# Core Animationプログラミング言語

**Graphics & Imaging > Quartz**

**2010-09-24**

Apple Inc.
© 2010 Apple Inc.
All rights reserved.

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

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

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

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

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

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

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/

Apple, the Apple logo, Cocoa, Cocoa Touch, iPhone, iPod, iPod touch, Mac, Mac OS, Objective-C, Quartz, and QuickTime are trademarks of Apple Inc., registered in the United States and other countries.

IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

OpenGL is a registered trademark of Silicon Graphics, Inc.

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

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

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

**一部の国や地域では、黙示あるいは偶発的または結果的損害に対する賠償の免責または制限が認められていないため、上記の制限や免責がお客様に適用されない場合があります。 この保証はお客様に特定の法的権利を与え、地域によってはその他の権利がお客様に与えられる場合もあります。**

# 目次





2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# 図、表、リスト





2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# Core Animationプログラミングガイドの紹介

この文書では、Core Animationの使用に関する基本概念を説明します。Core Animationは、高性能の合成エンジンと使いやすいアニメーションプログラミングインターフェイスを組み合わせた、Objective-Cのフレームワークです。

この文書を読めば、CocoaアプリケーションでのCore Animationの使いかたを理解できます。Core AnimationではObjective-Cのプロパティを広く利用しているため、この文書は『*The Objective-C Programming Language*』を読んでいることを前提としています。『*Key-Value Coding Programming Guide*』に説明されているキー値コーディングの知識も必要です。必須ではありませんが、『*Quartz 2D Programming Guide*』に説明されているQuartz 2Dイメージングテクノロジーの知識も役立ちます。

Cocoaアプリケーションは、Mac OS Xオペレーティングシステムと、iPhoneやiPod touchなどのマルチタッチデバイス用オペレーティングシステムであるiOSの2つのプラットフォーム用に作成できます。『Core Animationプログラミングガイド』では、できる限りの情報を盛り込み、必要に応じてプラットフォーム間の各相違点を指摘しながら、両方のプラットフォームに関係するCocoa関連の情報を紹介します。

## この書類の構成

『*Core Animation プログラミングガイド*』は、次の項目で構成されています。

- 「Core Animationとは？」（13 ページ）では、Core Animationの機能の概要を述べます。
- 「レイヤジオメトリと変換」（21 ページ）では、レイヤジオメトリと変換について説明します。
- 「レイヤツリー階層」（29 ページ）では、レイヤツリーについてと、アプリケーションがこれをどのように操作できるかについて説明します。
- 「レイヤコンテンツの指定」（33 ページ）では、基本的なレイヤコンテンツを指定する方法を説明します。
- 「アニメーション」（39 ページ）では、Core Animationアニメーションモデルについて説明します。
- 「レイヤアクション」（43 ページ）では、レイヤのアクションと暗黙的なアニメーションの実装方法を説明します。
- 「トランザクション」（47 ページ）では、トランザクションを使ってアニメーションをグループ化する方法を説明します。
- 「Core Animationレイヤの配置」（51 ページ）では、制約レイアウトマネージャについて説明します。
- 「キー値コーディングへのCore Animationの拡張」（55 ページ）では、Core Animationが提供するキー値コーディングへの拡張について説明します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

- 「レイヤスタイルプロパティ」（59 ページ）では、レイヤスタイルプロパティについてと、その視覚的効果の例を示します。
- 「サンプル：Core Animation Kiosk Menu Styleアプリケーション」（69 ページ）では、Core Animationを使ったユーザインターフェイスを分析します。
- 「アニメーション化可能プロパティ」（81 ページ）では、アニメーション化が可能なレイヤとファイルのプロパティについて要約します。

# 関連項目

以下のプログラミングガイドでは、Core Animationによって使われるいくつかのテクノロジーを解説しています。

- 『*Animation Types and Timing Programming Guide*』では、Core Animationで使われるアニメーションクラスとタイミング機能を説明しています。
- 『*Core Animation Cookbook*』では、一般的なCore Animationタスクを示すコードを取り上げています。
- 『*Quartz 2D Programming Guide*』では、`CALayer`インスタンスのコンテンツの描画に使われる2次元の描画エンジンについて説明しています。
- 『*Core Image Programming Guide*』では、Mac OS Xのイメージ処理テクノロジーについてと、Core Image APIの使いかたを説明しています。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# Core Animationとは？

Core Animationは、グラフィック描画、投影、アニメーション化のためのObjective-Cクラスのコレクションです。Core Animationは、Application KitとCocoa Touchのビューアーキテクチャを使用するデベロッパにとっては親しみのある階層レイヤの抽象化を維持しながら、高度な合成効果を用いた流体アニメーションを可能にします。

ダイナミックな、アニメーション化されたユーザインターフェイスは作成が困難ですが、Core Animationの次の機能により、このようなインターフェイスの作成が容易になります。

- シンプルで分かりやすいプログラミングモデルを使った高性能合成機能。
- よく知られているビューに似た抽象化。これにより、レイヤオブジェクトの階層を使って複雑なユーザインターフェイスを作成できます。
- 軽量のデータ構造。何百ものレイヤを同時に表示し、アニメーション化できます。
- アニメーションをアプリケーションの実行ループから独立して別々のスレッドで実行できる、抽象アニメーションインターフェイス。一度アニメーションが設定され開始されると、Core Animationがそれをフレームレートで実行する役割を果たします。
- アプリケーションのパフォーマンスの向上。アプリケーションは、コンテンツが変更されたらそれを再描画する必要があるだけです。アプリケーションのやり取りは、サイズ変更とレイアウトサービスレイヤの提供に必要な最小限のものだけです。Core Animationはまた、アニメーションのフレームレートで実行するアプリケーションコードを排除します。
- レイヤの位置とサイズを、兄弟レイヤの属性を基準に相対的に設定できるマネージャなど、柔軟性のあるレイアウトマネージャモデル。

Core Animationを使用すると、デベロッパは自身のアプリケーション用に動的なユーザインターフェイスを作成でき、最適なアニメーションパフォーマンスを得るためにOpenGLなどの低レベルのグラフィックスAPIを使う必要がありません。

## Core Animationクラス

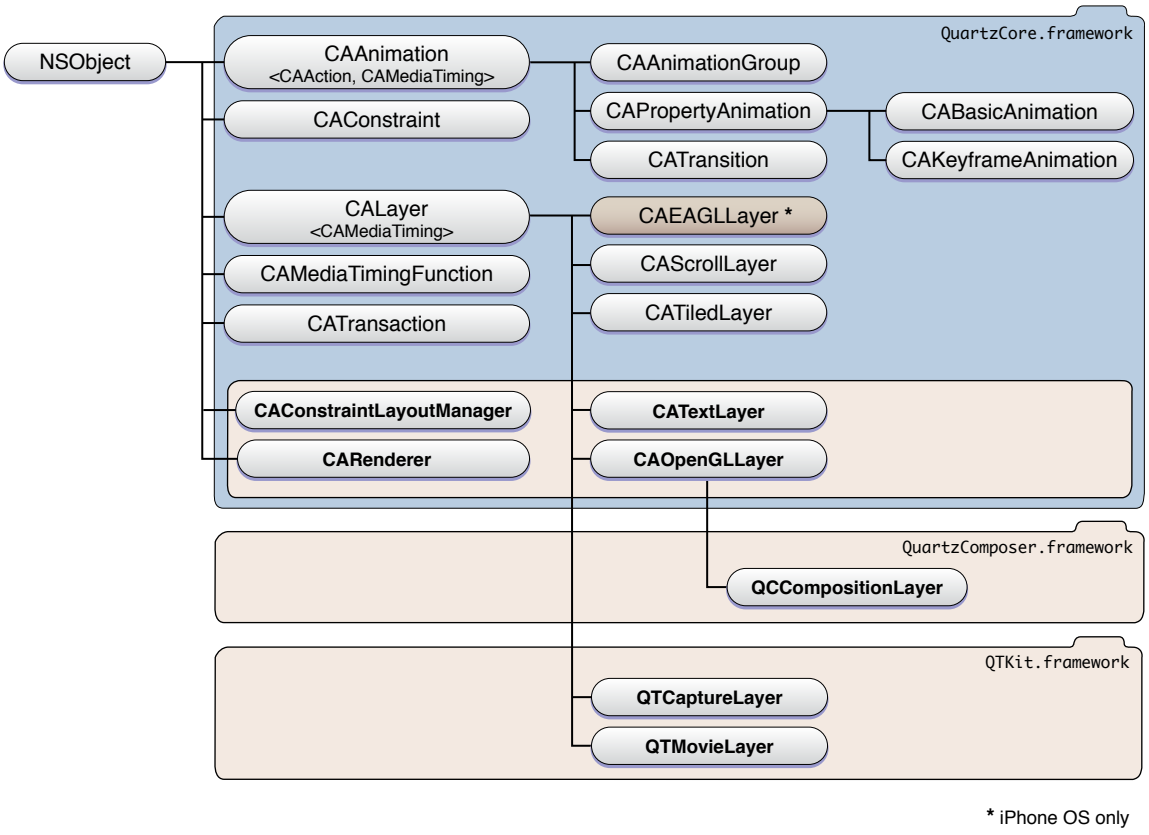
Core Animationクラスは、次の複数のカテゴリに分けられます。

- 表示用のコンテンツを提供するレイヤクラス
- アニメーションおよびタイミングクラス
- レイアウトおよび制約クラス
- 複数のレイヤの変更をアトミックな更新にまとめるトランザクションクラス


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

基本的なCore AnimationクラスはQuartz Coreフレームワークに含まれていますが、追加のレイヤクラスはほかのフレームワークで定義できます。「Core Animationのクラス階層」に、Core Animationのクラス階層を示します。

**図 1**　Core Animationのクラス階層




# レイヤクラス

レイヤクラスはCore Animationの基礎であり、`NSView`または`UIView`を使ったことのあるデベロッパには親しみやすい抽象化を提供します。基本的なレイヤ機能は、すべてのタイプのCore Animationレイヤの親クラスである`CALayer`クラスによって提供されます。

ビュークラスのインスタンスの場合と同様、`CALayer`には1つの親レイヤ（スーパーレイヤ）とサブレイヤのコレクションがあり、レイヤツリーと呼ばれるレイヤの階層を作っています。レイヤはビューと同様に背面から前面へ描画され、ローカル座標系を作成することによりスーパーレイヤを基準とした相対的なジオメトリを指定します。しかし、レイヤでは、レイヤコンテンツの回転、傾斜、拡大縮小、投影が可能な変換行列を組み込むことにより、複雑な視覚表示ができます。「レイヤジオメトリと変換」（21 ページ）で、レイヤジオメトリと変換について詳しく説明します。

`CALayer`は、コンテンツを表示するために`CALayer`をサブクラス化する必要がないという点で、Application KitビュークラスおよびCocoa Touchビュークラスとは異なります。`CALayer`インスタンスによって表示されるコンテンツは、次のようにして提供できます。

- レイヤのコンテンツプロパティを直接、またはデリゲートを通じてCore Graphicsイメージの表現に設定する。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

- Core Graphicsイメージコンテキストに直接描画するデリゲート（delegate、委任）を指定する。
- すべてのレイヤタイプが共通で持っている任意の数の視覚スタイルプロパティ（たとえば、背景色、不透明度、およびマスキング）を設定する。Mac OS Xアプリケーションは、Core Imageフィルタを使用する視覚プロパティへのアクセスも持っています。
- `CALayer`をサブクラス化して、よりカプセル化された方法で上記の手法を実装する。

「レイヤコンテンツの指定」（33 ページ）では、レイヤにコンテンツを指定する際に利用できるテクニックについて説明します。視覚スタイルプロパティとそれらがレイヤのコンテンツに適用される順番については、「レイヤスタイルプロパティ」（59 ページ）で説明します。

`CALayer`クラス以外にも、Core Animationクラスコレクションは次のクラスを提供しており、これを使ってアプリケーションは各種のコンテンツを表示できます。使用できるクラスは、Mac OS XとiOSで若干異なります。次のクラスは、Mac OS XとiOSの両方で使用できます。

- `CAScrollLayer`クラスは`CALayer`のサブクラスであり、レイヤの部分的な表示を簡素化します。`CAScrollLayer`オブジェクトのスクロール可能領域の範囲は、そのサブレイヤのレイアウトによって定義されます。`CAScrollLayer`にはキーボードまたはマウスイベント処理や、表示可能なスクローラは提供されていません。
- `CATextLayer`は、文字列または属性文字列からレイヤコンテンツを作成する簡易クラスです。
- `CATiledLayer`では、大きく複雑なイメージを段階的に表示することができます。

Mac OS Xは、上記に加えて次のクラスを提供しています。

- `CAOpenGLLayer`は、OpenGL描画環境を提供します。OpenGLを使ったコンテンツを提供するには、このクラスをサブクラス化する必要があります。コンテンツは固定的なものや、時間の経過に伴って更新されるものが可能です。
- `QCCompositionLayer`（Quartz Composerフレームワークによって提供される）は、Quartz Composerコンポジションをそのコンテンツとしてアニメーション化します。
- `QTMovieLayer`と`QTCaptureLayer`（QTKitフレームワークによって提供される）は、QuickTimeムービーとライブビデオの再生を可能にします。

iOSには、次のクラスが追加されています。

- `CAEAGLLayer`は、OpenGLES描画環境を提供します。

`CALayer`クラスは、**キー値コーディングに準拠したコンテナクラス**、すなわち、サブクラスを作成せずにキー値コーディングに準拠したメソッドを用いて属性値を格納できるクラスの概念を導入しています。`CALayer`はまた、非形式プロトコルである`NSKeyValueCoding`を拡張して、デフォルトのキー値と追加の構造体型（`CGPoint`、`CGSize`、`CGRect`、`CGAffineTransform`、`CATransform3D`）の自動オブジェクトラッピングをサポートしており、キーパスによってこれらの構造体のフィールドの多くへアクセスできる手段を提供しています。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

さらにCALayerは、レイヤに関連付けられたアニメーションとアクションの管理も行います。レイヤは、レイヤがレイヤツリーに挿入される場合と削除される場合、レイヤプロパティに変更が行われる場合、デベロッパによる明示的な要請がある場合などにアクショントリガを受け取ります。これらのアクションにより、通常はアニメーションが発生します。詳細については、「アニメーション」（39 ページ）および「レイヤアクション」（43 ページ）を参照してください。

## アニメーションおよびタイミングクラス

レイヤの視覚プロパティの多くは、暗黙的にアニメーション化可能です。単にアニメーション化可能プロパティの値を変更するだけで、レイヤは自動的に現在の値から新しい値へアニメーション化します。たとえば、レイヤの非表示プロパティをYESに設定すると、レイヤが徐々にフェードされるアニメーションがトリガされます。ほとんどのアニメーション化可能プロパティには、カスタマイズや交換が簡単なデフォルトのアニメーションが関連付けられています。アニメーション化可能プロパティとそのデフォルトのアニメーションの完全なリストは、「アニメーション化可能プロパティ」（81 ページ）に掲載されています。

アニメーション化可能プロパティは、明示的にもアニメーション化できます。プロパティを明示的にアニメーション化するには、Core Animationのアニメーションクラスの1つのインスタンスを作成し、必要な視覚効果を指定します。明示的なアニメーションでは、レイヤにおいてプロパティの値が変更されるのではなく、単に表示上でアニメーション化されるだけです。

Core Animationは、基本的なアニメーションとキーフレームアニメーションを使って、レイヤのコンテンツ全体、または選択した属性をアニメーション化できるアニメーションクラスを提供します。Core Animationのアニメーションクラスはすべて、抽象クラスのCAAnimationから派生しています。CAAnimationは、アニメーションの単純な再生時間、速度、繰り返し回数を指定するCAMediaTimingプロトコルを採用しています。CAAnimationはまた、CAActionプロトコルも採用しています。このプロトコルは、レイヤによってトリガされたアクションに応答して、アニメーションを開始する標準手段を提供します。

さらにこのアニメーションクラスは、アニメーションのペース配分を単純なベジエ曲線として記述する、タイミング関数も定義しています。たとえば、線形のタイミング関数は、アニメーションのペースがその再生時間全体に渡り均一になるように指定し、イーズアウトのタイミング関数は、アニメーションが再生時間の終わりに近づくにつれて速度を落とすように指定します。

Core Animationは、いくつかの追加の抽象および具象アニメーションクラスを提供しています。

- CATransitionは、レイヤのコンテンツ全体に作用するトランジションエフェクトを提供します。これは、アニメーション化の際にレイヤコンテンツのフェード、プッシュ、リビールを行います。独自のカスタムCore Imageフィルタを指定することによって、手持ちのトランジションエフェクトを拡張できます。

- CAAnimationGroupを使うと、アニメーションオブジェクトの配列をグループ化して、同時に実行できます。

- CAPropertyAnimationは、キーパスによって指定されたレイヤプロパティのアニメーション化を可能にする抽象サブクラスです。

- CABasicAnimationは、レイヤプロパティに対して単純な補間を提供します。

- CAKeyframeAnimationは、キーフレームアニメーションをサポートします。アニメーション化するレイヤプロパティのキーパスと、アニメーションの各ステージの値を表す値の配列、およびキーフレームの時間とタイミング関数の配列を指定します。アニメーションが実行するたびに、指定された補間を使って各値が設定されます。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

これらのアニメーションクラスは、Core AnimationとCocoa Animationの代理として使われます。これらのクラスはCore Animationに関連するため、「アニメーション」（39 ページ）で説明します。『*Animation Types and Timing Programming Guide*』ではこれらのクラスの機能を詳しく説明しています。

## レイアウトマネージャクラス

Application Kitビュークラスは、レイヤをそのスーパーレイヤを基準にして相対的に位置付ける、従来の「ストラット＆スプリング」モデルを提供します。レイヤはこのモデルをサポートしていますが、Mac OS X上のCore Animationは、デベロッパが独自のレイアウトマネージャを記述できる、より柔軟性のあるレイアウトマネージャメカニズムも提供しています。

Core Animationの`CAConstraint`クラスは、指定された一連の制約を使ってサブレイヤを整列させるレイアウトマネージャです。各制約（`CAConstraint`クラスのインスタンスによってカプセル化されます）は、あるレイヤの1つのジオメトリ属性（左端、右端、上端、下端、水平方向または垂直方向の中央）の関係を、その兄弟レイヤまたはスーパーレイヤのジオメトリ属性を基準にして相対的に表したものです。

レイアウトマネージャの概要と制約レイアウトマネージャについては、「Core Animationレイヤの配置」（51 ページ）で説明します。

## トランザクション管理クラス

レイアウトのアニメーション化可能プロパティへの変更はすべて、トランザクションの一部でなければなりません。`CATransaction`は、複数のアニメーション操作を表示へのアトミックな更新として一括処理する役割を果たすCore Animationクラスです。トランザクションはネスト化できます。

Core Animationは2つのタイプのトランザクションをサポートします。すなわち、暗黙的なトランザクションと明示的なトランザクションです。暗黙的なトランザクションは、レイヤのアニメーション化可能プロパティがアクティブなトランザクションなしでスレッドによって変更されると自動的に作成され、スレッドの実行ループの次の反復時に自動的にコミットされます。明示的なトランザクションは、レイヤの変更前にアプリケーションが`CATransaction`クラスに開始メッセージを送信し、その後コミットメッセージを送信したときに発生します。

トランザクション管理については、「トランザクション」（47 ページ）で説明します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

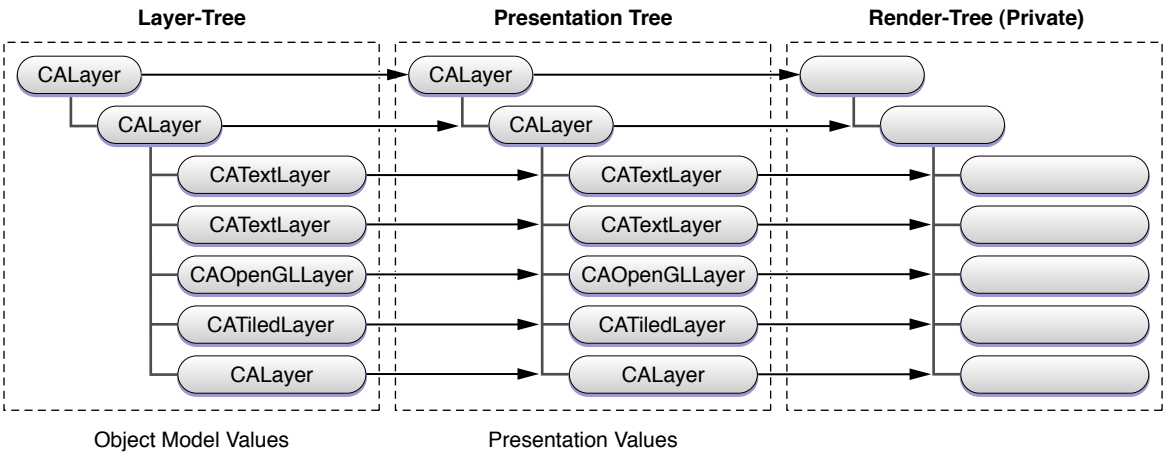
# Core Animation描画アーキテクチャ

Core AnimationのレイヤとCocoaのビューには明白な共通点がいくつかありますが、概念上の大きな違いがあり、レイヤは画面に直接的には描画しません。

`NSView`および`UIView`がModel-View-Controllerデザインパターンにおいて明白なビューオブジェクトであるのに対し、Core Animationレイヤは実際にはモデルオブジェクトです。Core Animationレイヤは、ジオメトリ、タイミング、および視覚プロパティをカプセル化し、表示するコンテンツを提供しますが、実際の表示はレイヤの役割ではありません。

可視のレイヤツリーはそれぞれ対応する2つのツリーによって支えられています。すなわち、プレゼンテーションツリーと描画ツリーです。図 1に、Mac OS Xで使用可能なCore Animationレイヤクラスを使用したレイヤツリーの例を示します。

**図 1** Core Animation描画アーキテクチャ




レイヤツリーには、各レイヤのオブジェクトモデルの値が含まれています。これらは、レイヤプロパティに値を代入するときに設定する値です。

プレゼンテーションツリーには、アニメーションを実行するたびに、ユーザに現在提示されている値が含まれます。たとえば、あるレイヤの`backgroundColor`に新しい値を設定すると、レイヤツリーの値が即座に変わります。しかしプレゼンテーションツリーの対応するレイヤの`backgroundColor`値は、ユーザに対して表示されるときに補間された色で更新されます。

描画ツリーは、レイヤを描画するときにはプレゼンテーションツリーの値を使用します。描画ツリーは、アプリケーションの動作とは独立して合成操作を実行します。つまり描画は別々のプロセスやスレッドで行われるため、アプリケーションの実行ループへの影響は最小限に抑えられます。

アニメーショントランザクションの処理中に、`CALayer`のインスタンスに、そのプレゼンテーションレイヤを問い合わせることができます。これは、現在のアニメーションを変更しようとしており、現在表示されている状態から新しいアニメーションを開始したい場合に非常に役立ちます。



2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# レイヤジオメトリと変換

この章では、レイヤのジオメトリの構成要素、要素どうしの関係、変換行列によりどのようにして複雑な視覚効果が生成されるかについて説明します。

## レイヤ座標系

レイヤの座標系は、現在のプラットフォームによって異なります。iOSでは、デフォルトの座標系の原点はレイヤの左上隅にあり、正の値は原点から下および右に向って大きくなります。Mac OS Xでは、デフォルトの座標系の原点はレイヤの左下隅にあり、正の値は原点から上および右に向って大きくなります。座標値はすべて浮動小数点として指定されます。また、特定のプラットフォーム上で作成したレイヤはいずれも、そのプラットフォームに関連付けられているデフォルトの座標系を使用します。

どのレイヤオブジェクトもそれぞれ専用の座標系を定義、維持し、レイヤ内のすべてコンテンツはこの座標系を基準にして相対的に位置決めされます。これは、レイヤコンテンツ自身とすべてのサブレイヤについても当てはまります。レイヤはそれぞれ自身の座標系を定義するため、`CALayer`クラスが、あるレイヤの座標系からほかのレイヤへの点、矩形、およびサイズの値を変換するメソッドを提供します。

レイヤ関連のいくつかのプロパティは、ユニット座標空間を使ってそれぞれの値を測定します。ユニット座標空間は、プロパティを厳密な境界矩形の値に結び付けずに、レイヤの境界矩形に対して相対的な値を指定する方法です。ユニット座標空間の特定のx座標またはy座標は、常に0.0から1.0の範囲内にあります。x軸上で0.0の値を指定すると、レイヤの左端に点が置かれ、1.0の値を指定するとレイヤの右端に点が置かれます（yの値については、どちらの値が上端でどちらの値が下端になるかは、そのプラットフォームに依存し、前述と同じ規則に従います）。(0.5, 0.5)の点は、レイヤの中心点を示します。

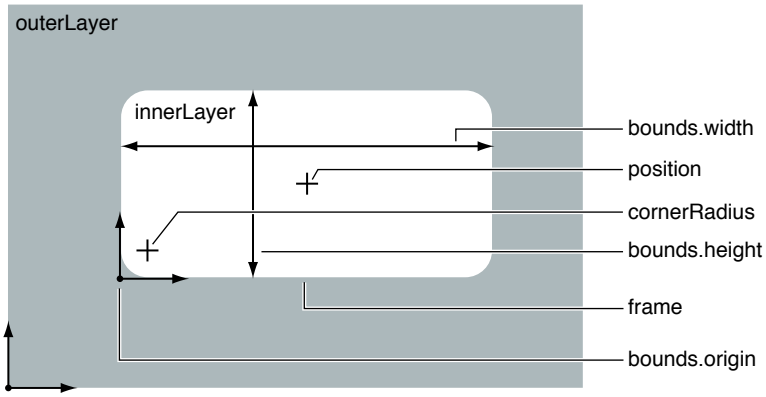
## レイヤのジオメトリの指定

レイヤとレイヤツリーは、多くの点でビューとビュー階層に似ていますが、レイヤのジオメトリは異なる方法で指定され、たいていの場合はよりシンプルです。レイヤの変換行列も含め、レイヤのジオメトリのプロパティはすべて、暗黙的にも明示的にもアニメーション化できます。

図 1に、コンテキスト内でレイヤのジオメトリを指定するプロパティを示します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 1**　CALayerジオメトリプロパティ




positionプロパティは、レイヤの位置をそのスーパーレイヤとの相対位置で指定するCGPointであり、スーパーレイヤの座標系で表現されます。

boundsプロパティは、レイヤのサイズ(bounds.size)と原点(bounds.origin)を指定するCGRectです。境界矩形の原点は、レイヤの描画メソッドをオーバーライドするときにグラフィックスコンテキストの原点として使われます。

レイヤには、positionプロパティ、boundsプロパティ、anchorPointプロパティ、およびtransformプロパティの役割を果たす暗黙のframeがあります。新しいフレーム矩形を設定すると、レイヤのpositionプロパティとboundsプロパティがそれに応じて変更されますが、フレーム自体は保存されません。新しいフレーム矩形が指定されても境界矩形の原点は変わらず、境界のサイズがフレームのサイズに設定されます。レイヤの位置は、アンカーポイントを基準に適切な位置に設定されます。frameプロパティ値を取得するときには、値はpositionプロパティ、boundsプロパティ、およびanchorPointプロパティを基準にして相対的に計算されます。

anchorPointプロパティは、レイヤの境界矩形内で位置座標に対応する位置を指定するCGPointです。アンカーポイントは、境界矩形がpositionプロパティに対してどのように位置決めされるかを指定するほか、変換処理が適用される点の役割も果たします。アンカーポイントは、ユニット座標系で表します。すなわち、値(0.0,0.0)はレイヤの原点に最も近い位置にあり、値(1.0,1.0)はその対角の位置にあります。レイヤの親（もしあれば）に変換を適用すると、親の座標系のy軸によってanchorPointの向きを変更できます。

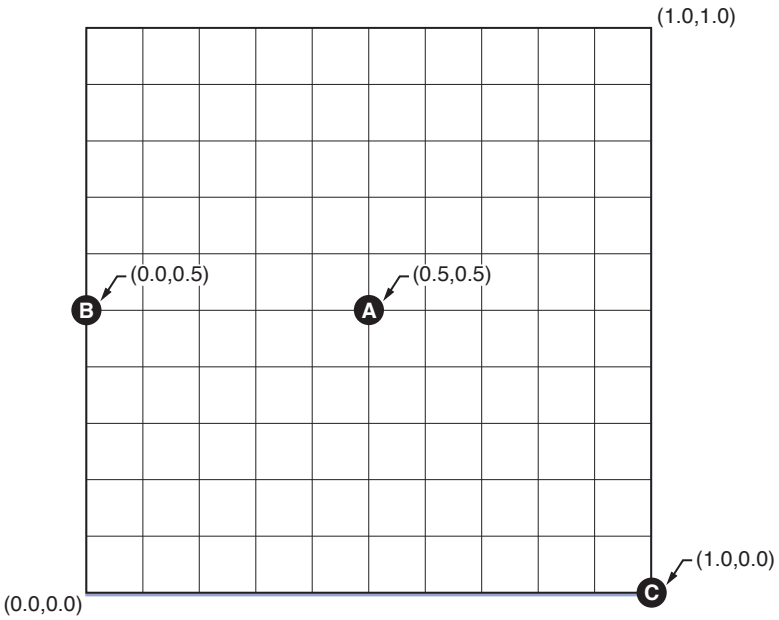
レイヤのフレームを指定するときには、positionがアンカーポイントを基準にして相対的に設定されます。レイヤのpositionを指定すると、boundsがアンカーポイントを基準にして相対的に設定されます。

**iOSにおける注意事項：** 次の例は、デフォルトの座標系の原点が左下隅に位置する、Mac OS Xのレイヤを示しています。iOSでは、レイヤの原点は左上隅に置かれ、正の値は下および右に向かって大きくなります。表示される値は変わりますが、概念は同じです。

図 2に、アンカーポイントの3つの例を示します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

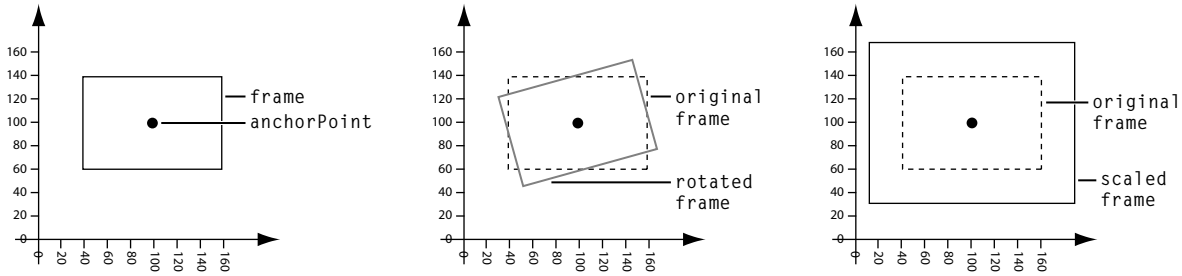
**図 2**　　3つのanchorPoint値




anchorPointのデフォルト値は(0.5,0.5)であり、これはレイヤの境界矩形の中心点に相当します（図2の点A）。点Bは、(0.0,0.5)に設定されたアンカーポイントの位置を示します。最後に点C(1.0,0.0)は、レイヤのpositionがフレームの右下角に設定されるように指定します。この図はMac OS Xでのレイヤに固有のものです。iOSでは、レイヤは異なるデフォルト座標系を使用します。(0.0,0.0)は左上隅となり、(1.0,1.0)は右下隅となります。

frameプロパティ、boundsプロパティ、positionプロパティ、およびanchorPointプロパティの関係を、図 3に示します。

**図 3**　　(0.5,000.5)のレイヤ原点




bounds = (0.0,0.0, 120.0,80.0)
frame = (40.0,60.0, 120.0,80.0)
anchorPoint = (0.5,0.5)
position = (100.0, 100.0)

rotation applied

scale applied

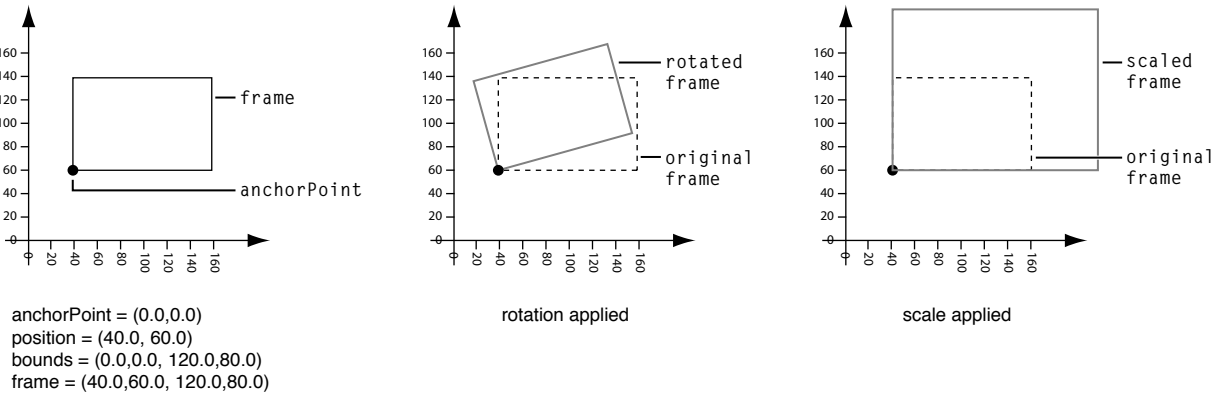
この例では、anchorPointはレイヤの中心点に相当する(0.5,0.5)のデフォルト値に設定されています。レイヤのpositionは(100.0,100.0)に設定されており、boundsは矩形(0.0, 0.0, 120.0, 80.0)に設定されています。これにより、frameプロパティは(40.0, 60.0, 120.0, 80.0)として計算されます。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

新しいレイヤを作成し、レイヤのframeプロパティだけを(40.0, 60.0, 120.0, 80.0)に設定すると、自動的にpositionプロパティは(100.0,100.0)に設定され、boundsプロパティは(0.0, 0.0, 120.0, 80.0)に設定されます。

図 4に、図 3のレイヤと同じframe矩形のあるレイヤを示します。しかし、この場合はレイヤのanchorPointは、レイヤの左下角に相当する(0.0,0.0)に設定されています。

**図 4**　(0.0,00.0)のレイヤ原点




(40.0, 60.0, 120.0, 80.0)に設定されたフレームでは、boundsプロパティの値は同じですが、positionプロパティの値は変更されています。

Cocoaのビューとは異なるレイヤジオメトリのもう1つの側面は、レイヤの角を丸める半径を指定できることです。cornerRadiusプロパティは、コンテンツの描画、サブレイヤの切り取り、およびボーダーと影の描画の際にレイヤが使用する半径を指定します。

zPositionプロパティは、レイヤの位置のZ軸コンポーネントを指定します。zPositionは、レイヤの視覚上の位置を、その兄弟レイヤを基準にして相対的に設定することを意図しています。これはレイヤの兄弟の順番を指定するためではなく、サブレイヤ配列内でレイヤの順番を変えるために使われます。

# レイヤのジオメトリの変換

レイヤのジオメトリが確定したら、行列変換を使ってこれを変換できます。Transformデータ構造体は、レイヤの回転、拡大縮小、オフセット、傾斜、レイヤへの透視変換の適用に使われる、3次元の同次変換（4×4の行列のCGFloat値）を定義します。

2つのレイヤプロパティが変換行列を指定します。すなわち、transformとsublayerTransformです。transformプロパティによって指定される行列は、レイヤとそのサブレイヤに対し、レイヤのanchorPointを基準にして相対的に適用されます。図 3は、レイヤがデフォルト値である(0.5,0.5)のanchorPointを使用しているときに、回転変換および拡大縮小変換によってどのように影響を受けるかを示しています。図 4は、(0.0,0.0)のanchorPointを使用した場合に、同じ変換行列によってレイヤがどのように影響を受けるかを示しています。sublayerTransformプロパティによって指定される行列は、レイヤ自身ではなく、レイヤのサブレイヤに対してのみ適用されます。

CATransform3Dデータ構造体は、次のいずれかの方法で作成および変更できます。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

- CATransform3D関数を使用する
- データ構造体のメンバを直接変更する
- キー値コーディングとキーパスを使用する

定数CATransform3DIdentityは同一行列、すなわち拡大縮小、回転、傾斜、透視変換の適用を行わない行列です。同一行列をレイヤに適用すると、レイヤはデフォルトのジオメトリで表示されます。

## 変換関数

Core Animationで使用可能な変換関数は、行列を操作します。これらの関数（表1に示す）を使い、レイヤまたはそのサブレイヤに適用する行列をtransformメソッドとsublayerTransformプロパティを変更してそれぞれ作成できます。変換関数には、CATransform3Dデータ構造体を操作するものと、これを返すものがあります。これにより、再利用しやすい、シンプルな変換や複雑な変換を作成できます。

**表 1**　平行移動、回転、拡大縮小のためのCATransform3D変換関数

| 関数 | 用途 |
| --- | --- |
| CATransform3DMake-Translation | '(tx, ty, tz)'の平行移動を行う変換を返します：t' = [1 0 0 0; 0 1 0 0; 0 0 1 0; tx ty tz 1] |
| CATransform3DTranslate | 't'を'(tx, ty, tz)'によって平行移動してその結果を返します：* t' = translate(tx, ty, tz) * t |
| CATransform3DMakeScale | `(sx, sy, sz)'の拡大縮小を行う変換を返します：* t' = [sx 0 0 0; 0 sy 0 0; 0 0 sz 0; 0 0 0 1] |
| CATransform3DScale | 't'を'(sx, sy, sz)'だけ拡大縮小し、その結果を返します：* t' = scale(sx, sy, sz) * t |
| CATransform3DMakeRotation | ベクトル'(x, y, z)'を中心にしてラジアン'angle'分の回転をする変換を返します。ベクトルの長さがゼロの場合は同一変換が返されます。 |
| CATransform3DRotate | ベクトル'(x, y, z)'を中心にして't'をラジアン'angle'だけ回転し、その結果を返します：t' = rotation(angle, x, y, z) * t |

回転角度は、度数ではなくラジアンで指定されます。次の関数を使って、ラジアンと度数の間を変換できます。

```
CGFloat DegreesToRadians(CGFloat degrees) {return degrees * M_PI / 180;};
CGFloat RadiansToDegrees(CGFloat radians) {return radians * 180 / M_PI;};
```

Core Animationは、行列の反転を行う変換関数CATransform3DInvertを提供しています。逆変換は一般に、変換済みのオブジェクト内の点を逆変換するために使います。逆変換は行列によって変換された値の元の値を取得する場合に便利です。つまり、行列を逆変換し、得られた値に逆変換行列を乗じた結果が元の値です。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

また、CATransform3D行列をCGAffineTransform行列で表現できる場合は、そのようにCATransform3D行列を変換する関数も提供されています。

**表 2**　CGAffineTransform変換のためのCATransform3D変換関数

| 関数 | 用途 |
| --- | --- |
| CATransform3DMake-AffineTransform | 渡されたアファイン変換と同じ効果のCATransform3Dを返します。 |
| CATransform3DIsAffine | 渡されたCATransform3Dをアファイン変換で正確に表現できる場合は、YESを返します。 |
| CATransform3DGet-AffineTransform | 渡されたCATransform3Dによって表現されたアファイン変換を返します。 |

変換行列と同一行列、または別の変換行列との同等性の比較を行う関数が提供されています。

**表 3**　同等性のテストのためのCATransform3D変換関数

| 関数 | 用途 |
| --- | --- |
| CATransform3DIsIdentity | 変換が同一変換であれば、YESを返します。 |
| CATransform3DEqualToTransform | 2つの変換がまったく同一であれば、YESを返します。 |

## 変換データ構造体の変更

ほかのデータ構造体の場合と同様に、CATransform3Dデータ構造体の任意のメンバの値を変更できます。リスト 1にはCATransform3Dデータ構造体の定義が含まれており、構造体メンバは対応する行列位置に表示されます。

**リスト 1**　CATransform3D構造体

```
struct CATransform3D
{
  CGFloat m11, m12, m13, m14;
  CGFloat m21, m22, m23, m24;
  CGFloat m31, m32, m33, m34;
  CGFloat m41, m42, m43, m44;
};

typedef struct CATransform3D CATransform3D;
```

リスト 2の例は、CATransform3Dを透視変換として構成する方法を示しています。

**リスト 2**　CATransform3Dデータ構造体の直接的な変更

```
 CATransform3D aTransform = CATransform3DIdentity;
// zDistanceの値は変換の精度に影響する。
zDistance = 850;
aTransform.m34 = 1.0 / -zDistance;
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# キーパスを使った変換の変更

Core Animationでは、キー値コーディングプロトコルが拡張されており、キーパスを通じてレイヤの`CATransform3D`行列の一般的な値の取得と設定ができるようになっています。表 4に、キー値コーディングとキー値監視に準拠しているレイヤの`transform`プロパティと`sublayerTransform`プロパティのキーパスについて説明します。

**表 4** CATransform3Dキーパス

| フィールドキーパス | 説明 |
| --- | --- |
| `rotation.x` | ラジアン単位によるX軸での回転。 |
| `rotation.y` | ラジアン単位によるY軸での回転。 |
| `rotation.z` | ラジアン単位によるZ軸での回転。 |
| `rotation` | ラジアン単位によるZ軸での回転。これは、`rotation.z`フィールドを設定することと同じです。 |
| `scale.x` | X軸の拡大縮小倍率。 |
| `scale.y` | Y軸の拡大縮小倍率。 |
| `scale.z` | Z軸の拡大縮小倍率。 |
| `scale` | 3つの拡大縮小倍率すべての平均。 |
| `translation.x` | X軸での平行移動。 |
| `translation.y` | Y軸での平行移動。 |
| `translation.z` | Z軸での平行移動。 |
| `translation` | XおよびY軸での平行移動。値はNSSizeまたはCGSizeです。 |

Objective-C 2.0のプロパティを使って構造体のフィールドキーパスを指定することはできません。次のような指定はできません。

```
myLayer.transform.rotation.x=0;
```

代わりに、次に示すように`setValue:forKeyPath:`または`valueForKeyPath:`を使う必要があります。

```
[myLayer setValue:[NSNumber numberWithInt:0] forKeyPath:@"transform.rotation.x"];
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# レイヤツリー階層

レイヤは、ビジュアルコンテンツの提供とアニメーションの管理に関して直接的な責任を負うほか、ほかのレイヤのコンテナとしての役割も果たし、これによりレイヤ階層を形成しています。

この章では、レイヤ階層と、階層内のレイヤの操作方法について説明します。

## レイヤツリー階層とは？

Core Animationのレイヤツリー階層は、Cocoaのビュー階層に相当するものです。`NSView`または`UIView`のインスタンスにスーパービューとサブビューがあるのと同様に、Core Animationレイヤにはスーパーレイヤとサブレイヤがあります。レイヤツリーには、次のようにビュー階層と同じ多くの利点があります。

- モノリシックで複雑なサブクラス化を避け、よりシンプルなレイヤを使って複雑なインターフェイスを組み立てられます。レイヤには複雑な合成機能があり、この種の「スタッキング」に非常に適しています。
- 各レイヤは、そのスーパーレイヤの座標系に対して相対的な専用の座標系を指定します。レイヤが変換されると、そのサブレイヤもレイヤ内で変換されます。
- レイヤツリーは動的です。アプリケーションを実行しながらレイヤの再設定ができます。レイヤの作成、ほかのレイヤのサブレイヤとしての追加、レイヤツリーからのレイヤの削除ができます。

## ビューでのレイヤの表示

Core Animationは、レイヤをウインドウに実際に表示する手段を提供しておらず、ビューによってレイヤをホストする必要があります。ビューとペアにする場合、ビューで対象のレイヤに対するイベント処理機能を提供しながら、レイヤではコンテンツの表示機能を提供する必要があります。

iOSのビューシステムは、Core Animationレイヤの一番上に直接作成されます。`UIView`のそれぞれのインスタンスは自動的に`CALayer`クラスのインスタンスを作成し、ビューの`layer`プロパティ値としてそれを設定します。必要に応じて、ビューのレイヤにサブレイヤを追加できます。

Mac OS X上では、`NSView`インスタンスを、レイヤをホストできるように設定する必要があります。レイヤツリーのルートレイヤを表示するには、リスト1に示すように、ビューのレイヤを設定し、次にそのビューがレイヤを使うように設定します。

**リスト1**　ビューへのレイヤの挿入

```
// theViewはウインドウの既存のビュー
// theRootLayerはレイヤツリーのルートレイヤ
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
[theView setLayer:theRootLayer];
[theView setWantsLayer:YES];
```

# レイヤの階層への追加と階層からの削除

レイヤインスタンスをインスタンス化しただけでは、レイヤツリーにレイヤは挿入されません。表1に示すメソッドを使って、レイヤツリーに対してレイヤの追加、挿入、置換、削除を行います。

**表 1**　　レイヤツリー管理メソッド

| メソッド | 結果 |
|---|---|
| `addSublayer:` | レイヤを対象レイヤのサブレイヤ配列に追加します。 |
| `insertSublayer: atIndex:` | レイヤを対象レイヤのサブレイヤとして、指定したインデックス位置に挿入します。 |
| `insertSublayer: below:` | レイヤを対象レイヤのサブレイヤ配列の指定したサブレイヤの下に挿入します。 |
| `insertSublayer: above:` | レイヤを対象レイヤのサブレイヤ配列の指定したサブレイヤの上に挿入します。 |
| `removeFromSuperlayer` | 対象レイヤを、サブレイヤ配列または対象レイヤのスーパーレイヤのマスクプロパティから削除します。 |
| `replaceSublayer: with:` | 対象レイヤのサブレイヤ配列内のレイヤを、指定した新しいレイヤに置き換えます。 |

レイヤの配列を使ってレイヤのサブレイヤを設定し、次に対象とするスーパーレイヤのサブレイヤのプロパティを設定することもできます。サブレイヤのプロパティを、レイヤオブジェクトが設定されている配列に設定する場合、それらのレイヤのスーパーレイヤが`nil`に設定されていることを確認する必要があります。

デフォルトでは、可視のレイヤツリーに対するレイヤの挿入と削除により、アニメーションが実行されます。あるレイヤがサブレイヤとして追加されると、アクション識別子`kCAOnOrderIn`に対して親レイヤから返されたアニメーションが実行されます。あるレイヤがレイヤのサブレイヤから削除されると、アクション識別子`kCAOnOrderOut`に対して親レイヤから返されたアニメーションが実行されます。あるレイヤがサブレイヤ内で置き換えられると、アクション識別子`kCATransition`に対して親レイヤから返されたアニメーションが実行されます。レイヤツリーの操作中にアニメーションを無効にしたり、任意のアクション識別子に対して使用されるアニメーションを変更したりできます。

# レイヤの位置とサイズの変更

レイヤの作成後、レイヤのジオメトリプロパティである`frame`、`bounds`、`position`、`anchorPoint`、`zPosition`の値を単に変更することにより、プログラミングによってレイヤの位置とサイズを変更できます。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

レイヤのneedsDisplayOnBoundsChangeプロパティがYESであれば、そのレイヤのコンテンツはレイヤの境界矩形が変わると再キャッシュされます。デフォルトでは、needsDisplayOnBoundsChangeプロパティはNOに設定されています。

デフォルトでは、プロパティframe、bounds、position、anchorPoint、zPositionを設定することにより、そのレイヤを新しい値に合わせてアニメーション化できます。

## レイヤの自動サイズ変更

CALayerは、スーパーレイヤの移動またはサイズ変更を受けて自動的にサブレイヤの移動とサイズ変更を行うメカニズムを提供しています。多くの場合、単にレイヤの自動サイズ変更マスクを設定すれば、アプリケーションに対する適切な動作が提供されます。

レイヤの自動サイズ変更マスクは、ビット単位のOR演算子を使ってCAAutoresizingMask定数群を組み合わせ、その結果にレイヤのautoresizingMaskプロパティを組み合わせることで指定します。表 2に、各マスク定数と、それがレイヤのサイズ変更動作にどのように影響するかを示します。

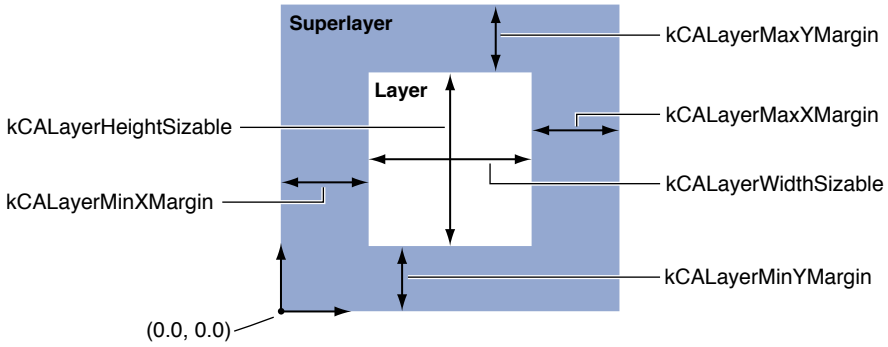
**表 2**　自動サイズ変更マスク値と説明

| 自動サイズ変更マスク | 説明 |
| --- | --- |
| kCALayerHeight-Sizable | 設定されると、レイヤの高さは、スーパーレイヤの高さの変更に比例して変更されます。それ以外の場合、レイヤの高さは、スーパーレイヤの高さに対して変更されません。 |
| kCALayerWidthSizable | 設定されると、レイヤの幅は、スーパーレイヤの幅の変更に比例して変更されます。それ以外の場合、レイヤの幅は、スーパーレイヤの高さに対して変更されません。 |
| kCALayerMinXMargin | 設定されると、レイヤの左端は、スーパーレイヤの幅の変更に比例して再配置されます。それ以外の場合、レイヤの左端は、スーパーレイヤの左端に対して同じ位置を維持します。 |
| kCALayerMaxXMargin | 設定されると、レイヤの右端は、スーパーレイヤの幅の変更に比例して再配置されます。それ以外の場合、レイヤの右端は、スーパーレイヤに対して同じ位置を維持します。 |
| kCALayerMaxYMargin | 設定されると、レイヤの上端は、スーパーレイヤの高さの変更に比例して再配置されます。それ以外の場合、レイヤの上端は、スーパーレイヤに対して同じ位置を維持します。 |
| kCALayerMinYMargin | 設定されると、レイヤの下端は、スーパーレイヤの高さの変更に比例して再配置されます。それ以外の場合、レイヤの下端は、スーパーレイヤに対して同じ位置を維持します。 |

たとえば、レイヤの位置をそのスーパーレイヤの左下角に維持するには、マスクkCALayerMaxXMargin | kCALayerMaxYMarginを使います。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。図 1に、定数値の位置を図で表します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 1**　レイヤの自動サイズ変更マスク定数




これらの定数の1つが省略されていると、レイヤのレイアウトはそのアスペクトにおいては固定されます。定数がマスクに含まれていると、レイヤのレイアウトはそのアスペクトにおいて可変になります。

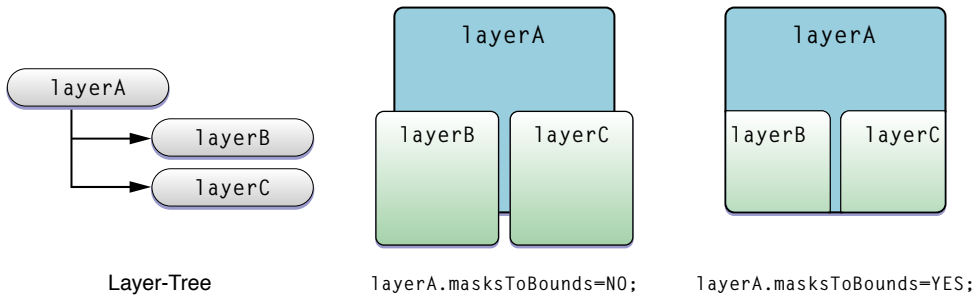
サブクラス化により、`CALayer`のメソッドである`resizeSublayersWithOldSize:`と`resizeWithOldSuperlayerSize:`をオーバーライドして、レイヤの自動サイズ変更動作をカスタマイズできます。レイヤの`resizeSublayersWithOldSize:`メソッドは、boundsプロパティが変更されるたびにレイヤによって自動的に呼び出され、各サブレイヤに`resizeWithOldSuperlayerSize:`メッセージを送ります。各サブレイヤは古い境界矩形サイズと新しいサイズを比較し、自動サイズ変更マスクに従ってその位置とサイズを調節します。

# サブレイヤの切り取り

Cocoaビューのサブビューは、親ビューの境界矩形から外れているときには親ビューに合わせて切り取られます。レイヤの場合はこの制限を取り除き、親レイヤに対する相対的な位置に関係なく、サブレイヤ全体を表示できます。

レイヤの`masksToBounds`プロパティの値によって、サブレイヤが親に合わせて切り取られるかどうかが決まります。`masksToBounds`プロパティのデフォルト値は`NO`であり、この場合サブレイヤは親に合わせて切り取られません。図 2に、`layerA`に対するmasksToBoundsの設定の結果と、それが`layerB`と`layerC`の表示にどのように影響するかを示します。

**図 2**　masksToBoundsプロパティ値の例





2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# レイヤコンテンツの指定

Cocoaビューを使って何かを表示するときには、NSViewまたはUIViewをサブクラス化し、drawRect:を実装する必要があります。しかし多くの場合は、サブクラスを必要としないでCALayerインスタンスを直接使うことができます。CALayerはキー値コーディングに準拠したコンテナクラスです。つまり、どのインスタンスにも任意の値を格納でき、多くの場合サブクラス化を完全に避けることができます。

## CALayerコンテンツの指定

CALayerインスタンスのコンテンツは、次のいずれかの方法で指定します。

- コンテンツイメージを含んでいるCGImageRefを使って、レイヤインスタンスのcontentsプロパティを明示的に設定する。
- コンテンツを提供、つまり描画するデリゲートを指定する。
- 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。

### コンテンツプロパティの設定

レイヤのコンテンツイメージは、CGImageRefへのcontentsプロパティによって指定されます。これは、レイヤの作成時（リスト 1）や、それ以外の任意の時点で別のオブジェクトから行えます。

**リスト 1**　　レイヤのコンテンツプロパティの設定

```
CALayer *theLayer;

// レイヤを作成し、境界矩形と位置を作成する
theLayer=[CALayer layer];
theLayer.position=CGPointMake(50.0f,50.0f);
theLayer.bounds=CGRectMake(0.0f,0.0f,100.0f,100.0f);

// コンテンツプロパティを、theImage（別の場所でロードされている）
// によって指定されたCGImageRefに設定する。
theLayer.contents=theImage;
```

### デリゲートを使ったコンテンツの指定

displayLayer:メソッドまたはdrawLayer:inContext:メソッドを実装するデリゲートクラスを作成し、レイヤコンテンツイメージを設定することで、レイヤのコンテンツの描画や、より優れたカプセル化を行うことができます。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

コンテンツを描画するデリゲートメソッドを実装しても、レイヤに、その実装を使用して自動的に描画させることはできません。代わりに、レイヤインスタンスに`setNeedsDisplay`または`setNeedsDisplayInRect:`メッセージを送信するか、またはレイヤインスタンスの`needsDisplayOnBoundsChange`プロパティを`YES`に設定することにより、レイヤインスタンスにコンテンツをキャッシュし直すよう、明示的に指示する必要があります。

`displayLayer:`メソッドを実装するデリゲートは、指定されたレイヤに対してどのイメージが表示されるべきかを決定し、それに応じてレイヤの`contents`プロパティを設定できます。リスト 2に示す`displayLayer:`の実装例では、**state**キーの値に応じて`theLayer`の`contents`プロパティを設定しています。`CALayer`インスタンスがキー値コーディングコンテナとして機能するため、`state`値を格納するためにサブクラス化する必要はありません。

**リスト 2**　　デリゲートメソッドdisplayLayer:の実装例

```
- (void)displayLayer:(CALayer *)theLayer
{
    // レイヤのstateキーの値をチェック
    if ([[theLayer valueForKey:@"state"] boolValue])
    {
        // yesのイメージを表示
        theLayer.contents=[someHelperObject loadStateYesImage];
    }
    else {
        // noのイメージを表示
        theLayer.contents=[someHelperObject loadStateNoImage];
    }
}
```

レイヤのコンテンツをイメージからロードせずに描画する場合は、`drawLayer:inContext:`デリゲートメソッドを実装します。デリゲートには、コンテンツを必要としているレイヤと、コンテンツの描画先となる`CGContextRef`が渡されます。

リスト 3に示す`drawLayer:inContext::`の実装例では、`theLayer`によって返された`lineWidth`キー値を使ってパスを描画します。

**リスト 3**　　デリゲートメソッドdrawLayer:inContext:の実装例

```
- (void)drawLayer:(CALayer *)theLayer
        inContext:(CGContextRef)theContext
{
    CGMutablePathRef thePath = CGPathCreateMutable();

    CGPathMoveToPoint(thePath,NULL,15.0f,15.f);
    CGPathAddCurveToPoint(thePath,
                          NULL,
                          15.f,250.0f,
                          295.0f,250.0f,
                          295.0f,15.0f);

    CGContextBeginPath(theContext);
    CGContextAddPath(theContext, thePath );

    CGContextSetLineWidth(theContext,
                          [[theLayer valueForKey:@"lineWidth"] floatValue]);
    CGContextStrokePath(theContext);
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
    // パスを解放する
    CFRelease(thePath);
}
```

## サブクラス化によるCALayerコンテンツの指定

多くの場合は不要ですが、CALayerをサブクラス化し、描画メソッドと表示メソッドを直接オーバーライドすることができます。これは通常、デリゲートを通じては提供できないようなカスタムの動作がレイヤに要求される場合に行います。

サブクラス化により、CALayer表示メソッドをオーバーライドし、レイヤのコンテンツに適切なイメージを設定できます。リスト 4に示す例は、リスト 2に示したdisplayLayer:のデリゲートの実装と同じ機能を提供します。違いは、サブクラスではCALayerのキー値コーディングコンテナ機能に依存するのではなく、stateをインスタンスプロパティとして定義する点です。

**リスト 4**　CALayer表示メソッドのオーバーライドの例

```
- (void)display
{
    // レイヤのstateキーの値をチェック
    if (self.state)
    {
        // yesのイメージを表示
        self.contents=[someHelperObject loadStateYesImage];
    }
    else {
        // noのイメージを表示
        self.contents=[someHelperObject loadStateNoImage];
    }
}
```

CALayerサブクラスは、drawInContext:をオーバーライドすることにより、レイヤのコンテンツをグラフィックスコンテキストに描画できます。リスト5に示す例では、リスト3に示したデリゲートの実装と同じコンテンツイメージを生成します。この場合も、実装の唯一の違いはサブクラスのインスタンスプロパティとして、lineWidthとlineColorが宣言される点です。

**リスト 5**　CALayer drawInContext:メソッドのオーバーライドの例

```
- (void)drawInContext:(CGContextRef)theContext
{
    CGMutablePathRef thePath = CGPathCreateMutable();

    CGPathMoveToPoint(thePath,NULL,15.0f,15.f);
    CGPathAddCurveToPoint(thePath,
                          NULL,
                          15.f,250.0f,
                          295.0f,250.0f,
                          295.0f,15.0f);

    CGContextBeginPath(theContext);
    CGContextAddPath(theContext, thePath );
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
    CGContextSetLineWidth(theContext,
                          self.lineWidth);
    CGContextSetStrokeColorWithColor(theContext,
                                     self.lineColor);
    CGContextStrokePath(theContext);
    CFRelease(thePath);
}
```

CALayerをサブクラス化し、描画メソッドの1つを実装しても、自動的に描画が行われるわけではありません。インスタンスにsetNeedsDisplayまたはsetNeedsDisplayInRect:メッセージを送信するか、またはインスタンスのneedsDisplayOnBoundsChangeプロパティをYESに設定することにより、インスタンスに明示的にコンテンツをキャッシュし直させる必要があります。

# レイヤ内のコンテンツの位置決め

CALayerのプロパティcontentsGravityにより、レイヤの境界内におけるレイヤのcontentsイメージの位置と縮尺を指定できます。デフォルトでは、コンテンツイメージはレイヤの境界矩形全体に塗りつぶされ、そのイメージに適したアスペクト比は無視されます。

contentsGravity位置決め定数を使って、イメージをレイヤの端に沿って配置するか、レイヤの角に合わせて配置するか、レイヤの境界矩形内の中心に配置するかを指定できます。ただし、位置決め定数を使用する場合は、contentsCenterプロパティは使いません。表 1 （36 ページ）に、位置決め定数とそれに対応する位置の一覧を示します。

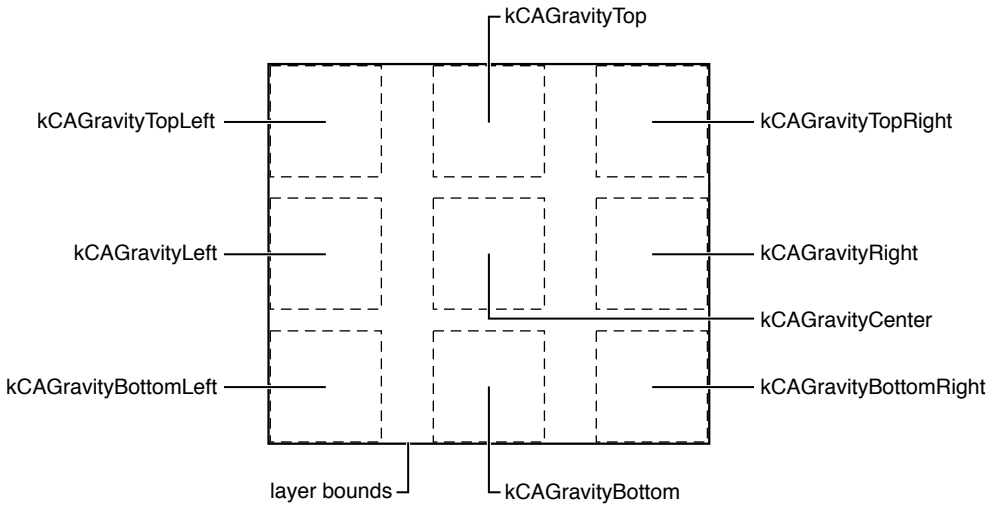
**表 1**　　レイヤのcontentsGravityプロパティに対する位置決め定数

| 位置決め定数 | 説明 |
|---|---|
| kCAGravityTopLeft | コンテンツイメージをレイヤの左上角に配置します。 |
| kCAGravityTop | コンテンツイメージを、レイヤの上端に沿って水平方向の中央に配置します。 |
| kCAGravityTopRight | コンテンツイメージをレイヤの右下角に配置します。 |
| kCAGravityLeft | コンテンツイメージを、レイヤの左端に沿って垂直方向の中央に配置します。 |
| kCAGravityCenter | コンテンツイメージをレイヤの中央に配置します。 |
| kCAGravityRight | コンテンツイメージを、レイヤの右端に沿って垂直方向の中央に配置します。 |
| kCAGravityBottomLeft | コンテンツイメージをレイヤの左下角に配置します。 |
| kCAGravityBottom | コンテンツイメージを、レイヤの下端に沿って中央に配置します。 |
| kCAGravityBottomRight | コンテンツイメージをレイヤの右下角に配置します。 |

図 1に、サポートされているコンテンツ位置とそれに対応する定数を示します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 1**　レイヤのcontentsGravityプロパティに対する位置決め定数




`contentsGravity`プロパティを表 2 （37 ページ）に示す定数の1つに設定することによって、コンテンツイメージを拡大縮小できます。`contentsCenter`プロパティがコンテンツイメージに影響を及ぼすのは、これらのサイズ変更定数の1つを使った時だけです。

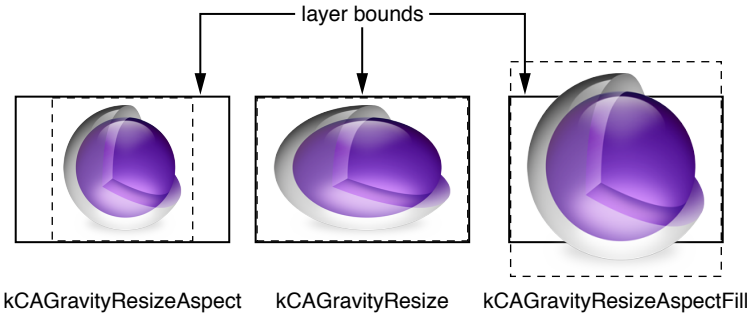
**表 2**　レイヤの`contentsGravity`プロパティに対する拡大縮小定数

| 拡大縮小定数 | 説明 |
| --- | --- |
| `kCAGravityResize` | コンテンツイメージを、レイヤの境界矩形全体を完全に塗りつぶすようにサイズ変更します。コンテンツに適したアスペクト比は無視される可能性があります。これはデフォルトです。 |
| `kCAGravityResize-Aspect` | コンテンツイメージを、アスペクト比を維持しながらレイヤの境界矩形内にできるだけ大きなサイズで表示されるように拡大縮小します。 |
| `kCAGravityResize-AspectFill` | コンテンツイメージを、アスペクト比を維持しながらレイヤの境界矩形内いっぱいに表示されるように拡大縮小します。これにより、コンテンツがレイヤの境界矩形の範囲を超えることがあります。 |

図 2は、サイズ変更モードを使うことによって、正方形のイメージが矩形のレイヤの境界内にどのように埋められるかを示しています。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 2**　レイヤのcontentsGravityプロパティに対する拡大縮小定数




**注：** `kCAGravityResize`、`kCAGravityResizeAspect`、および`kCAGravityResizeAspectFill`のいずれかの定数を使用すると、表 1 （36 ページ）の位置決め定数を無効にできます。コンテンツはレイヤの境界矩形をいっぱいに満たすため、これらの定数を使用してコンテンツの位置決めをすることはできません。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# アニメーション

アニメーションは、今日のユーザインターフェイスの重要な要素です。Core Animationを使うと、アニメーションは完全に自動化されます。アニメーションループやタイマはありません。アプリケーション側では、フレームごとの描画や、アニメーションの現在の状況の追跡を行う必要がありません。アニメーションは別々のスレッドで自動的に実行され、アプリケーションとのそれ以上のやり取りは発生しません。

この章では、アニメーションクラスの概要、および暗黙的なアニメーションと明示的なアニメーションの両方を作成する方法を説明します。

## アニメーションクラスおよびタイミング

Core Animationでは、アプリケーションで使用できる数々のアニメーションクラスが用意されています。

- `CABasicAnimation`は、レイヤプロパティの値の間で単純補間を行います。
- `CAKeyframeAnimation`は、キーフレームアニメーションをサポートします。アニメーション化するレイヤプロパティのキーパスと、アニメーションの各ステージの値を表す値の配列、およびキーフレームの時間とタイミング関数の配列を指定します。アニメーションが実行するたびに、指定された補間を使って各値が設定されます。
- `CATransition`は、レイヤのコンテンツ全体に作用するトランジションエフェクトを提供します。これは、アニメーション化の際にレイヤコンテンツのフェード、プッシュ、リビールを行います。独自のカスタムCore Imageフィルタを指定することによって、手持ちのトランジションエフェクトを拡張できます。
- `CAAnimationGroup`を使うと、アニメーションオブジェクトの配列をグループ化して、同時に実行できます。

実行するアニメーションのタイプの指定のほかに、アニメーションの再生時間、ペース（補間された値を再生時間に対してどのように配分するか）、アニメーションを繰り返すかどうかとその回数、1回のサイクルが完了するたびに自動的に反転させるかどうか、アニメーションが完了したときの視覚的な状態も指定する必要があります。アニメーションクラスと`CAMediaTiming`プロトコルは、このすべての機能と、さらに多数の機能を提供しています。

`CAAnimation`とそのサブクラス、およびタイミングプロトコルは、Core AnimationとCocoa Animation Proxy機能の両方によって共有されます。これらのクラスについては、『*Animation Types and Timing Programming Guide*』で詳しく説明されています。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# 暗黙的なアニメーション

Core Animationの暗黙的なアニメーションモデルは、アニメーション可能なレイヤプロパティへの変更はすべて、段階的で非同期に行う必要があるものと想定しています。動的にアニメーション化される画面は、明示的にアニメーション化されたレイヤがなくても可能です。アニメーション化可能なレイヤプロパティの値を変更すると、レイヤは暗黙的に古い値から新しい値への変更をアニメーション化できます。アニメーションの実行中に新しいターゲット値を設定すると、アニメーションは現在の状態から新しいターゲット値へ移行します。

リスト1に、現在の位置から新しい位置へレイヤをアニメーション化する暗黙的なアニメーション化のトリガがいかにシンプルかを示しています。

**リスト1**　レイヤの位置プロパティの暗黙的なアニメーション化

```
// レイヤの現在の位置は(100.0,100.0)にあるものとする
theLayer.position=CGPointMake(500.0,500.0);
```

一度に、あるいは何度も、単一のレイヤプロパティを暗黙的にアニメーション化できます。また、複数のレイヤを同時に明示的にアニメーション化することもできます。リスト2に示すコードでは、4つの明示的なアニメーションが同時に実行されます。

**リスト2**　複数のレイヤの複数のプロパティの暗黙的なアニメーション化

```
// レイヤ内で離れる方向に移動している間は、theLayerの不透明度を
// 0にアニメーション化する
theLayer.opacity=0.0;
theLayer.zPosition=-100;

// レイヤ内で近づく方向に移動している間は、theLayerの不透明度を
// 1にアニメーション化する
anotherLayer.opacity=1.0;
anotherLayer.zPosition=100.0;
```

暗黙的なアニメーションは、暗黙的または明示的なトランザクションにおいてオーバーライドされていない限り、そのプロパティのデフォルトのアニメーションに指定されている再生時間を使います。詳細については、「暗黙的なアニメーションの再生時間をオーバーライドする」（48ページ）を参照してください。

# 明示的なアニメーション

Core Animationは明示的なアニメーションモデルもサポートしています。明示的なアニメーションモデルの場合は、アニメーションオブジェクトの作成と、開始と終了値の設定が必要です。明示的なアニメーションは、アニメーションをレイヤに適用するまで開始されません。リスト3に示すコードは、レイヤの不透明度を完全に不透過な状態から完全に透明な状態に移行し、3秒間の再生で元に戻る、明示的なアニメーションを作成します。アニメーションは、レイヤに追加されるまでは開始しません。

**リスト3**　明示的なアニメーション

```
CABasicAnimation *theAnimation;
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
theAnimation=[CABasicAnimation animationWithKeyPath:@"opacity"];
theAnimation.duration=3.0;
theAnimation.repeatCount=2;
theAnimation.autoreverses=YES;
theAnimation.fromValue=[NSNumber numberWithFloat:1.0];
theAnimation.toValue=[NSNumber numberWithFloat:0.0];
[theLayer addAnimation:theAnimation forKey:@"animateOpacity"];
```

明示的なアニメーションは、連続的に実行するアニメーションを作成する場合に特に役立ちます。リスト4に、CoreImageブルームフィルタをレイヤに適用し、その輝度をアニメーション化する明示的なアニメーションを作成する方法を示します。これにより「選択レイヤ」を振動させ、ユーザの注目を引くことができます。

**リスト 4**　　連続的に実行する明示的なアニメーションの例

```
// この選択レイヤは連続的に振動する。
// レイヤにブルームフィルタを設定することでこれを達成する。

// フィルタを作成し、デフォルト値を設定する
CIFilter *filter = [CIFilter filterWithName:@"CIBloom"];
[filter setDefaults];
[filter setValue:[NSNumber numberWithFloat:5.0] forKey:@"inputRadius"];

// keypathを使ってフィルタのinputIntensity属性をアニメーション化できるように
// フィルタに名前を付ける
[filter setName:@"pulseFilter"];

// フィルタを選択レイヤのフィルタに設定する
[selectionLayer setFilters:[NSArray arrayWithObject:filter]];

// 振動を処理するアニメーションを作成する。
CABasicAnimation* pulseAnimation = [CABasicAnimation animation];

// アニメーション化する属性は
// pulseFilterのinputIntensity
pulseAnimation.keyPath = @"filters.pulseFilter.inputIntensity";

// これを値0から1までアニメーション化する
pulseAnimation.fromValue = [NSNumber numberWithFloat:0.0];
pulseAnimation.toValue = [NSNumber numberWithFloat:1.5];

// 1回の再生時間は1秒間で、実行回数は
// 無制限
pulseAnimation.duration = 1.0;
pulseAnimation.repeatCount = HUGE_VALF;

// フェードオンとフェードオフを実行したいため、
// 自動的にオートリバースする必要がある。これにより輝度の入力は
// 0、1、0の順番になる
pulseAnimation.autoreverses = YES;

// イージーイン、イージーアウトのタイミング曲線を使用..
pulseAnimation.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut];

// 選択レイヤにアニメーションを追加する。これにより
// アニメーションが開始する。アニメーションキー名として
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
// pulseAnimationを使用する
[selectionLayer addAnimation:pulseAnimation forKey:@"pulseAnimation"];
```

## 明示的な開始と停止

明示的なアニメーションを開始するには、`addAnimation:forKey:`メッセージをターゲットレイヤに送り、パラメータとしてアニメーションと識別子を渡します。明示的なアニメーションは、一度ターゲットレイヤに追加されると、アニメーションが完了するかレイヤから削除されるまで実行します。アニメーションをレイヤに追加するために使われた識別子は、`removeAnimationForKey:`を呼び出してアニメーションを停止するときにも使われます。レイヤに`removeAllAnimations`メッセージを送ることによって、レイヤのすべてのアニメーションを停止できます。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# レイヤアクション

レイヤアクションは、レイヤがレイヤツリーに挿入される場合とツリーから削除される場合、レイヤプロパティの値が変更される場合、アプリケーションから明示的に要請がある場合に発生します。通常は、アクショントリガの結果としてアニメーションが表示されます。

## アクションオブジェクトの役割

アクションオブジェクトは、CAActionプロトコルを通じてアクション識別子に応答するオブジェクトです。アクション識別子は、標準的なドットで区切られたキーパスを使って命名されます。レイヤは、アクション識別子を適切なアクションオブジェクトにマップする役割を果たします。識別子に対応するアクションオブジェクトが見つかると、オブジェクトにはCAActionプロトコルで定義されているメッセージが送信されます。

CALayerクラスは、デフォルトのアクションオブジェクト（CAActionプロトコルに準拠したクラスであるCAAnimationのインスタンス）をアニメーション化可能なレイヤプロパティに対して提供します。CALayerはまた、直接的にはプロパティにリンクしていない次のアクショントリガと、表 1 に示すアクション識別子も定義しています。

**表 1**　　アクショントリガとその識別子

| トリガ | アクション識別子 |
| --- | --- |
| レイヤが可視のレイヤツリーに挿入されるか、hiddenプロパティがNOに設定される。 | アクション識別子定数kCAOnOrderIn。 |
| レイヤが可視のレイヤツリーから削除されるか、hiddenプロパティがYESに設定される。 | アクション識別子定数kCAOnOrderOut。 |
| レイヤがreplaceSublayer: with:を使って可視のレイヤツリー内で既存のレイヤと置き換わる。 | アクション識別子定数kCATransition。 |

## アクションキー向け検索パターンの定義

アクショントリガが発生すると、レイヤのactionForKey:メソッドが呼び出されます。このメソッドは、パラメータとして渡されたアクション識別子に対応するアクションオブジェクトを返すか、アクションオブジェクトが存在しない場合はnilを返します。

ある識別子の照合のためにactionForKey:のCALayerの実装が呼び出されると、次の検索パターンが使われます。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

1. レイヤにデリゲートがあり、メソッドactionForLayer:forKey:が実装されていればこれが呼び出され、パラメータとしてレイヤとアクション識別子が渡されます。デリゲートのactionForLayer:forKey:の実装は、次のように応答する必要があります。
    - アクション識別子に対応するアクションオブジェクトを返す。
    - アクション識別子を処理しない場合はnilを返す。
    - アクション識別子を処理せず、検索を終了する必要がある場合は、NSNullを返す。
2. レイヤのactionsディクショナリを対象に、アクション識別子に対応するオブジェクトが検索されます。
3. 目的の識別子が含まれているactionsディクショナリを対象に、レイヤのstyleプロパティが検索されます。
4. レイヤのクラスにdefaultActionForKey:メッセージが送られます。識別子に対応するアクションオブジェクトを返すか、または見つからない場合はnilを返します。

# CAActionプロトコルの採用

CAActionプロトコルは、アクションオブジェクトがどのように呼び出されるかを定義します。CAActionプロトコルを実装するクラスには、シグネチャrunActionForKey:object:arguments:のメソッドがあります。

アクションオブジェクトがrunActionForKey:object:arguments:メッセージを受け取るときには、アクション識別子、そのアクションが発生すべきレイヤ、オプションのパラメータのディクショナリが渡されます。

通常、アクションオブジェクトはCAAnimationサブクラスのインスタンスであり、CAActionプロトコルを実装しています。ただし、このプロトコルを実装するあらゆるクラスのインスタンスを返すことができます。そのインスタンスは、runActionForKey:object:arguments:メッセージを受け取ったら、対応するアクションを実行することによって応答しなければなりません。

CAAnimationのインスタンスがrunActionForKey:object:arguments:メッセージを受け取ると、自身をレイヤのアニメーションに追加することによって応答し、その結果アニメーションが実行されます（リスト 1 （44 ページ）を参照）。

**リスト 1**　アニメーションを開始するrunActionForKey:object:arguments:の実装

```
- (void)runActionForKey:(NSString *)key
                 object:(id)anObject
              arguments:(NSDictionary *)dict
{
     [(CALayer *)anObject addAnimation:self forKey:key];
}
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# 暗黙的なアニメーションのオーバーライド

アクション識別子に対して別の暗黙的なアニメーションを指定することができます。これを行うには、CAAnimationのインスタンスをactionsディクショナリまたはstyleディクショナリのアクションディクショナリに挿入するか、デリゲートメソッドactionForLayer:forKey:を実装するか、レイヤクラスをサブクラス化してdefaultActionForKey:をオーバーライドし、該当するアクションオブジェクトを返すようにします。

リスト2に示す例では、デリゲートを使ってcontentsプロパティのデフォルトの暗黙的なアニメーションを置き換えています。

**リスト 2**　　コンテンツプロパティの暗黙的なアニメーション

```
- (id<CAAction>)actionForLayer:(CALayer *)theLayer
                        forKey:(NSString *)theKey
{
    CATransition *theAnimation=nil;

    if ([theKey isEqualToString:@"contents"])
    {

        theAnimation = [[CATransition alloc] init];
        theAnimation.duration = 1.0;
        theAnimation.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseIn];
        theAnimation.type = kCATransitionPush;
        theAnimation.subtype = kCATransitionFromRight;
    }

    return theAnimation;
}
```

リスト 3 （45 ページ）に示す例では、デリゲートを使ってactionsディクショナリパターンを使ってsublayersプロパティのデフォルトの暗黙的なアニメーションを無効にしています。

**リスト 3**　　サブレイヤプロパティの暗黙的なアニメーション

```
// 現在のアクションディクショナリの可変のバージョンを取得する
NSMutableDictionary *customActions=[NSMutableDictionary
dictionaryWithDictionary:[theLayer actions]];

// サブレイヤ用の新しいアクションを追加する
[customActions setObject:[NSNull null] forKey:@"sublayers"];

// theLayerアクションを、更新したディクショナリに設定する
theLayer.actions=customActions;
```

# アクションを一時的に無効にする

デフォルトでは、アニメーション化可能なプロパティを変更するといつでも適当なアニメーションが発生します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

トランザクションを使ってレイヤのプロパティを変更するときに、アクションを一時的に無効にすることができます。詳細については「レイヤアクションを一時的に無効にする」（48 ページ）を参照してください。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# トランザクション

レイヤへの変更はすべて、トランザクションの一部として行われます。CATransactionは、複数のレイヤツリーへの変更を描画ツリーへのアトミックな更新として一括処理する役割を果たすCore Animationクラスです。

この章では、Core Animationがサポートする2つのタイプのトランザクション、すなわち、暗黙的なトランザクションと明示的なトランザクションについて説明します。

## 暗黙的なトランザクション

暗黙的なトランザクションは、レイヤツリーがアクティブなトランザクションなしでスレッドによって変更されると自動的に作成され、スレッドの実行ループの次の反復時に自動的にコミットされます。

リスト 1の例では、結果のアニメーションが同時に実行されるようにする暗黙的なトランザクションを使用して、レイヤのopacityプロパティ、zPositionプロパティ、およびpositionプロパティを変更しています。

**リスト 1**　　暗黙的なトランザクションを使ったアニメーション

```
theLayer.opacity=0.0;
theLayer.zPosition=-200;
thelayer.position=CGPointMake(0.0,0.0);
```

**重要：** 実行ループのないスレッドからレイヤプロパティを変更する場合は、明示的なトランザクションを使う必要があります。

## 明示的なトランザクション

レイヤツリーの変更前にCATransactionクラスにbeginメッセージを送り、その後commitメッセージを送ることにより、明示的なトランザクションを作成します。明示的なトランザクションは、一度に多数のレイヤのプロパティを設定する場合（たとえば複数レイヤのレイアウト時など）、レイヤアクションを一時的に無効にする場合、あるいは結果として生じた暗黙的なアニメーションの再生時間を一時的に変更する場合などに特に役立ちます。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

## レイヤアクションを一時的に無効にする

トランザクションのkCATransactionDisableActionsの値をtrueに設定することによって、レイヤのプロパティ値の変更時にレイヤのアクションを一時的に無効にすることができます。そのトランザクションの範囲内でどのような変更が行われても、アニメーションは生じません。リスト2に、可視のレイヤツリーからaLayerを削除したときに発生するフェードアニメーションを無効にする例を示します。

**リスト2**　レイヤのアクションを一時的に無効にする

```
[CATransaction begin];
[CATransaction setValue:(id)kCFBooleanTrue
                 forKey:kCATransactionDisableActions];
[aLayer removeFromSuperlayer];
[CATransaction commit];
```

## 暗黙的なアニメーションの再生時間をオーバーライドする

トランザクションのkCATransactionAnimationDurationキーの値に異なる再生時間を設定することによって、レイヤプロパティの変更を受けて実行するアニメーションの再生時間を一時的に変更できます。そのトランザクションの範囲内で生じるすべてのアニメーションは、自身の時間ではなくその再生時間を使用します。リスト3に、アニメーションをzPositionおよびopacityアニメーションによって指定された再生時間よりも10秒長く再生させる例を示します。

**リスト3**　アニメーションの再生時間のオーバーライド

```
[CATransaction begin];
[CATransaction setValue:[NSNumber numberWithFloat:10f]
                 forKey:kCATransactionAnimationDuration];
theLayer.zPosition=200.0;
theLayer.opacity=0.0;
[CATransaction commit];
```

上記の例は、明示的なトランザクションであるbeginとcommitによって囲まれた再生時間を示していますが、これらを省略して暗黙的なトランザクションを代わりに使うことも考えられます。

## トランザクションのネスト

明示的なトランザクションをネストし、アニメーションの一部に対してアクションを無効にすることや、変更されたプロパティの暗黙的なアニメーションに対して異なる再生時間を使用することなどができます。最も外側のトランザクションがコミットされた場合のみ、アニメーションが発生します。

リスト4に、2つのトランザクションのネストの例を示します。外側のトランザクションは、暗黙的なアニメーションの再生時間を2秒に設定し、レイヤのpositionプロパティを設定します。内側のトランザクションは、暗黙的なアニメーションの再生時間を5秒に設定し、レイヤのopacityとzPositionを設定します。

**リスト4**　明示的なトランザクションのネスト

```
[CATransaction begin]; // 外側のトランザクション
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
// アニメーションの再生時間を2秒に変更する
[CATransaction setValue:[NSNumber numberWithFloat:2.0f]
                 forKey:kCATransactionAnimationDuration];
// レイヤを新しい位置に移動する
theLayer.position = CGPointMake(0.0,0.0);

[CATransaction begin]; // 内側のトランザクション
// change the animation duration to 5 seconds
[CATransaction setValue:[NSNumber numberWithFloat:5f]
                  forKey:kCATransactionAnimationDuration];

// zPositionと不透明度を変更する
theLayer.zPosition=200.0;
theLayer.opacity=0.0;

[CATransaction commit]; // 内側のトランザクション

[CATransaction commit]; // 外側のトランザクション
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# Core Animationレイヤの配置

NSViewは、ビューのサイズを変更する場合にそのスーパーレイヤを基準にして相対的に位置変更を行う、従来の「ストラット＆スプリング」モデルを提供します。レイヤはこのモデルをサポートしていますが、Mac OS X上のCore Animationは、デベロッパが独自のレイアウトマネージャを記述できる、より一般的なレイアウトマネージャメカニズムを提供しています。カスタムレイアウトマネージャ（CALayoutManagerプロトコルを実装）をレイヤに対して指定できます。これは次にレイヤのサブレイヤのレイアウトを指定する役割を担います。

この章では、制約レイアウトマネージャと一連の制約の設定方法について説明します。

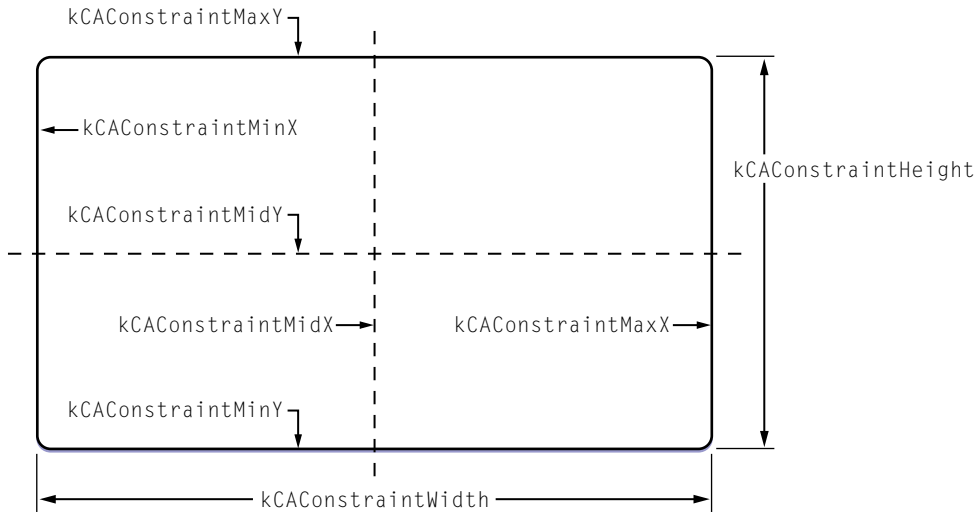
**iOSにおける注意事項：** iOSのCALayerクラスがサポートしているのは、モデルの位置変更を行う「ストラット＆スプリング」だけです。カスタムレイアウトマネージャは提供していません。ただし、特定のビューに関連付けられたレイヤを手動で配置する場合は、ビューのlayoutSubviewsメソッドをオーバーライドして独自のレイアウトコードをそこに実装できます。iOSアプリケーションにおけるビューベースのレイアウトの処理に関する詳細については、『*View Programming Guide for iOS*』を参照してください。

## 制約レイアウトマネージャ

制約に基づくレイアウトでは、レイヤの位置とサイズを、その兄弟レイヤやスーパーレイヤとの関係を使って指定できます。関係は、サブレイヤのconstraintsプロパティ内の配列に格納されているCAConstraintクラスのインスタンスによって表されます。

図 1に、関係を指定するときに使用できるレイアウト属性を示します。

**図 1**　制約レイアウトマネージャの属性





2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

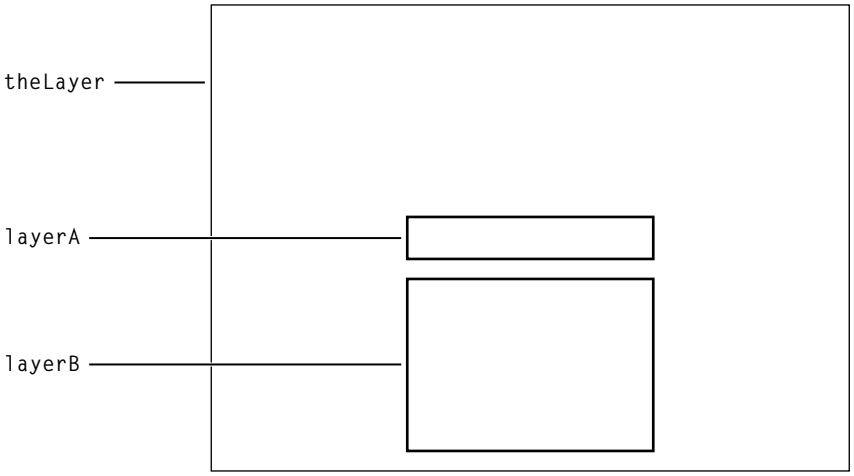
制約レイアウトを使用する場合は、まずCAConstraintLayoutManagerのインスタンスを作成して、これを親レイヤのレイアウトマネージャとして設定します。次に、CAConstraintオブジェクトをインスタンス化してこれらをaddConstraint:を使用してサブレイヤの制約に追加することによって、サブレイヤの制約を作成します。各CAConstraintインスタンスは同じ軸上の2つのレイヤ間で1つのジオメトリ関係をカプセル化します。

兄弟レイヤは、レイヤのnameプロパティを使って名前によって参照されます。特殊な名前であるsuperlayerは、スーパーレイヤを参照するために使用されます。

2つの関係の最大値を各軸に指定しなければなりません。レイヤの左右の端に制約を指定すると、幅が変わります。左端と幅に制約を指定すると、レイヤの右端がスーパーレイヤのフレームを基準にして相対的に移動します。多くの場合は、1つの端だけに制約を指定し、同じ軸のレイヤのサイズは2番目の関係として使用されます。

リスト 1に示すコード例では、レイヤを作成し、次に制約を使用して位置決めされたサブレイヤを追加します。図 2に、結果のレイアウトを示します。

**図 2**　制約に基づくレイアウトの例




**リスト 1**　レイヤの制約の設定

```
// theLayerの制約レイアウトマネージャを作成して設定する
theLayer.layoutManager=[CAConstraintLayoutManager layoutManager];

CALayer *layerA = [CALayer layer];
layerA.name = @"layerA";

layerA.bounds = CGRectMake(0.0,0.0,100.0,25.0);
layerA.borderWidth = 2.0;

[layerA addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMidY
                                                 relativeTo:@"superlayer"
                                                  attribute:kCAConstraintMidY]];

[layerA addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMidX
                                                 relativeTo:@"superlayer"
                                                  attribute:kCAConstraintMidX]];
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
[theLayer addSublayer:layerA];

CALayer *layerB = [CALayer layer];
layerB.name = @"layerB";
layerB.borderWidth = 2.0;

[layerB addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintWidth
                                                 relativeTo:@"layerA"
                                                  attribute:kCAConstraintWidth]];

[layerB addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMidX
                                                 relativeTo:@"layerA"
                                                  attribute:kCAConstraintMidX]];


[layerB addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMaxY
                                                 relativeTo:@"layerA"
                                                  attribute:kCAConstraintMinY
                                                     offset:-10.0]];

[layerB addConstraint:[CAConstraint constraintWithAttribute:kCAConstraintMinY
                                                 relativeTo:@"superlayer"
                                                  attribute:kCAConstraintMinY
                                                     offset:+10.0]];

[theLayer addSublayer:layerB];
```

このコードが実行することを以下に示します。

1. CAConstraintLayoutManagerのインスタンスを作成し、これをtheLayerのlayoutManagerプロパティとして設定します。
2. CALayer (layerA)のインスタンスを作成し、レイヤのnameプロパティを「layerA」に設定します。
3. layerAの境界矩形は(0.0,0.0,100.0,25.0)に設定されます。
4. CAConstraintオブジェクトを作成し、これをlayerAの制約として追加します。

   この制約は、layerAの水平方向の中心をスーパーレイヤの水平方向の中心に合わせます。
5. 2番目のCAConstraintオブジェクトを作成し、これをlayerAの制約として追加します。

   この制約は、layerAの垂直方向の中心をスーパーレイヤの垂直方向の中心に合わせます。
6. layerAをtheLayerのサブレイヤとして追加します。
7. CALayer (layerB)のインスタンスを作成し、レイヤのnameプロパティを「layerB」に設定します。
8. CAConstraintオブジェクトを作成し、これをlayerAの制約として追加します。

   この制約は、layerAの幅に対してlayerBの幅を設定します。
9. 2番目のCAConstraintオブジェクトを作成し、これをlayerBの制約として追加します。

   この制約は、layerBの水平方向の中心をlayerAの水平方向の中心と同じになるように設定します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

10. 3番目のCAConstraintオブジェクトを作成し、これをlayerBの制約として追加します。

    この制約は、layerBの上の端をlayerAの下の端より10ポイント下に設定します。

11. 4番目のCAConstraintオブジェクトを作成し、これをlayerBの制約として追加します。

    この制約は、layerBの下端をスーパーレイヤの下端より10ポイント上に設定します。

**警告：** 制約は、同じ属性の循環参照となるように作成することもできます。レイアウトを算出できない場合の動作は定義されていません。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# キー値コーディングへのCore Animationの拡張

CAAnimationクラスとCALayerクラスは、NSKeyValueCodingプロトコルを拡張して、キーのデフォルト値、拡張したラッピング規則、およびキーパスのサポートをCGPoint、CGRect、CGSize、およびCATransform3Dに対して追加します。

## キー値コーディング準拠のコンテナクラス

CALayerとCAAnimationはどちらもキー値コーディング準拠のコンテナクラスです。任意のキーに値を設定し、アーカイブできます。つまり、「someKey」はCALayerクラスの宣言されたプロパティではありませんが、キー「someKey」に次のように値を設定できます。

```
[theLayer setValue:[NSNumber numberWithInteger:50] forKey:@"someKey"];
```

キー「someKey」の値は、次のコードを使用して取得します。

```
someKeyValue=[theLayer valueForKey:@"someKey"];
```

**Mac OS Xにおける注意事項：** Mac OS Xでは、CALayerクラスとCAAnimationクラスはNSCodingプロトコルをサポートしており、これらのクラスのインスタンス用に設定したすべての追加のキーを自動的にアーカイブします。

## デフォルト値のサポート

Core Animationでは、キー値コーディングに新しい規則を追加しており、これによりクラスは、対象となるキーに値が設定されていない場合にデフォルト値を提供できます。CALayerとCAAnimationはどちらも、クラスメソッドdefaultValueForKey:を使ってこの規則をサポートします。

あるキーにデフォルト値を指定するには、クラスのサブクラスを作成してdefaultValueForKey:をオーバライドします。サブクラスの実装がそのキーパラメータを調べて適切なデフォルト値を返します。リスト 1に、レイヤプロパティmasksToBoundsに新しいデフォルト値を指定するdefaultValueForKey:の実装例を示します。

**リスト 1**　　defaultValueForKey:の実装例

```
+ (id)defaultValueForKey:(NSString *)key
{
    if ([key isEqualToString:@"masksToBounds"])
         return [NSNumber numberWithBool:YES];

    return [super defaultValueForKey:key];
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
}
```

# ラッピング規則

キー値コーディングメソッドを使って、標準のキー値コーディングのラッピング規則でサポートされるオブジェクト以外の値を持つプロパティにアクセスする場合は、次のラッピング規則を使用します。

| Cの型 | クラス |
| --- | --- |
| `CGPoint` | `NSValue` |
| `CGSize` | `NSValue` |
| `CGRect` | `NSValue` |
| `CGAffineTransform` | `NSAffineTransform`（Mac OS Xのみ） |
| CATransform3D | `NSValue` |

# 構造体フィールドのためのキーパスサポート

`CAAnimation`は、キーパスを使って選択した構造体のフィールドにアクセスするためのサポートを提供しています。これは、これらの構造体フィールドをアニメーションのキーパスとして指定したり、`setValue:forKeyPath:`と`valueForKeyPath:`を使って値の設定と取得を行う場合に役立ちます。

`CATransform3D`は次のフィールドを公開します。

| 構造体フィールド | 説明 |
| --- | --- |
| `rotation.x` | ラジアン単位によるX軸での回転。 |
| `rotation.y` | ラジアン単位によるY軸での回転。 |
| `rotation.z` | ラジアン単位によるZ軸での回転。 |
| `rotation` | ラジアン単位によるZ軸での回転。これは、`rotation.z`フィールドを設定することと同じです。 |
| `scale.x` | X軸の拡大縮小倍率。 |
| `scale.y` | Y軸の拡大縮小倍率。 |
| `scale.z` | Z軸の拡大縮小倍率。 |
| `scale` | 3つの拡大縮小倍率すべての平均。 |
| `translation.x` | X軸での平行移動。 |


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

| 構造体フィールド | 説明 |
| --- | --- |
| translation.y | Y軸での平行移動。 |
| translation.z | Z軸での平行移動。 |
| translation | XおよびY軸での平行移動。値はNSSizeまたはCGSizeです。 |

CGPointは次のフィールドを公開します。

| 構造体フィールド | 説明 |
| --- | --- |
| x | 点のX成分。 |
| y | 点のY成分。 |

CGSizeは次のフィールドを公開します。

| 構造体フィールド | 説明 |
| --- | --- |
| width | サイズの幅の成分。 |
| height | サイズの高さの成分。 |

CGRectは次のフィールドを公開します。

| 構造体フィールド | 説明 |
| --- | --- |
| origin | 矩形の原点をCGPointとして指定します。 |
| origin.x | 矩形の原点のX成分。 |
| origin.y | 矩形の原点のY成分。 |
| size | 矩形のサイズをCGSizeとして指定します。 |
| size.width | 矩形サイズの幅の成分。 |
| size.height | 矩形サイズの高さの成分。 |

Objective-C 2.0のプロパティを使って構造体のフィールドキーパスを指定することはできません。次のような指定はできません。

```
    myLayer.transform.rotation.x=0;
```

代わりに、次に示すようにsetValue:forKeyPath:またはvalueForKeyPath:を使う必要があります。

```
    [myLayer setValue:[NSNumber numberWithInt:0]
forKeyPath:@"transform.rotation.x"];
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# レイヤスタイルプロパティ

レイヤのスタイルプロパティは、レイヤが表示するメディアの種類に関係なく、描画ツリーによって、レイヤを合成するときに適用されます。

この章では、レイヤスタイルプロパティについて説明し、サンプルのレイヤでその効果の例を示します。

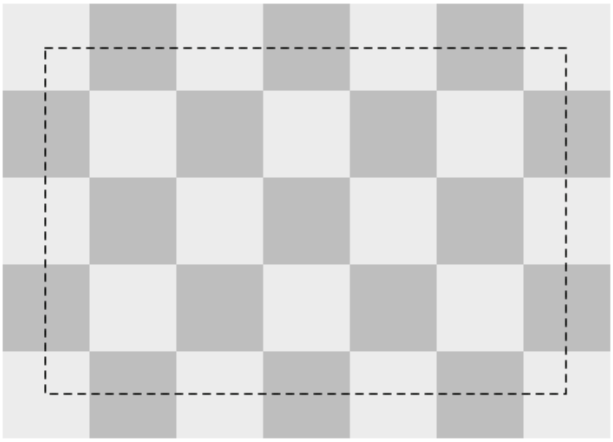
**注：** Mac OS XとiOSとで使用できるレイヤスタイルプロパティは異なり、以下で説明します。

## ジオメトリプロパティ

レイヤのジオメトリプロパティは、そのレイヤが親レイヤを基準にして相対的にどのように表示されるかを指定するものです。ジオメトリはまた、レイヤの角を丸める半径と、レイヤやそのサブレイヤに適用する変換方法を指定します。

図 1に、サンプルのレイヤのジオメトリを示します。

**図 1**　レイヤジオメトリ



次の`CALayer`プロパティがレイヤのジオメトリを指定します。

- `frame`
- `bounds`
- `position`
- `anchorPoint`


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

- cornerRadius
- transform
- zPosition

**iOSにおける注意事項：** cornerRadiusプロパティは、iOS 3.0以降でのみサポートされています。

# 背景プロパティ

次にこのレイヤに背景を描画します。背景色はCore Imageフィルタと同じように定義できます。

図 2に、backgroundColorが設定されたサンプルレイヤを示します。

**図 2** 背景色付きのレイヤ

背景フィルタは、対象レイヤの背後のコンテンツに適用されます。たとえば、ブラーフィルタを背景フィルタに適用すると、レイヤのコンテンツをより目立たせることができます。

次のCALayerプロパティがレイヤのボーダーの背景に影響します。

- backgroundColor
- backgroundFilters


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

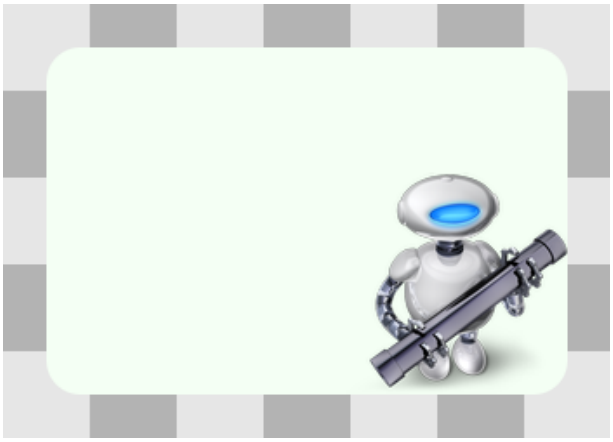
**iOSにおける注意事項：** iOSの`CALayer`クラスが`backgroundColor`プロパティを公開している間、Core Imageは使用できません。このプロパティで使用可能なフィルタは現在定義されていません。

# レイヤコンテンツ

次に、レイヤのコンテンツが設定されていれば、これを描画します。レイヤコンテンツは、Quartzグラフィック環境、OpenGL、QuickTime、またはQuartz Composerを使って作成できます。

図 3に、コンテンツを合成したサンプルレイヤを示します。

**図 3**　コンテンツイメージを表示したレイヤ



デフォルトでは、レイヤのコンテンツは、その境界矩形や角の丸みに合わせて切り取られることはありません。`masksToBounds`プロパティを`true`に設定すると、レイヤコンテンツをその値に切り取ることができます。

次の`CALayer`プロパティがレイヤのコンテンツの表示に影響します。

- `contents`
- `contentsGravity`

# サブレイヤコンテンツ

レイヤには、子レイヤの階層であるサブレイヤがあるのが一般的です。これらのサブレイヤは、親レイヤのジオメトリを基準にして、再帰的に描画されます。各サブレイヤに、親レイヤの`sublayerTransform`が親レイヤのアンカーポイントを基準にして相対的に適用されます。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 4**　サブレイヤのコンテンツを表示したレイヤ

デフォルトでは、レイヤのサブレイヤはレイヤの境界矩形や角の丸みに合わせて切り取られることはありません。`masksToBounds`プロパティを`true`に設定すると、レイヤコンテンツをその値に切り取ることができます。サンプルレイヤの`maskToBounds`プロパティは`false`です。モニタとテストのパターンを表示しているこのサブレイヤは、一部が親レイヤの境界矩形の外側にあることに注目してください。

次の`CALayer`プロパティがレイヤのサブレイヤの表示に影響します。

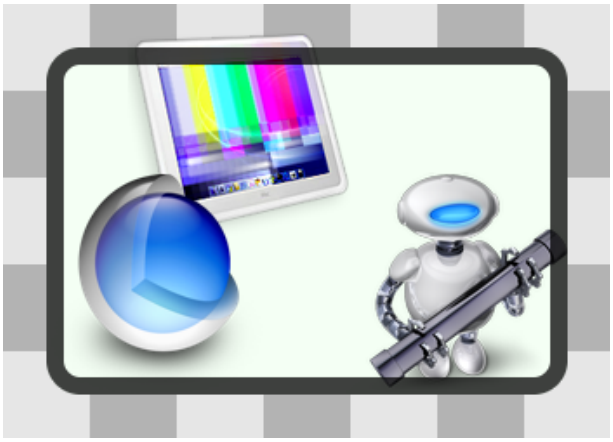
- `sublayers`
- `masksToBounds`
- `sublayerTransform`

# ボーダー属性

レイヤは、指定した色と幅で任意指定のボーダーを表示できます。図 5に、ボーダーを適用した後のサンプルレイヤを示します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 5**　ボーダー属性のあるコンテンツを表示したレイヤ



次のCALayerプロパティがレイヤのボーダーの表示に影響します。

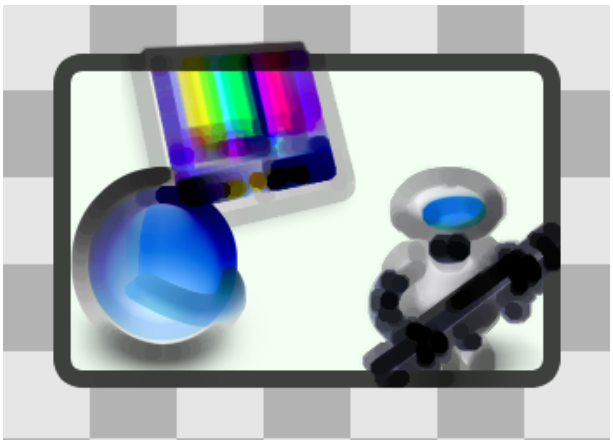
- borderColor
- borderWidth

**iOSにおける注意事項：** borderColorプロパティおよびborderWidthプロパティは、iOS 3.0以降でのみサポートされています。

## フィルタプロパティ

Core Imageフィルタの配列は、レイヤに適用できます。これらのフィルタは、レイヤのボーダー、コンテンツ、および背景に影響します。図6に、Core Imageのポスタライズフィルタを適用したサンプルレイヤを示します。

**図 6**　フィルタプロパティを表示したレイヤ




2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

次のCALayerプロパティがレイヤのコンテンツフィルタを指定します。

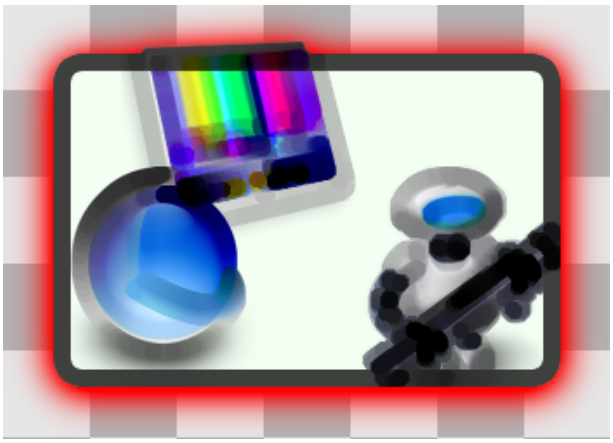
- filters

**iOSにおける注意事項：** iOSのCALayerクラスがfiltersプロパティを公開している間、Core Imageは使用できません。このプロパティで使用可能なフィルタは現在定義されていません。

## シャドウプロパティ

任意で、レイヤに不透明度、色、オフセット、ブラー半径を指定することによってシャドウを表示できます。図 7に、赤いシャドウを適用したサンプルレイヤを示します。

**図 7** シャドウプロパティを表示したレイヤ



次のCALayerプロパティがレイヤのシャドウの表示に影響します。

- shadowColor
- shadowOffset
- shadowOpacity
- shadowRadius

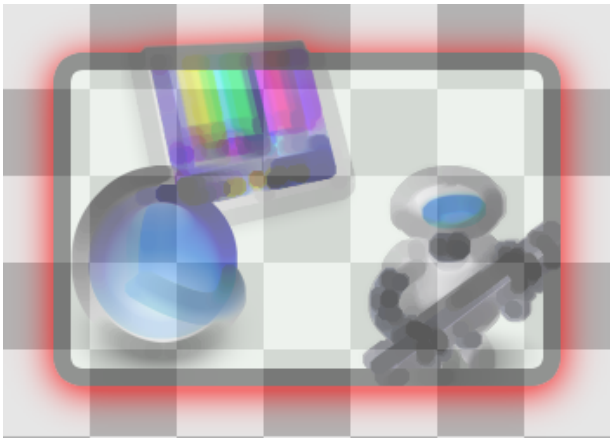

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**iOSにおける注意事項：** `shadowColor`プロパティ、`shadowOffset`プロパティ、`shadowOpacity`プロパティ、および`shadowRadius`プロパティは、iOS 3.2以降でのみサポートされています。

## 不透明度プロパティ

レイヤの不透明度を設定することで、レイヤの透明度を調節できます。図 8に、不透明度が0.5のサンプルレイヤを示します。

**図 8**　不透明度プロパティを含むレイヤ



次の`CALayer`プロパティがレイヤの不透明度を指定します。

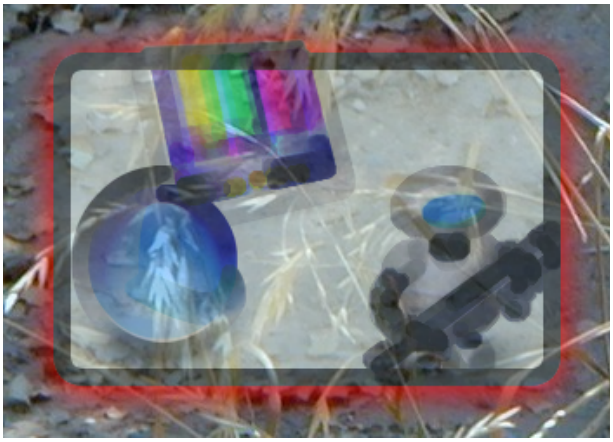
- `opacity`

## 合成プロパティ

レイヤの合成フィルタを使って、レイヤのコンテンツをその背後のレイヤと結合します。デフォルトでは、レイヤはソースオーバーを使って合成されます。図 9に、合成フィルタを適用したサンプルレイヤを示します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 9**　compositingFilterプロパティを使用して合成されたレイヤ



次のCALayerプロパティがレイヤの合成フィルタを指定します。

- compositingFilter

**iOSにおける注意事項：** iOSのCALayerクラスがcompositingFilterプロパティを公開している間、Core Imageは使用できません。このプロパティで使用可能なフィルタは現在定義されていません。

## マスクプロパティ

最後に、マスクとして機能するレイヤを指定して、描画したレイヤの表示方法をさらに変更できます。マスクレイヤの透明度は、レイヤが合成されたときのマスク処理を決定します。図10に、マスクレイヤを使って合成されたサンプルレイヤを示します。

**図 10**　マスクプロパティを使用して合成されたレイヤ

次のCALayerプロパティがレイヤのマスクを指定します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

- `mask`

**iOSにおける注意事項：** `mask`プロパティは、iOS 3.0以降でのみサポートされています。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# サンプル：Core Animation Kiosk Menu Styleのアプリケーション

サンプルのCore Animation Kiosk Style Menuは、シンプルな選択項目を表示します。ユーザインターフェイスの生成とアニメーション化にCore Animationレイヤを使用しています。100行足らずのコードで、次の機能とデザインパターンを示します。

- ビューでのレイヤ階層のルートレイヤのホスティング。
- レイヤの作成とレイヤ階層への挿入。
- `QCCompositionLayer`を使ってQuartz Composerコンポジションをレイヤコンテンツとして表示。無地の色を使用した場合のパフォーマンスの向上も示します。
- 連続的に実行する明示的なアニメーションの使用。
- Core Imageフィルタ入力のアニメーション化。
- 選択項目の位置の暗黙的なアニメーション化。
- ビューをホストするMenuViewインスタンスを通じたキーイベントの処理。

このアプリケーションはCore ImageフィルタおよびQuartz Composerコンポジションを多用します。そのため、結果として、Mac OS X上でのみ実行します。レイヤ階層の管理、暗黙的アニメーションと明示的アニメーション、およびイベント処理について示されている手法は、どちらのプラットフォームにも共通です。

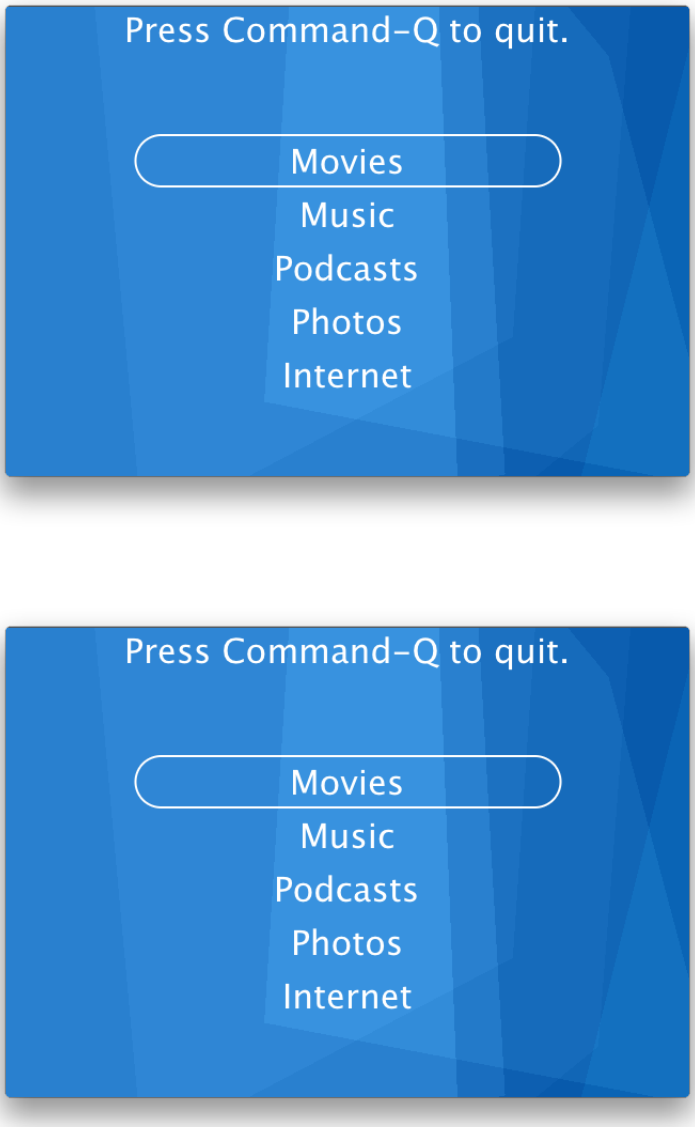
アプリケーションのサンプルコードは、次の2つのバージョンが用意されています。1つは背景に`QCCompositionLayer`を使用するバージョン、もう1つは背景に無地の黒を使用するバージョンです。アプリケーションの作業用コピーについては、*CoreAnimationKioskStyleMenu*サンプルを参照してください。コードの最初のウォークスルー解説ではQuartz Composerの背景を使うバージョンのアプリケーションを説明します。無地の色を使用した際のパフォーマンスの向上については、この章で後述します。

## ユーザインターフェイス

**QCCoreAnimationKioskMenuStyle**サンプルは、ごく基本的なキオスクスタイルのユーザインターフェイスを提供します。ユーザインターフェイスが画面全体を占有し、ユーザはメニューから1つの項目を選択できます。ユーザはキーボードの上下の矢印でメニューを移動します。選択項目が変わると、選択インジケータ（白い角丸矩形）が新しい位置に移動します。選択インジケータには、連続的にアニメーション化されるブルームフィルタが設定されています。これにより、ユーザの注意を引き付けることができます。背景には、人目を引くように継続的に実行するQuartz Composerアニメーションを使用しています。図 1に、このアプリケーションのインターフェイスを示します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 1** Core Animation Kiosk Menuインターフェイス



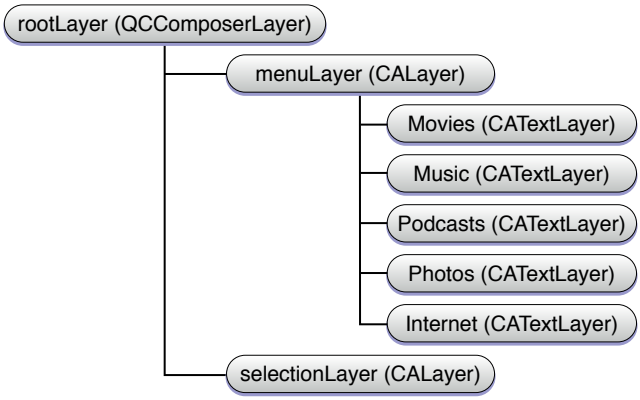

## nibファイルの確認

## レイヤ階層

CoreAnimationKioskMenuStyleアプリケーションのレイヤ階層（レイヤツリーとも呼ばれます）を次に示します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 2**　QCCoreAnimationKioskStyleMenuアプリケーションのレイヤ階層




`rootLayer`は、`QCComposerLayer`のインスタンスです。ルートレイヤであるこのレイヤは、`MenuView`インスタンスと同じサイズであり、ウインドウがサイズ変更されても変わりません。

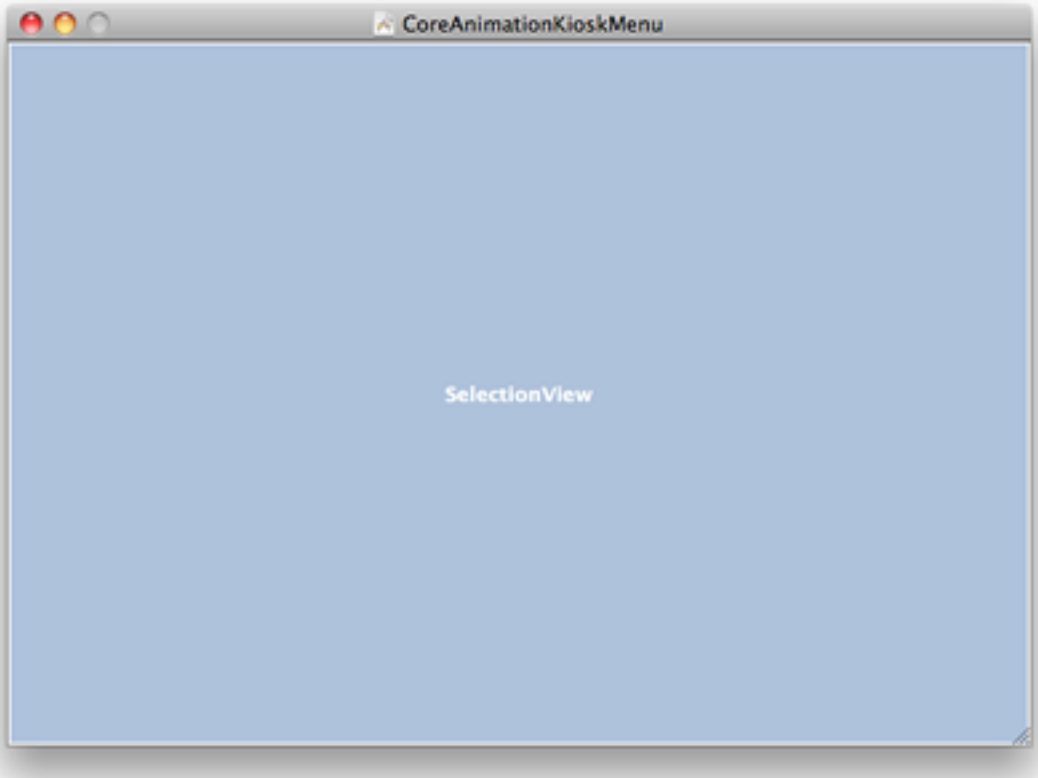
`menuLayer`は、`rootLayer`のサブレイヤです。これは空のレイヤです。このレイヤには、`contents`プロパティとして設定されているものがなく、スタイルプロパティも設定されていません。`menusLayer`は、単にメニュー項目レイヤのコンテナとして使われます。このアプローチにより、アプリケーションは`menusLayers.sublayers`配列内の位置によってメニュー項目のサブレイヤに簡単にアクセスできます。`menusLayer`は`rootLayer`と同じサイズで、`rootLayer`に重なります。これは、現在のメニュー項目を基準にして相対的に`selectionLayer`を位置決めする場合に、座標系間で変換を行う必要をなくすために意図的になされました。

# アプリケーションのnibファイルの確認

`MainMenu.nib`は非常に単純です。`CustomView`のインスタンスは、Interface Builderパレットからドラッグされてウインドウに配置されます。これは、ウインドウ全体を塗りつぶすようにサイズ変更されます。`SelectionView.h`ファイルは、Menu.nibウインドウにドラッグすることによってInterface Builderにインポートされます。ここで`CustomView`はSelection View Identityパレットで選択され、Classが`SelectionView`に変わります。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 3**　MainMenu.nibファイル




ほかのコネクションを作成する必要はありません。nibファイルがロードされるとウインドウが`SelectionView`インスタンスとともに展開されます。`SelectionView`クラスのレイヤが、クラスの`awakeFromNib`実装内で構成されます。

# アプリケーションコードの確認

アプリケーションのnibファイルと全体的なデザインを確認したので、次に`SelectionView`クラスの実装を詳しく見ていきます。`SelectionView.h`ファイルには、`SelectionView`のプロパティとメソッドが宣言されており、SelectionView.mには実装が含まれています。実装ファイル内でコードを簡単に見つけられるように、`#pragma mark`文がそれぞれ対応する部分に追加されています。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

## QCCoreAnimationKioskStyleMenu.hファイルとQCCoreAnimationKioskStyleMenu.mファイル

CodeAnimationKioskMenuAppDelegate.hファイルおよび.mファイルは、アプリケーションが作成された際に自動的に作成されました。これらのファイルには、このサンプルに関連するコードは含まれていません。

## SelectionView.hの確認

SelectionViewクラスはNSViewのサブクラスです。ここでは4つのプロパティとSelectionView.hが実装しているメソッドが宣言されています。アプリケーションのウインドウにおける唯一のビューで、アプリケーションのレイヤツリー内のrootLayerとその他のレイヤのコンテナの役割を果たします。

**リスト 1**　SelectionView.hファイルリスト

```
#import <Cocoa/Cocoa.h>
#import <QuartzCore/Quartz.h>


// SelectionViewクラスは、ウインドウに挿入されるビューの
// サブクラスであり、rootLayerをホストしイベントに応答する
@interface SelectionView : NSView {

    // 選択されたメニュー項目インデックスを含む
    NSInteger selectedIndex;

    // メニュー項目レイヤを含むレイヤ
    CALayer *menuLayer;

    // 選択項目を表示するレイヤ
    CALayer *selectionLayer;

    // メニュー項目名を含むレイヤの配列
    NSArray *names;

}

@property  NSInteger selectedIndex;
@property (retain) CALayer *menuLayer;
@property (retain) CALayer *selectionLayer;
@property (retain) NSArray *names;

-(void)awakeFromNib;
-(void)setupLayers;
-(void)changeSelectedIndex:(NSInteger)theSelectedIndex;
-(void)moveUp:(id)sender;
-(void)moveDown:(id)sender;
-(void)dealloc;

@end
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**注：** Quartz/CoreAnimation.hがインポートされることに注意してください。QuartzCore.frameworkは、Core Animationを使用するすべてのプロジェクトに追加する必要があります。次の例ではQuartz Composerを使用するため、SelectionView.hのヘッダはQuartz/Quartz.hもインポートしています。Quartz.frameworkがプロジェクトに追加されます。

## SelectionView.mの確認

SelectionViewクラスはこのアプリケーションの中心的な役割を果たします。SelectionViewは、ビューがnibによってロードされると応答して、現在の画面いっぱいに広がって、表示すべきレイヤのセットアップ、アニメーションの作成、選択項目を変更するキーの処理を行います。

SelectionView.mファイルリストは、次のように分けられます。

- awakeFromNibの実装
- レイヤのセットアップ
- 選択レイヤの動きのアニメーション化
- キーイベントへの応答
- クリーンアップ

### awakeFromNibの実装

awakeFromNibメソッドは、MainMenu.nibがロードされて展開されると呼び出されます。ビューはawakeFromNibでセットアップを完了することを期待されています。

MainMenuViewのawakeFromNibの実装では次の処理を行います。

- メニュー項目の表示に使用する文字列、名前の配列を作成する。
- カーソルを非表示にする。通常、フルスクリーンアプリケーションはカーソルを表示しないため、アプリケーションでは完全にキーボード入力に頼ることになります。
- ウインドウがサイズ変更した場合に、ビューは自動的に画面いっぱいになるようにサイズ変更する。こうすることでキオスクの外観になります。
- ウインドウで、firstResponder内にあるビューが上下の矢印イベントを受信するように設定する。
- レイヤを設定するsetupLayersメソッドを呼び出す（「setupLayersメソッド」（75ページ）で説明します）。
- 最後に、ウインドウを前面に移動し、表示する。

リスト2に抜粋したコードに、上記の機能を行うコードを示します。これは、サンプルプロジェクトのSelectionView.mの#pragma mark "Implementation of awakeFromNib"にあります。

**リスト2**　awakeFromNibの実装

```
#pragma mark - Listing 1: Implementation of awakeFromNib
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
- (void)awakeFromNib
{

    // 各種文字列を含む配列を
    // 作成する
    self.names=[NSArray arrayWithObjects:@"Movies",@"Music",
                @"Podcasts",@"Photos",@"Internet",
                nil];

    // 選択項目ではカーソルを使用しないため非表示にする
    [NSCursor hide];

    // キオスクアプリケーションのようにフルスクリーンにする
    [self enterFullScreenMode:[self.window screen] withOptions:NULL];

    // ウインドウでファーストレスポンダがキーストロークを取得するようにする
    [self.window makeFirstResponder:self];

    // 個別のレイヤを設定する
    [self setupLayers];

    // ウインドウを前面に移す
    [self.window makeKeyAndOrderFront:self];

}
```

## setupLayersメソッド

QCCoreAnimationKioskStyleMenuアプリケーションのコードの大半は、`setupLayers`メソッドにあります。このメソッドは、レイヤとアニメーションの作成と、現在選択されている項目の構成を行います。

リスト 3 （75 ページ）に`#pragma mark "Configuration of the Background rootLayer"`を示します。このコード例では、子レイヤすべての`rootLayer`を設定しています。

コードのこの部分には、次のような注目すべき重要な要素が2つあります。

- `rootLayer`は、作成されてビュー用に設定されて初めて、そのビューでレイヤを利用できるようになります。この順番で行うと、ビューは独自のレイヤを作成せずに、指定したレイヤを使用することができます。**レイヤビューホスティング**と言います。「レイヤビューホスティング」とする。レイヤビューホスティングでは、Core Animationがすべての描画を担う必要があります。`NSView`の描画機能やメソッドを使用することはできませんが、ビューのイベント処理ルーチンは通常通り使用できます。
- もう1つの要素はパフォーマンスです。ハードウェアによっては、バックグラウンドでのQuartz Composerアニメーションの実行はパフォーマンス問題を引き起こす可能性があります。無地の色を使うとパフォーマンスが向上します。これについては、「パフォーマンスの検討事項」 （79 ページ）で説明します。

**リスト 3**　　バックグラウンドrootLayerの構成

```
#pragma mark Listing: "Configuration of the Background rootLayer"

-(void)setupLayers;
{
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
    // Quartz Compositionレイヤを作成する
    // 注記：QCCompositionを実行するとパフォーマンスに大きく影響する可能性がある
    QCCompositionLayer* rootLayer=[QCCompositionLayer compositionLayerWithFile:
                [[NSBundle mainBundle] pathForResource:@"Background"
ofType:@"qtz"]];

    // QCCompositionLayerをルートレイヤとして設定し、
    // その後wantsLayerをオンにする。この順番で行うと
    // ビュー側でレイヤをホストする動作になる。
    [self setLayer:rootLayer];
    [self setWantsLayer:YES];
```

リスト 4（76 ページ）に#pragma mark Setup menuLayers Array. The Selectable Menu Items.を示します。このコードはアプリケーションのmenuLayerを、rootLayerのサブレイヤとして挿入して初期化しました（アプリケーションのレイヤ階層については、図 2（71 ページ）を参照してください）。その後、それぞれのテキストエントリ用に別々のレイヤをmenuLayerに挿入しています。このコードはサンプルコードの#pragma mark - Setup menuLayers Array. The Selectable Menu Items.にあります。

**リスト 4**　menuLayers Arrayの設定。選択メニュー項目。

```
    // メニューを含めるレイヤを作成する
    self.menuLayer=[CALayer layer] ;
    self.menuLayer.frame=rootLayer.frame;
    self.menuLayer.layoutManager=[CAConstraintLayoutManager layoutManager];
    [rootLayer addSublayer:self.menuLayer];

    // 個別に選択可能な項目のサイズと位置を設定し、計算する
    CGFloat width=400.0;
    CGFloat height=50.0;
    CGFloat spacing=20.0;
    CGFloat fontSize=32.0;
    CGFloat initialOffset=self.bounds.size.height/2-(height*5+spacing*4)/2.0;

    // テキストの描画、およびselectionLayerでも使用するwhiteColorを作成する
    CGColorRef whiteColor=CGColorCreateGenericRGB(1.0f,1.0f,1.0f,1.0f);


    // 選択項目の名前のリストを反復処理して、それぞれにレイヤを作成する
    // menuItemLayerもこのループの実行中に位置決めされる
    NSInteger i;
    for (i=0;i<[names count];i++) {

        CATextLayer *menuItemLayer=[CATextLayer layer];
        menuItemLayer.string=[self.names objectAtIndex:i];
        menuItemLayer.font=@"Lucida-Grande";
        menuItemLayer.fontSize=fontSize;
        menuItemLayer.foregroundColor=whiteColor;
        [menuItemLayer addConstraint:[CAConstraint
                                      constraintWithAttribute:kCAConstraintMaxY
                                       relativeTo:@"superlayer"
                                       attribute:kCAConstraintMaxY
                                    offset:-(i*height+spacing+initialOffset)]];
        [menuItemLayer addConstraint:[CAConstraint
                                      constraintWithAttribute:kCAConstraintMidX
                                       relativeTo:@"superlayer"
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
                                        attribute:kCAConstraintMidX]];
        [self.menuLayer addSublayer:menuItemLayer];
    } // ループの最後
    [self.menuLayer layoutIfNeeded];
```

リスト5（77ページ）に#pragma mark Setup selectionLayer. Used to Display the Currently Selected Item.を示します。コードにはコメントが付いていますが、要約すると次のようになります。

- selectionLayer CALayerを作成する。
- CIBloomフィルタをレイヤに追加し、選択インジケータに振動アニメーションを継続的に提供するよう設定する。
- selectionIndexに、初期値0を設定する。
- selectionLayerをサブレイヤとしてrootLayerレイヤに追加する。

実装については、サンプルコードの#pragma mark - Setup selectionLayer. Displays the Currently Selected Item.を参照してください。

**リスト5**　selectionLayerの設定。現在の選択項目の表示に使用。

```
#pragma mark - Setup selectionLayer. Displays the Currently Selected Item.

    // 追加のレイヤ、selectionLayerを使用して、
    // 現在の項目が選択されていることを示す
    self.selectionLayer=[CALayer layer];
    self.selectionLayer.bounds=CGRectMake(0.0,0.0,width,height);
    self.selectionLayer.borderWidth=2.0;
    self.selectionLayer.cornerRadius=25;
    self.selectionLayer.borderColor=whiteColor;
    CIFilter *filter = [CIFilter filterWithName:@"CIBloom"];
    [filter setDefaults];
    [filter setValue:[NSNumber numberWithFloat:5.0] forKey:@"inputRadius"];
    [filter setName:@"pulseFilter"];
    [selectionLayer setFilters:[NSArray arrayWithObject:filter]];

    // selectionLayerは、表示されているような微妙な振動を
    // 示す。コードのこの部分では振動アニメーションを作成し、
    // filters.pulsefilter.inputintensityに0から2の範囲を設定する。
    // これにより、1秒に1回オートリバースが行われ、無限に繰り返される。
    CABasicAnimation* pulseAnimation = [CABasicAnimation animation];
    pulseAnimation.keyPath = @"filters.pulseFilter.inputIntensity";
    pulseAnimation.fromValue = [NSNumber numberWithFloat:0.0];
    pulseAnimation.toValue = [NSNumber numberWithFloat:2.0];
    pulseAnimation.duration = 1.0;
    pulseAnimation.repeatCount = 1e100f;
    pulseAnimation.autoreverses = YES;
    pulseAnimation.timingFunction = [CAMediaTimingFunction functionWithName:
                                     kCAMediaTimingFunctionEaseInEaseOut];
    [selectionLayer addAnimation:pulseAnimation forKey:@"pulseAnimation"];

    // 選択されたとおりに最初の項目を設定する
    [self changeSelectedIndex:0];

    // 最後に、選択レイヤをルートレイヤに追加する
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
    [rootLayer addSublayer:self.selectionLayer];

    // クリーンアップする
    CGColorRelease(whiteColor);
    // setupLayersの最後
}
```

## キーイベントへの応答

レイヤはレスポンダチェーンに関与しない（つまりイベントを受け付けない）ため、**レイヤホスト**として機能するSelectionViewビューインスタンスがその役割を引き継ぐ必要があります。SelectionViewのビューインスタンスがawakeFromNibメソッドにファーストレスポンダとして登録されているからです。

moveUp:メッセージおよびmoveDown:メッセージは、すべてのビュークラスと同様に、SelectionViewがその子孫となるNSResponderで提供されます。moveUp:およびmoveDown:メッセージは、それぞれ上矢印と下矢印が押されると呼び出されます。これらのメソッドを使用すると、アプリケーションは、ユーザによって指定され、再マップされた機能上の任意の矢印キーに従います。そして、デベロッパ自身のアプリケーションではもっと複雑なキー処理をする必要があるでしょうが、これはkeyDown:を実装するよりも簡単です。

リスト 6 （78 ページ）に示すメソッド実装では、現在の選択レイヤの設定を実装しています。これは、サンプルコードの#pragma mark "Handle Changes in the Selection"にあります。

**リスト 6**　　選択時の変更処理

```
#pragma mark Handle Changes in the Selection

-(void)changeSelectedIndex:(NSInteger)theSelectedIndex
{
    self.selectedIndex=theSelectedIndex;

    if (self.selectedIndex == [names count])
        self.selectedIndex=[names count]-1;
    if (self.selectedIndex < 0)
        self.selectedIndex=0;

    CALayer *theSelectedLayer=[[self.menuLayer sublayers]
objectAtIndex:self.selectedIndex];

    // 現在の選択項目を示すためにselectionLayerを
    // 移動する。これは、トランジションを
    // 表示するためにアニメーションを使用して
    // 実現する。
    self.selectionLayer.position=theSelectedLayer.position;
};
```

上矢印が押下されると、changeSelectedIndex:の呼び出しによってselectedIndexが1つ減らされ、更新されます。リスト 7 （78 ページ）に示す通り、これは正しい項目をハイライトするために選択項目を移動します。サンプルコードでは、#pragma mark Handle Keystrokesにあります。

**リスト 7**　　上下矢印キーの押下処理

```
#pragma mark Handle Keystrokes
```


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

```
-(void)moveUp:(id)sender
{
    [self changeSelectedIndex:self.selectedIndex-1];
}

-(void)moveDown:(id)sender
{
    [self changeSelectedIndex:self.selectedIndex+1];

}
```

SelectionViewを閉じたら、インスタンス変数をクリーンアップする必要があります。deallocの実装で、menuLayer、selectionLayer、およびnamesにnilを設定（つまり解放）します。#pragma mark Dealloc and Cleanupにサンプルコードの該当部分を示します。

**リスト 8**　DeallocとCleanup

```
#pragma mark Dealloc and Cleanup


-(void)dealloc
{
    [self setLayer:nil];
    self.menuLayer=nil;
    self.selectionLayer=nil;
    self.names=nil;
    [super dealloc];
}
```

# パフォーマンスの検討事項

このタイプのインターフェイスの背景にQuartz Composerアニメーションを設定することで、興味深く目を引く表示にすることができます。しかし、ターゲットのハードウェア構成で必ずパフォーマンステストを行う必要があります。選択インジケータをQuartz Composerアニメーションの上層で実行すると、必ずしもすべてのハードウェア構成でスムーズなアニメーションを維持できるわけではありません。背景には無地の色や静止画像を使うことを検討すると良いでしょう。

パフォーマンスの影響を評価しやすいように、サンプルコードプロジェクトには、Quartz Composerアニメーションではなく無地の背景上にコンテンツを描画する追加のプロジェクト**CoreAnimationKioskStyleMenu**が含まれています。潜在的な影響の例として、2つのアプリケーションのパフォーマンスを比較すると参考になります。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

**図 4**　黒い背景を使ったもう1つのインターフェイス




2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# アニメーション化可能プロパティ

CALayerおよびCIFilterの多くのプロパティはアニメーション化できます。この章ではこれらのプロパティを、デフォルトで使用されるアニメーションとともに一覧に示します。

## CALayerのアニメーション化可能プロパティ

次のCALayerのクラスプロパティは、Core Animationでアニメーション化できます。詳細については、CALayerを参照してください。

- anchorPoint

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- backgroundColor

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します（サブプロパティは基本アニメーションを使ってアニメーション化されます）。

- backgroundFilters

  表 2 （83 ページ）で説明するデフォルトの暗黙的なCATransitionAnimationを使用します。フィルタのサブプロパティは、表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用してアニメーション化されます。

- borderColor

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- borderWidth

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- bounds

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- compositingFilter

  表 2 （83 ページ）で説明するデフォルトの暗黙的なCATransitionAnimationを使用します。フィルタのサブプロパティは、表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用してアニメーション化されます。

- contents

- contentsRect


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- cornerRadius

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- doubleSided

  デフォルトの暗黙的なアニメーションは設定されていません。

- filters

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。フィルタのサブプロパティは、表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用してアニメーション化されます。

- frame

  frameプロパティ自体はアニメーション化されません。代わりに、boundsとpositionを変更して同じ結果をアーカイブできます。

- hidden

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- mask

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- masksToBounds

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- opacity

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- position

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- shadowColor

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- shadowOffset

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- shadowOpacity

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。

- shadowRadius

  表 1 （83 ページ）で説明するデフォルトの暗黙的なCABasicAnimationを使用します。


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

- `sublayers`

  表 2 （83 ページ）で説明するデフォルトの暗黙的な`CATransitionAnimation`を使用します。

- `sublayerTransform`

  表 1 （83 ページ）で説明するデフォルトの暗黙的な`CABasicAnimation`を使用します。

- `transform`

  表 1 （83 ページ）で説明するデフォルトの暗黙的な`CABasicAnimation`を使用します。

- `zPosition`

  表 1 （83 ページ）で説明するデフォルトの暗黙的な`CABasicAnimation`を使用します。

**表 1** デフォルトの暗黙的な基本アニメーション

| 説明 | 値 |
| --- | --- |
| クラス | CABasicAnimation |
| `duration` | 0.25秒または現在のトランザクションの所要時間 |
| keyPath | レイヤのプロパティタイプに依存 |

**表 2** デフォルトの暗黙的なトランジション

| 説明 | 値 |
| --- | --- |
| クラス | CATransition |
| duration | 0.25秒または現在のトランザクションの所要時間 |
| type | Fade (`kCATransitionFade`) |
| startProgress | 0.0 |
| endProgress | 1.0 |

# CIFilterのアニメーション化可能プロパティ

Core Animationは次のアニメーション化可能プロパティをCore Imageの`CIFilter`クラスに追加しています。詳細については、`CIFilter Core Animation Additions`を参照してください。これらのプロパティは、Mac OS Xでのみ使用できます。

- `name`
- `enabled`


2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

# 書類の改訂履歴

この表は「*Core Animation プログラミング言語*」の改訂履歴です。

| 日付 | メモ |
| --- | --- |
| 2010-09-24 | iOS 4.2のCore Animationサポートを反映するように文書全体を更新しました。 |
| 2010-08-12 | iOSの原情報を修正しました。Mac OS Xモデルをベースにした例で使用している座標系の原点について、分かりやすい説明にしました。 |
| 2010-05-25 | 自動サイズ変更マスクの表を修正しました。 |
| 2010-03-24 | 「レイヤコンテンツの指定」におけるcontentGravityプロパティのサイズ変更の表に不足していた定数を追加しました。 |
| 2010-02-24 | Core Animation Kiosk Style Menuのチュートリアルプロジェクトを更新しました。 |
| 2010-01-20 | repeatCountの無限値を更新しました。 |
| 2009-10-19 | セクションヘッダを修正しました。 |
| 2009-08-13 | iOS v3.0以降のcornerRadiusの機能を修正しました。 |
| 2008-11-13 | iOS SDKコンテンツをMac OS Xコンテンツに導入しました。フレームアニメーションの機能について修正しました。 |
| 2008-09-09 | 誤植を修正しました。 |
| 2008-06-18 | iOS用に更新しました。 |
| 2008-05-06 | 誤植を修正しました。 |
| 2008-03-11 | 誤植を修正しました。 |
| 2008-02-08 | 誤植を修正しました。RadiansToDegrees()の計算を修正しました。 |
| 2007-12-11 | 誤植を修正しました。 |
| 2007-10-31 | プレゼンテーションツリーに関する情報を追加しました。アプリケーションウォークスルーの例を追加しました。 |
|  | Core Animationの主要なコンポーネントとサービスを紹介する新規文書。 |
|  | 「Key-Value Coding Additions」の章が追加されました。 |
|  | 新しいCore Animation APIプレフィックスを反映するようにクラス名を更新しました。 |



2010-09-24 | © 2010 Apple Inc. All Rights Reserved.

2010-09-24 | © 2010 Apple Inc. All Rights Reserved.