Ios 视频采集

作者 Arthur.liu

[AVFoundation 3](#_Toc526783176)

[AVCaptureVideoDataOutputSampleBufferDelegate 4](#_Toc526783177)

[AVCaptureSession 4](#_Toc526783178)

[1. 创建AVCaptureSession 4](#_Toc526783179)

[2 给Session添加input输入 4](#_Toc526783180)

[3. 给session添加output输出 5](#_Toc526783181)

[AVCaptureInput 6](#_Toc526783182)

[AVCaptureOutput 7](#_Toc526783183)

[AVCaptureDevice 7](#_Toc526783184)

[AVCaptureVideoPreviewLayer 9](#_Toc526783185)

[CMTimeMake 9](#_Toc526783186)

[CMTime的创建 9](#_Toc526783187)

[CMTime计算 9](#_Toc526783188)

[AVCaptureVideoDataOutput 10](#_Toc526783189)

[AVCaptureConnection 10](#_Toc526783190)

[CMSampleBufferRef 10](#_Toc526783191)

[CVImageBufferRef 12](#_Toc526783192)

[CIImage 12](#_Toc526783193)

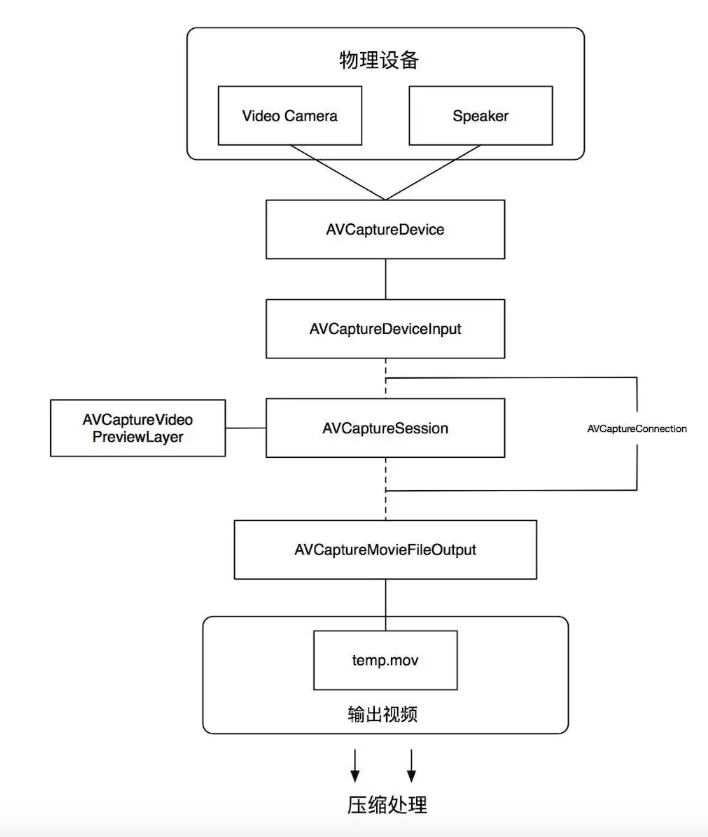
[dispatch\_async 12](#_Toc526783194)

[dispatch\_queue\_t 13](#_Toc526783195)

[CALayer 14](#_Toc526783196)

部分内容参考<https://www.jianshu.com/p/24828a14570d>

# AVFoundation



AVFoundation中提供了很多现成的播放器和录音机，但是事实上它还有更加底层的内容可以供开发者使用。因为AVFoundation中抽了很多和底层输入、输出设备打交道的类，依靠这些类开发人员面对的不再是封装好的音频播放器AVAudioPlayer、录音机（AVAudioRecorder）、视频（包括音频）播放器AVPlayer，而是输入设备（例如麦克风、摄像头）、输出设备（图片、视频）等。

# AVCaptureVideoDataOutputSampleBufferDelegate

# AVCaptureSession

## 1. 创建AVCaptureSession

设置SessionPreset,用于设置output输出流的bitrate或者说画面质量

// 1 创建session

AVCaptureSession \*session = [AVCaptureSession new];

//设置session显示分辨率

if ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPhone)

[session setSessionPreset:AVCaptureSessionPreset640x480];

else

[session setSessionPreset:AVCaptureSessionPresetPhoto];

## 2 给Session添加input输入

一般是Video或者Audio数据,也可以两者都添加,即AVCaptureSession的输入源AVCaptureDeviceInput.

// 2 获取摄像头device,并且默认使用的后置摄像头,并且将摄像头加入到captureSession中

AVCaptureDevice \*device = [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];

AVCaptureDeviceInput \*deviceInput = [AVCaptureDeviceInput deviceInputWithDevice:device error:&error];

isUsingFrontFacingCamera = NO;

if ([session canAddInput:deviceInput]){

[session addInput:deviceInput];

}

## 3. 给session添加output输出

添加AVCaptureOutput,即AVCaptureSession的输出源.一般输出源分成:音视频源,图片源,文件源等.

* 音视频输出AVCaptureAudioDataOutput,AVCaptureVideoDataOutput.
* 静态图片输出AVCaptureStillImageOutput(iOS10中被AVCapturePhotoOutput取代了)
* AVCaptureMovieFileOutput表示文件源.

通常如果需要音视频帧,需要在将output加入到session之前,设置videoSetting或者audioSetting,主要是音视频的格式或者回调的delegate以及dispatch queue.

// 4 创建拍照使用的AVCaptureStillImageOutput,并且注册observer观察capturingStillImage,并将output加入到session. 使用observer的作用监控"capturingStillImage",如果为YES,那么表示开始截取视频帧.在回调方法中显示闪屏效果

stillImageOutput = [AVCaptureStillImageOutput new];

[stillImageOutput addObserver:self forKeyPath:@"capturingStillImage" options:NSKeyValueObservingOptionNew context:(\_\_bridge void \* \_Nullable)(AVCaptureStillImageIsCapturingStillImageContext)];

if ([session canAddOutput:stillImageOutput]){

[session addOutput:stillImageOutput];

}

// 5 创建预览output,设置预览videosetting,然后设置预览delegate使用的回调线程,将该预览output加入到session

videoDataOutput = [AVCaptureVideoDataOutput new];

// we want BGRA, both CoreGraphics and OpenGL work well with 'BGRA'

NSDictionary \*rgbOutputSettings = [NSDictionary dictionaryWithObject:

[NSNumber numberWithInt:kCMPixelFormat\_32BGRA] forKey:(id)kCVPixelBufferPixelFormatTypeKey];

[videoDataOutput setVideoSettings:rgbOutputSettings];

[videoDataOutput setAlwaysDiscardsLateVideoFrames:YES]; // discard if the data output queue is blocked (as we process the still image)

// create a serial dispatch queue used for the sample buffer delegate as well as when a still image is captured

// a serial dispatch queue must be used to guarantee that video frames will be delivered in order

// see the header doc for setSampleBufferDelegate:queue: for more information

videoDataOutputQueue = dispatch\_queue\_create("VideoDataOutputQueue", DISPATCH\_QUEUE\_SERIAL);

[videoDataOutput setSampleBufferDelegate:self queue:videoDataOutputQueue];

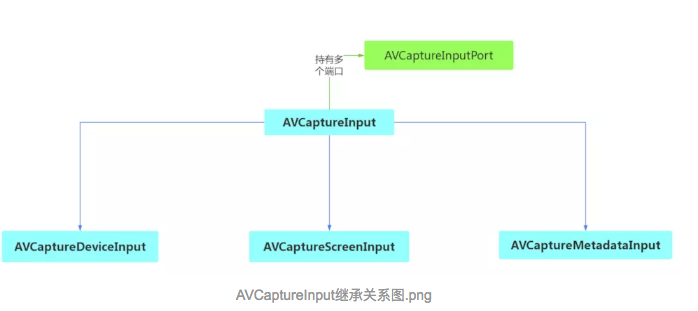
if ([session canAddOutput:videoDataOutput]){

[session addOutput:videoDataOutput];

}

# AVCaptureInput

由于 AVCaptureInput 是个抽象类，无法直接使用，所以我们一般使用它的子类类管理输入数据。我们常用的AVCaptureInput的子类有三个：



AVCaptureInput继承关系图.png

* AVCaptureDeviceInput：用于从AVCaptureDevice对象捕获数据。
* AVCaptureScreenInput：从macOS屏幕上录制的一种捕获输入。
* AVCaptureMetadataInput：它为AVCaptureSession提供AVMetadataItems。AVCaptureMetadataInputs呈现了一个且只有一个AVCaptureInputPort，它目前可能只连接到AVCaptureMovieFileOutput。通过输入端口提供的元数据由客户机提供，并且必须符合客户机提供的CMFormatDescription。AVMetadataItems在AVTimedMetadataGroup中提供。

# AVCaptureOutput

在ios中，分为MovieFile、VideoData、AudioData和StillImage几种output，使用方式类似，只是范围不同。另外，它们都继承于AVCaptureOutput。 第一个是输出成movie文件，第二个适用于逐个Frame的处理，第三个适用于声音采集，第四个是still image(静态图像)相关。

# AVCaptureDevice

AVCaptureDevice.h 主要用来获取iPhone一些关于相机设备的属性。  
必须引用一个系统包：#import

1.前置和后置摄像头  
typedef NS\_ENUM(NSInteger, AVCaptureDevicePosition) {  
AVCaptureDevicePositionUnspecified = 0,  
AVCaptureDevicePositionBack = 1,  
AVCaptureDevicePositionFront = 2  
} NS\_AVAILABLE(10\_7, 4\_0);

2.闪光灯开关  
typedef NS\_ENUM(NSInteger, AVCaptureFlashMode) {  
AVCaptureFlashModeOff = 0,  
AVCaptureFlashModeOn = 1,  
AVCaptureFlashModeAuto = 2

} NS\_AVAILABLE(10\_7, 4\_0);

3.手电筒开关--其实就是相机的闪光灯  
typedef NS\_ENUM(NSInteger, AVCaptureTorchMode) {  
AVCaptureTorchModeOff = 0,  
AVCaptureTorchModeOn = 1,  
AVCaptureTorchModeAuto = 2,

} NS\_AVAILABLE(10\_7, 4\_0);

4.焦距模式调整  
typedef NS\_ENUM(NSInteger, AVCaptureFocusMode) {  
AVCaptureFocusModeLocked = 0,  
AVCaptureFocusModeAutoFocus = 1,  
AVCaptureFocusModeContinuousAutoFocus = 2,

} NS\_AVAILABLE(10\_7, 4\_0);

5.曝光量调节  
typedef NS\_ENUM(NSInteger, AVCaptureExposureMode) {  
AVCaptureExposureModeLocked = 0,  
AVCaptureExposureModeAutoExpose = 1,  
AVCaptureExposureModeContinuousAutoExposure = 2,  
AVCaptureExposureModeCustom NS\_ENUM\_AVAILABLE\_IOS(8\_0) = 3,

} NS\_AVAILABLE(10\_7, 4\_0);

6.白平衡  
typedef NS\_ENUM(NSInteger, AVCaptureWhiteBalanceMode) {  
AVCaptureWhiteBalanceModeLocked = 0,  
AVCaptureWhiteBalanceModeAutoWhiteBalance = 1,  
AVCaptureWhiteBalanceModeContinuousAutoWhiteBalance = 2,

} NS\_AVAILABLE(10\_7, 4\_0);

7.距离调整  
typedef NS\_ENUM(NSInteger, AVCaptureAutoFocusRangeRestriction) {  
AVCaptureAutoFocusRangeRestrictionNone = 0,  
AVCaptureAutoFocusRangeRestrictionNear = 1,  
AVCaptureAutoFocusRangeRestrictionFar = 2,

} NS\_AVAILABLE\_IOS(7\_0);

作者：Smallwolf\_JS  
链接：https://www.jianshu.com/p/10902fb27638  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

# AVCaptureVideoPreviewLayer

相机拍摄预览图层，是CALayer的子类，使用该对象可以实时查看拍照或视频录制效果，创建该对象需要指定对应的AVCaptureSession对象。

# CMTimeMake

CMTime一般用的不多，但是在时间精度有要求的应用中会用到，比如音频，视频的处理。

## CMTime的创建

我们一般使用CMTimeMake函数创建，如下：

CMTime t1 = CMTimeMake(3, 1);

CMTime t2 = CMTimeMake(6, 3);

我们也可以通过CMTimeShow函数打印CMTime的相关信息如：

CMTimeShow(t1);

CMTimeShow(t2);

结果如：  
{3/1 = 3.000}  
{6/3 = 2.000}

## CMTime计算

相加

CMTime t3 = CMTimeAdd(t1, t2);

想减

CMTime t4 = CMTimeSubtract(t3, t1);

我们也可以通过CMTimeGetSeconds函数获取时间的秒数。

作者：jiangamh  
链接：https://www.jianshu.com/p/f110bf6e0471  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

# AVCaptureVideoDataOutput

参考<https://blog.csdn.net/wangrui13931182709/article/details/51682956>

# AVCaptureConnection

# CMSampleBufferRef

CMSampleBufferRef 与 UIImage 的转换

在取得  CMSampleBufferRef 之后，还必须透过一连串的转换才能够得到 UIImage，CMSampleBufferRef –> CVImageBufferRef –> CGContextRef –> CGImageRef –> UIImage，你可以将以下程序代码任意实作于上述两个内部函数中来取得连续影像片段中的 UIImage。  
//制作 CVImageBufferRef  
CVImageBufferRef buffer;  
buffer = CMSampleBufferGetImageBuffer(sampleBuffer);

CVPixelBufferLockBaseAddress(buffer, 0);

    //从 CVImageBufferRef 取得影像的细部信息  
    uint8\_t \*base;  
    size\_t width, height, bytesPerRow;  
    base = CVPixelBufferGetBaseAddress(buffer);  
    width = CVPixelBufferGetWidth(buffer);  
    height = CVPixelBufferGetHeight(buffer);  
    bytesPerRow = CVPixelBufferGetBytesPerRow(buffer);

    //利用取得影像细部信息格式化 CGContextRef  
    CGColorSpaceRef colorSpace;  
    CGContextRef cgContext;  
    colorSpace = CGColorSpaceCreateDeviceRGB();  
    cgContext = CGBitmapContextCreate(base, width, height, 8, bytesPerRow, colorSpace, kCGBitmapByteOrder32Little | kCGImageAlphaPremultipliedFirst);  
    CGColorSpaceRelease(colorSpace);

    //透过 CGImageRef 将 CGContextRef 转换成 UIImage  
    CGImageRef cgImage;  
    UIImage \*image;  
    cgImage = CGBitmapContextCreateImage(cgContext);  
    image = [UIImage imageWithCGImage:cgImage];  
    CGImageRelease(cgImage);  
    CGContextRelease(cgContext);

CVPixelBufferUnlockBaseAddress(buffer, 0);

//成功转换成 UIImage  
//[myImageView setImage:image];

最后，如果你希望改变撷取影像时的方向，则可以对内部函数中的 AVCaptureConnection 做 setVideoOrientation: 旋转影像，或 setVideoMirrored: 镜射影像。

# CVImageBufferRef

# CIImage

# dispatch\_async

常用的方法dispatch\_async

为了避免界面在处理耗时的操作时卡死，比如读取网络数据，IO,数据库读写等，我们会在另外一个线程中处理这些操作，然后通知主线程更新界面。

用GCD实现这个流程的操作比前面介绍的NSThread  NSOperation的方法都要简单。代码框架结构如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/totogo2010/article/details/8016129)

1. dispatch\_async(dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0), ^{
2. // 耗时的操作
3. dispatch\_async(dispatch\_get\_main\_queue(), ^{
4. // 更新界面
5. });
6. });

如果这样还不清晰的话，那我们还是用上两篇博客中的下载图片为例子，代码如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/totogo2010/article/details/8016129)

1. dispatch\_async(dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0), ^{
2. NSURL \* url = [NSURL URLWithString:@"http://avatar.csdn.net/2/C/D/1\_totogo2010.jpg"];
3. NSData \* data = [[NSData alloc]initWithContentsOfURL:url];
4. UIImage \*image = [[UIImage alloc]initWithData:data];
5. **if (data != nil) {**
6. dispatch\_async(dispatch\_get\_main\_queue(), ^{
7. self.imageView.image = image;
8. });
9. }
10. });

# dispatch\_queue\_t

dispatch队列的生成可以有这几种方式：

**1. dispatch\_queue\_t queue = dispatch\_queue\_create("com.dispatch.serial", DISPATCH\_QUEUE\_SERIAL);** //生成一个串行队列，队列中的block按照先进先出（FIFO）的顺序去执行，实际上为单线程执行。第一个参数是队列的名称，在调试程序时会非常有用，所有尽量不要重名了。

**2. dispatch\_queue\_t queue = dispatch\_queue\_create("com.dispatch.concurrent", DISPATCH\_QUEUE\_CONCURRENT);**//生成一个并发执行队列，block被分发到多个线程去执行

**3. dispatch\_queue\_t queue = dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0);** //获得程序进程缺省产生的并发队列，可设定优先级来选择高、中、低三个优先级队列。由于是系统默认生成的，所以无法调用dispatch\_resume()和dispatch\_suspend()来控制执行继续或中断。需要注意的是，三个队列不代表三个线程，可能会有更多的线程。并发队列可以根据实际情况来自动产生合理的线程数，也可理解为dispatch队列实现了一个线程池的管理，对于程序逻辑是透明的。

官网文档解释说共有三个并发队列，但实际还有一个更低优先级的队列，设置优先级为**DISPATCH\_QUEUE\_PRIORITY\_BACKGROUND**。Xcode调试时可以观察到正在使用的各个dispatch队列。

**4. dispatch\_queue\_t queue = dispatch\_get\_main\_queue();** //获得主线程的dispatch队列，实际是一个串行队列。同样无法控制主线程dispatch队列的执行继续或中断。

# CALayer

|  |  |
| --- | --- |
| layer | 英[ˈleɪə(r)] |
| 参考<https://www.cnblogs.com/monicaios/p/3519936.html>  一、CALayer简介  Core Animation是跨平台的，支持iOS环境和Mac OS X环境  凡是支持跨平台的框架，都不能直接使用UIKit框架，因为UIKit框架只能应用在iOS而不能用于Mac  需要先理解CALayer，因为核心动画操作的对象是CALayer，而不是UIView  CALayer是核心动画的基础，可以做圆角、阴影、边框等效果  每个UIView内部都有一个Layer的属性  在实现核心动画时，本质上是将CALayer中的内容转换成位图（一种图像格式），从而便于图形硬件的操纵  二、UIView的CALayer基本属性  设置UIView中的CALayer属性  圆角、边框、阴影及3D形变属性  在UIView中CALayer只是一个类声明，需要添加QuartzCore框架  在使用颜色时，不能直接使用UIColor而需要将颜色转成CGColor  修改图层相当于修改UIView属性，即修改了界面属性  要设置阴影，需要同时指定阴影的偏移尺寸、颜色和透明度  形变属性既可以用形变函数指定，也可以用keyPath指定  三、UIImageView的CALayer基本属性  设置UIImageView中的CALayer属性  圆角、边框、阴影  UIImageView中不仅一个子图层，因此设置圆角时需要使用setMasksToBounds:YES，让所有子图层跟随边框，不过设置该属性后，无法使用阴影效果  可以在底层附加一个UIView实现阴影效果  设置UIImageView中的CALayer属性  transform属性可以调整CALayer的形变，其中包括：旋转、缩放、平移  transform属性的参数查询：CATransform  四、图层和视图之间的关系  创建视图对象时，视图会自己创建一个层，视图在绘图（如drawRect:）时，会将内容画在自己的层上。当视图在层上完成绘图后，系统会将图层拷贝至屏幕（CALayer绘图的上下文是图像，整个画完后，才显示，提前绘制提高性能以及用户体验）。每个视图都有一个层，每个图层又可以有多个子层  Layer的设计目的不是为了取代视图，不能基于CALayer创建一个独立的可视化组件  Layer的设计目的是提供视图的基本可视内容，以便提高动画的执行效率  除提供可视内容外，Layer不负责视图的事件响应等工作，同时Layer不能参与到响应者链条中  五、CALayer的使用说明  通过UIView的layer属性可以拿到对应的根层，这个层不允许重新创建，但可以往层里面添加子层(调用CALayer的addSublayer)  要具体使用CALayer，需要引入<QuartzCore/QuartzCore.h>  获取当前图层或使用静态方法layer初始化CALayer后，可以设置以下属性  bounds：宽度和高度  position：位置(默认指中心点，具体由anchorPoint决定)  anchorPoint：锚点（x,y的范围都是0-1），决定了position的含义  backgroundColor: 背景颜色(CGColorRef类型)  borderColor：边框颜色(CGColorRef类型)  borderWidth：边框宽度  cornerRadius：圆角半径  contents: 内容（比如设置为图片CGImageRef）  虽然CALayer可以使用frame，但最好还是使用bounds和position。为层设置动画时，用bounds和position会方便一点  六、创建自定义图层  创建自定义图层，并设置以下属性：  bounds：宽度和高度  backgroundColor: 背景颜色(CGColorRef类型)  position：位置（默认指中点，具体由anchorPoint决定）  anchorPoint：锚点（x,y的范围都是0-1），决定了position的含义  contents：内容CGImageRef  锚点和位置的关系，以及在旋转转换时对图层的影响  UIView有一个addSubview方法，而layer有一个addSubLayer方法  锚点在游戏开发中使用比较频繁，而在应用开发中极少使用  七、CALayer中图像及颜色的注意事项  CALayer中使用CGColorRef和CGImageRef的数据类型，而不用UIColor和UIImage  QuartzCore（包含CALayer类）和Core Graphics（包含CGImageRef、CGColorRef）框架都能在iOS和Mac OS X上使用，但是UIKit（包含UIImage和其他UI开头的类）只能在iOS中使用  为了保证可移植性，QuartzCore不能使用UIImage，只能使用CGImageRef  不过很多情况下，可以通过UIKit对象的特定方法，可以得到Core Graphics对象，如UIImage的CGImage方法和UIColor的CGColor方法  八、CALayer的隐式动画属性  每一个UIView内部都默认关联着一个CALayer，称这个Layer为Root Layer。所有的非Root Layer都存在着隐式动画，隐式动画的默认时长为1/4秒。  当修改非Root Layer的部分属性时，相应的修改会自动产生动画效果，能执行隐式动画的属性被称为“可动画属性”，诸如:  bounds: 缩放动画  position: 平移动画  opacity: 淡入淡出动画（改变透明度）  在文档中搜素animatable可以找到所有可动画属性  如果要关闭默认的动画效果，可以通过动画事务方法实现：  [CATransaction begin];  [CATransaction setDisableActions:YES];  [CATransaction commit];  九、在CALayer上绘图  要在CALayer上绘图，有两种方法：  创建一个CALayer的子类，然后覆盖drawInContext:方法，可以使用Quartz2D API在其中进行绘图  设置CALayer的delegate，然后让delegate实现drawLayer:inContext:方法进行绘图  不能再将UIView设置为这个CALayer的delegate，因为UIView对象已经是内部层的delegate，再次设置会出问题  无论使用哪种方法，都必须向层发送setNeedsDisplay消息，以触发相应绘图方法的调用  十、CALayer、UIView以及上下文之间的关系  当UIView收到setNeedsDisplay消息时，CALayer会准备好一个CGContextRef，然后向它的delegate即UIView，发送消息，并且传入已经准备好的CGContextRef对象。UIView在drawLayer:inContext:方法中会调用自己的drawRect:方法  平时在drawRect:中通过UIGraphicsGetCurrentContext()获取的就是由CALayer传入的CGContextRef对象，在drawRect:中完成的所有绘图都会填入CALayer的CGContextRef中，然后被拷贝至屏幕  CALayer的CGContextRef用的是位图上下文（Bitmap Graphics Context）  十一、获取CALayer中的内容——截屏  UIGraphicsBeginImageContext(self.view.bounds.size);  [self.view.layer renderInContext:UIGraphicsGetCurrentContext()];  UIImage \*image = UIGraphicsGetImageFromCurrentImageContext();  UIGraphicsEndImageContext();  说明：使用renderInContext方法可以将指定图层及其子图层中的内容渲染输出到指定的上下文中 |  |