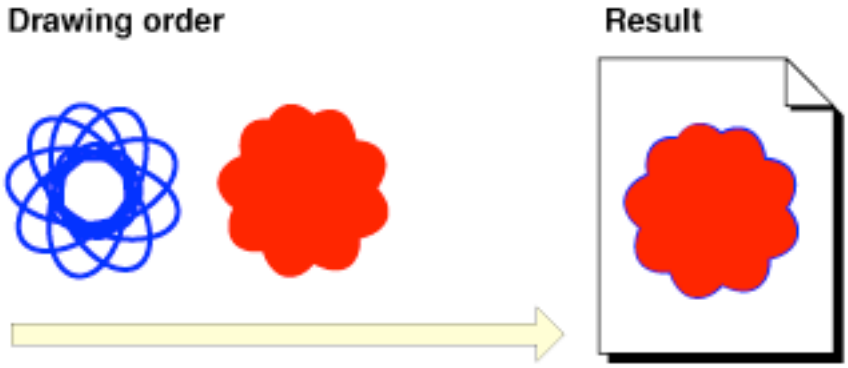
Quartz 2D 编程指南(1) — 概览

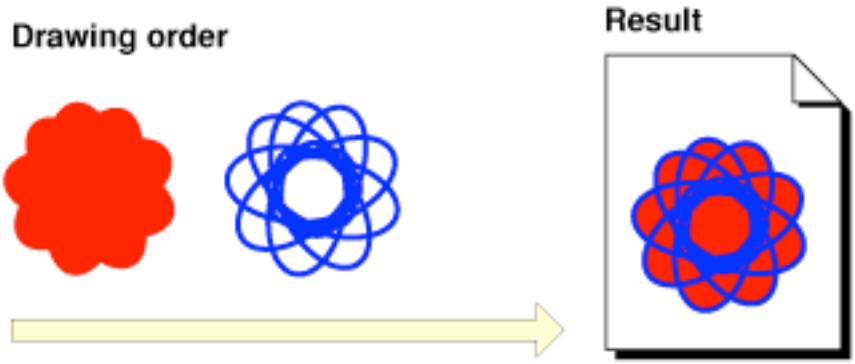
Quartz 2D 是一个二维图形绘制引擎，支持 iOS 环境和 Mac OS X 环境。我们可以使用 Quartz 2D API 来实现许多功能，如基本路径的绘制、透明度、描影、绘制阴影、透明层、颜色管理、反锯齿、PDF 文档生成和 PDF 元数据访问。在需要的时候，Quartz 2D 还可以借助图形硬件的功能。
在 Mac OS X 中，Quartz 2D 可以与其它图形图像技术混合使用，如 Core Image、Core Video、OpenGL、QuickTime。例如，通过使用 QuickTime 的 GraphicsImportCreateCGImage 函数，可以用 Quartz 从一个 QuickTime 图形导入器中创建一个图像。

Page

Quartz 2D 在图像中使用了绘画者模型(painter’s model)。在绘画者模型中，每个连续的绘制操作都是将一个绘制层(a layer of ‘paint’)放置于一个画布(‘canvas’)，我们通常称这个画布为(Page)。 Page 上的绘图可以通过额外的绘制操作来叠加更多的绘图。Page 上的图形对象只能通过叠加更多的绘图来改变。这个模型允许我们使用小的图元来构建复杂 的图形。
图 1-1 展示了绘画者模型如何工作。从图中可以看出不同的绘制顺序所产生的效果不一样。

Figure 1-1 The painter's model



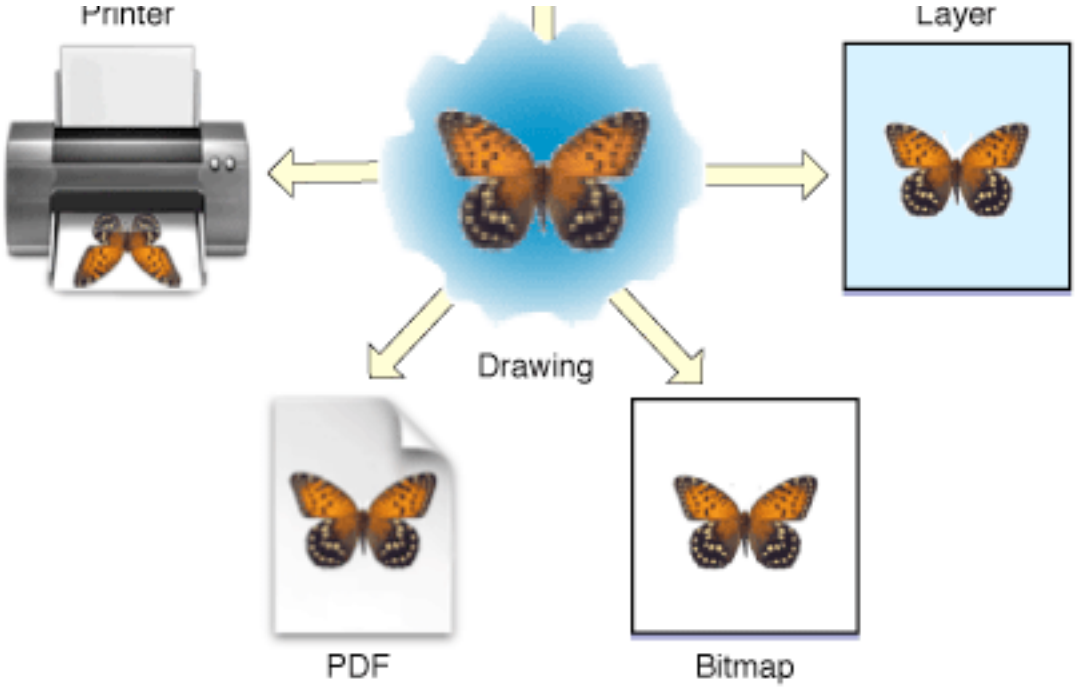


Page 可以是一张纸(如果输出设备是打印机)，也可以是虚拟的纸张(如果输出设备是 PDF 文件)，还可以是 bitmap 图像。这根据实际使用的 graphics context 而定。

绘制目标：Graphics Context

Graphics Context 是一个数据类型(CGContextRef)，用于封装 Quartz 绘制图像到输出设备的信息。设备可以是 PDF 文件、bitmap 或者显示器的窗口上。Graphics Context 中的信息包括在 Page 中的图像的图形绘制参数和设备相关的表现形式。Quartz 中所有的对象都是绘制到一个 Graphics Context 中。

我 们可以将 Graphics Context 想像成绘制目标，如图 1-2 所示。当用 Quartz 绘图时，所有设备相关的特性都包含在我们所使用的 Graphics Context 中。换句话说，我们可以简单地给 Quartz 绘图序列指定不同的 Graphics Context，就可将相同的图像绘制到不同的设备上。我们不需要任何设备相关的计算；这些都由 Quartz 替我们完成。

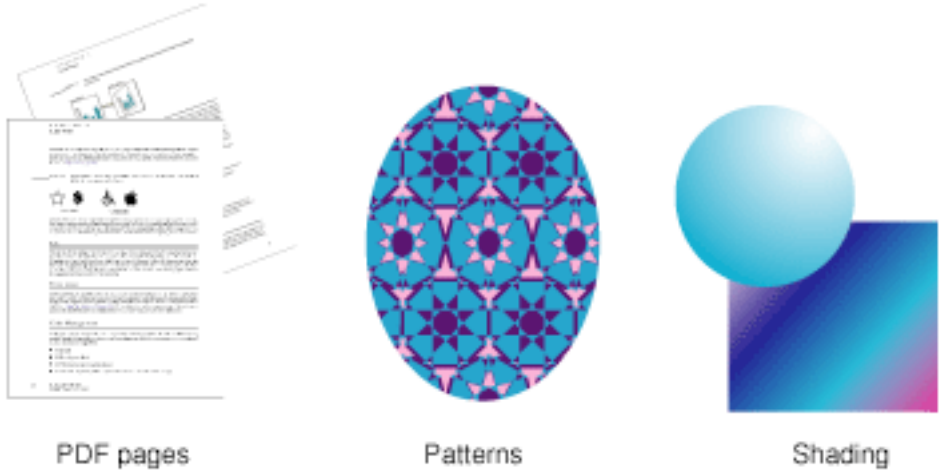


Quartz 提供了以下几种类型的 Graphics Context，详细的介绍将在后续章节说明。

- Bitmap Graphics Context
- PDF Graphics Context
- Window Graphics Context
- Layer Context
- Post Graphics Context

Quartz 2D 数据类型

除了 Graphics Context 之外，Quartz 2D API 还定义一些数据类型。由于这些 API 就 Core Graphics 框架的一部分，所以这些数据类型都是以 CG 开头的。Quartz 2D 使用这些数据类型来创建对象，通过操作这些对象来获取特定的图形。



下面列出了 Quartz 2D 包含的数据类型：

- CGPathRef：用于向量图，可创建路径，并进行填充或描画(stroke)
- CGImageRef：用于表示 bitmap 图像和基于采样数据的 bitmap 图像遮罩。
- CGLayerRef：用于表示可用于重复绘制(如背景)和幕后(offscreen)绘制的绘画层
- CGPatternRef：用于重绘图
- CGShadingRef、CGGradientRef：用于绘制渐变
- CGFunctionRef：用于定义回调函数，该函数包含一个随机的浮点值参数。当为阴影创建渐变时使用该类型
- CGColorRef, CGColorSpaceRef：用于告诉 Quartz 如何解释颜色
- CGImageSourceRef,CGImageDestinationRef：用于在 Quartz 中移入移出数据
- CGFontRef：用于绘制文本
- CGPDFDictionaryRef, CGPDFObjectRef, CGPDFPageRef, CGPDFStream, CGPDFStringRef, and CGPDFArrayRef：用于访问 PDF 的元数据
- CGPDFScannerRef, CGPDFContentStreamRef：用于解析 PDF 元数据
- CGPSConverterRef：用于将 PostScript 转化成 PDF。在 iOS 中不能使用。

图形状态

Quartz 通过修改当前图形状态(current graphics state)来修改绘制操作的结果。图形状态包含用于绘制程序的参数。绘制程序根据这些绘图状态来决定如何渲染结果。例如，当你调用设置填充颜色的函数时，你将改变存储在当前绘图状态中的颜色值。

Graphics Context 包含一个绘图状态栈。当 Quartz 创建一个 Graphics Context

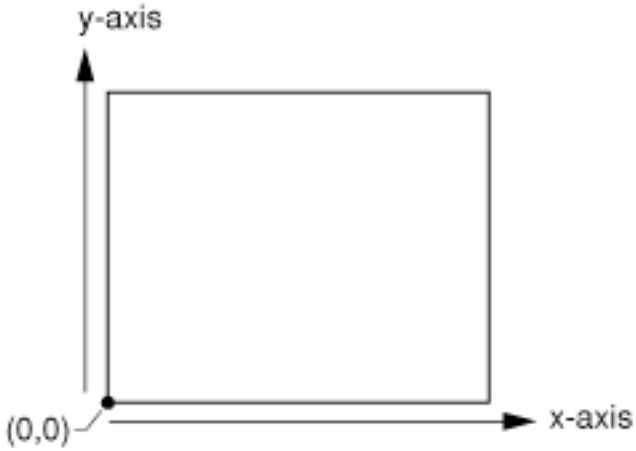
时，栈为空。当保存图形状态时，Quartz 将当前图形状态的一个副本压入栈中。当还原图形状态时，Quartz 将栈顶的图形状态出栈。出 栈的状态成为当前图形状态。
可使用函数 CGContextSaveGState 来保存图形状态，CGContextRestoreGState 来还原图形状态。
注意：并不是当前绘制环境的所有方面都是图形状态的元素。如，图形状态不包含当前路径(current path)。下面列出了图形状态相关的参数：

- Current transformation matrix (CTM)：当前转换矩阵
- Clipping area：裁剪区域
- Line: 线
- Accuracy of curve estimation (flatness)：曲线平滑度
- Anti-aliasing setting：反锯齿设置
- Color: 颜色
- Alpha value (transparency)：透明度
- Rendering intent：渲染目标
- Color space: 颜色空间
- Text: 文本
- Blend mode：混合模式

Quartz 2D 坐标系统

坐标系统定义是被绘制到 Page 上的对象的位置及大小范围，如图 1-4 所示。我们在用户空间坐标系统(user-space coordination system，简称用户空间)中指定图形的位置及大小。坐标值是用浮点数来定义的。

Figure 1-4 The Quartz coordinate system

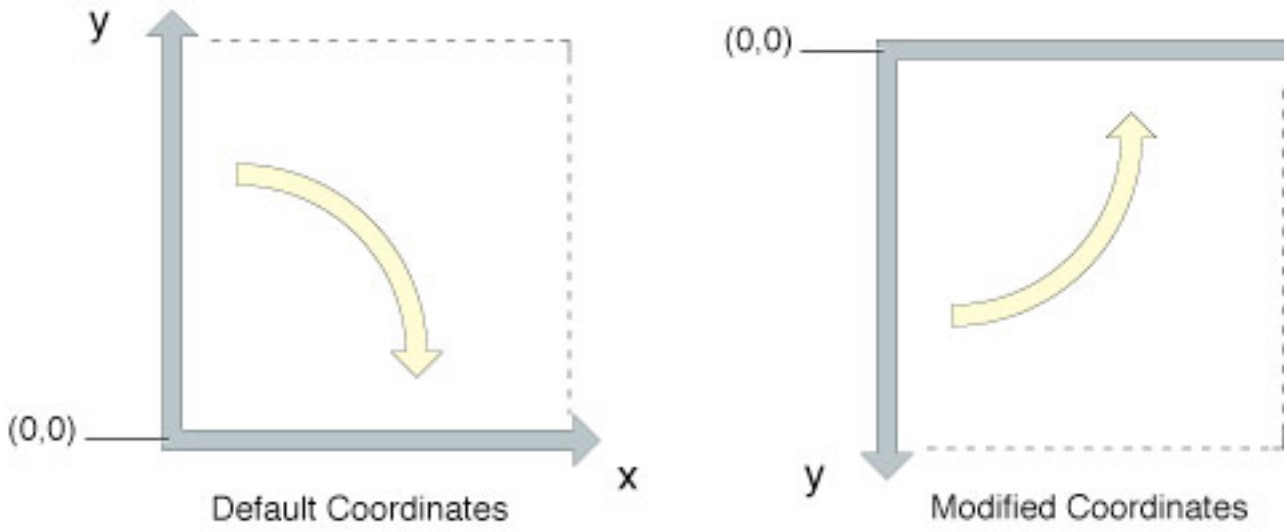


由于不同的设备有不同的图形功能，所以图像的位置及大小依赖于设备。例如，一个显示设备可能每英寸只能显示少于 96 个像素，而打印机可能每英寸能显示 300 个像素。如果在设备级别上定义坐标系统，则在一个设备上绘制的图形无法在其它设备上正常显示。

Quartz 通过使用当前转换矩阵(current transformation matrix， CTM)将一个独立的坐标系统(user space)映射到输出设备的坐标系统(device space)，以此来解决设备依赖问题。 CTM 是一种特殊类型的矩阵(affine transform, 仿射矩阵)，通过平移(translation)、旋转(rotation)、缩放(scale)操作将点从一个坐标空间映射到另外一个坐标空间。
CTM 还有另外一个目的：允许你通过转换来决定对象如何被绘制。例如，为了绘制一个旋转了 45 度的盒子，我们可以在绘制盒子之前旋转 Page 的坐标系统。Quartz 使用旋转过的坐标系统来将盒子绘制到输出设备中。
用户空间的点用坐标对(x, y)来表示，(0, 0)表示坐标原点。Quartz 中默认的坐标系统是：沿着 x 轴从左到右坐标值逐渐增大；沿着 y 轴从下到上坐标值逐渐增大。
有 一些技术在设置它们的 graphics context 时使用了不同于 Quartz 的默认坐标系统。相对于 Quartz 来说，这些坐标系统是修改的坐标系统(modified coordinate system)，当在这些坐标系统中显示 Quartz 绘制的图形时，必须进行转换。最常见的一种修改的坐标系统是原点位于左上角，而沿着 y 轴从上到下坐标 值逐渐增大。我们可以在如下一些地方见到这种坐标系统：

- 在 Mac OS X 中，重写过 isFlipped 方法以返回 yes 的 NSView 类的子类
- 在 IOS 中，由 UIView 返回的绘图上下文
- 在 IOS 中，通过调用 UIGraphicsBeginImageContextWithOpti*****函数

Figure 1-5 Modifying the coordinate system creates a mirrored image.



我们的应用程序负责调整 Quartz 调用以确保有一个转换应用到上下文中。例如，如果你想要一个图片或 PDF 正 确的绘制到一个 Graphics Context 中，你的应用程序可能需要临时调整 Graphics Context 的 CTM。在 IOS 中，如果使用 UIImage 对象来包裹创建的 CGImage 对象，可以不需要修改 CTM。UIImage 将自动进行补偿 以适用 UIKit 的坐标系统。

复制代码

1. 重 要：如果你打算在 IOS 上开发与 Quartz 相关的程序，了解以上所讨论的是很有用的，但不是必须的。在 IOS 3.2 及后续的版本中，当 UIKit 为你的应用程序创建一个绘图上下文时，也对上下文进行了额外的修改以匹配 UIKit 的约定。特别的， patterns 和 shadows(不被 CTM 影响)单独进行调整以匹配 UIKit 坐标系统。在这种情况下，没有一个等价的机制让 CTM 来转换 Quartz 和 UIKit 的 上下文。我们必须认识到在什么样的上下文中进行绘制，并调整行为以匹配上下文的预期。

内存管理：对象所有权

Quartz 使用 Core Foundation 内存管理模型(引用计数)。所以，对象的创建与销毁与通常的方式是一样的。在 Quartz 中， 需要记住如下一些规则：

- 如果创建或拷贝一个对象，你将拥有它，因此你必须释放它。通常，如果使用含有”Create”或“Copy”单词的函数获取一个对象，当使用完后必须释放，否则将导致内存泄露。
- 如果使用不含有”Create”或“Copy”单词的函数获取一个对象，你将不会拥有对象的引用， 不需要释放它。
- 如 果你不拥有一个对象而打算保持它，则必须 retain 它并且在不需要时 release 掉。可以使用 Quartz 2D 的函数来指定 retain 和 release 一个对象。例如，如果创建了一个 CGColorspace 对象，则使用函数 CGColorSpaceRetain 和 CGColorSpaceRelease 来 retain 和 release 对象。同样， 可以使用 Core Foundation 的 CFRetain 和 CFRelease，但是注意不能传递 NULL 值给这些函数。

来源于 教程 分类

Quartz 2D 编程指南(2) — 图形上下文(Graphics Contexts)

一个 Graphics Context 表示一个绘制目标。它包含绘制系统用于完成绘制指令的绘制参数和设备相关信息。Graphics Context 定义了基本的绘制属性，如颜色、裁减区域、线条宽度和样式信息、字体信息、混合模式等。

我 们可以通过几种方式来获取 Graphics Context：Quartz 提供的创建函数、Mac OS X 框架或 IOS 的 UIKit 框架提供的函数。Quartz 提供了多种 Graphics Context 的创建函数，包括 bitmap 和 PDF，我们可以使用这些 Graphics Context 创建自定义的内容。

本章介绍了 如何为不同的绘制目标创建 Graphics Context。在代码中，我们用 CGContextRef 来表示一个 Graphics Context。当获得一个 Graphics Context 后，可以使用 Quartz 2D 函数在上下文(context)中进行绘制、完成操作(如平移)、修改图形状态参数(如线宽和填充颜色)等。

在 iOS 中的视图 Graphics Context 进行绘制

在 iOS 应用程序中，如果要在屏幕上进行绘制，需要创建一个 UIView 对象，并实现它的 drawRect:方法。视图的 drawRect:方法在视图显示 在屏幕上及它的内容需要更新时被调用。在调用自定义的 drawRect:后，视图对象自动配置绘图环境以便代码能立即执行绘图操作。作为配置的一部分，视 图对象将为当前的绘图环境创建一个 Graphics Context。我们可以通过调用 UIGraphicsGetCurrentContext 函数来获取这个 Graphics Context。

UIKit 默认的坐标系统与 Quartz 不同。在 UIKit 中，原点位于左上角，y 轴正方向为向下。UIView 通过将修改 Quartz 的 Graphics Context 的 CTM[原点平移到左下角，同时将 y 轴反转(y 值乘以-1)]以使其与 UIView 匹配。

在 Mac OS X 中创建一个窗口 Graphics Context

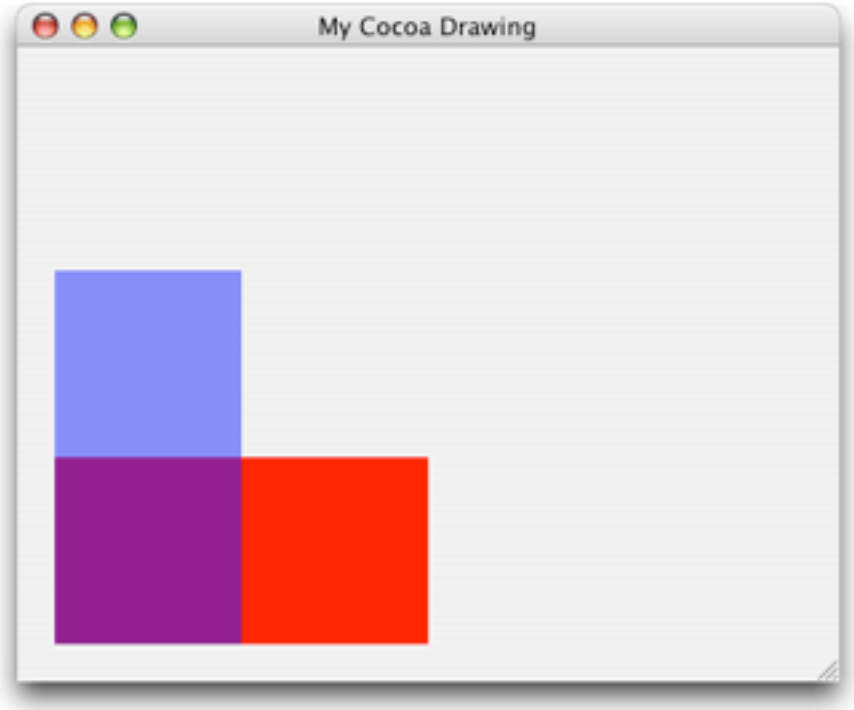
在 Mac OS X 中绘制时，我们需要创建一个窗口 Graphics Context。Quartz 2D API 没有提供函数来获取窗口 Graphics Context。取而代之的是用 Cocoa 框架来获取一个窗口上下文。

我们可以在 Cocoa 应用程序的 drawRect:中获取一个 Quartz Graphics Context，如下代码所示：
复制代码

```
1. CGContextRef myContext = [[NSGraphicsContext
   currentContext] graphicsPort];
```

currentContext 方法在当前线程中返回 NSGraphicsContext 实例。graphicsPort 方法返回一个低级别、平台相关的 Graphics Context(Quartz Graphics Context)。

Figure 2-1 A view in the Cocoa framework that contains Quartz drawing



为了实现图 2-1 实例，需要先创建一个 Cocoa 应用程序。在 Interface Builder 中，拖动一个 Custom View 到窗口中，并子类化。然后实现子类视图的，如代码清单 2-1 所示。视图的 drawRect:包含了所有的 Quartz 绘制代码。
引用
注：NSView 的 drawRect:方法在每次视图需要绘制时自动调用。

复制代码

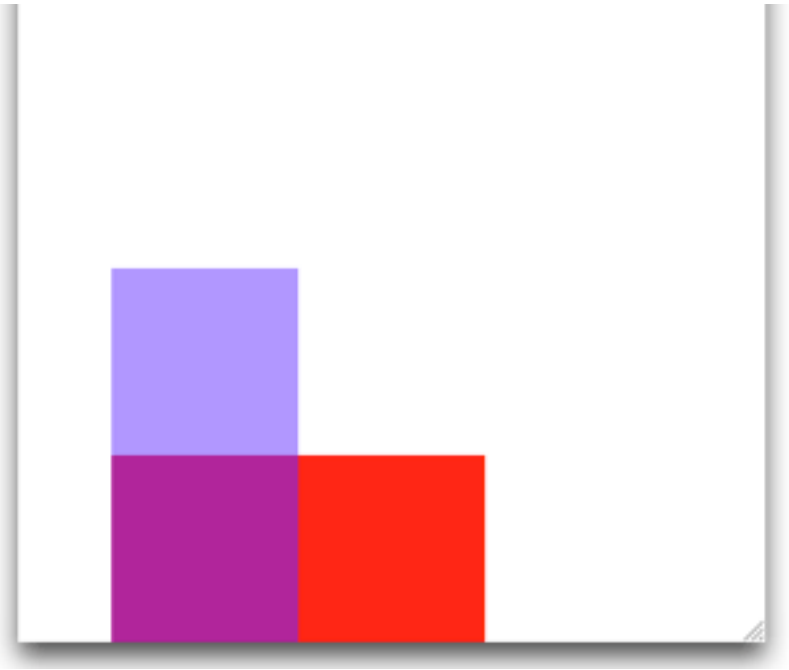
```
1. Listing 2-1    Drawing to a window graphics context
2. @implementation MyQuartzView
3. - (id)initWithFrame:(NSRect)frameRect
4. {
5.         self = [super initWithFrame:frameRect];
6.         return self;
7. }
8.
9. - (void)drawRect:(NSRect)rec
10. {
```

```
11.       CGContextRef myContext =
   [[NSGraphicsContext   currentContext]
   graphicsPort]; //1
12.
13.      // ********** Your drawing code here
   **********             //2
14.       CGContextSetRGBFillColor (myContext, 1, 0, 0,
   1);          //3
15.       CGContextFillRect (myContext, CGRectMake (0,
   0, 200, 100 ));      //4
16.       CGContextSetRGBFillColor (myContext, 0, 0,
   1, .5);          //5
17.       CGContextFillRect (myContext, CGRectMake (0,
   0, 100, 200));      //6
18.}
19.@end
```

代码说明：
1.为视图获取一个 Graphics Context
2.插入绘图代码的地方。以下四行是使用 Quartz 2D 函数的例子
3.设置完全不透明的红色填充色。
4.填充一个长方形，其原点为(0, 0)，大小为(200, 100)
5.设置半透明的蓝色填充色。
6.填充一个长方形，其原点为(0, 0)，大小为(100, 200)

创建一个 PDF Graphics Context

当 创建一个 PDF Graphics Context 并绘制时，Quartz 将绘制操作记录为一系列的 PDF 绘制命令并写入文件中。我们需要提供一个 PDF 输出的位置及一个默认的 media box(用于指定页面边界的长方形)。图 2-2 显示了在 PDF Graphics Context 中绘制及在 preview 打开 PDF 的结果。



Quartz 2D API 提供了两个函数来创建 PDF Graphics Context:

- CGPDFContextCreateWithURL：当你需要用 Core Foundation URL 指定 pdf 输出的位置时使用该函数。代码清单 2-2 显示了该函数的使用方法(代码 2-2 及后面代码的详细解释略)：

复制代码

```
1. Listing 2-2    Calling CGPDFContextCreateWithURL
   to create a PDF graphics context
2. CGContextRef MyPDFContextCreate (c*****t CGRect
   *inMediaBox, CFStringRef path)
3. {
4.       CGContextRef myOutContext = NULL;
5.       CFURLRef url;
6.       url = CFURLCreateWithFileSystemPath (NULL,
   path, kCFURLPOSIXPathStyle, false);
7.
8.       if (url != NULL) {
```

```
9.              myOutContext =
   CGPDFContextCreateWithURL
   (url,    inMediaBox,    NULL);
10.             CFRelease(url);
11.       }
12.       return myOutContext;
13.}
```

- CGPDFContextCreate：当需要将 pdf 输出发送给数据用户时使用该方法。代码清单 2-3 显示了该函数的使用方法：

复制代码

```
1. Listing 2-3     Calling CGPDFContextCreate to create
   a PDF graphics context
2. CGContextRef MyPDFContextCreate (c*****t CGRect
   *inMediaBox, CFStringRef path)
3. {
4.       CGContextRef             myOutContext = NULL;
5.       CFURLRef                   url;
6.       CGDataC*****umerRef     dataC*****umer;
7.
8.       url = CFURLCreateWithFileSystemPath
   (NULL,    path, kCFURLPOSIXPathStyle, false);
9.
10.      if (url != NULL)
11.      {
12.             dataC*****umer =
   CGDataC*****umerCreateWithURL (url);
13.             if (dataC*****umer != NULL)
14.             {
15.                    myOutContext =
   CGPDFContextCreate (dataC*****umer, inMediaBox,
   NULL);
16.                    CGDataC*****umerRelease
   (dataC*****umer);
17.             }
18.
19.             CFRelease(url);
20.      }
21.
22.      return myOutContext;
```

```
23.}
```

代码清单 2-4 显示是如何调用 MyPDFContextCreate 程序及绘制操作。
复制代码

```
1. Listing 2-4    Drawing to a PDF graphics context
2.       CGRect mediaBox;
3.
4.       mediaBox = CGRectMake (0, 0, myPageWidth, myPageHeight);
5.       myPDFContext = MyPDFContextCreate (&mediaBox,
   CFSTR("test.pdf"));
6.
7.       CFStringRef myKeys[1];
8.       CFTypeRef myValues[1];
9.
10.      myKeys[0] = kCGPDFContextMediaBox;
11.      myValues[0] = (CFTypeRef) CFDataCreate(NULL,(c*****t UInt8
   *)&mediaBox, sizeof (CGRect));
12.
13.      CFDictionaryRef pageDictionary = CFDictionaryCreate(NULL,
   (c*****t void **) myKeys,
14.
                        (c*****t void **) myValues, 1,
15.
                        &kCFTypeDictionaryKeyCallBacks,
16.
                        & kCFTypeDictionaryValueCallBacks);
17.
18.      CGPDFContextBeginPage(myPDFContext, &pageDictionary);
19.            // ********** Your drawing code here **********
20.            CGContextSetRGBFillColor (myPDFContext, 1, 0, 0, 1);
21.            CGContextFillRect (myPDFContext, CGRectMake (0, 0,
   200, 100 ));
22.            CGContextSetRGBFillColor (myPDFContext, 0, 0, 1, .5);
23.            CGContextFillRect (myPDFContext, CGRectMake (0, 0,
   100, 200 ));
24.      CGPDFContextEndPage(myPDFContext);
25.
26.      CFRelease(pageDictionary);
27.      CFRelease(myValues[0]);
28.      CGContextRelease(myPDFContext);
```

我们可以将任何内容(图片，文本，绘制路径)绘制到 pdf 中，并能添加链接及加

密。

创建位图 Graphics Context

一个位图Graphics Context接受一个指向内存缓存(包含位图存储空间)的指针，当我们绘制一个位图 Graphics Context 时，该缓存被更新。在释放 Graphics Context 后，我们将得到一个我们指定像素格式的全新的位图。
引用
注：位图 Graphics Context 有时用于后台绘制。CGLayer 对象优化了后台绘制，因为 Quartz 在显卡上缓存了层。

引用
iOS 提示：iOS 应用程序使用了 UIGraphicsBeginImageContextWithOpti***** 取代 Quartz 低层函数。如果使用 Quartz 创建一下后台 bitmap,bitmap Graphics Context 使用的坐标系统是 Quartz 默认的坐标系统。而使用 UIGraphicsBeginImageContextWithOpti***** 创建图形上下文，UIKit 将会对坐标系统使用与 UIView 对象的图形上下文一样的转换。这允许应用程序使用相同的绘制代码而不需要担心坐标系统问题。 虽然我们的应用程序可以手动调整 CTM 达到相同的效果，但这种做没有任何好处。

我们使用 CGBitmapContextCreate 来创建位图 Graphics Context，该函数有如下参数：

- data：一个指向内存目标的指针，该内存用于存储需要渲染的图形数据。内存块的大小至少需要(bytePerRow * height)字节。
- width：指定位图的宽度，单位是像素(pixel)。
- height：指定位图的高度， 单位是像素(pixel)。
- bitsPerComponent：指定内存中一个像素的每个组件使用的位数。例如，一个 32 位的像素格式和一个 rgb 颜色空间，我们可以指定每个组件为 8 位。
- bytesPerRow：指定位图每行的字节数。
- colorspace：颜色空间用于位图上下文。在创建位图 Graphics Context 时，我们可以使用灰度(gray), RGB, CMYK, NULL 颜色空间。
- bitmapInfo：位图的信息，这些信息用于指定位图是否需要包含 alpha 组件，像素中 alpha 组件的相对位置(如果有的话)，alpha 组件是否是预乘的，及颜色组件是整型值还是浮点值。

代码清单 2-5 显示了如何创建位图 Graphics Context。当向位图 Graphics Context 绘图时，Quartz 将绘图记录到内存中指定的块中。
复制代码

```
1. Listing 2-5    Creating a bitmap graphics context
2. CGContextRef MyCreateBitmapContext (int
   pixelsWide, int pixelsHigh)
3. {
4.        CGContextRef      context = NULL;
```

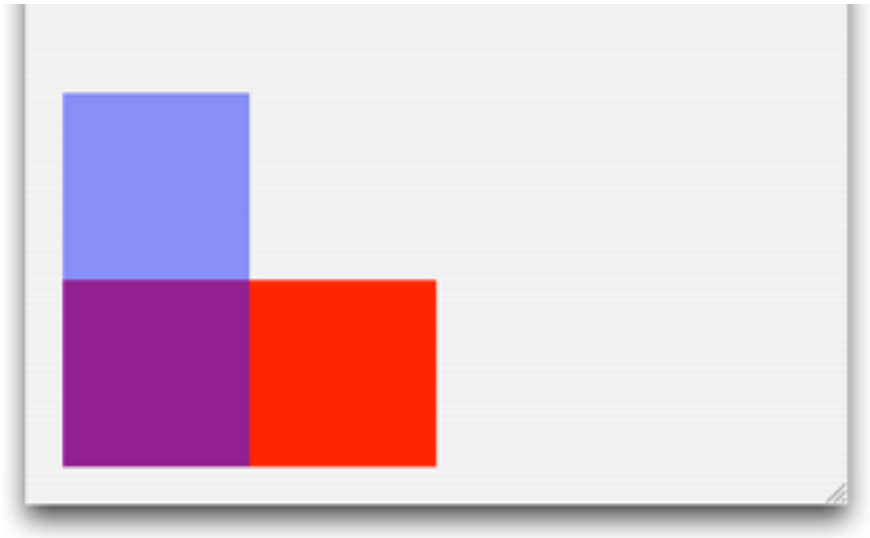
```
5.        CGColorSpaceRef colorSpace;
6.        void *                bitmapData;
7.        int                    bitmapByteCount;
8.        int                    bitmapBytesPerRow;
9.
10.       bitmapBytesPerRow    = (pixelsWide * 4);
11.       bitmapByteCount       = (bitmapBytesPerRow *
   pixelsHigh);
12.       colorSpace =
   CGColorSpaceCreateWithName(kCGColorSpaceGenericRG
   B);
13.       bitmapData = calloc( bitmapByteCount );
14.       if (bitmapData == NULL)
15.       {
16.             fprintf (stderr, "Memory not
   allocated!");
17.             return NULL;
18.       }
19.       context = CGBitmapContextCreate (bitmapData,
   pixelsWide, pixelsHigh, 8, bitmapBytesPerRow,
   colorSpace, kCGImageAlphaPremultipliedLast);
20.
21.       if (context== NULL)
22.       {
23.             free (bitmapData);
24.             fprintf (stderr, "Context not
   created!");
25.             return NULL;
26.       }
27.       CGColorSpaceRelease( colorSpace );
28.       return context;
29.}
```

代 码清单 2-6 显示了调用 MyCreateBitmapContext 创建一个位图 Graphics Context，使用位图 Graphics Context 来创建 CGImage 对象，然后将图片绘制到窗口 Graphics Context 中。绘制结果如图 2-3 所示：

复制代码

```
1. Listing 2-6    Drawing to a bitmap graphics context
2.        CGRect myBoundingBox;
3.        myBoundingBox = CGRectMake (0, 0, myWidth,
   myHeight);
4.        myBitmapContext = MyCreateBitmapContext
```

```
(400, 300);
5.
6.        // ********** Your drawing code here
**********
7.        CGContextSetRGBFillColor (myBitmapContext,
1, 0, 0, 1);
8.        CGContextFillRect (myBitmapContext,
CGRectMake (0, 0, 200, 100 ));
9.        CGContextSetRGBFillColor (myBitmapContext,
0, 0, 1, .5);
10.       CGContextFillRect (myBitmapContext,
CGRectMake (0, 0, 100, 200 ));
11.       myImage = CGBitmapContextCreateImage
(myBitmapContext);
12.       CGContextDrawImage(myContext,
myBoundingBox, myImage);
13.       char *bitmapData =
CGBitmapContextGetData(myBitmapContext);
14.       CGContextRelease (myBitmapContext);
15.       if (bitmapData) free(bitmapData);
16.       CGImageRelease(myImage);
```



支持的像素格式

表 2-1 总结了位图 Graphics Context 支持的像素格式，相关的颜色空间及像素格式支持的 Mac OS X 最早版本。像素格式用 bpp(每像素的位数)和 bpc(每个组件的位数)来表示。表格同时也包含与像素格式相关的位图信息常量。

表 2-1：位图 Graphics Context 支持的像素格式

Null8 bpp, 8 bpc, kCGImageAlphaOnly	Mac OS X, iOS
Gray8 bpp, 8 bpc, kCGImageAlphaNone	Mac OS X, iOS
Gray8 bpp, 8 bpc, kCGImageAlphaOnly	Mac OS X, iOS
Gray16 bpp, 16 bpc, kCGImageAlphaNone	Mac OS X
Gray32 bpp, 32 bpc, kCGImageAlphaNone|kCGBitmapFloatComponents	Mac OS X
RGB 16 bpp, 5 bpc, kCGImageAlphaNoneSkipFirst	Mac OS X, iOS
RGB 32 bpp, 8 bpc, kCGImageAlphaNoneSkipFirst	Mac OS X, iOS
RGB 32 bpp, 8 bpc, kCGImageAlphaNoneSkipLast	Mac OS X, iOS
RGB 32 bpp, 8 bpc, kCGImageAlphaPremultipliedFirst	Mac OS X, iOS

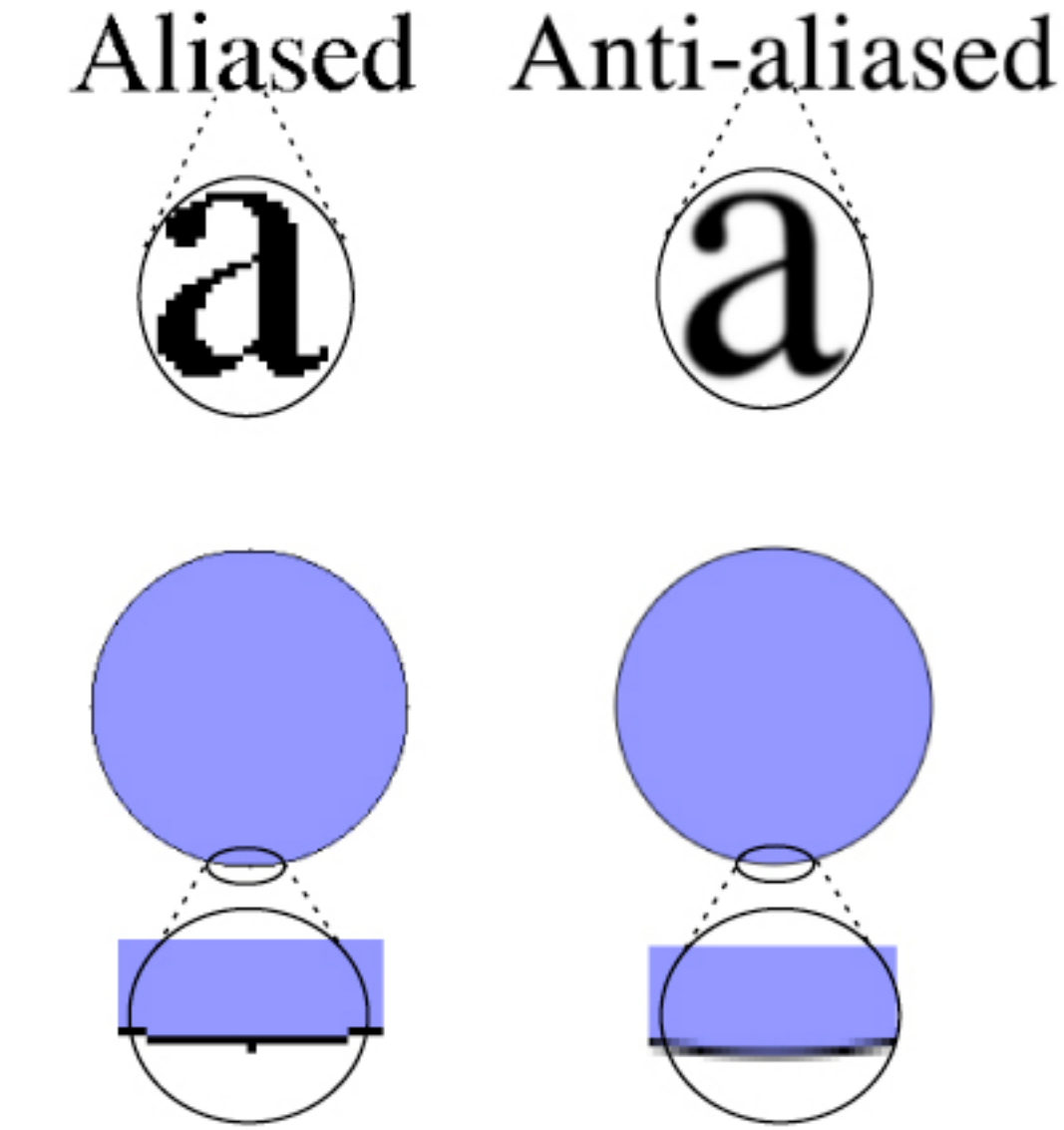
RGB 32 bpp, 8 bpc, kCGImageAlphaPremultipliedLast	Mac OS X, iOS
RGB 64 bpp, 16 bpc, kCGImageAlphaPremultipliedLast	Mac OS X
RGB 64 bpp, 16 bpc, kCGImageAlphaNoneSkipLast	Mac OS X
RGB 128 bpp, 32 bpc, kCGImageAlphaNoneSkipLast | kCGBitmapFloatComponents	Mac OS X
RGB 128 bpp, 32 bpc, kCGImageAlphaPremultipliedLast | kCGBitmapFloatComponents	Mac OS X
CMYK32 bpp, 8 bpc, kCGImageAlphaNone	Mac OS X
CMYK64 bpp, 16 bpc, kCGImageAlphaNone	Mac OS X
CMYK 128 bpp, 32 bpc, kCGImageAlphaNone | kCGBitmapFloatComponents	Mac OS X

反锯齿

位 图 Graphics Context 支持反锯齿，这一操作是人为的较正在位图中绘制文本或形状时产生的锯齿边缘。当位图的分辨率明显低于人眼的分辨率时就会产生锯齿。为了使位 图中的对象显得平滑，Quartz 使用不同的颜色来填充形状周边的像素。通过这种方式来混合颜色，使形状看起来更平滑。如图 2-4 显示的效果。我们可以通 过调用 CGContextSetShouldAntialias 来关闭位图 Graphics Context 的反锯齿效果。反锯齿设置是图形状态的一部分。

可以调用函数 CGContextSetAllowsAntialiasing 来控制一个特定 Graphics Context 是否支持反锯齿；false 表示不支持。该设置不是图形状态的一部分。当上下文及图形状态设置为 true 时，Quartz 执行反锯齿。

Figure 2-4 A comparison of aliased and anti-aliasing drawing



获取打印的 Graphics Context

Mac OS X 中的 Cocoa 应用程序通过自定义的 NSView 子类来实现打印。一个视图通过调用 print:方法来进行打印。然后视图以打印机为目标创建一个 Graphics Context，并调用 drawRect:方法。应用程序使用与在屏幕进行绘制相同的绘制代码。我们同样可以自定义 drawRect: 方法将图形绘制到打印机。

Quartz 2D 编程指南(4) — 颜色和颜色空间

不同的设备(显示器、打印机、扫描仪、摄像头)处理颜色的方式是不同的。每种设备都有其所能支持的颜色值范围。一种设备能支持的颜色可能在其它设备中无法支持。
为 了有效的使用颜色及理解 Quartz 2D 中用于颜色及颜色空间的函数，我们需要熟悉在 Color Management Overview 文档中所使用的术语。该文档中讨论了色觉、颜色值、设备依赖及设备颜色空间、颜色匹配问题、再现意图(rendering intent)、颜色管理模块和 ColorSync。
在本章中，我们将学习 Quartz 处理颜色和颜色空间，以及什么是 alpha 组件。本章同时也讨论如下问题：

- 创建颜色空间

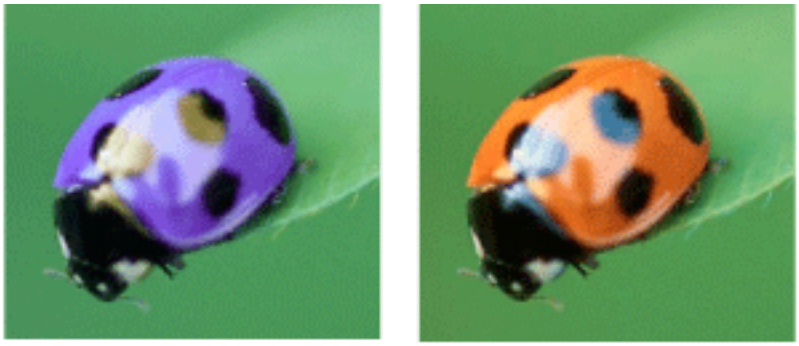
创建和设置颜色

- 设置再现意图

Table 4-1 Color values in different color spaces

Values	Color space	Components
240 degrees, 100%, 100%	HSB	Hue, saturation, brightness
0, 0, 1	RGB	Red, green, blue
1, 1, 0, 0	CMYK	Cyan, magenta, yellow, black
1, 0, 0	BGR	Blue, green, red

如果我们使用了错误的颜色空间，我们可能会获得完全不同的颜色，如图 4-1 所示。

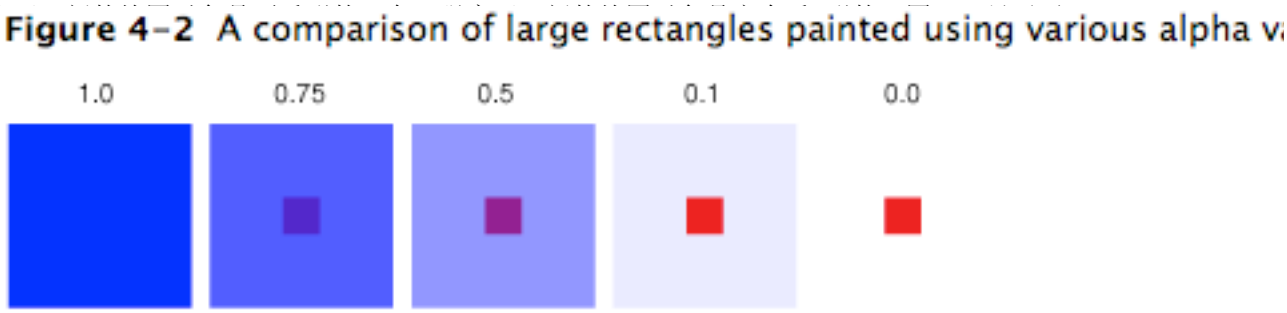


BGR color profile　　RGB color profile

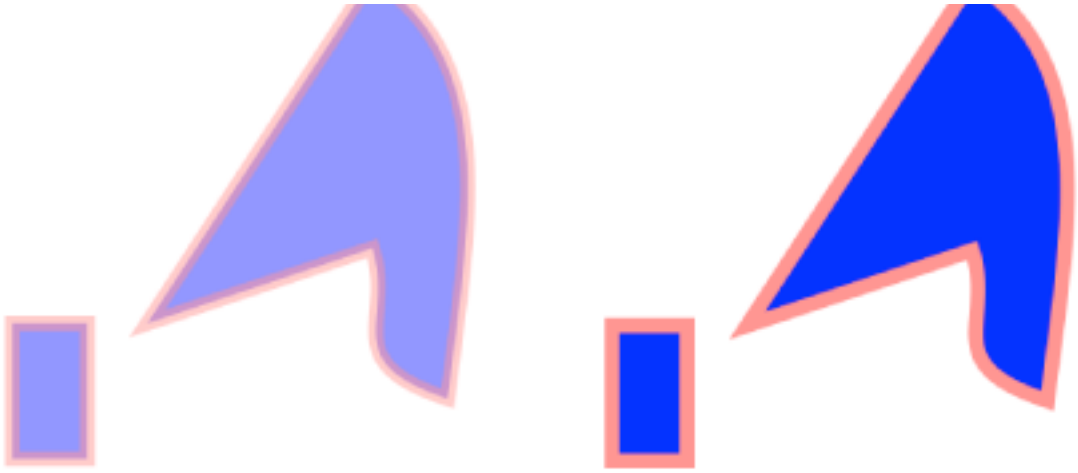
颜 色空间可以有不同数量的组件。表 4-1 中的颜色空间中其中三个只有三个组件，而 CMYK 有四个组件。值的范围与颜色空间有关。对大部分颜色空间来说，颜色 值范围为[0.0, 1.0]，1.0 表示全亮度。例如，全亮度蓝色值在 Quartz 的 RGB 颜色空间中的值是(0, 0, 1.0)。在 Quartz 中，颜色值同样有一个 alpha 值来表示透明度。在表 4-1 中没有列出该值。

alpha 值

alpha 值是图形状态参数，Quartz 用它来确定新的绘图对象如何与已存在的对象混合。在全强



我们可以将两个对象绘制到 page 上，而 page 可以在渲染前通过设置全局的 graphics context 来设置自己的透明度。图 4-3 显示了将全局的透明度设置为 0.5 和 1.0 的效果。



在标准混合模式(图形状态的默认模式)下，Quartz 使用下面的公式来混合源颜色和目标颜色的组件：
复制代码

```
1.  destination = (alpha * source) + (1 - alpha) * destination
```

其中源颜色是新绘制的颜色，目标颜色是背景颜色。该公式可用于新绘制的形状和图像。
对 于对象透明度来说，alpha 值为 1.0 时表示新对象是完全不透明的，值 0.0 表示新对象是完全透明的。0.0 与 1.0 之间的值指定对象的透明度。我们可 以为所有接受颜色的程序指定一个 alpha 值作为颜色值的最后一个组件。同样也可以使用 CGContextSetAlpha 函数来指定全局的 alpha 值。记住，如果同时设置以上两个值，Quartz 将混合全局 alpha 值与对象的 alpha 值。为了让 page 完全透明，我们可以调用 CGContextClearRect 函数来清除图形上下文(graphics context)的 alpha 通道。例如，我们可以在给图标创建一个透明遮罩或者使窗口的背景透明时，采用这种方法。

创建颜色空间

Quartz 支持颜色管理系统使用的标准颜色空间，也支持通用的颜色空间、索引颜色空间和模式(pattern)颜色空间。设备颜色空间以一种简便的方法在不同设备间 表示颜色。它用于在两种不同设备间的本地颜色空间转换颜色数据。设备依赖颜色空间的颜色在不同设备上显示时效果是一样的，它扩展了设备的能力。基于此，设 备依赖颜色空间是显示颜色时最好的选择。
如果应用程序有精确的颜色表示需求，则应该总是使用设备依赖颜色空间。通用颜色空间(generic color space)是一种常用的设备依赖颜色空间。通用颜色空间通过操作系统为我们的应用程序提供最好的颜色空间。它能使在显示器上与在打印机上打印效果是一样的。
引用
重要：IOS 不支持设备依赖颜色空间或通用颜色空间。IOS 应用程序必须使用设备颜色空间(device color space)。

创建设备依赖颜色空间

为了创建设备依赖颜色空间，我们需要给 Quartz 提供白色参考点，黑色参考点及特殊设备的 gamma 值。Quartz 使用这些信息将源颜色空间的颜色值转化为输出设备颜色空间的颜色值。
Quartz 支持设备依赖颜色空间，创建此空间的函数如下：

- L*a*b 是非线性转换，它属于 Munsell 颜色符号系统(该系统使用色度、值、饱和度来指定颜色)。 L 组件表示亮度值，a 组件表示绿色与红色之间的值，b 组件表示蓝色与黄色之间的值。该颜色空间设计用于模拟人脑解码颜色。使用函数 CGColorSpaceCreateLab 来创建。
- ICC 颜色空间是由 ICC(由国际色彩聪明，International Color C*****ortium)颜色配置而来的。ICC 颜色配置了设备支持的颜色域，该颜色域与其它设备属性相符，所以该信息可被用于将一个设备的颜色空间精 确地转换为另一个设备的颜色空间。大多数设备制造商都支持 ICC 配置。一些彩色显示器和打印机都内嵌了 ICC 信息，用于处理诸如 TIFF 的位图格式。使用 函数 CGColorSpaceCreateICCBased 来创建。
- 标准化 RGB 是设备依赖的 RGB 颜色空间，它表示相对于白色参考点(设备可生成的最白的颜色)的颜色。 使用函数 CGColorSpaceCreateCalibratedRGB 来创建。
- 标准化灰度是设备依赖的灰度颜色空间，它表示相对于白色参考点(设备可生成的最白的颜色)的颜色。 使用函数 CGColorSpaceCreateCalibratedGray 来创建。

创建通用颜色空间

通用颜色空间的颜色与系统匹配。大部分情况下，结果是可接受的。就像名字所暗示的那样，每个“通用”颜色空间(generic gray, generic RGB, generic CMYK)都是一个指定的设备依赖颜色空间。
通过颜色空间非常容易使用；我们不需要提供任何参考点信息。我们使用函数 CGColorSpaceCreateWithName 来创建一个通用颜色空间，该函数可传入以下常量值：

- kCGColorSpaceGenericGray：指定通用灰度颜色空间，该颜色空间是单色的，可以指定从 0.0(纯黑)到 1.0(纯白)范围内的颜色值。
- kCGColorSpaceGenericRGB：指定通用 RGB 颜色空间，该颜色空间中的颜色值由三个组件(red, green, blue)组成，主要用于彩色显示器上的像素。RGB 颜色空间中的每个组件的值范围是[0.0, 1.0]。
- kCGColorSpaceGenericCMYK：指定通用 CMYK 颜色空间，该颜色空间的颜色值由四个组件(cyan, magenta, yellow, black)，主要用于打印机。CMYK 颜色空间的每个组件的值范围是[0.0, 1.0]。

创建设备颜色空间

设 备颜色空间主要用于 IOS 应用程序，因为其它颜色空间无法在 IOS 上使用。大多数情况下，Mac OS X 应用程序应使用通用颜色空间，而不使用设备颜色空间。但是有些 Quartz 程序希望图像使用设备颜色空间。例如，如果调用 CGImageCreateWithMask 函数来指定一个图像作为遮罩，图像必须在设备的灰度颜色空间(device gray color space)中定义。
我们可以使用以下函数来创建设备颜色空间：

- CGColorSpaceCreateDeviceGray：创建设备依赖灰度颜色空间
- CGColorSpaceCreateDeviceRGB：创建设备依赖 RGB 颜色空间
- CGColorSpaceCreateDeviceCMYK：创建设备依赖 CMYK 颜色空间

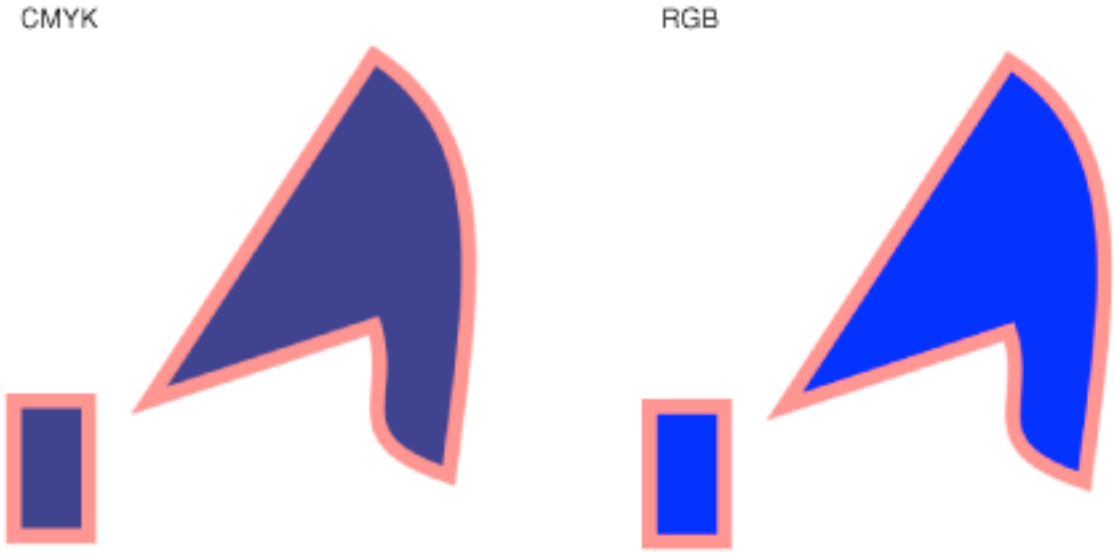
创建索引颜色空间和模式颜色空间

索引颜色空间包含一个有 256 个词目的颜色表，和词目映射到基础颜色空间。颜色表中每个词目指定一个基础颜色空间中的颜色值。使用 CGColorSpaceCreateIndexed 函数来创建。

模式颜色空间在绘制模式时使用。 使用 CGColorSpaceCreatePattern 函数来创建。

设置和创建颜色

Figure 4-4 A CMYK fill color and an RGB stroke color



我们可以使用CGContextSetFillColorSpace和CGContextSetStrokeColorSpace函数来设置填充和线框颜色空间，或者可以使用以下便利函数来设置设备颜色空间的颜色值。

Table 4-2 Color-setting functi*****

函数	用途
CGContextSetRGBStrokeColor CGContextSetRGBFillColor	设备 RGB。在生成 PDF 时，Quartz 像在相应的通用颜色空间中一样写入颜色。
CGContextSetCMYKStrokeColor CGContextSetCMYKFillColor	设备 CMYK。在生成 PDF 时，保持设备 CMYK
CGContextSetGrayStrokeColor CGContextSetGrayFillColor	设备灰度。在生成 PDF 时，Quartz 像在相应的通用颜色空间中一样写入颜色。
CGContextSetStrokeColorWithColor CGContextSetFillColorWithColor	任何颜色空间；提供一个指定颜色空间的 CGColor 对象。
CGContextSetStrokeColor	当前颜色空间。不推荐使用。更多时候我们使用 CGColor

CGContextSetFillColor 对象和函数 CGContextSetStrokeColorWithColor 和 CGContextSetFillColorWithColor。

我们在填充及线框颜色空间中指定填充及线框颜色值。例如，在 RGB 颜色空间中，我们使用数组(1.0, 0.0, 0.0, 1.0)来表示红色。前三个值指定红色值为全强度，而绿色和蓝色为零强度。第四个值为 alpha 值，用于指定颜色的透明度。

如 果需要在程序中重复使用颜色，最有效的方法是通过设置填充色和线框色来创建一个 CGColor 对象，然后将该对象传递给函数 CGContextSetFillColorWithColor 及 CGContextSetStrokeColorWithColor。我们可以按需要保 持 CGColor 对象，并可以直接使用该对象来改进应用程序的显示。

我们可以调用 CGColorCreate 函数来创建 CGColor 对象，该函数需要两个参数：CGColorspace 对象及颜色值数组。数组的最后一个值指定 alpha 值。

设置再现意图(Rending Intent)

“再 现意图”用于指定如何将源颜色空间的颜色映射到图形上下文的目标颜色空间的颜色范围内。如果不显示指定再现意图，Quartz 使用相对色度再现意图 (relative colorimetric rendering intent)应用于所有绘制(不包含位图图像)。对于位图图像，Quartz 默认使用感知(perceptual)再现意图。

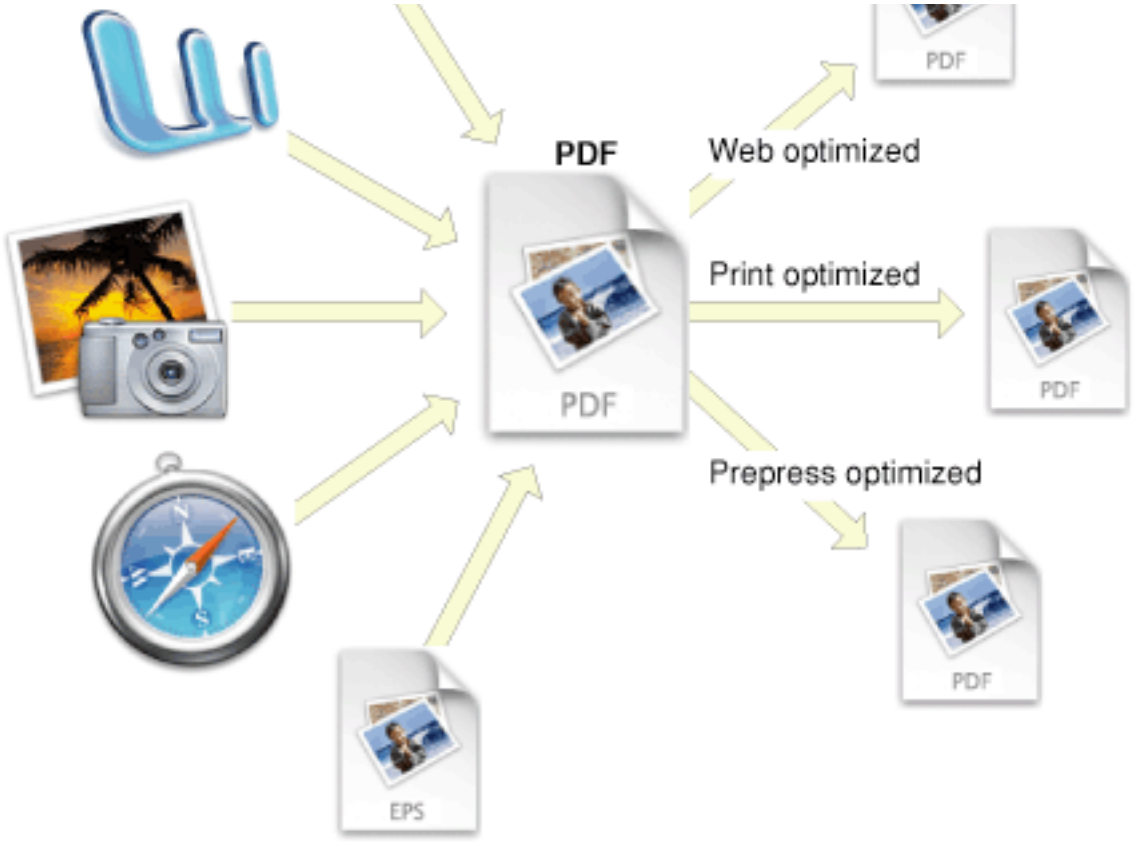
我们可以调用 CGContextSetRenderingIntent 函数来设置再现意图，并传递图形上下文(graphics context)及下例常量作为参数：

- kCGRenderingIntentDefault：使用默认的渲染意图。
- kCGRenderingIntentAbsoluteColorimetric： 绝对色度渲染意图。将输出设备颜色域外的颜色映射为输出设备域内与之最接近的颜色。这可以产生一个裁减效果，因为色域外的两个不同的颜色值可能被映射为色 域内的同一个颜色值。当图形使用的颜色值同时包含在源色域及目标色域内时，这种方法是最好的。常用于 logo 或者使用专色(spot color)时。
- kCGRenderingIntentRelativeColorimetric： 相对色度渲染意图。转换所有的颜色(包括色域内的)，以补偿图形上下文的白点与输出设备白点之间的色差。
  kCGRenderingIntentPerceptual：感知渲染意图。通过压缩图形上下文的色域来适应输出设备的色域，并保持源颜色空间的颜色之间 的相对性。感知渲染意图适用于相片及其它复杂的高细度图片。
- kCGRenderingIntentSaturation：饱和度渲染意图。把颜色转换到输出设备色域内时，保持颜色的相对饱和度。结果是包含亮度、饱和度颜色的图片。饱和度意图适用于生成低细度的图片，如描述性图表。

Quartz 2D 编程指南(13) — PDF 文档的创建、显示及转换

PDF 文档存储依赖于分辨率的向量图形、文本和位图，并用于程序的一系列指令中。一个 PDF 文档可以包含多页的图形和文本。PDF 可用于创建跨平台、只读的文档，也可用于绘制依赖于分辨率的图形。

Quartz 为所有应用程序创建高保真的 PDF 文档，这些文档保留应用的绘制操作，如图 13-1 所示。PDF 文档的结果将通过系统的其它部分或



Quartz 不仅仅只使用 PDF 作为它的数字页，它同样包含一些 API 来显示和生成 PDF 文件，及完成一些其它 PDF 相关的工作。

打开和查看 PDF

Quartz 提供了 CGPDFDocumentRef 数据类型来表示 PDF 文档。我们可以使用 CGPDFDocumentCreateWithProvider 或 CGPDFDocumentCreateWithURL 来创建 CGPDFDocument 对象。在创建 CGPDFDocument 对象后，我们可以将其绘制到图形上下文中。图 13-2 显示了在一个窗体中绘制 PDF 文档。

Figure 13-2 A PDF document

代码清单 13-1 显示了如何创建一个 CGPDFDocument 对象及获取文档的页数。
复制代码

```
1. CGPDFDocumentRefMyGetPDFDocumentRef (c*****t char
   *filename)
2. {
3. CFStringRef path;
4. CFURLRef url;
5. CGPDFDocumentRef document;
6. size_t count;
7.
8. path = CFStringCreateWithCString (NULL,
   filename,  kCFStringEncodingUTF8);
9.
10. url = CFURLCreateWithFileSystemPath (NULL,
   path,  kCFURLPOSIXPathStyle,0);   // 1 创建 CFURL
   对象
11.
```

```
12. CFRelease (path);
13.
14. document = CGPDFDocumentCreateWithURL
    (url);                              // 2 创建
    CFPDFDocument 对象
15. CFRelease(url);
16.
17. count = CGPDFDocumentGetNumberOfPages
    (document);       // 3 获取文档页数
18. if (count == 0) {
19.       printf("`%s' needs at least onepage!",
    filename);
20.       return NULL;
21. }
22.
23. return document;
24. }
```

代码清单显示了如何将一个 PDF 页绘制到图形上下文中。
复制代码

```
1.  void MyDisplayPDFPage (CGContextRefmyContext,  size_t
    pageNumber,  c*****t char *filename)
2.  {
3.  CGPDFDocumentRef document;
4.  CGPDFPageRef page;
5.
6.  document = MyGetPDFDocumentRef
    (filename);
      // 1 创建 PDFDocument 对象
7.  page = CGPDFDocumentGetPage (document,
    pageNumber);                    // 2 获取指定页的 PDF 文档
8.  CGContextDrawPDFPage (myContext,
    page);
      // 3 将 PDF 绘制到图形上下文中
9.  CGPDFDocumentRelease (document);
10. }
```

为 PDF 页创建一个转换

Quartz 提供了函数 CGPDFPageGetDrawingTransform 来创建一个仿射

变换，该变换基于将 PDF 页的 BOX 映射到指定的矩形中。函数原型是：
复制代码

```
1. CGAffineTransformCGPDFPageGetDrawingTransform (
2. CGPPageRef page,
3. CGPDFBox box,
4. CGRect rect,
5. int rotate,
6. bool preserveAspectRatio
7. );
```

该函数通过如下算法来返回一个仿射变换：
• 将在 box 参数中指定的 PDF box 的类型相关的矩形(media, crop, bleed, trim, art)与指定的 PDF 页的/MediaBox 入口求交集。相交的部分即为一个有效的矩形(effectiverectangle)。
• 将 effective rectangle 旋转参数/Rotate 入口指定的角度。
• 将得到的矩形放到 rect 参数指定的中间。
• 如果 rotate 参数是一个非零且是 90 的倍数，函数将 effective rectangel 旋转该值指定的角度。正值往右旋转；负值往左旋转。需要注意的是我们传入的是角度，而不是弧度。记住 PDF 页的/Rotate 入口也包含一个旋转，我们提供的 rotate 参数是与/Rotate 入口接合在一起的。
• 如果需要，可以缩放矩形，从而与我们提供的矩形保持一致。
• 如果我们通过传递 true 值给 preserveAspectRadio 参数以指定保持长宽比，则最后的矩形将与 rect 参数的矩形的边一致。

【注：上面这段翻译得不是很好】

例如，我们可以使用这个函数来创建一个与图 13-3 类似的 PDF 浏览程序。如果我们提供一个 Rotate Left/Rotate Right 属性，则可以调用 CGPDFPageGetDrawingTransform 来根据当前的窗体大小和旋转设置计算出适当的转换。

Figure 13-3 A PDF page rotated 90 degrees to the right

程序清单 13-3 显示了为一个 PDF 页创建及应用仿射变换，然后绘制 PDF。

复制代码

```
1. Listing 13-3    Creating an affine transform for
   aPDF page
2. void MyDrawPDFPageInRect(CGContextRef
   context,  CGPDFPageRef page,  CGPDFBox
   box,  CGRect rect,  int rotation,  bool
   preserveAspectRatio)
3. {
4. CGAffineTransform m;
5.
6. m = CGPDFPageGetDrawingTransform (page, box, rect,
   rotation,  preserveAspectRato);
```

```
7.
8.  CGContextSaveGState (context);
9.
10. CGContextConcatCTM (context, m);
11.
12. CGContextClipToRect (context,CGPDFPageGetBoxRect
    (page, box));
13.
14. CGContextDrawPDFPage (context, page);
15.
16. CGContextRestoreGState (context);
17.
18. }
```

创建 PDF 文件

使用 Quartz 创建 PDF 与绘制其它图形上下文一下简单。我们指定一个 PDF 文件地址，设置一个 PDF 图形上下文，并使用与其它图形上下文一样的绘制程序。如代码清单 13-4 所示的 MyCreatePDFFile 函数，显示了创建一个 PDF 的所有工作。注意，代码在 CGPDFContextBeginPage 和 CGPDFContextEndPage 中来绘制 PDF。我们可以传递一个 CFDictionary 对象来指定页属性，包括 media, crop, bleed, trim 和 art boxes。

Listing 13-4 Creating a PDF file

复制代码

```
1.  void MyCreatePDFFile (CGRectpageRect, c*****t char *filename)
2.  {
3.  CGContextRef pdfContext;
4.  CFStringRef path;
5.  CFURLRef url;
6.  CFData boxData = NULL;
7.  CFMutableDictionaryRef myDictionary = NULL;
8.  CFMutableDictionaryRef pageDictionary = NULL;
9.
10. path = CFStringCreateWithCString (NULL, filename,
    kCFStringEncodingUTF8);
11.
12. url = CFURLCreateWithFileSystemPath (NULL, path,
    kCFURLPOSIXPathStyle, 0);
13.
14. CFRelease (path);
15.
```

```
16. myDictionary = CFDictionaryCreateMutable(NULL,
    0,  &kCFTypeDictionaryKeyCallBacks,  &kCFTypeDictionaryValueCal
    lBacks);
17.
18. CFDictionarySetValue(myDictionary, kCGPDFContextTitle,
    CFSTR("MyPDF File"));
19.
20. CFDictionarySetValue(myDictionary, kCGPDFContextCreator,
    CFSTR("MyName"));
21.
22. pdfContext = CGPDFContextCreateWithURL (url,
    &pageRect,myDictionary);
23.
24. CFRelease(myDictionary);
25.
26. CFRelease(url);
27.
28. pageDictionary = CFDictionaryCreateMutable(NULL,
    0,  &kCFTypeDictionaryKeyCallBacks,  &kCFTypeDictionaryValueCal
    lBacks);
29.
30. boxData = CFDataCreate(NULL,(c*****t UInt8 *)&pageRect,
    sizeof(CGRect));
31.
32. CFDictionarySetValue(pageDictionary, kCGPDFContextMediaBox,
    boxData);
33.
34. CGPDFContextBeginPage (pdfContext, &pageRect);
35.
36. myDrawContent (pdfContext);
37.
38. CGPDFContextEndPage (pdfContext);
39.
40. CGContextRelease (pdfContext);
41.
42. CFRelease(pageDictionary);
43.
44. CFRelease(boxData);
45. }
```

添加链接

我们可以在 PDF 上下文中添加链接和锚点。Quartz 提供了三个函数，每个函数都以 PDF 图形上下文作为参数，还有链接的信息：

- CGPDFContextSetURLForRect 可以让我们指定在点击当前 PDF 页中的矩形时打开一个 URL。
- CGPDFContextSetDestinationForRect 指定在点击当前 PDF 页中的矩形区域时设置目标以进行跳转。我们需要提供一个目标名。
- CGPDFContextAddDestinationAtPoint 指定在点击当前 PDF 页中的一个点时设置目标以进行跳转。我们需要提供一个目标名。

保护 PDF 内容

为了保护 PDF 内容，我们可以在辅助字典中指定一些安全选项并传递给 CGPDFContextCreate。我们可以通过包含如下关键字来设置所有者密码、用户密码、PDF 是否可以被打印或拷贝：

- kCGPDFContextOwnerPassword: 定义 PDF 文 档的所有者密码。如果指定该值，则文档使用所有者密码来加密；否则文档不加密。该关键字的值必须是 ASCII 编码的 CFString 对象。只有前 32 位是 用于密码的。该值没有默认值。如果该值不能表示成 ASCII，则无法创建文档并返回 NULL。Quartz 使用 40-bit 加密。
- kCGPDFContextUserPassword: 定义 PDF 文档的用户密码。如果文档加密了，则该值是文档的用户密码。如果没有指定，则用户密码为空。该关键字的值必须是 ASCII 编码的 CFString 对象。只有前 32 位是用于密码的。如果该值不能表示成 ASCII，则无法创建文档并返回 NULL。
- kCGPDFContextAllowsPrinting:指定当使用用户密码锁定时文档是否可以打印。该值必须是 CFBoolean 对象。默认值是 kCGBooleanTrue。
- kCGPDFContextAllowsCopying: 指定当使用用户密码锁定时文档是否可以拷贝。该值必须是 CFBoolean 对象。默认值是 kCGBooleanTrue。

代码清单 14-4(下一章)显示了确认 PDF 文档是否被锁定，及用密码打开文档。