iPhoneアプリケーションプログラミングガイド

iPhone

2009-06-17

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

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

目次

図、表、リスト

第 1 章 コアアプリケーション 17

第 2 章 ウインドウとビュー 53

第 3 章 イベント処理 85

第 4 章 グラフィックスと描画 107

第 5 章 テキストとWeb 119

第 6 章 ファイルとネットワーク 131

第 7 章 マルチメディアサポート 143

第 8 章 デバイスサポート 165

第 9 章 アプリケーション環境設定 191

はじめに

注： この文書は、以前は『*iPhone OS Programming Guide*』というタイトルでした。

iPhone SDKは、ユーザのホーム(Home)画面にアイコンとして表示される、iPhoneのネイティブアプリケーションを作成するために必要なツールおよびリソースを提供します。Safari上で実行されるWebアプリケーションとは違い、ネイティブアプリケーションは、スタンドアロン実行可能ファイルとして、iPhone OSベースのデバイス上で直接実行されます。ネイティブアプリケーションは、加速度センサー、ロケーションサービス、Multi-Touchインターフェイスなど、iPhoneおよびiPod touchをおもしろくするすべての機能にアクセスできます。また、ローカルのファイルシステムにデータを保存したり、カスタムURLスキームを利用して、インストール済みのほかのアプリケーションと通信することもできます。

iPhone OSでは、UIKitフレームワークを使用してネイティブアプリケーションを開発します。このフレームワークは、基本的にインフラストラクチャとデフォルトの動作を提供します。このデフォルトの動作を利用すると、数分程度で機能的なアプリケーションが作成できます。UIKitフレームワーク（および、システム上のその他のフレームワーク）は、かなりの量のデフォルトの動作を提供していますが、それをカスタマイズしたり拡張するためのフックも提供しています。

対象読者

この文書は、iPhoneのネイティブアプリケーションを開発しようとしている、初心者および経験者の両方のiPhone OSデベロッパを対象としています。この文書の目的は、デベロッパの皆さんにiPhoneアプリケーションのアーキテクチャに慣れてもらい、UIKitおよびその他の重要なシステムフレームワークの主要なカスタマイズポイントを示すことです。その過程で、適切な設計判断をするために役立つガイダンスも示します。また、与えられたテーマについてのアドバイスや、さらに詳しい議論を提供してくれる追加文書も紹介します。

この文書で説明しているフレームワークの多くはMac OS Xにも存在しますが、必ずしもMac OS Xまたはそのテクノロジーに精通していることを前提とはしていません。

お読みになる前に

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

- アプリケーションを開発する上でのXcodeおよびInterface Builderとその役割についての基本情報
- 新規Objective-Cクラスの定義の仕方
- メモリ管理の方法（Objective-Cでオブジェクトを作成し、解放する方法を含む）
- アプリケーション動作を管理する上でのデリゲートオブジェクトの役割
- ユーザインターフェイスを管理する上でのターゲット／アクションパラダイムの役割

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

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

この書類の構成

この文書は、次の各章で構成されています。

- 「コアアプリケーション」（17ページ）には、すべてのiPhoneアプリケーションに共通する基本構成についての重要な情報が含まれています。それには、どのアプリケーションも対応しておく必要がある重要なタスクが含まれています。
- 「ウインドウとビュー」（53ページ）では、iPhoneウインドウモデルについて説明し、ビューを使ってユーザインターフェイスを編成する方法を紹介します。
- 「イベント処理」（85ページ）では、iPhoneイベントモデルについて説明し、Multi-Touchイベントおよびモーションベースのイベントの処理方法を紹介します。また、アプリケーションへコピー操作とペースト操作を組み込む方法を紹介します。

- 「グラフィックスと描画」（107 ページ）では、iPhone OSのグラフィックスアーキテクチャについて説明し、図形や画像の描画方法やコンテンツにアニメーションを組み込む方法を紹介します。
- 「テキストとWeb」（119 ページ）では、iPhone OSでサポートされているテキストについて説明します。ここには、システムキーボードの管理方法の例も含まれています。
- 「ファイルとネットワーク」（131 ページ）では、ファイルとネットワーク接続を扱う際のガイドラインを示します。
- 「マルチメディアサポート」（143 ページ）では、iPhone OSで利用できるオーディオとビデオの各テクノロジーを使用する方法を紹介します。
- 「デバイスサポート」（165 ページ）では、外部アクセサリ、位置追跡サービス、加速度センサー、および内蔵カメラをどのように利用するかについて紹介します。
- 「アプリケーション環境設定」（191 ページ）では、アプリケーション環境設定を構成したり、それを「設定(Settings)」アプリケーションに表示する方法を紹介します。

フィードバックの提供

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

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

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

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

関連項目

次の文書には、iPhone OS向けのアプリケーションを開発する前に、すべてのデベロッパが読んでおく必要がある重要な情報が書かれています。

- 『*iPhone Development Guide*』では、iPhoneの開発プロセスに関する重要な情報をツールの視点から提供しています。この文書は、ソフトウェアをビルド、実行、およびテストするための、デバイスの構成とXcode（およびその他のツール）の使用についても言及しています。
- 『*Cocoa Fundamentals Guide*』では、iPhoneアプリケーションの開発に使用するデザインパターンとデザインプラクティスに関する基本的な情報を提供しています。
- 『*iPhone Human Interface Guidelines*』では、iPhoneアプリケーションのユーザインターフェイスの設計方法についてのガイダンスと重要な情報を提供しています。

次のリファレンスおよび解説書は、iPhoneの重要なトピックスについての補足情報を提供しています。

- 『*UIKit Framework Reference*』および『*Foundation Framework Reference*』では、この文書で説明している各クラスに関するリファレンス情報を提供しています。
- 『*View Controller Programming Guide for iPhone OS*』では、iPhoneアプリケーションのインターフェイス作成のためのView Controllerの使用について情報を提供しています。
- 『*Table View Programming Guide for iPhone OS*』では、iPhoneアプリケーションで頻繁に使われるTable Viewを扱うための情報を提供しています。
- 『*The Objective-C 2.0 Programming Language*』では、Objective-Cと、iPhone OSのほとんどの動的な動作と拡張性の基礎となるObjective-Cランタイムシステムについて解説しています。

コアアプリケーション

すべてのiPhoneアプリケーションは、UIKitフレームワークを使用して作成されます。したがって、本質的に同じコアアーキテクチャを持っています。UIKitは、アプリケーションを実行したり、ユーザ入力の処理や画面へのコンテンツの表示を調整するために必要な、主要なオブジェクトを提供します。アプリケーションごとの相違点は、これらのデフォルトのオブジェクトをどのように構成するかと、アプリケーションのユーザインターフェイスや動作を追加するために、どこにカスタムオブジェクトを組み込むかです。

アプリケーションのユーザインターフェイスや基本動作へのカスタマイズは、アプリケーションのカスタムコード内で行われますが、アプリケーションの最上位レベルで行わなければならないカスタマイズもたくさんあります。これらのアプリケーションレベルのカスタマイズは、アプリケーションがシステムやデバイスにインストール済みのその他のアプリケーションとやり取りする方法に影響を与えるため、カスタマイズが必要な場合とデフォルトの動作で十分な場合を理解することが重要です。この章では、コアアプリケーションアーキテクチャの概要を説明し、カスタマイズが必要な場合とデフォルトの動作を使用する場合の判断をするために役立つ上位レベルのカスタマイズポイントを示します。

コアアプリケーションアーキテクチャ

ユーザがアプリケーションを起動したときから終了するまで、UIKitフレームワークは、アプリケーションの主要なインフラストラクチャの大部分を管理します。iPhoneアプリケーションは絶えずシステムからイベントを受信し、それらのイベントに応答しなければなりません。イベントの受信は`UIApplication`オブジェクトの仕事ですが、イベントへの応答はカスタムコードで処理する必要があります。イベントに応答する必要がある場所について理解するには、iPhoneアプリケーションのライフサイクル全体とイベントサイクルを少し理解しておくと役立ちます。以降のセクションでは、これらのサイクルについて説明し、iPhoneアプリケーションの開発全体を通して使われるいくつかの主要なデザインパターンの概要を示します。

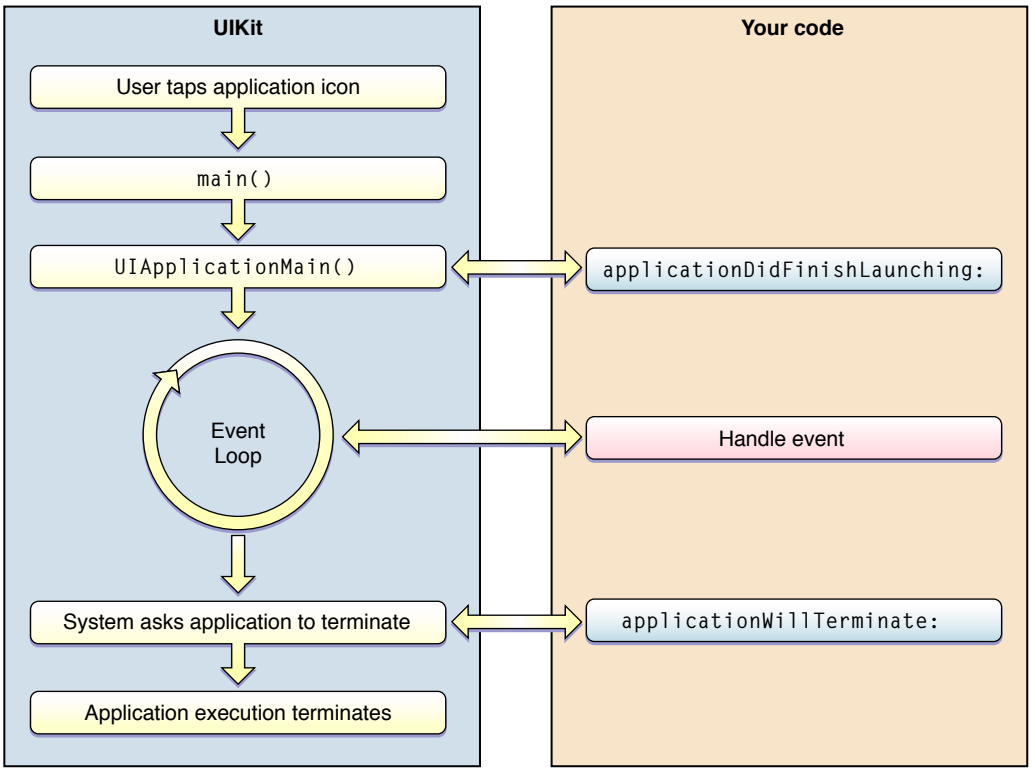
アプリケーションのライフサイクル

アプリケーションのライフサイクルは、アプリケーションの起動から終了までの間に発生する一連のイベントで構成されます。iPhone OSでは、ユーザはホーム(Home)画面上のアイコンをタップすることによってアプリケーションを起動します。タップが発生するとすぐに、システムはトランジショングラフィックスを表示し、そのアプリケーションの`main`を呼び出してアプリケーションの起動へと進みます。以降、初期化作業のほとんどはUIKitに渡されます。UIKitはアプリケーションのユーザインターフェイスをロードし、イベントループを準備します。イベントループの中では、UIKitはカスタムオブジェクトへのイベントの配信と、アプリケーションから発行されたコマンドへの応答を調整します。ユーザがアプリケーションを終了させるためのアクションを実行すると、UIKitはアプリケーションにそれを通知し、終了処理を開始します。

図1-1は、iPhoneアプリケーションのライフサイクルを単純化して示したものです。この図には、アプリケーションの起動時から終了時までに発生する一連のイベントが示されています。初期化時と終了時に、UIKitはそれを知らせるために特定のメッセージをアプリケーションデリゲートオブジェ

クトに送信します。イベントループの中で、UIKitは、アプリケーションのカスタムイベントハンドラにイベントを送信します。初期化イベントと終了イベントの処理については、後述の「初期化と終了」（34 ページ）で説明します。また、イベント処理プロセスについては「イベント処理サイクル」（20 ページ）で概要を、それ以降の各章で詳細を説明します。

図 1-1 アプリケーションのライフサイクル



Main関数

iPhoneアプリケーションでは、`main`関数の使用は極力抑えられます。代わりに、アプリケーションの実行に実際に必要な作業のほとんどは、`UIApplicationMain`関数によって処理されます。その結果、Xcodeで新規アプリケーションプロジェクトを開始すると、どのプロジェクトテンプレートにも、リスト 1-1に示すような標準的な`main`関数があります。`main`ルーチンが実行するのは、自動解放プールを作成して、`UIApplicationMain`を呼び出し、自動解放プールを解放するという3つの処理だけです。いくつかの例外はありますが、この関数の実装を変更する必要はありません。

リスト 1-1 iPhoneアプリケーションの`main`関数

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

int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

注： 自動解放プールは、メモリ管理に使用されます。このプールは、コードの機能ブロックの間に作成されるオブジェクトの解放を遅らせるために使用されるCocoaのメカニズムです。自動解放プールの詳細については、『*Memory Management Programming Guide for Cocoa*』を参照してください。iPhoneアプリケーションの自動解放プールに関連する特有のメモリ管理ガイドラインについては、「メモリを賢く割り当てる」（48 ページ）を参照してください。

前述のリストの`UIApplicationMain`関数は、4つのパラメータを取り、それらを使用してアプリケーションの初期化を行います。この関数に渡されるデフォルト値を変更する必要はありませんが、アプリケーションの起動という点からその目的を説明しておく価値はあります。`main`に渡される`argc`パラメータと`argv`パラメータのほかに、この関数は、主クラス（アプリケーションオブジェクトのクラス）とアプリケーションデリゲートのクラスを識別する2つの文字列パラメータを取ります。主クラスの文字列の値が`nil`の場合、UIKitはデフォルトで`UIApplication`クラスを使用します。アプリケーションデリゲートのクラスの値が`nil`の場合、UIKitは、アプリケーションデリゲートが、アプリケーションのメインnibファイルからロードされるオブジェクトの1つであることを前提にします（Xcodeのテンプレートを使用して作成されたアプリケーションの場合）。これらのパラメータのいずれかに`nil`以外の値が設定されている場合、`UIApplicationMain`関数は、アプリケーションの起動中に、それに対応するクラスのインスタンスを作成して、先に示した目的のためにそれを使用します。したがって、アプリケーションが`UIApplication`のカスタムサブクラスを使用する場合は（このような方法はお勧めしませんが、可能です）、このカスタムクラスの名前を3番目のパラメータで指定します。

アプリケーションデリゲート

アプリケーションの上位レベルの動作を監視するのは、デベロッパが提供するカスタムオブジェクトであるアプリケーションデリゲートオブジェクトの責任です。デリゲーションは、デフォルトの`UIApplication`オブジェクトなどの複雑なUIKitオブジェクトのサブクラス化を避けるために使用されるメカニズムです。サブクラス化してメソッドをオーバライドする代わりに、複雑なオブジェクトは変更せずに使用して、デリゲートオブジェクトの内部にカスタムコードを配置します。関心のあるイベントが発生すると、複雑なオブジェクトはデリゲートオブジェクトにメッセージを送ります。デベロッパはこれらの「フック」を使用して、カスタムコードを実行して必要な動作を実装できます。

重要： デリゲートデザインパターンは、アプリケーション開発の時間と労力を節約することを目的としています。したがって、理解しておくべき重要なパターンです。iPhoneアプリケーションで使用される主要なデザインパターンの概要については、「基礎となるデザインパターン」（22 ページ）を参照してください。デリゲーションおよびその他のUIKit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。

アプリケーションデリゲートオブジェクトは、いくつかの重要なシステムメッセージの処理を担当します。このため、すべてのiPhoneアプリケーションに存在*しなければなりません*。このオブジェクトは、`UIApplicationDelegate`プロトコルを採用してさえいれば、どのようなクラスのインスタンスでもかまいません。このプロトコルのメソッドは、アプリケーションライフサイクルへのフックを定義しているため、カスタム動作を実装する手段となります。これらのメソッドのすべてを実装する必要はありませんが、「重要なアプリケーションタスクの処理」（34 ページ）に記述されているメソッドは、すべてのアプリケーションデリゲートが実装しなければなりません。

`UIApplicationDelegate`プロトコルのメソッドの詳細については、『*UIApplicationDelegate Protocol Reference*』を参照してください。

メインnibファイル

初期化時に発生するもう1つのタスクは、アプリケーションのメインnibファイルのロードです。アプリケーションの情報プロパティリスト(`Info.plist`)ファイルに`NSMainNibFile`キーが含まれている場合、`UIApplication`オブジェクトは初期化処理の一部としてそのキーで指定されたnibファイルをロードします。メインnibファイルは自動的にロードされる唯一のnibファイルです。ただし、必要に応じて追加のnibファイルをロードすることができます。

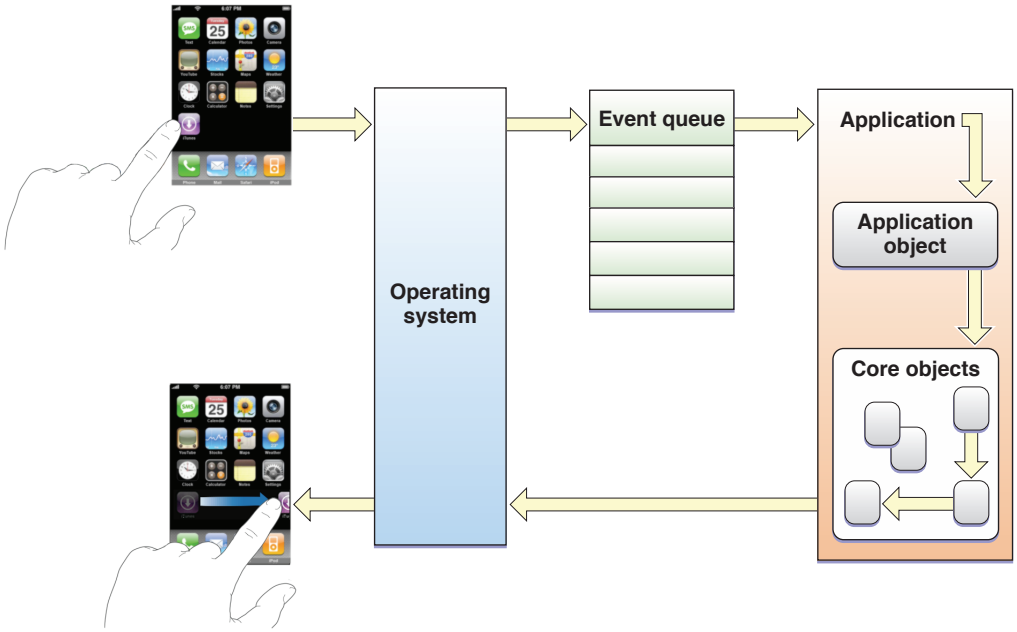
nibファイルは、ディスクベースのリソースファイルで、1つ以上のオブジェクトのスナップショットを保存しています。iPhoneアプリケーションのメインnibファイルには、通常、ウインドウオブジェクト、アプリケーションデリゲートオブジェクト、およびウインドウを管理するためのその他の重要なオブジェクトがおそらく1つ以上含まれています。nibファイルをロードすると、そのnibファイル内のオブジェクトが再構成されます。つまり、各オブジェクトは、ディスク上の表現から、アプリケーションによる操作が可能な実際のメモリ内バージョンに変換されます。nibファイルからロードされたオブジェクトは、プログラムで作成したオブジェクトとまったく違いはありません。ただし、ユーザインターフェイスに関しては、（Interface Builderアプリケーションを使用して）ユーザインターフェイスに関連付けられたオブジェクトをグラフィカルに作成して、それをnibファイルに保存する方が、プログラムでオブジェクトを作成するよりも通常は便利です。

nibファイルおよびiPhoneアプリケーションでのその使用の詳細については、「nibファイル」（34 ページ）を参照してください。アプリケーションのメインnibファイルを指定する方法の詳細については、「情報プロパティリスト」（28 ページ）を参照してください。

イベント処理サイクル

`UIApplicationMain`関数は、アプリケーションの初期化が終わると、アプリケーションのイベントおよび描画サイクルを管理するために必要なインフラストラクチャを起動します。その様子を図1-2に示します。ユーザがデバイスとやり取りすると、iPhone OSはタッチイベントを検出し、それをアプリケーションのイベントキューに配置します。`UIApplication`オブジェクトのイベント処理インフラストラクチャは、このキューの先頭からイベントを取り出して、それを処理するのに最も適したオブジェクトに送付します。たとえば、ボタンで発生したタッチイベントは、それに対応するボタンオブジェクトに送付されます。イベントが、コントロールオブジェクトや、アプリケーション内でタッチイベントの処理を間接的に担当するその他のオブジェクトに送付される場合もあります。

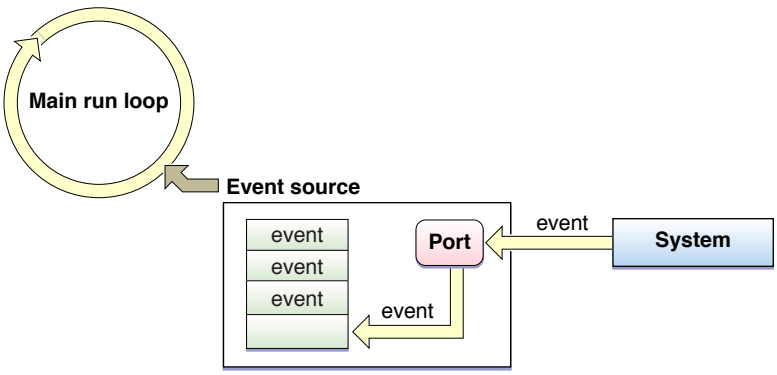
図 1-2 イベントと描画サイクル



iPhone OSのMulti-Touchイベントモデルでは、タッチデータは1つのイベントオブジェクト(`UIEvent`)にカプセル化されます。個々のタッチを追跡するために、このイベントオブジェクトには、画面にタッチしている指ごとに1つずつ，タッチオブジェクト(`UITouch`)が含まれています。ユーザが画面上に指を置いて移動してから、最後に画面から指を離すと、システムは、各指の変化をそれに対応するタッチオブジェクト内で報告します。

システムは、アプリケーションを起動するときに、そのアプリケーション用に1つのプロセスと1つのスレッドの両方を作成します。最初のスレッドは、アプリケーションのメインスレッドになり、そこに、`UIApplication`オブジェクトが**メイン実行ループ**をセットアップして、アプリケーションのイベント処理コードを設定します。図 1-3は、イベント処理コードとメイン実行ループの関係を表しています。システムによって送信されたタッチイベントは、アプリケーションのメイン実行ループで処理できるようになるまで順番待ちをします。

図 1-3 メイン実行ループでのイベント処理



注： 実行ループは、与えられた実行スレッドに対する入力ソースを監視します。入力ソースに処理すべきデータがあると、実行ループはそのスレッドをウェイクアップして、その入力ソース用のハンドラに制御を渡します。ハンドラが終了すると、実行ループに制御が戻ります。実行ループは次のイベントを処理するか、これ以上何もすることがなければそのスレッドをスリープさせます。FoundationフレームワークのNSRunLoopクラスを使用して、メインループにポートやタイマーなどの独自の入力ソースをインストールすることもできます。NSRunLoopおよび実行ループの詳細については、『*Threading Programming Guide*』を参照してください。

UIApplicationオブジェクトは、1つの入力ソースを持つメイン実行ループを構成します。この実行ループは、タッチイベントを適切なレスポンダオブジェクトに送付することによって、処理します。レスポンダオブジェクトは、UIResponderクラスから派生したオブジェクトで、1つのタッチイベントのさまざまなフェーズを処理するために、1つ以上のメソッドを実装しています。アプリケーション内のレスポンダオブジェクトには、UIApplication、UIWindow、UIView、およびすべてのUIViewサブクラスのインスタンスが含まれます。アプリケーションは、通常、アプリケーションのメインウインドウを表すUIWindowオブジェクトにイベントを送付します。次に、このウインドウオブジェクトは、そのイベントを**ファーストレスポンダ**に転送します。通常、ファーストレスポンダは、タッチが発生した場所のビューオブジェクト(UIView)です。

イベントを処理するために使用するメソッドを定義しているだけでなく、UIResponderクラスは、**レスポンダチェーン**のプログラム構造も定義しています。レスポンダチェーンは、協調的なイベント処理のためのCocoaメカニズムです。レスポンダチェーンは、1つのアプリケーション内のレスポンダオブジェクトの連鎖で、通常は、ファーストレスポンダから始まります。ファーストレスポンダオブジェクトは、イベントを処理できない場合、そのイベントをチェーン内の次のレスポンダに渡します。メッセージは、イベントが処理されるまで、ウインドウ、アプリケーション、アプリケーションのデリゲートなど、上位レベルのレスポンダオブジェクトに向かってチェーン内で渡され続けます。イベントが処理されなかった場合、そのイベントは破棄されます。

イベントを処理するレスポンダオブジェクトは、一連のプログラムアクションを起動する役割をします。その結果、アプリケーションは、ユーザインターフェイスの全体または一部を再描画します（サウンドの再生など、その他の結果が生じる場合もあります）。たとえば、コントロールオブジェクト（UIControlのサブクラス）は、別のオブジェクト（通常は、現在のアクティブビューセットを管理するコントローラ）にアクションメッセージを送信することによってイベントを処理します。アクションメッセージを処理している間に、コントローラは、ユーザインターフェイスを変更したり、いくつかのビューを再描画するためにビューの位置を調整します。これが実行されると、ビューおよびグラフィックスのインフラストラクチャは、それを引き継ぎ、必要な再描画イベントを最も効率的な方法で処理します。

イベント、レスポンダ、およびカスタムオブジェクト内でイベントを処理する方法の詳細については、「イベント処理」（85 ページ）を参照してください。ウインドウとビューが、どのようにしてイベント処理スキームに適応するかの詳細については、「ビューの対話モデル」（58 ページ）を参照してください。グラフィックスのインフラストラクチャとビューの更新方法の詳細については、「ビューの描画サイクル」（107 ページ）を参照してください。

基礎となるデザインパターン

UIKitフレームワークの設計には、Mac OS XのCocoaアプリケーションに見られるデザインパターンが数多く組み込まれています。これらのデザインパターンを理解することは、iPhoneアプリケーションを作成するために非常に重要です。したがって、少し時間を取ってこれらについて学ぶだけの価値はあります。以下に、これらのデザインパターンの概要を示します。

表 1-1 iPhoneアプリケーションで使われるデザインパターン

デザインパターン	説明
Model-View-Controller	**Model-View-Controller (MVC)**デザインパターンは、コードを独立した機能領域に分割する方法の1つです。**Model**部分は、アプリケーションのベースとなるデータエンジンを定義し、そのデータの完全性を維持する役割を果たします。**View**部分は、アプリケーションのユーザインターフェイスを定義し、そのインターフェイスに表示されるデータの出所についての明確な知識を持っていません。**Controller**部分は、モデルとビューの橋渡しとしての役割を果たし、これらの間の更新を容易にします。
デリゲーション	**デリゲーション**デザインパターンは、複雑なオブジェクトをサブクラス化せずに変更する方法の1つです。サブクラス化する代わりに、複雑なオブジェクトをそのまま使用し、そのオブジェクトの動作を変更するためのカスタムコードを別のオブジェクト（これを、デリゲートオブジェクトと呼ぶ）の内部に配置します。あらかじめ定義されたタイミングで、この複雑なオブジェクトは、デリゲートオブジェクトのメソッドを呼び出して、カスタムコードを実行する機会を提供します。
ターゲット／アクション	コントロールは、**ターゲット／アクション**デザインパターンを使用して、ユーザとの対話をアプリケーションに通知します。ユーザがあらかじ決められた方法で（ボタンをタップするなど）コントロールと対話すると、コントロールは、指定されたオブジェクト（ターゲット）にメッセージを送信（アクション）します。ターゲットオブジェクトは、アクションメッセージを受信するとすぐに、適切な方法で応答します（ボタンの押下に反応してアプリケーションの状態を更新するなど）。
マネージドメモリモデル	Objective-C言語は、オブジェクトをメモリから解放するタイミングを決定するために、参照カウントの仕組みを使用します。オブジェクトが初めて作成されたときに、そのオブジェクトには参照カウント1が与えられます。ほかのオブジェクトは、そのオブジェクトの`retain`、`release`、または`autorelease`の各メソッドを使用して、参照カウントを適切に増減させます。オブジェクトの参照カウントが0になると、Objective-Cランタイムはオブジェクトのクリーンアップルーチンを呼び出し、次にメモリ割り当てを解除します。

これらの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。

アプリケーションのランタイム環境

iPhone OSのランタイム環境は、プログラムを高速かつ安全に実行できるように設計されています。以降の各セクションでは、ランタイム環境の主要な側面について説明し、その環境の中で最善の動作をさせるためのガイダンスを示します。

すばやい起動、短時間の使用

iPhone OSベースのデバイスの強みは、即時性です。一般的なユーザは、ポケットや鞄からデバイスを取り出して数秒間（または数分間）使用したら、再びそれをしまいます。その間にユーザは、電話を受けたり、連絡先を検索したり、現在聴いている曲を変えたり、いくつかの情報を取得したりします。

iPhone OSでは、一度に実行するフォアグラウンドアプリケーションは1つだけです。つまり、ユーザがホーム(Home)画面上でアプリケーションのアイコンをタップするたびに、アプリケーションが起動し、遅延が最小になるようにすばやく初期化されなければなりません。アプリケーションの起動に長時間かかると、ユーザはそのアプリケーションを使う気がなくなってしまうかもしれません。

アプリケーションは、起動がすばやくできるだけでなく、終了もすばやくできるようにしておかなければなりません。ユーザが、ホーム(Home)ボタンを押すか、または別のアプリケーションのコンテンツを開く機能を使用してアプリケーションのコンテキストを離れる場合は、iPhone OSがアプリケーションに終了を通知します。その時点で、未保存の変更をディスクに保存し、できるだけすばやく終了する必要があります。アプリケーションの終了に5秒以上かかると、システムが即座にアプリケーションを強制終了します。

ユーザが別のアプリケーションに切り替えたときに、アプリケーションがバックグラウンドで実行されていないにもかかわらず、あたかも実行されているかのように見せることが奨励されます。アプリケーションが終了した場合は、未保存のデータだけでなく、アプリケーションの現在の状態に関する情報も保存する必要があります。起動時には、この状態情報を検索し、それを使用してアプリケーションを最後に使用したときの状態に復元する必要があります。これによって、ユーザがそのアプリケーションを最後に使用したときの状態にすぐに戻ることができるので、より一貫性のあるユーザ体験を提供できます。このようにユーザの場所を保存することによって、アプリケーションを起動するたびに、複数の画面に相当する情報を通過して元の場所に戻る必要がなくなるので、時間の節約にもなります。

アプリケーションサンドボックス

セキュリティ上の理由から、iPhone OSは、アプリケーション（アプリケーションの環境設定とデータも含む）をファイルシステム内の一意の場所に配置するよう制限しています。この制限は、アプリケーションの「サンドボックス」と呼ばれるセキュリティ機能の一部です。**サンドボックス**とは、ファイル、環境設定、ネットワークリソース、ハードウェアなどに対するアプリケーションのアクセスをきめ細かく制限する一連のコントロールです。iPhone OSでは、アプリケーションとそのデータは、ほかのアプリケーションからはアクセスできないセキュアな場所に置かれています。アプリケーションがインストールされると、そのアプリケーションを表す、一意で不透過な識別子をシステムが算出します。システムは、ルートアプリケーションディレクトリとこの識別子を使用して、アプリケーションのホームディレクトリのパスを作成します。このため、アプリケーションのホームディレクトリは次の構造を持つものとして表せます。

/*ApplicationRoot*/*ApplicationID*/

インストール処理中に、システムはアプリケーションのホームディレクトリといくつかの重要なサブディレクトリを作成し、アプリケーションのサンドボックスを設定します。そして、そのホームディレクトリにアプリケーションバンドルをコピーします。アプリケーションとそのデータごとに一意の場所を使用することで、バックアップと復元の操作、アプリケーションの更新、およびアンインストールが簡単になります。アプリケーションごとに作成されるアプリケーション固有のディレクトリと、アプリケーションの更新およびバックアップと復元の操作の詳細については、「ファイルとデータの管理」（131 ページ）を参照してください。

重要： サンドボックスによって、攻撃者がほかのアプリケーションやシステムに与える被害を抑えることができます。ただし、サンドボックスでは、攻撃そのものを防止することはできません。つまり、サンドボックスは、悪意のある者による直接攻撃からアプリケーションを保護するものではありません。たとえば、入力処理コードの中に悪用可能なバッファオーバーフローが存在してユーザ入力の検証ができない場合、攻撃者は、プログラムをクラッシュさせたり、それを利用して攻撃者のコードを実行できる可能性があります。

仮想メモリシステム

プログラムのメモリを管理するために、iPhone OSではMac OS Xと本質的に同じ仮想メモリシステムを使用します。iPhone OSでは、プログラムごとに固有の仮想アドレス空間を持ちますが、（Mac OS Xとは違って）使用可能な仮想メモリは、使用可能な物理メモリの量によって制限されます。これは、iPhone OSがメモリを使い切ると揮発性のページをディスクに書き込まないからです。その代わりに、仮想メモリシステムは、必要に応じて不揮発性メモリを解放してアプリケーションの実行に必要なメモリ容量を確保します。そのために、現在使われていないメモリページや、コードページなどの読み取り専用コンテンツを含むメモリページを削除します。このようなページは、後で再び必要なときにいつでもメモリにロードしなおせます。

メモリ不足の状態が続くと、システムは実行中のアプリケーションに通知を送って、メモリをさらに解放するように依頼します。すべてのアプリケーションは、この通知に対応して、メモリ不足を解消するためにそれぞれの責務を果たさなければなりません。アプリケーションでこのような通知を処理する方法については、「メモリ不足警告を監視する」（37 ページ）を参照してください。

自動スリープタイマー

iPhone OSが消費電力を節約する方法の1つに、自動スリープタイマーがあります。システムは、長時間タッチイベントを検出しないと、まず画面を暗くし、やがては電源を完全にオフにします。ほとんどのデベロッパは、このタイマーをオンのままにしておくべきです。ただし、ゲームやタッチ入力を使用しないアプリケーションを開発する際は、このタイマーを無効にして、アプリケーションの実行中は画面が暗くならないようにできます。このタイマーを無効にするには、共有の`UIApplication`オブジェクトの`idleTimerDisabled`プロパティを`YES`に設定します。

その結果、電力消費が増加するので、スリープタイマーを無効にするのは極力避けるべきです。スリープタイマーの使用を検討しなければならないのは、マッピングアプリケーション、ゲーム、およびタッチ入力に依存せずにデバイスの画面にビジュアルコンテンツを表示する必要があるアプリケーションだけです。オーディオアプリケーションでは、このタイマーを無効にする必要はありません。オーディオコンテンツは、画面が暗くなった後も再生され続けます。このタイマーを本当に無効にする場合は、できるだけ早く有効に戻して、システムが電力消費を抑えられるようにしてください。アプリケーションで電力を節約する方法の詳細については、「電力消費を抑える」（49ページ）を参照してください。

アプリケーションバンドル

iPhoneアプリケーションをビルドすると、Xcodeはアプリケーションをバンドルとしてパッケージ化します。**バンドル**とは、関連のあるリソースを1つの場所に集めた、ファイルシステム上のディレクトリです。iPhoneアプリケーションバンドルには、アプリケーションの実行可能ファイル、およびアプリケーションが使用するリソース（たとえば、アプリケーションアイコン、その他の画像、

ローカライズされたコンテンツ）が格納されています。表1-2に、標準的なiPhoneアプリケーションバンドルの内容を示します。この例では、MyAppという名前のアプリケーションの場合を示しますが、説明のみを目的としたものです。この表に示すファイルの一部は、実際のアプリケーションバンドルには含まれない場合があります。

表 1-2　標準的なアプリケーションバンドル

ファイル	説明
MyApp	アプリケーションのコードが格納された実行可能ファイル。このファイルの名前は、アプリケーションの名前から.app拡張子を取り除いたものです。このファイルは必須です。
Settings.bundle	Settingsバンドルは、アプリケーションの環境設定を「設定(Settings)」アプリケーションに追加するために使用するファイルパッケージです。このバンドルには、プロパティリスト、および環境設定を構成、表示するその他のリソースファイルが格納されます。詳細については、「アプリケーション環境設定の表示」（43 ページ）を参照してください。
Icon.png	デバイスのホーム(Home)画面でアプリケーションを示すために使用される、57×57ピクセルのアイコン。このアイコンに、光沢のエフェクトを含めてはなりません。光沢のエフェクトはシステムが自動的に付加します。このファイルは必須です。この画像ファイルの詳細については「アプリケーションアイコンと起動画像」（33 ページ）を参照してください。
Icon-Settings.png	「設定(Settings)」アプリケーション内でアプリケーションを示すために使用される、29×29ピクセルのアイコン。アプリケーションにSettingsバンドルが含まれる場合、このアイコンは「設定(Settings)」アプリケーションではアプリケーション名の横に表示されます。このアイコンファイルを指定しない場合は、Icon.pngファイルが拡大縮小されて代わりに使用されます。この画像ファイルの詳細については、「アプリケーション環境設定の表示」（43 ページ）を参照してください。
MainWindow.nib	アプリケーション起動時にロードされるデフォルトのインターフェイスオブジェクトが格納されているアプリケーションのメインnibファイル。通常、このnibファイルにはアプリケーションのメインウインドウオブジェクトと、アプリケーションデリゲートオブジェクトのインスタンスが格納されます。その他のインターフェイスオブジェクトは、追加のnibファイルからロードされるか、プログラミングによってアプリケーションが作成します（メインnibファイルの名前は、Info.plistファイルのNSMainNibFileキーに別の値を割り当てることにより変更できます。詳細については、「情報プロパティリスト」（28 ページ）を参照してください）。
Default.png	アプリケーション起動時に表示される、480×320ピクセルの画像。システムは、アプリケーションがウインドウとユーザインターフェイスをロードするまでの間、このファイルを一時的な背景として使用します。この画像ファイルの詳細については「アプリケーションアイコンと起動画像」（33 ページ）を参照してください。

ファイル	説明
iTunesArtwork	この512×512のアプリケーション用アイコンは、アドホック配布を使用して配布されます。このアイコンは、通常App Storeによって提供されます。ただしアドホック形式で配布されるアプリケーションはApp Storeを経由しないため、アプリケーションバンドルで提供する必要があります。iTunesは、このアイコンを使用してアプリケーションを表示します（このプロパティに指定したファイルは、App Storeに投稿したものと同じでなければなりません（通常は、JPEGまたはPNGファイル）。また、そのようにしてアプリケーションを配布するのはデベロッパの責任です。ファイル名は、左側に表示されるものだけにし、ファイル名の拡張子を含めてはいけません）。
Info.plist	このファイルは、バンドルID、バージョン番号、表示名など、アプリケーションの主要な値を定義するプロパティリストで、情報プロパティリストとも呼ばれます。詳細については、「情報プロパティリスト」（28 ページ）を参照してください。このファイルは必須です。
sun.png（またはその他のリソースファイル）	非ローカライズリソースは、バンドルディレクトリの最上位に置かれます（sun.pngは、この例における非ローカライズ画像ファイルを表します）。アプリケーションは、ユーザが選択した言語設定に関係なく、非ローカライズリソースを使用します。
en.lproj fr.lproj es.lproj その他の言語固有のプロジェクトディレクトリ	ローカライズされたリソースは、ISO 639-1で規定された言語略称に.lprojという接尾辞を加えた名前のサブディレクトリに置かれます（たとえば、en.lprojディレクトリ、fr.lprojディレクトリ、es.lprojディレクトリにはそれぞれ、英語、フランス語、スペイン語のローカライズされたリソースが格納されます）。詳細については、「アプリケーションの国際化」（44 ページ）を参照してください。

iPhoneアプリケーションは、国際化されている必要があり、サポート対象の言語ごとに*language*.lprojフォルダを持っています。アプリケーションのカスタムリソースのローカライズバージョンを提供するだけでなく、アプリケーションアイコン(Icon.png)、デフォルト画像(Default.png)、設定アイコン(Icon-Settings.png)も、そのファイルを言語固有のプロジェクトディレクトリに配置することによってローカライズできます。ただし、ローカライズバージョンを提供した場合でも、これらのファイルのデフォルトバージョンは、常にアプリケーションバンドルの最上位レベルに含まれます。デフォルトバージョンは、特定のローカライズが利用できない場合に使用されます。

NSBundleクラスのメソッド、またはCFBundleRef不透過型の関数を使用して、アプリケーションバンドルに格納されている、画像とサウンドの、ローカライズされたリソースと非ローカライズリソースへのパスを取得します。たとえば、画像ファイルsun.png（「割り込みへの応答」（35 ページ）を参照）へのパスを取得し、それをもとに画像ファイルを作成するには、次に示す2行のObjective-Cコードが必要です。

```
NSString* imagePath = [[NSBundle mainBundle] pathForResource:@"sun"
ofType:@"png"];
UIImage* sunImage = [[UIImage alloc] initWithContentsOfFile:imagePath];
```

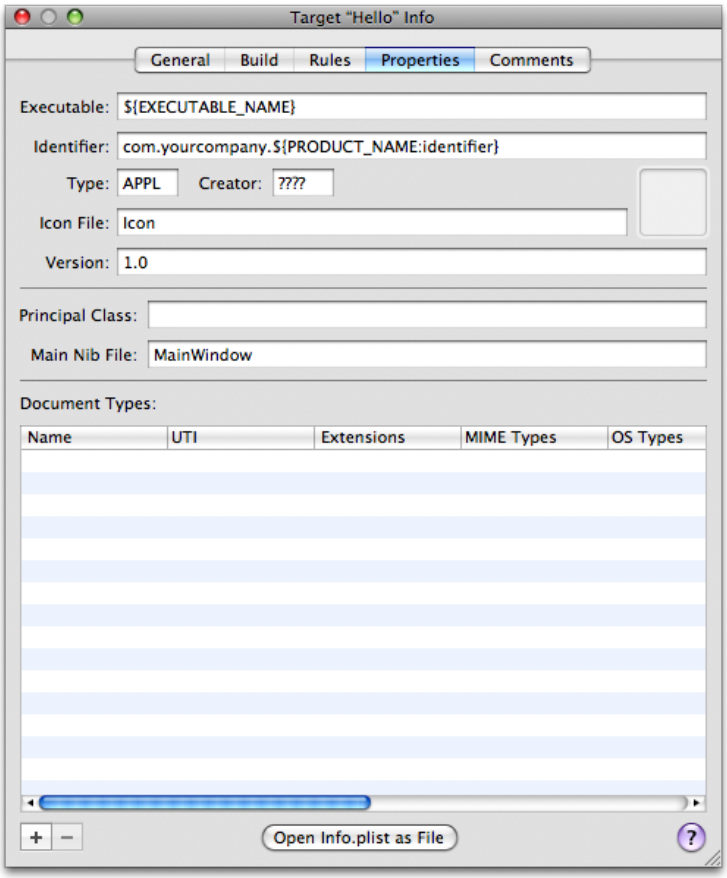
mainBundleクラスメソッドを呼び出すと、アプリケーションバンドルを表すオブジェクトが返されます。リソースのロードの詳細については『*Resource Programming Guide*』を参照してください。

情報プロパティリスト

情報プロパティリストは、`Info.plist`という名前がつけられたファイルで、Xcodeで作成するすべてのiPhoneアプリケーションプロジェクトに含まれます。このファイルは、キー値のペアによって、アプリケーションの重要なランタイム設定情報を指定するプロパティの一覧です。情報プロパティリストの要素は階層構造になっており、階層内の各ノードは、配列、辞書、文字列、その他のスカラー型などのエンティティです。

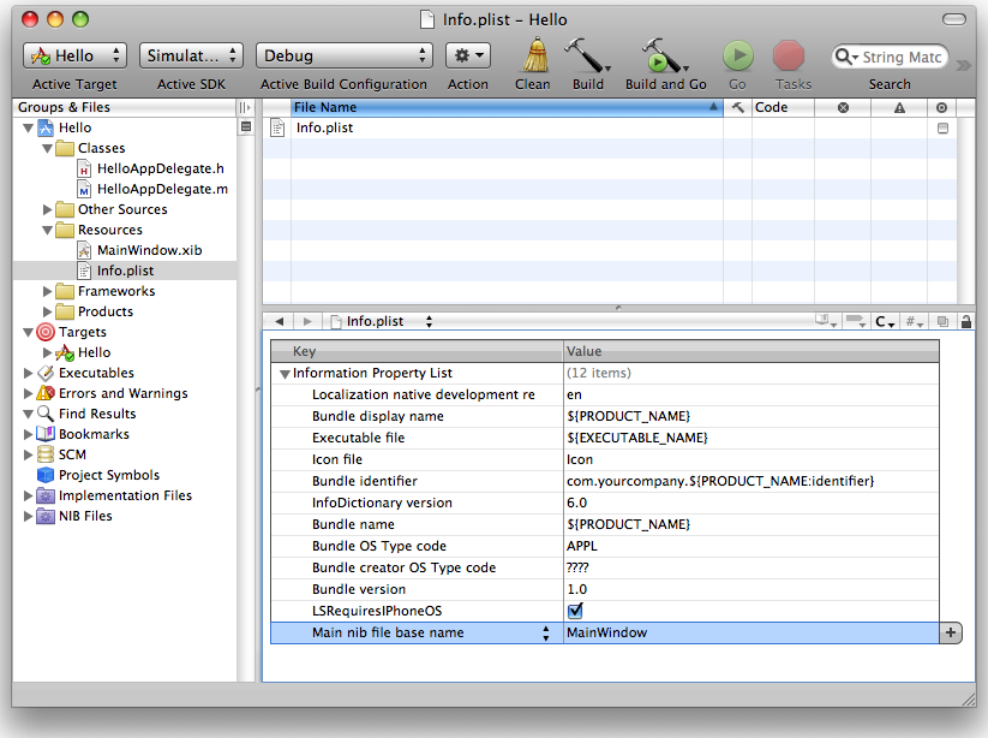
Xcodeでは、「プロジェクト(Project)」メニューの「アクティブターゲット*TargetName*を編集(Edit Active Target *TargetName*)」を選択することで、情報プロパティリストにアクセスできます。次にターゲットの情報ウインドウで、「プロパティ(Properties)」コントロールをクリックします。Xcodeに、図 1-4の例に示すような情報が含まれるペインが表示されます。

図 1-4　ターゲットの情報ウインドウの「プロパティ(Properties)」ペイン



「プロパティ(Properties)」ペインには、アプリケーションバンドルのプロパティの一部が表示されます。「ファイルとして開く(Open Info.plist as File)」ボタンをクリックするか、またはXcodeプロジェクト内で`Info.plist`ファイルを選択すると、Xcodeにより、図 1-5に示すような、プロパティリストの編集用ウインドウが表示されます。このウインドウを使用して、プロパティの値を編集したり、新規のキー値ペアを追加したりできます。Info.plistファイルに追加された実際のキー名を参照するには、エディタの「Information Property List」の項目をControlキーを押しながらクリックし、コンテキストメニューから「Show Raw Keys/Values」を選択します。

図 1-5 情報プロパティリストエディタ



これらのプロパティの中には、Xcodeによって自動的に値が設定されるものもありますが、明示的に設定する必要があるものもあります。表 1-3に、アプリケーションの`Info.plist`ファイルで指定することが推奨される重要なキーの一部を示します（Xcodeではデフォルトで実際のキー名が隠されています。そのため、Xcodeで表示される文字列も括弧内に示します。エディタで「Information Property List」のキーを、コントロールを押しながらクリックし、コンテキストメニューから「Show Raw Keys/Values」を選ぶことによって、すべてのキーの実際の名前を表示できます）。このファイルに含まれるプロパティの全リスト、およびそれらがシステムによってどのように使用されるかについての情報は、『*Runtime Configuration Guidelines*』を参照してください。

表 1-3 `Info.plist`ファイルの重要なキー

キー	値
`CFBundleDisplayName` (Bundle display name)	アプリケーションアイコンの下に表示する名前。この値は、サポートするすべての言語についてローカライズしてください。
`CFBundleIdentifier` (Bundle identifier)	システムに対してアプリケーションを識別するために提供する識別子文字列。この文字列は、英数字（`A-Z`、`a-z`、`0-9`）、ハイフン(`-`)、およびピリオド(`.`)のみで構成されるUTI (Uniform Type Identifier)でなければなりません。また、この文字列は逆DNS形式にする必要があります。たとえば、会社のドメインが`Ajax.com`で、Helloという名前のアプリケーションを作成する場合は、`com.Ajax.Hello`という文字列をアプリケーションのバンドル識別子として割り当てます。 バンドル識別子は、アプリケーションの署名の検証に使用します。

キー	値
CFBundleURLTypes (URL types)	アプリケーションが処理可能な、URLタイプの配列。各URLタイプは、アプリケーションが処理可能なスキーム（httpやmailtoなど）を定義する辞書です。このプロパティを使用することにより、アプリケーションは独自のURLスキームを登録できます。
CFBundleVersion (Bundle version)	バンドルのビルドバージョン番号を指定する文字列。この値は、単純増加していく値を示す文字列で、ピリオドで区切られた1個以上の整数で構成されます。この値はローカライズできません。
LSRequiresIPhoneOS	バンドルがiPhone OSでのみ実行可能かどうかを示すブール値。Xcodeはこのキーを自動的に追加し、値をtrueに設定します。このキーの値は変更してはいけません。
NSMainNibFile (Main nib file base name)	アプリケーションのメインnibファイルの名前を示す文字列。プロジェクトのために作成されたデフォルトのもの以外のnibファイルを使用する場合は、そのnibファイルの名前をこのキーに関連付けます。nibファイルの名前にはファイル名拡張子の.nibを含めないようにします。
UIStatusBarStyle	アプリケーション起動時のステータスバーのスタイルを示す文字列。この値は、UIApplication.hヘッダファイルで宣言されている、UIStatusBarStyle定数に基づいています。デフォルトのスタイルはUIStatusBarStyleDefaultです。アプリケーションは、起動処理が完了したときに、この初期のステータスバーのスタイルを変更できます。
UIStatusBarHidden	アプリケーション起動時に、ステータスバーの初期状態を、表示または非表示のどちらにするかを指定するブール値。ステータスバーを非表示にするにはこれをtrueに設定します。デフォルト値はfalseです。
UIInterface-Orientation	アプリケーションのユーザインターフェイスの最初の向きを示す文字列。この値は、UIApplication.hヘッダファイルに宣言されている、UIInterfaceOrientation定数に基づいています。デフォルトのスタイルはUIInterfaceOrientationPortraitです。 横長モードでのアプリケーションの起動の詳細については、「横長モードでの起動」（38 ページ）を参照してください。
UIPrerenderedIcon	アプリケーションアイコンにすでに光沢やベベルエフェクトが含まれているかを示すブール値。デフォルトのプロパティはfalseです。システムでこれらのエフェクトをアートワークに追加しない場合は、値をtrueに設定します。

キー	値
UIRequiredDevice-Capabilities	アプリケーションを実行するために必要なデバイス関連の機能をiTunesおよびApp Storeに知らせるための情報キー。iTunesおよびモバイル用のApp Storeでは、このリストを使用して、このリストにある機能をサポートしていないデバイスに顧客がアプリケーションをインストールするのを防止します。 このキーの値は、配列または辞書のいずれかです。配列を使用する場合は、与えられたキーが存在すればそれに対応する機能が必要であることを表します。辞書を使用する場合は、その機能が必要かどうかを表すブール値をキーごとに指定しなければなりません。どちらの場合も、キーが含まれていなければその機能は必要ないことを示します。 辞書に含めることができるキーのリストについては、表 1-4 （32 ページ）を参照してください。このキーはiPhone OS 3.0以降でサポートされます。
UIRequiresPersistent-WiFi	アプリケーションが通信にWi-Fiネットワークを使用していることをシステムに通知するブール値。一定の時間Wi-Fiを使用するアプリケーションは、このキーをtrueに設定する必要があります。そうしないと、デバイスは30分経過すると、節電のためWi-Fi接続を停止します。このフラグを設定することで、現在Wi-Fiを利用していなくても利用可能であれば、ネットワーク選択のダイアログを表示する必要があることもシステムに知らせます。デフォルト値はfalseです。 デバイスが待機中（つまり、画面がロックされている）の場合は、このプロパティの値がtrueであっても影響はありません。アプリケーションはある程度は動作しますが、アクティブではないと見なされ、Wi-Fi接続を持ちません。
UISupportedExternal-AccessoryProtocols	アプリケーションがサポートする外部アクセサリのプロトコルを表す文字列の配列。これらは、iPhoneまたはiPod touchに取り付けられているサードパーティ製のハードウェアと通信するためにアプリケーションが使用するプロトコルです。システムは、このリストに含まれるプロトコルを使用して、アクセサリがデバイスに接続されているときに開くことができるアプリケーションを識別します。 アクセサリおよびプロトコルの詳細については、「外部アクセサリとの通信」（166 ページ）を参照してください。このキーはiPhone OS 3.0以降でサポートされます。
UIViewGroupOpacity	Core Animationのサブレイヤがスーパーレイヤの不透明度を継承するかどうかを表すブール値。この機能によって、iPhone Simulatorではより洗練されたレンダリングが可能になりますが、パフォーマンスに重大な影響をもたらすことがあります。このキーが存在しない場合、このキーのデフォルト値はNOになります。 このキーはiPhone OS 3.0以降でサポートされます。
UIViewEdge-Antialiasing	Core Animationレイヤが、ピクセル境界に揃わないレイヤを描画する際にアンチエイリアスを使用するかどうかを表すブール値。この機能によって、iPhone Simulatorではより洗練されたレンダリングが可能になりますが、パフォーマンスに重大な影響をもたらすことがあります。このキーが存在しない場合、このキーのデフォルト値はNOになります。 このキーはiPhone OS 3.0以降でサポートされます。

ユーザインターフェイスに表示する文字列を指定するプロパティはローカライズしてください。特に、Info.plistの文字列は、ローカライズする言語のサブディレクトリにあるInfoPlist.stringsファイル内のローカライズされた文字列を指すキーとなります。詳細については、「アプリケーションの国際化」（44 ページ）を参照してください。

表 1-4は、UIRequiredDeviceCapabilitiesキーに対応する配列または辞書に含めることができるキーの一覧です。アプリケーションでは必ず必要な機能に対応するキーのみを含めるべきです。アプリケーションで、適切なコードパスを実行しないことによって欠けている機能に対応できる場合は、それに対応するキーを含める必要はありません。

表 1-4 UIRequiredDeviceCapabilitiesキー用の辞書キー

キー	説明
telephony	アプリケーションで「電話(Phone)」アプリケーションが必要な場合はこのキーを含めます。アプリケーションがtelスキームを利用してURLを開く場合は、この機能が必要になることがあります。
sms	アプリケーションで「SMS/MMS (Messages)」アプリケーションが必要な場合はこのキーを含めます。アプリケーションがsmsスキームを利用してURLを開く場合は、この機能が必要になることがあります。
still-camera	アプリケーションがUIImagePickerControllerインターフェイスを使用してデバイスのスチルカメラから画像をキャプチャする場合は、このキーを含めます。
auto-focus-camera	アプリケーションでデバイスのスチルカメラにオートフォーカス機能が必要な場合は、このキーを含めます。ほとんどのデベロッパはこのキーを含める必要はありませんが、アプリケーションでマクロ撮影をサポートしたり、ある種の画像処理のためにより鮮明な画像が必要な場合はこのキーを含めます。
video-camera	アプリケーションがUIImagePickerControllerインターフェイスを使用してデバイスのカメラからビデオをキャプチャする場合は、このキーを含めます。
wifi	アプリケーションでデバイスのネットワーク機能にアクセスする必要がある場合にこのキーを含めます。
accelerometer	アプリケーションがUIAccelerometerインターフェイスを使用して加速度センサーイベントを受信する場合は、このキーを含めます。アプリケーションでデバイスの向きの変化のみを検出する場合はこのキーを含める必要はありません。
location-services	アプリケーションがCore Locationフレームワークを使用してデバイスの現在位置にアクセスする場合は、このキーを含めます（このキーは、一般的なロケーションサービス機能を指します。GPSレベルの精度が特に必要な場合は、gpsキーも含める必要があります）。
gps	アプリケーションで位置を追跡する際に、より高い精度を得るためにGPS（またはAGPS）ハードウェアが必要な場合はこのキーを含めます。このキーを含める場合はlocation-servicesキーも含める必要があります。アプリケーションで、携帯電話無線やWi-Fi無線で得られるものより正確な位置データが必要な場合にのみGPSを必須にすべきです。

キー	説明
magnetometer	アプリケーションがCore Locationフレームワークを使用して方角に関連するイベントを受信する場合は、このキーを含めます。
microphone	アプリケーションで内蔵マイクを使用したりマイクを提供するアクセサリをサポートしたりする場合は、このキーを含めます。
opengles-1	アプリケーションでOpenGL ES 1.1インターフェイスが必要な場合は、このキーを含めます。
opengles-2	アプリケーションでOpenGL ES 2.0インターフェイスが必要な場合は、このキーを含めます。

アプリケーションアイコンと起動画像

ユーザのホーム(Home)画面に表示されるアイコンのファイルには、Icon.pngというデフォルトの名前が付いています（Info.plistファイルのCFBundleIconFileプロパティで名前を変更できます）。このファイルは、PNG形式の画像ファイルで、アプリケーションバンドルの最上位に置きます。アプリケーションアイコンは、光沢やラウンドベベルのエフェクトが適用されていない、57×57ピクセルの画像です。通常、システムはこれらのエフェクトを、表示前にアイコンに適用します。ただし、アプリケーションのInfo.plistファイルにUIPrerenderedIconキーを含めることによって、この動作をオーバーライドすることができます。詳細については、表 1-3 （29 ページ）を参照してください。

注： （App Store経由ではなく）アドホック配布を使用してローカルユーザにアプリケーションを配布する場合は、アプリケーションバンドルのiTunesArtworkと呼ばれるファイルに、512×512ピクセルバージョンのアプリケーションアイコンも含める必要があります。このファイルは、アプリケーションを配布するときにiTunesによって表示されるアイコンを提供します。

アプリケーションの起動画像に使用するファイルにはDefault.pngという名前を付けます。この画像は、アプリケーションの最初のユーザインターフェイスによく似た画像にする必要があります。システムは、アプリケーションがユーザインターフェイスを表示する準備ができるまで起動画像を表示することで、アプリケーションがすばやく起動された印象をユーザに与えます。起動画像も、PNG形式の画像ファイルとして、アプリケーションバンドルの最上位に置きます。アプリケーションがURLを使用して起動された場合、システムはDefault-*scheme*.pngという名前の起動画像を検索します（*scheme*はURLのスキームです）。このファイルが存在しない場合、Default.pngが代わりに選択されます。

Xcodeのプロジェクトに画像ファイルを追加するには、「プロジェクト(Project)」メニューの「プロジェクトに追加(Add to Project)」を選択し、ブラウザでファイルを指定して、「追加(Add)」をクリックします。

注： アプリケーションバンドルの最上位レベルにあるアイコンと起動画像のほかに、これらの画像のローカライズバージョンをアプリケーションの言語固有のプロジェクトサブディレクトリに置くこともできます。アプリケーションでのリソースのローカライズについては、「アプリケーションの国際化」（44 ページ）を参照してください。

nibファイル

nibファイルは、アプリケーションで後から使用する予定のオブジェクトを「フリーズドライ」にしたものを格納しておくデータファイルです。アプリケーションは、ユーザインターフェイスを構成するウインドウやビューを保存するために、nibファイルを最もよく使用します。nibファイルをアプリケーションにロードすると、nibロード用のコードによって、その内容がアプリケーションで操作可能な実際のオブジェクトに展開されます。このため、nibファイルを利用すると、コードを記述してプログラムによってオブジェクトを作成する必要がなくなります。

Interface Builderは、nibファイルを作成するために使用するビジュアル設計環境です。標準オブジェクト（UIKitフレームワークで提供されるウインドウやビューなど）とXcodeプロジェクトからのカスタムオブジェクトを組み合わせて、nibファイルを組み立てます。Interface Builder内でビュー階層を作成するには、決まった場所にビューをドラッグアンドドロップするだけです。また、インスペクタウインドウを使用して各オブジェクトの属性を設定したり、オブジェクトの実行時の関係を定義するために、オブジェクト間に接続を作成することもできます。ここで行ったすべての変更は、その後、nibファイルの一部として保存されます。

実行時に、nibファイルに含まれているオブジェクトが必要になったときは、そのnibファイルをアプリケーションにロードします。通常、nibファイルをロードするのは、ユーザインターフェイスが変更されたときや、画面上に新しいビューを表示する必要があるときです。アプリケーションがView Controllerを使用する場合は、View Controllerが自動的にnibロードプロセスを処理します。ただし、`NSBundle`クラスのメソッドを使用してnibファイルをデベロッパが自分でロードすることもできます。

アプリケーションのユーザインターフェイスの設計方法については、『*iPhone Human Interface Guidelines*』を参照してください。nibファイルの作成方法については、『*Interface Builder User Guide*』を参照してください。

重要なアプリケーションタスクの処理

このセクションでは、すべてのiPhoneアプリケーションが実行しなければならない、いくつかのタスクについて説明します。これらのタスクは、アプリケーションのライフサイクル全体の不可欠な要素であり、iPhone OSとアプリケーションを統合する重要な方法の一部です。最悪の場合、これらのタスクのいくつかの処理に失敗すると、オペレーティングシステムによってアプリケーションが強制終了させられることもあります。

初期化と終了

初期化と終了の間、`UIApplication`クラスは、アプリケーションデリゲートが必要なタスクを実行できるように、適切なメッセージを送ります。アプリケーションは、これらのメッセージに応答する必要はありませんが、ほとんどすべてのiPhoneアプリケーションがこれらのメッセージを処理す

る必要があります。初期化時は、アプリケーションのユーザインターフェイスを準備し、アプリケーションを初期実行状態にする期間です。同様に、終了時は、未保存のデータと主要なアプリケーション状態をディスクに書き込む期間です。

iPhoneアプリケーションは、ほかのアプリケーションが起動される前に終了しなければならないので、初期化と終了のコードを実行するのにかかる時間はできる限り短くする必要があります。初期化時は、すぐには使用しない大きなデータ構造のロードを始めるときではありません。起動時の目標は、アプリケーションのユーザインターフェイスをできるだけすばやく、できればそのアプリケーションを前回終了したときと同じ状態にして表示することです。アプリケーションが起動時にネットワークからデータをロードしたり、時間のかかるその他のタスクを実行するためにさらに時間が必要な場合は、まずインターフェイスを立ち上げて実行させた後で、バックグラウンドスレッドで時間のかかるタスクを起動します。それには、進捗インジケータやその他のフィードバックをユーザに表示して、アプリケーションが必要なデータをロード中であることや、重要な処理を実行中であることを示す必要があります。

表 1-5は、初期化と終了のタスクを処理するためにアプリケーションデリゲートに実装する、`UIApplicationDelegate`プロトコルのメソッドの一覧を示しています。この表には、各メソッド内で実行しなければならない重要な処理も列挙されています。

表 1-5　アプリケーションデリゲートの責務

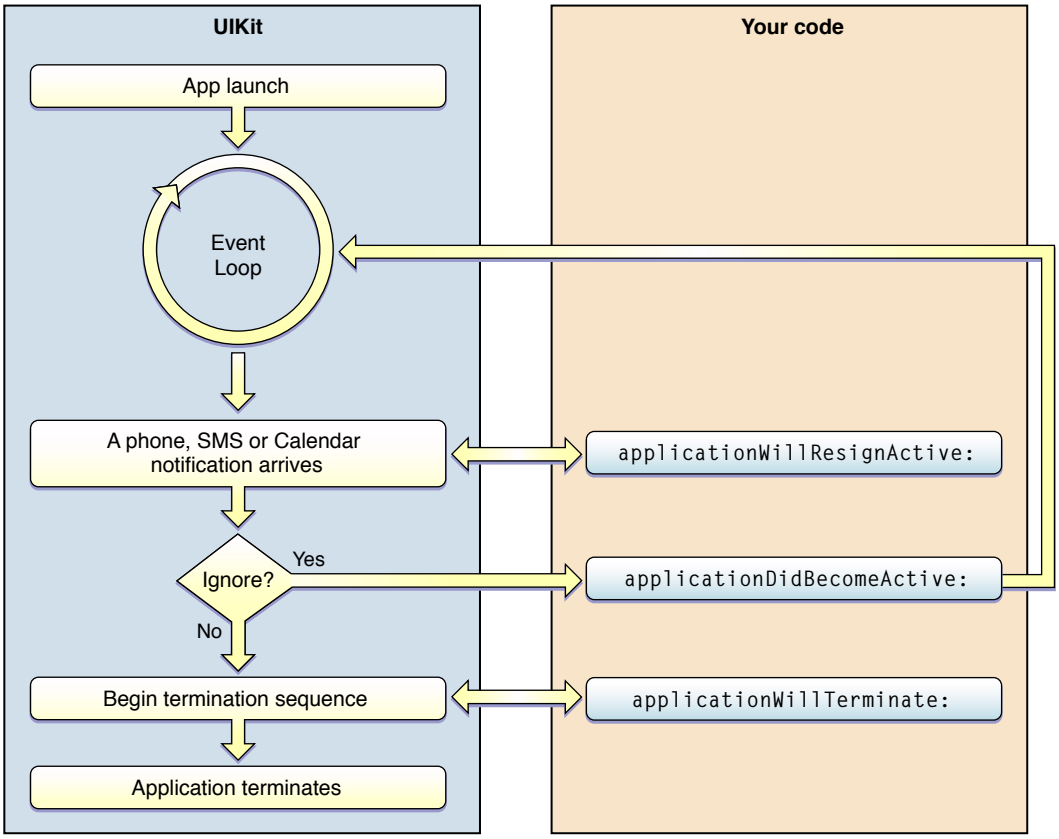
デリゲートメソッド	説明
`applicationDidFinish-Launching:`	このメソッドは、アプリケーションを前回のセッション中の状態に復元するために使用します。また、このメソッドは、アプリケーションのデータ構造やユーザインターフェイスを独自に初期化するためにも使用できます。
`applicationWill-Terminate:`	このメソッドは、未保存のデータや主要なアプリケーション状態をディスクに保存するために使用します。一時ファイルの削除などの追加のクリーンアップ操作を実行するために、このメソッドを使用することもできます。

割り込みへの応答

アプリケーションを終了するホーム(Home)ボタンのほかに、ユーザが重要なイベントに応答できるように、システムは、アプリケーションに一時的に割り込みをかけることができます。たとえば、アプリケーションは電話の着信、SMSメッセージ、カレンダー通知、またはユーザがデバイス上のスリープ(Sleep)ボタンを押す操作によって中断される可能性があります。ホーム(Home)ボタンを押すとアプリケーションが終了しますが、このような割り込みが一時的なだけの場合もあります。ユーザが割り込みを無視した場合、アプリケーションは今までどおり実行し続けます。しかし、ユーザが電話を受けたり、SMSメッセージに応答すると、システムは実行中のアプリケーションを終了させます。

図1-6は、電話の着信、SMSメッセージ、またはカレンダー通知を受けている間に発生するイベントのシーケンスを示しています。このすぐ後に示す手順で、このシーケンスのキーポイントを詳しく説明します。また、アプリケーションが、各イベントに応答して実行しなければならない処理についても説明します。このシーケンスには、ユーザがスリープ/スリープ解除(Sleep/Wake)ボタンを押したときの反応は反映されていません。この場合のシーケンスについては、手順の後で説明します。

図 1-6 割り込みの間のイベントの流れ



1. システムは、電話の着信、SMSメッセージ、またはカレンダーイベントが発生したことを検出します。

2. システムがアプリケーションデリゲートの`applicationWillResignActive:`メソッドを呼び出します。システムは、アプリケーションに対するタッチイベントの送信も無効にします。

   割り込みによって、アプリケーションによる制御が一時的に失われます。制御が失われることによって、アプリケーションの動作に影響したり、ユーザ体験に悪影響が生じる可能性がある場合は、それを避けるために、デリゲートメソッド内で適切な手順を実行する必要があります。たとえば、ゲームアプリケーションの場合は、ゲームを一時停止する必要があります。また、タイマーを無効にし、（OpenGLを使っている場合は）OpenGLフレームレートを落として、アプリケーションを全体としてスリープ状態にすることもできます。休止状態の間、アプリケーションは実行されていますが、何も重要な仕事はしません。

3. システムが、イベントに関する情報を警告パネルに表示します。ユーザはイベントを無視するか、イベントに応答するかを選択できます。

4. ユーザがイベントを無視すると、システムはアプリケーションデリゲートの`applicationDidBecomeActive:`メソッドを呼び出し、アプリケーションへのタッチイベントの配信を再開します。

このデリゲートメソッドを使用して、タイマーを再度有効にし、OpenGLフレームレートを上げて、アプリケーションをスリープ状態から戻すことができます。停止状態のゲームの場合は、ユーザがゲームを再開するまでは、ゲームを停止状態のままにしておくことを考慮する必要があります。たとえば、プレイを再開するためのコントロールが付いた警告パネルを表示することもできます。

5. ユーザが、イベントを無視せずにイベントに応答すると、システムはアプリケーションデリゲートの`applicationWillTerminate:`メソッドを呼び出します。アプリケーションは、次回起動時にアプリケーションの同じ場所にユーザを戻せるように、必要な文脈情報をすべて保存して通常どおりに終了します。

   アプリケーションが終了した後は、システムは、割り込みの処理を担当するアプリケーションを起動します。

割り込みに応答している間のユーザの操作によっては、システムは、割り込みが終了するとアプリケーションを再び起動します。たとえば、ユーザが電話の着信を受けた後、電話を切ると、システムはアプリケーションを再び起動します。通話の間に、ユーザがホーム(Home)画面に戻るか別のアプリケーションを起動した場合、システムは着信時に起動していたアプリケーションを再び起動しません。

重要： ユーザが電話を受けてから、通話中にアプリケーションを起動しなおすと、ユーザが通話中であることを示すためにステータスバーの高さが高くなります。同様に、ユーザが通話を終了すると、ステータスバーの高さは標準のサイズに縮小します。アプリケーションは、ステータスバーのこうした変化に備え、コンテンツ領域を適宜調整する必要があります。View Controllerは、この動作を自動的に処理します。ただし、ユーザインターフェイスをプログラミングによってレイアウトする場合、ビューをレイアウトする際にステータスバーの高さを考慮し、`layoutSubviews`メソッドを実装してレイアウトの動的な変化に対応する必要があります。

アプリケーションの実行中にユーザがデバイスのスリープ/スリープ解除(Sleep/Wake)ボタンを押した場合は、システムはアプリケーションデリゲートの`applicationWillResignActive:`メソッドを呼び出し、タッチイベントの配信を停止してデバイスをスリープ状態にします。その後、ユーザがデバイスのスリープ状態を解除すると、システムは、アプリケーションデリゲートの`applicationDidBecomeActive:`メソッドを呼び出して、アプリケーションへのイベントの配信を再開します。その他の割り込みを処理するときは、これらのメソッドを使用して、アプリケーションをスリープ状態にして（または、ゲームを中止して）から再度スリープを解除します。スリープ状態の間は、アプリケーションの消費電力をできるだけ少なくする必要があります。

メモリ不足警告を監視する

システムがメモリ不足の警告をアプリケーションに送ってきた場合は、それに注意を払う必要があります。空きメモリの容量が安全しきい値を下回ると、iPhone OSは最前面のアプリケーションに通知をします。アプリケーションがこの警告を受け取った場合は、不要になったオブジェクトを解放したり、後から簡単に作成しなおせるメモリキャッシュを空にするなどして、できる限り多くのメモリを解放する必要があります。

UIKitでは、メモリ不足の警告を受け取るための方法をいくつか提供しています。以下の方法が含まれます。

- アプリケーションデリゲートの`applicationDidReceiveMemoryWarning:`メソッドを実装する。

- UIViewControllerのカスタムサブクラスで、didReceiveMemoryWarningメソッドをオーバーライドする。
- UIApplicationDidReceiveMemoryWarningNotification通知を受け取るように登録する。

このような警告を受け取ったら、ハンドラメソッドではただちに不要なメモリを解放して対応する必要があります。View Controllerでは、現在オフスクリーンのビューをすべて消去します。また、アプリケーションデリゲートでは、解放できるデータ構造をすべて解放したり、ほかのアプリケーションオブジェクトにメモリを解放するように通知します。

カスタムオブジェクトが既知の消去可能なリソースを持つ場合は、これらのオブジェクトがUIApplicationDidReceiveMemoryWarningNotification通知を受け取れるように登録して、消去可能なリソースを直接解放させることもできます。消去可能なリソースのほとんどを管理するオブジェクトが少数で、これらのすべてのリソースを消去するのが適切な場合は、これらのオブジェクトを登録します。ただし、消去可能なオブジェクトがたくさんある場合や、これらのオブジェクトのサブセットだけを解放するように調整したい場合は、アプリケーションデリゲートを使用して希望のオブジェクトを解放します。

重要： テスト中にメモリ不足の警告を受けなかったとしても、システムアプリケーションと同様にアプリケーションでは常にメモリ不足の警告に対応する必要があります。システムアプリケーションでは、さまざまな要求を処理する間、少量のメモリを消費します。メモリ不足の状態が検出されると、システムはメモリ不足を緩和するために（読者の作ったアプリケーションを含め）実行中のすべてのプログラムにメモリ不足の警告を送ります。また、いくつかのバックグラウンドアプリケーションを終了させる場合があります。アプリケーションにメモリリークがあったり、アプリケーションが依然として大量のメモリを消費し続けているなどの理由で、十分なメモリが解放されなかった場合、システムによってアプリケーションが強制終了される場合があります。

アプリケーションの動作のカスタマイズ

基本的なアプリケーションの動作をカスタマイズして、希望通りのユーザ体験を提供するには、さまざまな方法があります。以降のセクションでは、アプリケーションレベルで行う必要があるカスタマイズについて説明します。

横長モードでの起動

iPhone OSのアプリケーションは、通常、ホーム(Home)画面の向きに合わせるために縦長モードで起動します。縦長と横長の両方のモードで実行されるアプリケーションの場合、最初は常に縦長モードで起動し、その後で、View Controllerによってデバイスの向きに基き、必要に応じてインターフェイスを回転します。ただし、アプリケーションを横長モードでのみ実行する場合は、最初から横長モードで起動するために次の手順を実行する必要があります。

- アプリケーションのInfo.plistファイルに、UIInterfaceOrientationキーを追加し、その値を横長モードに設定する。このキーの値は、UIInterfaceOrientationLandscapeLeftまたはUIInterfaceOrientationLandscapeRightに設定できます。
- ビューを横長モードにレイアウトし、ビューの自動サイズ変更オプションを確実に正しく設定する。

- View Controllerの`shouldAutorotateToInterfaceOrientation:`メソッドをオーバーライドして、希望する横長の向きの場合にのみ`YES`を返し、縦長の向きの場合は`NO`を返す。

重要： 前述の手順は、アプリケーションが、ビュー階層を管理するためにView Controllerを使用していることを前提としています。View Controllerは、その他の複雑なビュー関連イベントだけでなく、向きの変化を処理するために相当な量のインフラストラクチャを提供しています。アプリケーションでView Controllerを使用していない場合は（ゲームや、OpenGL ESベースのその他のアプリケーション）、横長モードでコンテンツを表示するために、必要に応じて描画サーフェイスを回転するのはデベロッパの責任になります。

`UIInterfaceOrientation`プロパティは、起動時にView Controllerが管理するビューの向きだけでなく、アプリケーションのステータスバー（表示される場合）の向きを設定するためのヒントをiPhone OSに与えます。iPhone OS 2.1以降では、View Controllerがこのプロパティを尊重し、それに合わせてビューの向きの初期値を設定します。このプロパティを使用することは、`UIApplication`の`setStatusBarOrientation:animated:`メソッドを、`applicationDidFinishLaunching:`メソッドの実行初期に呼び出すことと同等です。

注： iPhone OS v2.1より前のバージョンで、View Controllerベースのアプリケーションを横長モードで起動するには、前述のすべての手順に加えて、アプリケーションのルートビューの変換に90度の回転を適用する必要があります。iPhone OS 2.1より前のバージョンでは、View Controllerは、`UIInterfaceOrientation`キーの値に基づいて自動的にビューを回転しません。ただし、iPhone OS 2.1以降では、この手順は不要です。

他のアプリケーションとのやり取り

アプリケーションが既知の型のURLを扱う場合は、そのURLスキームを使用してそのアプリケーションとやり取りができます。ただし、ほとんどの場合、URLは別のアプリケーションを起動して、自分のアプリケーションに関連する情報を表示させるためだけに使われます。たとえば、アプリケーションでアドレス情報を管理する場合は、与えられたアドレスを含むURLを「マップ(Maps)」アプリケーションに送って、その場所を表示することができます。このレベルのやり取りによって、ユーザのためのより統合された環境を作ることができます。また、自分のアプリケーションでデバイス上のほかの場所に存在する機能を実行することが容易になります。

Appleでは、`http`、`mailto`、`tel`、および`sms`のURLスキームを組み込みでサポートしています。また、「マップ(Maps)」、「YouTube」、および「iPod」アプリケーションを対象にした`http`ベースのURLもサポートしています。アプリケーション固有のカスタムURLスキームを登録することもできます。アプリケーションとやり取りするには、適切な形式のコンテンツを含む`NSURL`オブジェクトを作成し、それを共有の`UIApplication`オブジェクトの`openURL:`メソッドに渡します。`openURL:`メソッドは、このタイプのURLを受け取るように登録されているアプリケーションを起動し、それにURLを渡します。その後、ユーザがこのアプリケーションを終了すると、通常、システムは元のアプリケーションを起動しなおします。ただし、常にそうするとは限りません。アプリケーションを起動しなおすかどうかの決定は、ハンドラアプリケーションでのユーザの動作に基づいて行われます。元のアプリケーションに戻るかどうかは、ユーザの立場からは意味があります。

次のコード例では、あるアプリケーションが別のアプリケーションのサービスを要求する方法を示します（この例に示す「todolist」は、アプリケーションによって登録された架空のカスタムスキームです）。

```
NSURL *myURL = [NSURL
URLWithString:@"todolist://www.acme.com?Quarterly%20Report#200806231300"];
```

```
[[UIApplication sharedApplication] openURL:myURL];
```

重要： デベロッパのURLタイプにAppleが定義したものと同じスキームが含まれている場合は、デベロッパのアプリケーションではなく、Appleが提供するアプリケーションが起動されます。複数のサードパーティアプリケーションが、同じURLスキームを処理するよう登録されている場合は、このタイプのURLを処理するために、どのアプリケーションがピックアップされるかは未定義になります。

アプリケーションで固有のカスタムURLスキームを定義する場合は、「カスタムURLスキームの実装」（40 ページ）で説明するような、そのスキーム用のハンドラを実装する必要があります。システムがサポートするURLスキームの詳細（URLの書式化方法の情報を含む）については、『*Apple URL Scheme Reference*』を参照してください。

カスタムURLスキームの実装

アプリケーションで使用するカスタムURLスキームを含むURLタイプを登録することができます。カスタムURLスキームは、サードパーティアプリケーションどうし、およびサードパーティアプリケーションとシステムとの間でのやりとりを可能にする仕組みです。カスタムURLスキームを利用することで、アプリケーションは、自身のサービスをほかのアプリケーションに利用させることができます。

カスタムURLスキームの登録

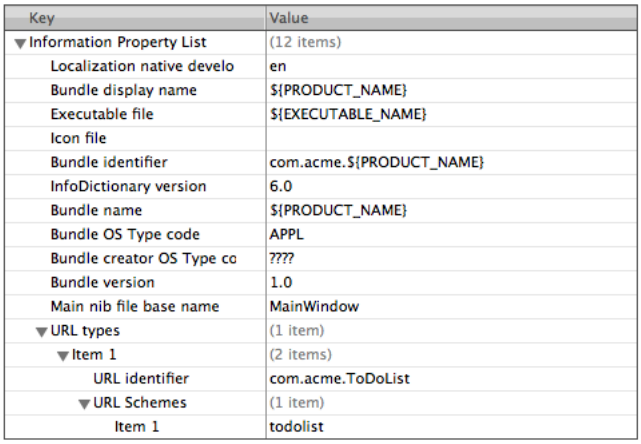
アプリケーションのURLタイプを登録するには、「情報プロパティリスト」（28 ページ）で紹介した`CFBundleURLTypes`プロパティのサブプロパティを指定する必要があります。`CFBundleURLTypes`プロパティは、アプリケーションの`Info.plist`ファイル内の辞書の配列です。それぞれの辞書は、アプリケーションがサポートするURLタイプを定義します。表1-6は、`CFBundleURLTypes`辞書のキーと値の説明です。

表 1-6 `CFBundleURLTypes`プロパティのキーと値

キー	値
`CFBundleURLName`	URLタイプを示す抽象名文字列。一意性を確保するため、`com.acme.myscheme`のように、逆DNS形式の識別子を指定することをお勧めします。 この文字列値として指定するURLタイプ名は、ローカライズされた言語のバンドルのサブディレクトリに格納されている`InfoPlist.strings`ファイルに含まれる、ローカライズされた文字列を指すキーとして使用されます。ローカライズされた文字列は、URLタイプを示す、所定の言語で人が読める形式で表した名前です。
`CFBundleURLSchemes`	このURLタイプに属するURLを表すURLスキームの配列。それぞれのスキームは文字列です。あるURLタイプに属するURLは、スキームの構成要素によって特徴付けられます。

図 1-7は、組み込みのXcodeエディタを使用して編集中の、アプリケーションのInfo.plistファイルを示しています。この図では、左の列のURLタイプエントリが、Info.plistファイルに直接追加したCFBundleURLTypesキーに相当します。同様に、「URL identifier」エントリと「URL Schemes」エントリは、CFBundleURLNameキーとCFBundleURLSchemesキーに相当します。

図 1-7 Info.plistファイルにおけるカスタムURLスキームの定義



CFBundleURLTypesプロパティを定義することによって、カスタムスキームのURLタイプを登録したら、次の方法でスキームをテストできます。

1. アプリケーションをビルドし、インストールして実行します。
2. ホーム(Home)画面を表示し、Safariを起動します（iPhone Simulatorでは、メニューから「ハードウェア(Hardware)」＞「ホーム(Home)」を選択することで、ホーム(Home)画面を表示できます）。
3. Safariのアドレスバーに、カスタムスキームを使用するURLを入力します。
4. アプリケーションが起動し、アプリケーションデリゲートにapplication:handleOpenURL:メッセージが送信されたことを確認します。

URLリクエストの処理

アプリケーションのデリゲートでは、application:handleOpenURL:メソッドを実装することでアプリケーションに送信されたURLリクエストを処理します。アプリケーション用にカスタムURLスキームを登録している場合は、特別にこのメソッドを実装したデリゲートが必要になります。

カスタムスキームに基づいたURLリクエストは、読者のアプリケーションが提供するサービスを要求するアプリケーションが理解できる一種のプロトコルを想定します。URLには、スキームを登録したアプリケーションが何らかの方法で処理または応答することが期待される、何らかの情報が含まれます。application:handleOpenURL:メソッドに渡されるNSURLクラスは、Cocoa TouchフレームワークのURLを表します。NSURLは、RFC 1808仕様に準拠し、RFC 1808で定義されている、ユーザ、パスワード、クエリ、コード、およびパラメータの文字列など、URLのさまざまな部分を返すメソッドが含まれています。カスタムスキームの「protocol」で、さまざまな種類の情報を伝えるためにURLのこうした部分を使用できます。

リスト 1-2に示すapplication:handleOpenURL:メソッドの実装では、渡されたURLオブジェクトが、クエリおよびコードの部分を使ってアプリケーション固有の情報を伝えます。デリゲートは、この情報（この例では、to-doタスクの名前とタスクの実行日時）を抽出し、それを使ってアプリケーションのモデルオブジェクトを作成します。

リスト 1-2　カスタムスキームに基づいたURLリクエストの処理

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
    if ([[url scheme] isEqualToString:@"todolist"]) {
        ToDoItem *item = [[ToDoItem alloc] init];
        NSString *taskName = [url query];
        if (!taskName || ![self isValidTaskString:taskName]) { // タスク名を持っ
ている必要がある
            [item release];
            return NO;
        }
        taskName = [taskName
stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];

        item.toDoTask = taskName;
        NSString *dateString = [url fragment];
        if (!dateString || [dateString isEqualToString:@"today"]) {
            item.dateDue = [NSDate date];
        } else {
            if (![self isValidDateString:dateString]) {
                [item release];
                return NO;
            }
            // 形式：yyyymmddhhmm（24時間）
            NSString *curStr = [dateString substringWithRange:NSMakeRange(0,
4)];
            NSInteger yeardigit = [curStr integerValue];
            curStr = [dateString substringWithRange:NSMakeRange(4, 2)];
            NSInteger monthdigit = [curStr integerValue];
            curStr = [dateString substringWithRange:NSMakeRange(6, 2)];
            NSInteger daydigit = [curStr integerValue];
            curStr = [dateString substringWithRange:NSMakeRange(8, 2)];
            NSInteger hourdigit = [curStr integerValue];
            curStr = [dateString substringWithRange:NSMakeRange(10, 2)];
            NSInteger minutedigit = [curStr integerValue];

            NSDateComponents *dateComps = [[NSDateComponents alloc] init];
            [dateComps setYear:yeardigit];
            [dateComps setMonth:monthdigit];
            [dateComps setDay:daydigit];
            [dateComps setHour:hourdigit];
            [dateComps setMinute:minutedigit];
            NSCalendar *calendar = [NSCalendar currentCalendar];
            NSDate *itemDate = [calendar dateFromComponents:dateComps];
            if (!itemDate) {
                [dateComps release];
                [item release];
                return NO;
            }
            item.dateDue = itemDate;
            [dateComps release];
        }
```

```
        [(NSMutableArray *)self.list addObject:item];
        [item release];
        return YES;
    }
    return NO;
}
```

アプリケーションに渡されたURLから取得した入力は必ず検証してください。URL処理に関連する問題を回避する方法については、『*Secure Coding Guide*』の「Validating Input」を参照してください。Appleが定義したURLスキームについては、『*Apple URL Scheme Reference*』を参照してください。

アプリケーション環境設定の表示

環境設定を使用して、アプリケーションの動作のさまざまな側面を制御する場合、この環境設定をどのようにユーザに公開するかは、プログラムにとってそれがどのくらい重要かによって異なります。

- アプリケーションを使用するのに不可欠な（また、直接実装するだけで十分な）環境設定は、カスタムインターフェイスを使用して、アプリケーションで直接提示します。
- 不可欠でない環境設定や、比較的複雑なインターフェイスを必要とする環境設定は、システムの「設定(Settings)」アプリケーションを使用して表します。

一連の環境設定が不可欠かどうかを判断するときは、意図した使用形態を考慮します。ユーザが比較的頻繁に環境設定を変更することが予想される場合や、環境設定がアプリケーションの動作の仕方に比較的重要な役割を果たす場合は、おそらくそれらは不可欠です。たとえば、ゲームの設定は、一般的にゲームのプレイに不可欠なので、ユーザはすばやくそれを変更したいと思います。一方、「設定(Settings)」アプリケーションは独立したアプリケーションなので、ユーザが頻繁にはアクセスしないと思われる環境設定のみに使用します。

アプリケーション内に環境設定を実装する場合は、これらの環境設定を管理するためのインターフェイスの定義とコードの記述はデベロッパの責任になります。一方、「設定(Settings)」アプリケーションを使用する場合は、環境設定を管理するためのSettingsバンドルをアプリケーション側が提供しなければなりません。

Settingsバンドルは、アプリケーションのバンドルディレクトリの最上位に含めるカスタムリソースです。`Settings.bundle`という名前の不透過ディレクトリ（Settingsバンドル）は、「設定(Settings)」アプリケーションに環境設定の表示方法を知らせる特別な形式のデータファイル（およびサポートリソース）を含んでいます。このデータファイルは、「設定(Settings)」アプリケーションに、設定結果の値を環境設定データベースのどこに格納するかを知らせる役割もします。環境設定データベースには、`NSUserDefaults`APIまたは`CFPreferences`APIを使用してアプリケーションからアクセスします。

Settingsバンドルを使用して環境設定を実装する場合は、環境設定用のカスタムアイコンも提供する必要があります。「設定(Settings)」アプリケーションは、アプリケーションバンドルの最上位レベルにある`Icon-Settings.png`という名前の画像ファイルを検索し、その画像をアプリケーション名の横に表示します。この画像ファイルは29×29ピクセルのPNG画像ファイルにします。アプリケーションバンドルの最上位レベルにこのファイルを提供しないと、「設定(Settings)」アプリケーションは、デフォルトで、アプリケーションアイコン（`Icon.png`）の縮小版を使用します。

アプリケーション用のSettingsバンドルの作成の詳細については、「アプリケーション環境設定」（191 ページ）を参照してください。

画面ロックの無効化

iPhone OSベースのデバイスでは、指定された時間内にタッチイベントを受信しないと画面がオフになり、タッチセンサーが無効になります。このような画面のロックは、消費電力を節約するための重要な手段です。したがって、アプリケーションで意図しない動作を防ぐためにどうしても必要な場合以外は、この機能は有効にしておくべきです。たとえば、アプリケーションが定期的には画面イベントを受信しないけれどもその他の機能（加速度センサーなど）を入力として使用する場合は、画面ロックを無効にします。

画面ロックのタイマーを無効にするには、共有の`UIApplication`オブジェクトの`idleTimerDisabled`プロパティを`YES`に設定します。アプリケーションで画面ロックを無効にする必要がなくなったときは、必ずこのプロパティを`NO`にリセットしてください。たとえば、ユーザがゲームを行っている最中は画面ロックを無効にしますが、ユーザがセットアップ画面を使用しているときや能動的にゲームを行っているのではないときは、画面ロックを有効に戻します。

アプリケーションの国際化

iPhoneアプリケーションがユーザに表示するテキスト、画像、およびその他のコンテンツは、複数言語にローカライズするのが理想的です。たとえば、警告を促すダイアログが表示するテキストは、ユーザが選択した言語であるべきです。特定の言語に合わせてローカライズされたコンテンツのためのプロジェクトを作成するプロセスを、国際化と呼びます。ローカライズの候補となるプロジェクトコンポーネントには次のものがあります。

- コードが生成するテキスト（日付、時刻、数値表記に関するロケール固有の側面など）
- 静的テキスト（例：アプリケーションのヘルプを表示するためにWebビューにロードされるHTMLファイル）
- アイコン（アプリケーションアイコンを含む）およびその他の画像（テキストが含まれている場合や特定の文化に固有の意味を表現する場合）
- 話し言葉を含むサウンドファイル
- nibファイル

ユーザは、「設定(Settings)」アプリケーションを使用して、アプリケーションのユーザインターフェイスに表示する言語を「言語(Language)」環境設定ビュー（図1-8参照）から選択します。このビューは、「一般(General)」設定の「言語環境(International)」からアクセスできます。

図 1-8　「言語(Language)」環境設定ビュー

選択する言語は、バンドルのサブディレクトリに対応します。サブディレクトリの名前は、ISO 639-1言語コードと.lprojという接尾辞で構成されます。言語コードは、アンダースコアに続けてISO 3166-1の地域指示子を付加することで、特定の地域で修飾できます。たとえば、米国で話されている英語にローカライズされたリソースを指定するには、バンドルの名前はen_US.lprojとなります。ローカライズされた言語のサブディレクトリは、慣例でlprojフォルダと呼ばれます。

注： ISO 639-1で定義されたコードの代わりに、ISO 639-2言語コードを使用することもできます。言語および地域を表すコードについては、『*Internationalization Programming Topics*』の「Language and Locale Designations」を参照してください。

各lprojフォルダには、特定言語（場合によっては、特定の地域）に対応してローカライズされたすべてのコンテンツが格納されます。現在選択されている言語についてローカライズされているリソースの場所（アプリケーションのlprojフォルダのいずれか）を指定するには、NSBundleクラス、またはCFBundleRef不透過型の機能を使用します。リスト 1-3は、英語(en)にローカライズされたコンテンツを格納するディレクトリの例です。

リスト 1-3　ローカライズされた言語のサブディレクトリ

```
en.lproj/
    InfoPlist.strings
    Localizable.strings
    sign.png
```

このサブディレクトリの例には次の項目が存在します。

- InfoPlist.stringsファイルには、プロジェクトのInfo.plistファイル（CFBundleDisplayNameなど）内の特定のプロパティをローカライズした値として割り当てられている文字列が格納されています。たとえば、Battleshipという名前の英語版のアプリケーションを表すCFBundleDisplayNameキーについて、fr.lprojサブディレクトリのInfoPlist.stringsには次のようなエントリがあります。

  ```
  CFBundleDisplayName = "Cuirassé";
  ```

- Localizable.stringsファイルには、アプリケーションコードが生成した文字列をローカライズしたものが格納されます。
- この例のsign.pngファイルは、ローカライズされた画像を格納しているファイルです。

ローカライズのために、コードに含まれる文字列を国際化するには、その文字列の代わりにNSLocalizedStringマクロを使用します。このマクロの宣言は次のとおりです。

```
NSString *NSLocalizedString(NSString *key, NSString *comment);
```

最初のパラメータは、所定のlprojフォルダに配置されたLocalizable.stringsファイルに含まれる、ローカライズされた文字列に対する一意のキーです。2番目のパラメータは、この文字列がどのように使われるかを示すコメントで、翻訳者に追加コンテキストを提供します。たとえば、ユーザインターフェイスにおいて、ラベル（UILabelオブジェクト）の内容を設定するとします。次のコードにより、ラベルのテキストが国際化されます。

```
label.text = NSLocalizedString(@"City", @"Label for City text field");
```

次に、与えられた言語のLocalizable.stringsファイルを作成して、適切なlprojフォルダに追加します。上記のキーの場合、このファイルには次のようなエントリが含まれます。

```
"City" = "Ville";
```

注： また、NSLocalizedString呼び出しをコードの適切な場所に挿入し、genstringsコマンドラインツールを実行することができます。このツールは、翻訳が必要な各文字列のキーとコメントを含んだLocalizable.stringsテンプレートを生成します。genstringsの詳細については、genstrings(1)のmanページを参照してください。

国際化の詳細については、『*Internationalization Programming Topics*』を参照してください。

パフォーマンスと応答性のチューニング

アプリケーション開発の各ステップでは、アプリケーションのパフォーマンス全体に対する設計選択の意味を考慮する必要があります。iPhoneアプリケーションの操作環境は、iPhoneデバイスとiPod touchデバイスのモバイルという性質上、より制約されます。以降のセクションでは、開発プロセスをとおして考慮しなければならない要因について説明します。

メインスレッドを妨害しない

アプリケーションのメインスレッドからどの作業を実行するかについては注意する必要があります。メインスレッドは、アプリケーションがタッチイベントやその他のユーザ入力を処理する場所です。アプリケーションが常にユーザに応答することを保証するには、時間のかかるタスクを実行したり際限なく続く可能性のあるタスク（ネットワークにアクセスするタスクなど）を実行したりするためにメインスレッドを使用すべきではありません。代わりに、これらのタスクは常にバックグラウンドのスレッドに移すべきです。そのために推奨される方法は、各タスクを操作オブジェクト内にラップしてそれを操作キューに追加する方法です。ただし、デベロッパ自身がスレッドを明示的に作成することもできます。

タスクをバックグラウンドに移すことでメインスレッドは解放されてユーザ入力の処理を継続できます。これは、アプリケーションの起動時や終了時には特に重要です。起動時や終了時は、アプリケーションはタイムリーにイベントに応答することを期待されます。起動時にアプリケーションのメインスレッドが機能しないと、起動が完了する前にシステムがアプリケーションを強制終了することもあります。終了時にメインスレッドが機能しないと、重要なユーザデータを書き込む前にシステムがアプリケーションを強制終了することもあります。

操作オブジェクトおよびスレッドの使用の詳細については、『*Threading Programming Guide*』を参照してください。

メモリを効率的に使用する

iPhone OSの仮想メモリモデルには、ディスクのスワップ領域が含まれていないため、アプリケーションには、使用可能なメモリ量について何らかの制限があります。大量のメモリを使用すると、深刻なシステムパフォーマンスの低下が生じ、システムがアプリケーションを強制終了する結果になります。したがって、設計するときは、アプリケーションが使用するメモリ量を削減することに高い優先度を置きます。

利用可能な空きメモリの量と、アプリケーションの相対的なパフォーマンスの間には、直接の相関があります。空きメモリがほとんどないと、システムが将来のメモリ要求を満たすことができない可能性が高くなります。この状態が発生した場合は、システムは常にメモリからコードページとその他の不揮発性リソースを削除できます。ただし、これらのリソースを削除することは一時的な解決にしかなりません。特に、リソースがすぐあとで必要になる場合はそうです。その代わり、まず第一にメモリの使用を最小限にすることと、本当に使用するメモリを適切なタイミングでクリーンアップすることです。

以降の各セクションでは、メモリを効率的に使用する方法と、利用可能なメモリが少量しかない場合の対応方法についての詳細なガイドラインを示します。

アプリケーションのメモリ占有量を削減する

表 1-7に、アプリケーションの全体的なメモリ占有量を削減するためのヒントを示します。少ないメモリ占有量で起動すれば、操作対象のデータにより多くのメモリを割り当てることができます。

表 1-7 アプリケーションのメモリ占有量を削減するためのヒント

ヒント	取るべきアクション
メモリリークをなくす	iPhone OSでは、メモリはきわめて重要なリソースであるため、アプリケーションにメモリリークがあってはなりません。リークがあると、後から必要となるメモリが得られない可能性があります。Instrumentsアプリケーションを使用すると、シミュレータと実際のデバイスの両方で、コード内にあるメモリリークを追跡できます。Instrumentsの使用方法の詳細については、『*Instruments User Guide*』を参照してください。
リソースファイルをできる限り小さくする	ファイルはディスク上に存在しますが、使用する前にはメモリにロードしなければなりません。プロパティリストファイルと画像の2つは、非常に簡単な操作でメモリを節約できる種類のリソースです。プロパティリストファイルが使用するメモリを削減するには、`NSPropertyList-Serialization`クラスを使用して、これらのファイルをバイナリ形式で保存します。画像については、すべての画像ファイルをできる限り小さくなるように圧縮します（PNG画像（iPhoneアプリケーションで推奨される画像形式）を圧縮するには、`pngcrush`ツールを使用します）。
大きなデータセットにはCore DataまたはSQLiteを使用する	アプリケーションで大量の構造化データを扱う場合は、そのデータをフラットなファイルではなく、Core Dataの永続ストアまたはSQLiteデータベースに保存します。Core DataおよびSQLiteは、データセット全体をメモリに一度にロードすることを要求することなく大容量のデータセットを管理する効率的な方法を提供しています。 Core Data機能はiPhone OS 3.0で導入されました。
リソースは後からロードする	実際に必要になるまでは、リソースファイルをロードしてはいけません。リソースファイルを事前にロードすることは時間を節約する方法のように思えますが、実際には、すぐにアプリケーションの速度が遅くなります。また、そのリソースを使用せずに終わった場合は、リソースのロードは単なるメモリの無駄使いです。
Thumbを使用してプログラムをビルドする	コンパイラフラグに`-mthumb`を追加すると、コードサイズを最大35%まで削減できます。ただし、浮動小数点数を多用するコードモジュールでは、Thumbを使用するとパフォーマンスが低下する可能性があるため、このオプションをオフにしてください。

メモリを賢く割り当てる

iPhoneアプリケーションでは、マネージドメモリモデルを使用します。このモデルでは、オブジェクトの保持と解放を明示的に行わなければなりません。表 1-8は、プログラム内でメモリを割り当てる際のヒントを示しています。

表 1-8 メモリ割り当てのヒント

ヒント	取るべきアクション
自動解放オブジェクトの使用を減らす	`autorelease`メソッドを使用して解放されたオブジェクトは、自動解放プールを明示的に空にするか次のイベントループに入るときまで、メモリに残ります。できる限り、`autorelease`メソッドの使用は避けて、代わりに、オブジェクトが占有していたメモリがすぐに解放される`release`メソッドを使用します。ある程度の数の自動解放オブジェクトを作成しなければならない場合は、ローカルの自動解放プールを作成し、定期的にそれを空にすることで、次のイベントループに入る前にこれらのオブジェクトに割り当てられたメモリを解放します。
リソースにサイズの上限を設ける	小さなリソースファイルで済む場合は、大きなリソースファイルのロードは避けます。高解像度の画像を使用する代わりに、iPhone OSベースのデバイスに適したサイズの画像を使用します。大きなリソースファイルを使用しなければならない場合は、その時に必要なファイルの部分だけをロードする方法を見つけます。たとえば、ファイル全体をメモリにロードする代わりに、`mmap`関数と`munmap`関数を使用して、ファイルの一部をメモリにマップしたりアンマップしたりします。メモリへのファイルのマッピングの詳細については、『*File-System Performance Guidelines*』を参照してください。
無限問題セットは避ける	無限問題セットは、際限なく大量のデータの計算を必要とする可能性があります。このセットが、利用可能な量よりも多くのメモリを必要とした場合、アプリケーションは計算を完了できなくなります。このようなセットの使用はできる限り避け、メモリの上限が明確な問題を処理すべきです。

iPhoneアプリケーションでのメモリ割り当て方法の詳細、および自動解放プールの詳細については、『*Cocoa Fundamentals Guide*』の「Cocoa Objects」を参照してください。

浮動小数点演算の留意点

iPhone OSベースのデバイスのプロセッサは、浮動小数点演算をハードウェアで実行することができます。ソフトウェアベースの固定小数点算術ライブラリを使用して計算を実行する既存のプログラムがある場合は、その代わりに浮動小数点算術ライブラリを使用するように変更することを検討すべきです。ハードウェアベースの浮動小数点演算は、一般に、それと同等のソフトウェアベースの固定小数点演算よりもはるかに高速です。

重要： もちろん、コード内で浮動小数点演算を広範囲に使用する場合は、`-mthumb`コンパイラオプションを付けずにコンパイルすることを忘れないでください。Thumbオプションを利用すると、コードモジュールのサイズは削減できますが、浮動小数点コードのパフォーマンスが低下する可能性があります。

電力消費を抑える

モバイルデバイスでの電力消費は、常に問題になります。iPhone OSの電源管理システムは、現在使われていないハードウェア機能をすべて停止することによって電力を節約します。CPUを多用する演算や、高いグラフィックスフレームレートに関与する演算を避けることに加えて、次の機能の使用を最適化することによって、バッテリーの持続時間を向上させることができます。

- CPU
- Wi-Fi無線およびベースバンド(EDGE、3G)無線
- Core Locationフレームワーク
- 加速度センサー
- ディスク

最適化の目的は、できる限り多くの作業を最も効率的な方法で実行することです。InstrumentsおよびSharkを使用して必ずアプリケーションのアルゴリズムを最適化する必要があります。ただし、最も最適化されたアルゴリズムでもデバイスのバッテリー持続時間に悪影響を与える場合があることを忘れないことが重要です。したがってコードを記述する際は、次のガイドラインを考慮する必要があります。

- ポーリングが必要な作業の実行は避ける。ポーリングによってCPUはスリープ状態に入れなくなります。ポーリングの代わりに、`NSRunLoop`クラスまたは`NSTimer`クラスを使用して必要に応じて作業をスケジューリングします。
- 共有の`UIApplication`オブジェクトの`idleTimerDisabled`プロパティは、できる限り`NO`に設定したままにする。アイドルタイマーは、アクティブでない状態が一定時間続くとデバイスの画面をオフにします。アプリケーションで画面をオンにしておく必要がない場合は、システムが画面をオフにするのを許可します。画面がオフになることによってアプリケーションに副次的な影響が生じる場合は、不必要にアイドルタイマーを無効にするのではなく、その副次的な影響がなくなるようにコードを修正すべきです。
- アイドル時間を最大にするためにできる限り作業をまとめる。長時間にわたって定期的に作業を実行する方が、同じ量の作業すべてを一度に実行するよりも電力を消費します。定期的に作業を実行すると、システムはより長い時間ハードウェアをオフにしておくことができません。
- ディスクへの過剰アクセスは避ける。たとえば、アプリケーションが状態情報をディスクに保存する場合は、状態情報が変化したときだけ保存を行いできる限り変更をまとめて小さな変更を頻繁に書き込むことを避けるようにします。
- 必要以上に高速に画面に描画しない。描画は電力の消費においては高価な操作です。フレームレートを抑えるためにハードウェアに依存してはいけません。アプリケーションで実際に必要なフレームのみを描画します。
- `UIAccelerometer`クラスを使用して定期的な加速度センサーイベントを受信する場合、必要がないときはこれらのイベントの配信を無効にする。同様に、イベントの配信頻度をニーズに合った最小値に設定します。詳細については、「加速度センサーイベントへのアクセス」（170 ページ）を参照してください。

ネットワークに伝送するデータが多いほど、無線を実行するために使用する電力は増加します。実際、ネットワークアクセスは、デバイス上で実行できる操作の中で最も電力を消費します。このため、次のガイドラインに従って最小限に抑える必要があります。

- 外部ネットワークサーバには必要なときだけ接続し、サーバのポーリングは行わない。
- ネットワークに接続しなければならない場合は、ジョブを実行するために必要な伝送データ量を最小限に抑える。コンパクトなデータ形式を使用し、無視されるだけの余分なコンテンツは含めないようにします。
- データは、伝送パケットに分けて時間をかけて伝送するのではなく、一括して伝送する。システムは、Wi-Fi無線や携帯電話用無線が使われていないことを検出すると、それをオフにします。同じ量のデータを伝送した場合、伝送時間が長いほど、短い時間で伝送したときよりも多くの電力を消費します。

- できる限りWi-Fi無線を使用してネットワークに接続する。Wi-Fi無線の方が消費電力が少ないため、ベースバンド無線よりもお勧めです。
- Core Locationフレームワークを使用して位置データを収集する場合は、位置の更新をできるだけ早く無効にし、距離フィルタと精度レベルを適切な値に設定する。Core Locationは、利用可能なGPS、携帯電話、およびWi-Fiネットワークを使用して、ユーザの位置を決定します。Core Locationではこれらの無線の使用を最小にするように精一杯努力していますが、Core Locationの精度およびフィルタの値を設定することによって、必要がない場合はハードウェアを完全にオフにすることができます。詳細については、「ユーザの現在位置の取得」（173 ページ）を参照してください。

コードをチューニングする

iPhone OSには、アプリケーションのパフォーマンスをチューニングするためのアプリケーションがいくつか付属しています。これらのツールのほとんどはMac OS Xで動作するため、シミュレータでアプリケーションを実行しながらコードのいくつかの側面をチューニングするのに適しています。たとえば、シミュレータを使用して、メモリリークをなくしたり、全体的なメモリ使用を最小限に抑えるようにしたりできます。また、効率の悪いアルゴリズムや、未知のボトルネックに起因するコード内の処理上のホットスポットを取り除くこともできます。

シミュレータでコードをチューニングした後は、Instrumentsアプリケーションを使用して、デバイス上でさらにコードをチューニングする必要があります。実際のデバイス上でコードを実行することが、コードを完全にチューニングするための唯一の方法です。シミュレータはMac OS Xで動作するので、実際のデバイスよりもCPUは高速で、使用可能なメモリも豊富です。そのため、一般的に、実際のデバイス上でのパフォーマンスよりも、ずっと良い結果になります。Instrumentsを使用して、実際のデバイス上でのコードを追跡することによって、チューニングが必要なパフォーマンスボトルネックがさらに検出される可能性があります。

Instrumentsの使用方法の詳細については、『*Instruments User Guide*』を参照してください。

ウインドウとビュー

ウインドウとビューは、iPhoneアプリケーションのインターフェイスを構成するために使用する、ビジュアルコンポーネントです。ウインドウは、コンテンツを表示するための背景プラットフォームを提供します。一方、ビューは、コンテンツの描画、およびユーザ操作への応答に関する処理のほとんどを実行します。この章では、ウインドウとビューの両方に関連する概念について説明しますが、システムにとって重要な役割を果たすビューの方に重点を置きます。

iPhoneアプリケーションでは、ビューは非常に重要な役割を果たすため、1つの章でそのすべての側面を説明することはできません。この章では、ウインドウとビューの基本的な特性、ウインドウとビューの関係、およびアプリケーションでそれらを作成したり操作したりする方法に焦点を当てて説明します。この章では、ビューがタッチイベントに対してどのように応答するか、またカスタムコンテンツをどのように描画するかについては言及しません。これらの内容の詳細については、「イベント処理」（85 ページ）および「グラフィックスと描画」（107 ページ）を参照してください。

ウインドウとビューとは？

Mac OS Xと同様に、iPhone OSではウインドウとビューを使用してグラフィックコンテンツを画面に表示します。ウインドウオブジェクトとビューオブジェクトは、どちらのプラットフォームにおいても互いに多くの類似点がありますが、ウインドウとビューの両方が果たす役割は、それぞれのプラットフォームで少し異なります。

UIWindowの役割

Mac OS Xアプリケーションとは対照的に、iPhoneアプリケーションは、通常、`UIWindow`クラスのインスタンスで表わされる**ウインドウ**を1つだけ持っています。アプリケーションは、起動時にこのウインドウを作成し（または、nibファイルからウインドウをロードし）、1つ以上のビューをそこに追加して表示します。その後は、このウインドウオブジェクトを再び参照することはほとんどありません。

iPhone OSでは、ウインドウオブジェクトには「閉じる」ボタンやタイトルバーなどの視覚的な装飾はありません。したがって、ユーザが直接ウインドウを閉じたり操作したりすることはできません。ウインドウに対するすべての操作は、プログラムインターフェイスを通して行います。また、アプリケーションは、イベントの送付を円滑に行うためにウインドウを使用します。たとえば、ウインドウオブジェクトは、現在のファーストレスポンダオブジェクトを常に追跡しており、`UIApplication`オブジェクトからの依頼があると、そのレスポンダにイベントを送ります。

経験豊富なMac OS Xデベロッパは、`UIWindow`クラスの継承関係に違和感を覚えるかもしれません。Mac OS Xでは、`NSWindow`の親クラスは`NSResponder`です。iPhone OSでは、`UIWindow`の親クラスは`UIView`です。したがって、iPhone OSでは、ウインドウはビューオブジェクトでもあります。親クラスは異なりますが、一般に、iPhone OSでのウインドウの扱いはMac OS Xと同様です。つまり、通常は`UIWindow`オブジェクトのビュー関連プロパティを直接操作することはありません。

アプリケーションウインドウを作成するとき、ウインドウの最初のフレームサイズは、必ず画面全体を覆うように設定するべきです。nibファイルからウインドウをロードする場合、Interface Builderが、画面サイズより小さいウインドウの作成を許可しません。しかし、プログラミングによってウインドウを作成する場合は、作成時に望みのフレーム矩形を明示的に渡さなければなりません。画面の矩形と異なる矩形は渡さないでください。画面の矩形は、`UIScreen`オブジェクトから次のようにして取得できます。

```
UIWindow* aWindow = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]] autorelease];
```

iPhone OSでは、ウインドウを互いに重ねることが可能ですが、アプリケーションで複数のウインドウを作成するべきではありません。システム自体が、システムステータスバー、重要な警告、その他のメッセージをアプリケーションウインドウの手前に表示するために、追加のウインドウを使用します。アプリケーションコンテンツの手前に警告を表示したい場合は、追加のウインドウを作成するのではなく、UIKitによって提供されている警告ビューを使用します。

UIViewの役割

`UIView`クラスのインスタンスである**ビュー**は、画面上の矩形領域を定義します。iPhoneアプリケーションでは、ユーザインターフェイスの表示と、そのインターフェイスを対象とした操作への応答の両面において、ビューは中心的な役割を果たします。各ビューオブジェクトは、その矩形領域内にコンテンツを表示する責務と、その領域内で発生したタッチイベントに応答する責務を負っています。この2つの振る舞いから、ビューが、アプリケーションにおいてユーザと対話するための主要なメカニズムであることがわかります。Model-View-Controllerアプリケーションでは、ビューオブジェクトは、まさしくアプリケーションのView部分に該当します。

コンテンツの表示とイベント処理のほかに、ビューは、1つ以上のサブビューを管理できます。**サブビュー**とは、元のビューオブジェクト（親ビューまたは**スーパービュー**と呼ばれる）のフレーム内に埋め込まれたビューオブジェクトのことです。ビューは、**ビュー階層**と呼ばれる階層構造になっており、階層には任意の数のビューを含めることができます。また、サブビューにさらにサブビューを追加して、ビューを何階層にもネストさせることができます。各サブビューは、その親ビューの手前に表示されるので、画面への表示はビュー階層内のビューの構成によって決まります。また、この構成によって、ビューがイベントや変更にどのように対応するかも決まります。各親ビューは、その直下のサブビューの管理を担当し、必要に応じてサブビューの位置やサイズを調整したり、サブビューで処理できないイベントに応答したりします。

ビューオブジェクトは、アプリケーションがユーザと対話するための主な手段となるので、たくさんの責務を負っています。以下に、その責務のほんの一部を示します。

- 描画とアニメーション
  - ビューは、その矩形領域にコンテンツを描画します。
  - ビューのプロパティには、新しい値への変化をアニメーション化できるものもあります。
- レイアウトとサブビューの管理
  - ビューは、サブビューのリストを管理します。
  - ビューは、親ビューを基準にして自身のサイズ変更動作を定義します。
  - ビューは、必要に応じて、手動でサブビューのサイズや位置を変更できます。
  - ビューは、そのビューの座標系で表された点を、ほかのビューやウインドウの座標系に変換できます。

- イベント処理
  - ビューはタッチイベントを受け取ります。
  - ビューは、レスポンダチェーンに含まれています。

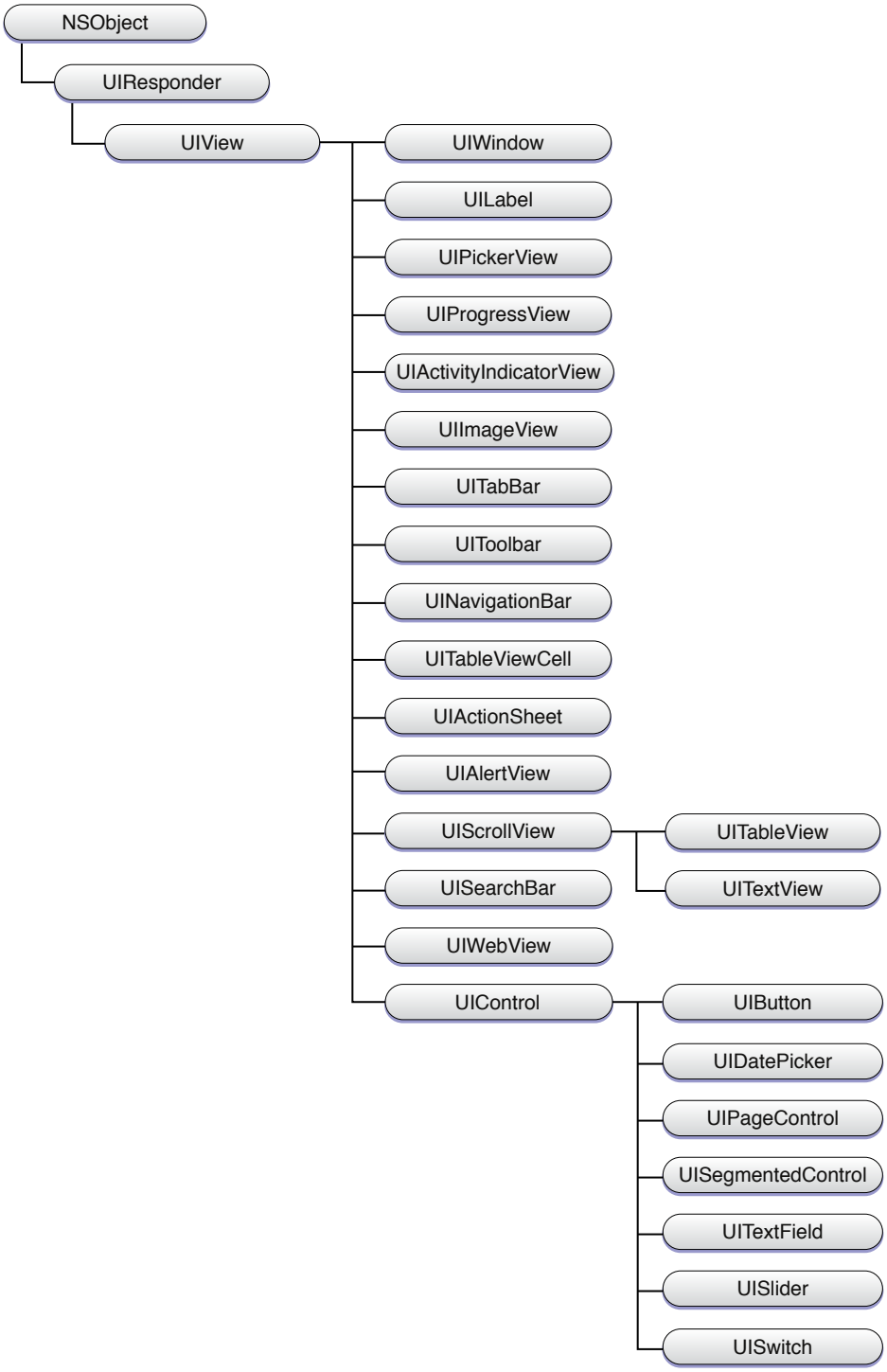
iPhoneアプリケーションでは、ビューは、View Controllerと密接に連携して、ビューの動作のさまざまな側面を管理します。View Controllerは、ビューのロードおよびアンロード、ユーザが物理的にデバイスを回転したことによるインターフェイスの回転、複雑なユーザインターフェイスを構成するために使われる高度なナビゲーションオブジェクトとの対話を処理します。詳細については、「View Controllerの役割」（58 ページ）を参照してください。

この章のほとんどの部分は、これらの責務の説明と、自分のコードを`UIView`の既存の動作と連携させる方法を示すことに当てられています。

UIKitのビュークラス

`UIView`クラスは、ビューの基本プロパティを定義しますが、ビジュアル表現は定義しません。その代わりに、UIKitではサブクラスを使用して、テキストフィールド、ボタン、ツールバーなどの標準的なシステム要素に固有の外観や動作を定義します。図 2-1に、UIKitのすべてのビュークラスの階層図を示します。`UIView`クラスと`UIControl`クラスを除いて、この階層内のほとんどのビューは、そのまま使用するか、デリゲートオブジェクトと組み合わせて使用するように設計されています。

図 2-1 ビュークラスの階層



このビュー階層は、大きく以下のカテゴリに分類できます。

- コンテナ

コンテナビューは、ほかのビューの機能を拡張したり、コンテンツを視覚的に区切ったりする働きをします。たとえば、UIScrollViewクラスは、コンテンツが大きすぎて一度にすべてを画面上に表示できないようなビューを表示するために使用します。UITableViewクラスはUIScrollViewのサブクラスで、データのリストを管理します。テーブルの行は選択可能なため、テーブルは、階層的なナビゲーション（たとえば、オブジェクトの階層を掘り下げる場合など）によく使用されます。

UIToolbarは、特殊なタイプのコンテナで、ボタンのようなアイテムをビジュアルにグループ化します。通常、ツールバーは画面の下端に沿って表示されます。「Safari」、「メール(Mail)」、「写真(Photos)」の各アプリケーションでは、よく使うコマンドを表すボタンを表示するためにツールバーを使用しています。ツールバーは、常に表示しておくことも、アプリケーションで必要なときだけ表示することもできます。

- コントロール

  コントロールは、典型的なアプリケーションユーザインターフェイスのほとんどを作成するために使われます。コントロールは、UIControlスーパークラスを継承する特殊なビューです。コントロールは一般に、特定の値を表示し、その値を変更するために必要なすべてのユーザ操作を処理します。また、コントロールは、ユーザ操作が発生したことを、標準的なシステムパラダイム（ターゲット／アクション、デリゲーションなど）を使用してアプリケーションに通知します。コントロールには、ボタン、テキストフィールド、スライダ、スイッチなどがあります。

- 表示ビュー

  コントロールおよびその他の多くのビューは、対話的な動作を提供しますが、情報を表示するだけのビューもあります。このような動作をするUIKitクラスには、UIImageView、UILabel、UIProgressView、およびUIActivityIndicatorViewなどがあります。

- Text ViewとWeb View

  Text ViewとWeb Viewは、アプリケーション内の複数行のテキストコンテンツを表示するための、より高度な手段を提供します。UITextViewクラスは、複数行のテキストをスクロール可能な領域に表示したり、それらを編集したりする機能をサポートします。UIWebViewクラスは、HTMLコンテンツを表示する手段を提供します。これを利用して、グラフィックスや高度なテキスト書式オプションを組み込んだり、コンテンツを独自の方法でレイアウトできます。

- Alert ViewとAction Sheet

  Alert ViewとAction Sheetは、ユーザの注意をすばやく喚起するために使用します。これらのビューは、ユーザに対してメッセージと共に1つ以上のオプションボタンを提示します。ユーザはこのボタンを使用して、メッセージに応答できます。Alert ViewとAction Sheetは、機能的には似ていますが、外観と動作は異なります。たとえば、UIAlertViewクラスは、画面上にポップアップする青い警告ボックスを表示し、UIActionSheetクラスは、画面の下端からスライドして出てくるボックスを表示します。

- ナビゲーションビュー

  タブバーとナビゲーションバーは、View Controllerと連携して動作し、ユーザインターフェイスのある画面から別の画面へナビゲートするためのツールを提供します。通常は、UITabBarとUINavigationBarの各項目を直接作成することはしません。その代わりに、適切なコントローラインターフェイス、またはInterface Builderを使用してそれらを設定します。

- ウインドウ

  ウインドウは、ほかのすべてのビューのルートコンテナであり、コンテンツの描画サーフェスを提供します。通常、1つのアプリケーションにつきウインドウは1つだけです。詳細については、「UIWindowの役割」（53 ページ）を参照してください。

ビューのほかに、UIKitには、それらのオブジェクトを管理するView Controllerがあります。詳細については、「View Controllerの役割」（58 ページ）を参照してください。

View Controllerの役割

iPhone OSで実行されるアプリケーションには、コンテンツを整理してユーザに表示するためのさまざまなオプションが用意されています。たくさんのコンテンツを含むアプリケーションでは、コンテンツを複数の画面の情報に分割することもできます。実行時には、各画面は、その特定の画面用のデータの表示を担当する一連のビューオブジェクトによって支えられています。1つの画面のビューは、1つのView Controllerオブジェクトによって支えられています。View Controllerの仕事は、ビューに表示するデータを管理することと、更新をアプリケーションのほかの部分と調整することです。

`UIViewController`クラスは、自身が管理する一連のビューを作成したり、メモリ不足状態のときにビューをメモリから削除する責務を負っています。また、View Controllerは、いくつかのシステム動作に自動的に応答します。たとえば、デバイスの向きが変更になったことに応答して、View Controllerは、その向きがサポートされている場合は、新しい向きに適合するように自身が管理するビューのサイズを変更します。また、View Controllerを使用して、現在のビューの前面に新規のモーダルビューを表示することもできます。

`UIViewController`基本クラスのほかに、UIKitには、このプラットフォームに共通する複雑なインターフェイス動作を処理する、より高度なサブクラスが含まれています。特に、ナビゲーションコントローラは、階層的な複数の画面におよぶコンテンツの表示を管理します。タブバーコントローラは、ユーザがアプリケーションの異なる操作モードを表す画面セット間を切り替えることを可能にします。

View Contollerを使用してユーザインターフェイス内のビューを管理する方法については、『*View Controller Programming Guide for iPhone OS*』を参照してください。

ビューのアーキテクチャとジオメトリ

ビューは、iPhoneアプリケーションの中心的なオブジェクトなので、ビューがシステムのほかの部分とどのように相互に作用するかを、少し理解しておくことは重要です。UIKitの標準ビュークラスは、かなりの数の動作をアプリケーションに無償で提供します。また、これらのビュークラスは、その動作をカスタマイズして、アプリケーションに必要な処理を実行するための、明確に定義された統合点を備えています。

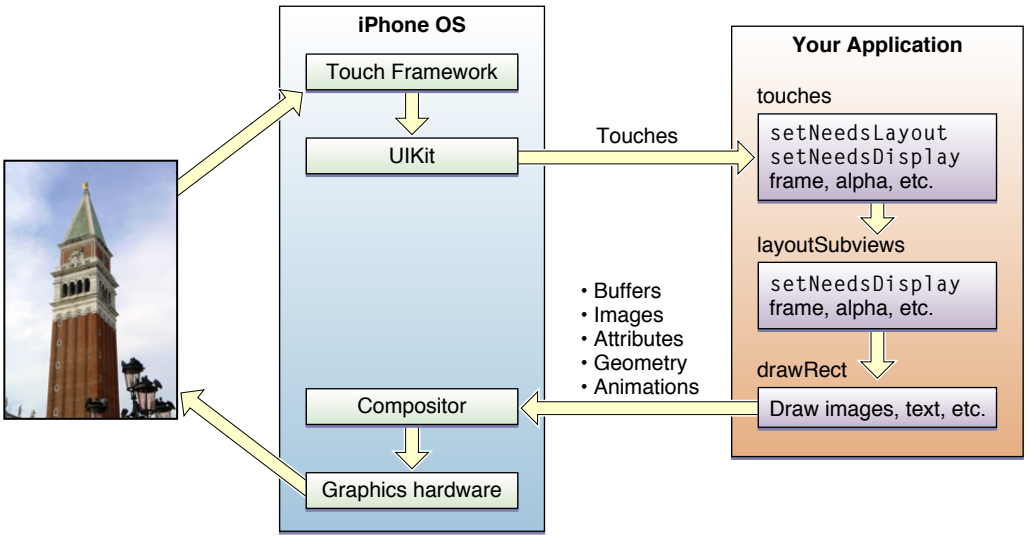
以降の各セクションでは、ビューの標準的な動作について説明し、カスタムコードと統合できる場所を示します。特定のクラスの統合点の詳細については、そのクラスのリファレンス文書を参照してください。クラスリファレンス文書の網羅的なリストは、『*UIKit Framework Reference*』から入手できます。

ビューの対話モデル

ユーザがユーザインターフェイスを操作したり、自分のコードでプログラムの中で何らかの変更を加えたりしたときは、それらの操作を処理するために、UIKitの内部で複雑なイベントシーケンスが発生します。そのシーケンス内の特定の時点で、UIKitは、ビュークラスを呼び出して、アプリケーションとして応答を行う機会をビューに与えます。これらの呼び出し点を理解することは、ビュー

をシステムに適合させる場所を理解するために重要です。図 2-2は、ユーザが画面に触れることによって始まり、それに応じてグラフィックスシステムが画面内容を更新することによって終わる、基本的なイベントシーケンスを示します。プログラムによるイベントも、最初のユーザ操作以外は、同じ基本ステップをたどります。

図 2-2 UIKitとビューオブジェクトとの対話



以下のステップは、図 2-2 （59 ページ）のイベントシーケンスをさらに細分化して、各ステージで何が発生し、それに応答してアプリケーションでどのように反応できるかを示します。

1. ユーザが画面に触れます。
2. ハードウェアは、そのタッチイベントをUIKitフレームワークに報告します。
3. UIKitフレームワークは、そのタッチを1つの`UIEvent`オブジェクトにまとめて、適切なビューに送付します（UIKitがイベントをビューに送付する方法の詳細については、「イベントの送付」 （87 ページ）を参照してください）。
4. ビューのイベント処理メソッドは、以下のいずれかを実行して、そのイベントに応答します。
   - ビューまたはサブビューのプロパティ（frame、bounds、alphaなど）を調整する。
   - ビュー（またはサブビュー）に、レイアウトの変更が必要であるというマークを付ける。
   - ビュー（またはサブビュー）に、再描画が必要であるというマークを付ける。
   - データの一部が変更されたことをコントローラに通知する。

   もちろん、これらの処理のどれを実行すべきかを判断し、それを実行するための適切なメソッドを呼び出すのは、ビューの責任です。
5. レイアウトが必要であるというマークがビューに付いている場合、UIKitは、そのビューの`layoutSubviews`メソッドを呼び出します。

   カスタムビューでこのメソッドをオーバーライドし、それを使用して任意のサブビューの位置とサイズを調整できます。たとえば、大きなスクロール領域を提供するビューでは、メモリに収まる可能性の低い大きなビューを1つ作成するのではなく、複数のサブビューを「タイル」の

ようにして使用する必要があります。このメソッドの実装の中で、ビューは、現在オフスクリーンにあるサブビューを非表示にしたり、サブビューを配置しなおしたりします。また、サブビューを使用して新たに表示するコンテンツを描画します。この処理の一部として、ビューは、新しいタイルに再描画が必要であるというマークを付けることもできます。

6. ビューの一部に再描画が必要であるというマークが付いている場合、UIKitは、そのビューの`drawRect:`メソッドを呼び出します。

   UIKitは、再描画が必要なビューに対してのみ、このメソッドを呼び出します。各ビューのこのメソッドの実装では、指定された領域をできる限り迅速に再描画しなければなりません。各ビューは、そのビューに固有のコンテンツのみを描画し、サブビューのコンテンツは描画しません。この時点で、ビューは、プロパティやレイアウトにそれ以上の変更を加えようとしてはなりません。

7. 更新されたビューは、残りの表示コンテンツと合成され、表示のためにグラフィックスハードウェアに送られます。

8. グラフィックスハードウェアは、レンダリングされたコンテンツを画面に転送します。

注： 上で述べた更新モデルは、主に、ネイティブなビューおよび描画テクノロジーを使用するアプリケーションに適用されます。アプリケーションが、OpenGL ESを使用してコンテンツを描画する場合は、通常、フルスクリーンビューを1つ設定して、OpenGLのグラフィックスコンテキストに直接描画します。ビューは依然としてタッチイベントを処理しますが、サブビューをレイアウトしたり、`drawRect:`メソッドを実装する必要がありません。OpenGL ESの使用の詳細については、「OpenGL ESを使用した描画」（117 ページ）を参照してください。

前述の一連のステップを踏まえ、カスタムビューのための主な統合点は以下のとおりです。

1. 以下のイベント処理メソッド：
   - `touchesBegan:withEvent:`
   - `touchesMoved:withEvent:`
   - `touchesEnded:withEvent:`
   - `touchesCancelled:withEvent:`
2. `layoutSubviews`メソッド
3. `drawRect:`メソッド

これらは、ほとんどのカスタムビューで、独自の動作を実行するために実装するメソッドです。ただし、これらすべてをオーバーライドする必要はありません。たとえば、サイズ変更が生じないビューを実装する場合は、`layoutSubviews`メソッドをオーバーライドする必要はありません。同様に、テキストや画像などの単純なコンテンツを表示するビューを実装する場合は、`UIImageView`オブジェクトや`UILabel`オブジェクトをサブビューとして組み込むだけで、描画をまったく行わずに済む場合もあります。

これらは、主な統合点であって、統合点はこれだけではないということを忘れないでください。`UIView`クラスのメソッドのなかには、サブクラスでオーバーライドされることを前提に設計されているものもあります。カスタム実装でオーバーライドするのに適したメソッドであるかどうかを判断するには、『*UIView Class Reference*』に記載されているメソッドの説明を参照してください。

ビューのレンダリングアーキテクチャ

画面上にコンテンツを表示するにはビューを使用しますが、`UIView`クラス自体は、実際にはその基本的な動作のほとんどを別のオブジェクトに依存しています。UIKitのビューオブジェクトはすべて、Core Animationレイヤオブジェクト（`CALayer`クラスのインスタンス）に支えられています。このレイヤクラスは、ビューのコンテンツのレイアウトとレンダリング、およびそのコンテンツの合成とアニメーション化の基本部分をサポートします。

Mac OS X（Core Animationのサポートはオプション）とは対照的に、iPhone OSでは、Core Animationがビューのレンダリング実装の中心に組み込まれています。Core Animationは中心的な役割を果たしますが、UIKitは、Core Animationの最上位に透過的なレイヤを提供して、プログラミング作業を効率化します。この透過的なレイヤによって、Core Animationレイヤに直接アクセスする必要はほとんどなくなります。その代わりに、`UIView`クラスのメソッドとプロパティを使用して、同様の動作にアクセスできます。Core Animationが重要になるのは、`UIView`クラスではアプリケーションのニーズをすべて満たすことができない場合です。その場合は、Core Animationレイヤの内部に入り込んで、アプリケーションのために非常に高度なレンダリングを行うことができます。

以降の各セクションでは、Core Animationの基礎的な情報を提供し、`UIView`クラスを通じて無償で提供されるいくつかの機能について説明します。Core Animationを使用して高度なレンダリングを行う方法の詳細については、『*Core Animation Programming Guide*』を参照してください。

Core Animationの基礎

Core Animationは、ハードウェアの高速性と最適化されたアーキテクチャを利用して、高速なレンダリングとリアルタイムアニメーションを実現します。ビューの`drawRect:`メソッドが初めて呼び出されたときに、レイヤはその結果をキャプチャしてビットマップに変換します。その後の再描画呼び出しでは、このキャッシュされたビットマップが使用され、負荷がかかる`drawRect:`メソッドの呼び出しはできる限り回避されます。このプロセスによって、Core Animationは、合成操作を最適化して望みどおりのパフォーマンスを実現します。

Core Animationは、ビューオブジェクトに関連付けられたレイヤを、**レイヤツリー**と呼ばれる階層に格納します。ビューと同様に、レイヤツリー内の各レイヤは、1つの親を持ち、任意の数のサブレイヤを組み込むことができます。デフォルトでは、レイヤツリー内のオブジェクトは、ビュー階層内のビューとまったく同じ構成になります。ただし、対応するビューを追加せずに、レイヤを追加することもできます。ビューを必要としない特殊な視覚効果を実装するために、このような追加を行う場合があります。

レイヤオブジェクトは、iPhone OSのレンダリングおよびレイアウトシステムの背後で、実際にそれを推進する働きをします。ほとんどのビュープロパティは、実際には、この基盤レイヤオブジェクトのプロパティを包む薄いラッパーです。（`CALayer`オブジェクトを直接使用して）レイヤツリー内のレイヤのプロパティを変更した場合は、変更した値が、すぐにそのレイヤオブジェクトに反映されます。ただし、変更がアニメーションをトリガする場合、その変更はすぐには画面に反映されず、アニメーションによって一定の時間をかけて画面に反映されます。この種のアニメーションを管理するために、Core Animationは、**プレゼンテーションツリー**と**レンダリングツリー**という2つのレイヤオブジェクトのセットを保持しています。

プレゼンテーションツリーは、レイヤの状態を、現在ユーザに提示されているとおりに反映します。レイヤ値の変更をアニメーション化する場合は、プレゼンテーションレイヤには、アニメーションが始まるまで古い値が反映されます。アニメーションが進むにつれて、Core Animationは、アニメーションの現在のフレームに基づいてプレゼンテーションツリーレイヤの値を更新します。レンダリングツリーは、プレゼンテーションツリーと連動して、画面上の変化をレンダリングしま

す。レンダリングツリーは、独立のプロセスまたはスレッドで実行されるので、その処理は、アプリケーションのメイン実行ループに影響を与えません。レイヤツリーとプレゼンテーションツリーは共に公開(public)のAPIですが、レンダリングツリーは非公開(private)のAPIです。

ビューの背後にあるレイヤオブジェクトの配置は、描画コードのパフォーマンスに重要な影響を与えます。レイヤを使用することの有利な点は、ビューに対するジオメトリ変更のほとんどを、再描画する必要がないことです。たとえば、ビューの位置とサイズを変更しても、システムはビューのコンテンツを再描画する必要はありません。レイヤによって作成されたキャッシュ済みのビットマップを再利用するだけで済みます。このキャッシュされたコンテンツをアニメーション化すれば、毎回そのコンテンツを再描画するのに比べて大幅な効率化になります。

レイヤを使用することの不利な点は、キャッシュデータが増えることで、アプリケーションのメモリが圧迫される可能性があることです。アプリケーションが作成するビューが多すぎたり、非常に大きかったりすると、すぐにメモリ不足になります。アプリケーション内でビューを使用することを控える必要はありませんが、既存のビューを再利用できる場合は、新規のビューを作成しないようにします。言い換えれば、同時にメモリ内に保持するビューの数を最小にするようなアプローチを追求しましょう。

Core Animationの概要、オブジェクトツリー、およびアニメーションの作成方法の詳細については、『*Core Animation Programming Guide*』を参照してください。

ビューのレイヤの変更

iPhone OSでは、ビューは対応するレイヤを1つ持つ必要があります。このため、`UIView`クラスは初期化時にこのレイヤを自動的に作成します。このようにして作成されたレイヤには、ビューの`layer`プロパティを介してアクセスできます。ただし、ビューが作成された後でこのレイヤオブジェクトを変更することはできません。

ビューで異なる種類のレイヤを使用するには、そのビューの`layerClass`クラスメソッドをオーバーライドして、ビューで使用したいレイヤのクラスオブジェクトを返す必要があります。異なるレイヤクラスを返すようにする理由として一番よくあるのは、OpenGLベースのアプリケーションを実装するためです。OpenGLの描画コマンドを使用するには、基盤となるビューのレイヤは`CAEAGLLayer`クラスのインスタンスでなければなりません。この種類のレイヤは、OpenGLのレンダリング呼び出しとやり取りをして、望みのコンテンツを画面に表示します。

重要： ビューのレイヤの`delegate`プロパティを変更してはいけません。このプロパティはビューへのポインタを格納しており、非公開と考えるべきです。同様に、ビューは、1つだけのレイヤのデリゲートとして動作するので、ビューをほかのレイヤオブジェクトのデリゲートとして割り当ててはいけません。このようなことを行うと、アプリケーションがクラッシュする原因になります。

アニメーションのサポート

iPhone OSのすべてのビューの背後にレイヤオブジェクトを持つことの利点の1つは、コンテンツをアニメーション化しやすくなることです。アニメーションは、必ずしも視覚的な楽しみを演出するためのものではありません。アニメーションは、アプリケーションのユーザインターフェイスで生じた変更のコンテキストをユーザに提供します。たとえば、トランジションを使用して、ある画面から別の画面に移動すると、これらの画面に関連があることをユーザに示すことができます。システムは、最もよく使われるアニメーションの多くを自動的にサポートします。しかし、インターフェイスのその他の部分のためにアニメーションを作成することもできます。

UIViewクラスのプロパティの多くは、アニメーション化できます。**アニメーション化可能な**プロパティとは、ある値から別の値へのアニメーション化が半自動的にサポートされているプロパティです。その場合も、UIKitにアニメーションの実行を指示する必要があります。しかし、アニメーションが開始されると、それを実行するすべての責任はCore Animationが負います。UIViewオブジェクトでアニメーション化可能なプロパティには、以下のものが含まれます。

```
frame
bounds
center
transform
alpha
```

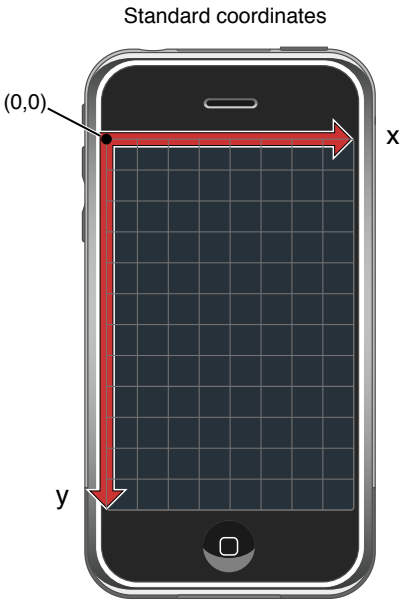
その他のビュープロパティは、直接アニメーション化できませんが、明示的にアニメーションを作成できるものもあります。明示的なアニメーションを作成するには、アニメーションとレンダリングされるコンテンツの管理にかかわる処理の多くを自分で行う必要があります。ただし、その場合も、すぐれたパフォーマンスを得るために基盤となるCore Animationインフラストラクチャを使用します。

UIViewクラスを使用したアニメーションの作成の詳細については、「ビューのアニメーション化」（76 ページ）を参照してください。明示的なアニメーションの作成の詳細については、『*Core Animation Programming Guide*』を参照してください。

ビューの座標系

UIKitの座標は、左上隅を原点とし、下方向と右方向に伸びる座標軸をもつ座標系に基づいています。座標の値は、浮動小数点数を使用して表されます。これによって、コンテンツの正確なレイアウトと位置決めが可能になり、解像度からの独立性を維持できます。図 2-3 （64 ページ）は、この座標系を、画面を基準にして示します。ただし、この座標系は、UIWindowクラスおよびUIViewクラスでも使用されます。この特殊な向きは、QuartzやMac OS Xで使われているデフォルトの座標系とは異なりますが、ユーザインターフェイスにコントロールとコンテンツを配置しやすくするために選択されました。

図 2-3 ビューの座標系



インターフェイスのコードを記述する際には、現在有効になっている座標系に注意します。どのウインドウオブジェクトやビューオブジェクトも、固有のローカルな座標系を持っています。ビュー内での描画はすべて、そのビューのローカルな座標系に基づいて行われます。ただし、各ビューのフレーム矩形は、その親ビューの座標系を使用して指定します。また、イベントオブジェクトの一部として送付される座標は、それを含んでいるウインドウの座標系に基づいています。便利なように、`UIWindow`クラスと`UIView`クラスは、異なるオブジェクトの座標系の間で相互に変換を行うメソッドを備えています。

Quartzで使用される座標系は、左上隅を原点として使用しませんが、ほとんどのQuartz呼び出しではそれは問題になりません。ビューの`drawRect:`メソッドを呼び出す前に、UIKitが、左上隅を原点として使用するように、自動的に描画環境を設定します。この環境内で行われたQuartz呼び出しは、ビュー内に正しく描画されます。座標系の違いを考慮する必要があるのは、Quartzを使用して描画環境を自分でセットアップするときだけです。

座標系、Quartz、および描画全般の詳細については、「グラフィックスと描画」（107 ページ）を参照してください。

フレーム(frame)、境界(bounds)、および中心(center)の関係

ビューオブジェクトは、`frame`、`bounds`、および`center`の各プロパティを使用して、そのサイズと位置を決定します。`frame`（フレーム）プロパティは、親ビューの座標系に基づいて、そのビューの位置とサイズを定義した矩形（**フレーム矩形**）を表します。`bounds`（境界）プロパティは、ビュー固有のローカルな座標系に基づいて、ビューの位置とサイズを定義した矩形（**境界矩形**）を表します。境界矩形の原点は、通常(0, 0)に設定されますが、必ずしもその必要はありません。`center`（中心）プロパティは、フレーム矩形の**中心点**を表します。

`frame`、`bounds`、および`center`の各プロパティは、コード内で異なる目的で使用します。境界矩形は、そのビューのローカルな座標系を表すので、描画やイベント処理のコードの中で、何らかの変化がビュー内のどこで発生したかを知る必要があるときに、最もよく使います。中心点は、ビュー

の既知の中心点を表すので、ビューの位置を操作するのに最適な方法です。フレーム矩形は、boundsとcenterを使用して計算される便利な値です。これは、ビューの変換が、恒等変換に設定されている場合にのみ有効です。

図 2-4に、フレーム矩形と境界矩形の関係を表します。右側の完全な画像は、(0, 0)を起点とするビューに描画されています。一方、境界のサイズが、画像全体のサイズと一致しないため、境界矩形の外側にある画像部分だけが、自動的にクリッピングされます。このビューが親ビューと合成されると、親ビュー内のビューの位置は、そのビューのフレーム矩形の原点（この例では、(5, 5)）によって決まります。その結果、このビューのコンテンツは、親ビューの原点から右下にずれた場所に表示されます。

図 2-4　ビューのフレームと境界の関係

ビューに対して変形がいっさい適用されない場合、ビューの位置とサイズは、上記の相互に関連する3つのプロパティによって決まります。ビューのframeプロパティは、initWithFrame:メソッドを使用してビューオブジェクトをプログラミングによって作成した場合に設定されます。また、このメソッドは、bounds矩形を初期化して、(0.0, 0.0)を原点として、ビューのフレームと同じサイズになるようにします。次に、centerプロパティが、そのフレームの中心点に設定されます。

これらのプロパティを別々に設定することもできますが、値の1つを設定すると、それ以外の値は次のように変化します。

- frameプロパティを設定すると、boundsプロパティのサイズは、frameプロパティのサイズと一致するように設定されます。また、centerプロパティは、その新しいフレームの中心点に一致するように調整されます。
- centerプロパティを設定すると、それに応じて、frameプロパティの原点が変更されます。
- bounds矩形のサイズを設定すると、frame矩形のサイズは、それに一致するように変更されます。

ほかの2つのプロパティを変えずに、boundsの原点を変更することもできます。その場合、ビューには、すでに確認している基盤になる画像の一部が表示されます。図 2-4 （65 ページ）では、元のboundsの原点は(0.0, 0.0)に設定されています。図 2-5では、この原点が(8.0, 24.0)に移動されています。その結果、基盤になる画像の別の部分がビューに表示されます。ただし、フレーム矩形は変化していないので、新しいコンテンツは、親ビュー内の以前と同じ位置に表示されます。

図 2-5 ビューの境界の変更

注： デフォルトでは、ビューのフレームは、親ビューのフレームに合わせてクリッピングされません。サブビューを強制的にクリッピングするには、親ビューの`clipsToBounds`プロパティを`YES`に設定します。

座標系の変換

座標系の変換は、描画をしやすくするために、ビューの`drawRect:`メソッドでよく使用されますが、iPhone OSでは、ビューに対する視覚効果を実現するために使用することもできます。たとえば、`UIView`クラスには、さまざまな種類の変換、拡大縮小効果などをビュー全体に適用できるようにする`transform`プロパティがあります。デフォルトでは、このプロパティの値はビューの変更が行われない恒等変換に設定されています。変換を追加するには、このプロパティに格納されている`CGAffineTransform`構造体を取得し、該当するCore Graphics関数を使用して変換を適用します。その後、変更を加えた変換構造体をビューの`transform`プロパティに代入し直します。

注： 変換をビューに適用すると、すべての変換はそのビューの中心点を基準にして実行されます。

ビューを変換すると、そのビューのコンテンツの描画に伴ってすべてのサブビューが移動します。サブビューの座標系にもこれらの変換が継承されるため、拡大縮小はサブビューの描画にも影響します。ビューのコンテンツの拡大縮小の制御方法の詳細については、「コンテンツモードと拡大縮小」（67 ページ）を参照してください。

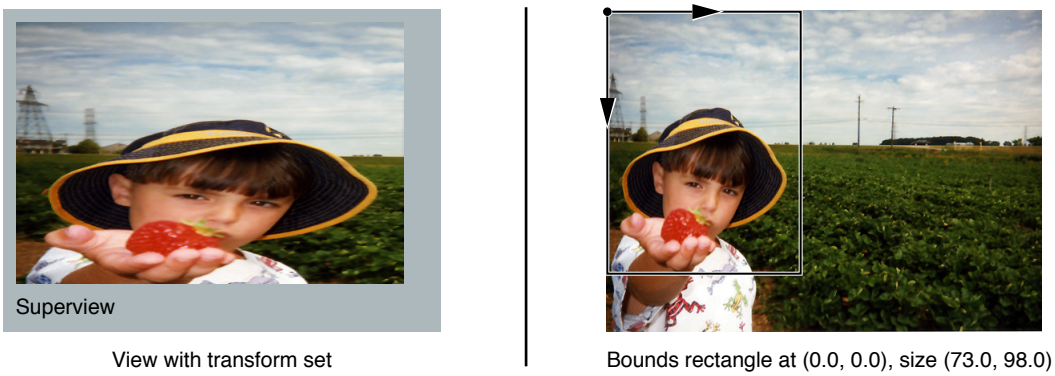
重要： `transform`プロパティが恒等変換でない場合は、`frame`プロパティの値は未定義となるため、無視する必要があります。変換を設定した後は、`bounds`プロパティと`center`プロパティを使用して、ビューの位置とサイズを取得します。

`drawRect:`メソッドと組み合わせて変換を使用する方法については、「座標と座標変換」（108 ページ）を参照してください。`CGAffineTransform`構造体を変更するために使用する関数については、『*CGAffineTransform Reference*』を参照してください。

コンテンツモードと拡大縮小

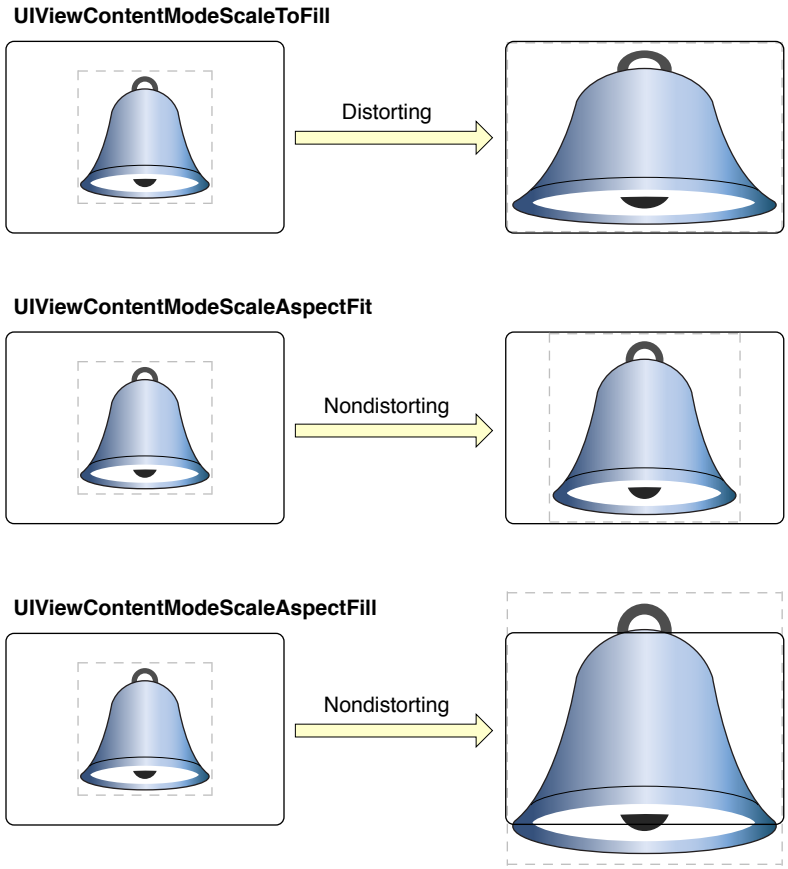
ビューの境界を変更したり、ビューの`transform`プロパティに倍率を適用した場合、フレーム矩形もそれに応じた量だけ変化します。ビューに関連付けられているコンテンツモードによって、そのビューのコンテンツが、変更に対応するために拡大縮小されたり、再配置される場合もあります。ビューの`contentMode`プロパティによって、境界の変化および拡大縮小操作がビューに与える影響が決まります。デフォルトでは、このプロパティの値は`UIViewContentModeScaleToFill`に設定されています。このため、ビューのコンテンツは、常に新しいフレームサイズに合わせて拡大縮小されます。たとえば、図 2-6は、ビューの水平方向の倍率を2倍にしたときの様子を示します。

図 2-6 scale-to-fillコンテンツモードを使用して拡大されたビュー



初めてビューが表示されたときに、レンダリングされたコンテンツが基盤レイヤにキャッシュされているので、このビューのコンテンツは拡大されます。ビューの境界が変化したり、倍率が適用されるたびに、強制的にビューそのものを再描画する代わりに、UIKitは、そのビューのコンテンツモードを使用して、キャッシュされているコンテンツの表示方法を決定します。図 2-7は、異なるコンテンツモードを使用して、ビューの境界を変更したり、ビューに倍率を適用したりした結果を比較したものです。

図 2-7 コンテンツモードの比較



拡大縮小倍率を適用すると、ビューのコンテンツは常に拡大縮小されますが、ビューの境界が変化したときには、ビューのコンテンツを拡大縮小しないコンテンツモードもあります。`UIViewContentMode`定数の中には、`UIViewContentModeTop`や`UIViewContentModeBottomRight`のように、現在のコンテンツを、ビューの別の角や、別の辺に沿って表示するものもあります。コンテンツをビューの中央に表示するモードもあります。これらのコンテンツモードのいずれかに設定した状態で境界矩形を変更すると、既存のコンテンツは、単純に新しい境界矩形の適切な場所に移動するだけです。

アプリケーション内にサイズ変更可能なコントロールを実装する場合は、コンテンツモードの使用を検討します。これによって、コントロールのゆがみを回避できると同時に、カスタム描画コードを記述しなくても済みます。ボタンや区切りのあるコントロールは、コンテンツモードに基づく描画に特に適しています。通常、これらは複数の画像を使用してコントロールの外観を作成します。水平方向に伸縮するボタンでは、固定サイズの2つの終端画像のほかに、1ピクセルの幅の伸縮可能な中央画像を使用します。各画像をそれぞれのビューに表示し、伸縮可能な中央画像のコンテンツモードを`UIViewContentModeScaleToFill`に設定することによって、両端画像をゆがめることなく、ボタンのサイズを大きくできます。さらに重要なことは、各画像ビューに関連付けられている画像はCore Animationによってキャッシュされるので、独自の描画コードを書かなくてもアニメーション化できる点です。その結果パフォーマンスも向上します。

コンテンツモードは、ビューのコンテンツの再描画を避けるための優れた方法ですが、拡大縮小やサイズ変更の操作中にビューの外観を特別に制御したい場合は、UIViewContentModeRedrawコンテンツモードを使用することもできます。ビューのコンテンツモードをこの値に設定すると、コンテンツの拡大縮小やサイズ変更を自動的に行う代わりに、強制的にビューのコンテンツを無効して、ビューのdrawRect:メソッドを呼び出すように、Core Animationに指示することができます。

自動サイズ変更動作

ビューのフレーム矩形を変更した場合、通常は、それに含まれているサブビューの位置とサイズも、親ビューの新しいサイズに合わせて変更する必要があります。ビューのautoresizesSubviewsプロパティがYESに設定されている場合、そのビューのサブビューは、autoresizingMaskプロパティの値に応じて、自動的にサイズ変更されます。多くの場合、ビューに対して自動サイズ変更マスクを設定するだけで、アプリケーションのための適切な動作が提供されます。適切な動作が提供されない場合は、layoutSubviewsメソッドをオーバーライドして、アプリケーションの責任で、サブビューの位置とサイズを変更する必要があります。

ビューの自動サイズ変更動作を設定するには、ビット単位のORを使用して望みの自動サイズ変更定数を組み合わせて、その結果をビューのautoresizingMaskプロパティに代入します。表2-1に、自動サイズ変更定数の一覧を示し、それぞれの定数がビューのサイズと配置に与える効果を説明します。たとえば、ビューを親ビューの左下隅に固定するには、UIViewAutoresizingFlexibleRightMargin定数とUIViewAutoresizingFlexibleTopMargin定数を追加し、それをautoresizingMaskプロパティに代入します。ある軸に沿って複数の側面を可変にした場合、サイズ変更の量はそれぞれの側面に均等に配分されます。

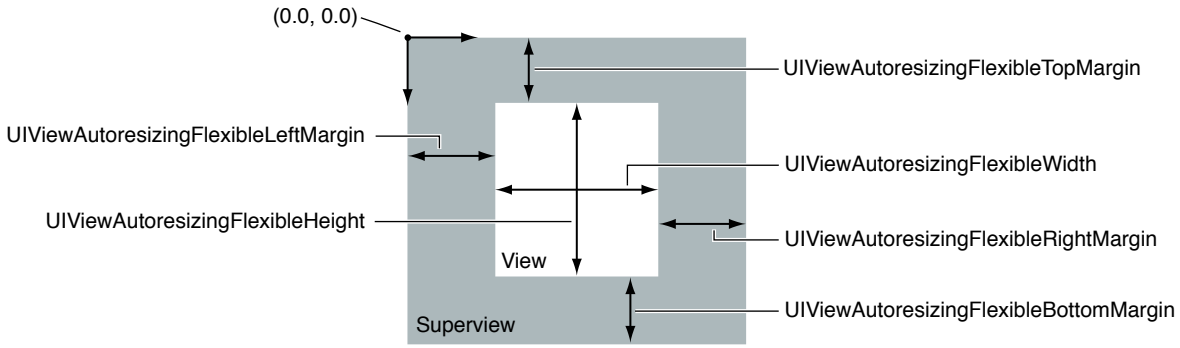
表 2-1　自動サイズ変更マスク定数

自動サイズ変更マスク	説明
UIViewAutoresizingNone	この定数を設定すると、ビューの自動サイズ変更は行われません。
UIViewAutoresizing-FlexibleHeight	この定数を設定すると、ビューの高さは、スーパービューの高さの変化に比例して変更されます。設定しない場合、スーパービューの高さの変化に比例したビューの高さの変更は行われません。
UIViewAutoresizing-FlexibleWidth	この定数を設定すると、ビューの幅は、スーパービューの幅の変化に比例して変更されます。設定しない場合、スーパービューの幅の変化に比例したビューの幅の変更は行われません。
UIViewAutoresizing-FlexibleLeftMargin	この定数を設定すると、ビューの左端は、スーパービューの幅の変化に比例して再配置されます。設定しない場合、ビューの左端とスーパービューの左端との距離は変更されません。
UIViewAutoresizing-FlexibleRightMargin	この定数を設定すると、ビューの右端は、スーパービューの幅の変化に比例して再配置されます。設定しない場合、ビューの右端とスーパービューの右端との距離は変更されません。
UIViewAutoresizing-FlexibleBottomMargin	この定数を設定すると、ビューの下端は、スーパービューの高さの変化に比例して再配置されます。設定しない場合、ビューの下端とスーパービューの下端との距離は変更されません。

自動サイズ変更マスク	説明
`UIViewAutoresizing-FlexibleTopMargin`	この定数を設定すると、ビューの上端は、スーパービューの高さの変化に比例して再配置されます。設定しない場合、ビューの上端とスーパービューの上端との距離は変更されません。

図 2-8に、この定数値の位置を図で表します。これらの定数の1つが省略された場合、ビューのレイアウトはその側面については固定されます。定数がマスクに含まれていると、ビューのレイアウトはその側面については可変になります。

図 2-8 ビューの自動サイズ変更マスク定数



Interface Builderを使用して、ビューを設定する場合は、「Size」インスペクタの「Autosizing」コントロールを使用して、各ビューの自動サイズ変更動作を設定できます。上の図の可変の幅と高さを表す定数は、Interface Builderの同じ場所にあるバネと同じ動作をします。マージン関係の定数は、それと逆の動作をします。つまり、可変の右マージン自動サイズ変更動作を、Interface Builder内のビューに適用するには、「Autosizing」コントロールの右側のスペースを空のままにし、そこにストラット（柱）を置かないようにする必要があります。幸い、Interface Builderは、自動サイズ変更動作の変更が、ビューにどのような影響を与えるかをアニメーションとして示します。

ビューの`autoresizesSubviews`プロパティが`NO`に設定されている場合は、そのビューの直下のサブビューに設定されている自動サイズ変更動作はすべて無視されます。同様に、サブビューの自動サイズ変更マスクが`UIViewAutoresizingNone`に設定されている場合は、そのサブビューのサイズは変化しません。したがって、その直下のサブビューのサイズも変更されません。

注： 自動サイズ変更が正しく動作するには、ビューの`transform`プロパティが恒等変換に設定されていなければなりません。恒等変換に設定されていない場合、自動サイズ変更動作は未定義となります。

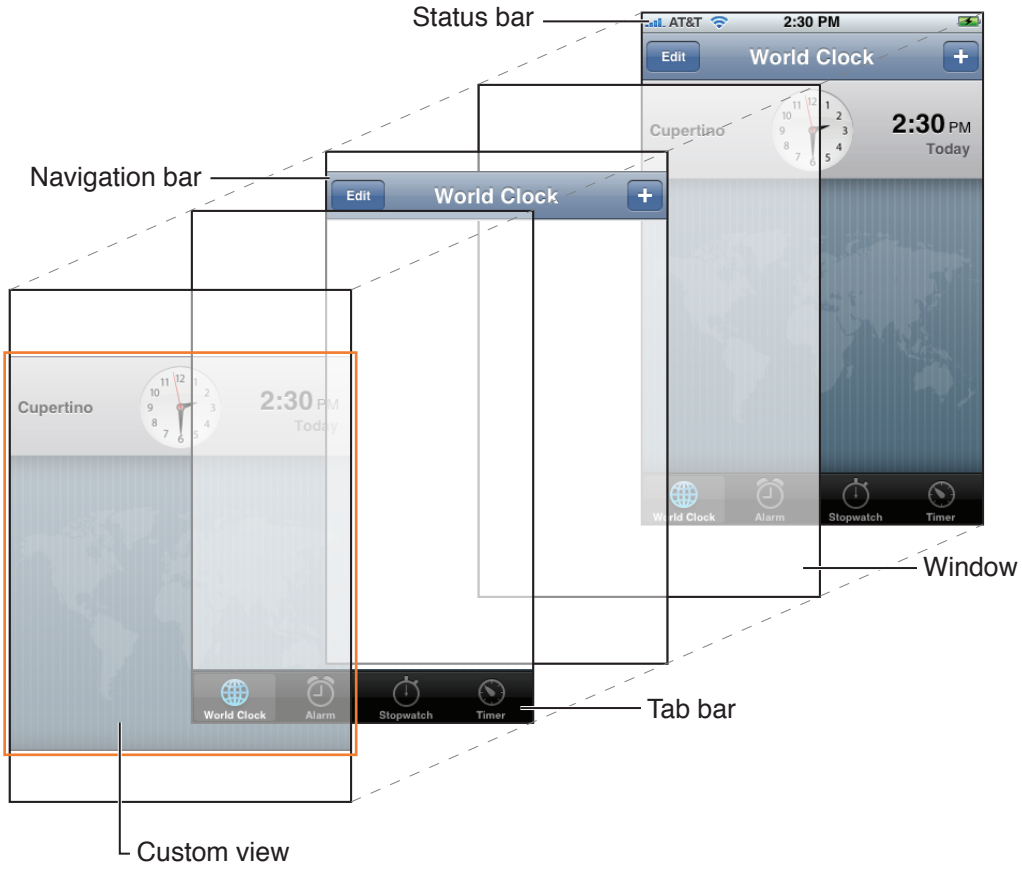
自動サイズ変更動作は、いくつかのレイアウトのニーズに適していますが、ビューのレイアウトをより細かく制御したい場合は、適切なビュークラスの`layoutSubviews`メソッドをオーバーライドする必要があります。ビューのレイアウト管理の詳細については、「レイアウト変更への対応」（78 ページ）を参照してください。

ビュー階層の作成と管理

ユーザインターフェイスのビュー階層の管理は、アプリケーションのユーザインターフェイスの開発において極めて重要な部分です。ビューをどのように構成するかによって、アプリケーションの表示だけでなく、変更に対するアプリケーションの応答方法が決まります。ビュー階層の親子関係は、アプリケーションにおいてタッチイベントの処理を担うオブジェクトのチェーンを定義するのに役立ちます。また、親子関係は、ユーザがデバイスを回転したときに、ユーザインターフェイスの向きの変化に応じて、各ビューのサイズと位置を変更する方法を定義するのにも寄与します。

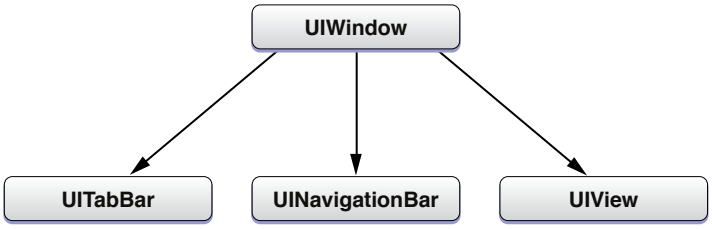
図2-9は、ビューをレイヤ化して、望みどおりの視覚効果を得る様子を示す簡単な例です。この「時計(Clock)」アプリケーションでは、タブバービューとナビゲーションバービューをカスタムビューと合成して、インターフェイス全体を実現しています。

図 2-9　「時計(Clock)」アプリケーションのレイヤ化されたビュー



「時計(Clock)」アプリケーションにおけるビューのオブジェクトの関係を見ると、「ビューのレイヤの変更」で示した関係に似ています。ウインドウオブジェクトが、このアプリケーションのタブバー、ナビゲーションバー、およびカスタムビューのルートビューとしての役割を果たします。

図 2-10　「時計(Clock)」アプリケーションのビュー階層



iPhoneアプリケーションでビュー階層を作成するには、Interface Builderでグラフィカルに行う方法や、コード内でプログラミングによる方法など、いくつかの方法があります。以降の各セクションでは、ビュー階層の組み立て方を示します。また、ビュー階層を作成した後で、階層内のビューを検索する方法、異なるビュー座標系の間の変換を行う方法についても説明します。

ビューオブジェクトの作成

ビューを作成する最も簡単な方法は、Interface Builderを使用して、その結果生成されたnibファイルからビューをロードすることです。Interface Builderのグラフィック環境で、ライブラリから新規ビューをドラッグして、ウインドウまたはビューにドロップすることで、ビュー階層をすばやく作成できます。Interface Builderは、実際のビューオブジェクトを使用するので、インターフェイスをグラフィカルに作成すると、実行時にビューがロードされたときの様子がそのまま表示されます。また、各ビューをビュー階層に配置して初期化するための煩雑なコードを記述する必要もありません。

Interface Builderとnibファイルを使用してビューを作成せずに、プログラミングによってビューを作成することもできます。ビューオブジェクトを新しく作成するには、そのビューオブジェクトのためのメモリを割り当てて、そのオブジェクトに`initWithFrame:`メッセージを送信して初期化します。たとえば、`UIView`クラスのインスタンスを新規に作成し、ほかのビューのコンテナとして使用するには、次のようなコードを使用します。

```
CGRect  viewRect = CGRectMake(0, 0, 100, 100);
UIView* myView = [[UIView alloc] initWithFrame:viewRect];
```

注： すべてのシステムオブジェクトは、`initWithFrame:`メッセージをサポートしますが、代わりに優先して使用すべき初期化メソッドを持たせることもできます。独自の初期化メソッドの詳細については、そのクラスのリファレンス文書を参照してください。

ビューを初期化するときに指定するフレーム矩形の位置とサイズは、親ビュー候補の座標系で表します。ビューを画面に表示するには、そのビューをウインドウまたは別のビューに追加する必要があります。それによって、UIKitは、指定されたフレーム矩形を使用して、親ビュー内にそのビューを配置します。ビューをビュー階層に追加する方法については、「サブビューの追加と削除」（72 ページ）を参照してください。

サブビューの追加と削除

Interface Builderは、ビュー階層を作成するための最も便利な方法です。これを利用すると、実行時にビューが表示される様子をそのまま見ることができます。Interface Builderは、ビューオブジェクトと、それらのビューの階層関係をnibファイルに保存します。システムは、実行時にこのファイル

を使用して、ビューオブジェクトとその関係をアプリケーション内に再現します。nibファイルがロードされると、システムは、ビュー階層を再現するために必要なUIViewメソッドを自動的に呼び出します。

Interface Builderとnibファイルを使用してビュー階層を作成したくない場合は、プログラミングによって作成することもできます。ビューが必須のサブビューを持つ場合は、そのビューと一緒に確実にサブビューを作成して初期化するために、そのビューのinitWithFrame:メソッドでそれらのサブビューを作成します。アプリケーション設計の一部ではあっても、ビュー操作のために必須ではないサブビューは、ビューの初期化コード以外の場所で作成します。iPhoneアプリケーションで、プログラミングによってビューやサブビューを最もよく作成する場所は、アプリケーションデリゲートのapplicationDidFinishLaunching:メソッドと、View ControllerのloadViewメソッドの2つです。

ビュー階層内のビューを操作するには、以下のメソッドを使用します。

- 親ビューにサブビューを追加するには、親ビューのaddSubview:メソッドを呼び出します。このメソッドは、親のサブビューリストの末尾にサブビューを追加します。
- 親のサブビューリストの中間にサブビューを挿入するには、親ビューのinsertSubview:...メソッドのいずれかを呼び出します。
- 親ビュー内の既存のサブビューの順番を変更するには、親ビューのbringSubviewToFront:メソッド、sendSubviewToBack:メソッド、またはexchangeSubviewAtIndex:withSubviewAtIndex:メソッドを呼び出します。サブビューを削除して挿入し直すよりも、これらのメソッドを使用した方が高速です。
- 親ビューからサブビューを削除するには、（親ビューではなく）サブビューのremoveFromSuperviewメソッドを呼び出します。

サブビューを追加すると、そのサブビューの現在のフレーム矩形が、親ビュー内でのそのビューの初期位置として使われます。この位置は、サブビューのframeプロパティを変更することで、いつでも変更できます。親ビューの表示領域の外側にはみ出しているフレームを持つサブビューは、デフォルトではクリッピングされません。クリッピングを有効にするには、親ビューのclipsToBoundsプロパティをYESに設定する必要があります。

リスト 2-1は、アプリケーションデリゲートオブジェクトのapplicationDidFinishLaunching:メソッドの例を示します。この例では、アプリケーションデリゲートが、起動時にユーザインターフェイス全体をプログラムで作成します。このインターフェイスは、原色を表示する2つの汎用UIViewオブジェクトで構成されます。次に、各ビューをウインドウに組み込んでいます。このウインドウもUIViewのサブクラスなので、親ビューになることができます。親はサブビューを保持するので、このメソッドでは新しく作成したビューが重複して保持されることのないようにそれらを解放します。

リスト 2-1　ビューを含むウインドウの作成

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // ウインドウオブジェクトを作成して
    // それをアプリケーションデリゲートのwindowインスタンス変数に代入する
    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    window.backgroundColor = [UIColor whiteColor];

    // 単純な赤い四角形を作成する
    CGRect redFrame = CGRectMake(10, 10, 100, 100);
    UIView *redView = [[UIView alloc] initWithFrame:redFrame];
    redView.backgroundColor = [UIColor redColor];
```

```
    // 単純な青い四角形を作成する
    CGRect blueFrame = CGRectMake(10, 150, 100, 100);
    UIView *blueView = [[UIView alloc] initWithFrame:blueFrame];
    blueView.backgroundColor = [UIColor blueColor];

    // これらの四角形ビューをウインドウに追加する
    [window addSubview:redView];
    [window addSubview:blueView];

    // ビューをウインドウに追加したら、各ビューの保持カウントが
    // 増えないように、そのビューを解放する
    [redView release];
    [blueView release];

    // ウインドウを表示する
    [window makeKeyAndVisible];
}
```

重要： メモリ管理を考慮するときには、サブビューを、ほかの任意のコレクションオブジェクトと同等なものとして考えます。特に、`addSubview:`を使用してサブビューとしてビューを挿入すると、そのサブビューはスーパービューによって保持されます。逆に、`removeFromSuperview`メソッドを使用してスーパービューからサブビューを削除すると、そのサブビューは自動的に解放されます。ビュー階層にビューを追加した後でビューを解放することによって、ビューが重複して保持されるのを防止できます。重複保持はメモリリークの原因になります。

Cocoaのメモリ管理規則の詳細については、『*Memory Management Programming Guide for Cocoa*』を参照してください。

親ビューにサブビューが追加されると、UIKitは、何が起こっているかを知らせるために親ビューと子ビューの両方にいくつかのメッセージを送信します。カスタムビューで、`willMoveToSuperview:`、`willMoveToWindow:`、`willRemoveSubview:`、`didAddSubview:`、`didMoveToSuperview`、`didMoveToWindow`などのメソッドをオーバーライドして、変更が生じる前や後にそれらに変更を処理したり、その変更に応じてビューの状態情報を更新したりできます。

ビュー階層を作成したら、ビューの`superview`プロパティを使用してその親を取得したり、`subviews`プロパティを使用してその子を取得したりできます。`isDescendantOfView:`メソッドを使用して、特定のビューが親ビューのビュー階層内に含まれているかどうかを判別することもできます。ビュー階層のルートビューには親はないので、その`superview`プロパティは`nil`に設定されています。現在オンスクリーンのビューの場合、通常はウインドウオブジェクトがその階層のルートビューです。

ビューの`window`プロパティを使用して、（もし存在すれば）そのビューを現在保持しているウインドウへのポインタを取得することもできます。ビューが現在ウインドウに保持されていない場合、このプロパティは`nil`に設定されます。

ビュー階層での座標の変換

しばしば、特にイベント処理の際に、アプリケーションが座標値を参照フレームから別のフレームに変換しなければならない場合があります。たとえば、通常タッチイベントではタッチ位置をウインドウの座標系を使用して報告しますが、ビューオブジェクトはビューのローカル座標（ウインドウの座標系とは異なる場合がある）で、その情報を必要とします。`UIView`クラスには、ビューのローカル座標系との間で相互に座標変換を行うために、以下のメソッドが定義されています。

```
convertPoint:fromView:
```

```
convertRect:fromView:
convertPoint:toView:
convertRect:toView:
```

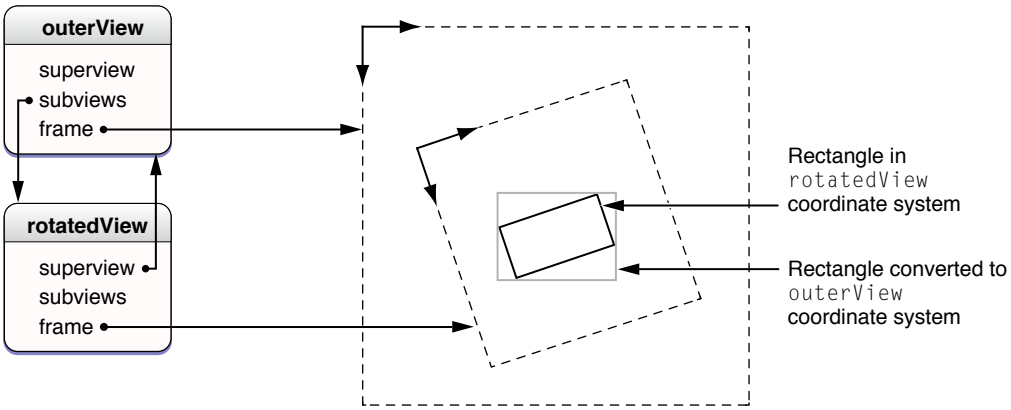
convert...:fromView:メソッド群は、座標をそのビューのローカル座標系に変換します。一方、convert...:toView:メソッド群は、座標をそのビューのローカル座標系から指定されたビューの座標系に変換します。これらのメソッドの参照ビューとしてnilを指定すると、そのビューを含んでいるウインドウの座標系との間での変換が行われます。

UIViewの変換メソッドのほかに、UIWindowにも、いくつかの変換メソッドが定義されています。これらのメソッドは、ビューのローカル座標系との間での変換の代わりに、ウインドウの座標系との間での変換を行うということ以外は、UIViewの変換メソッドと同様です。

```
convertPoint:fromWindow:
convertRect:fromWindow:
convertPoint:toWindow:
convertRect:toWindow:
```

どちらのビューも回転していない場合や点のみを扱う場合は、座標の変換は簡単です。回転が異なるビューの間で矩形やサイズを変換する場合は、変換後の座標が正しくなるようにジオメトリ構造を適切な方法で変更しなければなりません。矩形を変換する場合、UIViewクラスは元の画面領域をカバーすることを前提にします。このため、変換後の矩形は、適切なビューに配置されたときに元の矩形を完全に覆うよう拡大されます。図 2-11は、rotatedViewオブジェクトの座標系の矩形を、スーパービューであるouterViewの座標系の矩形に変換する様子を示します。

図 2-11 回転したビューの値の変換



サイズ情報を変換するとき、UIViewはサイズ情報を単純に、あるビューから別のビューに変換する必要のある、(0.0, 0.0)からのデルタオフセットとして扱います。オフセットの距離は同じに保たれますが、2つの軸とのバランスは回転に応じて変化します。サイズを変換するとき、UIKitは常にサイズを正の数で返します。

ビューのタグ付け

UIViewクラスには、個々のビューオブジェクトに整数値のタグを付けるために使用できるtagプロパティがあります。タグを使用して、ビュー階層内のビューを一意に識別したり、実行時のこれらのビューを検索したりできます（タグを使用した検索は、ビュー階層そのものを走査するより高速です）。tagプロパティのデフォルト値は0です。

タグ付きのビューを検索するには、UIViewのviewWithTag:メソッドを使用します。このメソッドは、対象ビューを起点として、深さ優先探索を使用してそのビューのサブビューを検索します。

実行時のビューの変更

アプリケーションはユーザからの入力を受け取ると、それに対応してユーザインターフェイスを変更します。アプリケーションは、インターフェイス内のビューを再配置したり、変更されたデータを含む既存のビューを更新したり、まったく新しいビューをロードしたりします。どの手法を使用するかを決定する場合は、インターフェイスや実行しようとしている内容を考慮します。ただし、これらの手法を開始する方法はどのアプリケーションでも同じです。以降の各セクションでは、これらの手法と、これらを使用して実行時にユーザインターフェイスを更新する方法について説明します。

注： UIKitとカスタムコードの間で、どのようにイベントとメッセージがやり取りされるかについては、前述の「ビューの対話モデル」（58 ページ）を参照してください。

ビューのアニメーション化

アニメーションは、ユーザインターフェイスの異なる状態の間を流れるようにビジュアルに遷移させます。iPhone OSでは、ビューの位置やサイズの変更、ビューをフェードインまたはフェードアウトするためのアルファ値の変更などに、アニメーションが幅広く使われています。アニメーションのサポートは、アプリケーションを使いやすくするために極めて重要です。このためUIKitでは、アニメーションの作成処理を簡単にするために、そのサポートが直接UIViewクラスに組み込まれています。

UIViewクラスには、もとから**アニメーション化可能な**プロパティがいくつかあります。つまり、これらのプロパティが現在の値から新しい値に変化する様子をアニメーション化するためのサポート機能がビューに組み込まれています。このアニメーションの実行に必要な作業は、UIViewクラスによって自動的に処理されますが、それにはアニメーションを実際に実行したいということをビューに伝える必要があります。それには、対象プロパティに対する変更をアニメーションブロックでラップします。

アニメーションブロックは、UIViewのbeginAnimations:context:クラスメソッドの呼び出しで始まり、commitAnimationsクラスメソッドの呼び出しで終わります。これらの呼び出しの間で、アニメーションパラメータを設定したり、アニメーション化するプロパティを変更したりします。commitAnimationsメソッドが呼び出されるとすぐに、UIKitは、アニメーションを実行し、現在の値から、設定したばかりの新しい値への変更をアニメーション化します。アニメーションブロックをネストさせることもできます。ただしネストされたアニメーションは、一番外側のアニメーションブロックがコミットされるまで開始されません。

表 2-2に、UIViewクラスのアニメーション化可能なプロパティを示します。

表 2-2 アニメーション化可能プロパティ

プロパティ	説明
`frame`	スーパービューの座標系で表したビューのフレーム矩形。
`bounds`	ビューの座標系で表したビューの境界矩形。
`center`	スーパービューの座標系で表したフレームの中心。
`transform`	境界の中心を基準にしてビューに適用される変換。
`alpha`	ビューの透明度を決定するビューのアルファ値。

アニメーションパラメータの設定

アニメーションブロック内で、プロパティ値を変更することのほかに、アニメーションの進め方を決定する追加パラメータを設定することもできます。それには、`UIView`クラスの以下のメソッドを呼び出します。

- `commitAnimations`メソッドから戻った後にアニメーションの開始日を設定するには、`setAnimationStartDate:`メソッドを使用します。デフォルトの動作では、アニメーションスレッドですぐにアニメーションが実行されるようにスケジュールされます。
- `commitAnimations`メソッドから戻ってから実際にアニメーションを開始するまでの遅延時間を設定するには、`setAnimationDelay:`メソッドを使用します。
- アニメーションを実行する秒数を設定するには、`setAnimationDuration:`メソッドを使用します。
- アニメーション実行中のアニメーションの相対的な速度を設定するには、`setAnimationCurve:`メソッドを使用します。たとえば、最初は徐々に速度を上げていき、終わりに近づくにつれて徐々に速度を落としたり、最初から最後まで同じ速度を保ったりすることができます。
- アニメーションの繰り返し回数を設定するには、`setAnimationRepeatCount:`メソッドを使用します。
- アニメーションが目標値に達したら自動的に巻き戻すかどうかを設定するには、`setAnimationRepeatAutoreverses:`メソッドを使用します。このメソッドを`setAnimationRepeatCount:`メソッドと組み合せて使用すると、一定の時間の間に各プロパティを初期値と最終値の間で滑らかに切り替えることができます。

`commitAnimations`クラスメソッドは、アニメーションの開始前に即座に戻ります。UIKitは、アプリケーションのメインイベントループとは別のスレッドでアニメーションを実行します。`commitAnimations`メソッドは、この独立したスレッドにアニメーションをポストします。アニメーションは、このスレッドで実行の準備が整うまで待機させられます。デフォルトでは、Core Animationは現在実行中のアニメーションブロックが終了してから待機中のアニメーションを開始します。ただし、アニメーションブロック内で`setAnimationBeginsFromCurrentState:`クラスメソッドに`YES`を渡すことによって、この動作をオーバーライドして、ただちにアニメーションを開始させることもできます。これにより、現在実行中のアニメーションは停止し、新規のアニメーションが現在の状態から開始されます。

デフォルトでは、アニメーションブロック内の、アニメーション化可能なプロパティのすべての変更がアニメーション化されます。アニメーションブロック内の一部の変更をアニメーション化したくない場合は、`setAnimationsEnabled:`メソッドを使用して、一時的にアニメーションを無効にし

てから、変更を行い、その後でアニメーションを再度有効にします。setAnimationsEnabled:が値NOを指定して呼び出された後に行われた変更は、同じメソッドが値YESを指定して呼び出されるか、アニメーションブロックがコミットされるまで、アニメーション化されません。アニメーションが現在有効になっているかどうかを判別するには、areAnimationsEnabledメソッドを使用します。

アニメーションデリゲートの設定

アニメーションブロックにデリゲートを割り当て、そのデリゲートを使用して、アニメーションの開始時と終了時にメッセージを受信できます。これによって、アニメーションの開始直前または終了直後に追加タスクを実行することができます。UIViewのsetAnimationDelegate:クラスメソッドを使用してデリゲートを設定し、setAnimationWillStartSelector:メソッドとsetAnimationDidStopSelector:メソッドを使用して、メッセージを受信するセレクタを指定します。これらのメソッドのシグニチャは、以下のとおりです。

```
- (void)animationWillStart:(NSString *)animationID context:(void *)context;
- (void)animationDidStop:(NSString *)animationID finished:(NSNumber *)finished
 context:(void *)context;
```

両方のメソッドに渡される*animationID*パラメータと*context*パラメータは、アニメーションブロックの先頭でbeginAnimations:context:メソッドに渡されたパラメータと同じです。

- *animationID*—アプリケーションが提供する文字列で、アニメーションブロック内のアニメーションを識別するために使用されます。
- *context*—アプリケーションが提供するもう1つのオブジェクトで、追加情報をデリゲートに渡すために使用できます。

setAnimationDidStopSelector:セレクタメソッドには、もう1つ引数（ブール値）があります。この値は、アニメーションが最後まで実行され、別のアニメーションによって途中でキャンセルまたは停止されなかった場合にYESになります。

レイアウト変更への対応

ビューのレイアウトが変更されるたびに、UIKitは、各ビューに自動サイズ変更動作を適用します。そして、layoutSubviewsメソッドを呼び出して、そのビューに含まれるサブビューの配置を調整する機会を提供します。レイアウトの変更は、以下のいずれかの場合に発生します。

- ビューの境界矩形のサイズが変化した場合。
- スクロールビューのコンテンツのオフセット値、つまり、表示されているコンテンツ領域の原点が変化した場合。
- ビューに関連付けられている変換が変化した場合。
- ビューのレイヤに関連付けられているCore Animationのサブレイヤセットが変化した場合。
- ビューのsetNeedsLayoutメソッドまたはlayoutIfNeededメソッドの呼び出しによって、アプリケーションが強制的にレイアウトを実行した場合。
- ビューの基盤レイヤオブジェクトのsetNeedsLayoutメソッドの呼び出しによって、アプリケーションが強制的にレイアウトを実行した場合。

ビューの自動サイズ変更動作は、サブビューを配置するという最初の処理を実行します。これらの動作を適用することによって、ビューは、意図したサイズに近くなります。自動サイズ変更動作が、ビューのサイズと位置にどのような影響を与えるかについては、「自動サイズ変更動作」（69 ページ）を参照してください。

自動サイズ変更動作にすべてを任せずに、`layoutSubviews`を使用して手動でサブビューのレイアウトを調整したい場合もあります。たとえば、複数のサブビュー要素で構成されるカスタムコントロールを実装している場合は、サブビューを手動で調整した方が、コントロールの外観を一定の範囲で正確に設定できます。あるいは、スクロール可能な大きなコンテンツ領域を表すビューを、一連のサブビューを並べることによって表示することもできます。スクロール中は、画面の一方の端から消えたビューが再利用されて、新しいコンテンツと共にもう一方の端から表示されます。

注： `layoutSubviews`メソッドを使用して、ビューのレイヤにサブレイヤとして追加されたカスタム`CALayer`オブジェクトの位置とサイズを調整することもできます。ビューの背後にあるカスタムレイヤ階層を管理することによって、Core Animationを直接使用した高度なアニメーションを実行できます。Core Animationを使用してレイヤ階層を管理する方法の詳細については、『*Core Animation Programming Guide*』を参照してください。

レイアウトコードを記述する際には、アプリケーションがサポートするインターフェイスの向きごとに、コードをテストしてください。横長と縦長の両方の向きをサポートするアプリケーションでは、それぞれの向きでレイアウトが正しく処理されることを確認する必要があります。同様に、アプリケーションは、ステータスバーの高さの変更など、その他のシステム変更にも対応できるようにしておく必要があります。このような状況は、ユーザが通話中にアプリケーションを使用し、その後その通話を切ると発生します。電話を切った時点で、管理を行うView Controllerはステータスバーの縮小に合わせてビューのサイズを変更します。このような変更は、アプリケーション内の残りのビューにも上から順に適用されます。

ビューのコンテンツの再描画

時々、アプリケーションのデータモデルの変更によって、それに対応するユーザインターフェイスの変更が必要になる場合があります。このような変更を行うには、対応するビューをダーティとしてマークし、更新が必要であることを示します（`setNeedsDisplay`メソッドまた`setNeedsDisplayInRect:`メソッドのいずれかを使用する）。ビューをダーティとしてマークすると、単純にグラフィックスコンテキストを作成して描画するのに比べて、システムがより効率的に描画操作を処理できる場合があります。たとえば、特定の周期において同じビューの複数の領域をダーティとしてマークすると、システムはこれらのダーティ領域をひとまとめにして、そのビューの`drawRect:`メソッドを1回呼び出します。その結果、グラフィックスコンテキストを1つ作成すれば、影響を受けるすべての領域を描画できます。このやり方は、立て続けに複数のグラフィックスコンテキストを作成するよりもはるかに効率的です。

`drawRect:`メソッドを実装するビューは、常に、そのメソッドに渡された矩形をチェックし、それを使用して描画操作の範囲を制限すべきです。描画は比較的負荷のかかる操作なので、このように描画を制限することはパフォーマンスを向上させるためによい方法です。

デフォルトでは、ビューのジオメトリを変更してもビューは自動的には再描画されません。その代わりに、ほとんどのジオメトリ変更はCore Animationによって自動的に処理されます。特に、ビューの`frame`、`bounds`、`center`、または`transform`の各プロパティを変更した場合、Core Animationは、そのビューのレイヤに関連付けられているキャッシュ済みのビットマップに対してジオメトリ変更を適用します。多くの場合、このアプローチで十分です。しかし、その結果では不十分な場合は、UIKitを使用して強制的にビューを再描画することもできます。Core Animationが暗黙のうちにジオメ

トリ変更を適用するのを防止するには、ビューのcontentModeプロパティをUIViewContentModeRedrawに設定します。コンテンツモードの詳細については、「コンテンツモードと拡大縮小」（67 ページ）を参照してください。

ビューの非表示

ビューのhiddenプロパティの値を変更することによって、ビューを表示したり非表示にしたりできます。このプロパティをYESに設定するとビューが非表示になり、NOに設定すると表示されます。ビューを非表示にすると、その内部に含まれるサブビューも、それら自身のhiddenプロパティを設定したかのように、非表示になります。

ビューを非表示にしても、そのビューはビュー階層内に残っています。ただし、そのコンテンツは表示されず、タッチイベントも受信しません。ビュー階層内に残っているため、非表示のビューも、自動サイズ変更やその他のレイアウト操作の対象になります。現在ファーストレスポンダになっているビューを非表示にした場合、そのビューが自動的にファーストレスポンダでなくなることはありません。ファーストレスポンダ宛てのイベントは、非表示のビューに送られ続けます。レスポンダチェーンの詳細については、「レスポンダオブジェクトとレスポンダチェーン」（88 ページ）を参照してください。

カスタムビューの作成

UIViewクラスは、画面にコンテンツを表示したり、タッチイベントを処理したりするための基本的な仕組みを提供します。ただし、そのインスタンスは、背景色（アルファ値を使用）とサブビュー以外は何も描画しません。アプリケーションが、特定の方法でコンテンツを表示したり、タッチイベントを処理したりする必要がある場合は、UIViewのカスタムサブクラスを作成しなければなりません。

以降の各セクションでは、カスタムビューオブジェクトで実装する候補となる主なメソッドとその動作について説明します。サブクラス化の詳細については、『*UIWindow Class Reference*』を参照してください。

カスタムビューの初期化

新規に定義するビューオブジェクトには、それぞれカスタムのinitWithFrame:初期化メソッドを含める必要があります。このメソッドは、作成時にクラスを初期化して、ビューオブジェクトを既知の状態に設定する処理を担当します。コード内でプログラムによってビューのインスタンスを作成する場合に、このメソッドを使用します。

リスト 2-2に、標準的なinitWithFrame:メソッドの実装の骨格を示します。このメソッドでは、継承した実装を最初に呼び出します。次に、このクラスのインスタンス変数と状態情報を初期化した後、初期化済みのオブジェクトを返します。継承した実装の呼び出しは、慣例として最初に実行されます。そこで問題が発生した場合に、初期化コードを中断して、nilを返せるようにするためです。

リスト 2-2　ビューのサブクラスの初期化

```
- (id)initWithFrame:(CGRect)aRect {
    self = [super initWithFrame:aRect];
```

```
    if (self) {
          // ビューの初期プロパティを設定する
          ...
       }
    return self;
}
```

nibファイルからカスタムビュークラスのインスタンスをロードする予定の場合は、iPhone OSでは、nibをロードするコードが、新規のビューオブジェクトをインスタンス化するためにinitWithFrame:メソッドを使用しないことを認識しておく必要があります。その代わりに、NSCodingプロトコルの一部として定義されているinitWithCoder:メソッドを使用します。

たとえビューがNSCodingプロトコルを採用していたとしても、Interface Builderはそのビューのカスタムプロパティを知らないので、それらのプロパティをnibファイルにエンコードしません。その結果、nibファイルからクラスがロードされたときに、カスタムのinitWithCoder:メソッドは、そのクラスを正しく初期化するのに必要な情報を得ることができません。この問題を解決するには、クラスにawakeFromNibメソッドを実装して、nibファイルからロードされたときに、このメソッドを使用してクラスを明示的に初期化するようにします。

ビューのコンテンツの描画

ビューのコンテンツを変更した場合は、setNeedsDisplayメソッドまたはsetNeedsDisplayInRect:メソッドを使用して、そのビューの一部を再描画する必要があることをシステムに通知します。アプリケーションが実行ループに戻ると、アプリケーションは、すべての描画要求をまとめて更新が必要なインターフェイス部分を計算します。次に、ビュー階層を走査して、更新が必要なビューにdrawRect:メッセージを送ります。この走査は、ビュー階層のルートビューから始まって下位のサブビューへと進み、背面から前面へ向って処理します。可視領域内にカスタムコンテンツを表示するビューには、それらのコンテンツをレンダリングするためのdrawRect:メソッドを実装する必要があります。

ビューのdrawRect:メソッドを呼び出す前に、UIKitは、ビューの描画環境を設定します。UIKitは、グラフィックスコンテキストを作成して、ビューの座標系と境界に合わせて、座標系とクリッピング領域を調整します。したがって、drawRect:メソッドが呼び出されたときには、UIKitのクラスと関数、Quartz関数、またはそれらの組み合わせを使用して、描画を開始するだけで済みます。現在のグラフィックスコンテキストにアクセスする必要がある場合は、UIGraphicsGetCurrentContext関数を使用して、そのポインタを取得できます。

重要： 現在のグラフィックスコンテキストは、ビューのdrawRect:メソッドの1回の呼び出しの間だけ有効です。UIKitは、このメソッドが呼び出されるたびに別のグラフィックスコンテキストを作成します。したがって、後から使用するために、このオブジェクトをキャッシュするべきではありません。

リスト2-3に、10ピクセルの幅の赤い境界をビューの周囲に描く処理を行うdrawRect:メソッドの簡単な実装を示します。UIKitの描画操作は、内部の実装にQuartzを使用しているので、ここで示すように描画呼び出しを組み合わせた場合でも、期待どおりの結果を得ることができます。

リスト 2-3　描画メソッド

```
- (void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGRect    myFrame = self.bounds;
```

```
    CGContextSetLineWidth(context, 10);

    [[UIColor redColor] set];
    UIRectFrame(myFrame);
}
```

ビューの描画コードによって、不透明なコンテンツでビューのサーフェス全体が常に覆われることがわかっている場合は、ビューのopaqueプロパティをYESに設定することによって、描画コード全体の効率を向上させることができます。ビューを不透明としてマークすると、UIKitはそのビューのすぐ背後にあるコンテンツの描画を行いません。これによって、描画にかかる時間が削減されるだけでなく、そのコンテンツを合成するために必要な作業が最小になります。ビューが不透明なコンテンツを提供する場合にのみ、このプロパティをYESに設定してください。ビューのコンテンツが常に不透明であることを保証できない場合は、このプロパティをNOに設定してください。

特にスクロール中の描画パフォーマンスを向上させるもう1つの方法は、ビューのclearsContextBeforeDrawingプロパティをNOに設定することです。このプロパティをYESに設定すると、UIKitは、drawRect:メソッドが呼び出される前に、このメソッドによって更新される予定の領域を透明な黒で自動的に塗りつぶします。このプロパティをNOに設定すると、この塗りつぶし操作のためのオーバーヘッドがなくなります。ただし、drawRect:メソッドに渡された更新領域内のビューの部分を完全に再描画するのはアプリケーションの責任になります。このような最適化は、一般に、スクロールの際に検討に値します。

イベントへの応答

UIViewクラスは、UIResponderのサブクラスです。したがって、ビューのコンテンツに対するユーザ操作に対応するタッチイベントを受け取れます。タッチイベントは、タッチが発生したビューで始まり、それが処理されるまで、レスポンダチェーンに沿って受け渡しされます。ビュー自体がレスポンダなので、ビューはレスポンダチェーンに含まれています。したがって、関連するサブビューから送付されたタッチイベントを受け取ることができます。

一般に、タッチイベントを処理するビューは、以下のメソッドをすべて実装しています。これらの詳細については、「イベント処理」（85 ページ）を参照してください。

```
touchesBegan:withEvent:
touchesMoved:withEvent:
touchesEnded:withEvent:
touchesCancelled:withEvent:
```

デフォルトでは、ビューは一1度に1つのタッチイベントにしか応答しないことを忘れないでください。ユーザが2番目の指を画面に置いても、システムはそのタッチイベントを無視し、ビューに報告しません。ビューのイベント処理メソッドから複数の指によるジェスチャを追跡したい場合は、ビューのmultipleTouchEnabledプロパティをYESに設定して、マルチタッチイベントを有効にする必要があります。

ラベルや画像のように、最初からイベント処理が完全に無効になっているビューもあります。ビューがイベントを処理するかどうかは、そのビューのuserInteractionEnabledプロパティを変更することによって制御できます。時間のかかる処理を実行している間、ユーザがビューのコンテンツを操作できないようするには、このプロパティを一時的にNOに設定します。イベントがどのビューにも届かないようにするには、UIApplicationオブジェクトのbeginIgnoringInteractionEventsメソッドとendIgnoringInteractionEventsメソッドを使用することもできます。これらのメソッドは、1つのビューだけでなくアプリケーション全体のイベントの配信に影響を与えます。

タッチイベントを処理するとき、UIKitは、UIViewのhitTest:withEvent:メソッドとpointInside:withEvent:メソッドを使用して、タッチイベントがそのビューで発生したかどうかを判断します。これらのメソッドをオーバーライドする必要が生じることはほとんどありませんが、ビューに独自のタッチ動作を実装するために、オーバーライドすることもできます。たとえば、これらのメソッドをオーバーライドして、サブビューがタッチイベントを処理しないようにすることができます。

ビュー使用後のクリーンアップ

ビュークラスが、メモリを割り当てたり、カスタムオブジェクトへの参照を格納したり、ビューが解放されたときに解放しなければならないリソースを保持したりしている場合は、deallocメソッドを実装する必要があります。ビューの保持カウントが0になり、そのビュー自体が解放されるときに、システムはdeallocメソッドを呼び出します。このメソッドの実装では、リスト2-4に示すように、オブジェクトとそのオブジェクトが保持しているリソースを解放した後に、継承した実装を呼び出します。

リスト 2-4　deallocメソッドの実装

```
- (void)dealloc {
    // 保持しているUIColorオブジェクトを解放する
    [color release];

    // 継承した実装を呼び出す
    [super dealloc];
}
```

イベント処理

この章では、iPhone OSにおけるイベントの種類と処理方法について説明します。また、iPhone OS 3.0で導入されたUIPasteboardクラスの機能を使用して、アプリケーション内またはアプリケーション間でデータのコピーおよびペーストを行う方法についても説明します。

iPhone OSは、タッチイベントとモーションイベントの2種類のイベントをサポートしています。UIEventクラスは、タッチイベントとモーションイベントだけでなく将来の新しい種類のイベントにも対応するために、iPhone OS 3.0で拡張されました。各イベントには、イベントタイプとそれに対応するサブタイプ定数があります。これらには、UIEventのtypeプロパティとsubtypeプロパティを介してそれぞれアクセスできます。イベントタイプには、タッチイベントとモーションイベントの両方が含まれます。iPhone OS 3.0では、シェイクモーションサブタイプ(UIEventSubtypeMotionShake)のみがあります。

タッチイベント

iPhone OSのタッチイベントは、Multi-Touchモデルに基づいています。ユーザは、マウスとキーボードを使用する代わりに、デバイスの画面をタッチしてオブジェクトを操作したりデータを入力したりその他の意図を伝えたりします。iPhone OSは、1本以上の指で画面にタッチする操作を**Multi-Touchシーケンス**の一部として認識します。このシーケンスは、最初の指が画面に触れたときに始まって、最後に指が画面から離れたときに終わります。iPhone OSは、マルチタッチシーケンスの間中、画面に触れている指を追跡して、それぞれのタッチの特徴を記録します。それには、画面上での指の位置と、タッチが発生した時間が含まれます。通常、アプリケーションは、複数のタッチの特定の組み合わせをジェスチャとして認識して、ユーザに直観的にわかる方法でそれに応答します。たとえば、ピンチジェスチャに応答してコンテンツを拡大したり、フリック（はじく）ジェスチャに応答してコンテンツをスクロールしたりします。

注： 画面上の指は、マウスポインタとはかなり異なるレベルの精度を提供します。ユーザが画面に触れたとき、その接触領域は実際には楕円形で、ユーザ自身が触れたと考えている点の下方にずれている傾向があります。この「接触面」はまた、どの指が画面に触れたか、指のサイズ、画面上の指の圧力、指の向き、その他の要因などによってもサイズや形状が変わります。基盤のMulti-Touchシステムは、このすべての情報を分析し、単一のタッチポイントを計算します。

UIKitのクラスの多くは、そのクラスのオブジェクト特有の方法でマルチタッチイベントを処理します。このことは、特に、UIControlのサブクラス（UIButton、UISliderなど）に当てはまります。これらのサブクラスのオブジェクト（コントロールオブジェクトと呼ばれる）は、タップや特定の方向へのドラッグなど、特定のタイプのジェスチャを受け付けます。正しく設定されていれば、そのジェスチャが発生したときに対象となるオブジェクトにアクションメッセージを送信します。UIkitのその他のクラスは、ジェスチャを別のコンテキストで処理します。たとえば、UIScrollViewは、Table View、Text View、そして大きなコンテンツ領域を持つその他のビューに対してスクロール動作を提供します。

アプリケーションによっては、自分で直接イベント処理を行わずに、この動作をUIKitのクラスに任せることもできます。ただし、`UIView`のカスタムサブクラスを作成し（iPhone OSの開発ではよくあるケースです）、そのビューを特定のタッチイベントに応答させる場合は、それらのイベントを処理するために必要なコードを実装しなければなりません。さらに、UIKitオブジェクトに、イベントに対して異なる対応をさせる場合は、そのフレームワーククラスのサブクラスを作成して適切なイベント処理メソッドをオーバーライドする必要があります。

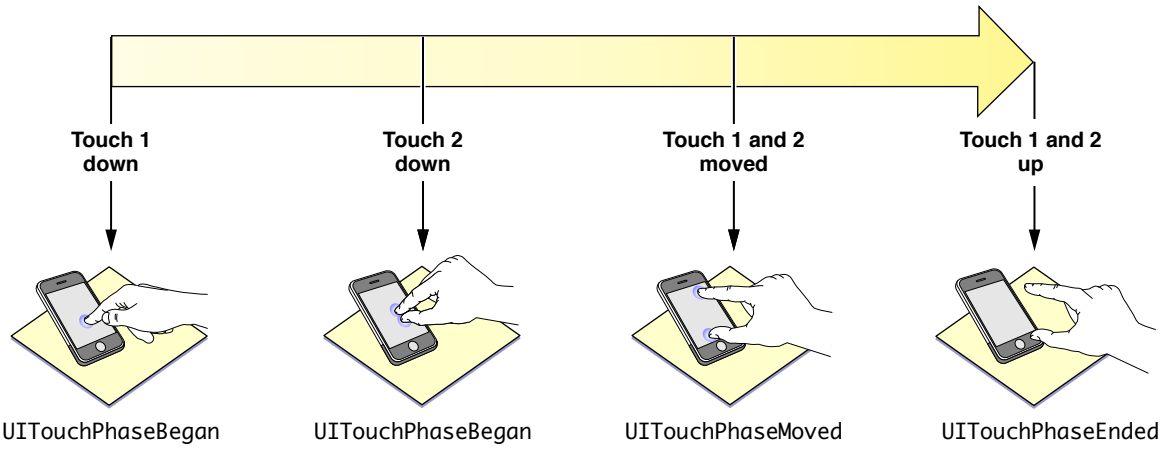
イベントとタッチ

iPhone OSでは、**タッチ**とは画面上での1本の指の存在または移動を表し、一意の**マルチタッチシーケンス**に属します。たとえば、ピンチ-クローズジェスチャには、画面上で2本の指がそれぞれ反対方向から互いに近づくという2つのタッチが含まれています。タップ、ダブルタップ、フリック（画面上を1本の指ですばやくはじく）など、1本の指の単純なジェスチャもあります。アプリケーションが、それよりも複雑なジェスチャを認識することもあります。たとえば、アプリケーションにダイヤル型のカスタムコントロールを備え、ユーザがそれを複数の指で回して何らかの変数の微調整を行えるようにすることが考えられます。

イベントは、指が画面に触れたり、画面の表面を移動したりしたときに、システムが断続的にアプリケーションに送信するオブジェクトです。イベントは、1つのマルチタッチシーケンスの間のすべてのタッチのスナップショットを提供します。最も重要なのは、特定のビューへの新規のタッチや、変化のあったタッチです。マルチタッチシーケンスは、1本の指が最初に画面に触れたときに始まります。それに続けて、その他の指が画面に触れたり、すべての指が画面上を移動したりする場合もあります。シーケンスは、最後の指が画面を離れたときに終了します。アプリケーションは、すべてのタッチの各フェーズで、イベントオブジェクトを受け取ります。

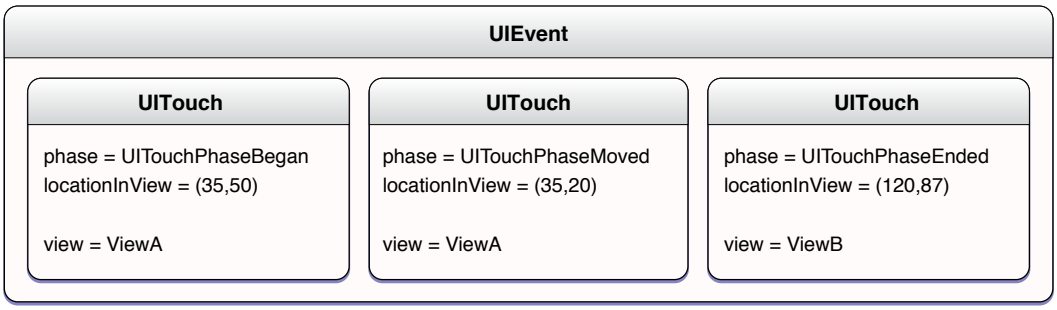
タッチには、時間的な側面と空間的な側面の両方があります。フェーズと呼ばれる時間的な側面は、タッチが始まったばかりのとき、移動しているか静止しているか、タッチが終わったとき（指が画面から離れたとき）を表します（図 3-1を参照）。タッチは、ビューまたはウインドウ内での現在位置と、（必要であれば）直前の位置も持っています。1本の指が画面に触れるとそのタッチはウインドウおよびビューに関連付けられます。その関連付けは、イベントが存続している間ずっと維持されます。複数のタッチが同時に行われた場合は、それらが同じビューに関連付けられている場合にのみまとめて扱われます。同様に、2つのタッチがすばやく立て続けに行われた場合はそれらが同じビューに関連付けられている場合にのみ、マルチタップとして扱われます。

図 3-1　マルチタッチシーケンスとタッチフェーズ



iPhone OSでは、UITouchオブジェクトがタッチを表し、UIEventオブジェクトがイベントを表します。1つのイベントオブジェクトには、現在のマルチタッチシーケンスを構成するすべてのタッチオブジェクトが含まれ、1つのビューまたはウインドウに固有のタッチオブジェクトを提供します（図 3-2を参照）。1つのタッチオブジェクトは、1つの指に対応してシーケンスの間中、存続します。また、UIKitは、その間ずっとその指を追跡しながらタッチオブジェクトを変遷させます。変遷するタッチ属性は、タッチのフェーズ、ビュー内の位置、直前の位置、およびタイムスタンプです。イベント処理コードは、これらの属性を評価してイベントへの対応方法を決定します。

図 3-2 UIEventオブジェクトとそれに対応するUITouchオブジェクトの関係



システムは、いつでもマルチタッチシーケンスをキャンセルできるため、イベント処理アプリケーションでは適切に対応できるように備えておく必要があります。キャンセルは、電話の着信など優先されるシステムイベントによって発生します。

イベントの送付

イベント処理のためのオブジェクトへのイベントの送付は、特定の経路に沿って行われます。「コアアプリケーションアーキテクチャ」（17 ページ）で説明したように、ユーザがデバイスの画面にタッチすると、iPhone OSは一連のタッチを認識して、それらを1つのUIEventオブジェクトにまとめ、それを現在のアプリケーションのイベントキューに入れます。このイベントオブジェクトは、ある時点のマルチタッチシーケンスに対応するタッチをカプセル化しています。アプリケーションを管理しているUIApplicationシングルトンオブジェクトは、イベントキューの先頭からイベントを1つ取り出して、処理のためにそれを送付します。通常、イベントは、そのアプリケーションの中心となるウインドウ（その時点でユーザイベントの対象となっているウインドウ）に送付されます。そのウインドウを表すUIWindowオブジェクトは、イベント処理のために、そのイベントをファーストレスポンダに送ります（ファーストレスポンダについては、「レスポンダオブジェクトとレスポンダチェーン」で説明しています）。

アプリケーションは、ヒットテストを使用してイベントのファーストレスポンダを探します。つまり、ビュー階層内の各ビューを対象に、hitTest:withEvent:を再帰的に（階層の下へ向かって）呼び出して、タッチが発生したサブビューを判断します。タッチは、後からビューの外に移動したとしても、そのビューが存続する限りそのビューに関連付けられています。「イベント処理のテクニック」（94 ページ）では、ヒットテストのプログラム上の注意点について説明します。

UIApplicationオブジェクトとすべてのUIWindowオブジェクトは、sendEvent:メソッドの中でイベントを送付します（両方のクラスに、同じ名前のメソッドが宣言されています）。これらのメソッドは、アプリケーションが受け取るイベントの入口であるため、UIApplicationまたはUIWindowのサブクラスを作成して、sendEvent:メソッドをオーバーライドし、イベントを監視したり特殊なイベント処理を実行したりできます。ただし、ほとんどのアプリケーションではその必要はありません。

レスポンダオブジェクトとレスポンダチェーン

レスポンダオブジェクトは、イベントに対応し、それを処理することができるオブジェクトです。すべてのレスポンダオブジェクトの基本クラスは、UIResponderです。このクラスには、イベント処理だけでなくレスポンダに共通する動作のインターフェイスが定義されています。UIApplication、UIView、およびUIViewから派生したすべてのUIKitクラス（UIWindowを含む）は、UIResponderを直接または間接的に継承しています。

ファーストレスポンダは、アプリケーション内で最初にタッチを受け取るレスポンダオブジェクト（通常は、UIViewオブジェクト）です。UIWindowオブジェクトは、メッセージにイベントを含めてファーストレスポンダに送付し、そのイベントを最初に処理する機会を与えます。このファーストレスポンダがそのイベントを処理しない場合は、レスポンダチェーン内の次のレスポンダにイベントを（メッセージに含めて）渡し、イベントを処理できるかどうかを調べます。

レスポンダチェーンとは、一連のレスポンダオブジェクトが連結されたものです。これによって、レスポンダオブジェクトは、イベント処理の役割をほかの上位レベルのオブジェクトに転送することができます。イベントは、アプリケーションがイベントを処理できるオブジェクトを見つけるまで、レスポンダチェーンをさかのぼります。レスポンダチェーンは、以下の順番で並べられた「次のレスポンダ」の連鎖で構成されます。

1. ファーストレスポンダは、自身のView Controllerにイベントを渡します（View Controllerを備えている場合）。次に、イベントをスーパービューに渡します。
2. 階層内の以降の各ビューも、同様に、まず自身のView Controllerにイベントを渡してから（View Controllerを備えている場合）、スーパービューに渡します。
3. 最上位の最後のビューは、イベントをUIWindowオブジェクトに渡します。
4. UIWindowオブジェクトは、そのイベントをUIApplicationシングルトンオブジェクトに渡します。

イベントを処理するレスポンダオブジェクトが見つからない場合、アプリケーションはそのイベントを破棄します。

レスポンダチェーン内のどのレスポンダオブジェクトもUIResponderイベント処理メソッドを実装できるため、イベントメッセージを受け取ることができます。しかし、レスポンダは、特定のイベントの処理を辞退したりイベントの一部だけを処理したりすることができます。その場合は、以下と同様のメッセージによって、次のレスポンダにイベントメッセージを転送できます。

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch* touch = [touches anyObject];
    NSUInteger numTaps = [touch tapCount];
    if (numTaps < 2) {
        [self.nextResponder touchesBegan:touches withEvent:event];
   } else {
        [self handleDoubleTap:touch];
   }
}
```

注： レスポンダが、マルチタッチシーケンスの最初のフェーズ(touchesBegan:withEvent:)のイベント処理メッセージを次のレスポンダに転送した場合は、そのシーケンスのほかのすべてのイベント処理メッセージを転送しなければなりません。

アクションメッセージもレスポンダチェーンを利用します。ユーザが、ボタンやページコントロールなどのUIControlオブジェクトを操作すると、（正しく設定されていれば）そのコントロールオブジェクトは、アクションメッセージをターゲットオブジェクトに送信します。ただし、ターゲットとしてnilが指定されている場合、アプリケーションは、イベントメッセージのときと同様に、ファーストレスポンダにこのメッセージを転送します。ファーストレスポンダがアクションメッセージを処理しない場合は、それが次のレスポンダに送信され、次々とレスポンダチェーンをさかのぼります。

イベント送付の制御

UIKitは、アプリケーションでのイベント処理を容易にしたり、イベントストリームを完全にオフにするためのプログラミング手段を提供しています。以下に、これらのアプローチの概要を示します。

- **イベントの送付を停止する**。デフォルトでビューはタッチイベントを受け取りますが、userInteractionEnabledプロパティをNOに設定してイベントの受け取りを停止することができます。ビューはまた、非表示であったり透過的であったりすると、イベントを受け取りません。
- **一定時間イベントの送付を停止する**。アプリケーションは、UIApplicationのbeginIgnoringInteractionEventsメソッドを呼び出して、その後でendIgnoringInteractionEventsメソッドを呼び出すことができます。最初のメソッドによって、アプリケーションはタッチイベントメッセージの受け取りを完全に停止することができます。2番目のメソッドを呼び出すと、マルチタッチイベントメッセージの受け取りが再開されます。アニメーションを実行しているときは、イベントの送付を停止したい場合などに便利です。
- **複数タッチの送付を有効にする**。デフォルトでは、ビューは、マルチタッチシーケンスの間、最初のタッチ以外のすべてのタッチを無視します。ビューで複数のタッチを処理する場合は、ビューに対する複数タッチを有効にする必要があります。これは、プログラムでビューのmultipleTouchEnabledプロパティにYESを設定するか、Interface Builderで関連するビューのInspectorを使って行うことができます。
- **イベントの送付先を1つのビューに限定する**。デフォルトでは、ビューのexclusiveTouchプロパティはNOに設定されています。このプロパティをYESに設定すると、このビューがタッチを追跡している間、これがウインドウの中でタッチを追跡する唯一のビューとなります。ウインドウ内のほかのビューはこれらのタッチを受け取れません。ただし、このように「排他的タッチ」のマークが付いたビューは、同じウインドウ内の別のビューに関連付けられているタッチは受け取りません。排他的タッチのビューに指が触れると、そのビューがウインドウ内で指を追跡している唯一のビューである場合にのみ、そのタッチが送付されます。非排他的ビューに指が触れると、排他的タッチのビューで別の指が追跡されていない場合にのみタッチが送付されます。
- **イベントの送付先をサブビューに限定する**。UIViewのカスタムクラスで、hitTest:withEvent:をオーバーライドして、マルチタッチイベントの送付先をサブビューに限定することができます。この手法に関する詳細な解説については、「イベント処理のテクニック」 （94 ページ）を参照してください。

マルチタッチイベントの処理

マルチタッチイベントを処理するには、UIViewカスタムサブクラス（または、頻度は低いけれどもUIApplicationやUIWindowのカスタムサブクラスの場合もある）が、イベント処理用のUIResponderメソッドを少なくとも1つ実装していなければなりません。以降の各セクションでは、これらのメソッドについて説明し、一般的なジェスチャの処理方法を示します。さらに、複雑なマルチタッチイベントを処理するレスポンダオブジェクトの例も示します。最後に、イベント処理のテクニックをいくつか提案します。

イベント処理メソッド

1つのマルチタッチシーケンスの間、アプリケーションは一連のイベントメッセージをディスパッチします。これらのメッセージを受け取って処理するには、レスポンダオブジェクトのクラスに、UIResponderで宣言されている以下のメソッドを少なくとも1つ実装している必要があります。

```
- (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
```

タッチフェーズにおいて新しいタッチまたは変化したタッチがあったときに、アプリケーションは以下のようなメッセージを送付します。

- 1本以上の指が画面に触れたときは、touchesBegan:withEvent:メッセージを送ります。
- 1本以上の指が移動したときは、touchesMoved:withEvent:メッセージを送ります。
- 1本以上の指が画面から離れたときは、touchesEnded:withEvent:メッセージを送ります。
- タッチシーケンスが電話の着信などのシステムイベントによってキャンセルされたときは、touchesCancelled:withEvent:メッセージを送ります。

これらのメソッドは、それぞれ1つのタッチフェーズ（たとえば、UITouchPhaseBegan）に関連付けられています。どのUITouchオブジェクトの場合も、phaseプロパティを評価すればフェーズがわかります。

イベント処理メソッドを起動するメッセージは、2つのパラメータを渡します。第1のパラメータは、該当するフェーズの新規タッチ、または変化のあったタッチを表すUITouchオブジェクトのセットです。第2のパラメータは、この特定のイベントを表すUIEventオブジェクトです。このイベントオブジェクトから、イベントに対応するすべてのタッチオブジェクトを取得したり(allTouches)、特定のビューまたはウインドウに対応するタッチオブジェクトだけを抽出したサブセットを取得したりできます。これらのタッチオブジェクトの中には、前回のイベントメッセージ以降変化のないタッチや、変化はあったもののフェーズが異なるタッチを表すオブジェクトも含まれます。

レスポンダオブジェクトは、1つ以上のUITouchオブジェクトをパラメータとして受け取り、それらのプロパティを評価したり、位置を取得したりして、特定のフェーズのイベントを頻繁に処理します（タッチオブジェクトを問わなければ、NSSetオブジェクトにanyObjectメッセージを送信できます）。重要なメソッドの1つに、locationInView:があります。このメソッドにselfというパラメータを渡すと、そのタッチの位置をレスポンダオブジェクトの座標系で取得できます（ただし、このレスポンダオブジェクトがUIViewであり、パラメータとして渡されたビューがnilでない場合）。同様のメソッドによって、そのタッチの以前の位置を取得できます(previousLocationInView:)。UITouchインスタンスのプロパティによって、タップの回数(tapCount)、そのタッチが作成また最後に変化した時点(timestamp)、およびそのタッチがどのフェーズ(phase)に属しているかがわかります。

レスポンダクラスが、上記の3つのイベントメソッドすべてを実装している必要はありません。たとえば、指が画面から離れたことだけを検出したい場合は、touchesEnded:withEvent:だけを実装します。

レスポンダがマルチタッチシーケンスでイベントを処理している間に永続オブジェクトを作成する場合は、システムがそのシーケンスをキャンセルしたときに、そのオブジェクトを破棄できるように、touchesCancelled:withEvent:を実装する必要があります。通常は、外部からのイベント（たとえば、電話の着信）によって現在のアプリケーションのイベント処理が中断されたときにキャンセルが発生します。レスポンダオブジェクトは、マルチタッチシーケンスの最後のtouchesEnded:withEvent:を受け取ったときにも、これらのオブジェクトを破棄しなければならない点に注意してください（シーケンス内の最後のタッチアップを判断する方法については、「イベント処理のテクニック」（94 ページ）を参照してください）。

単一のタップジェスチャと複数のタップジェスチャの処理

iPhoneアプリケーションでよく使われるジェスチャはタップです。タップは、ユーザが自分の指でオブジェクトを軽くたたくことを指します。レスポンダオブジェクトは、シングルタップ、ダブルタップ、場合によってはトリプルタップに対してそれぞれ別の方法で対応できます。ユーザがレスポンダオブジェクトをタップした回数を判断するには、UITouchオブジェクトのtapCountプロパティの値を取得します。

この値を調べるのに最適な場所は、touchesBegan:withEvent:メソッドとtouchesEnded:withEvent:メソッドです。多くの場合、ユーザがタップして指を離したときのタッチフェーズに対応しているため、後者のメソッドの方が適しています。このタッチアップフェーズ(UITouchPhaseEnded)でのタップ回数を調べることによって、たとえば、指が画面に触れた後でドラッグしているのではなく、指が本当にタップしていることを確認できます。

リスト 3-1に、ビューのいずれかにダブルタップが発生したかを検出する方法を示します。

リスト 3-1　ダブルタップジェスチャの検出

```
- (void) touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event
{
    UITouch       *touch = [touches anyObject];

    if ([touch tapCount] == 2) {
        CGPoint tapPoint = [theTouch locationInView:self];
        // ダブルタップジェスチャの処理
    }
}
```

難しいのはレスポンダオブジェクトで、シングルタップおよびダブルタップジェスチャを異なる方法で処理する場合です。たとえば、シングルタップでオブジェクトを選択し、ダブルタップで、ダブルタップされた項目を編集するためのビューを表示するようなケースです。レスポンダオブジェクトで、シングルタップと、ダブルタップの最初のタップをどうやって区別すればよいのでしょう。レスポンダオブジェクトが、先に説明したイベント処理メソッドを使用して、このような状況を処理する方法を以下に示します。

1. touchesEnded:withEvent:で、タップ回数が1の場合、レスポンダオブジェクトは自分自身にperformSelector:withObject:afterDelay:メッセージを送ります。セレクタは、このシングルタップジェスチャを処理するために、このレスポンダによって実装されている別のメソッドを識別します。2番目のパラメータは、それに関連するUITouchオブジェクトを格納するNSValueオブジェクトまたはNSDictionaryオブジェクトです。遅延は、シングルタップジェスチャとダブルタップジェスチャの間の妥当な間隔にします。

> **注：** タッチオブジェクトを格納するためにNSValueオブジェクトまたは辞書を使用する理由は、オブジェクトを渡すことによってそのオブジェクトが保持できるからです。ただし、イベントを処理するときはUITouchオブジェクトを保持すべきではありません。

2. touchesBegan:withEvent:で、タップ回数が2の場合、レスポンダオブジェクトは自分自身にcancelPreviousPerformRequestsWithTarget:メッセージを送って、保留中の遅延実行呼び出しをキャンセルします。タップ回数が2でない場合は、前のステップでシングルタップジェスチャ用としてセレクタによって識別されたメソッドが遅延の後に実行されます。

3. touchesEnded:withEvent:で、タップ回数が2の場合は、レスポンダはダブルタップジェスチャを処理するために必要なアクションを実行します。

スワイプジェスチャの検出

水平および垂直のスワイプは、コードの中で簡単に追跡してアクションに使用できる単純なジェスチャです。スワイプジェスチャを検出するには、望みの移動軸に沿ってユーザの指の動きを追跡する必要がありますが、何がスワイプを構成するのかを決めるのはデベロッパ自身です。つまり、ユーザの指が十分な距離を移動したか、きちんと直線的に移動したか、また十分な速度で移動したかを判断する必要があります。これは、最初のタッチの場所を保存し、それを引き続くタッチ移動イベントで報告される位置と比較して行います。

リスト 3-2に、ビュー内での水平スワイプを検出するために使用できる基本的な追跡手法を示しています。この例では、ビューが、タッチの開始位置をstartTouchPositionメンバ変数に保存します。ユーザの指の移動に伴って、コードはタッチの現在位置と開始位置とを比較してそれがスワイプかどうかを判断します。タッチの垂直方向の移動が大きすぎる場合は、スワイプではないとみなされて別の処理が行われます。しかし、指が水平の軌道をたどっている場合は、それがスワイプであるとみなしてコードの処理が継続します。スワイプが水平方向に十分に移動してジェスチャが完了したと見なされたときに、処理ルーチンはアクションをトリガします。垂直方向のスワイプジェスチャを検出するには、xコンポーネントとyコンポーネントを入れ替えただけの同様のコードを使用します。

リスト 3-2　ビュー内のスワイプジェスチャの追跡

```
#define HORIZ_SWIPE_DRAG_MIN  12
#define VERT_SWIPE_DRAG_MAX    4

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    startTouchPosition = [touch locationInView:self];
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    CGPoint currentTouchPosition = [touch locationInView:self];

    // スワイプが正しい軌跡をたどっている場合
    if (fabsf(startTouchPosition.x - currentTouchPosition.x) >=
HORIZ_SWIPE_DRAG_MIN &&
        fabsf(startTouchPosition.y - currentTouchPosition.y) <=
VERT_SWIPE_DRAG_MAX)
```

```
    {
        // スワイプだとみなす
        if (startTouchPosition.x < currentTouchPosition.x)
            [self myProcessRightSwipe:touches withEvent:event];
        else
            [self myProcessLeftSwipe:touches withEvent:event];
    }
    else
    {
        // スワイプ以外のイベントの処理
    }
}
```

複雑なマルチタッチシーケンスの処理

タップとスワイプは単純なジェスチャです。より複雑なマルチタッチシーケンスの処理（つまり、アプリケーション固有のジェスチャの解釈）は、アプリケーションが実行しようとしている内容によって異なります。全フェーズを通してすべてのタッチを追跡して、変化のあったタッチ属性を記録し、内部状態を適切に変更しなければならない場合もあります。

複雑なマルチタッチシーケンスをどのように処理したらよいかを伝えるのに一番よい方法は、例を示すことです。リスト 3-3に、UIViewカスタムオブジェクトでどのようにタッチに対応するかを示します。ここでは、指の動きに合わせて「Welcome」という掲示が画面の中を動き回る様子をアニメーション化し、ユーザのダブルタップジェスチャに応じて「Welcome」の言語を変化させることでタッチに対応します（このコード例は、*MoveMe*サンプルコードプロジェクトに由来します。このサンプルプロジェクトを参照することで、イベント処理のコンテキストについての理解を深めることができます）。

リスト 3-3　複雑なマルチタッチシーケンスの処理

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [[event allTouches] anyObject];
    // タッチがplacardビューの内側で生じた場合にのみplacardビューを移動する
    if ([touch view] != placardView) {
        // placardビューの外側でダブルタップが生じた場合、placardの表示文字列を更新する
        if ([touch tapCount] == 2) {
            [placardView setupNextDisplayString];
        }
        return;
    }
    // placardビューを拡大してから縮小することによって、placardビューの"脈動"を表す
    // UIViewの組み込みアニメーション機能を使用する
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.5];
    CGAffineTransform transform = CGAffineTransformMakeScale(1.2, 1.2);
    placardView.transform = transform;
    [UIView commitAnimations];

    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.5];
    transform = CGAffineTransformMakeScale(1.1, 1.1);
    placardView.transform = transform;
    [UIView commitAnimations];

    // placardViewをタッチの下に移動する
```

```
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.25];
    placardView.center = [self convertPoint:[touch locationInView:self]
fromView:placardView];
    [UIView commitAnimations];
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [[event allTouches] anyObject];

    // タッチがplacardView内で生じた場合には、その位置までplacardViewを移動する
    if ([touch view] == placardView) {
        CGPoint location = [touch locationInView:self];
        location = [self convertPoint:location fromView:placardView];
        placardView.center = location;
        return;
    }
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [[event allTouches] anyObject];

    // タッチがplacardView内で生じた場合には、中央に戻す
    if ([touch view] == placardView) {
        // インタラクティブ操作を無効にして、以降のタッチがアニメーションに影響しないように
する
        self.userInteractionEnabled = NO;
        [self animatePlacardViewToCenter];
        return;
    }
}
```

注： 一般に、処理対象のイベントに応答して自分自身を再描画するようなカスタムビューは、イベント処理メソッドの中では描画状態の設定だけを行い、すべての描画処理を`drawRect:`メソッドで実行するようにします。ビューコンテンツの描画の詳細については、「グラフィックスと描画」（107 ページ）を参照してください。

イベント処理のテクニック

ここでは、コードで使用できるイベント処理テクニックを示します。

■ UITouchオブジェクトの変化を追跡する

イベント処理コードで、タッチの状態に関連するビット情報を保存しておき、後で、変遷後の`UITouch`インスタンスと比較することができます。たとえば、各タッチの終了位置を開始位置と比較する場合を考えます。`touchesBegan:withEvent:`メソッドで、`locationInView:`メソッドから各タッチの開始位置を取得し、`UITouch`オブジェクトのアドレスをキーにして、それを`CFDictionaryRef`不透過型に格納できます。その後、`touchesEnded:withEvent:`メソッドで、渡された各`UITouch`オブジェクトのアドレスを使用して、そのオブジェクトの開始位置を取得し現在位置と比較します（その際、`NSDictionary`オブジェクトではなく、`CFDictionaryRef`型を使用する必要があります。`NSDictionary`はアイテムをコピーしますが、`UITouch`クラスは、オブジェクトのコピーに必要な`NSCopying`プロトコルを採用していないからです）。

- **サブビューまたはレイヤを対象としたタッチのヒットテストを行う**

  カスタムビューは、タッチを受け取ったサブビューまたはレイヤを見つけて、イベントを適切に処理するために、`UIView`の`hitTest:withEvent:`メソッド、または`CALayer`ビューの`hitTest:`メソッドを使用できます。次の例は、カスタムビューのレイヤ内にある「Info」画像がタップされたことを検出するコードです。

```
- (void)touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event {
    CGPoint location = [[touches anyObject] locationInView:self];
    CALayer *hitLayer = [[self layer] hitTest:[self convertPoint:location
fromView:nil]];

    if (hitLayer == infoImage) {
        [self displayInfo];
    }
}
```

  カスタムビューがサブビューを持つ場合は、タッチをサブビューレベルで処理するか、スーパービューレベルで処理するかを判断する必要があります。サブビューにおいて、`touchesBegan:withEvent:`、`touchesEnded:withEvent:`、または`touchesMoved:withEvent:`を実装してタッチを処理しない場合は、これらのメッセージはレスポンダチェーンに沿ってさかのぼり、スーパービューに送られます。ところが、マルチタップやマルチタッチは、それらが最初に発生したサブビューに関連付けられているため、スーパービューではこれらのタッチを受け取ることができません。すべての種類のタッチを確実に受け取れるようにするには、スーパービューで`hitTest:withEvent:`をオーバーライドして、サブビューではなく自分自身を返すようにします。

- **マルチタッチシーケンスの最後の指が離れたことを検出する**

  マルチタッチシーケンスの最後の指がビューを離れたことを検出するには、渡された`UITouch`オブジェクトの数と、渡された`UIEvent`オブジェクトが管理しているビューのタッチ数を比較します。たとえば次のようにします。

```
- (void)touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event {
    if ([touches count] == [[event touchesForView:self] count]) {
        // 最後の指が離れた...
    }
}
```

モーションイベント

iPhoneまたはiPod touchデバイスは、ユーザが特定の方法（デバイスをシェイクするなど）でデバイスを動かした場合にモーションイベントを生成します。モーションイベントの発生元はデバイスの加速度センサーです。システムは加速度センサーのデータを評価してそれが一定の基準を満たす場合にジェスチャとして解釈します。そして、このジェスチャを表す`UIEvent`オブジェクトを作成し、処理するために現在アクティブなアプリケーションに対してイベントオブジェクトを送信します。

注： iPhone 3.0では、ジェスチャとして解釈されてモーションイベントになるのはシェイク動作だけです。

モーションイベントはタッチイベントよりもはるかに単純です。システムは、モーションの始まったときと止まったときをアプリケーションに通知するだけで、個々のモーションが発生したタイミングは伝えません。さらに、タッチイベントには、タッチとそれに関連する情報のセットが含まれているのに対して、モーションイベントには、イベントタイプ、イベントサブタイプ、およびタイムスタンプ以外の状態は含まれていません。システムは、向きの変化と混同しないような方法でモーションジェスチャを解釈します。

モーションイベントを処理するには、`UIResponder`から派生したクラスに、`motionBegan:withEvent:`メソッドと`motionEnded:withEvent:`メソッドのいずれか、または両方を実装する必要があります。たとえば、アプリケーションで水平シェイクと垂直シェイクに異なる意味を割り当てたい場合は、`motionBegan:withEvent:`で現在の加速度軸の値をキャッシュし、`motionEnded:withEvent:`でキャッシュされた値と同じ軸の値を比較して、その結果に応じた処理を実行します。また、レスポンダは`motionCancelled:withEvent:`メソッドを実装して、システムがモーションイベントをキャンセルするために送信するイベントに対応する必要があります。これらのイベントはしばしば、モーションが結局は有効なジェスチャではなかったということを反映します。

アプリケーションとそのキーウインドウは、モーションイベントをウインドウのファーストレスポンダに処理のために送付します。ファーストレスポンダがそれを処理しない場合は、タッチイベントと同様に、イベントはそれが処理されるか無視されるまでレスポンダチェーンをさかのぼります（詳細については「イベントの送付」（87 ページ）を参照）。ただし、タッチイベントとシェイクモーションイベントには重要な違いが1つあります。ユーザがデバイスのシェイクを開始すると、システムは`motionBegan:withEvent:`メッセージによってモーションイベントをファーストレスポンダに送信します。ファーストレスポンダがそのイベントを処理しない場合、そのメッセージはレスポンダチェーンをさかのぼります。シェイクの持続時間が約1秒未満の場合、システムはファーストレスポンダに`motionEnded:withEvent:`メッセージを送信します。一方、シェイクの持続時間がそれよりも長い場合やシステムがそのモーションをシェイクだと判断しなかった場合は、ファーストレスポンダは`motionCancelled:withEvent:`メッセージを受信します。

シェイクモーションイベントが処理されないままレスポンダチェーンをさかのぼってウインドウに到達したときに、`UIApplication`の`applicationSupportsShakeToEdit`プロパティが`YES`に設定されている場合、iPhone OSは「取り消し(Undo)」コマンドと「やり直し(Redo)」コマンドを持つシートを表示します。デフォルトではこのプロパティは`NO`に設定されています。

コピー操作、カット操作、ペースト操作

iPhone OS 3.0から、ユーザは1つのアプリケーション内でテキスト、画像、その他のデータをコピーして、そのデータを同じアプリケーション内または別のアプリケーションにペーストできるようになりました。たとえば、電子メールメッセージの誰かのアドレスをコピーして、「連絡先(Contacts)」アプリケーションでそれを適切なフィールドにペーストできます。UIKitフレームワークでは現在のところ、コピー-カット-ペーストは`UITextView`クラス、`UITextField`クラス、および`UIWebView`クラスで実装されています。自分のアプリケーションにおいてこの動作が必要な場合は、これらのクラスのオブジェクトを使用するか、コピー-カット-ペーストを実装します。

以降の各セクションでは、コピー、カット、およびペーストの各操作に使用するUIKitのプログラムインターフェイスと、それらの使いかたを説明します。

注： コピー操作とペースト操作に関連する使用ガイドラインについては、『*iPhone Human Interface Guidelines*』の「Supporting Copy and Paste」を参照してください。

コピー-ペースト操作のためのUIKitの機能

UIKitフレームワークのいくつかのクラスと非形式プロトコルでは、コピー、カット、およびペーストの各操作をアプリケーションで実装するために必要なメソッドを提供します。

- `UIPasteboard`クラスは、ペーストボード（1つのアプリケーション内またはアプリケーション間でデータを共有するための保護された領域）を提供します。このクラスは、ペーストボードとの間でデータアイテムを読み書きするためのメソッドを提供します。
- `UIMenuController`クラスは、コピー、カット、またはペーストする選択領域の上または下に編集メニューを表示します。編集メニューのコマンドには、（必要に応じて）「コピー(Copy)」、「カット(Cut)」、「ペースト(Paste)」、「選択(Select)」、および「全選択(Select All)」があります。
- `UIResponder`クラスには、`canPerformAction:withSender:`メソッドが宣言されています。レスポンダクラスはこのメソッドを実装して、現在のコンテキストに応じて編集メニューのコマンドを表示したり削除したりできます。
- `UIResponderStandardEditActions`非形式プロトコルには、コピー(Copy)、カット(Cut)、ペースト(Paste)、選択(Select)、および全選択(Select All)の各コマンドを処理するためのインターフェイスが宣言されています。ユーザが編集メニューのコマンドの1つのタップすると、それに対応する`UIResponderStandardEditActions`メソッドが呼び出されます。

ペーストボードの概念

ペーストボードは、アプリケーション内またはアプリケーション間でデータをやり取りするための標準化されたメカニズムです。ペーストボードが最もよく使われるのは、次のようなコピー操作、カット操作、およびペースト操作の処理です。

- ユーザがアプリケーション内のデータを選択し、メニューコマンドから「コピー(Copy)」（または「カット(Cut)」）を選ぶと、選択されたデータがペーストボードに格納されます。
- （同じアプリケーション、または別のアプリケーションで）ユーザが「ペースト(Pastea)」メニューコマンドを選ぶと、ペースボード上のデータが現在のアプリケーションにコピーされます。

iPhone OSでは、ペーストボードは「検索(Find)」操作をサポートするためにも使われます。さらに、コピー、カット、およびペーストの各コマンドの代わりにカスタムURLスキーマを使用して、アプリケーション間でデータを転送するためにペーストボードを使用することもできます。この手法の詳細については「他のアプリケーションとのやり取り」（39 ページ）を参照してください。

これらのどの操作においても、ペーストボードオブジェクトを利用して実行する基本的なタスクは、ペーストボードへのデータの書き込みとペーストボードからのデータの読み取りです。これらのタスクは概念的には単純ですが、そこには重要な詳細情報がいくつか隠れています。最も複雑な点は、データを表現する方法がいくつかあることです。この複雑さのために効率を考慮しなければなりません。以降の各セクションで、これらの問題とその他の問題について説明します。

名前付きペーストボード

ペーストボードには公開のものと非公開のものがあります。公開ペーストボードは**システムペーストボード**と呼ばれます。非公開ペーストボードはアプリケーションによって作成されます。したがって、**アプリケーションペーストボード**と呼ばれます。ペーストボードは一意の名前を持っていなければなりません。`UIPasteboard`には2つのシステムペーストボードが定義されており、それぞれ次のような名前と目的を持っています。

- `UIPasteboardNameGeneral`は、幅広いデータ型に関係するカット、コピー、およびペースト操作に使われます。`generalPasteboard`クラスメソッドを呼び出すことによって、Generalペーストボードを表すシングルトンオブジェクトを取得できます。
- `UIPasteboardNameFind`は検索操作に使われます。ユーザがSerach Bar (`UISearchBar`)に入力した文字列がこのペーストボードに書き込まれます。これによって複数のアプリケーション間でこの文字列を共有できます。名前に`UIPasteboardNameFind`を渡して`pasteboardWithName:create:`クラスメソッドを呼び出すことによって、Findペーストボードを表すオブジェクトを取得できます。

通常はシステム定義のペーストボードのいずれか1つを使用しますが、必要であれば、`pasteboardWithName:create:`を使用してアプリケーション固有のペーストボードを作成することもできます。`pasteboardWithUniqueName`を呼び出すと、`UIPasteboard`によって一意の名前を持つアプリケーションペーストボードを取得できます。ペーストボードの名前は、`name`プロパティによって変えられます。

ペーストボードの永続性

ペーストボードには、それを使用しているアプリケーションが終了した後も存続するように永続性を示すマークを付けることができます。永続的でないペーストボードは、それを作成したアプリケーションが終了したときに削除されます。システムペーストボードは永続的です。アプリケーションペーストボードはデフォルトでは永続的ではありません。アプリケーションペーストボードを永続的にするには`persistent`プロパティを`YES`に設定します。永続的なアプリケーションペーストボードは、それを所有するアプリケーションをユーザがアンインストールしたときに削除されます。

ペーストボードの所有者とアイテム

ペーストボードに最後にデータを格納したオブジェクトは、そのペーストボードの**所有者**と呼ばれます。ペーストボードに格納されたデータの各部分は、ペーストボードの**アイテム**と見なされます。ペーストボードは単一のアイテムまたは複数のアイテムを保持できます。アプリケーションは、必要な数だけいくつでもアイテムを格納したり取得したりできます。たとえば、ユーザがテキストと画像の両方を含むビューで選択を行ったとします。ペーストボードによってこのテキストと画像は別々のアイテムとしてペーストボードにコピーされます。ペーストボードから複数のアイテムを読み取るアプリケーションは、そのアプリケーションがサポートするアイテムだけを選択して取得することができます（たとえば、テキストは取得するが画像は取得しないなど）。

重要： アプリケーションがデータをペーストボードに書き込むと、それが単一のアイテムであったとしてもペーストボードの現在の内容はそのデータに置き換わります。`UIPasteboard`の`addItems:`メソッドを使用するとアイテムを追加できますが、このクラスの書き込みメソッドはペーストボードの現在の内容にアイテムを追加しません。

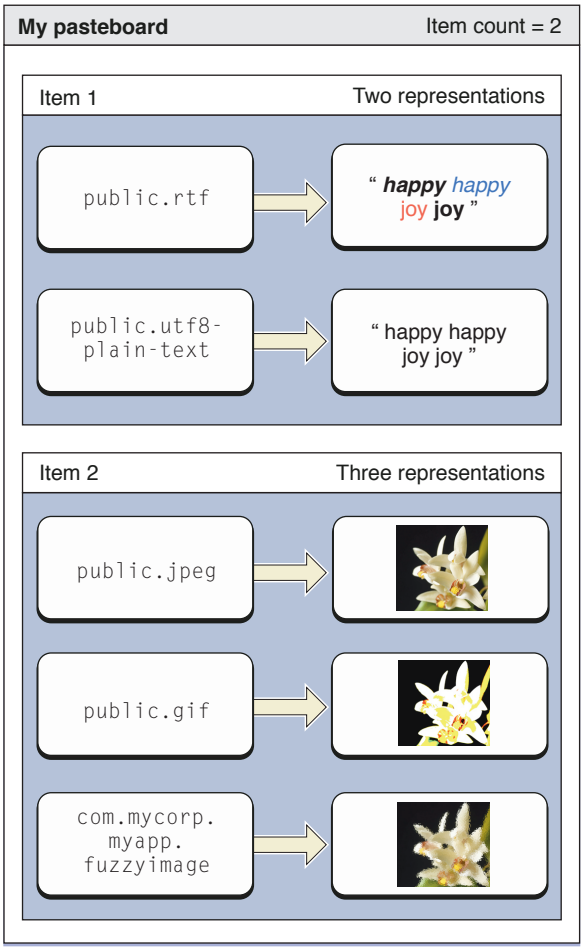
データ表現とUTI

ペーストボードの操作は、2つの異なるアプリケーション間で実行されるのが通常です。どちらのアプリケーションも相手のアプリケーションについて（そのアプリケーションが扱うことができるデータの種類も含む）知っている必要がありません。共有できる可能性を最大限に高めるために、ペーストボードは同じペーストボードアイテムに対して複数の**表現**を持つことができます。たとえば、リッチテキストエディタは、コピーされたデータに対してHTML、PDF、およびプレーンテキストの表現を提供できます。ペーストボード上の1つのアイテムには、そのデータに対してアプリケーションで提供可能なすべての表現が含まれています。

ペーストボードアイテムの各表現は、通常UTI (Unique Type Identifier)で識別されます（UTIは、特定のデータ型を一意に識別する単純な文字列です）。UTIはデータ型を識別するための共通の手段を提供します。カスタムのデータ型をサポートしたい場合は、それに対応する一意の識別子を作成する必要があります。それには一意性を保証する表現型文字列として逆DNS形式を使用できます。たとえば、カスタム表現型として`com.myCompany.myApp.myType`を使用できます。UTIの詳細については、『*Uniform Type Identifiers Overview*』を参照してください。

たとえば、アプリケーションがリッチテキストと画像の選択をサポートするとします。アプリケーションは、1つのテキスト選択に対してはリッチテキストとUnicodeの両方のバージョン、1つの画像選択に対してはさまざまな表現をペーストボードに格納できます。各アイテムのそれぞれの表現はそれ固有のデータと一緒に格納されます。図 3-3はその様子を示しています。

図 3-3 ペーストボードアイテムと表現



一般に、共有できる可能性を最大限に高めるために、ペーストボードアイテムにはできる限り多くの種類の表現を含めるべきです。

ペーストボードリーダは（もしあれば）その機能に最も適したデータ型を見つけなければなりません。通常これは、最も豊富な情報量を持つ型を選択することを意味します。たとえば、テキストエディタは、コピーされたデータに対して、HTML（リッチテキスト）とプレーンテキストの表現を提供できます。リッチテキストをサポートするアプリケーションはHTML表現を取得すべきであり、プレーンテキストのみをサポートするアプリケーションはプレーンテキストバージョンを取得すべきです。

変更カウント

変更カウントは、ペーストボードの内容が変更されるたびに（特に、アイテムが追加、変更、または削除されたときに）インクリメントされるペーストボードごとの変数です。（`changeCount`プロパティを介して）変更カウントを調べることによって、アプリケーションはペーストボード内の現在のデータが最後に取得したデータと同じかどうかを判断できます。変更カウントがインクリメントされるたびにペーストボードは関連するオブザーバに通知を送信します。

選択とメニューの管理

ビュー内の何かをコピーまたはカットするにはその「何か」を選択しなければなりません。それは一定範囲のテキスト、画像、URL、色、またはその他のデータ表現（カスタムオブジェクトを含む）の可能性があります。カスタムビューでコピーアンドペースト動作を実装するには、そのビュー自身でオブジェクトの選択を管理しなければなりません。特定のタッチジェスチャ（たとえば、ダブルタップ）によってユーザがビュー内のオブジェクトを選択した場合は、そのイベントを処理する必要があります。内部的にはその選択を記録して（以前の選択を解除し）、ビュー内に新しい選択範囲を表示する必要があります。ユーザがコピー、カット、およびペーストの各操作でビュー内の複数のオブジェクトを選択できる場合は、複数選択動作を実装する必要があります。

注： タッチイベントとそれを処理する方法については、「タッチイベント」（85 ページ）で説明しています。

ユーザが編集メニュー（選択を行うアクションの場合もある）を要求したとアプリケーションが判断した場合は、次の手順に従ってメニューを表示すべきです。

1. `UIMenuController`の`sharedMenuController`クラスメソッドを呼び出して、グローバルのメニューコントローラインスタンスを取得します。
2. 選択範囲の境界を計算し、その結果得られた矩形を利用して`setTargetRect:inView:`メソッドを呼び出します。編集メニューがこの矩形の上または下に表示されます。上か下かは選択範囲が画面の上端または下端にどの程度近いかによって決まります。
3. （両方の引数に`YES`を渡して）`setMenuVisible:animated:`メソッドを呼び出し、編集メニューを選択範囲の上または下に表示するアニメーションを作成します。

リスト 3-4は、`touchesEnded:withEvent:`メソッドの実装内で編集メニューを表示する方法を示しています（この例では選択を処理するコードの部分は省略されています）。次のコードでは、カスタムビューが自身に`becomeFirstResponder`メッセージを送信して、そのビューがその後のコピー、カット、およびペースト操作のファーストレスポンダであることを保証している様子も示しています。

リスト 3-4　編集メニューの表示

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *theTouch = [touches anyObject];

    if ([theTouch tapCount] == 2  && [self becomeFirstResponder]) {

        // 選択を管理するコードがここに来る...

        // 編集メニューを表示する
        UIMenuController *theMenu = [UIMenuController sharedMenuController];
       CGRect selectionRect = CGRectMake(currentSelection.x, currentSelection.y,
 SIDE, SIDE);
        [theMenu setTargetRect:selectionRect inView:self];
        [theMenu setMenuVisible:YES animated:YES];

    }
}
```

メニューには、最初、ファーストレスポンダが対応するUIResponderStandardEditActionsメソッド実装（copy:、paste:など）のすべてのコマンド含まれています。ただし、メニューを表示する前にシステムはファーストレスポンダ（多くの場合はそのカスタムビュー自身）にcanPerformAction:withSender:メッセージを送信します。このメソッドの実装内で、レスポンダは、そのコマンド（第1引数のselectorで表される）が現在のコンテキストで適用可能かどうかを評価します。たとえば、セレクタがpaste:で、ビューが扱うことができる型のペーストボード内にデータが存在しない場合、レスポンダはNOを返して「ペースト(Paste)」コマンドを非表示にする必要があります。ファーストレスポンダがcanPerformAction:withSender:メソッドを実装していない場合や、指定されたコマンドを処理しない場合、そのメッセージはレスポンダチェーンをさかのぼります。

リスト 3-5は、cut:、copy:、およびpaste:の各セレクタに一致するメッセージを検索するcanPerformAction:withSender:メソッドの実装を示しています。このメソッドでは、現在の選択コンテキスト（ペーストコマンドの場合は、ペーストボードの内容）に基づいてコピー、カット、ペーストの各メニューコマンドを有効または無効にします。

リスト 3-5　条件によってメニューコマンドを有効にする

```
- (BOOL)canPerformAction:(SEL)action withSender:(id)sender {
    BOOL retValue = NO;
    ColorTile *theTile = [self colorTileForOrigin:currentSelection];

    if (action == @selector(paste:))
        retValue = (theTile == nil) &&
             [[UIPasteboard generalPasteboard] containsPasteboardTypes:
             [NSArray arrayWithObject:ColorTileUTI]];
    else if ( action == @selector(cut:)|| action == @selector(copy:))
        retValue = (theTile != nil);
    else
        retValue = [super canPerformAction:action withSender:sender];
    return retValue;
}
```

このメソッドの最後のelse節では、スーパークラスの実装を呼び出してサブクラスが無視したコマンドを処理する機会をスーパークラスに与えています。

あるメニューコマンドが実行されたことによって、ほかのメニューコマンドのコンテキストが変化する場合があります。たとえば、ユーザがビュー内のすべてのオブジェクトを選択した場合は、「コピー(Copy)」コマンドと「カット(Cut)」コマンドをメニューに含める必要があります。この場合、レスポンダは、メニューが表示されている間はメニューコントローラのupdateを呼び出すことができます。その結果、ファーストレスポンダのcanPerformAction:withSender:が再度呼び出されます。

選択内容のコピーとカット

ユーザが編集メニューの「コピー(Copy)」コマンドまたは「カット(Cut)」コマンドをタップすると、システムはレスポンダオブジェクトが実装しているcopy:メソッドまたはcut:メソッドをそれぞれ呼び出します。通常はファーストレスポンダ（カスタムビュー）がこれらのメソッドを実装していますが、ファーストレスポンダがこれらのメソッドを実装していない場合はいつものようにメッセージはレスポンダをさかのぼります。UIResponderStandardEditActions非形式プロトコルでこれらのメソッドは宣言されています。

注： UIResponderStandardEditActionsは非形式プロトコルのため、アプリケーションのどのクラスでもそのメソッドを実装できます。ただし、レスポンダチェーンをさかのぼるためのデフォルトの動作を活用するには、このメソッドを実装するクラスがUIResponderから派生しており、レスポンダチェーンにインストールされている必要があります。

copy:またはcut:メッセージに応答して、デベロッパは、選択範囲で示されたオブジェクトまたはデータをできる限り多くの種類の表現でペーストボードに書き込みます。この操作には次の手順が伴います（単一のペーストボードアイテムを想定した場合）。

1. 選択範囲から、オブジェクトまたはそのオブジェクトに対応するバイナリデータを識別または取得する。

   バイナリデータは、NSDataオブジェクトにカプセル化されている必要があります。別のタイプのオブジェクトをペーストボードに書き込む場合は、そのオブジェクトはプロパティリストオブジェクト（NSString、NSArray、NSDictionary、NSDate、NSNumber、またはNSURLのいずれかのオブジェクト）でなければなりません（プロパティリストオブジェクトの詳細については、『*Property List Programming Guide*』を参照してください）。

2. 可能な場合は、そのオブジェクトまたはデータに対して1つ以上の表現を作成する。

   たとえば、前の手順で選択画像を表すUIImageオブジェクトを作成した場合は、UIImageJPEGRepresentation関数とUIImagePNGRepresentation関数を使用して、この画像を異なる表現に変換できます。

3. ペーストボードオブジェクトを取得する。

   多くの場合、これはGeneralペーストボード（generalPasteboardクラスメソッドを利用して取得できる）です。

4. ペーストボードアイテムに書き込まれたそれぞれのデータ表現に、適切なUTIを割り当てる。

   このテーマに関する説明については、「ペーストボードの概念」（97 ページ）を参照してください。

5. 表現タイプごとに最初のペーストボードアイテムにデータを書き込む。

   - データオブジェクトを書き込むにはペーストボードオブジェクトにsetData:forPasteboardType:メッセージを送信します。
   - プロパティリストオブジェクトを書き込むにはペーストボードオブジェクトにsetValue:forPasteboardType:メッセージを送信します。

6. コマンドが「カット(Cut)」（cut:メソッド）の場合は、選択範囲で示されたオブジェクトをアプリケーションのデータモデルから削除してビューを更新します。

リスト 3-6は、copy:メソッドとcut:メソッドの実装を示しています。cut:メソッドは、copy:メソッドを呼び出し、その後選択されているオブジェクトをビューおよびデータモデルから削除します。copy:メソッドはカスタムオブジェクトをアーカイブして、setData:forPasteboardType:によってペーストボードに渡すことができるNSDataオブジェクトを取得している点に注意してください。

リスト 3-6　コピー操作とカット操作

```
- (void)copy:(id)sender {
    UIPasteboard *gpBoard = [UIPasteboard generalPasteboard];
    ColorTile *theTile = [self colorTileForOrigin:currentSelection];
    if (theTile) {
       NSData *tileData = [NSKeyedArchiver archivedDataWithRootObject:theTile];
        if (tileData)
            [gpBoard setData:tileData forPasteboardType:ColorTileUTI];
    }
}

- (void)cut:(id)sender {
     [self copy:sender];
     ColorTile *theTile = [self colorTileForOrigin:currentSelection];

     if (theTile) {
         CGPoint tilePoint = theTile.tileOrigin;
         [tiles removeObject:theTile];
          CGRect tileRect = [self rectFromOrigin:tilePoint inset:TILE_INSET];
         [self setNeedsDisplayInRect:tileRect];
     }
}
```

選択内容のペースト

ユーザが編集メニューの「ペースト(Paste)」コマンドをタップすると、システムはレスポンダオブジェクトが実装しているpaste:メソッドを呼び出します。通常はファーストレスポンダ（カスタムビュー）がこのメソッドを実装していますが、ファーストレスポンダがこのメソッドを実装していない場合はいつものようにメッセージはレスポンダをさかのぼります。paste:メソッドはUIResponderStandardEditActions非形式プロトコルで宣言されています。

paste:メッセージに応答して、デベロッパは、アプリケーションがサポートする表現でオブジェクトをペーストボードから読み取ります。次に、ペーストされたオブジェクトをアプリケーションのデータモデルに追加し、ビュー内のユーザが指定した場所にこの新規オブジェクトを表示します。この操作には次の手順が伴います（単一のペーストボードアイテムを想定した場合）。

1. ペーストボードオブジェクトを取得する。

   多くの場合、これはGeneralペーストボード（generalPasteboardクラスメソッドを利用して取得できる）です。

2. containsPasteboardTypes:メソッドまたはpasteboardTypesメソッドを呼び出して、戻されたタイプ配列を調べることによって、最初のペーストボードアイテムにアプリケーションで扱うことができる表現のデータが含まれているか確認する。

   この手順は、canPerformAction:withSender:の実装ですでに実行しているはずです。

3. ペーストボードの最初のアイテムにアプリケーションで扱うことができるデータが含まれている場合は、次のいずれかのメソッドを呼び出してそれを読み取る。

   - dataForPasteboardType:（読み取るデータがNSDataオブジェクトにカプセル化されている場合）

- valueForPasteboardType:（読み取るデータがプロパティリストオブジェクトにカプセル化されている場合（「選択内容のコピーとカット」（102 ページ）を参照））

4. オブジェクトをアプリケーションのデータモデルに追加する。
5. オブジェクトの表現をユーザインターフェイス内のユーザが指定した場所に表示する。

リスト 3-7は、paste:メソッドの実装例です。ここでは、cut:メソッドとcopy:メソッドの組み合わせの逆を実行しています。カスタムビューは、まずGeneralペーストボードにデータのカスタム表現が格納されているかどうかを調べます。格納されている場合は、そのデータをペーストボードから読み取りアプリケーションのデータモデルに追加します。そして、その部分（現在の選択範囲）に再描画用のマークを付けます。

リスト 3-7　ペーストボードのデータを選択範囲にペーストする

```
- (void)paste:(id)sender {
     UIPasteboard *gpBoard = [UIPasteboard generalPasteboard];
     NSArray *pbType = [NSArray arrayWithObject:ColorTileUTI];
     ColorTile *theTile = [self colorTileForOrigin:currentSelection];
     if (theTile == nil && [gpBoard containsPasteboardTypes:pbType]) {

        NSData *tileData = [gpBoard dataForPasteboardType:ColorTileUTI];
        ColorTile *theTile = (ColorTile *)[NSKeyedUnarchiver
unarchiveObjectWithData:tileData];
         if (theTile) {
             theTile.tileOrigin = self.currentSelection;
             [tiles addObject:theTile];
             CGRect tileRect = [self rectFromOrigin:currentSelection
inset:TILE_INSET];
             [self setNeedsDisplayInRect:tileRect];
         }
     }
}
```

編集メニューの消去

cut:、copy:、またはpaste:コマンドの実装から戻ると、編集メニューは自動的に非表示になります。次のコードを利用すると編集メニューを表示したままにできます。

```
[UIMenuController setMenuController].menuVisible = YES;
```

システムはいつでも編集メニューを非表示にできます。たとえば、警告が表示されたりユーザが画面の別の領域をタップした場合、システムは編集メニューを非表示にします。編集メニューが表示されているかどうかに応じて状態や表示を変更したい場合は、UIMenuControllerWillHideMenuNotificationという名前の通知をリッスンして適切なアクションを実行する必要があります。

グラフィックスと描画

高品質のグラフィックスは、アプリケーションのユーザインターフェイスのなかで重要な部分を占めます。高品質のグラフィックスを提供することでアプリケーションの外観がよくなるだけでなく、アプリケーションをシステムのほかの部分の自然な延長のように見せることができます。iPhone OSには、システムで高品質のグラフィックスを作成できるよう、Open GLのほかに、QuartzやCore Animation、UIKitを使用したネイティブレンダリングという、2つの手段が用意されています。

Open GLフレームワークは主に、ゲームの開発や、高フレームレートを要求するアプリケーションを対象にしています。Open GLは、デスクトップコンピュータ上で2Dや3Dのコンテンツを作成するために使用する、C言語ベースのインターフェイスです。iPhone OSは、OpenGL ESフレームワークを通じてOpenGLの描画をサポートしています。OpenGL ESフレームワークは、OpenGL ES 2.0およびOpenGL ES v1.1のどちらの仕様もサポートしています。OpenGL ESは、特に組み込みハードウェアシステム上での使用を念頭において設計されており、デスクトップバージョンのOpenGLとは多くの点で異なります。

よりオブジェクト指向の描画手法を望むデベロッパのため、iPhone OSはQuartz、Core Animation、そしてUIKitのグラフィックスサポート機能を提供しています。Quartzはメインの描画インターフェイスで、パスベースの描画、アンチエイリアスを施したレンダリング、グラデーション付き塗りつぶしパターン、画像、カラー、座標空間変換、PDF文書の作成、表示、解析をサポートします。UIKitは、Quartzの画像とカラーの操作向けにObjective-Cラッパーを提供します。Core Animationは、UIKitの多くのビュープロパティについて変更をアニメーション化するための基盤をサポートします。カスタムアニメーションの実装に使用することもできます。

この章では、iPhoneアプリケーションの描画プロセスの概要を取り上げます。また、サポートする描画テクノロジーごとに、描画テクニックも紹介します。さらに、iPhone OSプラットフォームの描画コードを最適化するためのヒントやガイダンスも示します。

UIKitのグラフィックスシステム

iPhone OSでは、OpenGL、Quartz、UIKit、またはCore Animationのいずれによるものかに関係なく、すべての描画が`UIView`オブジェクトの範囲内で実行されます。ビューは、描画が発生する、画面内の領域を規定します。システムに用意されているビューを使用する場合、ビューの描画は自動的に処理されます。しかし、カスタムビューを定義する場合は、自ら描画コードを用意する必要があります。OpenGLを使用して描画するアプリケーションでは、レンダリングサーフェスをセットアップしたら、OpenGLが指定する描画モデルを使用します。

Quartz、Core Animation、およびUIKitの場合、以降の各セクションで説明する描画概念を用います。

ビューの描画サイクル

`UIView`オブジェクトの基本的な描画モデルでは、要求に応じてコンテンツを更新します。ただし、`UIView`クラスでは、更新要求を集め、最も適切なタイミングで描画コードに引き渡すことにより、更新プロセスをより簡単かつ効率的に実行できます。

ビューの一部が再描画を必要とする場合は必ず、そのUIViewオブジェクトに組み込まれている描画コードが、このオブジェクトのdrawRect:メソッドを呼び出します。描画コードはこのメソッドに対し、ビューのなかで再描画が必要な部分を示す矩形を渡します。カスタムビューサブクラスではこのメソッドをオーバーライドして、ビューのコンテンツを描画するために使用します。初めてビューが描画されるとき、drawRect:メソッドに渡される矩形には、ビューの表示されている領域全体が含まれています。しかし、この矩形はその後の呼び出しでは、ビューのなかで実際に再描画が必要な部分だけを表します。次に示すアクションは、ビュー更新のトリガとなります。

- ビューの一部を隠している別のビューの移動または除去
- 非表示になっていたビューの再表示（hiddenプロパティをNOに設定）
- ビューを画面外までスクロールし、スクロールし戻す
- ビューのsetNeedsDisplayメソッドまたはsetNeedsDisplayInRect:メソッドの明示的な呼び出し

drawRect:メソッドを呼び出した後、ビューは自らを更新済みとしてマークを付け、新たなアクションが到着して別の更新サイクルがトリガされるのを待機します。ビューに静的コンテンツが表示されている場合、必要となるのは、スクロールやほかのビューの存在によって生じる、ビューの見え方の変化に対応することだけです。しかし、ビューのコンテンツを定期的に更新する場合は、更新をトリガするsetNeedsDisplayメソッドまたはsetNeedsDisplayInRect:メソッドを呼び出すタイミングを指定しなければなりません。たとえば、1秒あたり数回コンテンツを更新する場合、タイマーを設定してビューを更新することができます。ビューを更新するタイミングとしては、ユーザによるインタラクティブな操作やビュー内での新しいコンテンツの作成に応答したりする場合も考えられます。

座標と座標変換

「ビューの座標系」（63 ページ）で説明したように、ウインドウまたはビューの原点は左上隅に置かれ、正数の座標値は原点から下および右に向かって大きくなります。描画コードを記述するときは、この座標系を使用して、描画対象のコンテンツの個々の点の場所を指定します。

デフォルトの座標系を変更する必要がある場合は、現在の変換行列を変更します。**現在の変換行列(CTM：Current Transformation Matrix)**は数学的行列で、ビューの座標系上の点をデバイスの画面上の点に対応づけます。ビューのdrawRect:メソッドが最初に呼び出されるとき、座標系の原点がビューの原点と一致し、正数の座標値が下および右に向かって大きくなるように、CTMが設定されます。ただし、CTMに、拡大縮小、回転、変換などの係数を追加して変更を加えて、サイズ、向き、そして基盤になるビューやウインドウを基準にしたデフォルトの座標系の位置を変更することができます。

CTMに変更を加える方法は、手間が大幅に省けるため、ビューにコンテンツを描画する標準的手法として用いられています。現行描画システムにおいて、座標点(20, 20)から始まる10×10の正方形を描画する場合、(20, 20)に移動するパスを作成した後、必要な直線をひととおり描画して正方形を完成させます。ただし、この正方形を後で点(10, 10)まで移動することにした場合は、新たな開始点を指定してパスを作成し直す必要があります。実際、原点を変更するたびにパスを作成し直す必要が生じます。パスの作成は負担の多い操作です。これに比べて、原点が(0, 0)の正方形を作成し、望みの描画の原点に一致するようにCTMに変更を加える処理は少ない負担で済みます。

Core Graphicsフレームワークでは、CTMに変更を加える方法は2通りあります。CTMは、*CGContext Reference*で定義されているCTM操作関数を使用して直接変更できます。CGAffineTransform構造体を作成し、望みの変換を適用して、その変換をCTMに反映させることもできます。アフィン変換を使用して変換をグループとしてひとまとめにし、それらの変換をCTMに一度に適用することができ

ます。アフィン変換を評価、反転し、その結果を用いて、コード内の点、サイズ、矩形の値を変更することもできます。アフィン変換の使用の詳細については、『*Quartz 2D Programming Guide*』および『*CGAffineTransform Reference*』を参照してください。

グラフィックスコンテキスト

カスタムの`drawRect:`メソッドを呼び出す前に、ビューオブジェクトは自動的に描画環境を設定し、コードがただちに描画を開始できるようにします。描画環境の設定の一環として、現在の描画環境に対応する`UIView`オブジェクトはグラフィックスコンテキスト（`CGContextRef`不透過型）を作成します。このグラフィックスコンテキストには、描画システムがその後の描画コマンド実行に必要とする情報が格納されます。グラフィックスコンテキストは、描画時に使用する色、クリッピング領域、線の幅とスタイルの情報、フォント情報、合成オプションなど、基本的な描画属性を定義します。

ビュー以外の場所に描画したい場合は、カスタムのグラフィックスコンテキストオブジェクトを作成できます。Quartzでは、この作業は主に、一連の描画コマンドを取得して、画像やPDFファイルの作成にこれらのコマンドを使用したい場合に行います。グラフィックスコンテキストを作成するには、`CGBitmapContextCreate`関数または`CGPDFContextCreate`関数を使用します。グラフィックスコンテキストを作成したら、それをコンテンツの作成に必要な描画関数に渡すことができます。

カスタムグラフィックスコンテキストを作成する場合、そのコンテキストの座標系は、iPhone OSが使用するネイティブの座標系とは異なります。グラフィックスコンテキストの座標系の原点は、描画サーフェスの左上隅ではなく、左下隅に存在し、軸は上と右に伸びていきます。描画コマンドで指定する座標は、この点を考慮する必要があります。座標系の違いを考慮しない場合、結果として画像やPDFファイルが正しくレンダリングされない可能性があります。

重要： ビットマップまたはPDFのコンテキストを描画するときに左下の原点を使用するため、作成されたコンテンツをビューにレンダリングするときにその座標系を補正しなければなりません。言い換えると、画像を作成し`CGContextDrawImage`関数を使用して描画すると、その画像はデフォルトのままでは上下が逆になって表示されるということです。これを修正するには、CTMのY軸を反転（Y軸の座標に`-1`を乗じる）し、ビューの左下隅から左上隅へと原点を移動する必要があります。

作成する`CGImageRef`を`UIImage`オブジェクトを使用してラップする場合、CTMを修正する必要はありません。`UIImage`オブジェクトは、`CGImageRef`型の反転した座標系を自動的に補正します。

グラフィックスコンテキスト、グラフィックスの状態情報の修正、グラフィックスコンテキストを使用したカスタムコンテンツの作成の詳細については、『*Quartz 2D Programming Guide*』を参照してください。グラフィックスコンテキストに関連して用いる関数の一覧については、『*CGContext Reference*』、『*CGBitmapContext Reference*』、*CGPDFContext Reference*』を参照してください。

点とピクセル

Quartz描画システムは、ベクトルベースの描画モデルを使用します。描画コマンドが個々のピクセルを処理するラスタベースの描画モデルと比較して、Quartzの描画コマンドは、**ユーザ座標空間**と呼ばれる、縮尺が固定の描画空間を使用して指定されます。そしてiPhone OSは、この描画空間の座標を、デバイスの実際のピクセルに対応させます。この描画モデルの利点は、アフィン変換を使用して拡大縮小したときに、ベクタコマンドを使用して描画したグラフィックスの見た目がよいということです。

ベクトルベースの描画システム本来の精度を維持するため、描画座標は、整数値ではなく、浮動小数点値を使用して指定します。座標に浮動小数点値を使用することにより、プログラムのコンテンツ位置を非常に正確に指定することができます。ほとんどの場合、これらの値が最終的にデバイスの画面にどう対応づけられているかは心配する必要はありません。

ユーザ座標空間が、すべての描画コマンドで使用する環境となります。ユーザ座標空間は、点を単位として表されます。**デバイス座標空間**とは、デバイスにとってネイティブの座標空間で、ピクセルを単位として表されます。デフォルトでは、ユーザ座標空間の1個の点が、デバイス空間の1ピクセルに対応し、点1個が160分の1インチとなります。ただし、この1対1の比率が常に当てはまるわけではありません。

色および色空間

iPhone OSは、Quartzで利用可能な色空間全体をサポートしていますが、ほとんどのアプリケーションで必要となるのはRGB色空間だけです。iPhone OSは組み込みハードウェアで実行され、画面にグラフィックスを表示するよう設計されているため、RGB色空間は最も使用に適しています。

`UIColor`オブジェクトは、RGB、HSB、グレイスケールの値を使用してカラー値を指定できる簡易メソッドを備えています。この方法で色を作成する場合、色空間を指定する必要はありません。色空間は、`UIColor`オブジェクトが自動的に指定してくれます。

Core Graphicsフレームワークでは、`CGContextSetRGBStrokeColor`関数および`CGContextSetRGBFillColor`関数を使用して色を作成し、設定することもできます。Core Graphicsフレームワークでは、ほかの色空間を使用した色の作成およびカスタム色空間の作成がサポートされていますが、描画コードでこれらの色を使用することは推奨されません。描画コードでは常にRGBカラーを使用してください。

サポートされる画像形式

表 4-1に、iPhone OSが直接サポートする画像形式の一覧を示します。これらの形式のうち、PNG形式がアプリケーションで使用する画像形式として最も推奨されます。

表 4-1　サポートされる画像形式

形式	ファイル名拡張子
Portable Network Graphic (PNG)	`.png`
Tagged Image File Format (TIFF)	`.tiff`, `.tif`
Joint Photographic Experts Group (JPEG)	`.jpeg`, `.jpg`
Graphic Interchange Format (GIF)	`.gif`
Windows Bitmap Format (DIB)	`.bmp`, `.BMPf`
Windowsアイコン形式	`.ico`
Windowsカーソル	`.cur`
XWindowビットマップ	`.xbm`

描画に関するヒント

以降の各セクションでは、エンドユーザから見たアプリケーションの外観を魅力的なものにしながら、質の高い描画コードを作成するためのヒントを紹介します。

カスタム描画コードを使用する場面の判断

作成するアプリケーションのタイプに応じて、カスタム描画コードをほとんど、あるいはまったく使用しないで済ませることができます。没入型のアプリケーションでは通常、カスタム描画コードを惜しみなく使用しますが、ユーティリティ型アプリケーションや生産性型アプリケーションであれば、標準のビューやコントロールを使用してコンテンツを表示できることが少なくありません。

カスタム描画コードは、表示するコンテンツを動的に変化させる必要のある場合に限定するべきです。たとえば、描画アプリケーションであれば、カスタム描画コードを使用してユーザの描画コマンドを追跡する必要があり、ゲームであれば、常に変化するゲームの状況を反映するため絶えず画面を更新することになります。これらの状況では、適切な描画テクノロジーを選択し、カスタムビュークラスを作成してイベントを処理し、表示を適切に更新する必要があります。

一方、アプリケーションのインターフェイスの大半が固定されている場合、あらかじめインターフェイスを1つ以上の画像ファイルにレンダリングしておき、`UIImageView`オブジェクトを使用して、それらの画像を実行時に表示することができます。インターフェイスを構築する際に、必要に応じてほかのコンテンツとImage View群を重ねることができます。たとえば、`UILabel`オブジェクトを使用して、設定変更可能なテキストを表示し、ボタンなどのコントロールを追加して対話機能を提供できます。

描画パフォーマンスの改善

描画は、どのプラットフォームでも比較的負荷の高い処理です。描画コードを最適化することは、開発プロセスにおいて常に重要です。表 4-2では、描画コードをできる限り最適化するためのヒントを紹介します。これらのヒントに加え、コードのテスト、ホットスポットや冗長部分の除去を行うためのパフォーマンスツールを必ず使用してください。

表 4-2　描画パフォーマンス改善のヒント

ヒント	アクション
描画を最小限にする	各更新サイクルの間は、ビューのなかで実際に変更された部分だけを更新します。`UIView`の`drawRect:`メソッドを使用して描画を実行する場合、そのメソッドに渡される更新矩形を使用することで、描画の範囲を限定します。OpenGLによる描画の場合、自ら更新を追跡する必要があります。
不透過なビューは不透過としてマークする	コンテンツが不透過なビューの合成は、部分的に透過的なビューを合成する場合よりもはるかに負担が少なくて済みます。ビューを不透過にするためには、ビューコンテンツに透明効果が含まれないようにし、ビューの`opaque`プロパティを`YES`に設定します。

ヒント	アクション
不透過なPNGファイルからアルファチャネルを削除する	PNG画像の各ピクセルが不透過な場合、アルファチャネルを削除することにより、その画像を含むレイヤのブレンドが不要となります。これにより、画像の合成が大幅に簡素化され、描画パフォーマンスが改善します。
スクロール時に表のセルとビューを再利用する	スクロール時に新たなビューを作成することはいかなる場合でも避けてください。新しいビューを作成するのに時間を費やすと、画面の更新に使用できる時間が少なくなり、スクロール動作が滑らかでなくなります。
スクロール時に、前のコンテンツのクリアを避ける	デフォルトでは、UIKitはビューの現在のコンテキストバッファをクリアしてから`drawRect:`メソッドを呼び出して、その同じ領域を更新します。ビュー内のスクロールイベントに応答する場合、スクロールによる更新の間に繰り返しこの領域をクリアすると、負荷が大きくなる可能性があります。この動作を無効にするには、`clearsContextBeforeDrawing`プロパティの値を`NO`に変更します。
描画中のグラフィックス状態の変更を最小限に抑える	グラフィックス状態の変更は、ウインドウサーバに負担がかかります。類似の状態情報を利用するコンテンツを描画する必要のある場合、そうしたコンテンツをまとめて描画し、必要な状態変更の回数を減らすことを試みてください。

画像品質の維持

ユーザインターフェイスに高品質の画像を提供することは、設計における優先事項です。画像は、複雑なグラフィックスを表示するとても効率的な方法です。効果的な場面では積極的に使用するべきです。アプリケーション用の画像を作成するときは、次のガイドラインを念頭に置くようにしてください。

- **PNG形式の画像を使用する。**PNG形式は、高品質の画像コンテンツを提供する形式で、iPhone OSにおける推奨画像形式です。さらに、iPhone OSには、PNG画像向けに最適化された、ほかの形式より基本的に高効率な描画パスが組み込まれています。
- **リサイズが不要となるよう画像を作成する。**ある特定サイズの画像を使用する予定のある場合は、対応する画像リソースをそのサイズで作成するようにします。拡大縮小では必要なCPUサイクルが増え、補間処理が必要となるため、大きめの画像を作成してから適切なサイズに縮小する方法はとらないようにしてください。可変サイズの画像を表示する必要のある場合は、その画像を異なるサイズで複数作成しておき、対象のサイズに比較的近い画像を縮小するようにします。

QuartzとUIKitを使用した描画

Quartzとは、iPhone OSにおけるネイティブウインドウサーバと描画テクノロジーを表す総称です。Core Graphicsフレームワークは、Quartzの核を構成し、コンテンツの描画に使用される主要インターフェイスです。このフレームワークは、以下を操作するためのデータ型と関数を提供します。

- グラフィックスコンテキスト

- パス
- 画像とビットマップ
- 透明レイヤ
- 色、パターン色、色空間
- グラデーションと陰影
- フォント
- PDFコンテンツ

UIKitは、グラフィックス関連の操作に焦点を合わせた各種クラスを提供し、Quartzの基本機能に基づいて構成されています。UIKitのグラフィックスクラスは、描画ツールの総合セットを意図したものではありません。その役割はCore Graphicsがすでに担っています。代わりに、UIKitグラフィックスクラスは、ほかのUIKitクラスの描画をサポートします。UIKitのサポート機能には、次のクラスと関数が含まれます。

- `UIImage`。画像を表示する不変クラスを実装します。
- `UIColor`。デバイスカラーの基本サポートを提供します。
- `UIFont`。フォント情報を必要とするクラスにフォント情報を提供します。
- `UIScreen`。画面に関する基本情報を提供します。
- `UIImage`オブジェクトをJPEG形式またはPNG形式で生成する関数
- 矩形を描画する関数および描画領域をクリッピングする関数
- 現在のグラフィックスコンテキストを変更、取得する関数

UIKitを構成するクラスとメソッドの詳細については、『*UIKit Framework Reference*』を参照してください。Core Graphicsフレームワークを構成する不透過型と関数に関する詳細については、『*Core Graphics Framework Reference*』を参照してください。

グラフィックスコンテキストの設定

`drawRect:`メソッドが呼び出されるまでに、ビューに組み込まれている描画コードはすでに、デフォルトのグラフィックスコンテキストの作成と設定を完了しています。このグラフィックスコンテキストへのポインタは、`UIGraphicsGetCurrentContext`関数を呼び出して取得できます。この関数は、`CGContextRef`型への参照を返します。この参照を、Core Graphics関数に渡して現在のグラフィックス状態を変更します。表 4-3に、グラフィックス状態の変更に使用する主な関数を示します。すべての関数の一覧については、『*CGContext Reference*』を参照してください。この表は、UIKitに代替できるものがあれば、それも示しています。

表 4-3　グラフィックスの状態を変更するCore Graphics関数

グラフィックスの状態	Core Graphics関数	UIKitの代替

現在の変換行列(CTM)	`CGContextRotateCTM`
	`CGContextScaleCTM`
	`CGContextTranslateCTM`
	`CGContextConcatCTM`
		なし

グラフィックスの状態	Core Graphics関数	UIKitの代替
クリッピング領域	`CGContextClipToRect`	なし

線：幅、結合、キャップ、ダッシュ、マイター上限	`CGContextSetLineWidth`
	`CGContextSetLineJoin`
	`CGContextSetLineCap`
	`CGContextSetLineDash`
	`CGContextSetMiterLimit`
		なし

曲線推定（平坦度）の精度	`CGContextSetFlatness`	なし
アンチエイリアスの設定	`CGContextSetAllowsAntialiasing`	なし
色：塗りつぶしとストロークの設定	`CGContextSetRGBFillColor` `CGContextSetRGBStrokeColor`	`UIColor`クラス
アルファ値（透明度）	`CGContextSetAlpha`	なし
レンダリングインテント	`CGContextSetRenderingIntent`	なし
色空間：塗りつぶしとストロークの設定	`CGContextSetFillColorSpace` `CGContextSetStrokeColorSpace`	なし

テキスト：フォント、フォントサイズ、文字間のスペース、テキスト描画モード	`CGContextSetFont`
	`CGContextSetFontSize`
	`CGContextSetCharacterSpacing`
		`UIFont`クラス

ブレンドモード	`CGContextSetBlendMode`	`UIImage`クラスと各種描画関数を使用して、使用するブレンドモードを指定することができます。

グラフィックスコンテキストには、保存されたグラフィックス状態のスタックがあります。Quartzがグラフィックスコンテキストを作成する時点で、スタックは空です。`CGContextSaveGState`関数を使用すると、現在のグラフィックス状態のコピーがスタックにプッシュされます。以後、グラフィックス状態に加えた変更はその後の描画操作に影響を与えますが、スタックに格納されたコピーには影響しません。変更が完了した時点で、`CGContextRestoreGState`関数を使用して、保存されている状態をスタックの一番上からポップすることにより、前のグラフィックス状態に戻ることができます。このようにグラフィックス状態をプッシュしたり、ポップしたりする方法は、前の状態にすばやく戻る方法であり、状態の変更を個別に取り消す必要をなくします。この方法は、状態の特定の側面（クリッピングパスなど）を元の設定に復元するただ1つの方法でもあります。

グラフィックスコンテキストの概要およびグラフィックスコンテキストを使用した描画環境の設定については、『*Quartz 2D Programming Guide*』の「Graphics Contexts」を参照してください。

画像の作成と描画

iPhone OSは、UIKitとCore Graphicsフレームワークの両方を使用した画像のロードと表示をサポートします。画像の描画に使用するクラスと関数をどのように決めるかは、画像の使用目的によって異なります。ただし、できる限り、コード内で画像を表示するときはUIKitのクラスを使用することをお勧めします。表 4-4に、考えられるいくつかの画像使用のシナリオと、それぞれの場合に推奨される処理方法を示します。

表 4-4　画像使用のシナリオ

シナリオ	推奨される処理方法
画像をビューのコンテンツとして表示	`UIImageView`クラスを使用して画像をロードし、表示します。この方法は、ビューのコンテンツが画像だけの場合に使用できます。画像のビューの上にさらにビューのレイヤを追加して、ほかのコントロールやコンテンツを描画することもできます。
ビューの一部分に、装飾として画像を表示	`UIImage`クラスを使用して画像をロードし、表示します。
ビットマップデータを画像オブジェクトに保存	`UIGraphicsBeginImageContext`関数を使用して、画像ベースの新しいグラフィックスコンテキストを作成します。グラフィックスコンテキストを作成したら、それに画像コンテンツを描画し、`UIGraphicsGetImageFromCurrentImageContext`関数を使用して描画したコンテンツに基づく画像を生成します（必要であれば、追加の画像の描画と生成を続けます）。画像の作成が完了したら、`UIGraphicsEndImageContext`関数を使用してグラフィックスコンテキストを閉じます。 Core Graphicsを使用する場合は、`CGBitmapContextCreate`関数を使用して、ビットマップ用のグラフィックスコンテキストを作成し、それに画像コンテンツを描画します。描画が完了したら、`CGBitmapContextCreateImage`関数を使用し、そのビットマップコンテキストから`CGImageRef`を作成します。Core Graphicsの画像を直接描画したり、この画像を使用して`UIImage`オブジェクトを初期化することができます。
画像をJPEGファイルまたはPNGファイルとして保存	`UIImage`オブジェクトを元の画像データから作成します。`UIImage-JPEGRepresentation`関数または`UIImagePNGRepresentation`関数を呼び出して`NSData`オブジェクトを取得し、取得したオブジェクトのメソッドを使用してデータをファイルに保存します。

次の例は、アプリケーションバンドルから画像をロードする方法を示します。画像をロードした後、この画像オブジェクトを使用して`UIImageView`オブジェクトを初期化できます。また、この画像を保存しておき、ビューの`drawRect:`メソッドの中で明示的に描画できます。

```
NSString* imagePath = [[NSBundle mainBundle] pathForResource:@"myImage"
ofType:@"png"];
UIImage* myImageObj = [[UIImage alloc] initWithContentsOfFile:imagePath];
```

ビューのdrawRect:メソッドで画像を明示的に描画するときは、UIImageで利用可能な任意の描画メソッドを使用できます。これらのメソッドでは、ビューのどこに画像を描画するかを指定できるため、描画の前に別途変換を作成し、適用する必要がありません。anImageというメンバ変数に、事前にロードした画像を保存したと仮定した場合、次の例ではその画像を、ビューの座標の(10, 10)の位置に描画します。

```
- (void)drawRect:(CGRect)rect
{
    // 描画する
    [anImage drawAtPoint:CGPointMake(10, 10)];
}
```

重要： CGContextDrawImage関数を使用してビットマップ画像を直接描画する場合、デフォルトでは、描画する画像データはY軸に沿って反転します。これは、Quartzの画像が、左下隅を原点とし、正の座標軸が原点から上方向と右方向に伸びる座標系を前提としているためです。描画の前に変換を適用することもできますが、それよりも簡単で推奨されるQuartz画像の描画方法は、Quartz画像をUIImageオブジェクトでラップすることです。このオブジェクトでは、座標空間の違いが自動的に補正されます。Core Graphicsを使用した画像の作成と描画の詳細については、『*Quartz 2D Programming Guide*』を参照してください。

パスの作成と描画

パスとは、連続する直線とベジェ曲線を使用して表される2Dのジオメトリシーンを記述したものです。UIKitには、ビュー内に矩形などの単純なパスを描画するためのUIRectFrame関数とUIRectFill関数が含まれています。Core Graphicsにも、矩形や楕円といった単純なパスを作成するための簡易関数が含まれています。より複雑なパスの場合、Core Graphicsフレームワークの関数を使用して自らパスを作成する必要があります。

パスを作成するには、CGContextBeginPath関数を使用して、パスコマンドを受け取るグラフィックスコンテキストを設定します。この関数を呼び出した後、ほかのパス関連関数を使用して、パスの開始点の設定、直線や曲線の描画、矩形と楕円の追加などを実行します。パスのジオメトリを指定し終えたら、パスを直接ペイントしたり、CGPathRefデータ型またはCGMutablePathRefデータ型を作成し、パスへの参照をそこに保存して後で利用したりできます。

ビューにパスを描画する際には、パスのストローク、パスの塗りつぶし、またはストロークと塗りつぶしの両方を実行できます。CGContextStrokePathなどの関数を使用してパスをストロークすると、現在のストローク色を使用して、パスを中心とした直線が作成されます。CGContextFillPath関数によるパスの塗りつぶしでは、現在の塗りつぶし色または塗りつぶしパターンを使用して、パスの線分で囲まれている領域が塗りつぶされます。

複雑なパス要素を構成する点を指定する方法を含め、パスを描画する方法の詳細については、『*Quartz 2D Programming Guide*』の「Paths」を参照してください。パスの作成に使用する関数については、『*CGContext Reference*』および『*CGPath Reference*』を参照してください。

パターン、グラデーション、陰影の作成

Core Graphicsフレームワークには、パターン、グラデーション、陰影を作成する関数も用意されています。これらはモノクロ以外の色の作成や、作成したパスの塗りつぶしに使用します。パターンは画像やコンテンツの繰り返しを元に作成します。グラデーションと陰影はそれぞれ、色から色への滑らかな移り変わりを生み出すことができます。

パターン、グラデーション、陰影の作成と使用の詳細については、『*Quartz 2D Programming Guide*』を参照してください。

OpenGL ESを使用した描画

Open Graphics Library (OpenGL)は、C言語ベースのクロスプラットフォームインターフェイスで、デスクトップシステムにおいて2Dおよび3Dのコンテンツを作成するために使用します。通常、ゲームデベロッパや、高フレームレートで描画する必要のあるユーザに使用されています。OpenGLの関数は、点、直線、多角形などの基本構造を指定したり、テクスチャや特殊効果をこれらの構造に適用して外見に変化を付けるために使用します。呼び出したOpenGL関数は、レンダリングを行う、基盤となるハードウェアにグラフィックスコマンドを送信します。レンダリングはほとんどがハードウェアで実行されるため、通常、OpenGLの描画は非常に高速です。

OpenGL for Embedded SystemsはOpenGLの機能を一部省略したバージョンで、モバイルデバイス向けに、最新のグラフィックスハードウェアを生かすよう設計されています。iPhone OSベースのデバイス、つまりiPhoneやiPod Touch向けにOpenGLコンテンツを作成する場合は、OpenGL ESを使用します。iPhone OSで提供されるOpenGL ESフレームワーク(`OpenGLES.framework`)は、OpenGL ES v1.1およびOpenGL ES v2.0のどちらの仕様もサポートしています。

iPhone OSでのOpenGL ESサポートの詳細については、『*OpenGL ES Programming Guide for iPhone*』を参照してください。

Core Animation効果の適用

Core AnimationはObjective-Cフレームワークで、なめらかなリアルタイムアニメーションをすばやく簡単に作成する基盤を提供します。Core Animation自体は、形状や画像、その他のタイプのコンテンツを作成する基本ルーチンを提供するわけではないという点で、描画テクノロジーではありません。その代わり、別のテクノロジーを使用して作成したコンテンツを操作したり、表示したりするためのテクノロジーであるといえます。

ほとんどのアプリケーションは、iPhone OSにおいてなんらかの形でCore Animationを利用するメリットがあります。アニメーションは、今起きていることをユーザに伝えるフィードバックの働きがあります。たとえば、ユーザが「設定(Settings)」アプリケーションを操作するとき、ユーザが環境設定の下位階層に進んだり、ルート階層に戻ったりするのに合わせて、画面が視野にすべり込んだり、視野からすべり出たりします。この種のフィードバックは重要で、ユーザに対して文脈情報を提供します。アニメーションは、アプリケーションの見た目を充実させる働きもあります。

ほとんどの場合、ごくわずかな労力でCore Animationのメリットを享受することができます。たとえば、`UIView`クラスのいくつかのプロパティ（ビューのフレーム、中心、色、不透明度など）について、値が変化するとアニメーションが開始するよう設定することができます。UIKitにこれらのアニメーションを実行するように伝えるために、一定の作業が必要となりますが、アニメーション自体は自動的に作成され、実行されます。組み込みのビューアニメーションをトリガする方法については、「ビューのアニメーション化」（76 ページ）を参照してください。

基本アニメーションからさらに充実させるには、Core Animationのクラスやメソッドを直接扱う必要があります。以降の各セクションでは、Core Animationに関する情報を紹介し、iPhone OSにおいて、Core Animaitonのクラスやメソッドを使用して典型的なアニメーションを作成する方法について説明します。Core Animationの詳細と使用方法については、『*Core Animation Programming Guide*』を参照してください。

レイヤについて

Core Animationにおける重要なテクノロジーがレイヤオブジェクトです。レイヤは、ビューに性質がよく似た軽量オブジェクトですが、実際には、表示するコンテンツのジオメトリやタイミング、視覚的プロパティをカプセル化する、モデルオブジェクトです。コンテンツは、次に示す3つのうち、いずれかの方法で提供されます。

- `CGImageRef`をレイヤオブジェクトの`contents`プロパティに割り当てることができます。
- デリゲートをレイヤに割り当てて、デリゲートに描画を処理させることができます。
- `CALayer`をサブクラス化し、表示メソッドの1つをオーバーライドすることができます。

レイヤオブジェクトのプロパティを操作すると、実際に操作されるのはモデルレベルのデータで、このデータによって、関連付けられているコンテンツの表示方法が決まります。そのコンテンツの実際のレンダリングはコードとは別に処理され、高速にレンダリングされるよう徹底して最適化されます。デベロッパは、レイヤコンテンツとアニメーションプロパティを設定するだけでよく、あとはCore Animationが引き継いで処理してくれます。

レイヤに関する情報と使用方法については、『*Core Animation Programming Guide*』を参照してください。

アニメーションについて

レイヤのアニメーション化では、Core Animationは独立したアニメーションオブジェクトを使用してアニメーションのタイミングと動作を制御します。`CAAnimation`クラスとそのサブクラスにより、コード内で使用可能なさまざまな種類のアニメーション動作が提供されます。ある値から別の値へとプロパティを移行させるシンプルなアニメーションを作成することも、指定した値とタイミング関数を通じてアニメーションを追跡する、複雑なキーフレームアニメーションを作成することもできます。

Core Animationでは、複数のアニメーションをトランザクションと呼ばれる単位にまとめることもできます。`CATransaction`オブジェクトは、アニメーションのグループを1つの単位として扱います。このクラスのメソッドを使用して、アニメーションの再生時間を設定することもできます。

カスタムアニメーションの作成例については、『*Animation Types and Timing Programming Guide*』を参照してください。

テキストとWeb

iPhone OSのテキストシステムは、モバイルユーザのささやかなニーズを念頭において設計されました。このテキストシステムは、電子メールやSMSのプログラムでよく使われる単一行および複数行のテキスト入力を処理するために設計されました。また、このテキストシステムはUnicodeをサポートし、複数の異なる入力方式を備えています。このため、様々な言語でテキストを表示したり読んだりしやすくなっています。

テキストとWebのサポート

iPhone OSのテキストシステムは、非常に簡単に使えるにも関わらず、絶大な威力を発揮します。UIKitフレームワークには、テキストの表示と入力を管理するための高度なクラスがいくつか含まれています。また、このフレームワークには、HTMLやJavaScriptベースのコンテンツを表示するためのより高度なクラスも含まれています。

以降の各セクションでは、iPhone OSでのテキストとWebの基本サポートについて説明します。このセクションで取り上げた各クラスの詳細については、『*UIKit Framework Reference*』を参照してください。

Text View

UIKitフレームワークには、テキストコンテンツを表示するために、次の3つの主要なクラスがあります。

- `UILabel`：静的なテキスト文字列を表示する
- `UITextField`：編集可能な単一行のテキストを表示する
- `UITextView`：編集可能な複数行のテキストを表示する

これらのクラスは、任意の大きさのテキストの表示をサポートします。ただし、Label FieldやText Fieldは、通常、比較的小さなテキストを表示するために使われます。ただし、iPhone OSベースのデバイスの小さな画面で表示されたテキストを読みやすくするために、これらのクラスでは、Mac OS Xのようなデスクトップオペレーティングシステムに見られるような高度な書式化はサポートしていません。この3つのクラスすべてにおいて、フォント情報（サイズ、スタイルオプションなど）を指定できます。ただし、指定したフォント情報は、そのオブジェクトに関連付けられているテキストすべてに適用されます。

図5-1は、利用可能なテキストクラスの例を画面に表示した様子を示しています。この例は、*UICatalog*サンプルアプリケーションから抜粋したものです。このサンプルアプリケーションは、UIKitで利用可能なさまざまなビューやコントロールの使い方を示しています。左の画像は、さまざまなスタイルのText Fieldを示しています。右の画像は、1つのText Viewを示しています。グレーの背景に表示さ

れたコールアウトは、異なるビューを表示するために使われるテーブルセル内に埋め込まれたUILabelオブジェクトそのものです。また、左の画面の下端には、「Left View」というテキストを含むUILabelオブジェクトがあります。

図 5-1 UICatalogアプリケーションのテキスト関連クラス

編集可能なText Viewで作業をする場合は、常に、編集セッションを管理するデリゲートオブジェクトを提供する必要があります。デリゲートに、編集の開始と終了を知らせたり、編集動作をオーバーライドする機会を与えるために、Text Viewはさまざま通知をデリゲートに送信します。たとえば、デリゲートは、現在のテキストに有効な値が含まれているかを判別し、有効な値でない場合は編集セッションが終了するのを防止できます。最終的に編集が終わると、デリゲートは結果のテキスト値を取得し、アプリケーションのデータモデルを更新します。

Text Viewは、それぞれ意図した使い方が少し異なるので、各Text Viewのデリゲートメソッドも少し異なります。UITextFieldクラスをサポートするデリゲートは、UITextFieldDelegateプロトコルのメソッドを実装しています。同様に、UITextViewクラスをサポートするデリゲートは、UITextViewDelegateプロトコルのメソッドを実装しています。どちらの場合も、プロトコルメソッ

ドを実装する必要はありませんが、プロトコルメソッドを実装しないと、Text Fieldがあまり役に立たない場合もあります。これら2つのプロトコルのメソッドの詳細については、『*UITextFieldDelegate Protocol Reference*』および『*UITextViewDelegate Protocol Reference*』を参照してください。

Web View

`UIWebView`クラスを利用すると、事実上、小型のWebブラウザをアプリケーションのユーザインターフェイスに組み込むことができます。`UIWebView`クラスは、HTML、CSS、およびJavaScriptコンテンツを完全にサポートする、iPhone OSの「Safari」を実装するために使われているのと同じWebテクノロジーを完全に利用しています。また、このクラスは、「Safari」でユーザに親しまれている、たくさんの組み込みジェスチャーもサポートしています。たとえば、ページをダブルクリックまたはピンチすると拡大や縮小ができます。また、指をドラッグするとページをスクロールできます。

コンテンツを表示することのほかに、Web Viewオブジェクトを使用すると、Webフォームを使用してユーザからの入力を収集することもできます。UIKitのその他のテキストクラスと同様に、Webページ内のフォームに編集可能なText Fieldがある場合は、そのフィールドをタップすると、テキスト入力用のキーボードが表示されます。これは、Web体験には不可欠な部分なので、キーボードの表示と非表示はWeb Viewが自ら管理します。

図5-2は、UiKitで利用可能なさまざまなビューとコントロールの使い方を示す、*UICatalog*サンプルアプリケーションから抜粋した`UIWebView`オブジェクトの例を示しています。HTMLコンテンツを表示するだけなので、Webブラウザのようにユーザがページを移動できるようにするには、そのためのコントロールを追加する必要があります。たとえば、この図のWeb Viewは、ターゲットURLを含むText Fieldの下の領域を占有しており、このText Fieldそのものは含んでいません。

図 5-2 Web View

Web Viewは、それに関連付けられたデリゲートオブジェクトを通して、いつページがロードされたかと、ロードエラーがあったかどうかについての情報を提供します。Webデリゲートは、1つ以上の`UIWebViewDelegate`プロトコルメソッドを実装したオブジェクトです。デリゲートメソッドを実装することによって、エラーに応答したり、Webページのロードに関連するその他のタスクを実行したりできます。`UIWebViewDelegate`プロトコルのメソッドの詳細については、『*UIWebViewDelegate Protocol Reference*』を参照してください。

キーボードと入力方法

テキスト入力を受け付けることができるオブジェクト内でユーザがタップすると、そのオブジェクトは、適切なキーボードを表示するようシステムに依頼します。プログラムのニーズとユーザが希望する言語に応じて、システムは、複数の異なるキーボードの中の1つを表示します。アプリケーションでは、ユーザが希望する言語（およびキーボードの入力方法）を制御することはできませんが、ユーザが意図した使い方を示すキーボードの属性（特殊キーの設定とその動作など）を制御することはできます。

アプリケーションのテキストオブジェクトを利用して、キーボードの属性を直接設定します。UITextFieldクラスとUITextViewクラスは共に、キーボード設定用のプロパティを定義するUITextInputTraitsプロトコルに従います。プログラムやInterface BuilderのInspectorウインドウでこれらのプロパティを設定すると、システムは、指定されたタイプのキーボードを表示します。

注： UIWebViewクラスは、UITextInputTraitsプロトコルを直接的にはサポートしませんが、テキスト入力要素用にいくつかのキーボード属性を設定できます。特に、次の例のように入力要素の定義にautocorrect属性とautocapitalization属性を含めると、キーボードの動作を指定できます。

```
<input type="text" size="30" autocorrect="off" autocapitalization="on">
```

入力要素にキーボードタイプを指定することはできません。Web Viewは、デフォルトのキーボードを基に、要素間のナビゲーション用のコントロールを追加したカスタムキーボードを表示します。

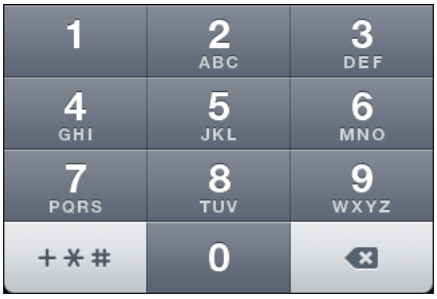
デフォルトのキーボードは、一般的なテキスト入力用に設計されています。図 5-3は、異なるキーボード設定を持つデフォルトキーボードを示しています。デフォルトキーボードには、最初アルファベットキーボードが表示されますが、ユーザはそれを切り替えて、数字と記号を表示することもできます。その他のキーボードのほとんどは、デフォルトキーボードと同様の機能を提供しますが、特定のタスクに特化した追加ボタンを提供します。ただし、電話と数字のキーボードは、数字入力がしやすいように大幅に異なるレイアウトになっています。

図 5-3 さまざまなキーボードタイプ

Default

Email

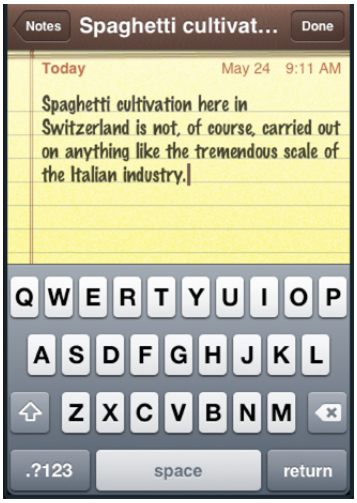
URL



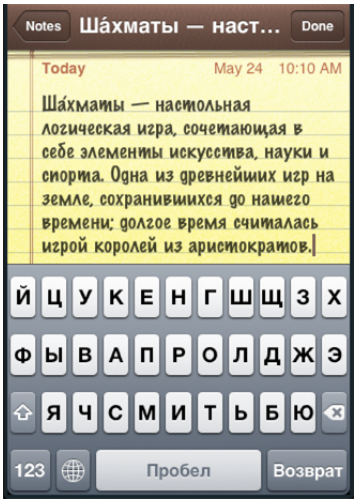
Phone

さまざまなユーザの言語環境設定を容易にするために、iPhone OSは、さまざまな入力方式とさまざまな言語用のキーボードレイアウトをサポートしています。そのいくつかを図 5-4に示します。入力方式とキーボードレイアウトは、ユーザの言語環境設定によって決まります。

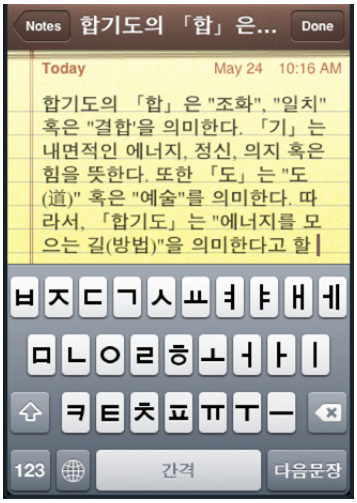
図 5-4 さまざまなキーボードと入力方式



English



Russian



Korean

Japanese - Romanji

Japanese - Kana

Chinese - Handwriting

キーボードの管理

多くのUIKitオブジェクトは、ユーザとの対話に応答して自動的にキーボードを表示しますが、キーボードを設定したり管理する責任の一部はアプリケーションにあります。以降の各セクションでは、これらの責務について説明します。

キーボード通知を受信する

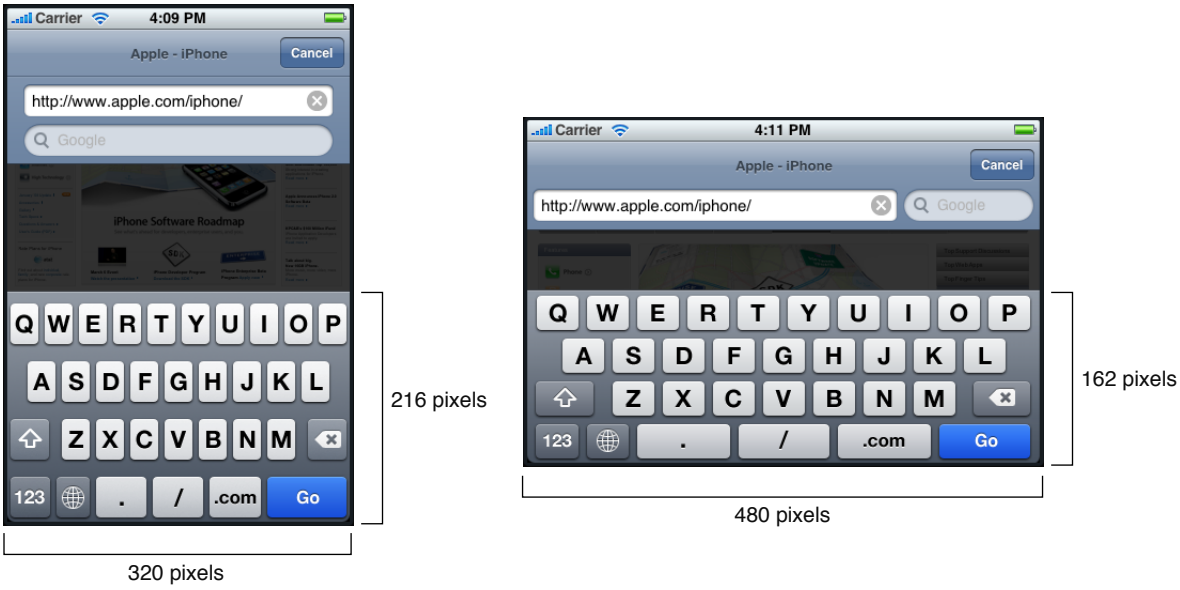
キーボードが表示または非表示になると、iPhone OSは、登録されているオブザーバに以下の通知を送信します。

- UIKeyboardWillShowNotification
- UIKeyboardDidShowNotification
- UIKeyboardWillHideNotification
- UIKeyboardDidHideNotification

キーボードが最初に表示されたとき、キーボードが消えたとき、キーボードの所有者が変わったとき、またはアプリケーションの向きが変わったときに、システムはキーボード通知を送信します。それぞれの状況で、システムは、これらのメッセージの適切なサブセットのみを送信します。たとえば、キーボードの所有者が変わった場合、システムは現在の所有者に`UIKeyboardWillHideNotification`メッセージを送信しますが、`UIKeyboardDidHideNotification`メッセージは送信しません。それは、この変更によってキーボードが非表示にならないからです。`UIKeyboardWillHideNotification`の送付は、キーボードフォーカスを失うおうとしている現在の所有者に警告を与える単純な方法です。一方、キーボードの向きが変わると、willHideとdidHideの両方の通知が送信されます。これは、それぞれの向きでキーボードが異なるので、新しいキーボードを表示する前に元のキーボードを消去する必要があるからです。

各キーボード通知には、画面上のキーボードのサイズと位置についての情報が含まれています。キーボードが特定のサイズ、または特定の場所にあると仮定するのではなく、これらの通知の情報を常に使用すべきです。キーボードのサイズは、どの入力方式でも同じであるとは限りません。また、iPhone OSのリリースの違いによって異なる場合もあります。さらに、同じ言語で同じシステムリリースの場合ですら、アプリケーションの向きによってキーボードの寸法が変わる可能性があります。たとえば、図5-5は、縦長モードと横長モードの両方のURLキーボードのサイズの比較を示しています。キーボード通知内の情報を使用することによって、正しいサイズと位置の情報を確実に取得できます。

図 5-5　縦長モードと横長モードのキーボードサイズの比較



注： 情報ディクショナリの`UIKeyboardBoundsUserInfoKey`に含まれる矩形は、サイズ情報のためだけに使用すべきです。矩形の原点（常に{0.0,0.0}）は、矩形どおしを交差させる操作には使用しないでください。キーボードはアニメーションによって配置されるので、キーボードの実際の境界矩形は時間とともに変化します。したがって、キーボードの開始位置と終了位置は、情報ディクショナリの`UIKeyboardCenterBeginUserInfoKey`キーと`UIKeyboardCenterEndUserInfoKey`キーの下に格納され、これらの値から原点を計算できます。

キーボード通知を使用する理由の1つは、キーボードが表示されたときに、キーボードによって隠されるコンテンツを再配置できるからです。このシナリオの処理方法については、「キーボードの下に隠れているコンテンツを移動する」（127 ページ）を参照してください。

キーボードを表示する

ユーザがビューをタップすると、システムは自動的にそのビューをファーストレスポンダとして指名します。編集可能なテキストを含むビューに対してこのイベントが発生すると、そのビューは対象テキストの編集セッションを起動します。キーボードがまだ表示されていない場合は、編集セッションの最初に、ビューはキーボードの表示をシステムに依頼します。キーボードがすでに表示されている場合は、ファーストレスポンダ内の変更によって、キーボードからのテキスト入力は新たにタップされたビューにリダイレクトされます。

ビューがファーストレスポンダになるとキーボードは自動的に表示されるので、通常は、キーボードを表示するために何らかの処理を実行する必要はありません。ただし、編集可能なテキストビューの`becomeFirstResponder`メソッドを呼び出すことによって、プログラムでキーボードを表示することもできます。このメソッドを呼び出すと、ターゲットビューがファーストレスポンダになり、ユーザがビューをタップしたときとまったく同じように、編集プロセスが始まります。

アプリケーションが1つの画面上で複数のテキストベースビューを管理する場合は、後でキーボードをしまうことができるように、現在どのビューがファーストレスポンダになっているかを追跡するのは、よい考えです。

キーボードをしまう

システムは、通常、キーボードを自動的に表示しますが、キーボードを自動的にはしまいません。その代わり、適切なときにキーボードをしまうのはアプリケーションの責任です。通常は、ユーザのアクションに応答してキーボードをしまいます。たとえば、ユーザがキーボード上の「Return」ボタンまたは「Done」ボタンをタップしたときや、アプリケーションのインターフェイス内のその他のボタンをタップしたときに、キーボードをしまいます。キーボードの設定によっては、キーボードをしまいやすくするために、ユーザインターフェイスにその他のコントロールを追加する必要があるかもしれません。

キーボードをしまうには、現在ファーストレスポンダになっているテキストベースビューの`resignFirstResponder`メソッドを呼び出します。Text Viewがファーストレスポンダの状態でなくなると、現在の編集セッションは終了し、そのことがデリゲートに通知されます。そして、キーボードが消えます。つまり、`myTextField`という変数が、現在ファーストレスポンダになっている`UITextField`オブジェクトを指す場合、キーボードをしまうには以下のコードを実行するだけです。

```
[myTextField resignFirstResponder];
```

それ以降に発生したすべてのことは、このテキストオブジェクトによって自動的に処理されます。

キーボードの下に隠れているコンテンツを移動する

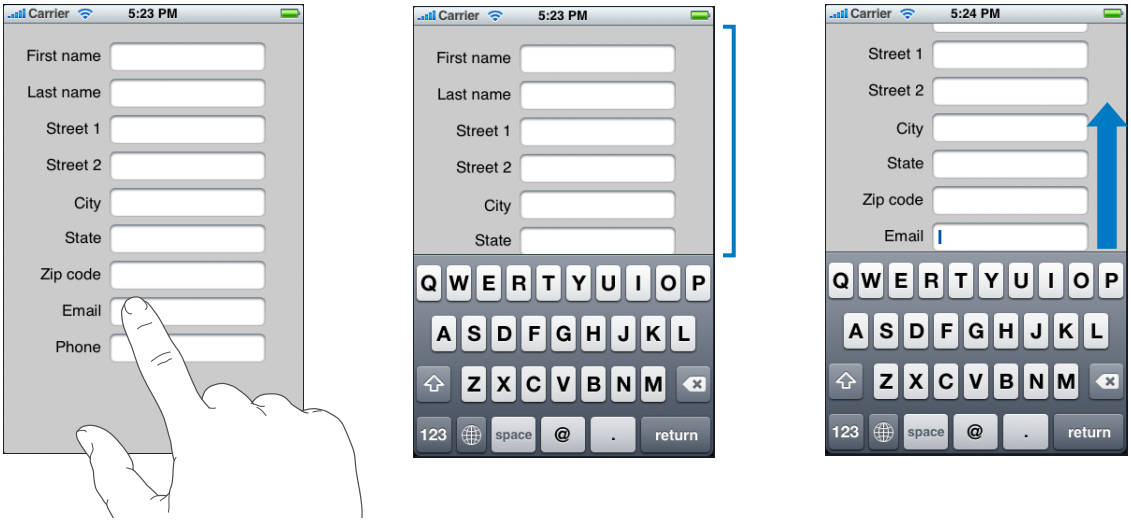
システムは、キーボードの表示を要求されると、画面の下端からキーボードをスライドさせてアプリケーションのコンテンツの上に配置します。キーボードは、コンテンツの上に配置されるため、キーボードが、編集対象のテキストオブジェクトの上に置かれる可能性があります。このような場合は、編集対象のオブジェクトが見えるようにコンテンツを調整しなければなりません。

通常、コンテンツを調整するには、1つ以上のビューを一時的にサイズ変更したり、テキストオブジェクトが見えるようにビューを配置する必要があります。テキストオブジェクトをキーボードと一緒に管理する最も簡単な方法は、テキストオブジェクトを`UIScrollView`オブジェクト（または、そのサブクラス（`UITableView`など））内に埋め込むことです。キーボードが表示されたときに実行しなければならないことは、Scroll Viewをサイズ変更して、対象のテキストオブジェクトを移動させることです。したがって、`UIKeyboardDidShowNotification`に応答して、処理メソッドでは以下の処理を実行します。

1. キーボードのサイズを取得する。
2. Scroll Viewの高さからキーボードの高さを差し引く。
3. 編集対象のText Fieldが表示されるようにスクロールする。

図 5-6は、`UIScrollView`オブジェクト内に複数のText Fieldが埋め込まれた単純なアプリケーションで、上の手順を実行した様子を示しています。キーボードが表示されると、通知処理メソッドがScroll Viewのサイズを変更します。次に、`UIScrollView`の`scrollRectToVisible:animated:`メソッドを使用して、タップされたText Field（この場合は、「Email」フィールド）が見えるようにスクロールします。

図 5-6 キーボードに合わせてコンテンツを調整する



1. User taps email field
2. Scroll view resized to new height
3. Email field scrolled into view

注： 独自のScroll Viewをセットアップするときは、必ず、コンテンツビューに対する自動サイズ変更規則を適切に設定してください。前の図では、Text Fieldが実際には汎用的なUIViewオブジェクトのサブビューであり、このオブジェクト自体がUIScrollViewオブジェクトのサブビューです。汎用的なビューのUIViewAutoresizingFlexibleWidthとUIViewAutoresizingFlexibleHeightの各自動サイズ変更オプションが設定されている場合、Scroll Viewのフレームサイズが変化すると、汎用的なビューのフレームも変化します。これによって、望まない結果が生じることがあります。これらのオプションを無効にすることによって、ビューのサイズが維持されて、コンテンツが正しくスクロールされることが保証されます。

リスト 5-1は、キーボード通知を受信できるように登録するためのコードと、これらの通知用のハンドラメソッドを示しています。このコードは、Scroll Viewを管理するView Controllerによって実装されます。また、scrollView変数は、Scroll Viewオブジェクトを指す接続口です。各ハンドラメソッドは、キーボード通知の情報ディクショナリからキーボードサイズを取得し、対応する分だけScroll Viewの高さを調節します。さらに、keyboardWasShown:メソッドは、アクティブなText Fieldの矩形をスクロールします。このフィールドは、View Controllerのメンバ変数であるカスタム変数（この例ではactiveFieldと呼ぶ）に格納されており、textFieldDidBeginEditing:デリゲートメソッドに設定されます（リスト 5-2 （129 ページ）に例を示します）（この例では、View Controllerは、それぞれのText Fieldのデリゲートとして役割も果たしています）。

リスト 5-1 キーボード通知の処理

```
// View Controllerセットアップコードのどこかでこのメソッドを呼び出す
- (void)registerForKeyboardNotifications
{
    [[NSNotificationCenter defaultCenter] addObserver:self
            selector:@selector(keyboardWasShown:)
            name:UIKeyboardDidShowNotification object:nil];

    [[NSNotificationCenter defaultCenter] addObserver:self
            selector:@selector(keyboardWasHidden:)
            name:UIKeyboardDidHideNotification object:nil];
}

// UIKeyboardDidShowNotificationが送信されたときに呼び出される
- (void)keyboardWasShown:(NSNotification*)aNotification
{
    if (keyboardShown)
        return;

    NSDictionary* info = [aNotification userInfo];

    // キーボードのサイズを取得する
    NSValue* aValue = [info objectForKey:UIKeyboardBoundsUserInfoKey];
    CGSize keyboardSize = [aValue CGRectValue].size;

    // Scroll View（このウインドウのルートビュー）をサイズ変更する
    CGRect viewFrame = [scrollView frame];
    viewFrame.size.height -= keyboardSize.height;
    scrollView.frame = viewFrame;

    // アクティブなText Fieldが表示されるようにスクロールする
    CGRect textFieldRect = [activeField frame];
    [scrollView scrollRectToVisible:textFieldRect animated:YES];

    keyboardShown = YES;
```

```
}


// UIKeyboardDidHideNotificationが送信されたときに呼び出される
- (void)keyboardWasHidden:(NSNotification*)aNotification
{
    NSDictionary* info = [aNotification userInfo];

    // キーボードのサイズを取得する
    NSValue* aValue = [info objectForKey:UIKeyboardBoundsUserInfoKey];
    CGSize keyboardSize = [aValue CGRectValue].size;

    // Scroll Viewの高さを元の値に戻す
    CGRect viewFrame = [scrollView frame];
    viewFrame.size.height += keyboardSize.height;
    scrollView.frame = viewFrame;

    keyboardShown = NO;
}
```

前のコードリストのkeyboardShown変数は、キーボードがすでに表示されているかどうかを追跡するために使用するブール値です。インターフェイスに複数のText Fieldが含まれている場合、ユーザは、各フィールド内の値を編集するために、これらのフィールドの間をタップすることができます。このような場合は、キーボードは消えませんが、システムは、新しいText Fieldで編集が始まるたびにUIKeyboardDidShowNotification通知を生成します。キーボードが実際に消えたかどうかを追跡することによって、このコードでは、Scroll Viewのサイズが2回以上小さくなることを防止しています。

リスト 5-2に、前述の例のactiveField変数のセットとクリアを行うために、View Controllerによって使用される追加のコードを示します。初期化時に、インターフェイス内の各Text Fieldは、View Controllerをデリゲートとして設定します。そのため、Text Fieldがアクティブになるとこれらのメソッドが呼び出されます。Text Fieldおよびそのデリゲート通知の詳細については、*UITextField Class Reference*を参照してください。

リスト 5-2　アクティブなText Fieldを追跡する追加のメソッド

```
- (void)textFieldDidBeginEditing:(UITextField *)textField
{
    activeField = textField;
}

- (void)textFieldDidEndEditing:(UITextField *)textField
{
    activeField = nil;
}
```

テキストの描画

テキストを表示したり編集したりするUIKitクラスのほかに、iPhone OSには、画面に直接テキストを描画するための方法がいくつかあります。単純な文字列を描画するための最も簡単で効率的な方法は、NSStringクラスのUIKit拡張を使用することです。これらの拡張には、さまざまな属性を使用し

て画面上の望みの場所に文字列を描画するメソッドが含まれています。また、実際に文字列を描画する前に、レンダリングされた文字列のサイズを計算するメソッドもあります。これは、アプリケーションのコンテンツを正確にレイアウトするために役立ちます。

重要： パフォーマンスに影響するので、テキストを直接描画するのはできるだけ避けてください。静的なテキストは、1つ以上の`UILabel`オブジェクトを使用すると、独自の描画ルーチンを使用するよりもはるかに効率的に描画できます。同様に、`UITextField`クラスには、編集可能なテキスト領域をコンテンツに組み込みやすくする、さまざまなスタイルが含まれています。

インターフェイスにカスタムテキスト文字列を描画する必要のある場合は、`NSString`のメソッドを使用します。UIKitには、基本の`NSString`クラスの拡張機能が含まれており、それを利用してビューに文字列を描画できます。それらのメソッドを使用することで、レンダリングされるテキストの位置を細かく調節したり、ビューコンテンツの残りの部分と合成したりできます。このクラスのメソッドを利用して、望みのフォントとスタイル属性に基づいて、テキストの境界矩形をあらかじめ計算することもできます。詳細については、『*NSString UIKit Additions Reference*』を参照してください。

描画中に使用するフォントをきめ細かく制御する必要がある場合は、Core Graphicsフレームワークの関数を使用して描画します。Core Graphicsフレームワークは、図やテキストの正確な描画と配置のためのメソッドを提供します。これらの関数とその使用法の詳細については、『*Quartz 2D Programming Guide*』および『*Core Graphics Framework Reference*』を参照してください。

Web Viewへのコンテンツの表示

ユーザインターフェイスに`UIWebView`オブジェクトが含まれる場合、ローカルコンテンツまたはネットワークからロードされたコンテンツを表示できます。ローカルコンテンツをロードする際には、コンテンツを動的に作成したり、ファイルからロードして`loadData:MIMEType:textEncodingName:baseURL:`や`loadHTMLString:baseURL:`メソッドを使用してそれを表示したりできます。ネットワークからコンテンツをロードするには、`NSURLRequest`オブジェクトを作成し、それをWeb Viewの`loadRequest:`メソッドに渡します。

ネットワークベースのロードリクエストを開始した後になんらかの理由でWeb Viewを解放しなければならない場合、Web Viewを解放する前に保留中のリクエストをキャンセルする必要があります。ロードリクエストはWeb Viewの`stopLoading`メソッドを使用してキャンセルできます。このコードを挿入する標準的な場所は、所有しているView Controllerの`viewWillDisappear:`メソッドの中です。リクエストがまだ保留中かを確認するには、Web Viewの`loading`プロパティ内の値をチェックします。

ファイルとネットワーク

iPhone OSで実行中のアプリケーションは、ローカルのファイルシステムとネットワークにアクセスできます。それには、Core OSフレームワークとCore Servicesフレームワークの両方を使用します。ローカルのファイルシステム上のファイルを読み書きできることによって、ユーザデータとアプリケーションの状態を次回使用するときのために保存できます。ネットワークアクセスを利用すると、リモート操作の実行とデータの送受信のためにネットワークサーバと通信できます。

ファイルとデータの管理

iPhone OSのファイルは、フラッシュメモリの領域を、ユーザのメディアおよび個人ファイルと共有しています。セキュリティのために、アプリケーションは、そのアプリケーション専用のディレクトリに配置され、そのディレクトリ内のファイルしか読み書きできないように制限されます。以降の各セクションでは、アプリケーションのローカルファイルシステムの構造と、ファイルを読み書きするためのいくつかの手法について説明します。

よく使われるディレクトリ

セキュリティ目的のため、アプリケーションにはデータと環境設定を書き込むことのできる場所は数か所だけです。アプリケーションがデバイス上にインストールされると、アプリケーション用のホームディレクトリが作成されます。表6-1は、アプリケーションからアクセスする可能性のある、ホームディレクトリ内の重要なサブディレクトリのリストです。この表には、各ディレクトリの用途とアクセス制限、およびディレクトリの内容がiTunesによってバックアップされるかどうかが記述されています。バックアッププロセスと復元プロセスの詳細については、「バックアップと復元」（133 ページ）を参照してください。アプリケーションのホームディレクトリの詳細については、「アプリケーションサンドボックス」（24 ページ）を参照してください。

表 6-1 iPhoneアプリケーションのディレクトリ

ディレクトリ	説明
<*Application_Home*>/*AppName*.app	これはアプリケーション自体を含む、バンドルのディレクトリです。アプリケーションは署名されている必要があるため、実行時にこのディレクトリの内容を変更してはなりません。変更を加えると、後でアプリケーションを起動できなくなります。 iPhone OS 2.1以降では、このディレクトリの内容がiTunesによってバックアップされません。ただし、App Storeから購入したアプリケーションの最初の同期はiTunesによって実行されます。

ディレクトリ	説明
<Application_Home>/Documents/	アプリケーション固有のすべてのデータファイルを書き込むために使用するディレクトリです。ユーザデータや、定期的にバックアップする必要があるその他のデータを保存するために、このディレクトリを使用します。このディレクトリのパスの取得方法の詳細については、「アプリケーションディレクトリへのパスの取得」（134 ページ）を参照してください。 このディレクトリの内容はiTunesによってバックアップされます。
<Application_Home>/Library/Preferences	このディレクトリには、アプリケーション固有の環境設定ファイルを保存します。環境設定ファイルを直接作成しないでください。その代わりに、NSUserDefaultsクラスまたは*CFPreferences*APIを使用して、アプリケーションの環境設定を取得したり設定したりします（「Settingsバンドルの追加」（197 ページ）を参照）。 このディレクトリの内容はiTunesによってバックアップされます。
<Application_Home>/Library/Caches	アプリケーションが終了しても次の起動まで保持しておきたいアプリケーション固有のサポートファイルを書き込むために、このディレクトリを使用します。一般に、これらのファイルの追加と削除はアプリケーション側で処理します。ただし、iTunesは、デバイスの完全復元中にこれらのファイルを削除するので、必要に応じて再作成する必要があります。このディレクトリにアクセスするには、「アプリケーションディレクトリへのパスの取得」（134 ページ）で説明するインターフェイスを使用して、ディレクトリへのパスを取得します。 iPhone OS 2.2以降では、このディレクトリの内容がiTunesによってバックアップされません。
<Application_Home>/tmp/	アプリケーションが終了しても次の起動まで保持しておきたい一時ファイルを書き込むために、このディレクトリを使用します。アプリケーションは、これらのファイルが不要になったと判断したら、このディレクトリから削除する必要があります（アプリケーションが実行されていないときに、システムが古いファイルをこのディレクトリから削除する場合もあります）。このディレクトリのパスの取得方法の詳細については、「アプリケーションディレクトリへのパスの取得」（134 ページ）を参照してください。 iPhone OS 2.1以降では、このディレクトリの内容がiTunesによってバックアップされません。

バックアップと復元

アプリケーションで、バックアップ操作や復元操作に備える必要はありません。iPhone OS 2.2以降では、デバイスがコンピュータに接続されて同期されると、iTunesがすべてのファイルの段階的なバックアップを実行します。ただし、以下のディレクトリ内のファイルは除きます。

- *<Application_Home>*/*AppName*.app
- *<Application_Home>*/Library/Caches
- *<Application_Home>*/tmp

iTunesは、アプリケーションバンドルをバックアップしますが、同期操作のたびにそれが実行されるわけではありません。App Storeから購入したデバイス上のアプリケーションは、そのデバイスが次回iTunesと同期したときにバックアップされます。アプリケーションバンドルに変更がない場合は（たとえば、アプリケーションが更新されていない場合）、次回の同期操作中にアプリケーションはバックアップされません。

同期処理に長期間かかるのを防ぐには、アプリケーションのホームディレクトリ内のファイルの置き場所を選択しておく必要があります。*<Application_Home>*/Documentsディレクトリは、ユーザデータファイルや、アプリケーションで簡単に再作成できるファイルを保存するために使用します。一時的なデータを保存するために使用するファイルは、*Application Home*/tmpディレクトリ内に置きます。そして、不要になったらアプリケーション側が削除します。次回起動するときに使用するデータファイルを作成する場合は、それらのファイルを*Application Home*/Library/Cachesに置きます。

注： 大きなデータファイルや、頻繁に変更されるファイルを作成する場合は、それらを*<Application_Home>*/Documentsディレクトリではなく、*Application Home*/Library/Cachesディレクトリに保存することを検討してください。大きなデータファイルをバックアップすると、バックアップ処理が大幅に遅くなる可能性があります。定期的に変更される（したがって、頻繁にバックアップしなければならない）ファイルの場合も同様です。これらのファイルをCachesディレクトリに置くと、同期操作のたびにバックアップされるのを防止できます（iPhone OS 2.2以降の場合）。

アプリケーションでのディレクトリの使い方に関するその他のガイダンスについては、表 6-1 （131 ページ）を参照してください。

アプリケーションの更新時に保存されるファイル

アプリケーションの更新によって、以前のアプリケーションはユーザがダウンロードした新しいアプリケーションと置き換えられます。このプロセス中、iTunesは新しいアプリケーションディレクトリに更新されたアプリケーションをインストールします。次に、古いインストールを削除する前に、古いインストールから新しいアプリケーションディレクトリにユーザのデータファイルを移します。次のディレクトリ内のファイルは、更新プロセスの間に保存されることが保証されています。

- *<Application_Home>*/Documents
- *<Application_Home>*/Library/Preferences

ほかのユーザディレクトリ内のファイルも同じように移動されますが、更新後にそれらのファイルがあるものとあてにすべきではありません。

キーチェーンデータ

キーチェーンは、パスワードなどの秘密情報のための安全で暗号化されたコンテナです。アプリケーションのキーチェーンデータは、アプリケーションのサンドボックスの外に保存されます。アプリケーションがアンインストールされると、データは自動的に削除されます。ユーザがアプリケーションデータをiTunesを使用してバックアップすると、キーチェーンデータもバックアップされます。ただし、キーチェーンは、バックアップされたデバイスにのみ復元できます。アプリケーションのアップグレードはそのキーチェーンデータに影響しません。

iPhone OSキーチェーンの詳細については、『*Keychain Services Programming Guide*』の「Keychain Services Concepts」を参照してください。

アプリケーションディレクトリへのパスの取得

システムのさまざまなレベルにおいて、アプリケーションサンドボックスのディレクトリに対するファイルシステムパスを、プログラムによって取得する方法が存在します。ただし、これらのパスを取得する推奨される方法は、Cocoaプログラミングのインターフェイスを使用する方法です。FoundationフレームワークのNSHomeDirectory関数は、最上位のホームディレクトリ、つまり、アプリケーション、Documents、Library、およびtmpの各ディレクトリを含んでいるディレクトリへのパスを返します。この関数のほかに、NSSearchPathForDirectoriesInDomains関数およびNSTemporaryDirectory関数を使って、Documents、Caches、およびtmpの各ディレクトリへのパスを取得することもできます。

NSHomeDirectory関数とNSTemporaryDirectory関数はどちらも、正しい形式のパス情報をNSStringオブジェクトとして返します。NSStringのパス関連メソッドを使用して、パス情報に変更を加えたり、新しいパス文字列を作成したりできます。たとえば、一時ディレクトリのパスを取得して、ファイル名を追加し、結果として得られた文字列を使って一時ディレクトリ内に指定した名前のファイルを作成することができます。

注： ANSI Cのプログラムインターフェイス（パスを受け取るインターフェイスを含む）でフレームワークを使用している場合、NSStringオブジェクトは、Core Foundation版の同等のオブジェクトと「toll-free bridging」（犠牲を伴わない橋渡し）であることを思い出してください。つまり、NSStringオブジェクト（前述の関数のいずれかの戻り値など）は、以下の例に示すように、CFStringRef型にキャストできるということです。

```
CFStringRef homeDir = (CFStringRef)NSHomeDirectory();
```

「toll-free bridge」の詳細については、『*Carbon-Cocoa Integration Guide*』を参照してください。

Foundationフレームワークの NSSearchPathForDirectoriesInDomains関数を使うと、アプリケーション関連のディレクトリへのフルパスを取得できます。iPhone OSでこの関数を使用するには、第1パラメータに適切な検索パス定数、第2パラメータにNSUserDomainMaskを指定します。表6-2は、最もよく使われる定数と、その定数が返すディレクトリのリストです。

表 6-2　よく使われる検索パス定数

定数	ディレクトリ
NSDocumentDirectory	*<Application_Home>*/Documents
NSCachesDirectory	*<Application_Home>*/Library/Caches

定数	ディレクトリ
NSApplicationSupportDirectory	*<Application_Home>*/Library/Application Support

NSSearchPathForDirectoriesInDomains関数は、もともと、このようなディレクトリが複数存在するMac OS X用に設計されているため、1つのパスではなくパスの配列を返します。iPhone OSでは、その結果の配列に、要求したディレクトリへのパスが1つ含まれています。リスト 6-1は、この関数の一般的な使用方法を示します。

リスト 6-1　アプリケーションのDocuments/ディレクトリへのファイルシステムパスの取得

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *documentsDirectory = [paths objectAtIndex:0];
```

NSUserDomainMask以外のドメインマスクパラメータや、表 6-2 （134 ページ）に示した定数以外のディレクトリ定数を使用して、NSSearchPathForDirectoriesInDomainsを呼び出すこともできますが、アプリケーションは、その戻り値として返されたディレクトリに書き込むことはできません。たとえば、ディレクトリパラメータとしてNSApplicationDirectoryを、ドメインマスクパラメータとしてNSSystemDomainMaskを指定すると、（デバイス上では）/Applicationsというパスが戻り値として得られます。しかし、アプリケーションからこの場所へのファイルの書き込みはできません。

その他の注意点としては、プラットフォーム間のディレクトリの場所における違いです。NSSearchPathForDirectoriesInDomains、NSHomeDirectory、NSTemporaryDirectory、および類似の関数から返されるパスは、アプリケーションを実行しているのがデバイス上かシミュレータ上かにより異なります。たとえば、リスト 6-1 （135 ページ）に示した関数呼び出しを見てみます。デバイス上では、返されるパス(documentsDirectory)は以下のようになります。

```
/var/mobile/Applications/30B51836-D2DD-43AA-BCB4-9D4DADFED6A2/Documents
```

一方、iPhone Simulatorでは、返されるパスは次の形式をとります。

```
/Volumes/Stuff/Users/johnDoe/Library/Application Support/iPhone
Simulator/User/Applications/118086A0-FAAF-4CD4-9A0F-CD5E8D287270/Documents
```

ユーザの環境設定を読み書きするには、NSUserDefaultsクラスまたはCFPreferences APIを使用します。これらのインターフェイスにより、Library/Preferences/ディレクトリへのパスを作成したり、環境設定ファイルのディレクトリへ読み書きしたりする必要がなくなります。これらのインターフェイスの使用の詳細については、「Settingsバンドルの追加」 （197 ページ）を参照してください。

アプリケーションバンドルにサウンド、画像、またはその他のリソースが含まれている場合には、NSBundleクラスまたはCFBundleRef不透過型を使用してこれらのリソースをロードする必要があります。バンドルは、アプリケーション内のリソースの格納場所についてもともと情報を備えています。加えて、バンドルはユーザの言語に関する環境設定を認識し、デフォルトのリソースの代わりにローカライズ済みのリソースを自動的に優先して選択することができます。バンドルの詳細については、「アプリケーションバンドル」 （25 ページ）を参照してください。

ファイルデータの読み取りおよび書き込み

iPhone OSはファイルの読み取り、書き込み、および管理の方法をいくつか提供します。

- Foundationフレームワーク：
  - アプリケーションのデータをプロパティリストとして表現できる場合、NSPropertyListSerializationAPIを使用して、プロパティリストをNSDataオブジェクトに変換します。その後、NSDataクラスのメソッドを使用して、データオブジェクトをディスクに書き込むことができます。
  - アプリケーションのモデルオブジェクトでNSCodingプロトコルを採用する場合、NSKeyedArchiverクラス、特にこのクラスのarchivedDataWithRootObject:メソッドを使用してこれらのモデルオブジェクトのグラフをアーカイブできます。
  - FoundationフレームワークのNSFileHandleクラスを使用すれば、ファイルの内容へのランダムアクセスが可能です。
  - FoundationフレームワークのNSFileManagerクラスは、ファイルシステム内のファイルの作成、および操作のメソッドを提供します。

- Core OS呼び出し：
  - fopen、fread、およびfwriteなどの呼び出しを使用しても、順次アクセスまたはランダムアクセスを行ってファイルデータを読み書きできます。
  - mmapおよびmunmap呼び出しは、大きなファイルをメモリにロードし、その内容にアクセスするための効率的な方法を提供します。

注： Core OS呼び出しの前述のリストは、一般的によく使われている呼び出しの一例に過ぎません。利用できる関数の詳細なリストについては、『*iPhone OS Manual Pages*』のセクション3の関数の一覧を参照してください。

以降の各セクションでは、ファイルの読み書きを行うための高レベルの手法のいくつかについて、その使用法の例を示します。Foundationフレームワークのファイル関連クラスの詳細情報については、『*Foundation Framework Reference*』を参照してください。

プロパティリストデータの読み取りおよび書き込み

プロパティリストとは、いくつかのFoundation（およびCore Foundation）データ型（辞書、配列、文字列、日付、バイナリデータ、および数値とブール値を含む）をカプセル化する、データ表現の形式です。プロパティリストは、一般的に構造化された設定データを格納するために使用します。たとえば、CocoaアプリケーションおよびiPhoneアプリケーションのそれぞれにあるInfo.plistファイルは、アプリケーション自体の設定情報を格納するプロパティリストです。プロパティリストを使用して、アプリケーションの終了時の状態など、追加の情報を格納できます。

通常、コードでは、辞書または配列をコンテナオブジェクトとして使うところから始めてプロパティリストを構成します。その後、（場合によっては）ほかの辞書や配列を含んだその他のプロパティリストオブジェクトを追加します。辞書のキーは文字列オブジェクトでなければなりません。これらのキーの値は、NSDictionary、NSArray、NSString、NSDate、およびNSNumberのインスタンスです。

データがプロパティリストオブジェクト（NSDictionaryオブジェクトなど）によって表現され得るアプリケーションの場合、リスト 6-2に示すメソッドを使って、プロパティリストをディスクに書き込めます。このメソッドは、プロパティリストオブジェクトをNSDataオブジェクトへとシリアライズし、writeApplicationData:toFile:メソッド（リスト 6-4 （138 ページ）に示す実装）を呼び出して、このデータをディスクに書き込みます。

リスト 6-2 NSDataオブジェクトへのプロパティリストオブジェクトの変換とストレージへの書き込み

```
- (BOOL)writeApplicationPlist:(id)plist toFile:(NSString *)fileName {
    NSString *error;
    NSData *pData = [NSPropertyListSerialization dataFromPropertyList:plist
format:NSPropertyListBinaryFormat_v1_0 errorDescription:&error];
    if (!pData) {
        NSLog(@"%@", error);
        return NO;
    }
    return ([self writeApplicationData:pData toFile:(NSString *)fileName]);
}
```

iPhone OSでプロパティリストファイルを書き込む場合、ファイルをバイナリ形式で格納することが重要です。これを行うには、dataFromPropertyList:format:errorDescription:メソッドの*format*パラメータにNSPropertyListBinaryFormat_v1_0キーを指定します。バイナリ形式のプロパティリストは、テキストベースのその他の形式に比べればかなりコンパクトです。このコンパクトさによって、ユーザのデバイス上で占有する領域が減るだけでなく、プロパティリストの読み書きに費やす時間も減ります。

リスト 6-3に、ディスクからプロパティリストファイルをロードし、そのプロパティリスト内のオブジェクトを再構成するコードを示します。

リスト 6-3 アプリケーションのDocumentsディレクトリからのプロパティリストオブジェクトの読み取り

```
- (id)applicationPlistFromFile:(NSString *)fileName {
    NSData *retData;
    NSString *error;
    id retPlist;
    NSPropertyListFormat format;

    retData = [self applicationDataFromFile:fileName];
    if (!retData) {
        NSLog(@"Data file not returned.");
        return nil;
    }
    retPlist = [NSPropertyListSerialization propertyListFromData:retData
mutabilityOption:NSPropertyListImmutable format:&format errorDescription:&error];
    if (!retPlist){
        NSLog(@"Plist not returned, error:%@", error);
    }
    return retPlist;
}
```

プロパティリストおよびNSPropertyListSerializationクラスの詳細については、『*Property List Programming Guide*』を参照してください。

アーカイバを使ったデータの読み取りおよび書き込み

アーカイバは、任意のオブジェクトコレクションをバイトストリームに変換します。NSPropertyListSerializationクラスで採用しているプロセスに似ているように感じるかもしれませんが、重要な違いが1つあります。つまり、プロパティリストのシリアライザが変換できるのは、限られたデータ型（ほとんどはスカラー）だけであるということです。アーカイバは、任意のObjective-Cオブジェクト、スカラー型、配列、構造体、文字列などを変換できます。

アーカイブプロセスの鍵は、対象オブジェクト自身の中にあります。アーカイバが操作するオブジェクトは、オブジェクトの状態を読み書きするインターフェイスを定義している、NSCodingプロトコルに準拠する必要があります。アーカイバは、オブジェクトの集合をエンコードするとき、それぞれのオブジェクトに対してencodeWithCoder:メッセージを送ります。オブジェクトは、これに基づいて、対応するアーカイブに重要な状態情報を書き出します。展開プロセスは、情報の流れを逆転します。展開する間、各オブジェクトは、initWithCoder:メッセージを受け取ります。オブジェクトは、このメッセージを使用して、アーカイブ内の現在の状況情報で自身を初期化します。展開プロセスが完了すると、バイトストリームは、以前にアーカイブに書き込んだものと同じ状態を持った新しいオブジェクトの集合として再構築されます。

Foundationフレームワークは、シーケンシャルとキー付きの2種類のアーカイブをサポートしています。キー付きアーカイバはより柔軟性が高く、アプリケーションで使用することをお勧めします。次の例では、キー付きアーカイバを使ってオブジェクトのグラフをアーカイブする方法を示します。_myDataSourceオブジェクトのrepresentationメソッドは、アーカイブに含めるすべてのオブジェクトを指す単一のオブジェクト（配列または辞書）を返します。その後、データオブジェクトは、myFilePath変数に指定されているパスのファイルに書き込まれます。

```
NSData *data = [NSKeyedArchiver archivedDataWithRootObject:[_myDataSource
representation]];
[data writeToFile:myFilePath atomically:YES];
```

注： archiveRootObject:toFile:メッセージをNSKeyedArchiverオブジェクトに送信すると、1ステップで、アーカイブを作成し、そのアーカイブをストレージに書き込めます。

ディスクからアーカイブの内容をロードするには、単にプロセスを逆転させます。ディスクからデータをロードしたら、NSKeyedUnarchiverクラスおよびそのunarchiveObjectWithData:クラスメソッドを使用して、モデルオブジェクトグラフを復元します。たとえば、前述の例からデータを展開するには、以下のコードを使用できます。

```
NSData* data = [NSData dataWithContentsOfFile:myFilePath];
id rootObject = [NSKeyedUnarchiver unarchiveObjectWithData:data];
```

アーカイバの使用方法およびオブジェクトでNSCodingプロトコルをサポートする方法の詳細については、『*Archives and Serializations Programming Guide for Cocoa*』を参照してください。

Documentsディレクトリへのデータの書き込み

アプリケーションデータを（アーカイブまたはシリアライズされたプロパティリストとして）カプセル化したNSDataオブジェクトを作成したら、リスト 6-4に示すメソッドを呼び出して、このデータをアプリケーションのDocumentsディレクトリに書き込めます。

リスト 6-4　アプリケーションのDocumentsディレクトリへのデータの書き込み

```
- (BOOL)writeApplicationData:(NSData *)data toFile:(NSString *)fileName {
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    if (!documentsDirectory) {
        NSLog(@"Documents directory not found!");
        return NO;
    }
    NSString *appFile = [documentsDirectory
stringByAppendingPathComponent:fileName];
```

```
    return ([data writeToFile:appFile atomically:YES]);
}
```

Documentsディレクトリからのデータの読み取り

アプリケーションのDocumentsディレクトリからファイルを読み取るには、ファイル名のパスを作成し、望みのメソッドを使用してメモリにファイルの内容を読み込みます。比較的小さなファイル、つまり、メモリページが数ページ分のサイズもないファイルについては、リスト6-5に示すコードを使用してファイルの内容のデータオブジェクトを取得できます。この例では、Documentsディレクトリ内のファイルへのフルパスを構成し、そこからデータオブジェクトを作成してそのオブジェクトを返します。

リスト 6-5　アプリケーションのDocumentsディレクトリからのデータの読み取り

```
- (NSData *)applicationDataFromFile:(NSString *)fileName {
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *appFile = [documentsDirectory
stringByAppendingPathComponent:fileName];
    NSData *myData = [[[NSData alloc] initWithContentsOfFile:appFile]
autorelease];
    return myData;
}
```

メモリに保持するために複数のメモリページを必要するファイルについては、ファイル全体を一度にロードしないようにします。これは、ファイルの一部のみを使用する場合、特に重要です。より大きなファイルについては、mmap関数、またはNSDataのinitWithContentsOfMappedFile:メソッドを使用して、メモリにファイルをマップすることを検討すべきです。

ファイルをマップするか、直接ロードするかの判断はデベロッパに任されています。必要なメモリページがごくわずか（3〜4ページ）の場合、ファイル全体をメモリにロードしても比較的安全です。しかし、数十ページ、または数百ページを必要とするファイルの場合、おそらくファイルをメモリにマップする方が効率的です。ただし、ほかの同様の決断をする場合のように、デベロッパはアプリケーションのパフォーマンスを測定し、ファイルのロードおよび必要なメモリの割り当てに要する時間を調べるべきです。

ファイルアクセスガイドライン

ファイルを作成したりファイルにデータを書き込む際には、以下のガイドラインを念頭に置いてください。

- ディスクに書き込むデータ量を最小限にする。ファイル操作は、比較的低速であり、寿命が限られているフラッシュディスクへの書き込みを伴います。ファイル関連操作を最小限にするための具体的なヒントを、以下に示します。
  - ファイルの変更部分だけを書き込み、変更はできるだけまとめる。わずか数バイトの変更のために、ファイル全体を書き込むようなことは避けるようにします。
  - ファイル形式を定義する際には、毎回ディスクに書き込む必要があるブロック数が最少になるように、頻繁に更新されるコンテンツをグループにまとめる。

- ランダムにアクセスされる構造化されたコンテンツで構成されるデータは、Core Data永続ストアまたはSQLiteデータベースに保存する。この点は、操作するデータの量が数メガバイトを越える可能性がある場合、特に重要です。

- ディスクへのキャッシュファイルの書き込みを避ける。この規則の唯一の例外は、アプリケーションを終了する際、次の起動時に同じ状態へアプリケーションを戻すために使用する、状態情報を書き込む必要がある時です。

状態情報を保存する

ユーザがホーム(Home)ボタンを押すと、iPhone OSはアプリケーションを終了して、ホーム(Home)画面に戻ります。同様に、アプリケーションが、別のアプリケーションによって処理されるスキームのURIを開くと、iPhone OSは現在のアプリケーションを終了して、他方のアプリケーションでそのURIを開きます。つまり、Mac OS Xでは、アプリケーションを一時停止したりバックグラウンドにするようなアクションが、iPhone OSではアプリケーションを終了させることになります。モバイルデバイスでは、このようなアクションがよく発生するので、アプリケーションは、揮発性のデータやアプリケーションの状態を管理する方法を変更しなければなりません。

ほとんどのデスクトップアプリケーションでは、いつファイルをディスクに保存するかをユーザが手動で選択します。これとは異なり、iPhoneアプリケーションでは、そのワークフローの中の重要な時点で、変更を自動的に保存する必要があります。いつデータを保存するかはデベロッパに任されていますが、考えられる選択肢は2つあります。1つは、ユーザが変更を行ったらすぐにその変更を保存することです。もう1つは、同じページでの変更をまとめておいて、そのページを閉じるとき、新しいページを開くとき、またはアプリケーションが終了するときに変更を保存することです。どのような場合にも、前のページのデータを保存せずに新しいページのコンテンツにユーザが進めるようにすべきではありません。

アプリケーションが終了する場合は、アプリケーションの現在の状態を一時キャッシュファイル、または環境設定データベースに保存します。次回ユーザがアプリケーションを起動したときには、その情報を使用して、アプリケーションを以前の状態に復元します。保存する状態情報は、できる限り最小にする必要があります。ただし、アプリケーションを適切な状態に正確に復元できるようにしなければなりません。ユーザが以前操作していたものとまったく同じ画面を復元しても意味がない場合は、そこまでする必要はありません。たとえば、ユーザが連絡先情報を編集した後に「電話(Phone)」アプリケーションを離れた場合、戻ったときに、「電話(Phone)」アプリケーションは、その連絡先情報の編集画面ではなく、連絡先情報のトップレベルのリストを表示します。

大文字小文字の区別

iPhone OSベースのファイルシステムは、大文字小文字を区別します。ファイル名を扱う場合は、大文字小文字まで正確に一致させる必要があります。さもなければ、コードからファイルを開けなかったり、ファイルにアクセスしたりできない場合があります。

ネットワーク

iPhone OSのネットワークスタックには、iPhoneおよびiPod touchデバイスの無線ハードウェアをカバーするインターフェイスがいくつか含まれています。主たるプログラミングインターフェイスはCFNetworkフレームワークです。これは、Core FoundationフレームワークのBSDソケットと不透過型

の最上位に構築されており、ネットワークエンティティと通信します。また、Foundationフレームワークの`NSStream`クラスや、システムのCore OSレイヤにある低レベルのBSDソケットを使用することもできます。

以降の各セクションでは、アプリケーションにネットワーク機能を組み込む必要があるデベロッパ向けに、iPhone固有のヒントを示します。ネットワーク通信にCFNetworkフレームワークを使用する方法の詳細については、『*CFNetwork Programming Guide*』および『*CFNetwork Framework Reference*』を参照してください。`NSStream`クラスの使用方法については、『*Foundation Framework Reference*』を参照してください。

効率的なネットワーク処理のためのヒント

ネットワーク経由で送受信を行うコードを実装する場合、その送受信がデバイス上で最も電力を集中的に使う操作の1つであると覚えておいてください。送受信にかける時間を最小限にすることがバッテリー持続時間の向上に寄与します。そのためには、ネットワーク関連コードの作成時には、以下のヒントを考慮してください。

- デベロッパのコントロール下にあるプロトコルについては、可能な限りコンパクトなデータ形式を定義する。
- やりとりを頻繁に行うプロトコルを使用した通信を避ける。
- 可能な場合はデータをバースト転送する。

携帯電話無線機能およびWi-Fi無線機能は、アクティブでないときには電源がオフになるように設計されています。しかし、無線の状態によりますが、これには数秒間かかる場合があります。アプリケーションが数秒ごとに少量データのバースト転送を行う場合、実際には何もしていなくても無線機能の電源がオフにならずに、電力を消費し続ける可能性があります。頻繁に少量のデータを転送するよりも、多量のデータを一度に転送するか、比較的長い間隔で転送するほうが良いでしょう。

ネットワーク経由で通信する場合、パケットはいつでも失われる可能性があると覚えておくことも重要です。ネットワークのコードを作成する際には、エラー処理に関してできる限り強固なコードにすべきです。ネットワーク状態の変化に対応するハンドラを実装することは理にかなっていますが、これらのハンドラが一貫性を持って呼び出されなくても驚かないでください。たとえば、Bonjourネットワーキングのコールバックは、ネットワークサービスが失われても、必ずしもすぐに呼び出されるわけではありません。Bonjourシステムサービスは、サービスが失われたという通知を受け取るとブラウズコールバックをすぐに呼び出しますが、ネットワークサービスは通知なく失われる場合もあります。たとえば、ネットワークサービスを提供しているデバイスが予期しないときにネットワーク接続を失ったり、通知が転送中に失われたりする可能性があります。

Wi-Fiの使用

Wi-Fi無線を使用してネットワークアクセスをする場合は、そのことをシステムに通知する必要があります。それには、アプリケーションの`Info.plist`ファイルに`UIRequiresPersistentWiFi`キーを含めます。このキーを含めると、アクティブなWi-Fiホットスポットを検出したら、ネットワーク選択パネルにそれを表示する必要があることをシステムに知らせることができます。また、アプリケーションの実行中はWi-Fiハードウェアを停止させてはならないことをシステムに知らせることもできます。

Wi-Fiハードウェアが電力を消費し過ぎるのを防ぐために、iPhone OSにはタイマーが組み込みこまれており、UIRequiresPersistentWiFiキーを通してWi-Fiハードウェアを利用するアプリケーションがない場合は、30分経過したらこのハードウェアが完全にオフになります。ユーザがこのキーを含むアプリケーションを起動すると、iPhone OSは、アプリケーションが実行している間は、事実上このタイマーを無効にします。ただし、アプリケーションが終了したらすぐに、システムはタイマーを有効にし直します。

注： UIRequiresPersistentWiFiの値がtrueの場合でも、デバイスが待機中（つまり画面がロックされている状態）であればなんの影響もありません。アプリケーションはある程度は動作しますが、アクティブではないと見なされ、Wi-Fi接続を持ちません。

UIRequiresPersistentWiFiキーおよびInfo.plistファイルのキーの詳細については、「情報プロパティリスト」（28 ページ）を参照してください。

機内モード警告

デバイスを機内モードにしてアプリケーションを起動している場合、システムがその事実をユーザに通知するためにパネルを表示することがあります。システムがこのパネルを表示するのは、次の条件のすべてを満たした場合です。

- アプリケーションの情報プロパティリスト(Info.plist)ファイルに、UIRequiresPersistentWiFiキーが含まれ、そのキーの値にtrueが設定されている場合。
- デバイスが機内モードでアプリケーションを起動した場合。
- デバイスのWi-Fiが、機内モードに切り替わった後、手動で有効に戻されていない場合。

マルチメディアサポート

マルチメディア機能がアプリケーションにとって中心的か付随的かに関わらず、iPhoneユーザは高い品質を期待しています。ビデオでは、デバイスの高解像度画面と高いフレームレートを活用すべきです。魅力的なオーディオによって、アプリケーションのユーザ体験全体は限りなく向上します。

iPhone OSのマルチメディアフレームワークを活用すると、次のような機能を追加できます。

- 高品質なオーディオの録音と再生
- 没入型ゲームのサウンド
- ライブボイスチャット
- ユーザのiPodライブラリからのコンテンツの再生
- サポートされているデバイス上でのビデオの再生と録画

この章では、これらのオーディオおよびビデオ機能をアプリケーションに追加するための、iPhone OSのマルチメディアテクノロジーについて紹介します。

iPhone OSでのサウンドの使用

iPhone OSは、アプリケーションでサウンドを扱うための豊富なツールセットを提供しています。これらのツールは、それが提供する機能に応じて次のような形でフレームワークに配置されています。

- 単純なObjective-Cインターフェイスを使用してオーディオの再生と録音を行うには、AV Foundationフレームワークを使用します。
- 同期機能付きでオーディオの再生と録音を行ったり、オーディオストリームを解析したり、オーディオフォーマットを変換したりするには、Audio Toolboxフレームワークを使用します。
- オーディオ処理用のプラグインに接続したりそれを使用したりするには、Audio Unitフレームワークを使用します。
- ゲームその他のアプリケーションで定位オーディオ再生を提供するには、OpenALフレームワークを使用します。iPhone OSでサポートされているOpenAL 1.1は、Core Audioの上に構築されています。
- ユーザのiPodライブラリから曲、オーディオブック、またはオーディオPodcastを再生するには、Media PlayerフレームワークのiPodライブラリアクセスAPIを使用します。

Core Audioフレームワーク（ほかのオーディオフレームワークと同等です）は、すべてのCore Audioサービスで使われるデータ型を提供します。

このセクションでは、幅広いオーディオ機能の実装に着手する際のガイダンスを示します。

- ユーザのiPodライブラリから曲、オーディオPodcast、およびオーディオブックを再生するには、「iPodライブラリアクセスによるメディアアイテムの再生」（148ページ）参照してください。
- 警告やユーザインタフェースのサウンドエフェクトを再生したり、デバイス上でバイブレーションを起こすには、System Sound Servicesを使用します。「System Sound Servicesを使用して短いサウンドを再生したり、バイブレーションを起動する」（149 ページ）を参照してください。
- 最小限のコードでオーディオの再生や録音を行うには、AV Foundationフレームワークを使用します。「AVAudioPlayerクラスを利用して簡単にサウンドを再生する」（150 ページ）および「AVAudioRecorderクラスを利用した録音」（156 ページ）を参照してください。
- ステレオポジション、音量制御、同時再生などのフル装備のオーディオ再生を提供するには、OpenALを使用します。「OpenALを使用したポジショニングを伴うサウンドの再生」（155 ページ）を参照してください。
- 遅延が最小のオーディオを提供するには（特に、VoIPアプリケーションなどのために入力と出力を同時に行う場合）、I/Oオーディオユニットを使用します。「iPhone OSでのオーディオユニットのサポート」（159 ページ）を参照してください。
- 最高レベルの制御でサウンドを再生するには（同期のサポートも含む）、Audio Queue Servicesを使用します。「Audio Queue Servicesを使用した制御を伴うサウンドの再生」（152 ページ）を参照してください。Audio Queue Servicesは録音もサポートします（「Audio Queue Servicesを利用した録音」（157 ページ）を参照）。
- ネットワーク接続からのオーディオストリームを解析するには、Audio File Stream Servicesを使用します。「ストリーミングされたオーディオの解析」（158 ページ）を参照してください。

iPhone OSベースのデバイス上でオーディオがどのように動作するかについての重要な情報が書かれているので、次のセクション「基礎：ハードウェアコーデック、オーディオフォーマット、オーディオセッション」（144 ページ）は必ずお読みください。また、「iPhoneオーディオのベストプラクティス」（160 ページ）もお読みください。ここでは、最高のパフォーマンスと最高のユーザ体験を実現するためのガイドラインを提供し、使用すべきオーディオフォーマットおよびファイル形式のリストを示します。

さらに深く学びたい場合は、iPhone Dev Centerで提供されているガイド、リファレンスブック、サンプルコード、その他を参照してください。一般的なオーディオタスクの実行方法についてのヒントは、『*Audio & Video Coding How-To's*』を参照してください。iPhone OSでのオーディオ開発についての詳しい説明は、『*Core Audio Overview*』、『*Audio Queue Services Programming Guide*』、および『*Audio Session Programming Guide*』を参照してください。

基礎：ハードウェアコーデック、オーディオフォーマット、オーディオセッション

iPhoneオーディオの開発を始めるには、iPhone OSベースのデバイスのハードウェアおよびソフトウェアのアーキテクチャについて少し理解しておくと大変役立ちます。

iPhoneオーディオのハードウェアコーデック

iPhone OSアプリケーションでは、幅広いオーディオデータフォーマットを使用できます。iPhone OS 3.0以降では、ほとんどのフォーマットでソフトウェアベースのエンコードとデコードを使用できます。すべてのフォーマットのサウンドを複数同時に再生できます。ただし、パフォーマンス上の理

由から与えられたシナリオでどのフォーマットが最適かを考慮する必要があります。ハードウェアデコードの方がソフトウェアデコードよりもパフォーマンスに与える影響が少ないのが一般的です。

次のiPhone OSのオーディオフォーマットでは、再生時にハードウェアデコードを採用できます。

- AAC
- ALAC (Apple Lossless)
- MP3

ハードウェアによって一度に再生できるのはこれらのいずれかのフォーマットの1つのインスタンスのみです。たとえば、ステレオMP3サウンドを再生している場合、別のMP3サウンドを同時に再生するにはソフトウェアデコードを使用します。同様に、ハードウェアを使用してAACとALACのサウンドを同時に再生することはできません。iPodアプリケーションがバックグラウンドでAACを再生している場合、アプリケーションはソフトウェアデコードを使用してAAC、ALAC、およびMP3オーディオを再生します。

複数のサウンドを最高のパフォーマンスで再生したり、バックグラウンドでiPodを再生している間に効率的にサウンドを再生したりするには、リニアPCM（圧縮なし）またはIMA4（圧縮あり）のオーディオを使用します。

デバイスで利用可能なハードウェアコーデックとソフトウェアコーデックの確認方法については、『*Audio Format Services Reference*』の`kAudioFormatProperty_HardwareCodecCapabilities`定数についての説明を参照してください。

オーディオの再生フォーマットと録音フォーマット

以下は、iPhone OSでサポートされているオーディオ再生フォーマットです。

- AAC
- HE-AAC
- AMR（Adaptive Multi-Rate。音声用のフォーマット）
- ALAC (Apple Lossless)
- iLBC（internet Low Bitrate Codec。音声向けのもう1つのフォーマット）
- IMA4 (IMA/ADPCM)
- リニアPCM（圧縮なし）
- µ-lawおよびa-law
- MP3 (MPEG-1 audio layer 3)

以下は、iPhone OSでサポートされているオーディオ録音フォーマットです。

- ALAC (Apple Lossless)
- iLBC（internet Low Bitrate Codec。音声向け）
- IMA/ADPCM (IMA4)
- リニアPCM
- µ-lawおよびa-law

以下のリストは、iPhone OSが、個別再生または複数再生のためにサポートしているオーディオフォーマットの要約です。

- **リニアPCMおよびIMA4 (IMA/ADPCM)**。リニアPCMフォーマットとIMA4のサウンドは、iPhone OSでCPUリソースの問題なしに複数同時に再生できます。これは、音声品質のAMRフォーマットとiLBCフォーマット、およびμ-lawおよびa-lawの各圧縮フォーマットについても同様です。圧縮フォーマットを使用する場合は、サウンドの品質がニーズに合うかどうかを確認してください。
- **AAC、MP3、ALAC (Apple Lossless)**。AAC、MP3、およびALACのサウンドの再生には、iPhone OSベースのデバイス上の効率的なハードウェアベースのデコードが使えます。ただし、これらのコーデックはすべて1つのハードウェアパスを共有します。ハードウェアによって一度に再生できるのは、これらのいずれかのフォーマットの1つのインスタンスのみです。

AAC、MP3、およびALACの再生に1つのハードウェアパスということは、仮想ピアノなどの「次々と再生する」スタイルのアプリケーション向けである言えます。ユーザがiPodアプリケーションでこれら3つのいずれかのフォーマットのサウンドを再生している場合、（そのオーディオと同時に再生するには）アプリケーションはソフトウェアデコードを採用します。

オーディオセッション

Core Audioのオーディオセッションインターフェイス（『*Audio Session Services Reference*』で説明されている）を利用すると、アプリケーションは一般的なオーディオ動作を定義して、そのアプリケーションが実行されているデバイスの大きい方のオーディオコンテキスト内でうまく動作させることができます。設定可能な動作には次のようなものが含まれます。

- 着信／サイレントスイッチによってオーディオを消音させるかどうか
- 画面ロックと同時にオーディオを停止するかどうか
- オーディオが始まったときに、iPodオーディオの再生を継続するか消音させるか

大きい方のオーディオコンテキストには、ユーザによる変更（ヘッドセットの接続時など）やイベント（「時計(Clock)」や「カレンダー(Calendar)」のアラーム、着信呼び出しなど）が含まれます。オーディオセッションを使用するとこのようなイベントに適切に応答できます。

Audio Session Servicesには、表 7-1に示す3つのプログラム機能があります。

表 7-1　オーディオセッションインターフェイスによって提供される機能

オーディオセッションの機能	説明
カテゴリ	カテゴリは、アプリケーションの一連のオーディオ動作を識別するキーです。カテゴリを設定することによって、オーディオの意図（たとえば、画面がロックされたときにオーディオの再生を続けるかどうかなど）をiPhone OSに伝えます。
割り込み、および出力先の変更	オーディオ割り込みの発生時、終了時、およびハードウェアのオーディオ出力先が変更されたときに、オーディオセッションから通知を送ります。これらの通知によって、大きい方のオーディオ環境での変更（電話の着信による割り込みなど）に、うまく応答できるようになります。

オーディオセッションの機能	説明
ハードウェアの特性	オーディオセッションに問い合わせて、アプリケーションが実行されているデバイスの特性（ハードウェアのサンプルレート、ハードウェアのチャネル数、オーディオ入力を使用できるかどうかなど）を調べることができます。

『*AVAudioSession Class Reference*』と『*AVAudioSessionDelegate Protocol Reference*』では、オーディオセッションを管理するための簡素化されたObjective-Cインターフェイスについて説明しています。割り込み用にオーディオセッションを設定するには、C言語ベースのAudio Session Serviceを直接利用します。このインターフェイスについては『*Audio Session Services Reference*』で説明しています。両方のインターフェイスのコードをアプリケーション内に混在させることもできます。

オーディオセッションは、すぐに開発に使用できるデフォルトの動作をいくつか備えています。ただし、このデフォルト動作は、特殊なケースを除いてオーディオを使用する製品版のアプリケーションには適しません。オーディオセッションを設定して使用することにより、オーディオの意図を表現したり、OSレベルのオーディオ判断に対応したりできます。

たとえば、デフォルトのオーディオセッションを使用すると、Auto-Lock期間がタイムアウトしたり画面がロックしたときに、アプリケーションのオーディオが停止します。画面がロックされても確実に再生を続けたい場合は、アプリケーションの初期化コードに次の行を含めます。

```
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryPlayback
error:nil];
[[AVAudioSession sharedInstance] setActive:YES error:nil];
```

`AVAudioSessionCategoryPlayback`カテゴリは、画面ロック時でも再生が継続することを保証します。オーディオセッションをアクティブにすると、指定されたカテゴリがアクティブになります。カテゴリの詳細については、『*Audio Session Programming Guide*』の「Audio Session Categories」を参照してください。

電話の着信や時計のアラームによる割り込みを処理する方法は、使用中のオーディオテクノロジーによって異なります。それを表 7-2に示します。

表 7-2　オーディオ割り込みの処理方法

オーディオテクノロジー	割り込み時の動作
System Sound Services	割り込みが開始されると、システムサウンドと警告音は鳴らなくなります。割り込みが終了すると（ユーザがアラームを消したり、ユーザが電話の着信呼び出しを無視した場合）、これらのサウンドは自動的に使用可能な状態に戻ります。このテクノロジーを使うサウンドについて、割り込み時の動作をアプリケーションが変更することはできません。
Audio Queue Services、OpenAL、I/Oオーディオユニット	これらのテクノロジーは、割り込みの処理を制御する際に最も柔軟性に富んでいます。これらのテクノロジーを使う場合は、アプリケーションのコード内に割り込みリスナーコールバック関数を作成します。『*Audio Session Programming Guide*』の「Responding to Audio Interruptions」の説明を参照してください。

オーディオテクノロジー	割り込み時の動作
AVAudioPlayerクラス	AVAudioPlayerクラスは、割り込みの開始と終了に関するデリゲートメソッドを備えています。必要に応じてaudioPlayerBeginInterruption:メソッドを実装することで、ユーザインターフェイスの更新処理ができます。再生の一時停止は、オーディオプレーヤーオブジェクトによって処理されます。audioPlayerEndInterruption:メソッドを実装することで、再生の再開処理と、必要に応じてユーザインターフェイスの更新処理ができます。オーディオセッションの再アクティブ化は、オーディオプレーヤーオブジェクトによって処理されます。

すべてのiPhone OSアプリケーションは、（稀に例外はありますが）Audio Session Servicesを採用するべきです。その方法については、『*Audio Session Programming Guide*』を参照してください。

オーディオの再生

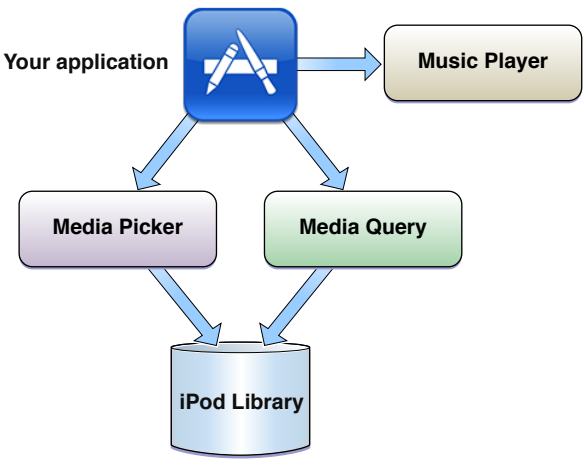
このセクションでは、iPodライブラリアクセス、System Sound Services、Audio Queue Services、AV Foundationフレームワーク、およびOpenALを使用した、iPhone OSでのサウンドの再生について紹介します。

iPodライブラリアクセスによるメディアアイテムの再生

iPhone OS 3.0から、iPodライブラリアクセスによりアプリケーションでユーザの曲、オーディオブック、およびオーディオPodcastを再生できます。APIの設計は基本的な再生を非常に単純にしつつ、高度な検索や再生制御もサポートしています。

図 7-1に示すように、アプリケーションには2通りのアイテムの取得方法があります。左側のメディアアイテムピッカーは、簡単に使用できるパッケージ済みのView Controllerで、組み込みのiPodアプリケーションのミュージック選択インターフェイスのように動作します。多くのアプリケーションにはこれで十分です。要求される特別なアクセス制御をピッカーで提供できない場合、メディアクエリインターフェイスを使用できます。これを使用して、iPodライブラリからアイテムを述語ベースで指定できます。

図 7-1 iPodライブラリアクセスの使用



図でアプリケーションの右側に描かれているように、このAPIが提供するミュージックプレーヤーを使用して、取得したメディアアイテムを再生します。

メディアアイテムの再生をアプリケーションに追加する方法の詳細については、『*iPod Library Access Guide*』を参照してください。

System Sound Servicesを使用して短いサウンドを再生したり、バイブレーションを起動する

ユーザインターフェイスのサウンドエフェクト（ボタンのクリック音など）や警告音を再生したり、デバイス上でバイブレーションを起動するには、System Sound Servicesを使用します。このコンパクトなインターフェイスについては、『*System Sound Services Reference*』で説明されています。iPhone Dev Centerの*SysSound*サンプルにサンプルコードがあります。

注： System Sound Servicesを使用して再生されたサウンドは、オーディオセッションを使用した設定を前提としていません。その結果、System Sound Servicesオーディオの動作を、アプリケーションのほかのオーディオ動作と同じように管理することはできません。このため、System Sound Servicesの本来の目的と異なるオーディオにはSystem Sound Servicesを使わないでください。

`AudioServicesPlaySystemSound`関数を利用すると、非常に簡単に短いサウンドファイルを再生できます。単純化されているために、この関数にはいくつかの制限があります。サウンドファイルは、以下の条件を満たす必要があります。

- 長さが30秒未満である
- リニアPCMフォーマットまたはIMA4 (IMA/ADPCM)フォーマットである
- `.caf`ファイル、`.aif`ファイル、または`.wav`ファイルになっている

さらに、`AudioServicesPlaySystemSound`関数を使用する場合は、以下の制限があります。

- サウンドは現在のシステムオーディオレベルで再生され、レベルの制御はできない
- すぐに再生される
- ループおよびステレオポジショニングは利用できない

類似の`AudioServicesPlayAlertSound`関数は、短いサウンドを警告として再生します。ユーザが着信設定(Ring Settings)でデバイスをバイブレーションに設定すると、この関数を呼び出すことによって、サウンドファイルを再生すると同時にバイブレーションを起動します。

注： システムが提供する警告音と、システムが提供するユーザインターフェイスのサウンドエフェクトは、アプリケーションでは利用できません。たとえば、`kSystemSoundID_UserPreferredAlert`定数をパラメータとして`AudioServicesPlayAlertSound`関数に渡しても、何も再生されません。

`AudioServicesPlaySystemSound`関数や`AudioServicesPlayAlertSound`関数でサウンドを再生するには、リスト 7-1に示すように、最初にサウンドIDオブジェクトを作成します。

リスト 7-1　サウンドIDオブジェクトの作成

```
// アプリケーションのメインバンドルを取得する
CFBundleRef mainBundle = CFBundleGetMainBundle ();
```

```
    // 再生対象のサウンドファイルのURLを取得する。この場合のファイルは
    // "tap.aiff"。
    soundFileURLRef  =    CFBundleCopyResourceURL (
                              mainBundle,
                              CFSTR ("tap"),
                              CFSTR ("aif"),
                              NULL
                          );

    // このサウンドファイルを表すシステムサウンドオブジェクトを作成する
    AudioServicesCreateSystemSoundID (
        soundFileURLRef,
        &soundFileObject
    );
```

次に、リスト 7-2に示すようにしてサウンドを再生できます。

リスト 7-2　システムサウンドの再生

```
- (IBAction) playSystemSound {
    AudioServicesPlaySystemSound (self.soundFileObject);
}
```

たびたび、または繰り返し再生するなどの典型的な使いかたでは、アプリケーションが終了するまでサウンドIDオブジェクトを保持します。起動時のサウンドなど、一度しかサウンドを使用しないことがわかっている場合は、サウンドを再生したらすぐにサウンドIDオブジェクトを破棄してメモリを解放します。

iPhone OSベースのデバイス上で実行される、バイブレーションをサポートするアプリケーションは、System Sound Servicesを使用してバイブレーションを起動します。バイブレーションオプションは`kSystemSoundID_Vibrate`識別子で指定します。バイブレーションを起動するには、リスト 7-3に示すように、`AudioServicesPlaySystemSound`関数を使います。

リスト 7-3　バイブレーションの起動

```
#import <AudioToolbox/AudioToolbox.h>
#import <UIKit/UIKit.h>
- (void) vibratePhone {
    AudioServicesPlaySystemSound (kSystemSoundID_Vibrate);
}
```

iPod touchでアプリケーションを実行している場合は、このコードでは何も起こりません。

AVAudioPlayerクラスを利用して簡単にサウンドを再生する

`AVAudioPlayer`クラスは、サウンドを再生するためのObjective-C簡易インターフェイスを提供します。アプリケーションでステレオポジショニングや正確な同期を必要とせず、ネットワークストリームからキャプチャしたオーディオを再生しない場合は、このクラスを再生に使用することをお勧めします。

オーディオプレーヤーを使用すると、次を実行できます。

- 任意の所要時間のサウンドの再生
- ファイルまたはメモリバッファからのサウンドの再生

- サウンドのループ再生
- 複数のサウンドの同時再生（ただし、正確な同期は伴わない）
- 再生中の各サウンドの相対的な再生レベルの制御
- サウンドファイル内での特定の位置へのシーク（早送りや巻き戻しなどのアプリケーション機能をサポートします）
- オーディオレベル測定機能に使用できるオーディオパワーデータの取得

AVAudioPlayerクラスを利用すると、iPhone OSで利用可能な任意のオーディオフォーマット（「オーディオの再生フォーマットと録音フォーマット」（145 ページ）を参照）のサウンドを再生できます。このクラスのインターフェイスの詳細については、『*AVAudioPlayer Class Reference*』を参照してください。

オーディオプレーヤーを再生用に設定するには、サウンドファイルを割り当てて、再生の準備を行い、デリゲートオブジェクトを指定します。リスト 7-4に示すコードは、通常、アプリケーションのコントローラクラスの初期化メソッドになります。

リスト 7-4　AVAudioPlayerオブジェクトの設定

```
// 対応する.hファイル内で：
// @property (nonatomic, retain) AVAudioPlayer *player;

@synthesize player; // プレーヤーオブジェクト

NSString *soundFilePath =
                [[NSBundle mainBundle] pathForResource:@"sound"
                                                ofType:@"wav"];

NSURL *fileURL = [[NSURL alloc] initFileURLWithPath:soundFilePath];

AVAudioPlayer *newPlayer =
                [[AVAudioPlayer alloc] initWithContentsOfURL:fileURL
                                                       error:nil];
[fileURL release];

self.player = newPlayer;
[newPlayer release];

[player prepareToPlay];
[player setDelegate:self];
```

サウンドの再生が完了したときに、デリゲートオブジェクト（コントローラオブジェクトを割り当てることもできる）を使用して割り込みを処理し、ユーザインターフェイスを更新します。AVAudioPlayerクラスのデリゲートメソッドについては、『*AVAudioPlayerDelegate Protocol Reference*』を参照してください。リスト 7-5は、1つのデリゲートメソッドを簡単に実装したものです。このコードは、サウンドの再生が完了したときに、「再生／一時停止(Play/Pause)」トグルボタンのタイトルを更新します。

リスト 7-5　AVAudioPlayerのデリゲートメソッドの実装

```
- (void) audioPlayerDidFinishPlaying:(AVAudioPlayer *) player
                        successfully:(BOOL) flag {
    if (flag == YES) {
        [self.button setTitle:@"Play" forState:UIControlStateNormal];
    }
```

```
}
```

AVAudioPlayerオブジェクトを再生、一時停止、または停止するには、そのいずれかの再生制御メソッドを呼び出します。再生が実行中かどうかを調べるには、playingプロパティを使用します。リスト 7-6は、再生を制御し、UIButtonオブジェクトのタイトルを更新する、基本的な再生／一時停止切り替えメソッドを示しています。

リスト 7-6 AVAudioPlayerオブジェクトの制御

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

    // すでに再生中の場合は一時停止する
    if (self.player.playing) {
        [self.button setTitle:@"Play" forState:UIControlStateHighlighted];
        [self.button setTitle:@"Play" forState:UIControlStateNormal];
        [self.player pause];

    // 停止または一時停止状態の場合は、再生を開始する
    } else {
        [self.button setTitle:@"Pause" forState:UIControlStateHighlighted];
        [self.button setTitle:@"Pause" forState:UIControlStateNormal];
        [self.player play];
    }
}
```

AVAudioPlayerクラスでは、サウンドに関する情報（サウンドのタイムライン内の再生ポイントなど）を管理し、再生オプション（音量やループ再生など）にアクセスするために、Objective-Cの宣言されたプロパティ機能を使用します。たとえば、オーディオプレーヤーの再生音量を設定するには次のようにします。

```
[self.player setVolume:1.0];    // 使用可能な範囲は0.0～1.0
```

AVAudioPlayerクラスの詳細については、『*AVAudioPlayer Class Reference*』を参照してください。

Audio Queue Servicesを使用した制御を伴うサウンドの再生

Audio Queue Servicesは、AVAudioPlayerで利用可能な再生機能に、さらにいくつかの再生機能を追加します。再生にAudio Queue Servicesを使用すると、以下のことができます。

- サウンドの再生を正確にスケジューリングし、同期を可能にする
- 音量を正確に制御する（バッファ単位で）
- Audio File Stream Servicesを使用してストリームからキャプチャしたオーディオを再生する

Audio Queue Servicesを利用すると、iPhone OSで利用可能な任意のオーディオフォーマット（「オーディオの再生フォーマットと録音フォーマット」（145 ページ）を参照）のサウンドを再生できます。「オーディオの録音」（155 ページ）で説明するように、録音にもこのテクノロジーを使用します。

このテクノロジーの使用の詳細については、『*Audio Queue Services Programming Guide*』および『*Audio Queue Services Reference*』を参照してください。サンプルコードについては、iPhone Dev Centerの*SpeakHere*を参照してください（Mac OS Xでの実装については、Core Audio SDKで入手できるAudioQueueToolsプロジェクトを参照してください。Mac OS XにXcodeツールをインストールした場合は、AudioQueueToolsプロジェクトは/Developer/Examples/CoreAudio/SimpleSDK/AudioQueueToolsから利用できます）。

オーディオキューオブジェクトの作成

再生のためのオーディオキューオブジェクトを作成するには、次の3つの手順を実行します。

1. 再生するデータのオーディオフォーマットなど、オーディオキューで必要な情報を管理するデータ構造を作成します。
2. オーディオキューバッファを管理するコールバック関数を定義します。このコールバック関数は、Audio File Servicesを使用して再生するファイルを読み込みます（iPhone OS 2.1以降では、Extended Audio File Servicesを使用してファイルを読み込むこともできます）。
3. `AudioQueueNewOutput`関数を使用して、再生オーディオキューをインスタンス化します。

リスト 7-7に、ANSI Cを使用した場合のこれらの手順を示します。*SpeakHere*サンプルプロジェクトは、Objective-Cプログラムのコンテキストでの同じ手順を示しています。

リスト 7-7　オーディオキューオブジェクトの作成

```
static const int kNumberBuffers = 3;
// オーディオキューで必要な情報を管理するためのデータ構造を作成する
struct myAQStruct {
    AudioFileID                  mAudioFile;
    CAStreamBasicDescription     mDataFormat;
    AudioQueueRef                mQueue;
    AudioQueueBufferRef          mBuffers[kNumberBuffers];
    SInt64                       mCurrentPacket;
    UInt32                       mNumPacketsToRead;
    AudioStreamPacketDescription *mPacketDescs;
    bool                         mDone;
};
// 再生オーディオキューのコールバック関数を定義する
static void AQTestBufferCallback(
    void                    *inUserData,
    AudioQueueRef           inAQ,
    AudioQueueBufferRef     inCompleteAQBuffer
) {
    myAQStruct *myInfo = (myAQStruct *)inUserData;
    if (myInfo->mDone) return;
    UInt32 numBytes;
    UInt32 nPackets = myInfo->mNumPacketsToRead;

    AudioFileReadPackets (
        myInfo->mAudioFile,
        false,
        &numBytes,
        myInfo->mPacketDescs,
        myInfo->mCurrentPacket,
        &nPackets,
        inCompleteAQBuffer->mAudioData
    );
    if (nPackets > 0) {
        inCompleteAQBuffer->mAudioDataByteSize = numBytes;
        AudioQueueEnqueueBuffer (
            inAQ,
            inCompleteAQBuffer,
            (myInfo->mPacketDescs ? nPackets :0),
```

```
            myInfo->mPacketDescs
        );
        myInfo->mCurrentPacket += nPackets;
    } else {
        AudioQueueStop (
            myInfo->mQueue,
            false
        );
        myInfo->mDone = true;
    }
}
// オーディオキューオブジェクトをインスタンス化する
AudioQueueNewOutput (
    &myInfo.mDataFormat,
    AQTestBufferCallback,
    &myInfo,
    CFRunLoopGetCurrent(),
    kCFRunLoopCommonModes,
    0,
    &myInfo.mQueue
);
```

再生レベルの制御

オーディオキューオブジェクトには、再生レベルを制御するための方法が2つあります。

再生レベルを直接設定するには、`AudioQueueSetParameter`関数に`kAudioQueueParam_Volume`パラメータを渡します。その例をリスト 7-8に示します。レベルの変更はすぐに有効になります。

リスト 7-8　再生レベルの直接設定

```
Float32 volume = 1;    // リニアスケール、範囲は0.0から1.0
AudioQueueSetParameter (
    myAQstruct.audioQueueObject,
    kAudioQueueParam_Volume,
    volume
);
```

`AudioQueueEnqueueBufferWithParameters`関数を使用して、オーディオキューバッファの再生レベルを設定することもできます。これを利用すると、実際にオーディオキューバッファが保持するオーディオキューの設定を、エンキュー時に割り当てることができます。このような変更は、オーディオキューバッファが再生を開始するときに有効になります。

どちらの場合も、オーディオキューのレベル変更は、再びそれを変更するまで有効です。

再生レベルの表示

現在の再生レベルは、以下の手順を実行してオーディオキューオブジェクトから取得できます。

1. `kAudioQueueProperty_EnableLevelMetering`プロパティを`true`に設定して、オーディオキューオブジェクトの測定を有効にする
2. オーディオキューオブジェクトの`kAudioQueueProperty_CurrentLevelMeter`プロパティを問い合わせる

このプロパティの値はAudioQueueLevelMeterState構造体の配列であり、チャネルごとに1つあります。リスト 7-9に、この構造体を示します。

リスト 7-9 AudioQueueLevelMeterState構造体

```
typedef struct AudioQueueLevelMeterState {
    Float32     mAveragePower;
    Float32     mPeakPower;
};  AudioQueueLevelMeterState;
```

複数のサウンドの同時再生

複数のサウンドを同時に再生するには、サウンドごとに再生オーディオキューオブジェクトを1つずつ作成します。各オーディオキューに対して、AudioQueueEnqueueBufferWithParameters関数を使用して、オーディオの最初のバッファが同時に開始するように指定します。

iPhone OSベースのデバイスで複数のサウンドを同時に再生する場合、オーディオフォーマットが非常に重要になります。複数のサウンドを同時に再生するには、リニアPCM（圧縮なし）オーディオフォーマット、または特定の圧縮ありのオーディオフォーマット（「オーディオの再生フォーマットと録音フォーマット」（145 ページ）を参照）を使用します。

OpenALを使用したポジショニングを伴うサウンドの再生

iPhone OSのOpenALフレームワークで利用可能な、オープンソースのOpenALオーディオAPIは、再生中にステレオ領域でのサウンドのポジショニングを行うために最適化されたインターフェイスを提供します。OpenALを使用すると、サウンドの再生、ポジショニング、および移動は、ほかのプラットフォームで実行する場合と同様に簡単です。OpenALを使用すると、サウンドのミキシングもできます。OpenALは、再生にCore AudioのI/Oを使用するため、遅延が最小になります。

これらの理由から、iPhone OSベースのデバイス上のゲームアプリケーションでサウンドエフェクトを再生する場合は、OpenALが最適な選択肢といえます。また、OpenALは一般的なiPhone OSアプリケーションのオーディオ再生ニーズにも適しています。

iPhone OSでサポートされているOpenAL 1.1は、Core Audioの上に構築されています。詳細については、『*OpenAL FAQ for iPhone OS*』を参照してください。OpenALのドキュメントについては、OpenALのWebサイト(http://openal.org)を参照してください。OpenALオーディオの再生方法のサンプルコードは、*oalTouch*を参照してください。

オーディオの録音

Core Audioは、iPhone OSでのAVAudioRecorderクラスとAudio Queue Servicesを使用したオーディオの録音をサポートします。これらのインターフェイスは、オーディオハードウェアへの接続、メモリの管理、および必要に応じたコーデックの採用を行います。「オーディオの再生フォーマットと録音フォーマット」（145 ページ）に示した任意のフォーマットでオーディオを録音できます。

このセクションでは、AVAudioRecorderクラスとAudio Queue Servicesを使用したiPhone OSでのサウンドの録音について紹介します。

AVAudioRecorderクラスを利用した録音

iPhone OSでサウンドを録音するための最も簡単な方法は、AVAudioRecorderクラス（『*AVAudioRecorder Class Reference*』を参照）を利用することです。このクラスは、非常に簡素化されたObjective-Cインターフェイスで、これを利用すると、録音の一時停止／再開やオーディオ割り込みの処理などの高度な機能を簡単に提供できます。同時に、録音フォーマットを完全に制御することができます。

録音するには、サウンドファイルのURLを渡して、オーディオセッションをセットアップし、録音オブジェクトを構成します。いくつかのセットアップ処理を実行するにはアプリケーションの起動時が適しています。そのコードをリスト 7-10に示します。soundFileURL、recordingなどの定数は、このクラスのインターフェイスで宣言されています。

リスト 7-10　オーディオセッションとサウンドファイルURLのセットアップ

```
- (void) viewDidLoad {

    [super viewDidLoad];

    NSString *tempDir = NSTemporaryDirectory ();
    NSString *soundFilePath = [tempDir stringByAppendingString:@"sound.caf"];

    NSURL *newURL = [[NSURL alloc] initFileURLWithPath:soundFilePath];
    self.soundFileURL = newURL;
    [newURL release];

    AVAudioSession *audioSession = [AVAudioSession sharedInstance];
    audioSession.delegate = self;
    [audioSession setActive:YES error:nil];

    recording = NO;
    playing = NO;
}
```

AVAudioSessionDelegate、AVAudioRecorderDelegate、AVAudioPlayerDelegate（再生もサポートする場合）というプロトコル名をインターフェイス宣言に追加することもできます。

次に、リスト 7-11に示すような録音メソッドを実装します。

リスト 7-11　AVAudioRecorderクラスを利用した録音／停止メソッド

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

    if (recording) {

        [soundRecorder stop];
        recording = NO;
        self.soundRecorder = nil;

        [recordOrStopButton setTitle:@"Record" forState:UIControlStateNormal];
        [recordOrStopButton setTitle:@"Record"
forState:UIControlStateHighlighted];

        [[AVAudioSession sharedInstance] setActive:NO error:nil];

    } else {
```

```
        [[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryRecord
 error:nil];

        NSDictionary *recordSettings =
            [[NSDictionary alloc] initWithObjectsAndKeys:
                [NSNumber numberWithFloat:44100.0],
AVSampleRateKey,
              [NSNumber numberWithInt:kAudioFormatAppleLossless], AVFormatIDKey,
                [NSNumber numberWithInt:1],
AVNumberOfChannelsKey,
                [NSNumber numberWithInt:AVAudioQualityMax],
AVEncoderAudioQualityKey,
            nil];

        AVAudioRecorder *newRecorder = [[AVAudioRecorder alloc]
initWithURL:soundFileURL

settings:recordSettings
                                                                    error:nil];
        [recordSettings release];
        self.soundRecorder = newRecorder;
        [newRecorder release];

        soundRecorder.delegate = self;
        [soundRecorder prepareToRecord];
        [soundRecorder record];
        [recordOrStopButton setTitle:@"Stop" forState:UIControlStateNormal];
       [recordOrStopButton setTitle:@"Stop" forState:UIControlStateHighlighted];

        recording = YES;
    }
}
```

`AVAudioRecorder`クラスの詳細については、『*AVAudioRecorder Class Reference*』を参照してください。

Audio Queue Servicesを利用した録音

Audio Queue Servicesを使用してオーディオを録音するには、アプリケーションでオーディオセッションを設定して、録音用のオーディオキューオブジェクトをインスタンス化し、コールバック関数を用意します。このコールバックではオーディオデータを、すぐに使用できるようにメモリに格納したり、長期保存するためにファイルに書き込んだりします。

iPhone OSでは、録音はシステムで定義されたレベルで行われます。システムは、ユーザが選択したオーディオソース、すなわち内蔵マイク、または（接続されている場合は）ヘッドセットのマイク、その他の入力ソースから録音します。

再生の場合と同様に、現在の録音レベルは、オーディオキューオブジェクトの`kAudioQueueProperty_CurrentLevelMeter`プロパティを問い合わせて取得できます（「再生レベルの表示」（154 ページ）を参照）。

Audio Queue Servicesを使用してオーディオを録音する方法の詳細な例については、『*Audio Queue Services Programming Guide*』の「Recording Audio」を参照してください。サンプルコードについては、iPhone Dev Centerの*SpeakHere*を参照してください。

ストリーミングされたオーディオの解析

ネットワーク接続からなど、ストリーミングされたオーディオコンテンツを再生するには、Audio Queue Servicesと一緒にAudio File Stream Servicesを使用します。Audio File Stream Servicesは、ネットワークのビットストリームの一般的なオーディオファイルコンテナフォーマットからのオーディオパケットとメタデータを解析します。ディスク上のファイルのパケットとメタデータを解析することもできます。

iPhone OSでは、Mac OS Xと同様に、次のオーディオファイルフォーマットとビットストリームフォーマットを解析できます。

- MPEG-1 Audio Layer 3。.mp3ファイルに使用する
- MPEG-2 ADTS。.aacオーディオデータフォーマットに使用する
- AIFC
- AIFF
- CAF
- MPEG-4。.m4a、.mp4、および.3gpファイルに使用する
- NeXT
- WAVE

オーディオパケットを取得したら、iPhone OSでサポートされている任意のフォーマット（「オーディオの再生フォーマットと録音フォーマット」（145 ページ）を参照）で再生できます。

最善のパフォーマンスを実現するには、ネットワークストリーミングアプリケーションではWi-Fi接続からのデータのみを使用するべきです。iPhone OSでは、System Configurationフレームワークと`SCNetworkReachability.h`のインターフェイスを利用して、どのネットワークが到達可能で利用可能かどうかを判断できます。サンプルコードについては、iPhone Dev Centerの*Reachability*を参照してください

ネットワークストリームに接続するには、`CFHTTPMesaage`インターフェイス（『*CFHTTPMessage Reference*』を参照）などの、iPhone OSのCore Foundationのインターフェイスを使用できます。Audio File Stream Servicesを使用して、ネットワークパケットを解析し、オーディオパケットを復元します。次に、そのオーディオパケットをバッファリングして、再生オーディオキューオブジェクトに送ります。

Audio File Stream Servicesは、`AudioFramePacketTranslation`構造体、`AudioFilePacketTableInfo`構造体などのAudio File Servicesのインターフェイスに依存します。これらについては、『*Audio File Services Reference*』を参照してください。

ストリームの使用の詳細については、『*Audio File Stream Services Reference*』を参照してください。サンプルコードについては、*<Xcode>*`/Examples/CoreAudio/Services/`ディレクトリにあるAudioFileStreamサンプルプロジェクトを参照してください。*<Xcode>*は、デベロッパツールディレクトリへのパスです。

iPhone OSでのオーディオユニットのサポート

iPhone OSは、任意のアプリケーションで使用可能なオーディオプラグインセット（*オーディオユニット*と呼ばれる）を提供しています。Audio Unitフレームワークのインターフェイスを利用すると、これらのオーディオユニットをオープン、接続、使用することができます。カスタムオーディオユニットを定義して、それをアプリケーション内で使用することもできます。カスタムオーディオユニットのコードはアプリケーションに静的にリンクする必要があります。そのため、自分で作成したオーディオユニットをiPhone OSのほかのアプリケーションで使用することはできません。

表 7-3に、iPhone OSで提供されているオーディオユニットを示します。

表 7-3　システムが提供するオーディオユニット

オーディオユニット	説明
Converterユニット	Converterユニット（`kAudioUnitSubType_AUConverter`型）を利用すると、オーディオデータのフォーマットを変換できます。
iPod Equalizerユニット	iPod EQユニット（`kAudioUnitSubType_AUiPodEQ`型）は、アプリケーションで使用できる簡単な、プリセットベースのイコライザを提供します。
3D Mixerユニット	3D Mixerユニット（`kAudioUnitSubType_AU3DMixerEmbedded`型）を利用すると、複数のオーディオストリームをミキシングしたり、ステレオ出力パンを指定したり、サンプルレートを操作できます。
Multichannel Mixerユニット	Multichannel Mixerユニット（`kAudioUnitSubType_MultiChannelMixer`型）を利用すると、複数のオーディオストリームを1つのストリームにミキシングできます。
Generic Outputユニット	Generic Outputユニット（`kAudioUnitSubType_GenericOutput`型）は、リニアPCMフォーマットとの間の変換をサポートし、グラフの開始と停止に使用できます。
I/Oユニット	IOユニット（`kAudioUnitSubType_RemoteIO`型）を利用すると、オーディオの入出力ハードウェアに接続でき、リアルタイムI/Oをサポートします。このオーディオユニットの使用方法のサンプルコードは、*aurioTouch*を参照してください。
Voice Processing I/Oユニット	Voice Processing I/Oユニット（`kAudioUnitSubType_VoiceProcessingIO`型）は、I/Oユニットと同じ特徴を持ち、双方向通信にエコー抑制機能を追加します。

システムオーディオユニットの使用の詳細については、『*System Audio Unit Access Guide*』を参照してください。

iPhoneオーディオのベストプラクティス

オーディオ操作のヒント

表 7-4に、iPhone OSでオーディオコンテンツを操作する際に覚えておくと役立つ基本的なヒントを示します。

表 7-4 オーディオのヒント

ヒント	アクション
圧縮オーディオは適切に使用する	AAC、MP3、およびALAC (Apple Lossless)のオーディオは、ハードウェアでデコードされるため効率的ですが、一度に再生できるオーディオストリームは1つに制限されます。複数のサウンドを同時に再生する必要がある場合は、IMA4フォーマット（圧縮あり）またはリニアPCMフォーマット（圧縮なし）を使用してサウンドを保存します。
必要なデータフォーマットおよびファイル形式に変換する	Mac OS Xの`afconvert`ツールを利用すると、幅広いオーディオデータフォーマットおよびファイル形式に変換できます。「iPhone OSの推奨オーディオフォーマット」（161 ページ）および`afconvert`のmanページを参照してください。
オーディオメモリの問題を評価する	Audio Queue Servicesを使用してサウンドを再生している場合は、短いセグメントのオーディオデータをオーディオキューバッファに送信するコールバックを記述します。ディスクアクセスを最小にできるため、サウンドファイル全体をメモリにロードして再生するのが最適な場合もあります。また、バッファが一杯になる分だけのデータをロードするのが最適な場合もあります。アプリケーションにとってどちらの方法が最適か、テストして評価してください。
サンプルレートとビット深度を制限して、オーディオファイルのサイズを小さくする	サンプルレートと、サンプルごとのビット数は、圧縮されていないオーディオのサイズに直接影響します。このようなサウンドをたくさん再生する必要がある場合は、これらの値を下げて、オーディオデータのメモリ占有量を削減することを検討します。たとえば、サウンドエフェクトに44.1 kHzのサンプリングレートではなく、32 kHz（または、それ以下）のサンプリングレートを使用しても、妥当な品質が得られます。
適切なテクノロジーを採用する	警告やユーザインターフェイスのサウンドエフェクトを再生するには、Core AudioのSystem Sound Servicesを使用します。ステレオ領域でサウンドをポジショニングするために便利で高度なインターフェイスがほしい場合や、遅延の少ない再生が必要な場合は、OpenALを使用します。ファイルやネットワークストリームからのオーディオパケットを解析するには、Audio File Stream Servicesを使用します。1つ以上のサウンドを簡単に再生するには、`AVAudioPlayer`クラスを使用します。その他のオーディオアプリケーション（ストリームオーディオの再生、オーディオの録音を含む）には、Audio Queue Servicesを使用します。
遅延の少ないコードにする	再生時の遅延を最小にするには、OpenALを使用するか、I/Oユニットを直接使用します。

iPhone OSの推奨オーディオフォーマット

圧縮されていない（高品質の）オーディオには、CAFファイルにパッケージされる16ビット、リトルエンディアン、リニアPCMオーディオデータを使用します。Mac OS Xで`afconvert`コマンドラインツールを次のように使用すると、オーディオファイルをこのフォーマットに変換できます。

```
/usr/bin/afconvert -f caff -d LEI16 {INPUT} {OUTPUT}
```

`afconvert`ツールを利用すると、幅広いオーディオデータフォーマットおよびファイル形式に変換できます。詳細については、`afconvert`のmanページを参照するか、シェルプロンプトで`afconvert -h`と入力してください。

圧縮されたオーディオを一度に1つ再生する場合、およびiPodアプリケーションと同時にオーディオを再生する必要がない場合は、CAFファイルまたはm4aファイルにパッケージされるAACフォーマットを使用します。

複数のサウンドを同時に再生するときのメモリ使用を少なくするには、IMA4 (IMA/ADPCM)圧縮を使用します。これによってファイルサイズは削減されますが、展開中はわずかにCPU負荷がかかります。リニアPCMデータと同様に、IMA4データはCAFファイルにパッケージします。

iPhone OSでのビデオの使用

ビデオの録画

iPhone OS 3.0から、デバイスがサポートしていればビデオ（オーディオを含む）を録画できるようになりました。ビデオ録画インターフェイスを表示するには、スチルカメラインターフェイスを表示する場合と同様に`UIImagePickerController`オブジェクトを作成してプッシュします。

ビデオを録画するには、まずカメラソース型(`UIImagePickerControllerSourceTypeCamera`)が利用可能であることと、ムービーメディア型(`kUTTypeMovie`)が利用可能であることを確認します。`mediaTypes`プロパティに割り当てたメディア型によって、ピッカーがスチルカメラまたはビデオカメラを直接表示するか、それをユーザが選べるようにする選択インターフェイスを表示するかが決まります。

`UIImagePickerControllerDelegate`プロトコルを使用して、画像ピッカーのデリゲートを登録します。デリゲートオブジェクトは、`imagePickerController:didFinishPickingMediaWithInfo:`メソッドを利用して録画された完全なビデオを受け取ります。

デバイスがサポートしていれば、ユーザのフォトライブラリから以前録画したビデオを選択することもできます。

画像ピッカークラスの詳細については、『*UIImagePickerController Class Reference*』を参照してください。

ビデオファイルの再生

iPhoneでは、Media Playerフレームワーク(`MediaPlayer.framework`)を使用して、アプリケーションから直接ビデオファイルを再生する機能をサポートしています。ビデオの再生は、フルスクリーンモードでのみサポートされ、ゲームデベロッパがカットシーンアニメーションを再生したり、その

他のデベロッパがメディアファイルを再生したりするために利用できます。アプリケーションからビデオを開始すると、メディアプレーヤーインターフェイスが処理を引き継ぎ、画面を徐々に暗くした後に、ビデオコンテンツを徐々に表示します。再生を調整するためのユーザコントロール付きのビデオ再生や、コントロールなしの再生ができます。（図 7-2に示すように）コントロールの一部または全部を有効にすると、ユーザは音量を変更したり、再生ポイントを変更したり、ビデオを開始したり停止したりすることができます。これらのコントロールをすべて無効にすると、ビデオは最後まで再生されます。

図 7-2　転送コントロール付きのメディアプレーヤーインターフェイス

ビデオの再生を開始するには、再生するファイルのURLを知っている必要があります。アプリケーションが提供するファイルの場合は、通常、これはアプリケーションバンドル内のファイルへのポインタです。ただし、リモートサーバ上のファイルへのポインタでもかまいません。このURLを使用して、`MPMoviePlayerController`クラスの新しいインスタンスをインスタンス化します。このクラスは、ビデオファイルの再生を統轄し、（転送コントロールが表示されている場合は）転送コントロールでのユーザタップなどのユーザとのやり取りを管理します。再生を開始するには、単純にこのコントローラの`play`メソッドを呼び出します。

リスト 7-12に、指定したURLのビデオを再生するメソッドの例を示します。この再生メソッドは、ムービーの再生中に呼び出し側に制御を戻す非同期呼び出しです。ムービーコントローラは、フルスクリーンビューにムービーをロードし、アプリケーションの既存のコンテンツの前面でムービーを再生します。再生が終了すると、ムービーコントローラはオブジェクトに通知を送ります。通知を受け取ったオブジェクトは、不要になったムービーコントローラを解放します。

リスト 7-12　フルスクリーンムービーの再生

```
-(void)playMovieAtURL:(NSURL*)theURL
{
    MPMoviePlayerController* theMovie = [[MPMoviePlayerController alloc]
initWithContentURL:theURL];

    theMovie.scalingMode = MPMovieScalingModeAspectFill;
    theMovie.movieControlMode = MPMovieControlModeHidden;

    // 再生終了通知のための登録
    [[NSNotificationCenter defaultCenter] addObserver:self
                selector:@selector(myMovieFinishedCallback:)
                name:MPMoviePlayerPlaybackDidFinishNotification
                object:theMovie];

    // ムービーの再生は非同期なので、このメソッドはすぐに戻る
    [theMovie play];
}
```

```
// ムービーが終了したら、コントローラを解放する
-(void)myMovieFinishedCallback:(NSNotification*)aNotification
{
    MPMoviePlayerController* theMovie = [aNotification object];

    [[NSNotificationCenter defaultCenter] removeObserver:self
                name:MPMoviePlayerPlaybackDidFinishNotification
                object:theMovie];

    // playMovieAtURLで作成されたムービーインスタンスを解放する
    [theMovie release];
}
```

Media Playerフレームワークのクラスの詳細については、『*Media Player Framework Reference*』を参照してください。サポートされているビデオフォーマットのリストは、『*iPhone OS Technology Overview*』を参照してください。

デバイスサポート

iPhone OSは、モバイルコンピューティング体験を魅力的にするさまざまな機能をサポートしています。iPhone OSを通じて、アプリケーションは、加速度センサーやカメラなどのハードウェア機能や、ユーザのフォトライブラリなどのソフトウェア機能にアクセスできます。以降の各セクションでは、これらの機能について説明しそれらをアプリケーションに組み込む方法を示します。

利用可能なハードウェアサポートの識別

iPhone OS用に設計されたアプリケーションは、さまざまなハードウェア機能を持つデバイス上で実行できなければなりません。加速度センサーやWi-Fiネットワーク機能はすべてのデバイスで利用できますが、デバイスによってはカメラやGPSハードウェアが付いていないものもあります。アプリケーションにこうした機能が必要な場合は、アプリケーションの購入前にユーザにその事実を必ず伝える必要があります。必須ではなくても、あればサポートする機能については、使用前にその機能があるかを確認する必要があります。

重要： アプリケーションの実行に不可欠な機能の場合は、アプリケーションの`Info.plist`ファイルで`UIRequiredDeviceCapabilities`キーを適切に設定します。このキーは、デバイスにない機能を必要とするアプリケーションをユーザがインストールできないようにします。ただし、当該機能の有無に関係なくアプリケーションが機能する場合はキーを含めるべきではありません。このキーの詳細については、「情報プロパティリスト」（28 ページ）を参照してください。

表 8-1に、特定の種類のハードウェアが利用可能かどうかを確認する方法を示します。その機能がなくてもアプリケーションは動作し、あればそれを使用する場合には、この表に示す方法を使用する必要があります。

表 8-1 利用可能なハードウェア機能の識別

機能	方法
ネットワークが利用可能かどうかを確認するには	Software Configurationフレームワークの到達性(Reachability)のためのインターフェイスを使用して、現在のネットワーク接続を調べます。Software Configurationフレームワークの使いかたの例については、*Reachability*を参照してください。
スチルカメラが利用可能かどうかを確認するには	`UIImagePickerController`クラスの`isSourceTypeAvailable:`メソッドを使用して、カメラが利用可能かどうかを判断します。詳細については、「カメラによる写真の撮影」（185 ページ）を参照してください。

機能	方法
オーディオ入力（マイクロフォン）が利用可能かどうかを確認するには	iPhone OS 3.0以降では、`AVAudioSession`クラスを使用してオーディオ入力が利用可能かどうかを確認します。このクラスは、iPhone OSベースのデバイス上のさまざまなオーディオ入力ソース（内蔵マイク、ヘッドセットジャック、接続されているアクセサリなど）に対応しています。詳細については、『*AVAudioSession Class Reference*』を参照してください。
GPSハードウェアが存在するかどうかを確認するには	位置情報の更新をアプリケーションに送信するように`CLLocationManager`オブジェクトを設定している場合は、高い精度レベルを指定します。Core Locationフレームワークは特定のハードウェアが利用可能かどうかについての直接的な情報を提供しませんが、代わりに精度値を使用して必要なデータを提供します。以降のロケーション関連イベントでレポートされた精度が不十分な場合は、それをユーザに知らせることができます。詳細については、「ユーザの現在位置の取得」（173 ページ）を参照してください。
特定のアクセサリが利用可能かどうかを確認するには	External Accessoryフレームワークのクラスを使用して適切なアクセサリオブジェクトを検索してそれに接続します。詳細については、「外部アクセサリとの通信」（166 ページ）を参照してください。

外部アクセサリとの通信

iPhone OS 3.0以降では、External Accessoryフレームワーク(`ExternalAccessory.framework`)が、iPhoneまたはiPod touchデバイスに接続されているアクセサリとの通信のためのコンジットを提供します。アプリケーションのデベロッパはこのコンジットを使用して、アクセサリレベルの機能をアプリケーションに組み込むことができます。

注： 以降の各セクションでは、iPhoneアプリケーションからアクセサリに接続する方法を示します。iPhoneまたはiPod touch用のアクセサリのデベロッパになりたい場合は、http://developer.apple.com/jpでその方法についての詳細を参照できます。

External Accessoryフレームワークの機能を使用するには、`ExternalAccessory.framework`をXcodeプロジェクトに追加し、関連するターゲットでそれをリンクする必要があります。このフレームワークのクラスとヘッダにアクセスするには、関連するソースファイルの先頭に`#import <ExternalAccessory/ExternalAccessory.h>`ステートメントを含めます。プロジェクトにフレームワークを追加する方法の詳細については、『*Xcode Project Management Guide*』の「Files in Projects」を参照してください。External Accessoryフレームワークのクラスの全般的な情報については、『*External Accessory Framework Reference*』を参照してください。

アクセサリの基礎

外部アクセサリと通信するには、アクセサリのメーカーと密接に連携してそのアクセサリが提供するサービスについて理解する必要があります。メーカーは、iPhone OSと通信するためにアクセサリハードウェアに明確なサポートを構築する必要があります。このサポートの一部として、アクセサリは少なくとも1つのコマンド**プロトコル**（アクセサリと接続されたアプリケーションとの間でデー

タのやり取りを行うためのカスタムスキーム）をサポートする必要があります。Appleではプロトコルの登録を管理していません。どのプロトコルをサポートするか、またカスタムプロトコルを使用するか、ほかのメーカーがサポートする標準プロトコルを使用するかを決定するのはメーカー次第です。

デベロッパは、アクセサリのメーカーとのやり取りの一部として、対象アクセサリがサポートするプロトコル情報を得る必要があります。名前空間の衝突を避けるには、プロトコル名を`com.apple.myProtocol`のような逆DNS形式の文字列として指定します。これによって各メーカーは、アクセサリ製品系列をサポートするために必要な数だけのプロトコルを定義できます。

アプリケーションは、特定のプロトコルを使用してそのアクセサリへのセッションをオープンし、アクセサリとの通信を行います。セッションをオープンするには`EASession`クラスのインスタンスを作成します。このクラスには、アクセサリと通信するための`NSInputStream`オブジェクトと`NSOutputStream`オブジェクトが含まれています。アプリケーションはこれらのストリームオブジェクトを使用して、未加工のデータパケットをアクセサリに送信したりその返信として同様のパケットを受信したりします。このためデベロッパは、サポートするプロトコルに基づき、各データパケットの有効な形式を理解している必要があります。

アプリケーションでサポートするプロトコルの宣言

外部アクセサリと通信可能なアプリケーションは、サポート対象のプロトコルを`Info.plist`ファイルで宣言する必要があります。特定のプロトコルのサポートを宣言することで、そのアクセサリが接続されているときはアプリケーションを起動できるということをシステムに知らせます。接続されているアクセサリをサポートするアプリケーションが存在しない場合、App Storeを起動して、アクセサリをサポートするアプリケーションを示すようにシステムを設定することもできます。

アプリケーションがサポートするプロトコルを宣言するには、アプリケーションの`Info.plist`ファイルに`UISupportedExternalAccessoryProtocols`キーを含める必要があります。このキーには、アプリケーションがサポートする通信プロトコルを識別する文字列の配列が含まれています。アプリケーションはこのリストに任意の数のプロトコルを、任意の順番で含めることができます。システムは、アプリケーションがどのプロトコルを選ぶべきかを決めるためにこのリストを使用するのはでありません。アプリケーションがアクセサリと通信できるかを確認するためだけにこのリストを使用します。アクセサリとの通信を開始するときに適切な通信プロトコルを選ぶのはコードの責任です。

実行時のアクセサリへの接続

アクセサリは、システムによって接続されて使用する準備ができるまでは、External Accessoryフレームワークから見えません。アクセサリが見えるようになると、アプリケーションは適切なアクセサリオブジェクトを取得して、そのアクセサリがサポートするプロトコルを1つまたは複数使用してセッションをオープンできます。

`EAAccessoryManager`共有オブジェクトは、アクセサリと通信しようとしているアプリケーション用のメインエントリポイントを提供します。このクラスには、すでに接続されているアクセサリオブジェクトの配列が含まれています。これを列挙することで、アプリケーションがサポートするアクセサリがあるかを確認できます。`EAAccessory`オブジェクト内のほとんどの情報（名前、メーカー、モデル情報など）は、表示することだけを目的としています。アプリケーションがアクセサリに接続できるかを確認するには、そのアクセサリのプロトコルを調べてアプリケーションがサポートするプロトコルかを確認する必要があります。

注： 複数のアクセサリオブジェクトが同じプロトコルをサポートする可能性もあります。その場合、どのアクセサリオブジェクトを使用するかを選ぶのはコードの責任です。

特定のアクセサリオブジェクトに対して、特定のプロトコルに使用できるのは一度に1つのセッションのみです。各EAAccessoryオブジェクトのprotocolStringsプロパティには、サポートされているプロトコルをキーとして持つ辞書が含まれています。すでに使用中のプロトコルを使ってセッションを作成しようとすると、External Accessoryフレームワークはエラーを発生させます。

リスト 8-1に、接続されているアクセサリのリストをチェックし、アプリケーションがサポートする最初のアクセサリを取得するメソッドを示します。ここでは指定されたプロトコル用のセッションを作成し、そのセッションの入力ストリームと出力ストリームを設定します。このメソッドがセッションオブジェクトを返す時点までには、アクセサリに接続されてデータの送受信を始める準備ができています。

リスト 8-1　アクセサリ用の通信セッションの作成

```
- (EASession *)openSessionForProtocol:(NSString *)protocolString
{
    NSArray *accessories = [[EAAccessoryManager sharedAccessoryManager]
                                connectedAccessories];
    EAAccessory *accessory = nil;
    EASession *session = nil;

    for (EAAccessory *obj in accessories)
    {
        if ([[obj protocolStrings] containsObject:protocolString])
        {
            accessory = obj;
            break;
        }
    }

    if (accessory)
    {
        session = [[EASession alloc] initWithAccessory:accessory
                                forProtocol:protocolString];
        if (session)
        {
            [[session inputStream] setDelegate:self];
           [[session inputStream] scheduleInRunLoop:[NSRunLoop currentRunLoop]
                                forMode:NSDefaultRunLoopMode];
            [[session inputStream] open];
            [[session outputStream] setDelegate:self];
          [[session outputStream] scheduleInRunLoop:[NSRunLoop currentRunLoop]
                                forMode:NSDefaultRunLoopMode];
            [[session outputStream] open];
            [session autorelease];
        }
    }

    return session;
}
```

入力ストリームと出力ストリームを設定したら、第一段階としてストリーム関連のデータの処理を行います。リスト 8-2は、デリゲートのストリーム処理コードの基本構成を示しています。このメソッドは、アクセサリの入力ストリームと出力ストリームの両方からのイベントに応答します。アクセサリがアプリケーションにデータを送信すると、読み取り可能なバイトがあることを示すイベントが届きます。同様に、アクセサリ側でアプリケーションからのデータを受信する準備ができると、そのことを示すイベントが届きます（もちろんアプリケーションは、イベントを受信するまで、ストリームへのバイトの書き出しを必ずしも待つ必要はありません。ストリームの`hasBytesAvailable`メソッドを呼び出してアクセサリがまだデータを受信できるかを確認することもできます）。ストリームおよびストリーム関連イベントの処理の詳細については、『*Stream Programming Guide for Cocoa*』を参照してください。

リスト 8-2　ストリームイベントの処理

```
// ストリームからの通信を処理する
- (void)stream:(NSStream*)theStream handleEvent:(NSStreamEvent)streamEvent
{
    switch (streamEvent)
    {
        case NSStreamHasBytesAvailable:
            // 受信ストリームデータを処理する
            break;

        case NSStreamEventHasSpaceAvailable:
            // 待機中の次のコマンドを送信する
            break;

        default:
            break;
    }

}
```

アクセサリ関連イベントの監視

External Accessoryフレームワークは、ハードウェアアクセサリが接続または切断されたときに通知を送る機能を備えています。ただし通知する機能はありますが、自動的に実行されるわけではありません。アプリケーションは、`EAAccessoryManager`クラスの`registerForLocalNotifications`メソッドを呼び出して、通知が生成されるよう明確に要求する必要があります。アクセサリが接続され、認証されて、アプリケーションとやり取りする準備ができると、フレームワークが`EAAccessoryDidConnectNotification`通知を送信します。アクセサリとの接続が切断されると、フレームワークが`EAAccessoryDidDisconnectNotification`通知を送信します。デフォルトの`NSNotificationCenter`を使用してこれらの通知を受信するように登録できます。どちらの通知にも対象のアクセサリについての情報が含まれています。

デフォルトの通知センターを通じて通知を受信するだけでなく、現在アクセサリとやり取りしているアプリケーションはそれに対応する`EAAccessory`オブジェクトにデリゲートを割り当てて変更に関する通知を受けることができます。デリゲートオブジェクトは、`EAAccessoryDelegate`プロトコル（現在のところオプションの`accessoryDidDisconnect:`メソッドを含んでいる）に従わなければなりません。このメソッドを使用すると、最初に通知オブザーバをセットアップしていなくても接続解除通知を受信できます。

通知を受信するための登録方法の詳細については、『*Notification Programming Topics for Cocoa*』を参照してください。

加速度センサーイベントへのアクセス

加速度センサーは、特定の線形パスに沿って速度の変化を時系列に測定します。iPhoneおよびiPod touchには、デバイスの主軸に沿って1つずつ、計3つの加速度センサーがあります。この加速度センサーを組み合わせることによって、任意の方向へのデバイスの動きを検出できます。このデータを使用して、デバイスの瞬間的な動きと、重力に対するデバイスの現在の向きの両方を追跡できます。

注： iPhone OS 3.0以降では、シェイクモーションなどの特定の種類のモーションを検出しようとする場合、加速度センサーインターフェイスの代わりにモーションイベントを使用してそのモーションを追跡することを検討するべきです。モーションイベントは特定の種類の加速度センサーの動きを検出するための一貫性のある手段を提供します。詳細については、「モーションイベント」（95 ページ）を参照してください。

どのアプリケーションも、加速度データを受け取るために使用できる`UIAccelerometer`というシングルトンオブジェクトを持っています。このクラスのインスタンスを取得するには、`UIAccelerometer`の`sharedAccelerometer`クラスメソッドを使用します。このオブジェクトを使用して、レポートの間隔や、加速度イベントを受け取るカスタムデリゲートを設定します。レポートの間隔は、最短で10ミリ秒（100Hzの更新レートに相当）に設定できます。ただし、ほとんどのアプリケーションはもっと長い間隔で十分に動作します。デリゲートオブジェクトを割り当てるとすぐに、加速度センサーはそのデリゲートオブジェクトにデータを送り始めます。その後、デリゲートオブジェクトは指定の更新間隔でデータを受け取ります。

リスト 8-3に、加速度センサーを設定するための基本的な手順を示します。この例では、更新頻度は50Hzです。これは、更新間隔20ミリ秒に相当します。`myDelegateObject`はカスタムオブジェクトです。このオブジェクトは、加速度データを受け取る方法を定義する`UIAccelerometerDelegate`プロトコルをサポートする必要があります。

リスト 8-3　加速度センサーの設定

```
#define kAccelerometerFrequency        50 //Hz
-(void)configureAccelerometer
{
    UIAccelerometer*  theAccelerometer = [UIAccelerometer sharedAccelerometer];
    theAccelerometer.updateInterval = 1 / kAccelerometerFrequency;

    theAccelerometer.delegate = self;
    // デリゲートイベントがすぐに開始する
}
```

共有の加速度センサーは、一定の間隔でイベントデータをデリゲートの`accelerometer:didAccelerate:`メソッドに送付します。その様子をリスト 8-4に示します。このメソッドを使用して、任意の方法で加速度データを処理できます。一般的には、何らかのフィルタを使用して、関心のあるデータ成分を分離する処理をお勧めします。

リスト 8-4　加速度センサーイベントの受け取り

```
- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration
{
    UIAccelerationValue x, y, z;
    x = acceleration.x;
```

```
    y = acceleration.y;
    z = acceleration.z;

    // 受け取った値を使って何か処理をする
}
```

加速度イベントの送付を停止するには、UIAccelerometer共有オブジェクトのデリゲートにnilを設定します。デリゲートオブジェクトにnilを設定すると、システムに、必要に応じて加速度センサーハードウェアをオフにしてバッテリー持続時間を節約できることを知らせます。

デリゲートメソッドで受け取る加速度データは、加速度センサーハードウェアによってレポートされる瞬間的な値を表します。デバイスがたとえ完全に静止していても、ハードウェアからレポートされる値はわずかに変動する可能性があります。これらの値を使用する場合、時間の経過とともに値を平均化したり、受け取ったデータを補正したりして、必ずこの変動を計算に入れる必要があります。たとえば、Bubble Levelサンプルアプリケーションでは、既知の表面に対して現在の角度を補正するコントロールを提供しています。補正後の計測では、補正後の角度に対する相対角度がレポートされます。自分のコードで同じレベルの精度が必要な場合、そのユーザインターフェイスにも何らかの補正手段を含めるべきです。

適切な更新間隔の選択

加速度イベントの更新間隔を設定するとき、アプリケーションのニーズを満たしながら、送付されるイベントの数を最小限に抑える間隔を選択するのが最善です。1秒間に100回も加速度イベントの送付を必要とするアプリケーションはほとんどありません。低い頻度を採用することでアプリケーションが頻繁に動作せずに済むため、バッテリー持続時間を向上させることができます。表8-2に、一般的な更新頻度とその頻度で生成された加速度データを使って何ができるかを示します。

表 8-2　加速度イベントの一般的な更新間隔

イベント頻度(Hz)	使用法
10–20	デバイスの現在の向きを表すベクトルを確認するのに適しています。
30–60	ゲーム、またはリアルタイムユーザ入力用に加速度センサーを使用するその他のアプリケーションに適しています。
70–100	高い頻度でモーションを検出する必要があるアプリケーションに適しています。たとえば、この間隔を使って、ユーザがデバイスをたたいたり高速でゆすったりすることを検出する場合が考えられます。

加速度データからの重力成分の分離

加速度センサーのデータを使用してデバイスの現在の向きを検出する場合は、デバイスの動きによる加速度データ成分から、重力による加速度データ成分を取り除く必要があります。それには、ローパスフィルタを使用して、加速度センサーのデータに対する瞬間的な変化の影響を減らすことができます。フィルタを適用した結果得られた値は、より安定した重力の影響を反映しています。

リスト 8-5に、簡単なローパスフィルタを示します。この例では、低い値のフィルタ係数を使用して、フィルタリングされていない加速度データの10%と、直前にフィルタリングした値の90%を使用した値を生成します。直前の値は、このクラスのメンバ変数であるaccelX、accelY、およびaccelZに格納されています。加速度データは規則的になるので、これらの値はすぐに安定し、瞬間的な動きの変化に対する反応は鈍くなります。

リスト 8-5　加速度センサーのデータからの重力の影響の分離

```
#define kFilteringFactor 0.1

- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration {
    // 基本的なローパスフィルタを使用して、各軸の重力成分だけを保持する
    accelX = (acceleration.x * kFilteringFactor) + (accelX * (1.0 -
kFilteringFactor));
    accelY = (acceleration.y * kFilteringFactor) + (accelY * (1.0 -
kFilteringFactor));
    accelZ = (acceleration.z * kFilteringFactor) + (accelZ * (1.0 -
kFilteringFactor));

   // この加速度データを使用する
}
```

加速度データからの瞬間的な動きの分離

加速度センサーのデータを使用してデバイスの瞬間的な動きだけを検出する場合は、一定である重力の影響から瞬間的な動きの変化を分離する必要があります。それには、ハイパスフィルタを使用します。

リスト 8-6に、簡単なハイパスフィルタの計算を示します。直前のイベントの加速度値は、このクラスのメンバ変数であるaccelX、accelY、およびaccelZに格納されています。この例では、瞬間的な動きの成分だけを取り出すために、ローパスフィルタの値を計算してそれを現在の値から引いています。

リスト 8-6　加速度センサーのデータからの瞬間的な動きの取得

```
#define kFilteringFactor 0.1

- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration {
    // ローパス値を現在の値から引いて、簡単なハイパスフィルタを取得する
    accelX = acceleration.x - ( (acceleration.x * kFilteringFactor) + (accelX
* (1.0 - kFilteringFactor)) );
    accelY = acceleration.y - ( (acceleration.y * kFilteringFactor) + (accelY
* (1.0 - kFilteringFactor)) );
    accelZ = acceleration.z - ( (acceleration.z * kFilteringFactor) + (accelZ
* (1.0 - kFilteringFactor)) );

   // この加速度データを使用する
}
```

現在のデバイスの向きの取得

向きの正確なベクトルではなく、単にデバイスの大まかな向きを知る必要がある場合、UIDeviceクラスのメソッドを使用してその情報を取得します。UIDeviceインターフェイスを使うのがより簡単で、デベロッパ自身が向きのベクトルを計算する必要がありません。

現在の向きを取得する前に、beginGeneratingDeviceOrientationNotificationsメソッドを呼び出すことによって、デバイス向きの通知の生成を開始するようにUIDeviceクラスに指示する必要があります。これを行うことで、加速度センサーハードウェアがオンになります。そうでなければ、節電のためにオフになっている可能性があります。

向きの通知を有効にしたらすぐ、UIDevice共有オブジェクトのorientationプロパティから現在の向きを取得できます。また、大まかな向きの変化が生じたるたびに送信される、UIDeviceOrientationDidChangeNotification通知の受け取りを登録することもできます。デバイスの向きは、デバイスが横長モードか縦長モードか、デバイスの表面が上に向いているか下に向いているかを示す、UIDeviceOrientationの定数を使ってレポートされます。これらの定数は、デバイスの物理的な向きを示し、アプリケーションのユーザインターフェイスの向きに対応する必要はありません。

デバイスの向きを知る必要がなくなったら、必ずUIDeviceのendGeneratingDeviceOrientationNotificationsメソッドを呼び出して、向きの通知を無効にするようにします。これを行うことで、システムはほかで加速度センサーハードウェアを使用していなければ、加速度センサーハードウェアを無効にできます。

ロケーションサービスとヘディングサービスの使用

Core Locationフレームワークは、ユーザの現在位置と方角の検出をサポートしています。このフレームワークは利用可能な内蔵ハードウェアから情報を収集し、アプリケーションに非同期にレポートします。データが利用できるかどうかは、デバイスの種類とそれに必要なハードウェアが現在有効になっているかどうかによって決まります。デバイスが機内モードの場合は、このハードウェアは有効になっていない可能性があります。

Core Locationフレームワークの機能を使用するには、CoreLocation.frameworkをXcodeプロジェクトに追加し、関連するターゲットでそれをリンクする必要があります。このフレームワークのクラスとヘッダにアクセスするには、関連するソースファイルの先頭に#import <CoreLocation/CoreLocation.h>ステートメントを含めます。フレームワークをプロジェクトに追加する方法の詳細については、『*Xcode Project Management Guide*』の「Files in Projects」を参照してください。

Core Locationフレームワークのクラスの全般的な情報については、『*Core Location Framework Reference*』を参照してください。

ユーザの現在位置の取得

Core Locationフレームワークを使用すると、デバイスの現在位置を検出してその情報をアプリケーションで使用できます。このフレームワークは、デバイスの組み込みハードウェアを利用して、利用可能な信号情報から三角法によって位置を特定します。次に、その位置をコードにレポートし、新規または更新された信号を受信するとその位置情報を更新します。

Core Locationフレームワークを使用する場合、使用を控えめにするとともに、ロケーションサービスを適切に設定してください。位置データを収集するためには、内蔵無線の電源をオンにし、利用可能な携帯電話の基地局、Wi-Fiホットスポット、またはGPS衛星を照会する必要があります。これには数秒かかる場合があります。さらに、より正確な位置データを必要とする場合は、長い時間無線をオンにしたままにしなければならない可能性もあります。このハードウェアを長時間オンにしたままにするとデバイスのバッテリーが消費されます。位置情報はそれほど頻繁に変化しないことを考慮すると、最初の位置を検出した後は、定期的に更新情報を取得すれば十分です。定期的な位置の更新が必要な場合は、サービスに最小距離のしきい値を設定することによって、コードで処理しなければならない位置更新の回数を最小にすることもできます。

ユーザの現在位置を取得するには、`CLLocationManager`クラスのインスタンスを作成して、それに望みの精度としきい値パラメータを設定します。位置通知の受け取りを開始するには、そのオブジェクトにデリゲートを割り当て、`startUpdatingLocation`メソッドを呼び出して、ユーザの現在位置の検出を開始します。新しい位置データが利用可能になると、Location Managerはその割り当てられたデリゲートオブジェクトに通知します。位置の更新がすでに送付されている場合は、次のイベントの送付を待たずに、最新の位置データを`CLLocationManager`オブジェクトから直接取得することもできます。

リスト 8-7に、`startUpdates`カスタムメソッドと、`locationManager:didUpdateToLocation:fromLocation:`デリゲートメソッドの実装を示します。`startUpdates`メソッドは、新しいLocation Managerオブジェクトを作成し（まだ作成されていない場合）、それを使用して位置更新の生成を開始します（この例では、`locationManager`変数は、`MyLocationGetter`クラスで宣言されているメンバ変数です。この変数は、`CLLocationManagerDelegate`プロトコルに従います）。このハンドラメソッドは、イベントのタイムスタンプを使用してイベントがどの程度新しいかを判断します。古いイベントの場合は、ハンドラはそれを無視して、より新しいイベントを待ちます。新しいイベントのときは、ロケーションサービスを無効にします。

リスト 8-7　位置更新の初期化と処理

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

@implementation MyLocationGetter
- (void)startUpdates
{
    // このオブジェクトがまだLocation Managerを持っていない場合は、
    // Location Managerを作成する
    if (nil == locationManager)
        locationManager = [[CLLocationManager alloc] init];

    locationManager.delegate = self;
    locationManager.desiredAccuracy = kCLLocationAccuracyKilometer;

    // 新しいイベント用に、移動のしきい値を設定する
    locationManager.distanceFilter = 500;

    [locationManager startUpdatingLocation];
}


// CLLocationManagerDelegateプロトコルのデリゲートメソッド
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
    fromLocation:(CLLocation *)oldLocation
{
    // 比較的新しいイベントの場合は、節電のために更新をオフにする
```

```
    NSDate* eventDate = newLocation.timestamp;
    NSTimeInterval howRecent = [eventDate timeIntervalSinceNow];
    if (abs(howRecent) < 5.0)
    {
        [manager stopUpdatingLocation];

        printf("latitude %+.6f, longitude %+.6f\n",
                newLocation.coordinate.latitude,
                newLocation.coordinate.longitude);
    }
    // それ以外の場合は、このイベントをスキップして次のイベントを処理する
}
@end
```

ロケーションサービスは、しばしば最後にキャッシュした位置イベントを即座に返すので、イベントのタイムスタンプをチェックすることをお勧めします。おおまかな位置を取得するまで数秒を要する場合があります。したがって、古いデータは、最後に取得した位置を表す手段としてのみ役立ちます。イベントを受け取るか判断する手段として精度を使用することもできます。ロケーションサービスは、より正確なデータを受け取ると、それに応じた修正を反映した精度の値を含む追加イベントを返します。

注： Core Locationフレームワークは、ロケーションクエリが戻るときではなく、それぞれのクエリの初めにタイムスタンプの値を記録します。Core Locationは異なる複数の手法を使って測位を取得するため、クエリは、タイムスタンプが示す時間と異なる順番で戻る場合もあり得ます。結果として、場合によっては新しいイベントが、前のイベントよりわずかに古いタイムスタンプになっていても問題ありません。フレームワークは、タイムスタンプの値に関係なく、送付する新しいイベントごとに位置データの精度を向上することに重点を置いています。

方角関連イベントの取得

Core Locationフレームワークは、方角関連の情報を取得する2つの異なる手段をサポートしています。GPSハードウェアを備えたデバイスは、ユーザの緯度と経度の決定に使われるものと同じロケーションイベントを使用して、現在の進行方向についての大体の情報を提供できます。磁力センサーを備えたデバイスは、ヘディング（方角）オブジェクト（`CLHeading`クラスのインスタンス）によって、より正確な方角情報を提供できます。

GPSハードウェアを使用しておおまかなロケーションイベントを取得するためのプロセスは、「ユーザの現在位置の取得」（173ページ）で説明したプロセスと同じです。アプリケーションのデリゲートにレポートされる`CLLocation`オブジェクトには、それに関連する情報を含む`course`プロパティと`speed`プロパティが含まれています。このインターフェイスは時間の経過に伴うユーザの移動を追跡するアプリケーション（自動車用のナビゲーションシステムを実装したアプリケーションなど）のほとんどに適用できます。コンパス（方位磁石）をベースとするアプリケーションや、ユーザが移動していない状態の現在の方角を検出する必要があるアプリケーションの場合は、ロケーションマネージャにヘディングオブジェクトを要請できます。

ヘディングオブジェクトを受け取るには、アプリケーションは磁力センサーを備えたデバイス上で実行していなければなりません。磁力センサーは地球から発生する近辺の磁場を測定し、デバイスの正確な向きを識別します。磁力センサーは局所的な磁場（オーディオスピーカー、モーター、その他の電子機器に含まれる固定磁石から発生する磁気など）の影響を受ける可能性がありますが、Core Locationフレームワークはヘディングオブジェクトが有用なデータを含むことを保証するために、各種の局所的な磁場を除去するように十分に高性能にできています。

注： アプリケーションで進路または方角の情報が必要な場合は、アプリケーションのInfo.plistファイルにUIRequiredDeviceCapabilitiesキーを適切に含める必要があります。このキーにより、アプリケーションの動作に必要な機能を正確に指定できるため、これを使ってGPSや磁力センサーハードウェアの存在を要件として指定することができます。このキーの値の設定方法の詳細については、「情報プロパティリスト」（28 ページ）を参照してください。

ヘディングイベントを受け取るには、リスト 8-8に示すように、CLLocationManagerオブジェクトを作成し、それにデリゲートを割り当ててstartUpdatingHeadingメソッドを呼び出します。ただし、ヘディングイベントを要求する前に必ずロケーションマネージャのheadingAvailableプロパティをチェックして適切なハードウェアが存在することを確認する必要があります。適切なハードウェアがない場合は、ロケーションベースのイベントを使用して進路情報を取得する方法に戻す必要があります。

リスト 8-8　ヘディングイベントの送信を開始する

```
CLLocationManager* locManager = [[CLLocationManager alloc] init];
if (locManager.headingAvailable)
{
   locManager.delegate = myDelegateObject; // カスタムデリゲートオブジェクトを割り当
てる
   locManager.headingFilter = 5;
   [locManager startUpdatingHeading];
}
else
   // ロケーションイベントを代わりに使う
```

デリゲートプロパティに割り当てるオブジェクトは、CLLocationManagerDelegateプロトコルに従わなければなりません。新しいヘディングイベントが届いたら、ロケーションマネージャオブジェクトはlocationManager:didUpdateHeading:メソッドを呼び出してそのイベントをアプリケーションに送信します。新しいイベントを受信するときは、リスト 8-9に示すように、headingAccuracyプロパティをチェックして、今受信したデータが有効であることを確認する必要があります。

リスト 8-9　ヘディングイベントを処理する

```
- (void)locationManager:(CLLocationManager*)manager
didUpdateHeading:(CLHeading*)newHeading
{
   // 精度が有効な場合は、イベント処理に進む
   if (newHeading.headingAccuracy > 0)
   {
      CLLocationDirection theHeading = newHeading.magneticHeading;

      // このイベントデータを使って何かを実行する
   }
}
```

CLHeadingオブジェクトのmagneticHeadingプロパティには、主要な方角データが含まれています。このデータは常に存在します。このプロパティは、磁北を基準とする方角の測定値を提供します。磁北の位置は北極点と同じではありません。北極点（地理学上の北とも呼ばれる）を基準とする方角を取得したい場合は、startUpdatingHeadingを呼び出す前に、startUpdatingLocationを呼び出して位置更新の送信を開始する必要があります。その後で、CLHeadingオブジェクトのtrueHeadingプロパティを使用すると、地理学上の北を基準とする方角を取得できます。

地図と注釈の表示

iPhone OS 3.0で導入されたMap Kitフレームワークを利用すると、完全な機能を備えたマップインターフェイスをアプリケーションウインドウに埋め込むことができます。このフレームワークが提供するマップサポートには、「マップ(Maps)」アプリケーションに一般的に見られる多くの機能が含まれています。標準的なストリートレベルの地図情報、航空写真、またはこれら2つの組み合わせを表示できます。プログラムによって地図を拡大縮小したり、パンすることもできます。またこのフレームワークは、ユーザによる地図の拡大縮小やパンを可能にするタッチイベントを自動的にサポートします。地図にはカスタム情報を注釈として付けることができます。また、このフレームワークの逆ジオコーダ機能を使用して地図座標に対応する住所を検索することもできます。

Map Kitフレームワークの機能を使用するには、`MapKit.framework`をXcodeプロジェクトに追加し、関連するターゲットでそれをリンクする必要があります。このフレームワークのクラスとヘッダにアクセスするには、関連するソースファイルの先頭に`#import <MapKit/MapKit.h>`ステートメントを含めます。フレームワークをプロジェクトに追加する方法の詳細については、『*Xcode Project Management Guide*』の「Files in Projects」を参照してください。Map Kitフレームワークのクラスの全般的な情報については、『*MapKit Framework Reference*』を参照してください。

重要： Map Kitフレームワークは Googleのサービスを使用して地図データを提供します。このフレームワークとそれに関連するインターフェイスを使用する場合は、Google Maps/Google Earth APIの利用規約に従わなければなりません。これらの利用規約は http://code.google.com/apis/maps/iphone/terms.html で参照できます。

ユーザインターフェイスへのMap Viewの追加

アプリケーションに地図を追加するには、`MKMapView`クラスのインスタンスをアプリケーションのビュー階層に埋め込みます。このクラスは、地図情報の表示と、ユーザと地図情報とのやり取りの管理の両方をサポートします。（`initWithFrame:`メソッドを利用して初期化することにより）プログラムからこのクラスのインスタンスを作成できます。または、Interface Builderを使用してインスタンスをnibファイルの1つに追加できます。

Map Viewはビューであるため、その`frame`プロパティを使用してビュー階層内での移動やサイズ変更が自由にできます。Map Viewには固有のコントロールはありませんが、Map Viewの上にツールバーやその他のビューを配置して、地図コンテンツとやり取りをするためのコントロールを付加することができます。Map Viewに追加したサブビューはすべて位置が固定されており、地図コンテンツと一緒にスクロールしません。地図そのものにカスタムコンテンツを追加してそのコンテンツを地図と一緒にスクロールさせたい場合は、注釈を作成しなければなりません（「注釈の表示」（179 ページ）で説明します）。

`MKMapView`クラスには、それを表示する前に設定可能なプロパティがいくつかあります。設定すべき最も重要なプロパティは`region`プロパティです。このプロパティは最初に表示する地図領域を定義します。また、地図コンテンツの拡大縮小やパンの方法を表します。

地図コンテンツの拡大縮小とパン

`MKMapView`クラスの`region`プロパティは、ある時点で表示される地図領域を制御します。地図を拡大縮小したりパンする場合は、このプロパティの値を適切に変更するだけで済みます。このプロパティは、次のような定義を持つ`MKCoordinateRegion`構造体で構成されています。

```
typedef struct {
   CLLocationCoordinate2D center;
   MKCoordinateSpan span;
} MKCoordinateRegion;
```

地図を新しい位置までパンするには、*center*フィールドの値を変更します。拡大縮小するには、*span*フィールドの値を変更します。これらのフィールドの値は地図座標（度、分、秒単位で表す）を使用して指定します。spanフィールドに対して指定する値は、緯度と経度の距離を表します。緯度の距離は1度につき約111キロメートルに固定されていますが、経度の距離は緯度によって変化します。赤道では、経度の値は1度につき約111キロメートルですが、極地では0に近づきます。もちろん、`MKCoordinateRegionMakeWithDistance`関数を使用すれば、いつでも度数の代わりにキロメートル値を使用して領域を作成できます。

変更をアニメーション化せずに地図を更新するには、`region`プロパティまたは`centerCoordinate`プロパティを直接変更します。変更をアニメーション化するには、`setRegion:animated:`メソッドまたは`setCenterCoordinate:animated:`メソッドのいずれかを使用します。`setCenterCoordinate:animated:`メソッドを使用すると、意図しない拡大縮小を生じさせずに地図をパンできます。`setRegion:animated:`メソッドを使用すると、パンと拡大縮小を同時に実行できます。たとえば、現在の地図の幅の半分だけ地図を左にパンするには、次のようなコードを使用して地図の左端の座標を検出し、それを新しい中心点として使用します。

```
CLLocationCoordinate2D mapCenter = myMapView.centerCoordinate;
mapCenter = [myMapView convertPoint:
               CGPointMake(1, (myMapView.frame.size.height/2.0))
               toCoordinateFromView:myMapView];
[myMapView setCenterCoordinate:mapCenter animated:YES];
```

中心の座標を変更せずに拡大または縮小するにはspanを変更します。拡大するにはspanを短くします。縮小するにはspanを長くします。したがって現在のspanが1度の場合、spanを2度にすると2分の1に縮小されます。

```
MKCoordinateRegion theRegion = myMapView.region;

// 縮小する
theRegion.span.longitudeDelta *= 2.0;
theRegion.span.latitudeDelta *= 2.0;
[myMapView setRegion:theRegion animated:YES];
```

ユーザの現在位置の表示

Map Kitフレームワークには、ユーザの現在位置を地図上に表示する機能が組み込まれています。この位置を表示するには、Map Viewオブジェクトの`showsUserLocation`プロパティを`YES`に設定します。これによって、Map ViewはCore Locationフレームワークを使用してユーザの現在位置を検出し、`MKUserLocation`型の注釈を地図に追加します。

`MKUserLocation`注釈オブジェクトが地図に追加されたことは、カスタム注釈が追加されたときと同様にdelegateによってレポートされます。カスタム注釈ビューをユーザの位置に関連付けたい場合は、デリゲートオブジェクトの`mapView:viewForAnnotation:`メソッドからそのビューを返す必要があります。デフォルトの注釈ビューを使用する場合は、このメソッドからは`nil`を返す必要があります。

座標とピクセル間の変換

通常、地図上の点は緯度と経度の値を使用して指定しますが、Map Viewオブジェクト内のピクセルに変換しなければならない場合もあります。たとえば、ユーザが地図サーフェス上で注釈をドラッグできるようにする場合、カスタム注釈ビュー用のイベントハンドラでは、対応する注釈オブジェクトを更新するためにフレーム座標を地図座標に変換する必要があります。`MKMapView`クラスには、地図座標とMap Viewオブジェクトのローカル座標系との間の変換を行うルーチンがいくつか含まれています。これには、次のようなものがあります。

```
convertCoordinate:toPointToView:
convertPoint:toCoordinateFromView:
convertRegion:toRectToView:
convertRect:toRegionFromView:
```

カスタム注釈でのイベント処理の詳細については、「注釈ビューでのイベント処理」（182ページ）を参照してください。

注釈の表示

注釈は、デベロッパが定義して地図そのもの上に配置する地図コンテンツの一部です。Map Kitフレームワークには、注釈オブジェクトとその注釈を表示するビューの2つの部分から構成される注釈が実装されています。たいていは、これらのカスタムオブジェクトを提供するのはデベロッパの責任です。ただし、フレームワークもデベロッパが使用できる標準的な注釈とビューを提供します。

注釈をMap Viewに表示するには、次の2段階の処理が必要になります。

1. 注釈オブジェクトを作成し、それをMap Viewを追加する。
2. デリゲートオブジェクトの`mapView:viewForAnnotation:`メソッドを実装し、そこでそれに対応する注釈ビューを作成する。

注釈オブジェクトは、`MKAnnotation`プロトコルに従う任意のオブジェクトです。通常、注釈オブジェクトは、注釈の座標とその注釈に関連する情報（名前など）が格納された比較的小さいデータオブジェクトです。注釈はプロトコルを使用して定義されるため、アプリケーション内のどのオブジェクトも注釈になり得ます。ただし、実際には注釈オブジェクトは軽量にすべきです。Map Viewは、注釈オブジェクトが明示的に削除されるまでその参照を保持するからです。ただし、これと同じことは注釈ビューには必ずしも当てはまりません。

注釈を画面に表示するときに、注釈オブジェクトがそれに対応する注釈ビューを保持するようにするのはMap Viewの責任です。それには、注釈の座標を画面に表示しようとするときにデリゲートオブジェクトの`mapView:viewForAnnotation:`メソッドを呼び出します。注釈ビューは、それに対応する注釈オブジェクトよりも重量なオブジェクトになる傾向があるため、Map Viewではたくさんの注釈ビューを同時にメモリ内に保持しようとはしません。そのために、Map Viewは注釈ビューのリサイクル方式を実装しています。これはTable Viewがスクロール中にテーブルセルを再利用する仕組みと似ています。注釈ビューが画面から消えるときには、Map Viewはそのビューと注釈オブジェクトとの関連づけを解除してビューを再利用キューに入れます。新しい注釈ビューを作成する前には、デリゲートの`mapView:viewForAnnotation:`メソッドで常にこの再利用キューをチェックし、Map Viewの`dequeueReusableAnnotationViewWithIdentifier:`メソッドを呼び出して既存のビュー

が利用できるかどうかを確認する必要があります。このメソッドが有効なビューオブジェクトを返した場合はそのビューを再度初期化して返します。それ以外の場合は、新しいビューオブジェクトを作成して返すようにします。

注釈オブジェクトの追加と削除

注釈ビューを直接地図に追加することはしません。代わりに、注釈オブジェクト（通常は、ビューではない）を追加します。注釈オブジェクトは、`MKAnnotation`プロトコルに従う、アプリケーション内の任意のオブジェクトです。注釈オブジェクトの最も重要な部分は、`coordinate`プロパティです。これは、`MKAnnotation`プロトコルの必須プロパティで、地図上の注釈にアンカーポイントを提供します。

Map Viewに注釈を追加するには、Map Viewオブジェクトの`addAnnotation:`メソッドまたは`addAnnotations:`メソッドを呼び出すだけです。Map Viewに注釈を追加したり、そのためのユーザインターフェイスを提供する適切なタイミングを決定するのは、デベロッパに任されています。新しい注釈をユーザが作成できるようにするコマンドを備えたツールバーを提供したり、ローカルまたはリモートの情報データベースを使用してプログラムで注釈を作成することもできます。

アプリケーションで古い注釈を破棄する必要がある場合は、注釈を削除する前に`removeAnnotation:`メソッドまたは`removeAnnotations:`メソッドを使用して必ずその注釈を地図から削除する必要があります。Map Viewにはすべての注釈が表示されるため、地図上に表示したくない場合は注釈を明示的に削除する必要があります。たとえばアプリケーションで、レストランや近辺の名所のリストをユーザがフィルタできるようにする場合は、フィルタの基準に適合しない注釈はすべて削除する必要があります。

注釈ビューの定義

Map Kitフレームワークは、`MKAnnotationView`と`MKPinAnnotationView`の2つの注釈ビュークラスを提供します。`MKAnnotationView`クラスは、すべての注釈ビューの基本動作を定義した具象ビューです。`MKPinAnnotationView`クラスは`MKAnnotationView`のサブクラスで、標準のシステムピン画像の1つを、対応する注釈の座標点に表示します。

単純な注釈を表示するには`MKAnnotationView`クラスをそのまま使用し、より対話的な動作を提供するにはこのクラスのサブクラスを定義することができます。このクラスをそのまま使用する場合は、地図上に表示するコンテンツを表すカスタム画像を提供し、それを注釈ビューの`image`プロパティに割り当てます。動的に変化するコンテンツは表示せず、ユーザとのやり取りもサポートしない場合は、このようなクラスの使い方で完全に対応できます。ただし、動的なコンテンツやユーザとのやり取りが必要な場合は、カスタムサブクラスを定義しなければなりません。

カスタムサブクラスでは、2つの方法のうちのいずれかを使用して動的なコンテンツを描画できます。注釈用の画像を表示するために画像プロパティを引き続き使用し、現在の画像を一定間隔で変更できるようにタイマーを設定します。また、ビューの`drawRect:`メソッドをオーバーライドしてコンテンツを明示的に描画することもできます。この場合も、ビューの`setNeedsDisplay`メソッドを定期的に呼び出せるようにタイマーを設定します。

`drawRect:`メソッドを使用してコンテンツを描画する場合は、必ず初期化の直後に注釈ビューのサイズを指定しなければなりません。注釈ビューのデフォルトの初期化メソッドは、パラメータとしてフレーム矩形を取りません。代わりに、`image`プロパティに指定した画像を使用して、後からフレームサイズを設定します。ただし、画像を設定しない場合は、レンダリングされたコンテンツを表示するためにフレームサイズを明示的に設定しなければなりません。

注釈ビューでユーザとのやり取りをサポートする方法の詳細については、「注釈ビューでのイベント処理」（182 ページ）を参照してください。タイマーのセットアップ方法の詳細については、『*Timer Programming Topics for Cocoa*』を参照してください。

注釈ビューの作成

デリゲートオブジェクトの`mapView:viewForAnnotation:`メソッドでは、必ず注釈ビューを作成します。新しいビューを作成する前に、`dequeueReusableAnnotationViewWithIdentifier:`メソッドを呼び出して使用されるのを待機している既存ビューがないかどうかを必ず確認する必要があります。このメソッドが`nil`以外の値を返した場合は、MapViewによって提供された注釈をそのビューの`annotation`プロパティに割り当て、必要な設定を行ってそのビューを既知の状態にしてからそれを返します。このメソッドが`nil`を返した場合は、新しい注釈ビューオブジェクトを作成してそれを返します。

リスト 8-10は、`mapView:viewForAnnotation:`メソッドの実装例を示しています。このメソッドは、カスタム注釈オブジェクトにピン注釈を提供します。既存のピン注釈ビューがすでに存在する場合、このメソッドは注釈オブジェクトをそのビューに関連付けます。再利用キューにビューが存在しない場合、このメソッドは新しいビューを作成してそのビューの基本プロパティを設定し、注釈のコールアウト用のアクセサリビューを設定します。

リスト 8-10　注釈ビューの作成

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView
                      viewForAnnotation:(id <MKAnnotation>)annotation
{
    // これがユーザの位置の場合は、単にnilを返す
    if ([annotation isKindOfClass:[MKUserLocation class]])
        return nil;

    // カスタム注釈を処理する
    if ([annotation isKindOfClass:[CustomPinAnnotation class]])
    {
        // まず、既存のピン注釈ビューをキューから取り出すことを試みる
        MKPinAnnotationView*    pinView = (MKPinAnnotationView*)[mapView
        dequeueReusableAnnotationViewWithIdentifier:@"CustomPinAnnotation"];

        if (!pinView)
        {
            // 既存のピン注釈ビューが利用できない場合は、新しいビューを作成する
          pinView = [[[MKPinAnnotationView alloc] initWithAnnotation:annotation
                       reuseIdentifier:@"CustomPinAnnotation"]
                              autorelease];
            pinView.pinColor = MKPinAnnotationColorRed;
            pinView.animatesDrop = YES;
            pinView.canShowCallout = YES;

            // 詳細ディスクロージャボタンをコールアウトに追加する
            UIButton* rightButton = [UIButton buttonWithType:
                                UIButtonTypeDetailDisclosure];
            [rightButton addTarget:self action:@selector(myShowDetailsMethod:)
                                forControlEvents:UIControlEventTouchUpInside];
            pinView.rightCalloutAccessoryView = rightButton;
        }
        else
            pinView.annotation = annotation;
```

```
        return pinView;
    }

    return nil;
}
```

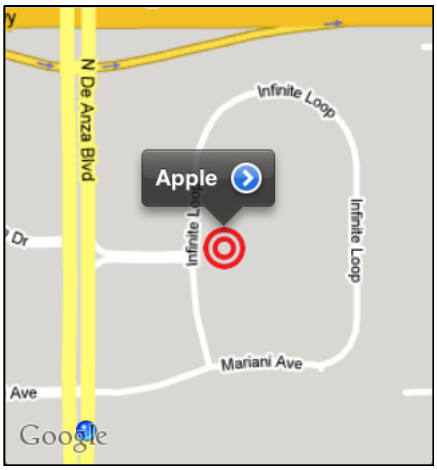
注釈ビューでのイベント処理

注釈ビューは地図コンテンツ上の特殊レイヤにありますが、タッチイベントを受け取ることができる完全なビューです。これらのイベントを使用すると、注釈についてのユーザとのやり取りをより多く提供できます。たとえば、ビュー内でタッチイベントを使用して地図のサーフェイス内でビューをドラッグできます。

注： 地図はスクロールするインターフェイスに表示されるため、一般に、ユーザがカスタムビューをタッチした時点とそれに対応するイベントが配信される時点の間に若干の遅れが生じます。この遅れの間に、基盤のスクロールビューはこのタッチイベントがスクロールジェスチャの一部かを判別できます。

次の一連のコードリストは、ユーザによる移動が可能な注釈ビューの実装方法を示しています。注釈ビューには注釈の座標点の真上に標的画像が表示されます。また、ターゲットについての詳細情報を表示するためのカスタムアクセサリビューが含まれています。図 8-1は、コールアウト用の吹き出しを持つ注釈ビューの例を示しています。

図 8-1 標的注釈ビュー



リスト 8-11は、`BullseyeAnnotationView`クラスの定義を示しています。このクラスには、ビューを正しく移動させるために追跡中に使用するメンバ変数がいくつか追加されています。このクラスにはMap View自身へのポインタも格納されています。この値は、注釈ビューを作成または初期化しなおすときに、`mapView:viewForAnnotation:`メソッド内のコードで設定されます。Map Viewオブジェクトは、イベントの追跡が終わったときに注釈オブジェクトの地図座標を調整するために必要です。

リスト 8-11 BullseyeAnnotationViewクラス

```
@interface BullseyeAnnotationView :MKAnnotationView
{
```

```
    BOOL isMoving;
    CGPoint startLocation;
    CGPoint originalCenter;

    MKMapView* map;
}

@property (assign,nonatomic) MKMapView* map;

- (id)initWithAnnotation:(id <MKAnnotation>)annotation;

@end

@implementation BullseyeAnnotationView
@synthesize map;
- (id)initWithAnnotation:(id <MKAnnotation>)annotation
{
    self = [super initWithAnnotation:annotation
                reuseIdentifier:@"BullseyeAnnotation"];
    if (self)
    {
        UIImage*    theImage = [UIImage imageNamed:@"bullseye32.png"];
        if (!theImage)
            return nil;

        self.image = theImage;
        self.canShowCallout = YES;
        self.multipleTouchEnabled = NO;
        map = nil;

        UIButton*    rightButton = [UIButton buttonWithType:
                      UIButtonTypeDetailDisclosure];
        [rightButton addTarget:self action:@selector(myShowAnnotationAddress:)
                      forControlEvents:UIControlEventTouchUpInside];
        self.rightCalloutAccessoryView = rightButton;
    }
    return self;
}
@end
```

標的ビューに最初にタッチイベントが届くと、このクラスの`touchesBegan:withEvent:`は、リスト 8-12に示すようにして最初のタッチイベントについての情報を記録します。後から`touchesMoved:withEvent:`メソッドでこの情報を使用して、ビューの位置を調整します。すべての位置情報はスーパービューの座標空間に格納されます。

リスト 8-12　ビューの位置の追跡

```
@implementation BullseyeAnnotationView (TouchBeginMethods)
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    // ビューをシングルタッチ専用に設定する
    UITouch* aTouch = [touches anyObject];
    startLocation = [aTouch locationInView:[self superview]];
    originalCenter = self.center;

    [super touchesBegan:touches withEvent:event];
}
```

```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch* aTouch = [touches anyObject];
    CGPoint newLocation = [aTouch locationInView:[self superview]];
    CGPoint newCenter;

    // ユーザの指が5ピクセル以上移動した場合、ドラッグが始まる
    if ( (abs(newLocation.x - startLocation.x) > 5.0) ||
         (abs(newLocation.y - startLocation.y) > 5.0) )
         isMoving = YES;

    // ドラッグが始まった場合は、ビューの位置を調整する
    if (isMoving)
    {
        newCenter.x = originalCenter.x + (newLocation.x - startLocation.x);
        newCenter.y = originalCenter.y + (newLocation.y - startLocation.y);
        self.center = newCenter;
    }
    else    // 親クラスにそれを処理させる
        [super touchesMoved:touches withEvent:event];
}
@end
```

ユーザが注釈ビューのドラッグを停止したときは、元の注釈の座標を調整してそのビューが新しい位置に確実に留まるようにする必要があります。リスト 8-13は、`BullseyeAnnotationView`クラスの`touchesEnded:withEvent:`メソッドを示しています。このメソッドは、mapメンバ変数を使用してピクセルベースの点を地図座標値に変換します。注釈の`coordinate`プロパティは通常は読み取り専用のため、ここでは注釈オブジェクトが`coordinate`プロパティを使用して、ローカルに格納したりレポートする値を更新するために、`changeCoordinate`カスタムメソッドを実装しています。何らかの理由でタッチイベントがキャンセルされた場合は、`touchesCancelled:withEvent:`メソッドが注釈ビューを元の位置に戻します。

リスト 8-13　最後のタッチイベントの処理

```
@implementation BullseyeAnnotationView (TouchEndMethods)
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (isMoving)
    {
        // 新しい位置を表すように地図座標を更新する
        CGPoint newCenter = self.center;
        BullseyeAnnotation* theAnnotation = self.annotation;
        CLLocationCoordinate2D newCoordinate = [map convertPoint:newCenter
                            toCoordinateFromView:self.superview];

        [theAnnotation changeCoordinate:newCoordinate];

        // 状態情報をクリーンアップする
        startLocation = CGPointZero;
        originalCenter = CGPointZero;
        isMoving = NO;
    }
    else
        [super touchesEnded:touches withEvent:event];
}

- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
```

```
{
    if (isMoving)
    {
        // ビューを最初の位置に戻す
        self.center = originalCenter;

        // 状態情報をクリーンアップする
        startLocation = CGPointZero;
        originalCenter = CGPointZero;
        isMoving = NO;
    }
    else
        [super touchesCancelled:touches withEvent:event];
}
@end
```

逆ジオコーダからプレースマーク情報を取得する

Map Kitフレームワークは、主に緯度と経度のペアで構成される地図座標値を扱います。地図座標はコードには適していますが、ユーザにとっては必ずしもわかりやすいとは限りません。これをユーザにわかりやすくするために、`MKReverseGeocoder`クラスを使用して特定の地図座標に対応する番地、市、県、国などのプレースマーク情報を取得できます。

`MKReverseGeocoder`クラスは、指定された地図座標についての情報を基盤となる地図サービスに問い合わせます。このクラスはネットワークに接続する必要があるため、逆ジオコーダオブジェクトは常に非同期にクエリを実行し、関連するデリゲートオブジェクトを使用してその結果を報告します。逆ジオコーダ用のデリゲートは、`MKReverseGeocoderDelegate`プロトコルに従っていなければなりません。

逆ジオコーダクエリを開始するには、`MKReverseGeocoder`クラスのインスタンスを作成し、適切なオブジェクトを`delegate`プロパティに割り当ててから、`start`メソッドを呼び出します。クエリが正常に終了すると、デリゲートはその結果を含む`MKPlacemark`オブジェクトを受け取ります。プレースマークオブジェクト自身が注釈オブジェクトです（つまり、`MKAnnotation`プロトコルを採用しています）。したがって、必要であればこのオブジェクトをMap Viewの注釈リストに追加できます。

カメラによる写真の撮影

UIKitは、`UIImagePickerController`クラスを通してデバイスのカメラへのアクセスを提供します。このクラスは、利用可能なカメラを使用して写真を撮影するための標準システムインターフェイスを提供します。また、写真を撮影した後で、画像のサイズを変更したり画像を切り抜いたりするためのオプションのコントロールもサポートします。またこのクラスは、ユーザのフォトライブラリから写真を選択するためにも使用できます。

カメラインターフェイスを表すビューは、`UIImagePickerController`クラスによって管理されるモーダルビューです。このビューにはコードから直接アクセスしてはいけません。このビューを表示するには、現在アクティブなView Controllerの`presentModalViewController:animated:`メソッドを呼び出して、`UIImagePickerController`オブジェクトを新しいView Controllerとして渡す必要があります。インストールされると、このPicker Controllerは自動的にカメラインターフェイスをスライドして配置します。カメラインターフェイスは、ユーザが写真を承認するか操作をキャンセルするまでアクティブなままになります。ユーザが承認またはキャンセルすると、このPicker Controllerは、ユーザの選択をデリゲートに通知します。

UIImagePickerControllerクラスによって管理されるインターフェイスは、一部のデバイスでは利用できない場合があります。カメラインターフェイスを表示する前に、UIImagePickerControllerクラスのisSourceTypeAvailable:クラスメソッドを呼び出して、そのインターフェイスが利用可能かどうかを必ず確認する必要があります。このメソッドの戻り値を、常に尊重してください。このメソッドがNOを返した場合は、現在のデバイスがカメラを搭載していないか、何らかの理由でカメラが現在利用できない状態にあることを意味します。このメソッドがYESを返した場合は、以下の手順を実行してカメラインターフェイスを表示します。

1. UIImagePickerControllerの新規オブジェクトを作成します。
2. Picker Controllerにデリゲートオブジェクトを割り当てます。

   ほとんどの場合、現在のView Controllerがピッカーに対するデリゲートとしての役割を果たします。ただし、必要であればまったく別のオブジェクトを使用することもできます。このデリゲートオブジェクトは、UIImagePickerControllerDelegateプロトコルおよびUINavigationControllerDelegateプロトコルに従わなければなりません。

   注： デリゲートがUINavigationControllerDelegateプロトコルに従っていない場合は、コンパイル中に警告が表示されます。ただし、このプロトコルのメソッドはオプションであるためこの警告はコードには影響を与えません。この警告が表示されないようにするには、デリゲートのクラスのサポートプロトコルのリストにUINavigationControllerDelegateプロトコルを含めます。ただしそれに対応するメソッドを実装する必要はありません。

3. ピッカーのタイプをUIImagePickerControllerSourceTypeCameraに設定します。
4. 必要に応じて、allowsImageEditingプロパティに適切な値を割り当てて、写真編集コントロールを有効または無効にします。
5. 現在のView ControllerのpresentModalViewController:animated:メソッドを呼び出して、このピッカーを表示します。

リスト 8-14に、前述の手順を表すコードを示します。presentModalViewController:animatedメソッドを呼び出すとすぐに、Picker Controllerが処理を引き継ぎ、カメラインターフェイスを表示し、そのインターフェイスが閉じられるまですべてのユーザ操作に対応します。ユーザのフォトライブラリから既存の写真を選択するには、ピッカーのsourceTypeプロパティの値をUIImagePickerControllerSourceTypePhotoLibraryに変更するだけです。

リスト 8-14　写真撮影用のインターフェイスの表示

```
-(BOOL)startCameraPickerFromViewController:(UIViewController*)controller
usingDelegate:(id<UIImagePickerControllerDelegate>)delegateObject
{
    if ( (![UIImagePickerController
isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera])
            || (delegateObject == nil) || (controller == nil))
        return NO;

    UIImagePickerController* picker = [[UIImagePickerController alloc] init];
    picker.sourceType = UIImagePickerControllerSourceTypeCamera;
    picker.delegate = delegateObject;
    picker.allowsImageEditing = YES;

    // ピッカーは非同期に表示される
```

```
    [controller presentModalViewController:picker animated:YES];
    return YES;
}
```

ユーザがカメラインターフェイスを閉じるために適切なボタンをタップした場合、UIImagePickerControllerはユーザの選択をデリゲートに通知しますが、インターフェイスを閉じません。ピッカーインターフェイスを閉じるのはデリゲートです（アプリケーションもまた、作業が終了したらピッカーを解放する責任があります。これは、デリゲートメソッドで行えます）。デリゲートが、そもそもピッカーを表示したView Controllerオブジェクトであるのは、このような理由からです。デリゲートメッセージを受け取ると、View Controllerは、dismissModalViewControllerAnimated:メソッドを呼び出して、カメラインターフェイスを閉じます。

リスト 8-15は、リスト 8-14 （186 ページ）で表示したカメラインターフェイスを閉じるデリゲートメソッドを示します。これらのメソッドは、MyViewControllerカスタムクラスで実装されています。この例では、このクラスはUIViewControllerのサブクラスであるため、ピッカーを最初に表示したオブジェクトと同じとみなされています。useImage:メソッドは、このクラスの独自バージョンで実行する作業のための空のプレースフォルダです。このメソッドは、カスタムコードで置き換える必要があります。

リスト 8-15　画像ピッカー用のデリゲートメソッド

```
@implementation MyViewController (ImagePickerDelegateMethods)

- (void)imagePickerController:(UIImagePickerController *)picker
                    didFinishPickingImage:(UIImage *)image
                    editingInfo:(NSDictionary *)editingInfo
{
    [self useImage:image];

    // ピッカーインターフェイスを削除し、ピッカーオブジェクトを解放する
    [[picker parentViewController] dismissModalViewControllerAnimated:YES];
    [picker release];
}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    [[picker parentViewController] dismissModalViewControllerAnimated:YES];
    [picker release];
}

// 独自のコードでこのメソッドを実装して、画像を処理する
- (void)useImage:(UIImage*)theImage
{
}
@end
```

画像の編集が有効になっている場合、ユーザが1つの画像を選択すると、imagePickerController:didFinishPickingImage:editingInfo:メソッドのimageパラメータに編集対象の画像が入ります。この画像は選択された画像として扱う必要があります。ただし、元の画像を保存しておきたい場合は、editingInfoパラメータの辞書から（クロップ矩形とともに）元の画像を取得できます。

フォトライブラリからの写真の選択

UIKitは、UIImagePickerControllerクラスを通してユーザのフォトライブラリへのアクセスを提供します。このコントローラは、写真ピッカーインターフェイスを表示します。このインターフェイスは、ユーザのフォトライブラリ内を移動したり、1つの画像を選択してアプリケーションに返すためのコントロールを提供します。また、ユーザ編集用のコントロールを有効にするオプションもあります。これを利用して、ユーザは返された画像をパンしたりクロップしたりできます。このクラスは、カメラインターフェイスを表示するためにも使用できます。

UIImagePickerControllerクラスは、カメラとユーザのフォトライブラリの両方のインターフェイスを表示するために使用されるので、このクラスを使用する手順はどちらの場合もほとんど同じです。唯一の違いは、ピッカーオブジェクトのsourceTypeプロパティに、UIImagePickerControllerSourceTypePhotoLibraryの値を割り当てる点です。カメラピッカーを表示する手順は、「カメラによる写真の撮影」（185 ページ）で説明しています。

注： カメラピッカーの場合と同様に、必ずUIImagePickerControllerクラスのisSourceTypeAvailable:クラスメソッドを呼び出し、メソッドから返された値を尊重する必要があります。対象デバイスにフォトライブラリが存在することを前提にしてはいけません。たとえ、デバイスにライブラリが存在しても、そのライブラリが現在利用できなければ、このメソッドはNOを返します。

メール作成インターフェイスの使用

iPhone OS 3.0以降では、MFMailComposeViewControllerクラスを使用して、独自のアプリケーション内に標準的なメール作成インターフェイスを表示できます。このインターフェイスを表示する前にこのクラスのメソッドを使用して、メールの宛先、件名、本文、および必要であれば添付ファイルを設定します。（標準的なView Controllerの手法を使用して）このインターフェイスを表示すると、ユーザは、送信するために電子メールを「メール(Mail)」アプリケーションに投稿する前に、その内容を編集するオプションを使用できます。またユーザは、メールを完全にキャンセルする権限も持っています。

注： iPhone OSのすべてのバージョンで、mailtoスキームを使用したURLを作成したり開いたりすることによって電子メールメッセージを作成できます。このタイプのURLは「メール(Mail)」アプリケーションによって自動的に処理されます。このタイプのURLを開く方法の詳細については、「他のアプリケーションとのやり取り」（39 ページ）を参照してください。

メール作成インターフェイスを使用するには、MessageUI.frameworkをXcodeプロジェクトに追加し、関連するターゲットでそれをリンクする必要があります。このフレームワークのクラスとヘッダにアクセスするには、関連するソースファイルの先頭に#import <MessageUI/MessageUI.h>ステートメントを含めます。フレームワークをプロジェクトに追加する方法の詳細については、『*Xcode Project Management Guide*』の「Files in Projects」を参照してください。

アプリケーションでMFMailComposeViewControllerクラスを使用するには、インスタンスを作成し、そのメソッドを使用して電子メールの初期データを設定します。また、ユーザが電子メールを承認またはキャンセルしたときにインターフェイスを消去するには、View Controllerの

mailComposeDelegateプロパティにオブジェクトを割り当てる必要があります。ここで指定するデリゲートオブジェクトはMFMailComposeViewControllerDelegateプロトコルに従わなければなりません。

メール作成インターフェイスに対して電子メールアドレスを指定する場合は、プレーンな文字列オブジェクトを指定します。ユーザの連絡先リストの電子メールアドレスを使用する場合は、Address Bookフレームワークを使用してその情報を取得できます。このフレームワークを使用して電子メールやその他の情報を取得する方法の詳細については、『*Address Book Programming Guide for iPhone OS*』を参照してください。

リスト 8-16は、MFMailComposeViewControllerオブジェクトを作成して、アプリケーション内に電子メール作成用のモーダルインターフェイスを表示するコードを示しています。カスタムのView Controllerの1つにdisplayComposerSheetメソッドを含めて、必要に応じてこのメソッドを呼び出してインターフェイスを表示します。この例では、親のView Controler自身がデリゲートとして割り当てられ、mailComposeController:didFinishWithResult:error:メソッドを実装します。デリゲートメソッドは、これ以上のアクションをとらずにデリゲートを破棄します。独自のアプリケーションでは、デリゲートを使用して*result*パラメータの値を調べることによって、ユーザが電子メールを送信したかキャンセルしたかを追跡できます。

リスト 8-16　メール作成インターフェイスのポスト

```
@implementation WriteMyMailViewController (MailMethods)

-(void)displayComposerSheet
{
    MFMailComposeViewController *picker = [[MFMailComposeViewController alloc]
 init];
    picker.mailComposeDelegate = self;

    [picker setSubject:@"Hello from California!"];

    // 宛先をセットアップする
    NSArray *toRecipients = [NSArray arrayWithObjects:@"first@example.com",
                                    nil];
    NSArray *ccRecipients = [NSArray arrayWithObjects:@"second@example.com",
                                    @"third@example.com", nil];
    NSArray *bccRecipients = [NSArray arrayWithObjects:@"four@example.com",
                                    nil];

    [picker setToRecipients:toRecipients];
    [picker setCcRecipients:ccRecipients];
    [picker setBccRecipients:bccRecipients];

    // 画像をメールに添付する
    NSString *path = [[NSBundle mainBundle] pathForResource:@"ipodnano"
                                   ofType:@"png"];
    NSData *myData = [NSData dataWithContentsOfFile:path];
    [picker addAttachmentData:myData mimeType:@"image/png"
                                   fileName:@"ipodnano"];

    // メールの本文テキストを設定する
    NSString *emailBody = @"It is raining in sunny California!";
    [picker setMessageBody:emailBody isHTML:NO];

    // メール作成インターフェイスを表示する
    [self presentModalViewController:picker animated:YES];
    [picker release]; // これでコントローラを安全に解放できる
```

```
}

// メール作成用View Controllerデリゲートのメソッド
- (void)mailComposeController:(MFMailComposeViewController *)controller
              didFinishWithResult:(MFMailComposeResult)result
              error:(NSError *)error
{
    [self dismissModalViewControllerAnimated:YES];
}
@end
```

インターフェイスを表示するための標準的なView Controllerの手法の詳細については、『*View Controller Programming Guide for iPhone OS*』を参照してください。Message UIフレームワークのクラスの詳細については、『*Message UI Framework Reference*』を参照してください。

アプリケーション環境設定

従来のデスクトップアプリケーションでは、環境設定はアプリケーションの動作や外観の設定に使われるアプリケーション固有の設定です。iPhone OSでも、アプリケーションに必須の部分としてではありませんが、アプリケーション環境設定をサポートしています。ただし、カスタムのユーザインターフェイスを使ってアプリケーションごとに環境設定を表示するのではなく、すべてのアプリケーションレベルの環境設定が、システムの提供する「設定(Settings)」アプリケーションを使って表示されます。

カスタムのアプリケーション環境設定を「設定(Settings)」アプリケーションに組み込むには、アプリケーションバンドルの最上位ディレクトリに専用の書式をもつSettingsバンドルを含める必要があります。このSettingsバンドルは、「設定(Settings)」アプリケーションにアプリケーションの環境設定に関する情報を提供します。「設定(Settings)」アプリケーションは、それらの環境設定を表示し、ユーザが指定する値で環境設定データベースを更新します。実行時には、アプリケーションが標準の取得用APIを使ってこれらの環境設定を取得します。以降の各セクションでは、Settingsバンドルの書式についてと、環境設定値の取得に使用するAPIについて説明します。

環境設定のためのガイドライン

アプリケーションの環境設定を「設定(Settings)」アプリケーションに追加するのに最も適しているのは、生産性型アプリケーションであり、一度設定した環境設定の値がほとんど変更されないような場合です。たとえば、「メール(Mail)」アプリケーションでは、これらの環境設定を使用してユーザのアカウント情報とメッセージチェックの設定を格納します。「設定(Settings)」アプリケーションでは環境設定を階層的に表示できるため、環境設定が数多くある場合にも、「設定(Settings)」アプリケーションから環境設定を操作するのがより適しています。アプリケーションでそのような一連の環境設定を用意すると、必要となる画面が多すぎて、ユーザを混乱させてしまう可能性があります。

アプリケーションにオプションが少ししかない場合や、ユーザが定期的に変更するオプションがある場合、「設定(Settings)」アプリケーションにそれらを置くのが妥当かどうかを慎重に検討する必要があります。たとえば、ユーティリティ型アプリケーションは、メインビューの裏面にカスタムの設定オプションを提供します。ビュー上にある専用のコントロールを使ってビューを反転してオプションを表示し、別のコントロールを使ってビューを元に戻せます。簡単なアプリケーションでは、こうした動作によってアプリケーションのオプションへすばやくアクセスすることができるため、「設定(Settings)」に移動するよりもユーザにとってははるかに便利です。

ゲームなどのフルスクリーンアプリケーションの場合は、「設定(Settings)」アプリケーションを使用したり、環境設定のためのカスタムスクリーンを独自に実装できます。多くの場合、カスタムスクリーンはゲームに適しています。ゲーム設定の一部として扱われるためです。ゲームの流れにとって妥当と考えられる場合は、環境設定に「設定(Settings)」アプリケーションを使用することもできます。

注： 「設定(Settings)」アプリケーションとアプリケーションのカスタムスクリーンにまたがって環境設定を分散させるべきではありません。たとえば、メインビューの裏側に環境設定をもったユーティリティ型アプリケーションは、「設定(Settings)」アプリケーションに設定可能な環境設定をもつべきではありません。環境設定がある場合には、いずれかの方法を採用してそれだけを使用するようにします。

環境設定インターフェイス

「設定(Settings)」アプリケーションは、アプリケーションの環境設定を行き来するための一連のページを階層構造として実装しています。「設定(Setting)」アプリケーションのメインページには、環境設定をカスタマイズできるシステムおよびサードパーティ製アプリケーションが表示されます。サードパーティ製アプリケーションを選択すると、そのアプリケーションの環境設定へ移動します。

各アプリケーションには、メインページと呼ばれる環境設定のページが少なくとも1ページあります。アプリケーションの環境設定がごく少数の場合、必要なページはメインページのみという場合もあります。一方、環境設定の数が多く、メインページに収まらない場合はページを追加できます。これらの追加ページはメインページの子ページになります。ユーザが子ページにアクセスするには、新しいページにリンクしている特別なタイプの環境設定をタップします。

表示するそれぞれの環境設定は、特定のタイプである必要があります。環境設定のタイプによって、その環境設定が「設定(Settings)」アプリケーションにどのように表示されるかが決まります。ほとんどの環境設定のタイプは、環境設定の値を設定するために使用される特定のコントロールのタイプを示します。一方、環境設定を編成するための方法を提供するタイプもあります。表9-1に、「設定(Settings)」アプリケーションでサポートされているさまざまな要素のタイプと、環境設定ページを独自に実装するためにそれぞれのタイプを使用する方法を示します。

表 9-1 環境設定の要素のタイプ

要素のタイプ	説明
Text Field	Text Fieldタイプは、省略可能なタイトルと編集可能なテキストフィールドを表示します。このタイプは、カスタムの文字列値をユーザが指定する必要のある環境設定に使用します。 このタイプのキーは`PSTextFieldSpecifier`です。
Title	Titleタイプは、読み取り専用の文字列の値を表示します。このタイプは、読み取り専用の環境設定の値を表示するために使用できます（環境設定に分かりにくい値や直感的に把握できない値がある場合、このタイプを使用して候補となる値をカスタム文字列にマッピングできます）。 このタイプのキーは`PSTitleValueSpecifier`です。
Toggle Switch	Toggle Switchタイプは、オン／オフ（ON／OFF）のトグルボタンを表示します。このタイプを使って、二者択一型の環境設定を設定できます。一般的に、このタイプを使用してブール値を含む環境変数を表せますが、ブール値以外の値を含む環境変数にも使うことができます。 このタイプのキーは`PSToggleSwitchSpecifier`です。

要素のタイプ	説明
Slider	Sliderタイプは、スライダコントロールを表示します。このタイプは、特定の範囲の値をもつ環境変数に使用できます。このタイプの値は実数で、その最小値と最大値はデベロッパが指定します。 このタイプのキーは`PSSliderSpecifier`です。
Multi Value	Multi Valueタイプは、候補値の一覧から1つの値をユーザに選択させます。このタイプは、相互に排他的な値をサポートする環境設定に使用できます。任意の型の値を設定できます。 このタイプのキーは`PSMultiValueSpecifier`です。
Group	Groupタイプは、1つのページに環境設定の複数のグループを編成するのに使用します。Groupタイプは、設定可能な環境設定ではありません。設定可能な1つ以上の環境設定の直前に表示するタイトル文字列が含まれているだけです。 このタイプのキーは`PSGroupSpecifier`です。
Child Pane	Child Paneタイプは、環境設定の新しいページにユーザを移動させます。このタイプを使用して階層型の環境設定を実装できます。この環境設定タイプの設定方法、および使用方法の詳細については、「階層型環境設定」（195 ページ）を参照してください。このタイプのキーは`PSChildPaneSpecifier`です。

それぞれの環境設定タイプの形式の詳細については、『*Settings Application Schema Reference*』を参照してください。「設定(Setting)」ページのファイルの作成と編集の方法を学習するには、「Settingsバンドルの追加と変更」（197 ページ）を参照してください。

Settingsバンドル

iPhone OSでは、専用のSettingsバンドルを使用してアプリケーションの環境設定を指定します。このバンドルは`Settings.bundle`という名前で、アプリケーションバンドルの最上位ディレクトリに置かれます。バンドルには、1つ以上のSettings Pageファイルが含まれており、このファイルがアプリケーションの環境設定に関する詳細情報を提供します。バンドルには、画像やローカライズされた文字列など、環境設定の表示に必要なその他のサポートファイルも含まれます。表 9-2に、標準的なSettingsバンドルの内容を示します。

表 9-2　`Settings.bundle`ディレクトリの内容

項目名	説明
`Root.plist`	ルートページの環境設定を含むSettings Pageファイル。このファイルの内容の詳細については、「Settings Pageのファイル形式」（194 ページ）で説明しています。
その他の`.plist`ファイル	Child Paneタイプを使用した階層型環境設定のセットを構築すると、それぞれの子ペインの内容は別々のSettings Pageファイルに保存されます。これらのファイル名の指定と、正しい子ペインにこれらのファイルを関連付ける責任はデベロッパにあります。

項目名	説明
1つ以上の.lprojディレクトリ	これらのディレクトリには、Settings Pageファイルのローカライズされた文字列リソースが格納されます。それぞれのディレクトリには1つの文字列ファイルが含まれ、そのタイトルはSettings Pageで指定されます。文字列ファイルは、それぞれの環境設定をユーザに対して表示するためのローカライズされた内容を提供します。
その他の画像	スライダコントロールを使用する場合、スライダの画像をバンドルの最上位のディレクトリに保存できます。

アプリケーションバンドルには、Settingsバンドルに加えて、アプリケーション設定用のカスタムアイコンを含めることができます。Icon-Settings.pngという名前のファイルがアプリケーションバンドルのディレクトリの最上位にあれば、そのアイコンが「設定(Settings)」アプリケーションにおいて、アプリケーションの環境設定を示すのに使用されます。そのような画像ファイルがない場合、「設定(Settings)」アプリケーションは代わりにアプリケーションのアイコンファイル（デフォルトでは、Icon.png）を使用し、必要に応じて拡大縮小します。Icon-Settings.pngファイルは、29×29ピクセルの画像にします。

「設定(Settings)」アプリケーションは起動すると、各カスタムアプリケーションを確認してSettingsバンドルがあるかどうかをチェックします。カスタムバンドルを見つけると、そのバンドルをロードし、対応するアプリケーション名とアイコンを「設定(Settings)」のメインページに表示します。ユーザがアプリケーションに対応する行をタップすると、「設定(Settings)」は該当するSettingsバンドルのRoot.plist Settings Pageファイルをロードし、そのファイルを使用してアプリケーションの環境設定のメインページを表示します。

バンドルのRoot.plist Settings Pageファイルをロードするほかに、「設定(Settings)」アプリケーションでは必要に応じてそのファイルの言語固有のリソースもロードします。各Settings Pageファイルには、ユーザに表示する文字列をローカライズした値を含んだ.stringsファイルを関連付けることができます。「設定(Settings)」アプリケーションは、表示する環境設定を用意する際に、ユーザの優先言語に対応する文字列リソースを探し、環境設定のページでそれらを置き換えてから表示します。

Settings Pageのファイル形式

SettingsバンドルのそれぞれのSettings Pageファイルは、構造化されたファイル形式であるiPhone Settingsプロパティリストファイル形式で保存されます。Settings Pageファイルを編集する最も簡単な方法は、Xcodeに組み込みのエディタ機能を使用することです。「編集用のSettings Pageの準備」（197 ページ）を参照してください。Xcodeツールに付属のProperty List Editorアプリケーションを使って、プロパティリストファイルを編集することもできます。

注： アプリケーションをビルドするとき、Xcodeはプロジェクト内のすべてのXMLベースのプロパティファイルを自動的にバイナリ形式に変換します。この変換によりスペースが節約されます。これはビルド時に自動的に行われます。

各Settings Pageファイルのルート要素には、表 9-3に示すキーが含まれます。実際に必要なキーは1つだけですが、両者を含めることをお勧めします。

表 9-3 環境設定のSettings Pageファイルのルートレベルキー

キー	型	値
`PreferenceSpecifiers`（必須）	配列	このキーの値は辞書配列です。それぞれの辞書には1つの環境設定要素が含まれます。要素タイプの一覧については、表 9-1（192 ページ）を参照してください。それぞれの要素のタイプに対応するキーの説明については、『*Settings Application Schema Reference*』を参照してください。
`StringsTable`	文字列	このファイルに関連付けられた文字列ファイルの名前です。このファイルのコピー（適切にローカライズされた文字列を含む）は、それぞれのバンドルの言語固有のプロジェクトディレクトリに配置する必要があります。このキーを含めない場合、このファイルの文字列はローカライズされません。これらの文字列がどのように使用されるかについては、「ローカライズされたリソース」（196 ページ）を参照してください。

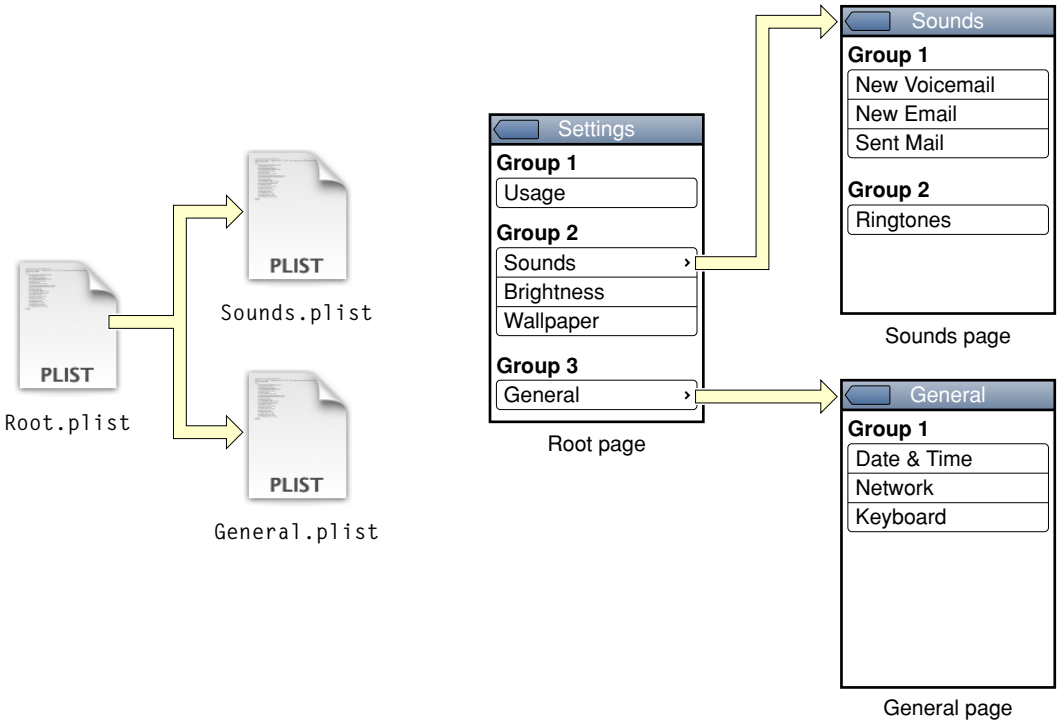
階層型環境設定

環境設定を階層構造にする場合は、定義する各ページがそれぞれ独立の`.plist`ファイルを持つ必要があります。各`.plist`ファイルには、そのページにのみ表示される環境設定一式を含めます。アプリケーションの環境設定のメインページは、必ず`Root.plist`ファイルに格納します。追加のページには任意の名前をつけられます。

親ページと子ページとの間のリンクを指定するには、Child Pane要素を親ページ内に含めます。子ペインの要素は1つの行を作成します。この行をタップすると、新しい設定ページが表示されます。子ペイン要素の`File`キーは、子ページのコンテンツを定義した`.plist`ファイルの名前を表します。`Title`キーは、子ページのタイトルを表します。このタイトルは、子ページを表示するためにユーザがタップする行のテキストとしても使われます。「設定(Settings)」アプリケーションは、ユーザが親ページに戻るためのナビゲーションコントロールを子ページ上に自動的に表示します。

図 9-1に、一連の階層型ページの仕組みを示します。図の左側は`.plist`ファイルを表し、右側は対応する各ページの関係を示します。

図 9-1 子ペインを使った環境設定の編成



子ペインの要素とそれに関連するキーの詳細については、『*Settings Application Schema Reference*』を参照してください。

ローカライズされたリソース

環境設定にはユーザに表示する文字列が含まれるため、Settingsバンドルにこれらの文字列をローカライズしたものを含めて提供する必要があります。それぞれの環境設定のページには、バンドルでサポートするローカライズごとに、`.strings`ファイルを関連付けることができます。「設定(Settings)」アプリケーションは、ローカライズ可能なキーを検出すると、適切なローカライズ`.strings`ファイルに、一致するキーがあるか確認します。該当するキーが見つかれば、そのキーに対応する値を表示します。

`.strings`ファイルなどのローカライズされたリソースを探す場合、「設定(Settings)」アプリケーションはMac OS Xアプリケーションと同じ規則に従います。まず、ユーザが選択した言語設定に一致するリソースのローカライズされたバージョンを探します。ユーザが選択した言語のリソースが存在しない場合、適切な代替言語が選択されます。

文字列ファイルの形式、言語固有のプロジェクトディレクトリ、および言語固有のリソースをバンドルから取得する方法の詳細については、『*Internationalization Programming Topics*』を参照してください。

Settingsバンドルの追加と変更

Xcodeでは、現在のプロジェクトにSettingsバンドルを追加するためのテンプレートを提供しています。デフォルトのSettingsバンドルには、`Root.plist`ファイルと、任意のローカライズされたリソースを格納するためのデフォルトの言語ディレクトリが含まれています。デベロッパは、このバンドルを拡張して、Settingsバンドルで必要な追加のプロパティリストファイルやリソースを含めることができます。

Settingsバンドルの追加

XcodeプロジェクトにSettingsバンドルを追加するには、次の手順を実行します。

1. 「ファイル(File)」＞「新規ファイル(New File)」を選びます。
2. 「iPhone OS」＞「Settings」＞「Settings Bundle」テンプレートを選択します。
3. ファイルに`Settings.bundle`と名前を付けます。

Xcodeは、プロジェクトに新規Settingsバンドルを追加するほかに、アプリケーションのターゲットのCopy Bundle Resourcesビルドフェーズにそのバンドルを自動的に追加します。つまり、デベロッパが行う必要があるのは、Settingsバンドルのプロパティリストファイルを修正し、必要なリソースを追加することだけです。

新しく追加された`Settings.bundle`バンドルの構造は、以下のとおりです。

```
Settings.bundle/
    Root.plist
    en.lproj/
        Root.strings
```

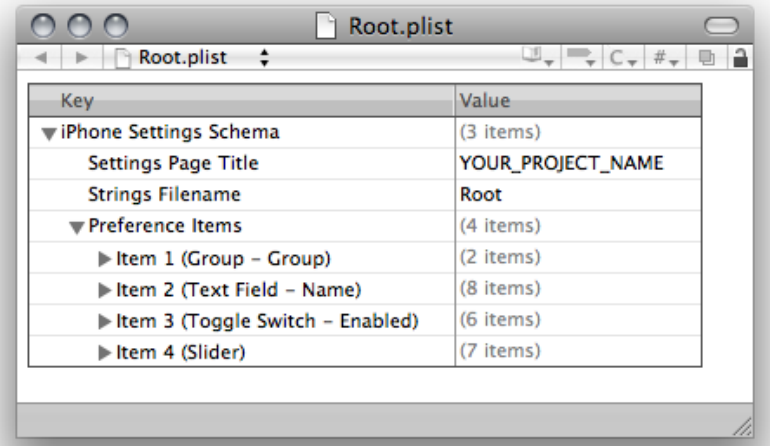
編集用のSettings Pageの準備

Settings Bundleテンプレートを使用してSettingsバンドルを作成したら、スキーマファイルの内容を書式化して編集しやすくできます。以下の手順は、Settingsバンドルの`Root.plist`ファイルに対してこれを実行する方法を示していますが、ほかのスキーマファイルの場合も手順は同じです。

1. Settingsバンドルの`Root.plist`ファイルの内容を表示します。
    a. 「グループとファイル(Groups & Files)」リストで、`Settings.bundle`を開き、内容を表示します。
    b. `Root.plist`ファイルを選択します。詳細ビューにその内容が表示されます。
2. 詳細ビューで、`Root.plist`ファイルのRootキーを選択します。
3. 「表示(View)」＞「プロパティリストタイプ(Property List Type)」＞「iPhone Settings plist」を選択します。

このコマンドは、詳細ビュー内のプロパティリストの内容を書式化します。プロパティリストのキー名と値を表示する代わりに、Xcodeは、そのファイルの内容を理解したり編集しやすくするために、わかりやすい文字列に置き換えます（図 9-2を参照）。

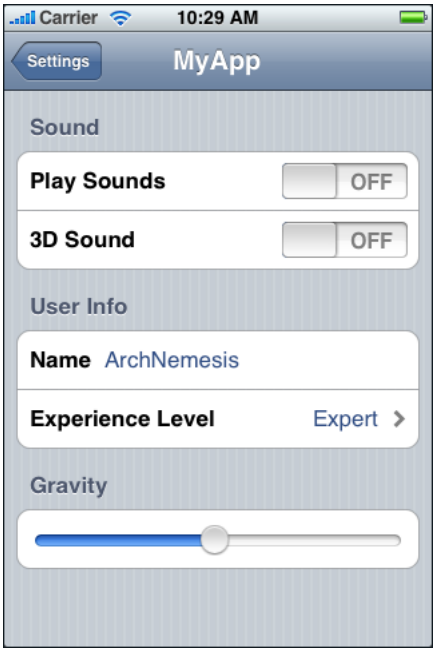
図 9-2 書式化されたRoot.plistファイルの内容



Settings Pageの設定：チュートリアル

このセクションには、Settingsページを設定して、必要なコントロールを表示する方法を示したチュートリアルが含まれています。このチュートリアルの目的は、図 9-2に示すようなページを作成することです。プロジェクト用のSettingsバンドルをまだ作成していない場合は、この手順に進む前に、「編集用のSettings Pageの準備」（197 ページ）で説明した手順を実行してください。

図 9-3 ルートのSettingsページ



1. Settings Page Titleキーの値を、アプリケーションの名前に変更します。

   `YOUR_PROJECT_NAME`テキストをダブルクリックし、そのテキストを`MyApp`に変更します。

2. Preference Itemsキーを展開し、テンプレートに含まれているデフォルトのアイテムを表示します。

3. `Item 1`のタイトルを`Sound`に変更します。

   - `Preference Items`の`Item 1`を開きます。
   - `Title`キーの値を`Group`から`Sound`に変更します。
   - `Type`キーは`Group`に設定されたままにします。

4. 新たに名前変更したSoundグループに、第1のトグルスイッチを作成します。

   - `Preference Items`の`Item 3`を選択し、「編集(Edit)」＞「カット(Cut)」を選びます。
   - `Item 1`を選択し、「編集(Edit)」＞「ペースト(Paste)」を選びます（これによって、Text Field Itemの前にToggle Switch Itemが移動します）。
   - Toggle Switch Itemを開き、設定用のキーを表示します。
   - `Title`キーの値を`Play Sounds`に変更します。
   - `Identifier`キーの値を`play_sounds_preference`に変更します。これで、このアイテムは次の図に示すように設定されます。

Item 2 (Toggle Switch – Play Sounds)	(6 items)
Type	Toggle Switch
Title	Play Sounds
Identifier	play_sounds_preference
Default Value	☑
Value for ON	YES
Value for OFF	NO

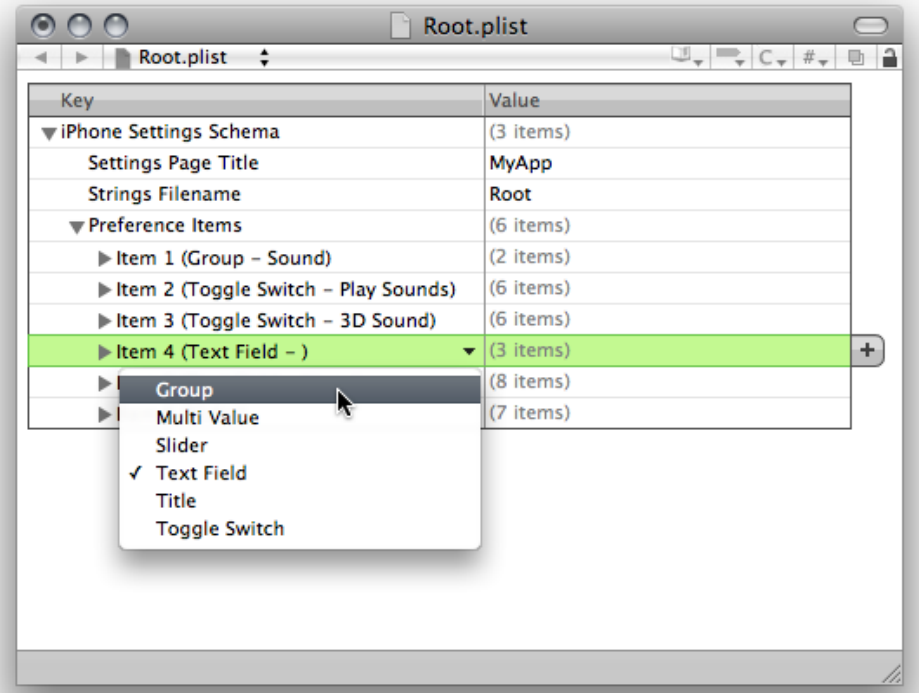
5. Soundグループに第2のToggle Switchを作成します。

   - `Item 2` (Play Sounds Toggle Switch)を選択します。
   - 「編集(Edit)」＞「コピー(Copy)」を選択します。
   - 「編集(Edit)」＞「ペースト(Paste)」を選択します。これで、このToggle Switchは第1のスイッチのすぐ後に配置されます。
   - この新しいToggle Switch Itemを展開し、設定用のキーを表示します。
   - `Title`キーの値を`3D Sound`に変更します。
   - `Identifier`キーの値を`3D_sound_preference`に変更します。

   この時点で、設定の第1グループは完成し、User Infoグループを作成する準備ができました。

6. `Item 4`をGroup要素に変更し、その名前を`User Info`にします。

   - `Preferences Items`の`Item 4`をクリックします。アイテムタイプリストのドロップダウンメニューが表示されます。
   - このドロップダウンメニューから、`Group`を選択し、この要素のタイプを変更します。



- `Item 4`の内容を展開します。
- `Title`キーの値を`User Info`に設定します。

7. Nameフィールドを作成します。
   - Preferences Itemの`Item 5`を選択します。
   - ドロップダウンメニューを使用して、タイプを`Text Field`に変更します。
   - `Title`キーの値を`User Info`に設定します。
   - `Identifier`キーの値を`user_name`に設定します。
   - このアイテムの展開ボタンを切り替えて、内容を非表示にします。

8. Experience Levelの設定を作成します。
   - `Item 5`を選択し、プラス記号(+)ボタンをクリックして（または、Returnキーを押して）、新しいアイテムを作成します。
   - 新しいアイテムをクリックし、タイプを`Multi Value`に設定します。
   - アイテムの内容を展開し、Titleを`Experience Level`に設定します。また、Identifierを`experience_preference`に、Default Valueを`0`に設定します。
   - Default Valueキーが選択されている状態で、プラス記号ボタンをクリックし、`Titles`配列を追加します。

- Titles配列の展開ボタンを開き、テーブルの右端にあるアイテムボタンをクリックします。このボタンをクリックすると新しいサブアイテムをTitlesに追加します。
- この新しいサブアイテムを選択し、プラス記号ボタンをもう2回クリックし、合計3個のサブアイテムを作成します。
- これらのサブアイテムの値を、Beginner、Expert、およびMasterに設定します。
- Titlesキーを再度選択し、展開ボタンをクリックしてサブアイテムを非表示にします。
- プラス記号ボタンをクリックし、Values配列を作成します。
- 3つのサブアイテムをValues配列に追加し、その値を0、1、2に設定します。
- Item 6の展開ボタンをクリックし、その内容を非表示にします。

9. 最後のグループをSettingsページに追加します。
   - 新しいアイテムを作成し、TypeをGroupに設定します。また、TitleをGravityに設定します。
   - 新しいアイテムをもう1つ作成し、TypeをSliderに設定します。また、Identifierをgravity_preferenceに、Default Valueを1に、Maximum Valueを2に設定します。

追加のSettings Pageファイルの作成

Settings BundleテンプレートにはRoot.plistファイルが含まれています。このファイルは、アプリケーションの最上位のSettingsページを定義します。追加のSettingsページを定義するには、Settingsバンドルにプロパティリストファイルを追加する必要があります。この操作は、FinderまたはXcodeから実行できます。

XcodeでSettingsバンドルにプロパティファイルを追加するには、次の手順を実行します。

1. 「グループとファイル(Groups & Files)」ペインで、Settingsバンドルを開き、Root.plistファイルを選択します。
2. 「ファイル(File)」＞「新規ファイル(New)」を選びます。
3. 「Other」＞「Property List」を選びます。
4. 新しいファイルを選択して、「表示(View)」＞「プロパティリストタイプ(Property List Type)」＞「iPhone Settings plist」を選び、それをSettingsファイルとして設定します。

新しいSettingsページをSettingsバンドルに追加したら、「Settings Pageの設定：チュートリアル」（198 ページ）での説明にあるように、そのページの内容を編集できます。このページの設定を表示するには、「階層型環境設定」（195 ページ）で説明したように、Child Pane要素から参照する必要があります。

環境設定へのアクセス

iPhoneアプリケーションでは、FoundationフレームワークおよびCore Foundationフレームワークのいずれかを使用して環境設定の値を取得したり設定したりします。Foundationフレームワークでは、`NSUserDefaults`クラスを使用して環境設定の値の取得と設定を行います。Core Foundationフレームワークでは、環境設定に関連するいかくつかの関数を使用して値の取得と設定を行います。

リスト 9-1に、アプリケーションから環境設定の値を読み取る方法の簡単な例を示します。この例では、`NSUserDefaults`クラスを使用して、「Settings Pageの設定：チュートリアル」（198 ページ）で作成した環境設定から値を読み取り、それをアプリケーション固有のインスタンス変数に代入しています。

リスト 9-1　アプリケーションの環境設定の値へのアクセス

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
{
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [self setShouldPlaySounds:[defaults boolForKey:play_sounds_preference]];

    // アプリケーションの初期化を終了
}
```

環境設定の読み取りと書き込みに使用する`NSUserDefaults`メソッドの詳細については、『*NSUserDefaults Class Reference*』を参照してください。環境設定の読み取りと書き込みに使用するCore Foundation関数の詳細については、『*Preferences Utilities Reference*』を参照してください。

シミュレートしたアプリケーションの環境設定のデバッグ

iPhone Simulatorでアプリケーションを実行すると、アプリケーションの環境設定の値はすべて`~/ライブラリ/Application Support/iPhone Simulator/User/Applications/`*<APP_ID>*`/Library/Preferences`に保存されます。*<APP_ID>*は、iPhone OSがアプリケーションを識別するために使用するディレクトリ名で、プログラムによって生成されます。

アプリケーションをインストールするたびに、iPhone OSはクリーンインストールを実行してそれまでの環境設定を削除します。つまり、Xcodeからアプリケーションをビルドしたり実行したりすると、常に新しいバージョンがインストールされ、古いコンテンツはすべて置き換えられるということです。実行を重ねる間の環境設定の変更をテストするには、Xcodeからではなくシミュレータインターフェイスから直接アプリケーションを実行する必要があります。

書類の改訂履歴

この表は「*iPhoneアプリケーションプログラミングガイド*」の改訂履歴です。

日付	メモ
2009-06-17	コンパスインターフェイスの使用についての情報を追加しました。
	OpenGLサポートについての情報を『*OpenGL ES Programming Guide for iPhone*』に移動しました。
	サポートされている`Info.plist`キーのリストを更新しました。
2009-05-14	iPhone 3.0向けに更新しました。
	「イベント処理」の章に「コピーアンドペースト操作」のコード例を追加しました。
	「ファイルとネットワーク」の章にキーチェーンデータについてのセクションを追加しました。
	マップインターフェイスとメールインターフェイスの表示方法についての情報を追加しました。
	若干の修正をしました。
2009-01-06	誤字をいくつか訂正し、「設定(Settings)」アプリケーションの子ページ作成プロセスを分かりやすくしました。
2008-11-12	浮動小数点演算の考慮事項についてガイダンスを追加しました。
	iTunesでのバックアップに関連する情報を更新しました。
2008-10-15	文書の内容を編成しなおしました。
	『*iPhone OS Technology Overview*』に、iPhone OSの高度な情報を移動しました。
	『*Apple URL Scheme Reference*』に、標準システムのURLスキームについての情報を移動しました。
	『*iPhone Development Guide*』に、開発ツール、およびデバイスの設定方法についての情報を移動しました。
	アプリケーションアーキテクチャを紹介し、iPhoneアプリケーション作成についての多くのガイダンスを取り上げている「コアアプリケーション」の章を作成しました。

日付	メモ
	TextクラスとWebクラスの使用および画面上のキーボード操作について取り上げた「テキストとWeb」の章を追加しました。
	「ファイルとネットワーク」の章を作成し、外部の情報をここに移動しました。
	『*iPhone OS Programming Guide*』からドキュメント名を変更しました。
2008-07-08	iPhone OSおよびiPhone OSアプリケーションの開発プロセスについて説明する新規文書。