iOS View
プログラミングガイド

Developer

3. UIKitフレームワークは、タッチを`UIEvent`オブジェクトにパッケージ化し、これを適切なビューにディスパッチします（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、『*Event Handling Guide for iOS*』を参照してください）。
4. ビューのイベント処理コードが、イベントに応答します。たとえば、コードは以下を実行します。
   - ビューまたはサブビューのプロパティ（frame、bounds、alphaなど）を変更します。
   - `setNeedsLayout`メソッドを呼び出して、ビュー（または、そのサブビュー）にレイアウトの更新が必要であることを示すマークを付けます。
   - `setNeedsDisplay`または`setNeedsDisplayInRect:`メソッドを呼び出して、ビュー（または、そのサブビュー）に再描画が必要であることを示すマークを付けます。
   - 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、ビューが実行することや呼び出すメソッドを決定するのはデベロッパです。
5. 何らかの理由でビューのジオメトリが変更された場合は、UIKitが以下の規則に従ってサブビューを更新します。
   a. ビューに自動サイズ変更の規則を設定している場合は、UIKitがその規則に従って各ビューを調整します。自動サイズ変更規則の動作については、“自動サイズ変更規則を使用した自動的なレイアウト変更の処理”（48 ページ）を参照してください。
   b. ビューが`layoutSubviews`メソッドを実装している場合は、UIKitがこれを呼び出します。

      カスタムビューでこのメソッドをオーバーライドし、これを使用してサブビューの位置とサイズを調整できます。たとえば、大きなスクロール可能領域を持つビューでは、メモリに収まらないような大きなビューを1つ作成するのではなく、複数のサブビューを「タイル」として使用する必要があります。このメソッドの実装で、ビューは現在画面外にあるサブビューを非表示にするか、これらを再配置して新しく表示されるコンテンツの描画に使用します。このプロセスの一部として、ビューのレイアウトコードは再描画を必要とするビューを無効にすることもできます。
6. ビューのいずれかの部分に再描画が必要であることを示すマークが付けられた場合、UIKitはビューに再描画を要求します。

   `drawRect:`メソッドを明示的に定義しているカスタムビューでは、UIKitがそのメソッドを呼び出します。このメソッドの実装では、ビューの指定された領域をできるだけ早く再描画し、ほかには何も処理しないようにします。ここでは、その他のレイアウト変更やアプリケーションのデータモデルの変更は行わないでください。このメソッドの目的は、ビューの視覚的なコンテンツを更新することです。

   標準のシステムビューは、一般的に`drawRect:`メソッドを実装しませんが、この時点で描画を管理しています。
7. 更新されたビューは、アプリケーションで表示中の残りの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信されます。

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。

> **注意:** 上の更新モデルは、主に標準のシステムビューと描画手法を使用するアプリケーションに適用されます。描画にOpenGL ESを使用するアプリケーションは、一般的に1つのフルスクリーンビュを設定して、関連するOpenGL ESグラフィックスコンテキストに直接描画します。この場合、ビューはタッチイベントを処理しますが、ビューがフルスクリーンであるため、サブビューの配置は必要ありません。OpenGL ESの使用方法については、『*OpenGL ES Programming Guide for iOS*』を参照してください。

前述の一連のステップにおける、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。

- イベント処理メソッド：
  - `touchesBegan:withEvent:`
  - `touchesMoved:withEvent:`
  - `touchesEnded:withEvent:`
  - `touchesCancelled:withEvent:`
- `layoutSubviews`メソッド
- `drawRect:`メソッド

オーバーライドするビューのメソッドとしては、これらのメソッドが最も一般的ですが、すべてをオーバーライドする必要はありません。ジェスチャリコグナイザを使用してイベントを処理する場合は、いずれのイベント処理メソッドもオーバーライドする必要がありません。同様に、ビューにサブビューが含まれていない場合や、ビューのサイズが変更されない場合は、`layoutSubviews`メソッドをオーバーライドする必要がありません。さらに、ビューのコンテンツが実行時に変更可能で、描画にUIKitやCore Graphicsなどのネイティブなテクノロジーを使用している場合、オーバーライドが必要なのは`drawRect:`メソッドのみです。

これらは主な統合場所であり、統合できる場所はこれだけではないことにも注意してください。`UIView`クラスのいくつかのメソッドは、サブクラスのオーバーライドポイントとなるように設計されています。『*UIView Class Reference*』のメソッドの説明で、どのメソッドがカスタム実装でのオーバーライドに適しているかを確認してください。

ビューを効果的に使用するためのヒント

カスタムビューは、標準のシステムビューで提供されていないものを描画する必要があるときに有用ですが、ビューの良好なパフォーマンスを保証する必要があります。UIKitは実行できる処理をすべて行って、ビューに関連する動作を最適化し、カスタムビューで良好なパフォーマンスを達成するための支援を行います。以下のヒントについて考慮すると、UIKitがこれらの処理を行う際に役立ちます。

> **Important:** 描画コードを最適化する前に、常にビューの現在のパフォーマンスに関するデータを収集してください。現在のパフォーマンスを測定することで、実際に問題があるかどうかを確認できます。また、問題がある場合は、最適化した結果を比較する際の基準として使用できます。

ビューには必ずしも対応するビューコントローラがあるとは限らない

アプリケーションで、個々のビューとビューコントローラの間に1対1の関係があることはまれです。ビューコントローラの役割はビュー階層を管理することで、一般的にビュー階層は自己完結型の機能を実装するために使用される複数のビューで構成されます。iPhoneアプリケーションの場合、一般的には各ビュー階層が画面全体に表示されますが、iPadアプリケーションでは、1つのビュー階層は画面の一部分しか占有しません。

アプリケーションのユーザインターフェイスを設計するときに、ビューコントローラの役割を考慮することは重要です。ビューコントローラは、画面上でのビューの表示の制御、画面からのビューの削除の制御、メモリ不足の警告に対するメモリの解放、インターフェイスの向きの変更に応じたビューの回転など、多くの重要な処理を提供します。これらの処理を使用しなければ、アプリケーションは正しく動作しないか、予期しない動作を行います。

ビューコントローラの詳細とアプリケーションにおけるその役割については、『*View Controller Programming Guide for iOS*』を参照してください。

カスタム描画を最小限にする

カスタム描画が必要な場合もありますが、可能な場合は使用を避けるようにしてください。カスタム描画が本当に必要となるのは、目的の外観や機能が既存のシステムビュークラスで提供されていないときだけです。既存のビューを組み合わせてコンテンツを構成できる場合は、これらのビューオブジェクトを組み合わせてカスタムビュー階層を構築することをお勧めします。

コンテンツモードを利用する

コンテンツモードは、ビューの再描画にかかる時間を最小化します。デフォルトでは、ビューは`UIViewContentModeScaleToFill`コンテンツモードを使用し、ビューのフレーム矩形に合わせてビューの既存のコンテンツを拡大縮小します。必要な場合は、このモードを変更してコンテンツを調

整できますが、可能であればUIViewContentModeRedrawコンテンツモードの使用は避けてください。現在のコンテンツモードにかかわらず、setNeedsDisplayまたはsetNeedsDisplayInRect:を呼び出すことで、常にビューのコンテンツを強制的に再描画できます。

可能な場合は常にビューを不透明として宣言する

UIKitでは、各ビューのopaqueプロパティを使用して、合成処理を最適化するよう設定できます。カスタムビューでこのプロパティの値をYESに設定すると、UIKitにビューの背面にあるコンテンツをレンダリングする必要がないことが通知されます。レンダリング少なければ、描画コードのパフォーマンスを向上させることができ、一般的にはこの設定が推奨されます。もちろん、opaqueプロパティをYESに設定した場合、ビューは境界矩形一杯に完全に不透明なコンテンツを表示する*必要があります*。

スクロール時のビューの描画動作を調整する

スクロールでは、短時間に大量のビューの更新が発生する場合があります。ビューの描画コードが適切に調整されていない場合、ビューのスクロール時のパフォーマンスが低下する可能性があります。ビューのコンテンツを常に最新に維持しようとするよりも、スクロール操作を開始するときに、ビューの動作を変更することを検討してください。たとえば、レンダリングされたコンテンツの品質を一時的に下げたり、スクロールの実行中にコンテンツモードを変更するなどの方法が考えられます。スクロールを停止したら、ビューを元の状態に戻し、必要に応じてコンテンツを更新します。

サブビューの埋め込みでコントロールをカスタマイズしない

標準のシステムコントロール（UIControlから継承するオブジェクト）にサブビューを追加することは技術的に可能ですが、この方法でコントロールをカスタマイズしないでください。コントロールをカスタマイズする場合は、コントロールクラス自身の明示的で十分に検証されたインターフェイスを通して行います。たとえば、UIButtonクラスにはボタンのタイトルや背景イメージを設定するメソッドが含まれています。定義済みのカスタマイズ方法を使用することで、コードは常に正しく動作します。これらのメソッドを使用せずに、ボタン内にカスタム画像ビューやラベルを埋め込んだ場合、アプリケーションはその時点で、または後にボタンの実装が変更された時点で正しく動作しなくなる可能性があります。

ウインドウ

すべてのiOSアプリケーションには少なくとも1つのウインドウ（`UIWindow`クラスのインスタンス）が必要で、アプリケーションによっては複数のウインドウが含まれる場合もあります。ウインドウオブジェクトには、いくつかの役割があります。

- アプリケーションが表示しているコンテンツを含みます。
- タッチイベントをビューやその他のアプリケーションオブジェクトに送信する際に、重要な役割を果たします。
- アプリケーションのビューコントローラと連携して、向きの変更を容易にします。

iOSでは、ウインドウはタイトルバー、閉じるボックス、またはその他の視覚的な装飾を持ちません。ウインドウは常に、1つまたは複数のビューのための空のコンテナです。また、アプリケーションは新しいウインドウを表示することでコンテンツを変更しません。表示されているコンテンツを変更する場合は、ウインドウの最前面にあるビューを変更します。

ほとんどのiOSアプリケーションは、ウインドウを1つだけ作成し、終了するまでそのウインドウを使用します。このウインドウはデバイスのメイン画面全体に表示され、アプリケーション起動後の初期の段階で、アプリケーションのメインnibファイルからロードされます（または、プログラムにより作成されます）。ただし、アプリケーションがビデオ出力用に外部ディスプレイの使用をサポートする場合は、追加ウインドウを作成してコンテンツを外部ディスプレイに表示できます。一般的に、その他のウインドウはすべてシステムによって作成され、通常は特定のイベント（電話の着信など）に応答して作成されます。

ウインドウの使用を伴うタスク

多くのアプリケーションでは、アプリケーションがウインドウとやり取りするのは、起動時にウインドウを作成するときだけです。ただし、アプリケーションのウインドウオブジェクトを使用して、アプリケーションに関するいくつかのタスクを実行できます。

- **ウインドウを使用して、点および矩形をウインドウのローカル座標系に変換したり、その逆を行います。**たとえば、ウインドウの座標で値が与えられている場合は、その値を使用する前に特定のビューの座標系に変換できます。座標を変換する方法については、“ビュー階層での座標の変換” （46 ページ）を参照してください。

- **ウインドウの通知を使用して、ウインドウに関連する変更を追跡します。**ウインドウは表示と非表示が切り替わったときや、キーウインドウになったとき、またはキーウインドウでなくなったときに通知を生成します。これらの通知を使用して、アプリケーションの別の部分で操作を実行できます。詳細については、「"ウインドウの変更の監視"（29 ページ）を参照してください。

ウインドウの作成と設定

アプリケーションのメインウインドウは、プログラムで作成および設定するか、Interface Builderを使用して作成および設定できます。いずれの場合も、起動時にウインドウを作成し、作成したウインドウを保持して、ウインドウへの参照をアプリケーションのデリゲートオブジェクトに格納する必要があります。アプリケーションが追加のウインドウを作成する場合は、そのウインドウが必要になるまで作成を遅らせます。たとえば、アプリケーションが外部ディスプレイへのコンテンツの表示をサポートする場合は、ディスプレイが接続されるのを待ってから、対応するウインドウを作成します。

アプリケーションがフォアグラウンドまたはバックグラウンドのどちらで起動されるかにかかわらず、アプリケーションのメインウインドウは常に起動時に作成する必要があります。ウインドウの作成と設定自体は、負荷の大きい操作ではありません。ただし、アプリケーションがそのままバックグラウンドで起動される場合は、アプリケーションがフォアグラウンドになるまでウインドウの表示を避ける必要があります。

Interface Builderでのウインドウの作成

Interface Builderを使用したアプリケーションのメインウインドウの作成は、Xcodeプロジェクトのテンプレートが必要な処理を行うため簡単です。すべての新しいXcodeアプリケーションプロジェクトには、アプリケーションのメインウインドウを含むメインnibファイルが（通常は、`MainWindow.xib`のような名前で）含まれます。また、これらのテンプレートは、アプリケーションデリゲートオブジェクトにウインドウのアウトレットも定義します。コードでウインドウオブジェクトにアクセスするには、このアウトレットを使用します。

Important: Interface Builderでウインドウを作成する場合は、「Attributes」インスペクタの「Launch」オプションで「Full Screen」を有効にすることをお勧めします。このオプションが有効でない場合、ウインドウがターゲットデバイスの画面より小さいと、一部のビューでタッチイベントが受信されません。これは、ウインドウが（すべてのビューと同様に）境界矩形の外側にあるタッチイベントを受信しないためです。ビューはデフォルトでウインドウの境界に合わせて切り取られることがないため、ビューは表示されますが、イベントはビューに到達しません。「Launch」オプションで「Full Screen」を有効にすると、ウインドウのサイズが現在の画面に合わせて適切に設定されます。

Interface Builderを使用するようにプロジェクトを更新している場合、Interface Builderを使用したウインドウの作成は、単にウインドウオブジェクトをnibファイルにドラッグするだけです。もちろん、以下の操作も必要です。

- 実行時にウインドウにアクセスするには、ウインドウをアウトレットに接続します。一般的に、アウトレットはアプリケーションのデリゲートまたはnibファイルのFile's Ownerで定義されています。
- プロジェクトの更新計画で新しいnibファイルをアプリケーションのメインnibファイルにすることが予定されている場合は、アプリケーションの`Info.plist`ファイルで、`NSMainNibFile`キーをnibファイルの名前に設定する必要もあります。このキーの値を変更すると、アプリケーションデリゲートの`application:didFinishLaunchingWithOptions:`メソッドが呼び出されるまでに、nibファイルがロードされ利用可能になることが保証されます。

nibファイルを作成および設定する方法については、『*Interface Builder User Guide*』を参照してください。実行時にnibファイルをアプリケーションにロードする方法については、『*Resource Programming Guide*』の「Nib Files」を参照してください。

プログラムによるウインドウの作成

アプリケーションのメインウインドウをプログラムで作成する場合は、アプリケーションデリゲートの`application:didFinishLaunchingWithOptions:`メソッドに、次のようなコードを追加する必要があります。

```
self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]]
autorelease];
```

上の例では、`self.window`はウインドウオブジェクトを保持するように設定されたアプリケーションデリゲートの宣言済みプロパティと見なされます。外部ディスプレイ用のウインドウを作成している場合は、これを別の変数に割り当て、そのディスプレイを表すメイン以外の`UIScreen`オブジェクトの境界を指定する必要があります。

ウインドウを作成するときには、常にウインドウのサイズを画面の境界一杯に設定してください。ステータスバーやその他の項目を考慮して、ウインドウのサイズを小さくする必要はありません。ステータスバーは常にウインドウの最前面に表示されるため、ステータスバーを考慮して小さくする必要があるのは、ウインドウに配置するビューだけです。また、ビューコントローラを使用している場合は、ビューコントローラがビューのサイズを自動的に変更します。

ウインドウへのコンテンツの追加

一般的に、各ウインドウは（対応するビューコントローラで管理される）ルートビューオブジェクトを1つ持ち、このオブジェクトにコンテンツを表すほかのビューがすべて含まれています。1つのルートビューを使用することでインターフェイスを変更するプロセスは簡略化され、新しいコンテンツを表示するために必要な操作はルートビューを置き換えることだけになります。ウインドウにビューを追加するには、`addSubview:`メソッドを使用します。たとえば、ビューコントローラによって管理されるビューを追加するには、以下のようなコードを使用します。

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

上のコードの代わりに、nibファイルでウインドウの`rootViewController`プロパティを設定することもできます。このプロパティを使用すると、プログラムの代わりにnibファイルを使用してウインドウのルートビューを設定できます。ウインドウがnibファイルからロードされるときにこのプロパティが設定されていると、UIKitは関連付けられたビューコントローラから自動的にビューをウインドウのルートビューとして追加します。このプロパティはルートビューを追加するためにだけ使用され、ビューコントローラと通信するためにウインドウが使用することはありません。

ウインドウのルートビューには、いずれのビューでも使用できます。インターフェイスの設計に応じて、ルートビューは1つ以上のサブビューのコントローラとして動作する汎用の`UIView`オブジェクト、標準のシステムビュー、または各自で定義したカスタムビューのいずれかになります。一般的にルートビューとして使用される標準のシステムビューには、スクロールビュー、テーブルビュー、画像ビューなどがあります。

ウインドウのルートビューを設定するとき、ウインドウ内でビューの初期サイズと位置を設定する必要があります。ステータスバーを含まないアプリケーション、または半透明のステータスバーを表示するアプリケーションでは、ビューのサイズをウインドウのサイズに一致するように設定します。不透明なステータスバーを表示するアプリケーションでは、ビューをステータスバーの下に配置し、サイズを適切に縮小します。ステータスバーの高さ分だけ小さくすることで、ビューの上部がステータスバーに隠されるのを防ぎます。

> **注意:** ウインドウのルートビューがコンテナビューコントローラ（タブバーコントローラ、ナビゲーションコントローラ、分割ビューコントローラなど）によって提供される場合は、ビューの初期サイズを各自で設定する必要はありません。ステータスバーの表示設定に基づいて、コンテナビューコントローラが自動的にビューのサイズを設定します。

ウインドウレベルの変更

各`UIWindow`オブジェクトには、設定可能な`windowLevel`プロパティがあります。このプロパティは、そのウインドウをほかのウインドウに対してどのように配置するかを決定します。ほとんどの場合、アプリケーションのウインドウのレベルは変更する必要がありません。新しいウインドウは、作成時に通常のウインドウレベルに自動的に関連付けられます通常のウインドウレベルは、ウインドウがアプリケーション関連のコンテンツを表示することを示します。これより上位のウインドウレベルは、アプリケーションコンテンツの前面に表示する必要がある情報（システムステータスバーや警告メッセージ）用に予約されています。ウインドウをこれらのレベルに割り当てることもできますが、通常この処理は特定のインターフェイスを使用するときにシステムによって行われます。たとえば、ステータスバーの表示と非表示を切り替えたり、警告ビューを表示するときに、システムは自動的に必要なウインドウを作成し、これらの項目を表示します。

ウインドウの変更の監視

ウインドウの表示と非表示の切り替えをアプリケーション内で追跡する場合は、これらのウインドウに関する通知を使用します。

- `UIWindowDidBecomeVisibleNotification`
- `UIWindowDidBecomeHiddenNotification`
- `UIWindowDidBecomeKeyNotification`
- `UIWindowDidResignKeyNotification`

これらの通知は、アプリケーションのウインドウでのプログラムによる変更に応じて送信されます。したがって、アプリケーションがウインドウの表示と非表示を切り替えるときには、`UIWindowDidBecomeVisibleNotification`および`UIWindowDidBecomeHiddenNotification`の通知が動作に応じて送信されます。アプリケーションがバックグラウンドの実行状態に移るとき、これらの通知は送信されません。アプリケーションがバックグラウンドで実行され、ウインドウが画面に表示されていない場合でも、アプリケーションのコンテキストではまだ表示されていると見なされます。

`UIWindowDidBecomeKeyNotification`および`UIWindowDidResignKeyNotification`の通知は、どのウインドウが**キーウインドウ**であるかを追跡する、つまり現在キーボードのイベントやその他のタッチ関連以外のイベントを受信しているウインドウを追跡するのに役立ちます。タッチイベントはタッチが発生したウインドウに送信されますが、座標値に関連付けられていないイベントはアプリケーションのキーウインドウに送信されます。キーウインドウになれるのは、常に1つのウインドウだけです。

外部ディスプレイでのコンテンツの表示

コンテンツを外部ディスプレイで表示する場合は、アプリケーションの追加ウインドウを作成し、これを外部ディスプレイを表す画面オブジェクトに関連付ける必要があります。通常、新しいウインドウはデフォルトでメイン画面に関連付けられます。ウインドウの関連付けられた画面オブジェクトを変更すると、そのウインドウのコンテンツは対応するディスプレイに送られます。ウインドウを正しい画面に関連付けた後は、アプリケーションのメイン画面の場合と同様に、ウインドウにビューを追加して表示することができます。

`UIScreen`クラスは、利用可能なハードウェアディスプレイを表す画面オブジェクトのリストを管理します。通常、iOSベースのデバイスはメインディスプレイを表す画面オブジェクトを1つだけ持ちますが、外部ディスプレイへの接続をサポートするデバイスは、利用可能な追加の画面オブジェクトを持つ場合があります。外部ディスプレイをサポートするデバイスには、Retinaディスプレイを備えたiPhoneおよびiPod touchデバイスやiPadがあります。iPhone 3GSなどの古いデバイスは、外部ディスプレイをサポートしません。

> **注意:** 外部ディスプレイは本質的にビデオ出力の接続であるため、外部ディスプレイに関連付けられるウインドウでは、ビューおよびコントロールのタッチイベントを期待すべきではありません。また、ウインドウのコンテンツを必要に応じて更新しなければなりません。したがって、メインウインドウのコンテンツをミラーリングする場合は、外部ディスプレイのウインドウ用に複製されたビューセットを作成し、これをメインウインドウのビューと一緒に更新する必要があります。

外部ディスプレイにコンテンツを表示するプロセスは後のセクションで説明することにして、以下の手順で基本プロセスを要約します。

1. アプリケーションの起動時に、画面の接続と接続解除の通知を登録します。
2. 外部ディスプレイにコンテンツを表示するときに、ウインドウを作成して設定します。
    - `UIScreen`の`screens`プロパティを使用して、外部ディスプレイの画面オブジェクトを取得します。
    - `UIWindow`オブジェクトを作成して画面（またはコンテンツ）に合ったサイズに変更します。

- 外部ディスプレイのUIScreenオブジェクトを、ウインドウのscreenプロパティに割り当てます。
- コンテンツをサポートするために必要な場合は、画面オブジェクトの解像度を調整します。
- ウインドウに適切なビューを追加します。

3. ウインドウを表示し、通常どおり更新します。

画面の接続および接続解除に関する通知の処理

画面の接続と接続解除に関する通知は、外部ディスプレイへの変更を適切に処理するために重要です。ユーザがディスプレイを接続または接続解除すると、システムは適切な通知をアプリケーションに送信します。これらの通知を使用してアプリケーションの状態を更新し、外部ディスプレイに関連付けられたウインドウを作成または解放します。

接続と接続解除の通知に関して重要な点は、アプリケーションがバックグラウンドで一時停止されているときでも、これらの通知が常に送信されてくることです。したがって、アプリケーションの実行時に存在する予定のオブジェクト（たとえば、アプリケーションデリゲート）からの通知を監視するのが最善の方法です。アプリケーションが一時停止している場合は、アプリケーションの停止状態が終了し、フォアグラウンドまたはバックグラウンドで実行が開始されるまで、通知はキューに入れられます。

リスト 2-1に、接続および接続解除の通知を登録するためのコードを示します。このメソッドは、初期化時にアプリケーションデリゲートによって呼び出されますが、アプリケーションの別の場所からでもこれらの通知を登録できます。ハンドラメソッドの実装は、リスト 2-2 （32 ページ）に示しています。

リスト 2-1　画面の接続および接続解除に関する通知の登録

```
- (void)setupScreenConnectionNotificationHandlers
{
    NSNotificationCenter* center = [NSNotificationCenter defaultCenter];

    [center addObserver:self selector:@selector(handleScreenConnectNotification:)
            name:UIScreenDidConnectNotification object:nil];
   [center addObserver:self selector:@selector(handleScreenDisconnectNotification:)
            name:UIScreenDidDisconnectNotification object:nil];
}
```

外部ディスプレイがデバイスに接続されたときにアプリケーションがアクティブな場合は、そのディスプレイ用に2つ目のウインドウを作成し、何らかのコンテンツを表示する必要があります。この場合のコンテンツは、表示したい最終的なコンテンツである必要はありません。たとえば、アプリケーションで追加画面を使用するための準備が完了していない場合は、2つ目のウインドウを使用してプレースホルダコンテンツを表示できます。画面用のウインドウを作成しない場合、またはウインドウを作成してもそれを表示しない場合、外部ディスプレイには黒色の画面が表示されます。

リスト2-2に、2つ目のウインドウを作成してコンテンツを表示する方法を示します。この例では、ハンドラメソッドで、画面の接続通知を受け取るために使用するウインドウを作成します（接続と接続解除の通知の登録については、リスト 2-1 （31 ページ）を参照）。接続通知のハンドラメソッドが2つ目のウインドウを作成し、これを新しく接続された画面に関連付けます。続いて、アプリケーションのメインビューコントローラのメソッドを呼び出し、ウインドウにコンテンツを追加して表示します。接続解除の通知のハンドラメソッドはウインドウを解放し、それに合わせて表示を調整できるようにメインビューコントローラに通知します。

リスト 2-2　接続および接続解除に関する通知の処理

```
- (void)handleScreenConnectNotification:(NSNotification*)aNotification
{
    UIScreen*     newScreen = [aNotification object];
    CGRect         screenBounds = newScreen.bounds;

    if (!_secondWindow)
    {
        _secondWindow = [[UIWindow alloc] initWithFrame:screenBounds];
        _secondWindow.screen = newScreen;

        // Set the initial UI for the window.
        [viewController displaySelectionInSecondaryWindow:_secondWindow];
    }
}

- (void)handleScreenDisconnectNotification:(NSNotification*)aNotification
{
    if (_secondWindow)
    {
        // Hide and then delete the window.
```

```
        _secondWindow.hidden = YES;
        [_secondWindow release];
        _secondWindow = nil;

        // Update the main screen based on what is showing here.
        [viewController displaySelectionOnMainScreen];
    }

}
```

外部ディスプレイ用のウインドウの設定

ウインドウを外部画面に表示するには、ウインドウを正しい画面オブジェクトに関連付ける必要があります。このプロセスでは、適切なUIScreenオブジェクトを決定し、このオブジェクトをウインドウのscreenプロパティに割り当てます。画面オブジェクトのリストは、screensのUIScreenクラスメソッドから取得できます。このメソッドから返される配列には常に、メイン画面を表すオブジェクトが少なくとも1つ含まれます。ほかのオブジェクトがある場合、そのオブジェクトは接続された外部ディスプレイを表します。

リスト 2-3に、外部ディスプレイが接続されているかどうかを確認するために、アプリケーションの起動時に呼び出されるメソッドを示します。外部ディスプレイが接続されている場合、メソッドはウインドウを作成して外部ディスプレイに関連付け、ウインドウを表示する前にプレースホルダコンテンツを追加します。この例では、プレースホルダコンテンツは白色の背景と、表示するコンテンツがないことを示すラベルで構成されています。ウインドウを表示するために、このメソッドはhiddenを呼び出す代わりに、makeKeyAndVisibleプロパティを変更しています。これは、ウインドウに含まれているのが静的なコンテンツだけで、ウインドウがイベントの処理に使用されないためです。

リスト 2-3　外部ディスプレイ用のウインドウの設定

```
- (void)checkForExistingScreenAndInitializeIfPresent
{
    if ([[UIScreen screens] count] > 1)
    {
        // Associate the window with the second screen.
        // The main screen is always at index 0.
        UIScreen*    secondScreen = [[UIScreen screens] objectAtIndex:1];
        CGRect        screenBounds = secondScreen.bounds;
```

```
        _secondWindow = [[UIWindow alloc] initWithFrame:screenBounds];
        _secondWindow.screen = secondScreen;

        // Add a white background to the window
       UIView*           whiteField = [[UIView alloc] initWithFrame:screenBounds];
        whiteField.backgroundColor = [UIColor whiteColor];

        [_secondWindow addSubview:whiteField];
        [whiteField release];

        // Center a label in the view.
       NSString*    noContentString = [NSString stringWithFormat:@"<no content>"];
        CGSize        stringSize = [noContentString sizeWithFont:[UIFont
systemFontOfSize:18]];

        CGRect        labelSize = CGRectMake((screenBounds.size.width -
stringSize.width) / 2.0,
                                     (screenBounds.size.height - stringSize.height)
 / 2.0,
                                      stringSize.width, stringSize.height);

        UILabel*    noContentLabel = [[UILabel alloc] initWithFrame:labelSize];
        noContentLabel.text = noContentString;
        noContentLabel.font = [UIFont systemFontOfSize:18];
        [whiteField addSubview:noContentLabel];

        // Go ahead and show the window.
        _secondWindow.hidden = NO;
    }
}
```

Important: ウインドウを表示する前に、常に画面をウインドウに関連付ける必要があります。現在表示しているウインドウの画面を変更することもできますが、この操作は負荷が大きいため避けてください。

外部画面のウインドウが表示されたらすぐに、ほかのウインドウと同様に更新を開始できます。必要に応じてサブビューの追加と削除、サブビューのコンテンツの変更、ビューに対する変更のアニメーション化、およびコンテンツの無効化を実行できます。

外部ディスプレイの画面モードの設定

コンテンツに応じて、ウインドウを画面に関連付ける前に画面モードを変更できます。多くの画面は複数の解像度をサポートし、一部の解像度ではピクセルのアスペクト比が異なる場合があります。画面オブジェクトはデフォルトで最も一般的な画面モードを使用しますが、コンテンツに適した画面モードに変更することができます。たとえば、OpenGL ESを使用するゲームを実装しているときに、テクスチャが640x480ピクセルの画面に合わせて設計されている場合、デフォルト解像度がより高い画面では画面モードを変更できます。

デフォルト以外の画面モードの使用を計画している場合は、画面をウインドウに割り当てる前に、使用するモードを`UIScreen`オブジェクトに適用する必要があります。`UIScreenMode`クラスは、単一の画面モードの属性を定義します。画面でサポートされるモードのリストは、`availableModes`プロパティから取得でき、リストから要求を満たすモードを選択できます。

画面モードの詳細については、『*UIScreenMode Class Reference*』を参照してください。

ビュー

ビューオブジェクトはアプリケーションがユーザとやり取りを行う主な手段であるため、さまざまな役割を担います。たとえば、以下のような処理を行います。

- レイアウトとサブビューの管理
  - ビューは、デフォルトのサイズ変更動作を親ビューとの関連において独自に定義します。
  - ビューはサブビューのリストを管理できます。
  - ビューは必要に応じてサブビューのサイズと位置をオーバーライドできます。
  - ビューは自身の座標系で表した点を、ほかのビューまたはウインドウの座標系に変換できます。
- 描画とアニメーション
  - ビューはコンテンツを矩形領域に描画します。
  - 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。
- イベント処理
  - ビューはタッチイベントを受信できます。
  - ビューはレスポンダチェーンに参加します。

この章では、ビューの作成、管理、および描画に関する手順と、ビュー階層のレイアウト操作および管理操作に関する手順について説明します。ビューでタッチイベント（およびその他のイベント）を処理する方法については、『*Event Handling Guide for iOS*』を参照してください。

ビューオブジェクトの作成と設定

ビューはプログラムまたはInterface Builderで自己完結型のオブジェクトとして作成し、これらをビュー階層にまとめて使用します。

Interface Builderを使用したビューオブジェクトの作成

ビューを作成するには、Interface Builderを使用して視覚的に組み立てる方法が最も簡単です。Interface Builderでは、ビューをインターフェイスに追加して、これらのビューを階層にまとめることができます。また、各ビューの設定を調整し、ビューに関連する動作をコードに関係付けることもできます。

Interface Builderは実際に有効なビューオブジェクト（ビュークラスの実際のインスタンス）を使用するため、設計時に表示される内容は実行時の結果と同じになります。続いて、これらの有効なオブジェクトをnibファイルに保存します。このファイルは、オブジェクトの状態と設定を保存するリソースファイルです。

通常は、アプリケーションのいずれかのビューコントローラのビュー階層全体を保存するために、nibファイルを作成します。nibファイルの最上位には、通常ビューコントローラのビューを表すビューオブジェクトが1つ含まれます（ビューコントローラ自体は、一般的にFile's Ownerオブジェクトによって表されます）。最上位のビューは、ターゲットデバイスに合わせて適切にサイズ設定され、表示されるほかのビューをすべて含みます。ビューコントローラのビュー階層の一部だけを保存するためにnibファイルを使用することは、ほとんどありません。

ビューコントローラでnibファイルを使用するときに必要な処理は、nibファイルの情報を使用してビューコントローラを初期化することだけです。ビューコントローラは、ビューのロードとアンロードを適切なタイミングで処理します。ただし、nibファイルがビューコントローラに関連付けられていない場合は、`NSBundle`または`UINib`オブジェクトを使用して、nibファイルの内容を手動でロードできます。これらのオブジェクトは、nibファイルのデータを使用してビューオブジェクトを再編成します。

Interface Builderを使用してビューを作成および設定する方法については、『*Interface Builder User Guide*』を参照してください。関連付けられたnibファイルをビューコントローラがロードおよび管理する方法については、『*View Controller Programming Guide for iOS*』の「Creating Custom Content View Controllers」を参照してください。プログラムでビューをnibファイルをからロードする方法については、『*Resource Programming Guide*』の「Nib Files」を参照してください。

プログラムによるビューオブジェクトの作成

ビューをプログラムで作成する場合は、標準的な割り当てと初期化のパターンを使用します。ビューのデフォルトの初期メソッドは`initWithFrame:`メソッドです。このメソッドは、ビューの初期サイズと位置を（すぐ後に設定される）親ビューを基準に設定します。たとえば、新しい汎用の`UIView`オブジェクトを作成する場合は、以下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];
```

> **注意:** すべてのビューは`initWithFrame:`メソッドをサポートしますが、一部のビューには優先される初期化メソッドがあり、この場合は優先メソッドを使用する必要があります。カスタム初期化メソッドの詳細については、対象のクラスのリファレンスドキュメントを参照してください。

作成したビューを表示するには、そのビューをウインドウ（または、ウインドウ内の別のビュー）に追加する必要があります。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” （41 ページ）を参照してください。

ビューのプロパティの設定

`UIView`クラスには、ビューの外観や動作を制御する宣言済みプロパティがあります。これらのプロパティは、ビューのサイズと位置、ビューの透明度、背景色、およびレンダリングの動作を操作するためのものです。これらのプロパティはすべてデフォルト値を持ち、必要に応じて後から変更できます。これらのプロパティの多くは、インスペクタウインドウを使用してInterface Builderから設定することもできます。

表 3-1に、一般的に使用されるプロパティ（およびメソッド）とその使用方法を示します。ビューの特定の面について制御できる内容を確認できるように、関連のあるプロパティを並べて示しています。

表 3-1 主なビューのプロパティの使用方法

プロパティ	用途
`alpha`、`hidden`、`opaque`	これらのプロパティはビューの透明度に影響します。`alpha`および`hidden`プロパティは、ビューの透明度を直接変更します。 `opaque`プロパティは、ビューを合成する方法をシステムに指示します。ビューのコンテンツが完全に不透明で、これより下にあるビューのコンテンツを何も表示しない場合は、このプロパティをYESに設定します。このプロパティをYESに設定すると、不要な合成操作が実行されないためパフォーマンスが向上します。

プロパティ	用途
bounds、frame、center、transform	これらのプロパティはビューのサイズと位置に影響します。centerおよびframeプロパティは、ビューの位置を親ビューを基準に表します。frameには、ビューのサイズも含まれます。boundsプロパティは、ビューに表示するコンテンツ領域を独自の座標系で定義します。 transformプロパティは、ビュー全体の複雑なアニメーション化または移動に使用されます。たとえば、ビューを回転または拡大縮小するにはtransformによる変換を使用します。現在の変換が恒等変換でない場合、frameプロパティは未定義となり、無視する必要があります。 bounds、frame、およびcenterプロパティの関係については、“frame、bounds、およびcenterプロパティの関係”（16 ページ）を参照してください。ビューに対する変換の影響については、“座標系の変換”（17 ページ）を参照してください。
autoresizingMask、autoresizesSubviews	これらのプロパティは、ビューとそのサブビューの自動サイズ変更の動作に影響します。autoresizingMaskプロパティは、親ビューの境界が変更されたときのビューの動作を制御します。autoresizesSubviewsプロパティは、現在のビューのサブビューをサイズ変更するかどうかを制御します。
contentMode、contentStretch、contentScaleFactor	これらのプロパティは、ビュー内部でのコンテンツのレンダリング動作に影響します。contentModeおよびcontentStretchプロパティは、ビューの幅または高さが変更されたときのコンテンツの処理方法を決定します。contentScaleFactorプロパティは、高解像度の画面のビューの描画動作をカスタマイズする必要がある場合にだけ使用します。 ビューに対するコンテンツモードの影響については、“コンテンツモード”（13 ページ）を参照してください。ビューに対するコンテンツの引き延ばし矩形の影響については、“引き延ばし可能なビュー”（13 ページ）を参照してください。拡大縮小については、『*Drawing and Printing Guide for iOS*』の「Supporting High-Resolution Screens In Views」を参照してください。
gestureRecognizers、userInteraction-Enabled、multipleTouchEnabled、exclusiveTouch	これらのプロパティは、ビューがタッチイベントを処理する方法に影響します。gestureRecognizersプロパティには、ビューにアタッチするジェスチャリコグナイザが含まれます。その他のプロパティは、ビューでサポートされるタッチイベントを制御します。 ビューでイベントに応答する方法については、『*Event Handling Guide for iOS*』を参照してください。

プロパティ	用途
backgroundColor、subviews、drawRect:メソッド、layer（layerClassメソッド）	これらのプロパティとメソッドは、ビューの実際のコンテンツを管理するのに役立ちます。簡単なビューでは、背景色を設定し、1つ以上のサブビューを追加できます。subviewsプロパティ自体にサブビューの読み出し専用のリストが含まれますが、サブビューを追加および再配置するためのメソッドがいくつかあります。カスタム描画を行うビューでは、drawRect:メソッドをオーバーライドする必要があります。 より高度なコンテンツでは、ビューのCore Animationのlayerを直接操作できます。ビューにタイプがまったく異なるレイヤを指定するには、layerClassメソッドをオーバーライドする必要があります。

すべてのビューに共通の基本的なプロパティについては、『*UIView Class Reference*』を参照してください。ビューに固有のプロパティについては、各ビューのリファレンスドキュメントを参照してください。

タグによるビューの識別

UIViewクラスに含まれるtagプロパティを使用して、個々のビューオブジェクトに整数値のタグを設定できます。タグを使用することで、ビュー階層内でビューを一意に識別し、実行時にこれらのビューを検索することができます（タグベースの検索は、ビュー階層を反復的に検索するよりも高速です）。tagプロパティのデフォルト値は0です。

タグを付けたビューを検索するには、viewWithTag:のUIViewメソッドを使用します。このメソッドは、レシーバーとそのサブビューを深さ優先で検索します。スーパービューやビュー階層のほかの部分は検索しません。したがって、このメソッドを階層のルートビューから呼び出すと、階層内のすべてのビューが検索対象になりますが、特定のサブビューから呼び出した場合は、ビューのサブセットのみが検索対象となります。

ビュー階層の作成と管理

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を開発する上で重要な部分です。ビューの編成は、アプリケーションの外観と、変更やイベントに対するアプリケーションの動作の両方に影響します。たとえば、特定のタッチイベントを処理するオブジェクトは、ビュー階層における親子関係によって決定されます。同様に、インターフェイスの向きが変更されたときの各ビューの動作も、親子関係によって定義されます。

図 3-1に、ビューを階層化してアプリケーションで目的の視覚効果を得る方法の例を示します。「時計(Clock)」アプリケーションの場合、ビュー階層に機能の異なるビューが混在しています。タブバーとナビゲーションのビューは、タブバーコントローラおよびナビゲーションコントローラオブジェクトで提供される特殊なビュー階層で、それぞれユーザインターフェイスの一部分を管理します。これらのバーの間にあるものはすべて、「時計(Clock)」アプリケーションが提供するカスタムビュー階層に属しています。

図 3-1　「時計(Clock)」アプリケーションにおけるビューの階層化

iOSアプリケーションのビュー階層は、Interface Builderを使用して視覚的に作成したり、コード内でプログラムによって作成するなど、さまざまな方法で構築できます。以降のセクションでは、ビュー階層を構築する方法、および構築したビュー階層でビューを検索して別のビュー座標系に変換する方法について説明します。

サブビューの追加と削除

Interface Builderを使用すると、ビュー階層を最も簡単に構築できます。Interface Builderでは、ビューを視覚的に整理し、ビュー間の関係を確認して、実行時にビューがどのように表示されるかを正確に確認できます。Interface Builderを使用する場合は、作成したビュー階層をnibファイルに保存し、実行時に対応するビューが必要になったときに、このファイルをロードします。

ビューをプログラムで作成する場合は、ビューを作成して初期化した後に、以下のメソッドを使用してビューを階層に配置します。

- サブビューを親に追加する場合は、親ビューの`addSubview:`メソッドを呼び出します。このメソッドは、親のサブビューのリストの末尾にサブビューを追加します。
- 親のサブビューのリストの中間にサブビューを挿入する場合は、親ビューの`insertSubview:...`メソッドのいずれかを呼び出します。リストの中間に挿入したサブビューは、リストでそれより後にあるビューの背面に配置されます。
- 既存のサブビューを親ビュー内で並べ替える場合は、親ビューの`bringSubviewToFront:`、`sendSubviewToBack:`、または`exchangeSubviewAtIndex:withSubviewAtIndex:`メソッドを呼び出します。これらのメソッドは、サブビューを削除して再度挿入するよりも高速です。
- サブビューを親ビューから削除する場合は、サブビューの`removeFromSuperview`メソッドを呼び出します（親ビューのメソッドではありません）。

サブビューを親に追加する場合、サブビューの現在のフレーム矩形は親ビュー内での初期位置を示します。フレームがスーパービューの表示境界より外にあるサブビューは、デフォルトでは境界に合わせて切り取られません。サブビューをスーパービューの境界に合わせて切り取る場合は、スーパービューの`clipsToBounds`プロパティを明示的に`YES`に設定する必要があります。

ほとんどのアプリケーションにあるapplication:didFinishLaunchingWithOptions:メソッドは、サブビューを別のビューに追加する最も一般的な例です。リスト3-1では、このメソッドがアプリケーションのメインビューコントローラからアプリケーションウインドウにビューを追加しています。ウインドウとビューコントローラはどちらもアプリケーションのメインnibファイルに格納され、このファイルはメソッドが呼び出される前にロードされます。ただし、ビューコントローラで管理されるビュー階層は、viewプロパティが呼び出されるまで実際にはロードされません。

リスト 3-1　ウインドウへのビューの追加

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Override point for customization after application launch.

    // Add the view controller's view to the window and display.
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];

    return YES;
}
```

サブビューをビュー階層に追加する一般的な方法としては、そのほかにビューコントローラのloadViewまたはviewDidLoadメソッドがあります。ビューをプログラムで構築している場合は、ビューの作成コードをビューコントローラのloadViewメソッドに記述します。ビューをプログラムで作成するかnibファイルからロードするかにかかわらず、viewDidLoadメソッドにはビューの設定コードを追加できます。

リスト 3-2に、『*UICatalog*』のサンプルアプリケーションで使用されているTransitionsViewControllerクラスのviewDidLoadメソッドを示します。TransitionsViewControllerクラスは、2つのビュー間のトランジションに関するアニメーションを管理します。アプリケーションの（ルートビューとツールバーで構成される）初期ビュー階層は、nibファイルからロードされます。続いて、viewDidLoadメソッドのコードが、トランジションを管理するためのContainer Viewと画像ビューを作成します。Container Viewの目的は、2つの画像ビューの間のトランジションアニメーションを実装するために必要なコードを簡素化することです。Container View自体は、実際のコンテンツを持ちません。

リスト 3-2　既存のビュー階層へのビューの追加

```
- (void)viewDidLoad
```

```
{
    [super viewDidLoad];

    self.title = NSLocalizedString(@"TransitionsTitle", @"");

   // create the container view which we will use for transition animation (centered
 horizontally)
    CGRect frame = CGRectMake(round((self.view.bounds.size.width - kImageWidth) /
 2.0),
                                                        kTopPlacement, kImageWidth,
 kImageHeight);
    self.containerView = [[[UIView alloc] initWithFrame:frame] autorelease];
    [self.view addSubview:self.containerView];

    // The container view can represent the images for accessibility.
    [self.containerView setIsAccessibilityElement:YES];
    [self.containerView setAccessibilityLabel:NSLocalizedString(@"ImagesTitle",
@"")];

    // create the initial image view
    frame = CGRectMake(0.0, 0.0, kImageWidth, kImageHeight);
    self.mainView = [[[UIImageView alloc] initWithFrame:frame] autorelease];
    self.mainView.image = [UIImage imageNamed:@"scene1.jpg"];
    [self.containerView addSubview:self.mainView];

    // create the alternate image view (to transition between)
    CGRect imageFrame = CGRectMake(0.0, 0.0, kImageWidth, kImageHeight);
   self.flipToView = [[[UIImageView alloc] initWithFrame:imageFrame] autorelease];
    self.flipToView.image = [UIImage imageNamed:@"scene2.jpg"];
}
```

Important: スーパービューは自動的にサブビューを保持します。したがって、サブビューを埋め込んだ後は、そのサブビューを解放しても安全です。実際に、アプリケーションがビューを過剰に保持して後でメモリリークが発生するのを避けるために、サブビューを解放することをお勧めします。ただし、サブビューをスーパービューから削除した後に再利用する予定の場合は、サブビューを再度保持する必要があることに注意してください。`removeFromSuperview`メソッドは、サブビューをスーパービューから削除する前に、サブビューを自動的に解放します。次のイベントループサイクルまでにビューを保持しない場合は、ビューが解放されます。

Cocoaのメモリ管理方法については、『*Advanced Memory Management Programming Guide*』を参照してください。

サブビューを別のビューに追加するときに、UIKitは親ビューと子ビューの両方に変更を通知します。カスタムビューを実装する場合は、`willMoveToSuperview:`、`willMoveToWindow:`、`willRemoveSubview:`、`didAddSubview:`、`didMoveToSuperview`、または`didMoveToWindow`メソッドの1つ以上をオーバーライドすることで、これらの通知をインターセプトできます。これらの通知を使用して、ビュー階層に関する状態情報を更新したり、追加タスクを実行することができます。

作成したビュー階層は、ビューの`superview`および`subviews`プロパティを使用してプログラムでナビゲートすることができます。各ビューの`window`プロパティには、ビューが現在表示されているウインドウが含まれます（存在する場合）。ビュー階層のルートビューには親がないため、ルートビューの`superview`プロパティは`nil`に設定されます。現在画面に表示されているビューの場合は、ウインドウオブジェクトがビュー階層のルートビューです。

ビューの非表示

ビューを非表示にするには、`hidden`プロパティを`YES`に設定するか、`alpha`プロパティを`0.0`に変更します。非表示のビューは、システムからタッチイベントを受信しません。ただし、ビュー階層に関する自動サイズ変更とその他のレイアウト操作は、非表示のビューにも関係します。したがって、特にビューを後でもう一度表示する予定の場合、通常はビュー階層からビューを削除する代わりにビューを非表示にする方が便利です。

Important: 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する場合、ビューはファーストレスポンダの状態を自動的に放棄しません。ファーストレスポンダをターゲットにしたイベントは、非表示にしたビューにこれまでどおり送信されます。これを避けるには、ビューを非表示にするときにファーストレスポンダの状態を強制的に放棄させます。これらのレスポンダチェーンの詳細については、『*Event Handling Guide for iOS*』を参照してください。

ビューのトランジションを表示から非表示（またはその逆）にアニメーション化する場合は、ビューの`alpha`プロパティを使用する必要があります。`hidden`プロパティはアニメーション化可能なプロパティではありません。したがって、変更した設定はただちに有効になります。

ビュー階層でのビューの検索

ビュー階層でビューを検索するには、2つの方法があります。

- 関連するビューへのポインタを適切な場所（ビューを所有するビューコントローラなど）に保存します。
- 各ビューの`tag`プロパティに一意の整数値を割り当て、`viewWithTag:`メソッドを使用して検索します。

関連するビューの参照の保存は、ビューを検索する最も一般的な方法で、ビューにアクセスする場合に便利です。Interface Builderを使用してビューを作成した場合は、アウトレットを使用して、nibファイルのオブジェクト（管理用のコントローラオブジェクトを表すFile's Ownerオブジェクトを含む）を別のオブジェクトに接続できます。プログラムで作成するビューでは、ビューへの参照をプライベートなメンバー変数に保存できます。アウトレットまたはプライベートなメンバー変数のどちらを使用する場合でも、デベロッパは必要に応じてビューを保持し、その後でビューを解放する必要があります。オブジェクトの保持と解放を適切に行うには、宣言済みプロパティを使用するのが最善の方法です。

タグは、ハードコードされた依存関係を減らす有効な方法で、より動的で柔軟なソリューションをサポートします。ビューへのポインタを保存する代わりに、タグを使用してビューを検索できます。タグは、ビューの参照方法としてより永続的でもあります。たとえば、現在アプリケーションに表示しているビューのリストを保存する場合は、表示している各ビューのタグをファイルに書き出します。この方法は、実際のビューオブジェクトをアーカイブするよりも簡単です。特に、現在表示しているビューを追跡しているだけの場合に有効な方法です。アプリケーションが後でロードされるときに、ビューを再作成し、保存したタグのリストを使用して各ビューの表示設定を行うことで、ビュー階層は以前の状態に戻ります。

ビューの平行移動、拡大縮小、回転

すべてのビューには、ビューのコンテンツを平行移動、拡大縮小、または回転するために使用できる、アフィン変換が関連付けられています。ビューの変換は、レンダリングされたビューの最終的な外観を変更し、多くの場合はスクロール、アニメーション、またはその他の視覚効果を実装するために使用されます。

`UIView`の`transform`プロパティには、適用する変換を表す`CGAffineTransform`構造体が含まれます。デフォルトでは、このプロパティはビューの外観を変更しない恒等変換に設定されます。このプロパティには、いつでも新しい変換を割り当てることができます。たとえば、ビューを45度回転させるには、以下のコードを使用できます。

```
// M_PI/4.0 is one quarter of a half circle, or 45 degrees.
CGAffineTransform xform = CGAffineTransformMakeRotation(M_PI/4.0);
```

```
self.view.transform = xform;
```

上のコードでビューに変換を適用すると、ビューが中心点を基準に時計回りに回転されます。図 3-2は、この変換をアプリケーションに埋め込まれた画像ビューに適用した場合の様子を示しています。

図 3-2　ビューの45度回転

1つのビューに複数の変換を適用する場合は、変換を`CGAffineTransform`構造体に追加する順序が重要になります。ビューを回転してから平行移動するのは、ビューを平行移動してから回転するのと同じではありません。回転と平行移動の量がそれぞれ同じでも、一連の変換が最終結果に影響します。また、追加する変換はいずれも、ビューの中心点を基準にして適用されます。したがって、回転を適用すると、ビューは中心点を基準にして回転します。ビューを拡大縮小すると、ビューの幅と高さは変わりますが、中心点は変わりません。

アフィン変換の作成と使用方法については、『*Quartz 2D Programming Guide*』の「Transforms」を参照してください。

ビュー階層での座標の変換

さまざまな状況で（特にイベントを処理する場合に）、アプリケーションでは座標値を異なる座標系の間で変換する必要が生じます。たとえば、タッチイベントは各タッチの位置をウインドウの座標系で報告しますが、多くの場合、ビューオブジェクトではビューのローカル座標系に基づいた情報が必要となります。`UIView`クラスには、座標をビューのローカル座標系との間で変換するための、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:`メソッドは、座標を別のビューの座標系から現在のビューのローカル座標系（境界矩形）に変換します。`convert...:toView:`メソッドは、これとは逆に、座標を現在のビューの座標系（境界矩形）から指定したビューの座標系に変換します。いずれのメソッドでも参照ビューに`nil`を指定すると、ビューを含んでいるウインドウの座標系との間で座標が変換されます。

`UIView`の変換メソッドのほかに、`UIWindow`クラスで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`UIView`のメソッドとほぼ同じですが、ビューのローカル座標系との間で変換を行う代わりに、ウインドウの座標系との間で変換を行います。

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

回転されたビューの座標を変換する場合、UIKitは変換された矩形が元の矩形の占める画面領域を反映するように変換を行います。図 3-3に、回転によって変換中に矩形のサイズが変更される例を示します。図では、外側の親ビューに回転されたサブビューが含まれています。サブビューの座標系にある矩形を親の座標系に変換すると、矩形が物理的に大きくなります。この大きい矩形は、`outerView`の境界内の、回転された矩形が完全に収まる最小の矩形です。

図 3-3 回転したビューの値の変換

実行時のビューのサイズと位置の調整

ビューのサイズを変更する場合は、それに合わせて必ずサブビューのサイズと位置を変更する必要があります。`UIView`クラスは、ビュー階層におけるビューの自動レイアウトおよび手動レイアウトの両方をサポートします。自動レイアウトでは、親ビューがサイズ変更されたときに各ビューが従うべき規則を設定し、その他のサイズ変更の操作は必要ありません。手動レイアウトでは、ビューのサイズと位置を必要に応じて調整します。

レイアウト変更への備え

レイアウトの変更は、ビューで以下のイベントが発生したときに常に起こります。

- ビューの境界矩形のサイズが変更される。
- インターフェイスの向きの変更が発生する。通常はルートビューの境界矩形の変更がトリガされます。
- ビューのレイヤに関連付けられたCore Animationサブレイヤのセットが変更され、レイアウト変更を必要としている。
- アプリケーションがビューの`setNeedsLayout`または`layoutIfNeeded`メソッドを呼び出して、強制的にレイアウト変更を行う。
- アプリケーションがビューの下位レベルのレイヤオブジェクトの`setNeedsLayout`メソッドを呼び出して、強制的にレイアウト変更を行う。

自動サイズ変更規則を使用した自動的なレイアウト変更の処理

ビューのサイズを変更する場合、通常は埋め込まれているサブビューの位置とサイズを親の新しいサイズに合わせて変更する必要があります。スーパービューの`autoresizesSubviews`プロパティは、サブビューをサイズ変更するかどうかを決定します。このプロパティを`YES`に設定すると、ビューは各サブビューの`autoresizingMask`プロパティを使用して、サブビューのサイズと位置を設定する方法を決定します。サブビューに対するサイズ変更は、埋め込まれているサブビューに対して同様のレイアウト調整をトリガします。

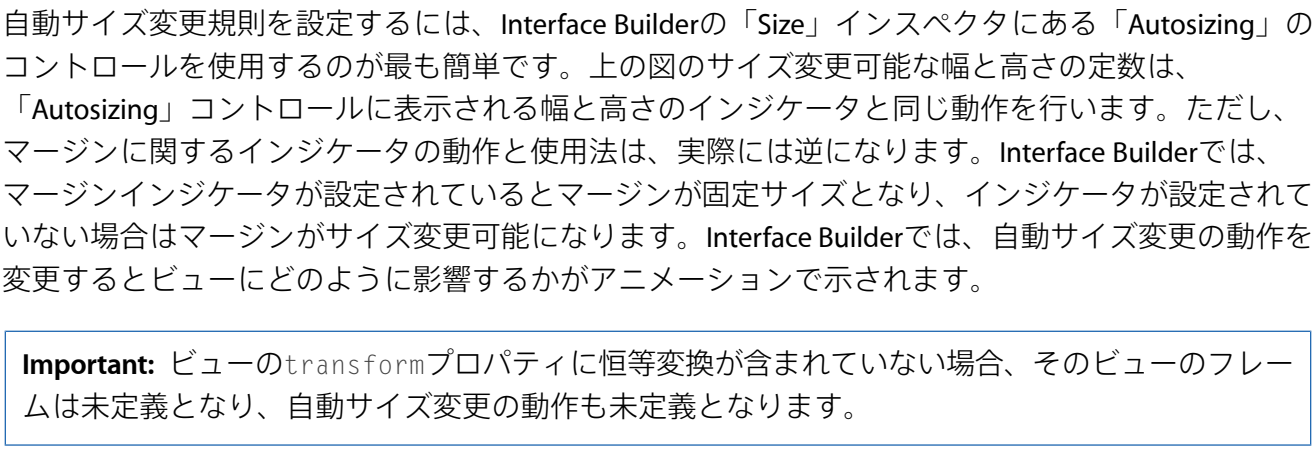
ビュー階層のビューごとに、そのビューの`autoresizingMask`プロパティを適切な値に設定することは、自動レイアウト変更の処理において重要です。表 3-2に、ビューに適用できる自動サイズ変更オプションと、レイアウト処理におけるそれぞれの効果を示します。`autoresizingMask`プロパティに定数を割り当てる前に、OR演算子を使用して定数を結合したり、単純に加算することができます。Interface Builderを使用してビューを構築している場合は、「Autosizing」インスペクタを使用して、これらのプロパティを設定できます。

表 3-2 自動サイズ変更マスク定数

自動サイズ変更マスク	解説
`UIViewAutoresizing-None`	ビューは自動サイズ変更されません（これがデフォルト値です）。
`UIViewAutoresizing-FlexibleHeight`	スーパービューの高さが変更されたときに、ビューの高さが変更されます。この定数が含まれていない場合、ビューの高さは変更されません。
`UIViewAutoresizing-FlexibleWidth`	スーパービューの幅が変更されたときに、ビューの幅が変更されます。この定数が含まれていない場合、ビューの幅は変更されません。
`UIViewAutoresizing-FlexibleLeftMargin`	ビューの左端からスーパービューの左端までの距離が、必要に応じて増減されます。この定数が含まれていない場合は、スーパービューの左端からビューの左端までの距離が固定されます。
`UIViewAutoresizing-FlexibleRightMargin`	ビューの右端からスーパービューの右端までの距離が、必要に応じて増減されます。この定数が含まれていない場合は、スーパービューの右端からビューの右端までの距離が固定されます。
`UIViewAutoresizing-FlexibleBottomMargin`	ビューの下端からスーパービューの下端までの距離が、必要に応じて増減されます。この定数が含まれていない場合は、スーパービューの下端からビューの下端までの距離が固定されます。
`UIViewAutoresizing-FlexibleTopMargin`	ビューの上端からスーパービューの上端までの距離が、必要に応じて増減されます。この定数が含まれていない場合は、スーパービューの上端からビューの上端までの距離が固定されます。

図 3-4は、自動サイズ変更マスクのオプションがビューに適用される様子を示しています。各定数が指定されている場合は、ビューの指定された部分がサイズ変更可能で、スーパービューの境界が変更されたときに変更可能であることを示します。定数が指定されていない場合は、ビューのレイアウトがその部分について固定であることを示します。1つの軸に対してサイズ変更可能な属性が2つ以上あるビューを設定する場合、UIKitは対応する空間にサイズ変更を均等に配分します。

図 3-4 ビューの自動サイズ変更マスク定数



自動サイズ変更規則を設定するには、Interface Builderの「Size」インスペクタにある「Autosizing」のコントロールを使用するのが最も簡単です。上の図のサイズ変更可能な幅と高さの定数は、「Autosizing」コントロールに表示される幅と高さのインジケータと同じ動作を行います。ただし、マージンに関するインジケータの動作と使用法は、実際には逆になります。Interface Builderでは、マージンインジケータが設定されているとマージンが固定サイズとなり、インジケータが設定されていない場合はマージンがサイズ変更可能になります。Interface Builderでは、自動サイズ変更の動作を変更するとビューにどのように影響するかがアニメーションで示されます。

Important: ビューの`transform`プロパティに恒等変換が含まれていない場合、そのビューのフレームは未定義となり、自動サイズ変更の動作も未定義となります。

影響を受けるすべてのビューに対して自動サイズ変更規則が適用された後、UIKitは各ビューにおいて、スーパービューに対する手動の調整を実行できるようにします。ビューのレイアウトを手動で管理する方法については、“手動によるビューのレイアウトの調整”（49 ページ）を参照してください。

手動によるビューのレイアウトの調整

ビューのサイズが変更されると、UIKitはビューのサブビューに自動サイズ変更を適用した後、ビューの`layoutSubviews`メソッドを呼び出して、手動による変更を実行できるようにします。自動サイズ変更による動作だけでは希望する結果が得られない場合は、カスタムビューで`layoutSubviews`メソッドを実装できます。このメソッドの実装では、以下の処理を実行できます。

- 直下のサブビューのサイズと位置を調整する。
- サブビューまたはCore Animationレイヤを追加または削除する。
- サブビューの`setNeedsDisplay`または`setNeedsDisplayInRect:`メソッドを呼び出して、サブビューを強制的に再描画する。

アプリケーションがサブビューを手動で配置する例としては、大きなスクロール可能領域を実装する場合があります。スクロール可能なコンテンツのために1つの大きなビューを使用することは現実的ではないため、多くの場合、アプリケーションは多数の小さいタイル状のビューを含むルートビューを実装します。各タイルは、スクロール可能なコンテンツの一部分を表します。スクロールイベントが発生すると、ルートビューは`setNeedsLayout`メソッドを呼び出して、レイアウトの変更を開始し

ます。続いて、`layoutSubviews`メソッドが、発生したスクロール量に基づいてタイル状のビューを再配置します。タイルがビューの表示領域からスクロールアウトすると、`layoutSubviews`メソッドはタイルをスクロールインする側の端に移動し、その過程でタイルのコンテンツを置き換えます。

レイアウトコードを作成するときには、以下の方法でコードをテストしてください。

- ビューの向きを変更したときに、サポートされるすべてのインターフェイスの向きに対してレイアウトが正しく表示されることを確認します。
- ステータスバーの高さを変更したときに、コードが適切に応答することを確認します。通話中はステータスバーの高さのサイズが大きくなり、ユーザが通話を終了するとステータスバーのサイズは小さくなります。

自動サイズ変更の動作がビューのサイズと位置にどのように影響するかについては、“自動サイズ変更規則を使用した自動的なレイアウト変更の処理”（48 ページ）を参照してください。タイル処理の実装例については、『*ScrollViewSuite*』のサンプルの例を参照してください。

実行時のビューの変更

アプリケーションはユーザから入力を受け取ると、入力に応じてユーザインターフェイスを調整します。アプリケーションは、ビューの再配置、ビューのサイズまたは位置の変更、ビューの表示と非表示の切り替え、または完全に新しいビューセットのロードによってビューを変更します。iOSアプリケーションでは、以下のさまざまな場所と場所でこれらの操作を実行します。

- ビューコントローラ：
  - ビューコントローラは、表示する前にビューを作成する必要があります。ビューをnibファイルからロードするか、プログラムで作成できます。ビューが不要になったら、これらのビューを破棄します。
  - デバイスの向きが変更されると、ビューコントローラはビューのサイズと位置を変更に合わせて調整します。新しい向きへの調整の一部として、一部のビューを非表示にしたり、別のビューを表示する場合もあります。
  - ビューコントローラが編集可能なコンテンツを管理する場合は、編集モードへの移行時および編集モードの終了時にビュー階層を調整することがあります。たとえば、さまざまなコンテンツの編集を容易にするために、ボタンやコントロールを追加する場合があります。このとき、追加されるコントロールに合わせて、既存のビューのサイズ変更が必要になる場合もあります。
- アニメーションブロック：
  - ユーザインターフェイスでビューセット間のトランジションを行う場合は、アニメーションブロック内で一方のビューを非表示にし、もう一方のビューを表示します。

    - 特殊効果を実装する場合は、アニメーションブロックを使用してビューの各種プロパティを変更します。たとえば、ビューのサイズ変更をアニメーション化する場合は、フレーム矩形のサイズを変更します。
- その他の方法：
    - タッチイベントやジェスチャが発生したときに、ユーザインターフェイスは新しいビューセットをロードしたり、現在のビューセットを変更することで応答します。イベントの処理については、『*Event Handling Guide for iOS*』を参照してください。
    - ユーザがスクロールビューを操作する場合は、スクロール可能な大きな領域でタイル状のサブビューの表示と非表示を切り替えます。スクロール可能なコンテンツの詳細については、『*Scroll View Programming Guide for iOS*』を参照してください。
    - キーボードを表示するときには、キーボードに隠れないようにビューを再配置またはサイズ変更します。キーボードとやり取りを行う方法については、『*Text,Web,andEditingProgramming Guide for iOS*』を参照してください。

ビューに対する変更を開始する場所としては、ビューコントローラが一般的です。ビューコントローラは表示されているコンテンツに関連付けられたビュー階層を管理するため、これらのビューで発生する処理は最終的にすべてビューコントローラが実行します。ビューをロードするとき、または向きの変更を処理するときに、ビューコントローラは新しいビューの追加や、既存のビューの非表示または置換を行ったり、ビューの表示準備のための変更を必要なだけ実行することができます。ビューのコンテンツを編集するためのサポートを実装する場合は、`UIViewController`の`setEditing:animated:`メソッドで、ビューを編集可能なバージョンに切り替えたり、元に戻したりすることができます。

アニメーションブロックも、ビューに関する変更を開始する一般的な場所です。`UIView`クラスに組み込まれたアニメーションのサポートにより、ビューのプロパティに対する変更を簡単にアニメーション化することができます。`transitionWithView:duration:options:animations:completion:`または`transitionFromView:toView:duration:options:completion:`メソッドを使用して、ビューセット全体を別のセットに入れ替えることもできます。

ビューのアニメーション化とビューのトランジションの開始については、“アニメーション”（60ページ）を参照してください。ビューコントローラを使用してビューに関する動作を管理する方法については、『*View Controller Programming Guide for iOS*』を参照してください。

Core Animationレイヤとのやり取り

各ビューオブジェクトは、画面上でビューのコンテンツの表示とアニメーションを管理する、専用のCore Animationレイヤを持ちます。ビューオブジェクトでもさまざまな処理を実行できますが、必要な場合は対応するレイヤオブジェクトを直接操作することもできます。ビューのレイヤオブジェクトは、ビューの`layer`プロパティに保存されます。

ビューに関連付けられたレイヤクラスの変更

ビューを作成した後に、ビューに関連付けられたレイヤのタイプを変更することはできません。したがって、各ビューは`layerClass`クラスメソッドを使用して、レイヤオブジェクトのクラスを指定します。このメソッドのデフォルトの実装は`CALayer`クラスを返します。この値を変更するには、サブクラス化してメソッドをオーバーライドし、別の値を返す以外に方法はありません。この値を変更して、別の種類のレイヤを使うことができます。たとえば、ビューが大きなスクロール可能領域を表示するためにタイル処理を使用している場合、ビューの背景として`CATiledLayer`クラスを使うとよいかも知れません。

`layerClass`メソッドの実装では、単純に目的の`Class`オブジェクトを作成して、これを返すだけにします。たとえば、タイリングを行うビューでは、このメソッドを次のように実装します。

```
+ (Class)layerClass
{
    return [CATiledLayer class];
}
```

各ビューは、初期化プロセスの早期に`layerClass`メソッドを呼び出し、返されたクラスを使用してレイヤオブジェクトを作成します。また、ビューは常に自身をレイヤオブジェクトのデリゲートとして割り当てます。この時点で、ビューはレイヤを所有し、ビューとレイヤの関係を変更できなくなります。同じビューを、別のレイヤオブジェクトのデリゲートとして割り当てることもできません。ビューの所有またはデリゲートの関係を変更すると、描画の問題が発生し、アプリケーションがクラッシュする可能性があります。

Core Animationで提供されるレイヤオブジェクトのタイプについては、『*Core Animation Reference Collection*』を参照してください。

ビューへのレイヤオブジェクトの埋め込み

ビューの代わりに主にレイヤオブジェクトを操作する場合は、必要に応じて、カスタムレイヤオブジェクトをビュー階層に組み込むことができます。カスタムレイヤオブジェクトは、ビューによって所有されない`CALayer`のインスタンスです。一般的には、カスタムレイヤをプログラムで作成し、Core Animationのルーチンを使用してこれらを組み込みます。カスタムレイヤはイベントの受信やレスポンダチェーンへの参加は行いませんが、自身の描画および親ビューまたはレイヤのサイズ変更への応答をCore Animation規則に従って行います。

リスト 3-3に、ビューコントローラのviewDidLoadメソッドの例を示します。このメソッドは、カスタムレイヤオブジェクトを作成し、これをルートビューに追加します。レイヤは、アニメーション化される静止画像を表示するために使用されます。レイヤをビュー自体に追加する代わりに、ビューの下位レベルのレイヤに追加します。

リスト 3-3　ビューへのカスタムレイヤの追加

```
- (void)viewDidLoad {
    [super viewDidLoad];

    // Create the layer.
    CALayer* myLayer = [[CALayer alloc] init];

    // Set the contents of the layer to a fixed image. And set
    // the size of the layer to match the image size.
    UIImage layerContents = [[UIImage imageNamed:@"myImage"] retain];
    CGSize imageSize = layerContents.size;

    myLayer.bounds = CGRectMake(0, 0, imageSize.width, imageSize.height);
    myLayer = layerContents.CGImage;

    // Add the layer to the view.
    CALayer*    viewLayer = self.view.layer;
    [viewLayer addSublayer:myLayer];

    // Center the layer in the view.
    CGRect        viewBounds = backingView.bounds;
    myLayer.position = CGPointMake(CGRectGetMidX(viewBounds),
CGRectGetMidY(viewBounds));

    // Release the layer, since it is retained by the view's layer
    [myLayer release];
}
```

必要な数のサブレイヤを追加して、サブレイヤ階層にまとめることもできます。ただし、いずれはこれらのレイヤをビューのレイヤオブジェクトにアタッチする必要があります。

レイヤを直接操作する方法については、『*Core Animation Programming Guide*』を参照してください。

カスタムビューの定義

標準のシステムビューでは必要な動作を実現できない場合は、カスタムビューを定義できます。カスタムビューを使用すると、アプリケーションのコンテンツの外観およびコンテンツとのやり取りを処理する方法を完全に制御することができます。

> **注意:** 描画にOpenGL ESを使っている場合、`UIView`のサブクラスを定義するのではなく、`GLKView`クラスを使わなければなりません。OpenGL ESで描画する方法について詳しくは、『*OpenGL ES Programming Guide for iOS*』を参照してください。

カスタムビューを実装する際のチェックリスト

カスタムビューの役目は、コンテンツを表示し、そのコンテンツとのやり取りを管理することです。ただし、カスタムビューを正しく実装するためには、単に描画とイベント処理を行う以上の作業が必要です。以下のチェックリストには、カスタムビューを実装するときにオーバーライドできる重要なメソッド（および提供する動作）を示しています。

- ビューの適切な初期化メソッドを定義します。
  - ビューをプログラムで作成する場合は、`initWithFrame:`メソッドをオーバーライドするか、カスタム初期化メソッドを定義します。
  - ビューをnibファイルからロードする場合は、`initWithCoder:`メソッドをオーバーライドします。このメソッドを使用して、ビューを初期化し、既知の状態に設定します。
- `dealloc`メソッドを実装して、カスタムデータのクリーンアップを処理します。
- カスタム描画を処理するには、`drawRect:`メソッドをオーバーライドして、そこで描画を行います。
- ビューの`autoresizingMask`プロパティを設定して、自動サイズ変更の動作を定義します。
- ビュークラスが1つ以上のサブビューを管理する場合は、以下の処理を行います。
  - ビューの初期化シーケンス中にこれらのサブビューを作成します。
  - 各サブビューの`autoresizingMask`プロパティを作成時に設定します。
  - サブビューでカスタムレイアウトが必要な場合は、`layoutSubviews`メソッドをオーバーライドし、そこでレイアウトコードを実装します。
- タッチベースのイベントを処理するには、以下の処理を行います。
  - `addGestureRecognizer:`メソッドを使用して、適切なジェスチャリコグナイザをビューにアタッチします。

- タッチを自分で処理する場合は、touchesBegan:withEvent:、touchesMoved:withEvent:、touchesEnded:withEvent:、およびtouchesCancelled:withEvent:メソッドをオーバーライドします（他にどのタッチ関連のメソッドをオーバーライドするかにかかわらず、touchesCancelled:withEvent:メソッドは常にオーバーライドが必要であることに注意してください）。

- ビューを印刷したときに画面上の表示と異なるようにする場合は、drawRect:forViewPrintFormatter:メソッドを実装します。ビューで印刷をサポートする方法については、『*Drawing and Printing Guide for iOS*』を参照してください。

メソッドのオーバーライドに加え、ビューの既存のプロパティとメソッドを利用してさまざまな処理を実行できることも覚えておいてください。たとえば、contentModeおよびcontentStretchプロパティを使用すると、レンダリングされたビューの最終的な外観を変更でき、コンテンツを自分で再描画する場合によく利用されます。UIViewクラス自体に加え、ビューの下位レベルのCALayerオブジェクトには、直接または間接的に設定できるさまざまな要素があります。レイヤオブジェクトそのもののクラスを変更することさえ可能です。

ビュークラスのメソッドとプロパティについては、『*UIView 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、ビューオブジェクトを既知の状態に設定します。ビューのインスタンスをコード内でプログラムによって作成する場合は、このメソッドを使用します。

リスト3-4に、標準的なinitWithFrame:メソッドの実装の骨格を示します。このメソッドは、最初にメソッドの継承された実装を呼び出します。次に、クラスのインスタンス変数と状態情報を初期化した後、初期化されたオブジェクトを返します。継承された実装の呼び出しは、慣例として最初に実行されます。問題がある場合に独自の初期化コードを終了して、nilを返すことができるようにするためです。

リスト 3-4　ビューのサブクラスの初期化

```
- (id)initWithFrame:(CGRect)aRect {
    self = [super initWithFrame:aRect];
    if (self) {
          // setup the initial properties of the view
          ...
       }
    return self;
```

```
}
```

カスタムビュークラスのインスタンスをnibファイルからロードする場合、iOSではnibをロードするコードが、新しい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に注意してください。代わりに、`NSCoding`プロトコルの一部である`initWithCoder:`メソッドを使用します。

ビューが`NSCoding`プロトコルを採用する場合でも、Interface Builderはビューのカスタムプロパティを認識していないため、これらのプロパティをnibファイルにエンコードしません。このため、独自の`initWithCoder:`メソッドでは、何らかの初期化コードを実行して、ビューを既知の状態に設定する必要があります。また、ビュークラスで`awakeFromNib`メソッドを実装し、このメソッドを使用してその他の初期化を実行することもできます。

描画コードの実装

カスタム描画を行う必要があるビューでは、`drawRect:`メソッドをオーバーライドし、そこで描画を行う必要があります。カスタム描画は、最後の手段として使用することをお勧めします。一般的に、ほかのビューを使用してコンテンツを表示できる場合は、それらのビューを使用してください。

`drawRect:`メソッドの実装では、コンテンツの描画だけを行ってください。このメソッドは、アプリケーションのデータ構造を更新したり、描画と関係ないタスクを実行する場所ではありません。描画環境を設定し、コンテンツを描画して、できるだけ早く終了します。`drawRect:`メソッドが頻繁に呼び出される場合は、可能な限り描画コードを最適化して、メソッドの各呼び出しで描画する量をできるだけ少なくします。

ビューの`drawRect:`メソッドを呼び出す前に、UIKitはビューの基本的な描画環境を設定します。具体的には、グラフィックスコンテキストを作成し、座標系および切り取り領域をビューの座標系および表示境界に一致するように調整します。したがって、`drawRect:`メソッドが呼び出されるときには、UIKitやCore Graphicsなどのネイティブの描画テクノロジーを使用してコンテンツの描画を開始できるようになっています。現在のグラフィックスコンテキストのポインタは、`UIGraphicsGetCurrentContext`関数を使用して取得できます。

> **Important:** 現在のグラフィックスコンテキストは、ビューの`drawRect:`メソッドが呼び出されたときに、その呼び出しの間だけ有効です。このメソッドがその後呼び出されるごとに、UIKitはそれぞれ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、オブジェクトをキャッシュして後で使用することはできません。

リスト 3-5に、`drawRect:`メソッドの簡単な実装を示します。この実装では、ビューの周りに幅10ピクセルの赤い枠を描画します。UIKitの描画操作は基礎となる実装にCore Graphicsを使用するため、ここで示すように描画の呼び出しを混在させても、期待どおりの結果が得られます。

リスト 3-5　描画メソッド

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

    // Set the line width to 10 and inset the rectangle by
    // 5 pixels on all sides to compensate for the wider line.
    CGContextSetLineWidth(context, 10);
    CGRectInset(myFrame, 5, 5);

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

ビューの描画コードが常にビュー全体に不透明なコンテンツを表示す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`に設定することで、システムのパフォーマンス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。これにより、描画にかかる時間が短縮されるだけでなく、ビューをほかのコンテンツと合成するために必要な処理も最小化されます。ただし、このプロパティを`YES`に設定するのは、ビューのコンテンツが完全に不透明であることが既知である場合のみにしてください。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、プロパティを`NO`に設定します。

特にスクロール中の描画パフォーマンスを向上させる方法として、ビューの`clearsContextBeforeDrawing`プロパティを`NO`に設定することもできます。このプロパティが`YES`に設定されていると、UIKitは`drawRect:`メソッドによって更新される領域を、メソッドが呼び出される前に自動的に透明な黒色で塗りつぶします。このプロパティを`NO`に設定すると、この塗りつぶし処理のオーバーヘッドを回避できますが、`drawRect:`メソッドに渡される更新矩形一杯にコンテンツを表示する処理によってアプリケーションに負担がかかります。

イベントへの応答

ビューオブジェクトはレスポンダオブジェクト（`UIResponder`クラスのインスタンス）であり、タッチイベントを受け取ることができます。タッチイベントが発生すると、ウインドウは対応するイベントオブジェクトをタッチイベントが発生したビューにディスパッチします。ビューでイベントを処理しない場合、ビューはこのイベントを無視するか、別のオブジェクトで処理されるようにレスポンダチェーンに渡すことができます。

タッチイベントを直接処理するほかに、ビューはジェスチャリコグナイザを使用して、タップ、スワイプ、ピンチ、およびその他の一般的なタッチに関するジェスチャを検出できます。ジェスチャリコグナイザは、タッチイベントを追跡し、これらが目的のジェスチャとして正しい基準に従っていることを確認するという面倒な処理を担います。アプリケーションでタッチイベントを追跡する代わりに、ジェスチャリコグナイザを作成し、適切なターゲットオブジェクトとアクションメソッドを割り当て、`addGestureRecognizer:`メソッドを使用してビューに組み込むことができます。対応するジェスチャが発生すると、ジェスチャリコグナイザがアクションメソッドを呼び出します。

タッチイベントを直接処理する場合は、ビューに以下のメソッドを実装できます。これらのメソッドについては、『*Event Handling Guide for iOS*』を参照してください。

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

ビューのデフォルトの動作では、一度に1つのタッチにだけ応答します。ユーザがタッチ中に別の指を画面に当てても、システムはタッチイベントを無視し、これを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。

ラベルやイメージなどの一部のビューは、初期状態ではイベント処理をすべて無効にします。ビューがタッチイベントを受信できるかどうかは、ビューの`userInteractionEnabled`プロパティの値を変更して制御できます。このプロパティを一時的に`NO`に設定すると、時間のかかる操作を処理している間に、ユーザがビューのコンテンツを操作するのを防げます。イベントがビューに届かないようにするには、`UIApplication`オブジェクトの`beginIgnoringInteractionEvents`および`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。

> **注意:** `UIView`のアニメーションに関するメソッドは、一般的にアニメーションを再生している間のタッチイベントを無効にします。アニメーションを適切に設定することで、この動作をオーバーライドできます。アニメーションの実行については、“アニメーション”（60ページ）を参照してください。

タッチイベントを処理するときに、UIKitは`UIView`の`hitTest:withEvent:`および`pointInside:withEvent:`メソッドを使用して、タッチイベントがビューの境界内で発生したかどうかを判定します。ほとんどの場合、これらのメソッドをオーバーライドする必要はありませんが、

ビューにカスタム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。

ビューを使用した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、ビューを解放するタイミングになると、システムは`dealloc`メソッドを呼び出します。リスト 3-6に示すように、このメソッドの実装では、ビューが保持しているオブジェクトまたはリソースを解放してから、継承された実装を呼び出します。このメソッドを使用して、ほかの種類のタスクを実行しないでください。

リスト 3-6　`dealloc`メソッドの実装

```
- (void)dealloc {
    // Release a retained UIColor object
    [color release];

    // Call the inherited implementation
    [super dealloc];
}
```

アニメーション

アニメーションは、ユーザインターフェイスをある状態から別の状態へ変更するときに滑らかな視覚効果を与えます。iOSでは、ビューの再配置、サイズ変更、ビュー階層からの削除、および非表示化に、数多くのアニメーションが使用されています。ユーザにフィードバックを返したり、関心を引くような視覚効果を実装する場合も、アニメーションを使用できます。

iOSでは、複雑なアニメーションを作成するために、描画コードを記述する必要はありません。この章で説明するすべてのアニメーション技術では、Core Animationで提供される組み込みのサポートを使用します。デベロッパに必要な処理は、アニメーションをトリガして、Core Animationに各フレームのレンダリングを処理させることだけです。これにより、数行のコードだけで複雑なアニメーションを簡単に作成できます。

アニメーション化の対象

UIKitとCore Animationはどちらもアニメーションのサポートを提供していますが、それぞれのテクノロジーで提供されるサポートのレベルは異なります。UIKitでは、アニメーションは`UIView`オブジェクトを使用して実行されます。ビューは、一般的な多くのタスクをカバーする基本的なアニメーションのセットをサポートします。たとえば、ビューのプロパティの変更をアニメーション化したり、トランジションアニメーションを使用してビューセットを入れ替えることができます。

表4-1に、`UIView`クラスの**アニメーション化可能**なプロパティ（アニメーションのサポートが組み込まれているプロパティ）を示します。アニメーション化可能とは、アニメーションが自動的に発生するという意味ではありません。これらのプロパティの値を変更すると、通常はプロパティ（およびビュー）がただちに更新されるだけで、アニメーションは実行されません。これらの変更をアニメーション化するには、プロパティの値をアニメーションブロック内で変更する必要があります。これについては、“ビューのプロパティに対する変更のアニメーション化”（62 ページ）で説明します。

表 4-1　アニメーション化可能な`UIView`のプロパティ

プロパティ	可能な変更
`frame`	スーパービューの座標系を基準にビューのサイズと位置を変更する場合は、このプロパティを変更します（`transform`プロパティに恒等変換が含まれていない場合は、代わりに`bounds`または`center`プロパティを変更します）。

プロパティ	可能な変更
`bounds`	ビューのサイズを変更する場合は、このプロパティを変更します。
`center`	スーパービューの座標系を基準にビューの位置を変更する場合は、このプロパティを変更します
`transform`	中心点を基準にビューを拡大縮小、回転、または平行移動する場合は、このプロパティを変更します。このプロパティを使用する変換は、常に2D空間で実行されます（3D変換を実行するには、Core Animationを使用してビューのレイヤオブジェクトをアニメーション化する必要があります）。
`alpha`	ビューの透明度を徐々に変更する場合は、このプロパティを変更します。
`backgroundColor`	ビューの背景色を変更する場合は、このプロパティを変更します。
`contentStretch`	利用可能な空間一杯にビューのコンテンツを引き延ばす方法を変更する場合は、このプロパティを変更します。

アニメーション化されたビューのトランジションは、ビューコントローラで提供されている以外の方法でビュー階層を変更する方法です。ビューコントローラを使用して簡潔なビュー階層を管理するべきですが、ビュー階層のすべてまたは一部を置き換える必要が生じる場合もあります。このような場合は、ビューベースのトランジションを使用して、ビューの追加や削除をアニメーション化できます。

より複雑なアニメーションが必要な場合、または`UIView`クラスでサポートされていないアニメーションが必要な場合は、Core Animationとビューの下位レベルのレイヤを使用してアニメーションを作成できます。ビューおよびレイヤオブジェクトは複雑にリンクされているため、ビューのレイヤに行った変更はビュー自身にも影響します。Core Animationを使用すると、ビューのレイヤに対する以下の変更をアニメーション化できます。

- レイヤのサイズと位置
- 変換を実行する際の中心点
- レイヤまたはサブレイヤに対する3D空間での変換
- レイヤ階層に対するレイヤの追加または削除
- ほかの兄弟レイヤに対するレイヤのZ方向の順序
- レイヤの影
- レイヤの枠（レイヤの角を丸めるかどうかを含む）
- サイズ変更操作中に引き延ばされるレイヤの部分

- レイヤの不透明度
- レイヤの境界より外側にあるサブレイヤを切り取る動作
- レイヤの現在のコンテンツ
- レイヤのラスタライズの動作

> **注意:** ビューがカスタムレイヤオブジェクト（関連付けられたビューを持たないレイヤオブジェクト）をホストする場合は、Core Animationを使用してレイヤオブジェクトに対する変更をアニメーション化する必要があります。

この章ではCore Animationの動作について少し触れていますが、ビューコードからの開始に関する説明しかありません。Core Animationを使用してレイヤをアニメーション化する方法については、『*Core Animation Programming Guide*』および『*Core Animation Cookbook*』を参照してください。

ビューのプロパティに対する変更のアニメーション化

`UIView`クラスのプロパティに対する変更をアニメーション化するには、これらの変更をアニメーションブロック内で行う必要があります。**アニメーションブロック**という用語は、一般的な意味でアニメーション化可能な変更を指定するコードを表しています。iOS 4以降では、ブロックオブジェクトを使用してアニメーションブロックを作成します。iOSの初期のバージョンでは、`UIView`クラスの特殊なクラスメソッドを使用して、アニメーションブロックの開始位置と終了位置を指定します。どちらの手法も同じ設定オプションをサポートし、アニメーションの実行について制御できる内容は同じです。ただし、可能な場合は常にブロックベースのメソッドを使用してください。

以降のセクションでは、ビューのプロパティに対する変更をアニメーション化するために必要なコードについて説明します。ビューセット間にアニメーション化されたトランジションを作成する方法については、“ビュー間のアニメーション化されたトランジションの作成”（71 ページ）を参照してください。

ブロックベースのメソッドを使用したアニメーションの開始

iOS 4以降では、ブロックベースのクラスメソッドを使用して、アニメーションを開始します。いくつかのブロックベースのメソッドが用意されており、それぞれ設定レベルの異なるアニメーションブロックを提供します。これらのメソッドは以下のとおりです。

- `animateWithDuration:animations:`
- `animateWithDuration:animations:completion:`
- `animateWithDuration:delay:options:animations:completion:`

これらはクラスメソッドであるため、これらのメソッドを使用して作成するアニメーションブロックは1つのビューだけに関連付けられません。したがって、これらのメソッドを使用して、複数のビューに対して変更を行う1つのアニメーションを作成できます。たとえば、リスト4-1は、ビューのフェードインと別のビューのフェードアウトを1秒間で実行するために必要なコードを示しています。このコードを実行すると、現在のスレッドまたはアプリケーションのメインスレッドをブロックしないように、指定したアニメーションが別のスレッドでただちに開始されます。

リスト 4-1　簡単なブロックベースのアニメーションの実行

```
[UIView animateWithDuration:1.0 animations:^{
        firstView.alpha = 0.0;
        secondView.alpha = 1.0;
}];
```

上の例のアニメーションは、イーズインイーズアウトのアニメーション曲線を使用して、1度だけ実行されます。デフォルトのアニメーションパラメータを変更する場合は、`animateWithDuration:delay:options:animations:completion:`メソッドを使用してアニメーションを実行する必要があります。このメソッドでは、以下のアニメーションパラメータをカスタマイズできます。

- アニメーションを開始するまでの遅延
- アニメーション中に使用するタイミング曲線の種類
- アニメーションを繰り返す回数
- アニメーションを最後まで再生した後、自動的に逆再生するかどうか
- アニメーションの再生中にタッチイベントをビューに送信するかどうか
- アニメーションを開始するときに、再生中のアニメーションを中断して開始するか、または再生中のアニメーションが完了するまで待機するか

また、`animateWithDuration:animations:completion:`と`animateWithDuration:delay:options:animations:completion:`メソッドはどちらも、完了ハンドラブロックを指定する機能をサポートします。完了ハンドラを使用すると、アプリケーションに特定のアニメーションが完了したことを通知できます。完了ハンドラは、別々のアニメーションをリンクする方法としても利用できます。

リスト4-2に、完了ハンドラを使用して、1つ目のアニメーションが完了した後に新しいアニメーションを開始するアニメーションブロックの例を示します。最初の`animateWithDuration:delay:options:animations:completion:`の呼び出しで、フェードアウト

のアニメーションを設定し、いくつかのカスタムオプションを設定します。このアニメーションが完了すると、完了ハンドラは2番目のアニメーションを実行および設定します。この例では、1秒後にビューをもう一度フェードインしています。

完了ハンドラの使用は、複数のアニメーションをリンクする際に用いられる主な方法です。

リスト 4-2　カスタムオプションを使用したアニメーションブロックの作成

```
- (IBAction)showHideView:(id)sender
{
    // Fade out the view right away
    [UIView animateWithDuration:1.0
        delay: 0.0
        options: UIViewAnimationOptionCurveEaseIn
        animations:^{
             thirdView.alpha = 0.0;
        }
        completion:^(BOOL finished){
            // Wait one second and then fade in the view
            [UIView animateWithDuration:1.0
                 delay: 1.0
                 options:UIViewAnimationOptionCurveEaseOut
                 animations:^{
                    thirdView.alpha = 1.0;
                 }
                 completion:nil];
        }];
}
```

Important: すでに実行中のアニメーションに関連するプロパティの値を変更しても、現在のアニメーションは停止しません。現在のアニメーションは続行され、プロパティに割り当てた新しい値までアニメーション化されます。

begin/commitメソッドを使用したアニメーションの開始

アプリケーションをiOS 3.2以前で実行する場合は、`UIView`のクラスメソッドである`beginAnimations:context:`および`commitAnimations`を使用して、アニメーションブロックを定義する必要があります。これらのメソッドは、アニメーションブロックの開始位置と終了位置を指定します。これらのメソッド間で変更したアニメーション化可能なプロパティは、`commitAnimations`メソッドを呼び出した後に、新しい値までアニメーション化されます。アニメーションの実行は、現在のスレッドまたはアプリケーションのメインスレッドをブロックしないように、別のスレッドで実行されます。

注意: iOS 4以降のアプリケーションを作成している場合は、コンテンツのアニメーション化にブロックベースのメソッドを使用する必要があります。これらのメソッドの使用方法については、“ブロックベースのメソッドを使用したアニメーションの開始” （62 ページ）を参照してください。

リスト 4-3に、リスト 4-1 （63 ページ）と同じ動作をbegin/commitメソッドを使用して実装するために必要なコードを示します。リスト 4-1でのように、このコードはビューのフェードアウトと別のビューのフェードインを1秒間で実行します。ただしこの例では、別のメソッド呼び出しを使用してアニメーションの再生時間を設定する必要があります。

リスト 4-3 簡単なbegin/commitアニメーションの実行

```
[UIView beginAnimations:@"ToggleViews" context:nil];
[UIView setAnimationDuration:1.0];

// Make the animatable changes.
firstView.alpha = 0.0;
secondView.alpha = 1.0;

// Commit the changes and perform the animation.
[UIView commitAnimations];
```

デフォルトでは、アニメーションブロック内で行われたアニメーション化可能なプロパティの変更は、すべてアニメーション化されます。一部の変更だけをアニメーション化する場合は、`setAnimationsEnabled:`メソッドを使用してアニメーションを一時的に無効にし、アニメーション化しない変更を行った後、もう一度`setAnimationsEnabled:`を呼び出してアニメーションを再有効化します。アニメーションが現在有効かどうかは、`areAnimationsEnabled`クラスメソッドを呼び出すことで確認できます。

> **注意:** 実行中のアニメーションに関連するプロパティの値を変更しても、現在のアニメーションは停止しません。アニメーションは続行され、プロパティに割り当てた新しい値までアニメーション化されます。

begin/commitアニメーションのパラメータの設定

begin/commitアニメーションブロックのアニメーションパラメータを設定するには、`UIView`のクラスメソッドを使用します。表4-2に、これらのメソッドのリストと、各メソッドを使用してアニメーションを設定する方法を示します。これらのメソッドの多くはbegin/commitアニメーションブロック内のみから呼び出しますが、一部のメソッドはブロックベースのアニメーションでも使用できます。アニメーションブロックでこれらのメソッドを呼び出さなかった場合は、対応する属性のデフォルト値が使用されます。各メソッドに関連付けられたデフォルト値については、『*UIView Class Reference*』を参照してください。

表 4-2 アニメーションブロックを設定するメソッド

メソッド	用途
`setAnimationStartDate:` `setAnimationDelay:`	アニメーションの実行をいつ開始するかを指定する場合は、これらのメソッドのいずれかを使用します。指定した開始日が過去の日付（または、遅延が0）である場合、アニメーションは可能な限りすぐに開始されます。
`setAnimationDuration:`	アニメーションの実行を継続する時間を設定します。
`setAnimationCurve:`	アニメーションのタイミング曲線を設定します。アニメーションを線形に実行するか、一定時間で速度を変更するかを制御します。
`setAnimationRepeat-Count:` `setAnimationRepeat-Autoreverses:`	アニメーションを繰り返す回数と、各サイクルの終わりにアニメーションを逆向きに実行するかどうかを設定します。これらのメソッドの使用方法については、“逆再生を行うアニメーションの実装” （70 ページ）を参照してください。

メソッド	用途

`setAnimationDelegate:`
`setAnimationWill-StartSelector:`
`setAnimationDid-StopSelector:`
	アニメーションの直前または直後にコードを実行する場合は、これらのメソッドを使用します。デリゲートの使用方法については、“アニメーションデリゲートの設定”（68 ページ）を参照してください。

`setAnimationBegins-FromCurrentState:`	すべてのアニメーションをただちに停止して、新しいアニメーションを停止した位置から開始します。このメソッドに、`YES`の代わりに`NO`を渡すと、新しいアニメーションは前のアニメーションが停止してから開始されます。

リスト 4-4に、リスト 4-2 （64 ページ）のコードと同じ動作をbegin/commitメソッドを使用して実装するために必要なコードを示します。前の例と同様に、このコードはビューをフェードアウトし、1秒待機してから再度フェードインします。アニメーションの後半部分を実装するために、コードはアニメーションデリゲートを設定し、停止時に呼び出されるメソッドを実装します。続いて、このハンドラメソッドがアニメーションの後半を設定して実行します。

リスト 4-4　begin/commitメソッドを使用したアニメーションパラメータの設定

```
// This method begins the first animation.
- (IBAction)showHideView:(id)sender
{
    [UIView beginAnimations:@"ShowHideView" context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseIn];
    [UIView setAnimationDuration:1.0];
    [UIView setAnimationDelegate:self];
    [UIView
setAnimationDidStopSelector:@selector(showHideDidStop:finished:context:)];

    // Make the animatable changes.
    thirdView.alpha = 0.0;

    // Commit the changes and perform the animation.
    [UIView commitAnimations];
}

// Called at the end of the preceding animation.
```

```
- (void)showHideDidStop:(NSString *)animationID finished:(NSNumber *)finished
context:(void *)context
{
    [UIView beginAnimations:@"ShowHideView2" context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseOut];
    [UIView setAnimationDuration:1.0];
    [UIView setAnimationDelay:1.0];

    thirdView.alpha = 1.0;

    [UIView commitAnimations];
}
```

アニメーションデリゲートの設定

アニメーションの直前または直後にコードを実行する場合は、デリゲートオブジェクトと開始または停止セレクタを、begin/commitアニメーションブロックに関連付ける必要があります。デリゲートオブジェクトは`UIView`のクラスメソッドである`setAnimationDelegate:`を使用して設定し、開始セレクタと停止セレクタは`setAnimationWillStartSelector:`および`setAnimationDidStopSelector:`クラスメソッドを使用して設定します。アニメーションの実行中に、アニメーションシステムがデリゲートメソッドを適切なタイミングで呼び出し、必要なコードを実行することができます。

アニメーションデリゲートメソッドには、以下のような署名が必要です。

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

各メソッドの`animationID`および`context`パラメータは、アニメーションブロックの先頭で`beginAnimations:context:`メソッドに渡す同じパラメータです。

- `animationID` - アニメーションを識別するための、アプリケーションが提供する文字列です。
- `context` - デリゲートに追加情報を渡すために使用できる、アプリケーションが提供するオブジェクトです。

setAnimationDidStopSelector:セレクタメソッドには、Booleanのパラメータがもう一つあります。アニメーションが最後まで実行されると、このパラメータの値がYESになります。このパラメータの値がNOの場合は、アニメーションがキャンセルされたか、ほかのアニメーションによって途中で停止されています。

> **注意:** アニメーションデリゲートはブロックベースのメソッドでも使用できますが、一般的に、この場合は使用する必要がありません。代わりに、アニメーションの前に実行するコードをブロックの先頭に置き、アニメーションが完了した後に実行するコードを完了ハンドラに配置します。

アニメーションブロックのネスト

別のアニメーションブロックをネストすることで、アニメーションブロックの各部分に異なるタイミングや設定オプションを割り当てることができます。ネストされたアニメーションブロックは、名前からわかるように、既存のアニメーションブロック内に作成された新しいアニメーションブロックです。ネストされたアニメーションは親アニメーションと同時に開始されますが、大部分は独自の設定オプションに従って実行されます。デフォルトでは、ネストされたアニメーションは親の再生時間とアニメーション曲線を継承しますが、これらのオプションも必要に応じてオーバーライドできます。

リスト 4-5に、ネストされたアニメーションを使用して一部のアニメーションのタイミング、再生時間、および動作をグループ全体で変更する方法の例を示します。この例では、2つのビューが完全に透明になるまで変化していますが、anotherViewオブジェクトの透明度は最終的に非表示なるまでに何度か元に戻っています。ネストされたアニメーションブロックで使用されているUIViewAnimationOptionOverrideInheritedCurveおよびUIViewAnimationOptionOverrideInheritedDurationキーで、1つ目のアニメーションの曲線と再生時間の値を、2つ目のアニメーションで変更することができます。これらのキーがない場合は、代わりに親のアニメーションブロックの再生時間と曲線が使用されます。

リスト 4-5　設定の異なるアニメーションのネスト

```
[UIView animateWithDuration:1.0
        delay: 1.0
        options:UIViewAnimationOptionCurveEaseOut
        animations:^{
            aView.alpha = 0.0;

            // Create a nested animation that has a different
            // duration, timing curve, and configuration.
            [UIView animateWithDuration:0.2
```

```
                delay:0.0
                options: UIViewAnimationOptionOverrideInheritedCurve |
                         UIViewAnimationOptionCurveLinear |
                         UIViewAnimationOptionOverrideInheritedDuration |
                         UIViewAnimationOptionRepeat |
                         UIViewAnimationOptionAutoreverse
                animations:^{
                     [UIView setAnimationRepeatCount:2.5];
                     anotherView.alpha = 0.0;
                }
                completion:nil];

        }
        completion:nil];
```

begin/commitメソッドを使用してアニメーションを作成している場合、ネストはブロックベースのメソッドの場合とほぼ同様に機能します。すでに開いているアニメーションブロック内で`beginAnimations:context:`を呼び出すごとに、新しいネストされたアニメーションブロックが作成され、必要に応じて設定できます。行った変更はすべて、最後に開いたアニメーションブロックに適用されます。すべてのアニメーションブロックは、アニメーションを送信して実行する前に、`commitAnimations`を呼び出して閉じる必要があります。

逆再生を行うアニメーションの実装

逆再生可能なアニメーションを作成するときに繰り返し回数を使用する場合は、繰り返し回数に整数以外の値を指定することも考慮します。自動で逆再生するアニメーションの場合は、アニメーションの1サイクルで、元の値から新しい値までアニメーションが実行された後、もう一度元の値まで戻ります。アニメーションを新しい値で終了する場合は、繰り返し回数に`0.5`を加算して、アニメーションを新しい値で終了するために必要な半分のサイクルを追加します。この半分のサイクルを追加しないと、アニメーションは元の値まで再生された後すぐに新しい値に変更され、必要な視覚効果が得られません。

ビュー間のアニメーション化されたトランジションの作成

ビューのトランジションを利用すると、ビュー階層でのビューの追加、削除、または非表示と表示の切り替えに関して、変更が突然表示されるのを防ぐことができます。以下のような変更を実装するには、ビューのトランジションを使用します。

- **既存のビューの表示しているサブビューを変更する。**既存のビューに比較的小規模な変更を行う場合は、一般的にこの方法を選択します。
- **ビュー階層で、あるビューを別のビューに置き換える。**画面の全体またはほとんどを占めるビュー階層を置き換える場合は、一般的にこの方法を選択します。

Important: ビューコントローラが開始するトランジション（モーダルビューコントローラの表示や、ナビゲーションスタックへの新しいビューコントローラのプッシュなど）と、ビューのトランジションを混同しないでください。ビューのトランジションはビュー階層にだけ影響しますが、ビューコントローラのトランジションはアクティブなビューコントローラも変更します。したがって、ビューのトランジションでは、トランジションの開始時にアクティブなビューコントローラはトランジションが完了したときもアクティブなままです。

ビューコントローラを使用して新しいコンテンツを表示する方法については、『*View Controller Programming Guide for iOS*』を参照してください。

ビューのサブビューの変更

ビューのサブビューを変更すると、ビューに中規模の変更を行うことができます。たとえば、サブビューを追加または削除して、スーパービューの2つの状態を切り替えることができます。アニメーションの終了時には、同じビューが表示されていますがコンテンツは異なります。

iOS 4以降では、`transitionWithView:duration:options:animations:completion:`メソッドを使用して、ビューのトランジションアニメーションを開始します。このメソッドに渡されるアニメーションブロックでは、通常アニメーション化される変更はサブビューの表示と非表示の切り替え、追加、または削除に関連する変更だけです。これらの変更だけにアニメーションを限定することで、ビューは変更の前後のスナップショットイメージを作成し、2つのイメージ間で効率よくアニメーション化を実行できます。ただし、その他の変更をアニメーション化する必要がある場合は、メソッドを呼び出すときに`UIViewAnimationOptionAllowAnimatedContent`オプションを指定できます。このオプションを指定すると、ビューによるスナップショットの作成が行われず、すべての変更が直接アニメーション化されます。

リスト 4-6に、トランジションアニメーションを使用して、新しいテキストの入力ページが追加されたように見せる例を示します。この例では、メインビューに埋め込みのテキストビューが2つ含まれています。2つのテキストビューの設定はほぼ同じですが、一方は常に表示され、もう一方は常に非表示になるように設定されています。ユーザがボタンをタップして新しいページを作成すると、この

メソッドは2つのビューの表示設定を切り替えます。結果的に、テキストを入力可能な空のテキストビューを使用して、新しい空のページが表示されます。トランジションが完了すると、ビューはプライベートなメソッドを使用して古いページのテキストを保存し、非表示にしたテキストビューを後で再利用できるようにリセットします。続いて、ビューはポインタを整理し、ユーザが別の新しいページを要求したときに同じ処理を行えるように準備します。

リスト 4-6　既存のテキストビューと空のテキストビューの切り替え

```
- (IBAction)displayNewPage:(id)sender
{
    [UIView transitionWithView:self.view
        duration:1.0
        options:UIViewAnimationOptionTransitionCurlUp
        animations:^{
            currentTextView.hidden = YES;
            swapTextView.hidden = NO;
        }
        completion:^(BOOL finished){
            // Save the old text and then swap the views.
            [self saveNotes:temp];

            UIView*    temp = currentTextView;
            currentTextView = swapTextView;
            swapTextView = temp;
        }];
}
```

iOS 3.2以前でビューのトランジションを実行する必要がある場合は、`setAnimationTransition:forView:cache:`メソッドを使用してトランジションのパラメータを指定できます。このメソッドに渡すビューは、`transitionWithView:duration:options:animations:completion:`メソッドに1つ目のパラメータとして渡すビューと同じです。リスト 4-7に、アニメーションブロックで作成する必要がある基本的な構造を示します。リスト 4-6 （72 ページ）に示す完了ブロックを実装するには、“アニメーションデリゲートの設定” （68 ページ）で説明するように、停止時に呼び出されるハンドラを使用してアニメーションデリゲートを設定する必要があります。

リスト 4-7　begin/commitメソッドを使用したサブビューの変更

```
    [UIView beginAnimations:@"ToggleSiblings" context:nil];
   [UIView setAnimationTransition:UIViewAnimationTransitionCurlUp forView:self.view
cache:YES];
    [UIView setAnimationDuration:1.0];


    // Make your changes


    [UIView commitAnimations];
```

別のビューへの置き換え

ビューの置き換えは、インターフェイスを劇的に変化させたいときに使用できます。この方法ではビューだけが入れ替わるため（ビューコントローラは入れ替わりません）、デベロッパはアプリケーションのコントローラオブジェクトを適切に設計する必要があります。この方法は、単に標準のトランジションをいくつか使用して新しいビューをすばやく表示するだけです。

iOS 4以降では、2つのビュー間でトランジションを行うには、`transitionFromView:toView:duration:options:completion:`メソッドを使用します。このメソッドは、実際に1つ目のビューをビュー階層から削除して、別のビューを挿入します。したがって、1つ目のビューを維持しておきたい場合は、ビューの参照を保持する必要があります。ビュー階層から削除する代わりにビューを非表示にする場合は、`UIViewAnimationOptionShowHideTransitionViews`キーをオプションの1つとして渡します。

リスト 4-8に、1つのビューコントローラで管理される2つのメインビューを入れ替えるために必要なコードを示します。この例では、ビューコントローラのルートビューは常に2つの子ビュー（`primaryView`または`secondaryView`）のいずれかを表示します。各ビューは同じコンテンツを表示しますが、表示する方法がそれぞれ異なります。ビューコントローラは、`displayingPrimary`メンバー変数（ブール値）を使用し、任意の時点でどちらのビューが表示されているかを常に追跡しています。入れ替える方向は、どちらのビューが表示されているかに応じて変化します。

リスト 4-8　ビューコントローラの2つのビューの切り替え

```
- (IBAction)toggleMainViews:(id)sender {
    [UIView transitionFromView:(displayingPrimary ? primaryView : secondaryView)
        toView:(displayingPrimary ? secondaryView : primaryView)
        duration:1.0
        options:(displayingPrimary ? UIViewAnimationOptionTransitionFlipFromRight
 :
```

```
                    UIViewAnimationOptionTransitionFlipFromLeft)
        completion:^(BOOL finished) {
            if (finished) {
                displayingPrimary = !displayingPrimary;
            }
    }];
}
```

注意: ビューの入れ替えに加え、ビューコントローラのコードでは、1つ目のビューと2つ目のビューの両方のロードおよびアンロードを管理する必要があります。ビューコントローラでビューをロードおよびアンロードする方法については、『*View Controller Programming Guide for iOS*』を参照してください。

複数のアニメーションのリンク

`UIView`アニメーションインターフェイスでは、独立したアニメーションブロックのリンクがサポートされ、これらのアニメーションを同時でなく順番に実行できます。アニメーションブロックをリンクするプロセスは、ブロックベースのアニメーションメソッドを使用しているか、begin/commitメソッドを使用しているかで異なります。

- ブロックベースのアニメーションでは、`animateWithDuration:animations:completion:`および`animateWithDuration:delay:options:animations:completion:`メソッドでサポートされる完了ハンドラを使用して、後に続くアニメーションを実行します。
- begin/commitアニメーションでは、デリゲートオブジェクトと停止時のセレクタをアニメーションに関連付けます。デリゲートをアニメーションに関連付ける方法については、“アニメーションデリゲートの設定”（68 ページ）を参照してください。

複数のアニメーションをリンクする代わりに、ネストしたアニメーションに別々の遅延を設定し、各アニメーションを異なるタイミングで開始することもできます。アニメーションをネストする方法については、“アニメーションブロックのネスト”（69 ページ）を参照してください。

ビューとレイヤの変更の同時アニメーション化

アプリケーションには、必要に応じてビューベースとレイヤベースのアニメーションコードを自由に混在させることができます。ただし、アニメーションパラメータを設定するプロセスはレイヤの所有者によって異なります。ビューが所有するレイヤの変更は、ビュー自体を変更するのと同じです。レイヤのプロパティに適用するアニメーションでは、現在のビューベースのアニメーションブロックに設定されたアニメーションパラメータが使用されます。デベロッパが作成したレイヤでは動作が異なります。カスタムレイヤオブジェクトはビューベースのアニメーションブロックのパラメータを無視し、代わりにデフォルトのCore Animationパラメータを使用します。

作成するレイヤのアニメーションパラメータをカスタマイズする場合は、直接Core Animationを使用する必要があります。一般的に、Core Animationを使用してレイヤをアニメーション化する場合は、`CABasicAnimation`オブジェクトまたは`CAAnimation`の具象サブクラスをいくつか作成します。その後、対応するレイヤにそのアニメーションを追加します。アニメーションはビューベースのアニメーションブロックの内部または外部のどちらからでも適用できます。

リスト4-9に、ビューとカスタムレイヤを同時に変更するアニメーションを示します。この例のビューは、境界の中心に配置されたカスタム`CALayer`オブジェクトを含んでいます。アニメーションはビューを反時計回りに回転し、レイヤを時計回りに回転します。ビューとレイヤが逆方向に回転しているため、レイヤは画面に対して元の向きを保ち、ほとんど回転しているようには見えません。ただし、レイヤの下にあるビューは360度回転して、元の向きに戻ります。この例の主な目的は、ビューとレイヤのアニメーションをどのように混在できるかを示すことです。正確なタイミングが必要な状況では、このタイプの混在を使用しないでください。

リスト 4-9　ビューとレイヤのアニメーションの混在

```
[UIView animateWithDuration:1.0
    delay:0.0
    options: UIViewAnimationOptionCurveLinear
    animations:^{
        // Animate the first half of the view rotation.
        CGAffineTransform  xform =
CGAffineTransformMakeRotation(DEGREES_TO_RADIANS(-180));
        backingView.transform = xform;

        // Rotate the embedded CALayer in the opposite direction.
        CABasicAnimation*    layerAnimation = [CABasicAnimation
animationWithKeyPath:@"transform"];
        layerAnimation.duration = 2.0;
        layerAnimation.beginTime = 0; //CACurrentMediaTime() + 1;
```

```
        layerAnimation.valueFunction = [CAValueFunction
functionWithName:kCAValueFunctionRotateZ];
        layerAnimation.timingFunction = [CAMediaTimingFunction
                        functionWithName:kCAMediaTimingFunctionLinear];
        layerAnimation.fromValue = [NSNumber numberWithFloat:0.0];
        layerAnimation.toValue = [NSNumber
numberWithFloat:DEGREES_TO_RADIANS(360.0)];
        layerAnimation.byValue = [NSNumber
numberWithFloat:DEGREES_TO_RADIANS(180.0)];
        [manLayer addAnimation:layerAnimation forKey:@"layerAnimation"];
    }
    completion:^(BOOL finished){
        // Now do the second half of the view rotation.
        [UIView animateWithDuration:1.0
             delay: 0.0
             options: UIViewAnimationOptionCurveLinear
             animations:^{
                 CGAffineTransform  xform =
CGAffineTransformMakeRotation(DEGREES_TO_RADIANS(-359));
                 backingView.transform = xform;
             }
             completion:^(BOOL finished){
                 backingView.transform = CGAffineTransformIdentity;
         }];
}];
```

注意: リスト 4-9 （75 ページ）で、CABasicAnimationオブジェクトをビューベースのアニメーションブロックの外部で作成および適用しても、同じ結果を得ることができます。すべてのアニメーションの実行は最終的にCore Animationに依存するため、ほぼ同時に送信されていれば、アニメーションは一緒に実行されます。

ビューベースのアニメーションとレイヤベースのアニメーションとの間に正確なタイミングが必要な場合は、すべてのアニメーションをCore Animationを使用して作成することをお勧めします。いくつかのアニメーションはCore Animationを使用した方が簡単に実行できます。たとえば、リスト

4-9 （75 ページ）におけるビューベースの回転では、180度を超える回転を行うために複数のステップを必要としていますが、Core Animation部分では回転の評価関数を使用し、開始値から終了値までの回転を中間値を経由して実行しています。

Core Animationを使用してアニメーションを作成および設定する方法については、『*Core Animation Programming Guide*』および『*Core Animation Cookbook*』を参照してください。

書類の改訂履歴

この表は「*iOS View プログラミングガイド*」の改訂履歴です。

日付	メモ
2014-09-17	TBD
2013-10-22	OpenGL ESを使って描画するビューに関するアドバイスを改訂しました。
2011-03-08	文書の内容を再編し、拡充しました。 ビューベースのアニメーションを作成する方法についての情報を追加しました。 コンテンツを外部ディスプレイに表示する方法についての情報を追加しました。 高解像度画面を操作する方法についての情報を追加しました。
2010-05-17	ビュー、ウインドウ、およびその他のビジュアルインターフェイス要素の作成と管理について説明した新規文書。

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

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

Offline copy. Trademarks go here.