のコンテンツの描画

```
// Core Textを使用して、ページ上のフレームにテキストを描画する
- (CFRange)renderPage:(NSInteger)pageNum withTextRange:(CFRange)currentRange
                andFramesetter:(CTFramesetterRef)framesetter
{
    // グラフィックスコンテキストを取得する
```

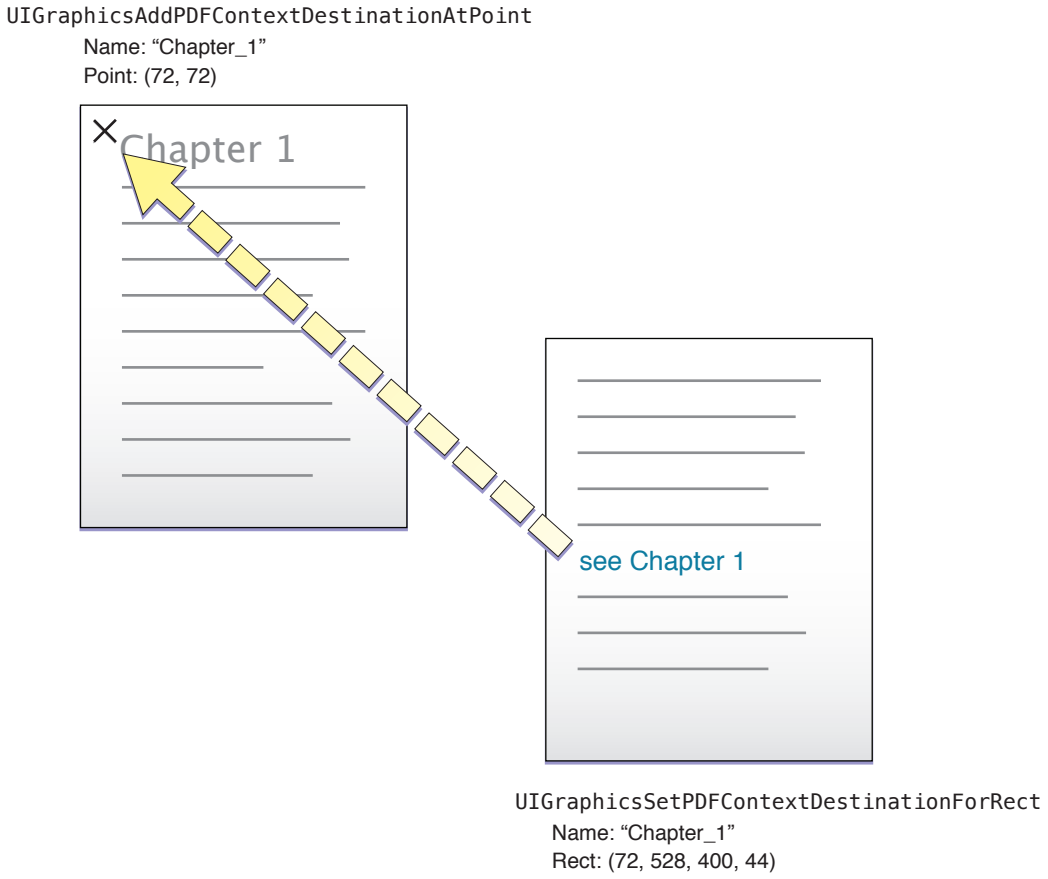
```
    CGContextRef    currentContext = UIGraphicsGetCurrentContext();

    // テキスト行列を既知の状態にする。これによって
    // 古い拡大縮小倍率が残っていないことが保証される
    CGContextSetTextMatrix(context, CGAffineTransformIdentity);

    // テキストを囲むパスオブジェクトを作成する。テキストの周囲の
    // マージンを72ポイントに設定する
    CGRect    frameRect = CGRectMake(72, 72, 468, 648);
    CGMutablePathRef framePath = CGPathCreateMutable();
    CGPathAddRect(framePath, NULL, frameRect);

    // レンダリングを実行するフレームを取得する
    // currentRange変数は、開始位置だけを指定する。framesetterは
    // フレームに入る分量のテキストを配置する
    CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange,
framePath, NULL);
    CGPathRelease(framePath);

    // Core Textは左下隅から上に向かって描画する。
    // このため、描画する前に現在の変換を反転する。
    CGContextTranslateCTM(currentContext, 0, 792);
    CGContextScaleCTM(currentContext, 1.0, -1.0);

    // フレームを描画する
    CTFrameDraw(frameRef, currentContext);

    // 描画結果に基づいて、現在の範囲を更新する
    currentRange = CTFrameGetVisibleStringRange(frameRef);
    currentRange.location += currentRange.length;
    currentRange.length = 0;
    CFRelease(frameRef);

    return currentRange;
}

- (void)drawPageNumber:(NSInteger)pageNum
{
    NSString* pageString = [NSString stringWithFormat:@"Page %d", pageNum];
    UIFont* theFont = [UIFont systemFontOfSize:12];
    CGSize maxSize = CGSizeMake(612, 72);

    CGSize pageStringSize = [pageString sizeWithFont:theFont
                                        constrainedToSize:maxSize
                                        lineBreakMode:UILineBreakModeClip];
    CGRect stringRect = CGRectMake(((612.0 - pageStringSize.width) / 2.0),
                                   720.0 + ((72.0 - pageStringSize.height) / 2.0)
 ,
                                    pageStringSize.width,
                                    pageStringSize.height);

    [pageString drawInRect:stringRect withFont:theFont];
}
```

PDFコンテンツ内でのリンクの作成

コンテンツを描画するだけでなく、同じPDFファイル内の別のページや外部URLへのリンクを含めることもできます。1つのリンクを作成するには、PDFページにソース矩形とリンク先を追加しなければなりません。リンク先の属性の1つは、そのリンクの一意の識別子としての役割を果たす文字列です。特定のリンク先へのリンクを作成するには、ソース矩形を作成するときに、そのリンク先の一意の識別子を指定します。

新しいリンク先をPDFコンテンツに追加するには、`UIGraphicsAddPDFContextDestinationAtPoint`関数を使用します。この関数は、名前付きのリンク先を現在のページの特定の位置に関連付けます。そのリンク先へのリンクを作成するには、`UIGraphicsSetPDFContextDestinationForRect`関数を使用して、リンク用のソース矩形を指定します。図 6-4に、PDF文書のページに適用した場合の、これら2つの関数呼び出しの関係を示します。「see Chapter 1」というテキストを囲む矩形をタップすると、それに対応するリンク先（Chapter 1の先頭）に移動できます。

図 6-4　リンク先とジャンプ元の作成



文書内へのリンクを作成できるほかに、`UIGraphicsSetPDFContextURLForRect`関数を使用して、文書の外部にあるコンテンツへのリンクも作成できます。この関数を使用してリンクを作成するときは、あらかじめリンク先を作成しておく必要はありません。必要なのは、この関数を使用して、ターゲットURLと、現在のページ上のソース矩形を指定することだけです。

テキストの独自処理と入力

iPadの広い画面を利用すると、単純なテキスト入力だけでなく、複雑なテキスト処理や独自の入力も、多くのアプリケーションに魅力的な可能性をもたらします。アプリケーションは、独自のテキストレイアウト、フォント管理、オートコレクト、カスタムキーボード、スペルチェック、選択による変更、およびマルチステージ入力などの機能を持つことができます。iPhone OS 3.2には、これらの機能を実現するいくつかのテクノロジーが含まれています。この章では、これらのテクノロジーについて説明し、それをアプリケーションに組み込むために必要な手順について説明します。

入力ビューと入力アクセサリビュー

UIKitフレームワークには、カスタム入力ビューと入力アクセサリビューのサポート機能が含まれています。アプリケーションは、ユーザがビュー内のテキストやその他の形式のデータを編集するときに、システムキーボードを独自の入力ビューに置き換えることができます。たとえば、アプリケーションは、カスタム入力ビューを使用して、ルーン文字を入力できるようにすることもできます。また、システムキーボードやカスタム入力ビューに入力アクセサリビューを付加することもできます。このアクセサリビューは、メインの入力ビューの上端に表示され、そこには、何らかの方法でテキストを変更するためのコントロールや、そのテキストについての情報を表示するラベルなどを含めることができます。

アプリケーションが、テキストの編集に`UITextView`オブジェクトや`UITextField`オブジェクトを使用している場合、この機能を利用するには、`inputView`プロパティと`inputAccessoryView`プロパティにカスタムビューを割り当てるだけです。これらのカスタムビューは、テキストオブジェクトがファーストレスポンダになるときに表示されます。

入力ビューや入力アクセサリビューは、フレームワークが提供するテキストオブジェクトに限られているわけではありません。`UIResponder`を直接または間接に継承する任意のクラス（通常はカスタムビュー）で、独自の入力ビューや入力アクセサリビューを指定できます。`UIResponder`クラスには、入力ビューと入力アクセサリビュー用に次の2つのプロパティが宣言されています。

```
@property (readonly, retain) UIView *inputView;
@property (readonly, retain) UIView *inputAccessoryView;
```

レスポンダオブジェクトがファーストレスポンダになっていて、`inputView`（または`inputAccessoryView`）が`nil`でない場合は、UIKitが入力ビューを親ビューの下にアニメーションで表示します（または、入力アクセサリビューを入力ビューの上端に添付します）。ファーストレスポンダは、`UIResponder`の`reloadInputViews`メソッドを呼び出すことによって、入力ビューと入力アクセサリビューを再ロードできます。

`UITextView`クラスでは、`inputView`プロパティと`inputAccessoryView`プロパティを`readwrite`として再宣言しています。`UITextView`オブジェクトのクライアントは、nibファイルをロードするか、コードでビューを作成することによって、入力ビューと入力アクセサリビューを取得し、それらをプロパティに割り当てるだけです。カスタムビュークラス（および、`UIResponder`を継承するその他のサブクラス）では、これらのプロパティの一方または両方と、それに関連するインスタンス変数を再宣言して、このプロパティのgetterメソッドをオーバーライドしなければなりません（つま

り、このプロパティのアクセサメソッドを合成してはいけません）。getterメソッドの実装では、ビューをロードするか、ビューがまだ存在しない場合は作成して、そのビューを返さなければなりません。

入力ビューや入力アクセサリビューのサイズやコンテンツの定義は、非常に柔軟に行うことができます。これらのビューの高さは自由に変更できますが、幅はシステムキーボードと同じにしておくべきです。UIKitは、自動サイズ変更マスクが`UIViewAutoresizingFlexibleHeight`の値に設定されている入力ビューを検出すると、その高さをキーボードと同じなるように変更します。入力ビューと入力アクセサリビューが持つことができるサブビュー（コントロールなど）の数に制限はありません。入力ビューと入力アクセサリビューに関する詳細なガイダンスについては、『*iPad Human Interface Guidelines*』を参照してください。

実行時にnibファイルをロードするには、まず、Interface Builderで入力ビューまたは入力アクセサリビューを作成します。そして、実行時にアプリケーションのメインバンドルを取得して、その`loadNibNamed:owner:options:`を呼び出します。その際に、nibファイルの名前、そのnibファイルのFile's Owner、および必要なオプションを渡します。このメソッドは、nibファイル内の最上位レベルのオブジェクトの配列を返します。この配列には、入力ビューや入力アクセサリビューが含まれています。このビューを、対応するプロパティに割り当てます。このテーマの詳細については、『*Resource Programming Guide*』の「Nib Files」を参照してください。

リスト 7-1に、`inputAccessoryView` getterメソッドで入力アクセサリビューを必要に応じて作成するカスタムビュークラスを示します。

リスト 7-1　プログラムによる入力アクセサリビューの作成

```
- (UIView *)inputAccessoryView {
    if (!inputAccessoryView) {
        CGRect accessFrame = CGRectMake(0.0, 0.0, 768.0, 77.0);
        inputAccessoryView = [[UIView alloc] initWithFrame:accessFrame];
        inputAccessoryView.backgroundColor = [UIColor blueColor];
       UIButton *compButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
        compButton.frame = CGRectMake(313.0, 20.0, 158.0, 37.0);
       [compButton setTitle:@"Word Completions" forState:UIControlStateNormal];
        [compButton setTitleColor:[UIColor blackColor]
forState:UIControlStateNormal];
        [compButton addTarget:self action:@selector(completeCurrentWord:)
            forControlEvents:UIControlEventTouchUpInside];
        [inputAccessoryView addSubview:compButton];
    }
    return inputAccessoryView;
}
```

システムキーボードの場合と同様に、UIKitは、`UIKeyboardWillShowNotification`、`UIKeyboardDidShowNotification`、`UIKeyboardWillHideNotification`、および`UIKeyboardDidHideNotification`の各通知を送信します。これらの通知を受け取ったオブジェクトは、入力ビューや入力アクセサリビューに関連するジオメトリ情報を取得して、それに応じて編集ビューを調整できます。

単純なテキスト入力

挿入位置にテキストを挿入したり、Deleteキーをタップしたときに挿入位置の前にある文字を削除したりできるカスタムビューを実装できます。たとえば、インスタントメッセージアプリケーションに、ユーザが自分の発言を入力するビューを持たせることができます。

単純なテキスト入力のためにこの機能を利用するには、UIViewクラス、またはUIResponderを継承する任意のビュークラスのサブクラスを作成して、UIKeyInputプロトコルを採用します。このビュークラスのインスタンスがファーストレスポンダになると、UIKitはシステムキーボードを表示します。UIKeyInput自身はUITextInputTraitsプロトコルを採用しています。したがって、キーボードタイプ、リターンキータイプ、およびその他のキーボード属性を設定できます。

注： UIKeyInputプロトコルを採用するクラスでは、利用可能なキーボードと言語のごく一部しか利用できません。

UIKeyInputを採用するには、このプロトコルで宣言されているhasText、insertText:、およびdeleteBackwardの3つのメソッドを実装しなければなりません。テキストを実際に描画するには、「テキスト描画とテキスト処理の機能」（85 ページ）にまとめたテクノロジーのいずれかを使用します。ただし、単純なテキスト入力（カスタムコントロール内の単一行テキストなど）の場合は、UIStringDrawing APIとCATextLayer APIが最も適しています。

リスト 7-2に、カスタムビュークラスのUIKeyInputの実装を示します。この例のtextStoreプロパティは、テキストのバッキングストアとしての役割を果たすNSMutableStringオブジェクトです。この実装では、（英数字キーが押されたか、deleteキーが押されたかによって）、文字列に文字が追加されるか、文字列から最後の文字が削除されるかのいずれかが実行されて、textStoreが再描画されます。

リスト 7-2　単純なテキスト入力の実装

```
- (BOOL)hasText {
    if (textStore.length > 0) {
        return YES;
    }
    return NO;
}

- (void)insertText:(NSString *)theText {
    [self.textStore appendString:theText];
    [self setNeedsDisplay];
}

- (void)deleteBackward {
    NSRange theRange = NSMakeRange(self.textStore.length-1, 1);
    [self.textStore deleteCharactersInRange:theRange];
    [self setNeedsDisplay];
}

- (void)drawRect:(CGRect)rect {
    CGRect rectForText = [self rectForTextWithInset:2.0]; // カスタムメソッド
    [self.theColor set];
    UIRectFrame(rect);
    [self.textStore drawInRect:rectForText withFont:self.theFont];
}
```

このコードでは、実際にビューにテキストを描画するために、NSStringに対して、UIStringDrawingカテゴリのdrawInRect:withFont:を使用しています。UIStringDrawingの詳細については、「テキスト描画とテキスト処理の機能」（85 ページ）を参照してください。

テキスト入力システムとのやり取り

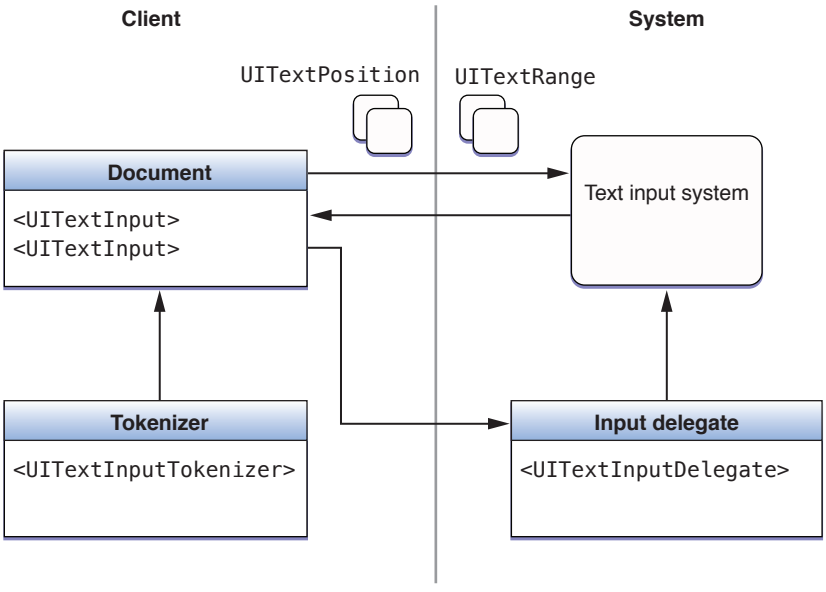
iPhone OSのテキスト入力システムは、キーボードを管理し、タップを特定の言語に適した特定のキーボードの特定のキーが押されたものと解釈して、対応する文字を挿入のためにターゲットビューに送信します。「単純なテキスト入力」（80 ページ）で説明したように、ビュークラスは、カレット（挿入位置）で文字を挿入または削除するために、`UIKeyInput`プロトコルを採用しなければなりません。

しかし、テキスト入力システムは、単純なテキスト入力以上の処理を行います。オートコレクトやマルチステージ入力も管理します。これらの処理は、すべて現在の選択内容とコンテキストに基づいています。マルチステージテキスト入力は、漢字（日本語）、繁体字（中国語）などの表意文字言語を表音文字キーボートから入力するのために必要になります。これらの機能を利用するには、カスタムテキストビューが、`UITextInput`プロトコルを採用して、それに関連するクライアント側クラスおよびプロトコルを実装することによって、テキスト入力システムとやり取りをしなければなりません。

クライアント側のテキスト入力の概要

テキスト文書のクラスでは、テキスト入力システムと十分なやり取りをするために`UITextInput`プロトコルを採用しなければなりません。このクラスは`UIResponder`を継承する必要があります。また、ほとんどの場合、このクラスはカスタムビューです。このクラスには、独自のテキストレイアウトとフォント管理を実装しなければなりません。そのためには、Core Textフレームワークをお勧めします（Core Textの概要については、「テキスト描画とテキスト処理の機能」（85 ページ）で説明します）。また、このクラスは`UITextInputTraits`プロトコルのデフォルトの実装を継承していますが、`UIKeyInput`プロトコルも採用して実装する必要があります。

図 7-1　テキスト入力システムとのコミュニケーション経路



テキスト文書が実装しているUITextInputメソッドは、テキスト入力システムによって呼び出されます。これらのメソッドの多くは、テキスト文書からテキスト位置やテキスト範囲のオブジェクトを要求したり、ほかのメソッド呼び出しで、テキスト位置やテキスト範囲のオブジェクトをテキスト文書に戻したりします。テキスト位置とテキスト範囲をやり取りする理由については、「UITextInputオブジェクトの仕事」（83 ページ）にまとめてあります。

これらのテキスト位置やテキスト範囲のオブジェクトは、ドキュメントに対応して、表示中のテキスト内の位置や範囲を表すカスタムオブジェクトです。これらのオブジェクトの詳細については、「テキスト位置とテキスト範囲」（83 ページ）で説明します。

UITextInputに準拠した文書は、トークナイザと入力デリゲートへの参照も保持しています。この文書は、UITextInputDelegateプロトコルで宣言されているメソッドを呼び出して、システムが提供する入力デリゲートに対してテキストや選択の変更を通知します。また、トークナイザオブジェクトとやり取りをして、テキスト単位の種類（たとえば、文字、単語、段落）を決定します。トークナイザは、UITextInputTokenizerプロトコルを採用したオブジェクトです。

テキスト位置とテキスト範囲

クライアントアプリケーションは、文書のテキスト内の位置と範囲を表すインスタンスの基になる2つのクラスを作成しなければなりません。これらのクラスは、それぞれUITextPositionとUITextRangeのサブクラスでなければなりません。

UITextPosition自身にはインターフェイスは宣言されていませんが、このオブジェクトは、テキスト文書とテキスト入力システムとの間でやり取りされる情報の不可欠な部分です。テキストシステムは、いわゆる整数値や構造体ではなく、テキスト内の位置を表すオブジェクトを要求します。さらに、UITextPositionオブジェクトは、表示されているテキスト内の位置が、そのテキストに対応する内部文字列とずれている場合でも、テキスト位置を表現できるため、実用的な用途にもかないます。このようなケースは、表示されない書式指定文字（RTF文書、HTML文書などを扱う場合）や埋め込みオブジェクト（添付ファイルなど）が文字列に含まれる場合に発生します。独自のUITextPositionクラスでは、表示される文字の文字列オフセット位置を決めるときに、このような表示されない文字を考慮します。最も簡単なケース（埋め込みオブジェクトを含まないプレーンテキスト文書）の場合は、独自のUITextPositionオブジェクトに、1つのオフセット整数をカプセル化します。

UITextRangeには、単純なインターフェイスが1つ宣言されています。このインターフェイスのプロパティのうちの2つは、開始位置と終了位置を表す独自のUITextPositionオブジェクトです。3番目のプロパティは、範囲が空（つまり、長さがない）かどうかを表すブール値を保持します。

UITextInputオブジェクトの仕事

UITextInputプロトコルを採用しているテキスト文書クラスには、このプロトコルのメソッドとプロパティのほとんどを実装しなければなりません。いくつかの例外はありますが、これらのメソッドは、独自のUITextPositionオブジェクトやUITextRangeオブジェクトをパラメータとして取ったり、これらのオブジェクトのいずれかを戻り値として返したりします。実行時に、テキストシステムはこれらのメソッドを呼び出します。また、ほとんどの場合は、何らかのオブジェクトまたは値が返されることを期待します。

UITextInputオブジェクトで実装されるメソッドは、次のような個別の仕事に分類できます。

- **テキスト範囲とテキスト位置の計算。**2つのテキスト位置に基づいてUITextRangeオブジェクト（つまり、テキスト範囲）を作成して返します。または、テキスト位置とオフセットに基づいてUITextPositionオブジェクト（つまり、テキスト位置）を作成して返します。
- **テキスト位置の評価。**2つのテキスト位置を比較したり、あるテキスト位置から別のテキスト位置までのオフセットを返したりします。
- **レイアウト問題への対処。**特定のレイアウト方向に拡張することによって、テキスト位置やテキスト範囲を決定します。
- **ヒットテスト。**指定された点に最も近いテキスト位置またはテキスト範囲を返します。
- **テキスト範囲やテキスト位置に対応する矩形を返す。**テキスト範囲を囲む矩形や、カレットのテキスト位置にある矩形を返します。
- **テキスト範囲を使用してテキストを返したり設定したりする。**

さらに、UITextInputオブジェクトは、（もしあれば）現在選択中のテキストの範囲と、現在マークが付いているテキストの範囲を維持しなければなりません。マークが付いているテキスト（マルチステージテキスト入力の一部）は、ユーザがまだ確定していない暫定的に挿入したテキストを表します。この部分は、はっきり識別できるスタイルで示されます。マークが付いているテキストの範囲には、選択中のテキストの範囲（複数文字の範囲の場合もあれば、カレットの場合もある）が必ず含まれています。

UITextInputオブジェクトには、1つ以上のプロトコルメソッドを任意で実装することもできます。これらを利用して、指定したテキスト位置で始まるテキストスタイル（フォント、テキストの色、背景色）を返したり、表示されているテキストの位置と文字オフセットを一致させたりすることができます（これらの値が同じではないUITextPositionオブジェクトの場合）。

適切なタイミングで、UITextInputオブジェクトは、textWillChange:、textDidChange:、selectionWillChange:、およびselectionDidChange:の各メッセージを入力デリゲートに送信しなければなりません（UITextInputは入力デリゲートへの参照を保持している）。

トークナイザ

トークナイザは、あるテキスト位置が、指定の種類のテキスト単位の内部にあるか、境界にあるかを判断できるオブジェクトです。トークナイザは、指定の種類のテキスト単位の範囲、またはそのテキスト単位の境界のテキスト位置を返します。現在定義されているテキスト単位の種類は、文字、単語、文、段落、行、文書です。UITextGranularity型のenum定数は、これらの種類を表します。テキスト単位の種類は、常にストレージやレイアウトの方向と関連付けて評価されます。

トークナイザは、UITextInputTokenizerプロトコルに準拠したクラスのインスタンスです。UITextInputStringTokenizerクラスは、西ヨーロッパ言語用のキーボードに適したUITextInputTokenizerプロトコルのデフォルトの基本実装を提供します。ざまざまな種類のテキスト単位をまったく新たに解釈するトークナイザが必要な場合は、UITextInputTokenizerを採用して、そのメソッドをすべて実装しなければなりません。行の種類やレイアウトの方向（左、右、上、下）を指定するだけでよい場合は、UITextInputStringTokenizerのサブクラスを作成します。

UITextInputStringTokenizerオブジェクトを初期化するときは、UITextInputオブジェクトを渡さなければなりません。UITextInputオブジェクトの方では、トークナイザプロパティのgetterメソッドで、必要に応じてトークナイザオブジェクトを作成しなければなりません。

テキスト描画とテキスト処理の機能

UIkitフレームワークには、アプリケーションのユーザインターフェイス（`UITextField`、`UILabel`、`UITextView`、および`UIWebView`）にテキスト表示することを主な目的とするクラスがいくつか含まれています。ただし、これらのクラスが提供する機能よりも高い柔軟性を必要とするアプリケーションもあります。つまり、アプリケーションでテキストを描画したり操作したりする場所や方法をより細かく制御したい場合です。このような場合、iPhone OSでは、UIKitだけでなく、Core Text、Core Graphics、およびCore Animationの各フレームワークのプログラムインターフェイスを利用できます。

注： Core TextまたはCore Graphicsを使用してテキストを描画する場合は、テキストが正しい向きで表示されるように（つまり、文字列の境界矩形の左上隅が描画の原点になるように）するために、現在のグラフィックスコンテキストに反転変換を適用しなければなりません。さらに、Core TextとCore Graphicsでテキストを描画するには、テキスト行列が設定されたグラフィックスコンテキストが必要になります。

Core Text

Core Textは、複雑なテキストレイアウトやフォント管理のためのテクノロジーです。このテクノロジーは、テキスト処理に大きく依存するアプリケーション（たとえば、ブックリーダー、ワードプロセッサなど）で使用することを目的にしています。このテクノロジーは、手続き型の(ANSI C) APIを公開するフレームワークとして実装されています。このAPIは、Core FoundationおよびCore GraphicsのAPIとの整合性を持っており、これらのほかのフレームワークと統合されています。たとえば、Core Textでは、多くの入出力パラメータに、Core FoundationやCore Graphicsのオブジェクトを使用しています。さらに、Core Foundationの多くのオブジェクトが、Foundationフレームワーク内の対応するオブジェクトに対して「toll-free bridge」である（互換性を持つ）ため、Core Text関数のパラメータにFoundationオブジェクトを使用することもできます。

独自のテキストレイアウトを実行したい場合を除けば、Core Textを使用するべきではありません。

注： Core Textは、iPhone OS 3.2で初めて導入されましたが、このフレームワークは、Mac OS X v10.5以降のMac OS Xではすでに利用可能になっていました。Core Textの詳細な説明と使用例（ただし、Mac OS Xにおける使用例）については、『*Core Text Programming Guide*』を参照してください。

Core Textには、レイアウトエンジンとフォントテクノロジーの2つの主要な部分があり、それぞれ、固有の不透過型コレクションに基づいています。

Core Textのレイアウト用の不透過型

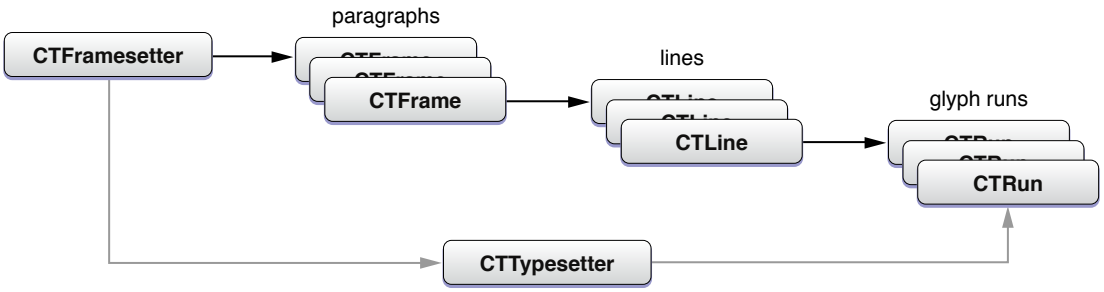
Core Textには、**属性付き文字列**(`CFAttributedStringRef`)と**グラフィックスパス**(`CGPathRef`)の2つのオブジェクトが必要になります。これらの不透過型はCore Textに固有のものではありません。属性付き文字列オブジェクトは、表示されるテキストに対応する内部文字列をカプセル化します。また、その文字列内の文字のスタイル（フォント、色など）を定義するプロパティ（つまり、「属性」）を持っています。グラフィックスパスは、テキストフレーム（段落に相当する）の形状を定義します。

Core Textのオブジェクトは、実行時に、処理対象テキストのレベルを反映した階層を形成します。この階層の最上位には、**framesetter**オブジェクト(`CTFramesetterRef`)がきます。属性付き文字列とグラフィックスパスを入力として、framesetterは1つ以上のテキスト**フレーム**(`CTFrameRef`)を生成します。テキストがフレームに配置されると、framesetterは、そのテキストに段落スタイル（配置、タブストップ、行間、インデント、改行モードなどの属性）を適用します。

フレームを生成するために、framesetterは**typesetter**オブジェクト(`CTTypesetterRef`)を呼び出します。typesetterは、属性付き文字列内の文字をいくつかのグリフに変換して、それらのグリフをテキストフレーム内の行に収めます（グリフとは、1つの文字を表すために使われるグラフィック形状）。フレーム内の行は**CFLine**オブジェクト(`CTLineRef`)で表されます。CTFrameオブジェクトには、CTLineオブジェクトの配列が含まれます。

CTLineオブジェクトには、**グリフの並び**（`CTRunRef`型のオブジェクトで表される）の配列が含まれます。グリフの並びとは、同じ属性と方向を持つ一連の連続するグリフのことです。

図 7-2　Core Textのレイアウトエンジンのアーキテクチャ



CTLine不透過型の関数を使用すると、CTFramesetterオブジェクトを介さずに、属性付き文字列から1行分のテキストを描画できます。それには、テキストの原点をそのテキストのベースラインに設定して、行オブジェクトに自身を描画する要求を出すだけです。

Core Textのフォント用の不透過型

フォントは、Core Textでのテキスト処理には不可欠です。typesetterオブジェクトは、フォントを（属性付きのソース文字列と一緒に）使用して、文字をグリフに変換し、これらのグリフを配置します。グラフィックスコンテキストは、そのコンテキストで発生するすべてのテキスト描画に対する現在のフォントを決定します。Core Textのフォントシステムは、ネイティブにはUnicodeフォントを扱います。

このフォントシステムには、CTFont、CTFontDescriptor、およびCTFontCollectionの3つの不透過型オブジェクトが含まれています。

- フォントオブジェクト(`CTFontRef`)は、ポイントサイズと特定の特性（変換行列による）で初期化されます。このフォントオブジェクトに対して照会を行い、文字からグリフへのマッピング、エンコーディング、グリフデータ、寸法（アセント、レディングなど）などを知ることができます。また、Core Textは、フォントカスケーディングと呼ばれる自動フォント代用メカニズムも提供します。
- フォント記述子オブジェクト(`CTFontDescriptorRef`)は、通常、フォントオブジェクトを作成するために使われます。複雑な変換行列を扱う代わりに、このオブジェクトを利用すると、PostScript名、フォントファミリ、スタイル、特性（太字、イタリックなど）などを含むフォント属性の辞書を指定できます。

- フォントコレクションオブジェクト(`CTFontCollectionRef`)は、フォント記述子のグループであり、フォントの列挙、グローバルまたはカスタムのフォントコレクションへのアクセスなど、さまざまなサービスを提供します。

Core TextとUIKitフレームワーク

Core Textと、UIKitフレームワークのテキストレイアウトおよびレンダリング機能は、互換性がありません。互換性がないということは、次のようなことを意味します。

- Core Textを使用してテキストのレイアウトを計算してから、`UIStringDrawing`などのAPIを使用してテキストを描画することはできません。
- アプリケーションでCore Textを使用する場合は、UIKitのテキスト関連機能（コピーアンドペーストなど）は利用できません。Core Textを使用していてこれらの機能が必要な場合は、自分で実装しなければなりません。

デフォルトでは、UIKitはカーニングを行いません。そのため行が欠落する場合があります。

UIStringDrawingとCATextLayer

`UIStringDrawing`と`CATextLayer`は、単純なテキスト描画に最適なプログラミング機能です。`UIStringDrawing`は、UIKitフレームワークで実装されている`NSString`のカテゴリです。`CATextLayer`は、Core Animationテクノロジーの一部です。

`UIStringDrawing`のメソッドを利用すると、iPhone OSアプリケーションは、特定の位置（単一行のテキストの場合）や、特定の矩形内（複数行の場合）に文字列を描画できます。描画に使われる属性（フォント、改行モード、ベースライン調整など）を渡すこともできます。特定のパラメータ（フォント、改行モード、幅の制限など）を渡すと、描画された文字列のサイズを返すメソッドもあります。これを利用すると、文字列を描画するときに、その文字列の境界矩形を計算できます。

Core Animationの`CATextLayer`クラスは、プレーンな文字列または属性付き文字列をコンテンツとして格納し、そのコンテンツに影響を与える属性セット（フォント、フォントサイズ、テキスト色、切り捨て動作など）を提供します。`CATextLayer`の利点は、（`CALayer`のサブクラスなので）何も手を加えなくてもプロパティをアニメーション化できる点です。Core Animationは、QuartzCoreフレームワークに関連付けられています。

`CATextLayer`のインスタンスは、現在のグラフィックスコンテキストに自身を描画する方法を知っているため、これらのインスタンスを使用するときは、明示的に描画コマンドを発行する必要はありません。

`UIStringDrawing`の詳細については、『*NSString UIKit Additions Reference*』を参照してください。`CATextLayer`、`CALayer`、およびCore Animationのその他のクラスの詳細については、『*Core Animation Programming Guide*』を参照してください。

Core Graphicsのテキスト描画

Core Graphics（つまり、Quartz）は、最も低レベルで2次元の画像を扱うシステムフレームワークです。テキスト描画は、その機能の1つです。一般に、Core Graphicsは非常に低レベルなので、Core Textや、テキスト描画用のその他のシステム機能を使用することをお勧めします。ただし、Core

Graphicsを使用してテキストを描画する方がメリットがある場合もあります。Core Graphicsを使用すると、描画時に使用するフォントをより細かく制御したり、グリフのレンダリングや配置をより精密に行ったりすることができます。

CGContext不透過型の関数を使用して、フォントの選択、テキスト属性の設定、およびテキストの描画を行います。たとえば、`CGContextSelectFont`を呼び出して、使用するフォントを設定します。次に、`CGContextSetFillColor`を呼び出して、テキストの色を設定します。そして、テキスト行列(`CGContextSetTextMatrix`)を設定し、`CGContextShowGlyphsAtPoint`を使用してテキストを描画します。

Core Graphicsのテキスト描画APIの詳細については、『*Quartz 2D Programming Guide*』の「Text」を参照してください。

Foundationレベルの正規表現

Foundationフレームワークの`NSString`クラスには、正規表現用の簡単なプログラムインターフェイスが含まれています。1つの範囲を返す3つのメソッドのいずれかを呼び出して、特定のオプション定数と正規表現の文字列を渡します。一致部分が存在する場合、このメソッドは、その部分文字列の範囲を返します。オプションは、`NSStringCompareOptions`ビットマスク型の`NSRegularExpressionSearch`定数です。この定数は、検索値として、リテラル文字列ではなく正規表現パターンが渡されることをメソッドに知らせます。サポートされる正規表現の構文は、ICU (International Components for Unicode)で定義されている構文です。

注： `NSString`の正規表現機能は、iPhone OS 3.2で導入されました。『ICU User Guide』には、ICUの正規表現の作成方法について説明されています(http://userguide.icu-project.org/strings/regexp)。

正規表現をサポートする`NSString`のメソッドは、次のとおりです。

```
rangeOfString:options:
rangeOfString:options:range:
rangeOfString:options:range:locale:
```

これらのメソッドに`NSRegularExpressionSearch`オプションを指定する場合は、`NSStringCompareOptions`オプション以外に指定できるのは、`NSCaseInsensitiveSearch`と`NSAnchoredSearch`だけです。正規表現検索で一致するものが見つからなかった場合や、正規表現の構文が間違っている場合、これらのメソッドは、`{NSNotFound, 0}`の値を含む`NSRange`構造体を返します。

リスト 7-3に、`NSString`の正規表現APIを使用する例を示します。

リスト 7-3　正規表現を使用した部分文字列の検索

```
// nnn-nnn-nnnn形式の電話番号を検索する
NSRange r;
NSString *regEx = @"[0-9]{3}-[0-9]{3}-[0-9]{4}";
r = [textView.text rangeOfString:regEx options:NSRegularExpressionSearch];
if (r.location != NSNotFound) {
    NSLog(@"Phone number is %@", [textView.text substringWithRange:r]);
} else {
    NSLog(@"Not found.");
}
```

これらのメソッドは、パターンに一致する部分文字列の範囲を1つしか返さないため、ICUライブラリの特定の正規表現機能は利用できません。必要であれば、プログラムで追加しなければなりません。さらに、後方検索、数字検索、ダイアクリティカルマークを区別しない検索などの`NSStringCompareOptions`は利用できません。また、キャプチャグループはサポートされていません。

注： 「ICU正規表現のサポート」（89 ページ）でも説明するように、Phone OS 3.2には正規表現に関連するICUライブラリが含まれています。しかし、`NSString`で代替したのでは十分にニーズを満たせない場合にのみ、ICU機能を使用するべきです。

戻り値として返された範囲をテストする場合は、リテラル文字列に基づく検索と、正規表現パターンに基づく検索の動作の違いを意識しなければなりません。正常にパターンに一致しても、`length`が0の`NSRange`構造体が返される場合もあります（`location`フィールドが目的の場合）。空文字列に正常に一致するパターンもあります（範囲パラメータを持つメソッドに、長さが0の検索範囲を指定した場合）。

ICU正規表現のサポート

システムのBSD（非フレームワーク）レベルには、4.2.1の修正バージョンのライブラリがiPhone OS 3.2に含まれています。ICU (International Components for Unicode)は、Unicodeのサポートとソフトウェアの国際化のためのオープンソースプロジェクトです。インストールされているバージョンのICUには、正規表現をサポートするために必要なヘッダファイルと、それらのインターフェイスに関連するいくつかの修正が含まれています。表 7-1に、これらのファイルの一覧を示します。

表 7-1 iPhone OS 3.2に含まれているICUファイル

`parseerr.h`	`platform.h`	`putil.h`
`uconfig.h`	`udraft.h`	`uintrnal.h`
`uiter.h`	`umachine.h`	`uregex.h`
`urename.h`	`ustring.h`	`utf_old.h`
`utf.h`	`utf16.h`	`utf8.h`
`utypes.h`	`uversion.h`	

ICU 4.2 APIのドキュメントとユーザガイドは、http://icu-project.org/apiref/icu4c/index.htmlを参照してください。

スペルチェックと単語補完

`UITextChecker`クラスのインスタンスを利用すると、文書のスペルチェックを行ったり、一部だけが入力された単語を完成させるための候補を提示したりできます。文書のスペルチェックを行うときは、`UITextChecker`オブジェクトが、指定されたオフセット位置から文書を検索します。間違ったスペルの単語を検出したときに、正しいスペルの候補を、ユーザに提示すべき順番に並べた（つまり、最も有力な単語が先頭になるように）配列を返すこともできます。通常は、ドキュメントご

とに1つのUITextCheckerインスタンスを使用します。ただし、無視する単語やその他の状態を共有したい場合は、1つのインスタンスを使用して、関連する複数のテキストのスペルチェックを行うこともできます。

注： UITextCheckerクラスは、スペルチェックを目的としています。オートコレクトが目的では*ありません*。オートコレクトは、「テキスト入力システムとのやり取り」（82 ページ）で説明したプロトコルを採用してサブクラスを実装することによって利用できる機能です。

間違ったスペルの単語がないか文書をチェックするために使用するメソッドは、rangeOfMisspelledWordInString:range:startingAt:wrap:language:です。正しいスペルの単語候補のリストを取得するために使用するメソッドは、guessesForWordRange:inString:language:です。これらのメソッドを、この順番で呼び出します。文書全体をチェックするには、1つのループ内でこの2つのメソッドを呼び出します。ループの各サイクルで、開始オフセットを、訂正済みの単語の次の文字に設定しなおします。そのコードをリスト 7-4に示します。

リスト 7-4　文書のスペルチェック

```
- (IBAction)spellCheckDocument:(id)sender {
    NSInteger currentOffset = 0;
    NSRange currentRange = NSMakeRange(0, 0);
    NSString *theText = textView.text;
    NSRange stringRange = NSMakeRange(0, theText.length-1);
    NSArray *guesses;
    BOOL done = NO;

   NSString *theLanguage = [[UITextChecker availableLanguages] objectAtIndex:0];
    if (!theLanguage)
        theLanguage = @"en_US";

    while (!done) {
        currentRange = [textChecker rangeOfMisspelledWordInString:theText
range:stringRange
            startingAt:currentOffset wrap:NO language:theLanguage];
        if (currentRange.location == NSNotFound) {
            done = YES;
            continue;
        }
       guesses = [textChecker guessesForWordRange:currentRange inString:theText
            language:theLanguage];
        NSLog(@"-----------------------------------------------");
        NSLog(@"Word misspelled is %@", [theText
substringWithRange:currentRange]);
        NSLog(@"Possible replacements are %@", guesses);
        NSLog(@" ");
        currentOffset = currentOffset + (currentRange.length-1);
    }
}
```

UITextCheckerクラスには、テキストチェッカーに単語を無視したり学習したりするように指示するメソッドが含まれています。リスト 7-4のメソッドのように、間違ったスペルの単語とその訂正候補を単に記録するのではなく、ユーザが正しいスペルを選択したり、テキストチェッカーに単語の無視や学習を指示したり、修正を行わずに次の単語に進んだりできるユーザインタフェースを表示するべきです。それを実現する方法の1つは、Table Viewに正しい単語候補のリストを表示して、「Replace（置換）」、「Learn（学習）」、「Ignore（無視）」などのボタンを含むPopoverビューを使用することです。

また、UITextCheckerを使用して、一部だけが入力された単語の補完候補を取得し、それらをPopover内のTable Viewに表示することもできます。このためには、completionsForPartialWordRange:inString:language:メソッドを呼び出して、チェック対象の特定の文字列の範囲を渡します。このメソッドは、一部だけが入力された単語を補完する単語候補の配列を返します。リスト 7-5に、このメソッドを呼び出して、Popoverビュー内に補完した単語を列挙したTable Viewを表示する方法を示します。

リスト 7-5　現在の不完全な文字列を補完した単語のリスト

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

    self.completionRange = [self computeCompletionRange];
    // UITextCheckerオブジェクトは、インスタンス変数にキャッシュされている
    NSArray *possibleCompletions = [textChecker
completionsForPartialWordRange:self.completionRange
        inString:self.textStore language:@"en"];

    CGSize popOverSize = CGSizeMake(150.0, 400.0);
    completionList = [[CompletionListController alloc]
initWithStyle:UITableViewStylePlain];
    completionList.resultsList = possibleCompletions;
    completionListPopover = [[UIPopoverController alloc]
initWithContentViewController:completionList];
    completionListPopover.popoverContentSize = popOverSize;
    completionListPopover.delegate = self;
    // rectForPartialWordRange: はカスタムメソッド
    CGRect pRect = [self rectForPartialWordRange:self.completionRange];
    [completionListPopover presentPopoverFromRect:pRect inView:self
        permittedArrowDirections:UIPopoverArrowDirectionAny animated:YES];
}
```

カスタム編集メニュー項目

「コピー(Copy)」、「カット(Cut)」、「ペースト(Paste)」、「選択(Select)」、「全選択(Select All)」、「削除(Delete)」の各システムコマンドを表示するために使われる編集メニューにカスタム項目を追加できます。ユーザがこの項目をタップすると、アプリケーション固有の方法で現在のターゲットに影響を与えるコマンドが発行されます。UIKitフレームワークは、ターゲット／アクションメカニズムを利用してこれを実行します。項目をタップすると、アクションメッセージを処理できるレスポンダチェーンの最初のオブジェクトに、このアクションメッセージが送信されます。図 7-3に、カスタムメニュー項目（「Change Color」）の例を示します。

図 7-3　カスタムメニュー項目を含む編集メニュー

UIMenuItemクラスのインスタンスが、カスタムメニュー項目を表します。UIMenuItemオブジェクトには、タイトルとアクションセレクタの2つのプロパティがあります。これらはいつでも変更できます。カスタムメニュー項目を実装するには、UIMenuItemインスタンスをこれらのプロパティで初期化して、そのインスタンスをメニューコントローラのカスタムメニュー項目の配列に追加しなければなりません。そして、そのコマンドを処理するアクションメソッドを適切なレスポンダサブクラスに実装しなければなりません。

カスタムメニュー項目の実装のその他の部分は、UIMenuControllerシングルトンオブジェクトを使用するすべてのコードに共通です。カスタムビューやオーバーライドされたビューで、そのビューをファーストレスポンダに設定し、共有のメニューコントローラを取得してターゲット矩形を設定します。次に、setMenuVisible:animated:を呼び出して編集メニューを表示します。リスト 7-6に示す簡単な例では、カスタムビューの色を赤と黒の間で切り替えるためのカスタムメニュー項目を追加します。

リスト 7-6　「Change Color」メニュー項目の実装

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {}
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {}
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *theTouch = [touches anyObject];
    if ([theTouch tapCount] == 2) {
        [self becomeFirstResponder];
       UIMenuItem *menuItem = [[UIMenuItem alloc] initWithTitle:@"Change Color"
 action:@selector(changeColor:)];
        UIMenuController *menuCont = [UIMenuController sharedMenuController];
        [menuCont setTargetRect:self.frame inView:self.superview];
        menuCont.arrowDirection = UIMenuControllerArrowLeft;
        menuCont.menuItems = [NSArray arrayWithObject:menuItem];
        [menuCont setMenuVisible:YES animated:YES];
    }
}
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {}

- (BOOL)canBecomeFirstResponder { return YES; }

- (void)changeColor:(id)sender {
    if ([self.viewColor isEqual:[UIColor blackColor]]) {
        self.viewColor = [UIColor redColor];
    } else {
        self.viewColor = [UIColor blackColor];
    }
    [self setNeedsDisplay];
}
```

注： リスト 7-6に示したUIMenuControllerのarrowDirectionプロパティは、iPhone OS 3.2で新たに導入されました。これを利用すると、編集メニューに付く矢印がターゲット矩形を指す方向を指定できます。「削除(Delete)」メニューコマンドも新たに追加されました。ユーザがこのメニューコマンドをタップすると、レスポンダチェーン内のオブジェクトで実装されているdelete:メソッド（存在する場合）が呼び出されます。delete:メソッドはUIResponderStandardEditActions非形式プロトコルで宣言されています。

書類の改訂履歴

この表は「*iPad プログラミングガイド*」の改訂履歴です。

日付	メモ
2010-03-24	iPad向けアプリケーションの作成方法について説明する新規文書。

書類の改訂履歴