iOS开发基础教程

动画

大纲

- 动画

- Core Animation基础
- 建立Layer对象
- 基于Layer的动画
- 参考阅读

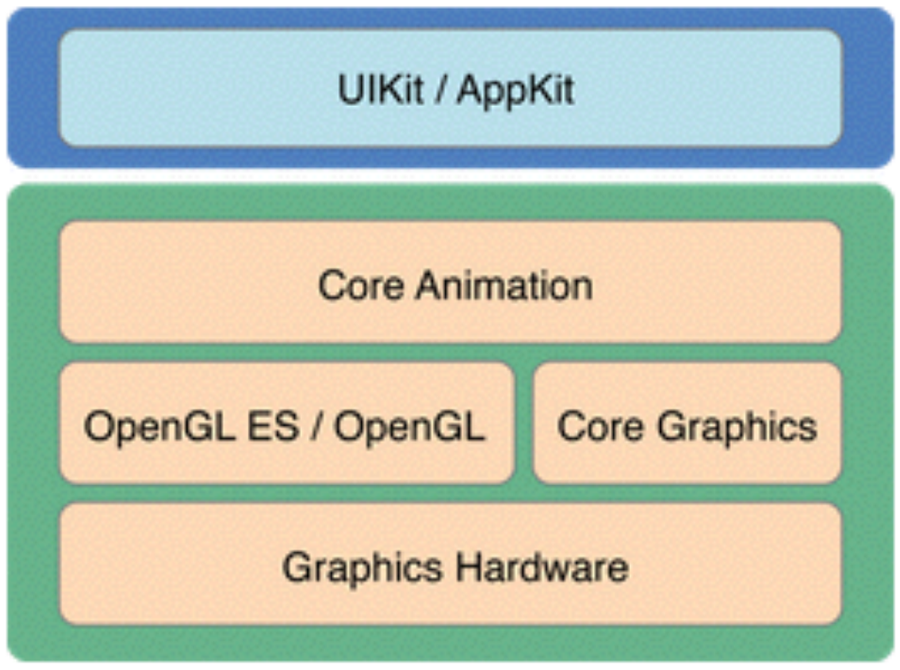
Core Animation

- Core Animation是iOS与OS X平台上负责图形渲染与动画的基础设施
- Core Animation可以动画视图和其他的可视元素
- Core Animation为你完成了实现动画所需的大部分绘帧工作，你只需在配置少量的动画参数（如开始点位置和结束点位置）就可启动Core Animation
- Core Animation将大部分实际的绘图任务交给了图形硬件处理，图形硬件会加速图形渲染的速度。这种自动化的图形加速让动画具有更高的帧率且更加平滑，但这并不会增加CPU的负担而导致影响你应用的运行速度。

Core Animation

- Core Animation位于AppKit和UIKit的底层。它被紧密的集成到了Cocoa和Cocoa Touch视图工作流中。

- Core Animation也存在扩展功能的接口，这些接口暴露给了应用的视图。使用这些接口能让你更细粒度地控制应用中的动画。



Core Animation

- Core Animation管理着你的应用内容
- Core Animation自身并不是一个绘图系统。它只是一个负责在硬件上合成和操纵应用内容的基础构件
- Core Animation的核心是图层对象，图层对象用于管理和操控你的应用内容
- 图层将捕获的内容放到一副位图中，图形硬件能够非常容易的操控你的位图
- 在大部分应用中，图层被作为一种管理视图内容的方式，但是你也可以创建标准的图层，这取决于你自身的需要。

Core Animation

- 更改图层属性会产生动画
- 你使用Core Animation创建的大部分动画都包含对图层属性的更改
- 像视图一样，图层对象也具有边框矩形、坐标原点、尺寸、不透明度、变换矩阵以及许多其他面向可视的属性（如backgroundColor）
- 大部分这些属性的值发生了变化都将会触发隐式动画被创建。隐式动画是一种从旧属性值动画到新属性值的动画形式。如果需要全面掌控动画行为，你可以考虑使用显式动画这些属性。

Core Animation

- 有组织的图层层级
- 有组织的图层之间存在着父子关系
- 图层的组织会影响到图层的可见内容。可见内容的管理与视图的管理方法相似
- 附到视图上的图层集合的层级镜像对应的视图层级
- 你可以将独立图层添加到图层的层级以扩充你的视图的可视内容

Core Animation

- 使用动作对象改变图层的默认行为
- 通过动作对象可以做到隐式的图层动画。动作对象是实现了一个预定义接口的常规对象
- Core Animation使用动作对象实现与图层关联的常规的默认动画集合
- 你可以创建属于你自己的动作对象，以实现自定义的动画或者实现其他行为类型，然后你将动作对象赋值给图层的某一属性
- 当属性发生变化，Core Animation检索你的动作对象并告诉动作对象执行对应的动作

大纲

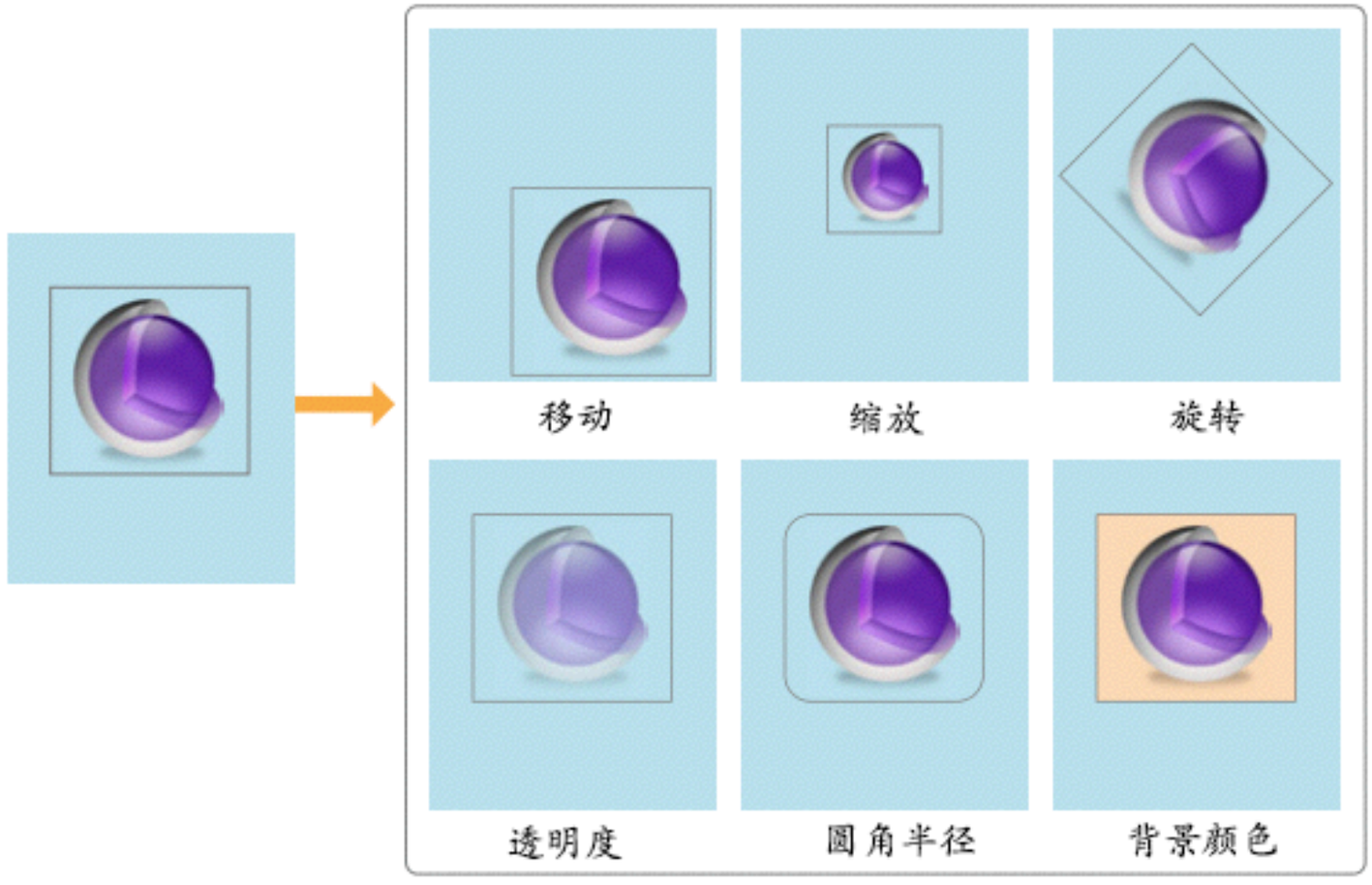
- 动画
- Core Animation基础
- 建立Layer对象
- 基于Layer的动画
- 参考阅读

Core Animation基础

- 图层是绘图与动画的基础
- 图层对象是组织在三维空间的二维平面。它是使用Core Animation执行任何操作的核心构件
- 图层的可管理信息包括几何结构、内容、可视属性
- 不同于视图，图层没有定义它自己的外观，图层仅管理周围位图的状态信息。位图可以是视图的绘图结果或者一张图片

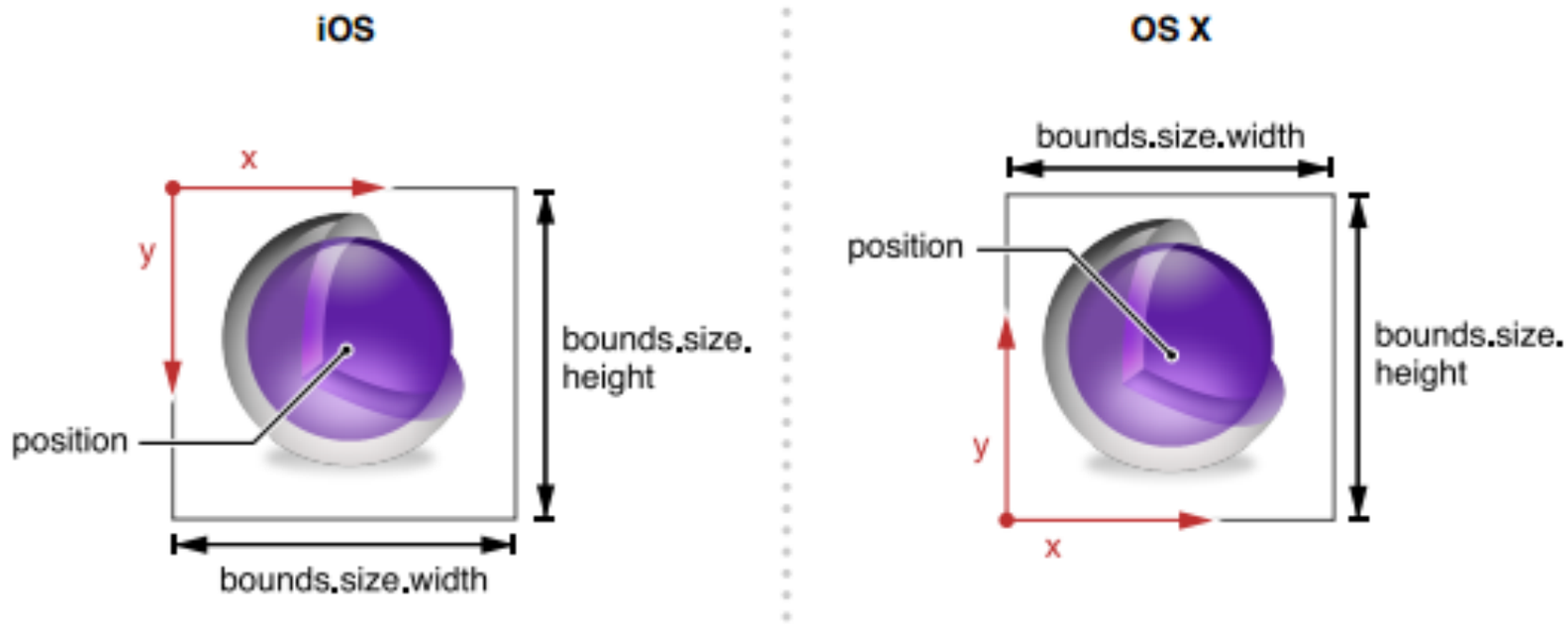
Core Animation基础

- 基于图层的动画
- 图层的数据和状态信息是从图层内容的可视呈现中被分离了出来的
- 这种分离性给了Core Animation介入以及从旧的属性值动画到新的属性值的机会
- 例如改变一个图层的position属性会引起Core Animation将图层从当前的位置移动到新的具体位置



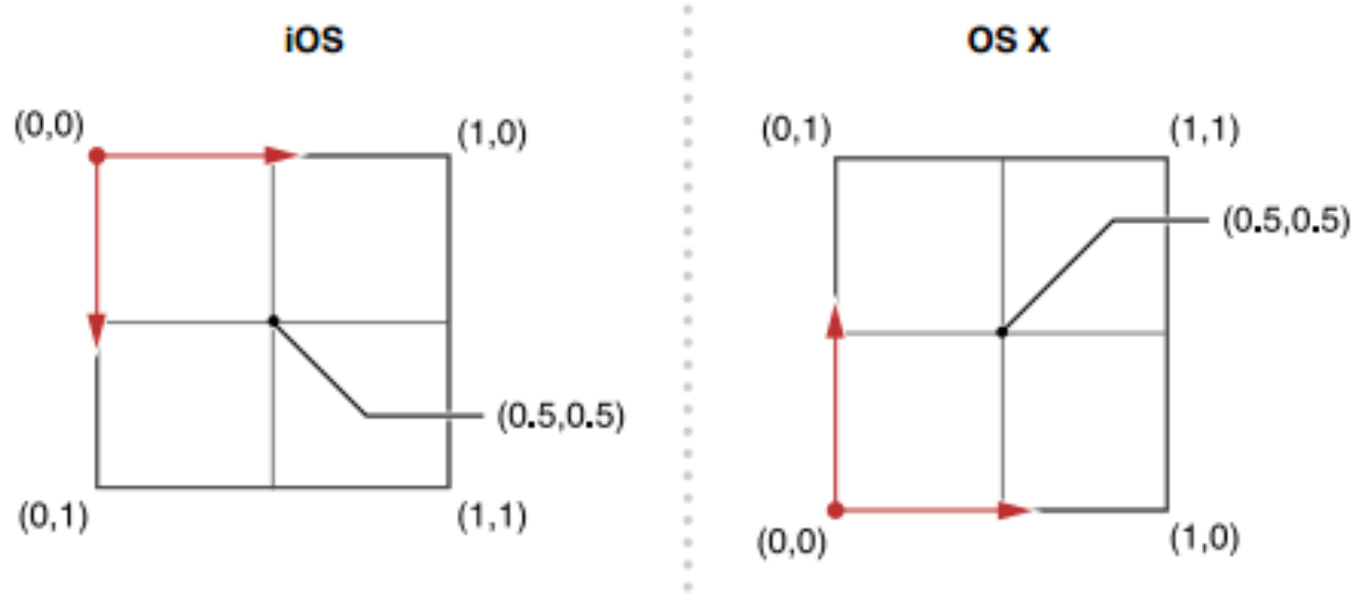
Core Animation基础

- 图层使用两种类型的坐标系统
- 一个图层有一个frame和bound属性，你可以使用这两个属性来定位图层和它的内容
- 图层也有一些视图所没有的属性，比如anchor属性，任何变换操作都是围绕该点运转的（可以理解为一个按在图层上的图钉）
- bound定义了图层自身的坐标系统并包含图层在屏幕上的尺寸。position属性定义了图层相对于父坐标系统的位置
- 在iOS中，默认情况下边界矩形的原点在图层的左上角。而OS X上边界矩形的原点则在左下角。



Core Animation基础

- 锚点是使用单位坐标系统的属性之一
- Core Animation中使用单位坐标表示的属性值可能会因为图层的尺寸变化而发生改变
- 你可以把单位坐标当做是总数的百分比。在单位坐标系统空间中每一个坐标的取值范围在0.0和1.0之间

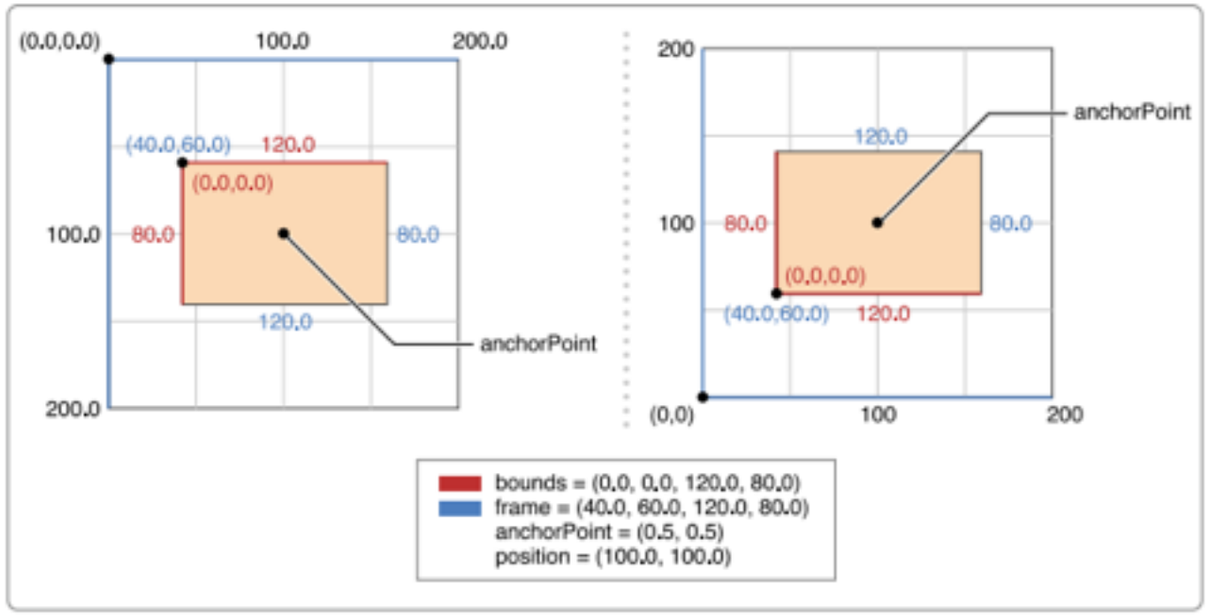


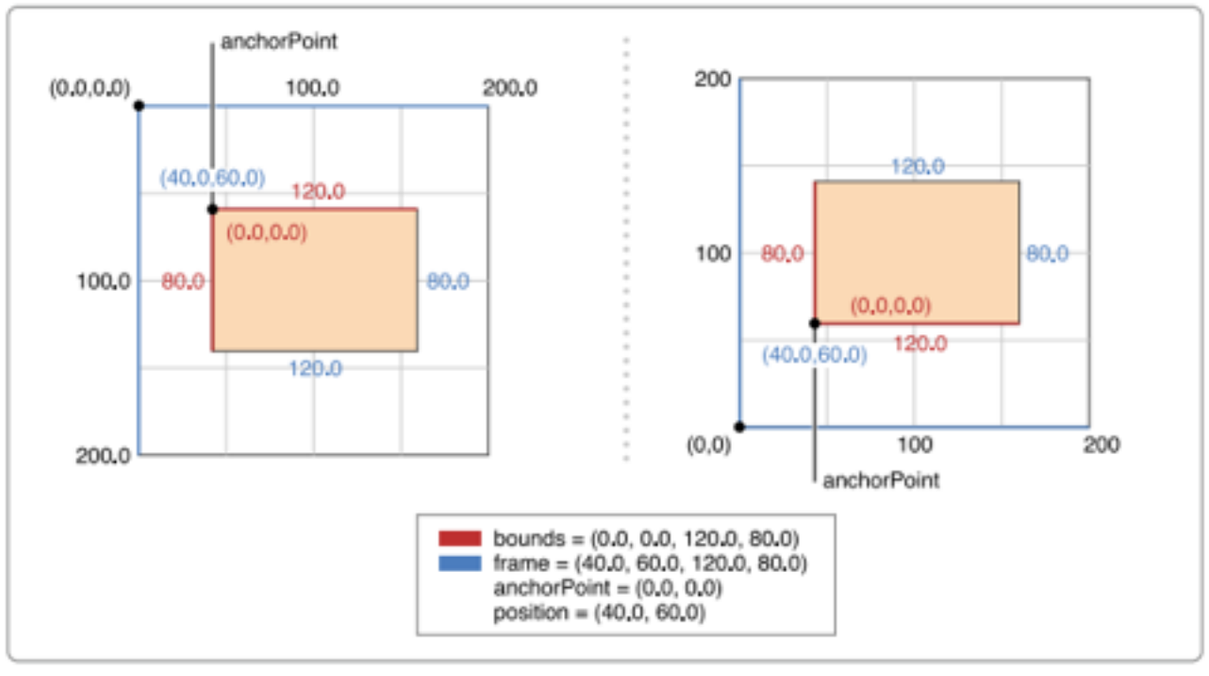
Core Animation基础

- 锚点是使用单位坐标系统的属性之一

- 图层的几何操是相对于图层的锚点进行的，anchorPoint属性可以访问图层的锚点值

- 当改变图层的postion和transform属性值，锚点的影响就很明显。position属性是相对于图层的锚点被指定。并且任何你对图层阴影的变换操作也是相对于锚点。



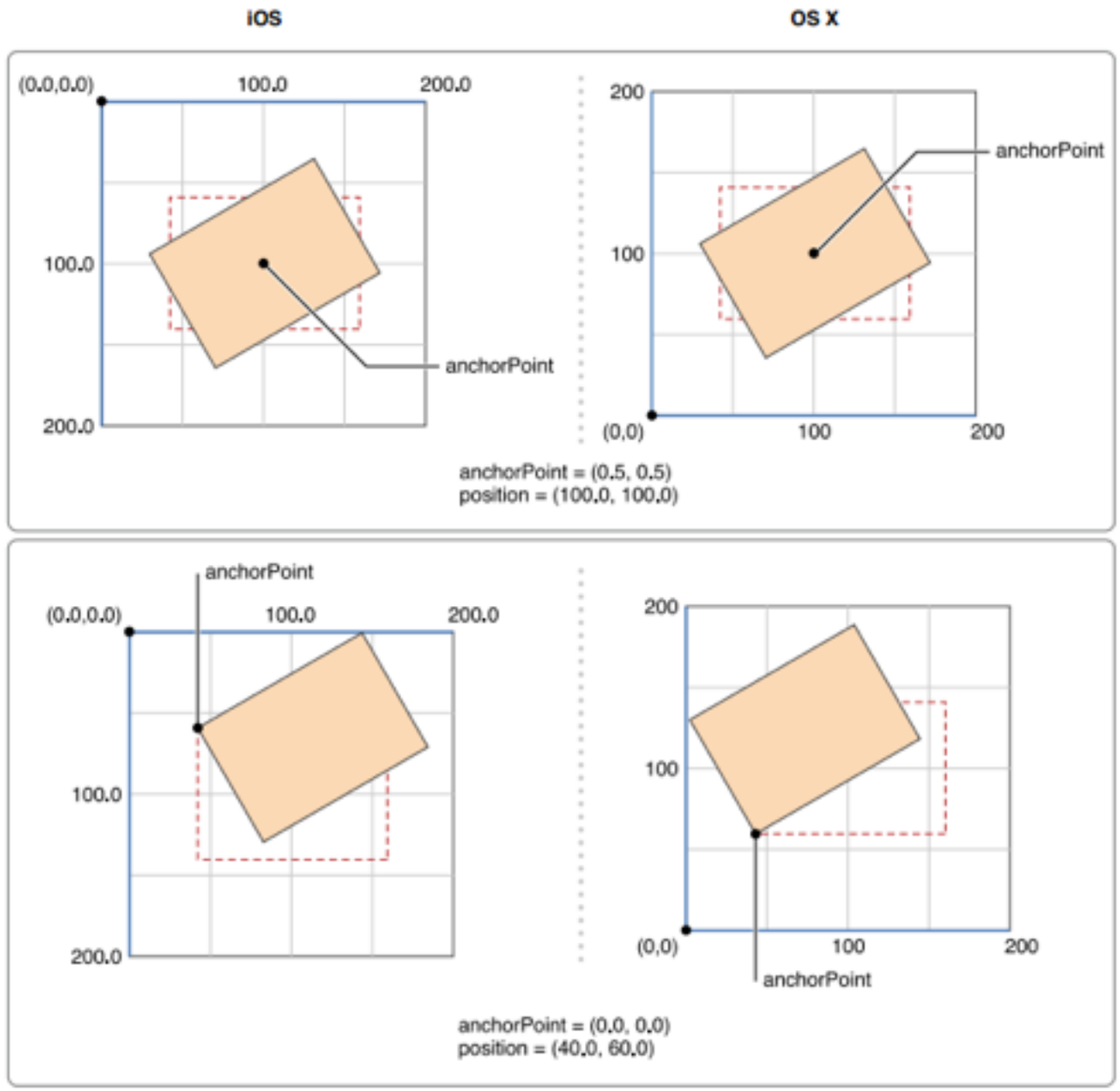


Core Animation基础

- 锚点是使用单位坐标系统的属性之一

- 图层的几何操是相对于图层的锚点进行的，anchorPoint属性可以访问图层的锚点值

- 当改变图层的postion和transform属性值，锚点的影响就很明显。position属性是相对于图层的锚点被指定。并且任何你对图层阴影的变换操作也是相对于锚点。

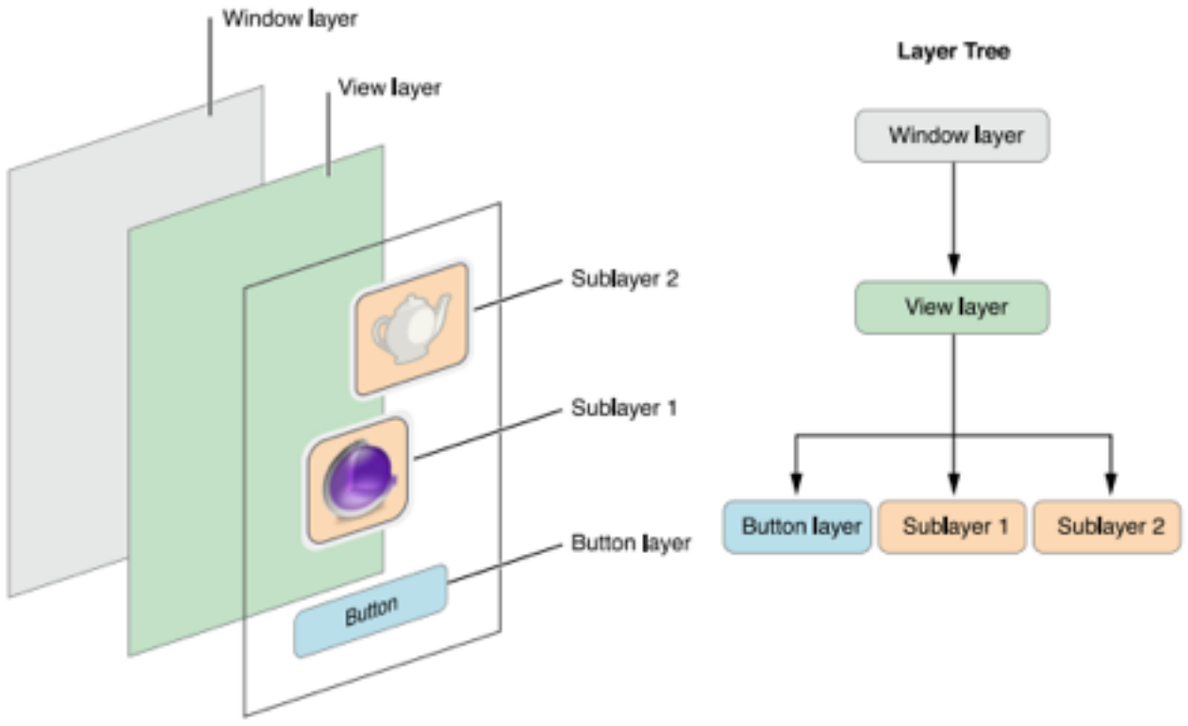


Core Animation基础

- 不同的图层树反映了不同的动画状态
- 使用Core Animation的app拥有三个图层对象集合。每一个图层对象集合在呈现app内容上都扮演着不同的角色。
- 模型图层树中的对象（或简称“图层树”）用的最多。在这个树中的对象是模型对象，模型对象负责存储所有动画的目标值。无论何时改变图层的属性值，你使用的始终是某一个模型对象。
- 呈现树中的对象包含所有运行中的动画的瞬时值。图层树对象包含的是动画的目标值，而呈现树中的对象代表显示在屏幕上动画的当前值。你不应该更改这个树中的对象。相反，你使用这些对象来读取当前动画的值，可能用于创建开始于这些值的新的动画。
- 在渲染树中的对象执行实际的动画，并且对Core Animation是不公开的。

Core Animation基础

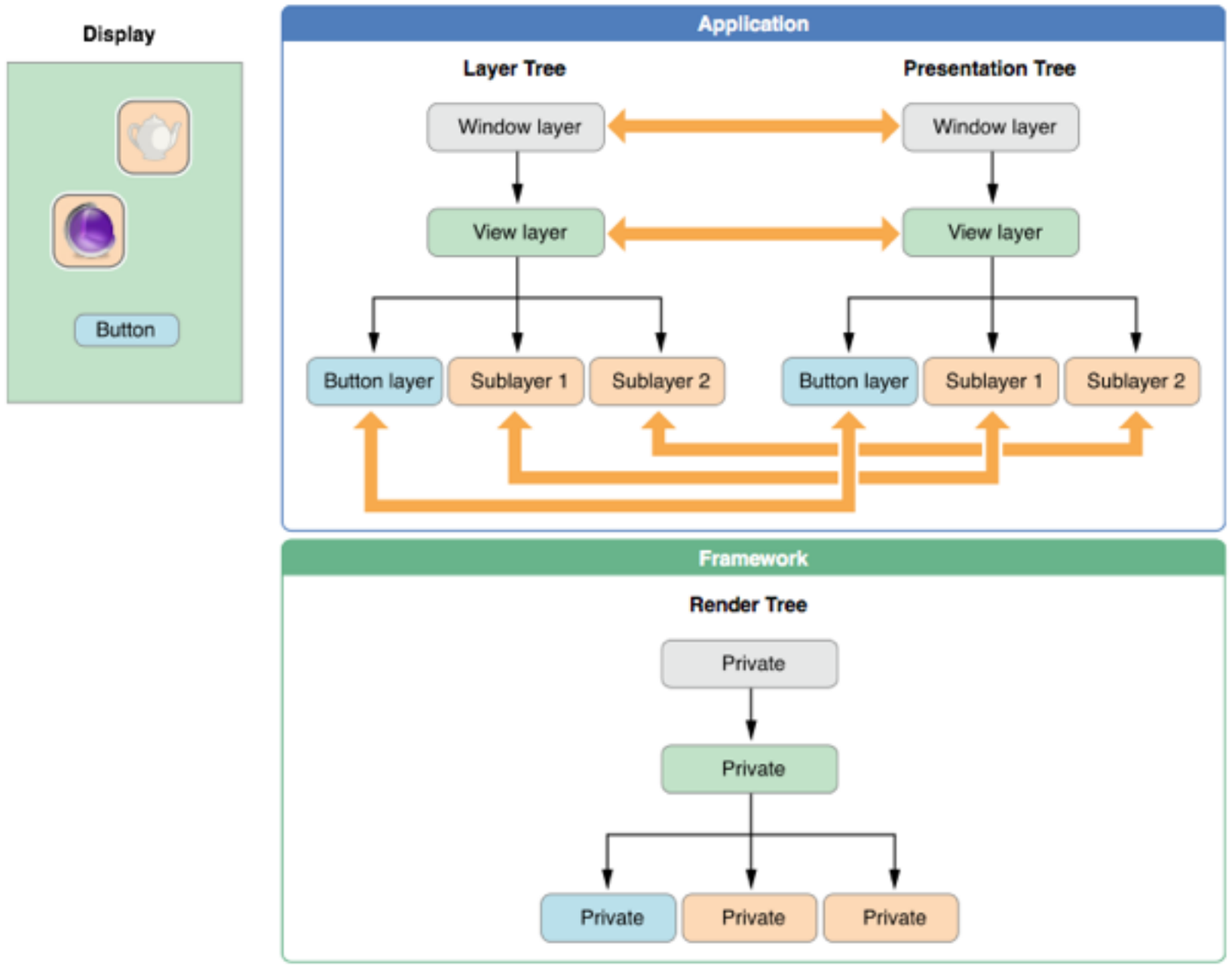
- 每一个图层对象集合被组织在一个层次结构中，类似于app中的视图
- 事实上，在一个所有视图都支持图层功能的app来说，每一个树的初始结构完全与图层的层次相匹配
- 一个app可以添加另外的图层对象，因此图层与视图是不相关联的。可以按照需求将图层对象插入到指定的视图层级中
- 可能为了优化app内容的性能而选择加入图层对象而非视图，原因在于图层的开销要比视图低。



Core Animation基础

- 在图层树中的每一个对象，在渲染树和呈现树中也存在一个与之匹配的对象，app主要与图层树中的对象进行交互，但可能有时会访问呈现树中的对象

- 访问图层树中对象的presentationLayer属性将返回一个在呈现树中相对应的对象。你可能会通过该对象获取在动画执行过程中的某一时刻的属性值。



大纲

- 动画
- Core Animation基础
- 建立Layer对象
- 基于Layer的动画
- 参考阅读

建立图层对象

- 图层是Core Animation的核心

- 图层管理着应用程序的可视内容，图层提供了更改内容样式与可视外观的选项。iOS是自动支持图层的，而如果你是一名OS X开发者，那你必须手动启用图层支持

- 支持图层的视图默认会创建一个CALayer类型的图层实例，一般情况下你可能不需要选择其他类型的图层实例。但是Core Animation根据不同的应用场景提供了一些不同类型的图层类，每个图层类型都拥有特殊的功能。选择适当的图层类可能会提升app的性能或者能以简单的方式支持指定的内容类型。比如CATiledLayer类在显示大图片上将更加高效。

建立图层对象

- 你可以通过覆盖iOS视图中的layerClass方法并返回一个需要的图层类对象。大部分的iOS视图通过创建一个CALayer对象，使用该对象储备视图的内容

- 使用默认的图层类型是个不错的选择，但在某些情况下，你可能需要使用特定特性的图层对象。比如在下述情况下你需要改变图层的类型：
    - 视图的绘图内容是由OpenGL ES实现，此种情况你需要使用CAEAGLLayer对象。
    - 特殊的图层让你拥有更强的表现性能。
    - 需要利用某些特殊的Core Animation类，比如粒子发射器或者拷贝器。
- 改变一个视图的图层类型非常的简单；所有你需要做得只是覆盖layerClass方法并返回一个你想要替代的类对象
- 视图调用layerClass方法并使用返回的类为其创建新的图层对象。一旦创建完成，视图的图层对象将不可改变。

创建图层对象

- Core Animation定义了许多标准的图层类，每一个图层类都有着各自的应用场景

- CALayer类是所有图层对象的根类，它定义了所有图层对象必须支持的行为，它也是支持图层的视图的默认图层类型。

类别	用途
CAEmitterLayer	用于实现基于Core Animation粒子发射系统。发射器层对象控制粒子的生成和起源
CAGradientLayer	用于绘制一个颜色渐变填充图层的形状（所有圆角矩形边界内的部分）
CAEAGLLayer/CAOpenGLLayer	用于设置需要使用OpenGL ES（iOS）或OpenGL（OS X）绘制的内容与内容储备。
CAReplicatorLayer	当你想自动生成一个或多个子层的拷贝。复制器为你生成拷贝并使用你指定的属性值以修改复制品的外观和属性。
CAScrollLayer	用于管理由多个子区域组成的大的可滚动区域
CAShaperLayer	用于绘制三次贝塞尔曲线。CAShaperLayer对绘制基于路径的形状非常有帮助。因为CAShaperLayer总是生成一个最新的路径。而如果将路径画在图层储备中，一旦图层被缩放，形状就变形了。
CATextLayer	用于渲染一个无格式或属性文本字符
CATransformLayer	用于渲染一个真3D的图层层级。而不是由其他图层类实现的2D图层层级。
QCCompositionLayer	用于渲染一个Quartz组件元素（仅在OS X中有效）

创建图层对象

- 提供图层的内容
- 图层是管理app内容的数据对象。图层的内容由包含可视数据的位图构成。使用下述三种方式之一可给提供图层的内容：
    - 直接赋值一个UIImage对象给图层对象contents属性。（这个技术适用于图层内容从不或几乎不改变的情形。）
    - 赋值一个代理给图层，由代理负责绘制图层内容。（该技术适用于图层内容可能偶尔改变，且内容可由外部对象提供，比如视图。）
    - 定义一个CALayer的子类并覆盖类的绘图方法，有覆盖的方法返回图层的内容。（该技术适用于你需要创建自定义图层的子类，或者你想改变图层基本的绘图行为。）
- 你需要为提供图层内容而担心的时刻仅在手动创建图层对象。如果你的app中只包含支持图层的视图。那你不需要担心使用刚刚提到的这些提供图层的方法提供图层内容。支持图层的视图会使用尽可能高效的方式为与之相关的图层提供内容。

创建图层内容

- 使用图片为图层提供内容
- 因为一个图层仅是管理位图图片的容器，所以你可以直接赋值一个图片给图层的contents属性
- 赋值一个图片给图层很简单，只需要指定一张你想显示在屏幕上的具体的图片就可以了
- 图层将直接使用你提供的图片对象，并不会尝试创建自己的图片拷贝。当你的应用使用相同的图片在多个地方时，该行为可以节省许多内存。
- 你赋值的图片类型必须是CGImageRef类型（在OS X 10.6或之前版本，你也可以赋值一个NSImage对象。）
- 当赋值图片时，记住提供的图片的分辨率要与本地设备的分辨率相匹配。对于Retina显示设备，这可能也需要你去调整图片的contentsScale属性。

创建图层内容

- 使用代理提供图层的内容
- 如果图层的内容是动态改变的，你可以使用一个代理对象在需要的时候提供图层并更新内容。图层显示的时候，图层调用你的代理方法以提供需要的内容：
    - 如果你的代理实现了displayLayer(_ layer: CALayer)方法，实现方法负责创建位图并赋值给contents属性。
    - 如果你的代理实现的是drawLayer(_ layer: CALayer, inContext ctx: CGContext)方法，Core Animation创建一个位图，创建一个用于绘制位图的上下文，并调用代理方法填充该位图。你的代理方法所要做的是将内容画在图形上下文上。
- 代理对象必须实现displayLayer(_ layer: CALayer)或者drawLayer(_ layer: CALayer, inContext ctx: CGContext)方法之一。如果代理对象把这两个方法都实现了，图层只调用displayLayer(_ layer: CALayer)方法。
- 覆盖displayLayer(_ layer: CALayer)方法在当你的app更倾向于载入或创建想要显示的位图的情况下适用
- 如果你没有预渲染的图片或者辅助对象来创建位图。代理对象可以使用drawLayer(_ layer: CALayer, inContext ctx: CGContext)方法动态的绘制内容
- 带有自定义内容并支持图层的视图，你应该去覆盖视图的绘图方法。一个支持图层的视图自动创建它自己的图层代理并实现需要的代理方法，你不应该改变这个配置。相反，你应该实现你视图的drawRect(_ rect: CGRect)方法以绘制你的内容。

创建图层内容

- 由子类提供图层的内容
- 如果你实现了一个自定义的图层类，你可以覆盖图层类的绘图方法完成任何绘图的操作。用这种方法生成图层对象的自定义内容是罕见的,但是某些图层却拥有管理显示的内容能力。如CATiledLayer类通过将大图片拆成更小的可管理、可独立渲染的碎片来管理大的图片。因为只有图层知道在某一时刻哪一个碎片需要被渲染，图层会直接管理绘图的行为。
- 当子类化图层类，你可使用下述的两种方式绘制你的图层内容：
  - 覆盖图层的display方法并使用在方法中直接设置图层的contents属性。
  - 覆盖图层的drawInContext(_ ctx: CGContext)方法并将需要的内容绘制到提供的图形上下文中。
- 选择何种方法依赖于在绘图过程中你需要多少的控制。display方法是更新图层内容的主要入口点，所以覆盖这个方法让你处于完全的过程控制中。覆盖display方法也意味你需要负责contents属性创建CGImageRef对象。如果你只是想绘制内容（或让你的图层管理绘图操作），你可以覆盖drawInContext(_ ctx: CGContext)方法并让图层为你创建内容储备。

大纲

- 动画
- Core Animation基础
- 建立Layer对象
- 基于Layer的动画
- 参考阅读

基于图层的动画

- Core Animation提供的基础设施让轻松创建复杂图层动画变得异常简单，Core Animation扩展了所有拥有图层的视图
- 例如改变图层框架矩形的尺寸，改变其在屏幕上的位置，应用旋转变换，改变它的透明度。使用Core Animation初始化一个动画和改变属性一样简单，但你也可以显式的创建一个动画并设置动画的参数。

基于图层的动画

- 用简单的动画表现图层属性的变化

- 你可以以显式或隐式的执行简单的动画。隐式动画使用默认的定时器和动画属性展现动画。而显式动画需要你为动画对象配置一些参数。所以当默认的定时器能够很好的为你服务并且你所要的动画效果不需要太多代码时，隐式动画则非常的适合你。

- 简单的动画包括改变一个图层的属性，以及随着时间的推移，让Core Animation以动画的形式展现这些属性的变化。图层定义了许多会影响图层可视外观的属性。改变这些属性是以动画方式展现外观变化的一种方式。例如将图层的透明度从1.0修改为0.0，这将引起图层的淡出特效，最后图层变为透明。

- 重要：虽然你可以使用Core Animation接口直接让支持图层的视图产生动画，但这样做经常需要额外的步骤。

- 为了触发隐式动画，你所要做的是更新图层对象的属性。当更改的目标是图层树中的图层对象，更改将立即反映到对象上。而图层对象的可视外观并不会立即发生变化，Core Animation将图层对象属性的变化当做是一个触发器，用以创建和安排一个或多个可执行的隐式动画

基于图层的动画

- 隐式动画的方式呈现图层属性的变化

```
layer.opacity = 0.0
```

- 显式动画的方式呈现图层属性的变化（改变图层的透明度）

```
let animation = CABasicAnimation(keyPath:
"opacity")
animation.fromValue = 1.0
animation.toValue = 0.0
animation.duration = 3.0
theView.layer.addAnimation(animation, forKey: nil)
theView.layer.opacity = 0.0
```

基于图层的动画

- 用简单的动画表现图层属性的变化

- 创建一个显示动画，推荐是赋值一个值给动画对象的fromValue属性。如果你没有为该属性指定值，Core Animation将使用图层的当前值作为开始值。如果已经更新了属性作为它的最终值，这将致使fromValue属性值遭到干扰，结果可能并不是你想要的。

- 隐式动画会更新图层对象的值。而显示动画不会更改图层树中的数据。显示动画仅是创建了一个动画。在动画结束之后，Core Animation从图层中移除该动画对象并使用当前的数据值重绘图层。如果你想让显示动画的改变成为永久性的，你必须更新图层属性。

- 隐式和显示动画都会在当前运行循环周期结束之后开始执行，并且当前的线程必须拥有一个用于执行动画的运行循环runloop。如果你改变了图层的动画属性或者给图层添加了多个动画对象。所有这些动画将会同时被执行。你可以让动画对象在一个特殊的时间开始执行。

基于图层的动画

- 用关键帧动画表现一个图层属性的变化

- 尽管一个基于属性的动画是从开始值到结束值改变一个属性，一个CAKeyfarmeAnimation对象让你通过一个目标集合产生动画，在某种程度上，动画可能是线性的或者非线性。一个关键帧动画由一个目标数据值集合组成。每个值的时间应该是可达的

- 最简单的配置是你使用一个数组指定值和时间。对于一个图层位置的变化，你可以有一个跟随路径的变化

- 动画对象使用你指定的关键帧并通过在某个值到另一个在给定时间区间内插值的方式构建动画。

基于图层的动画

- 用关键帧动画表现一个图层属性的变化

```
// 创建一个实现两个弧线的CGPath（一个弹跳路径）
let thePath = CGPathCreateMutable()
CGPathMoveToPoint(thePath, nil, 74.0, 74.0)
CGPathAddCurveToPoint(thePath, nil, 74.0, 500.0, 320.0,
500.0, 320.0, 74.0)
CGPathAddCurveToPoint(thePath, nil, 320.0, 500.0, 566.0,
500.0, 566.0, 74.0)

//创建一个动画对象，指定位置属性作为键路径
let animation = CAKeyframeAnimation(keyPath: "position")
animation.path = thePath
animation.duration = 5.0

//为图层添加动画
theView.layer.addAnimation(animation, forKey: nil)
```

基于图层的动画

- 指定关键帧的值
- 关键帧的值是关键帧动画中最重要的部分。这些值定义了动画在整个执行期间内的行为。指定关键帧值的主要方式以对象数组作为它的值。但是对于包含CGPoint数据类型（比如图层的anchorPoint属性和position属性），你可以指定一个CGPathRef数据类型替代。
- 当指定一个关键帧值数组，你放到数组中内容依赖于属性需要的数据类型。你可以直接添加一些对象到数组中。然而一些对象必须在添加到数组中之前被转换为id类型，所有标量类型或结构体必须被包装为对象，比如：
    - 对于属性类型为CGRect（例如bounds和frame属性），使用NSValue对象包装每一个矩形。
    - 对于图层的变换属性，使用NSValue包装每一个CATransform3D矩阵。动画这个属性将引起关键帧动画给图层轮流应用每个变换矩阵。
    - 对于borderColor属性，直接添加到数组。
    - 对于属性为CGFloat类型，直接添加到数组。
    - 为了动画图层的内容属性，指定一个CGImageRef数据类型属性。
- 对于一个CGPoint数据类型的属性，你可以创建一个点数组，或者使用CGPathRef对象指定跟踪的路径。当你指定一个点数组，关键帧动画对象在每一个连续的点之间绘制一条线，并沿着这些线移动。当你指定一个CGPathRef对象，动画起始于路径的开始点并跟随路径线移动，这包括沿着任何曲面。你可以使用开放的或者封闭的路径。

基于图层的动画

- 指定关键帧动画的定时器
- 关键帧动画的定时与步调比基本动画来的要复杂。以下是几个用于控制定时和步调的属性：
    - calculationMode属性定义了计算动画定时的算法。该属性值会影响其他与定时相关属性的使用方式。
    - 线性和曲线动画，动画的calculationMode属性被设置为kCAAnimationLinear或CAAnimationCubic，属性值被用于提供定时器信息以生成动画。这些模式值让你最大化控制动画的定时器。
    - 节奏动画，动画的calculationMode属性被设置为kCAAnimationPaced或kCAAnimationCubicPaced，这些属性值不依赖由keyTimes或timingFunctions属性提供的额外定时器值。相反，定时器值被隐式地计算以提供一个常速率动画。
    - 离散动画，动画的calculationMode属性被设置为kCAAnimationDiscrete，该值将引起动画属性从一个关键帧跳到另一个没有任何补间动画的下一个关键帧。计算模式使用keyTimes属性值，但忽略timingFunctions属性。
    - keyTimes属性为应用在每一关键帧指定应用到每一个关键帧上的计时器。该属性只在calculationMode属性被设置为kCAAnimationLinear，kCAAnimaitonDiscrete，kCAAnimationCubic时被使用。它不使用在节奏动画中。
    - timingFunctions属性指定使用在每一个关键帧部分的定时曲线(该属性替换了继承的timingFunction属性)。

基于图层的动画

- 指定关键帧动画的定时器

- 如果你想自己处理动画的定时，可以使用kCAAnimationLinear或kCAAnimaitonCubic模式与keyTimes和timingFunctions属性。keyTimes定义了应用在每一关键帧的时间点。所有中间值的定时由定时函数控制，定时函数允许你对各个部分应用缓入或缓出曲线定时。如果你不指定任何定时函数，动画将会是线性的。

- 停止一个隐式动画的运行

- 动画通常直到运行结束才会停止，但是你也可以根据需要使用以下技术提前停止动画：

    - 为了从图层上移除单独的动画对象，调用图层的removeAnimationForKey(_ key: String)方法移除你的动画对象。该方法使用的键要与调用addAnimation(_ anim: CAAnimation, forKey key: String?)方法传入的键一致。你指定的键必须不为nil。

    - 为了移除图层的所有动画对象。调用图层的removeAllAnimations方法。该方法立即会移除所有进行中的动画，并使用图层当前的状态信息重绘图层。

- 注意：你不能直接移除图层的隐式动画。当你从图层上移除一个动画，Core Animation通过使用图层当前的值重绘图层做出响应。因为当前的值通常是动画的结束值，这可能会引起图层的外观突然的跳跃。如果你想图层的外观仍然在动画最后一帧出现的地方。你可以检索在呈现树中的对象的最终值，并设置这些值到图层树中的对象。

基于图层的动画

- 同时动画多个属性变化
- 如果你想同时给一个图层对象应用多个动画，你可以使用CAAnimationGroup对象将这些动画放在一个组里。通过使用单独的配置点，使用组对象简化了对多个动画对象的管理
- 应用于动画组的定时器和持续值将使用相同的值覆盖单个动画对象。

```
//animation 1
let widthAnimation = CAKeyframeAnimation(keyPath:
"borderWidth")
let widthValues = [1.0, 10.0, 5.0, 30.0, 0.5,
15.0, 2.0, 50.0, 0.0]
widthAnimation.values = widthValues
widthAnimation.calculationMode = kCAAnimationPaced

//animation 2
let colorAnimation = CAKeyframeAnimation(keyPath:
"borderColor")
let colorValues = [UIColor.greenColor().CGColor,
UIColor.redColor().CGColor,
UIColor.blueColor().CGColor]
colorAnimation.values = colorValues
colorAnimation.calculationMode = kCAAnimationPaced

//animation group
let groupAnimation = CAAnimationGroup()
groupAnimation.animations =
[colorAnimation,widthAnimation]
groupAnimation.duration = 5.0
theView.layer.addAnimation(groupAnimation, forKey:
"borderChanges")
```

基于图层的动画

- 检测一个动画的结束
- Core Animaiton提供对动画开始与结束的检测支持。这些通知是执行所有与动画相关的内务处理的最佳时刻。比如说你可能使用开始通知设置一些相关状态信息，使用对应的结束通知清理这些状态。
- 有两种不同的方式获取关于动画状态的通知：
    - 使用setCompletionBlock：方法添加一个完成块给当前的事务。当事务中的所有动画完成后，事务将执行你的完成块。
    - 给CAAnimaiton对象赋值一个代理，该代理实现了animationDidStart(_ anim: CAAnimation)方法和animationDidStop(_ anim: CAAnimation, finished flag: BooI)代理方法。
- 如果你想将两个动画链接在一起，使得当第一个动画结束之后启动第二个动画。不要使用动画通知。相反，使用动画对象的beginTime属性在希望的时间启动动画。为了将两个动画链接在一起，设置第二个动画的开始时间为第一个动画的结束时间。

基于图层的动画

- 不基于图层的动画

- 如果图层属于层支持的视图，建议使用由UIKit或AppKit的提供的基于视图的动画接口来创建动画。有很多办法使用Core Animation的接口直接对图层创建动画，但如何创建这些动画取决于目标的平台。

- iOS中修改图层的规则

- 由于iOS的视图总是有一个底层，UIView类本身大部分数据都直接从层对象派生。其结果是，我们对图层进行的改变自动通过视图对象反映出来。这意味着，我们既可以使用Core Animation的接口也可以使用UIView的接口来创建动画。

- 如果你想使用Core Animation类来创建动画，你必须从基于视图的动画闭包内部发出动画的要求。 在默认情况下，UIView类禁用层动画，但是会在这些动画闭包中重新启用动画。所以动画块外的任何更改都不会生效。

基于图层的动画

- 这个例子展示了如何显式更改图层的不透明度和它的位置。在这个例子中，myNewPosition变量预先计算并且预先由闭包捕获。两个动画起始于相同的时间，但不透明度动画运行用默认定时，而位置动画运行与在其动画对象中指定的定时。

```
UIView.animateWithDuration(1.0, animations: {
    self.imageView.layer.opacity = 0.0

    let animation = CABasicAnimation(keyPath: "position")
    animation.fromValue = NSValue.init(CGPoint:
self.imageView.layer.position)
    animation.toValue = NSValue.init(CGPoint:
CGPoint.init(x: 150, y: 200))
    animation.duration = 3.0
    self.imageView.layer.addAnimation(animation, forKey:
"AnimateFrame")
})
```

大纲

- 动画
- Core Animation基础
- 建立Layer对象
- 基于Layer的动画
- 参考阅读

参考阅读

- **Core Animation Programming Guide** (https://developer.apple.com/library/prerelease/ios/documentation/Cocoa/Conceptual/CoreAnimation_guide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40004514-CH1-SW1)

- **CAAnimation Class Reference**(https://developer.apple.com/library/mac/documentation/GraphicsImaging/Reference/CAAnimation_class/#//apple_ref/occ/instm/NSObject/animationDidStop:finished:)

- **CALayer Class Reference**(https://developer.apple.com/library/ios/documentation/GraphicsImaging/Reference/CALayer_class/#//apple_ref/occ/instm/CALayer/addAnimation:forKey:)