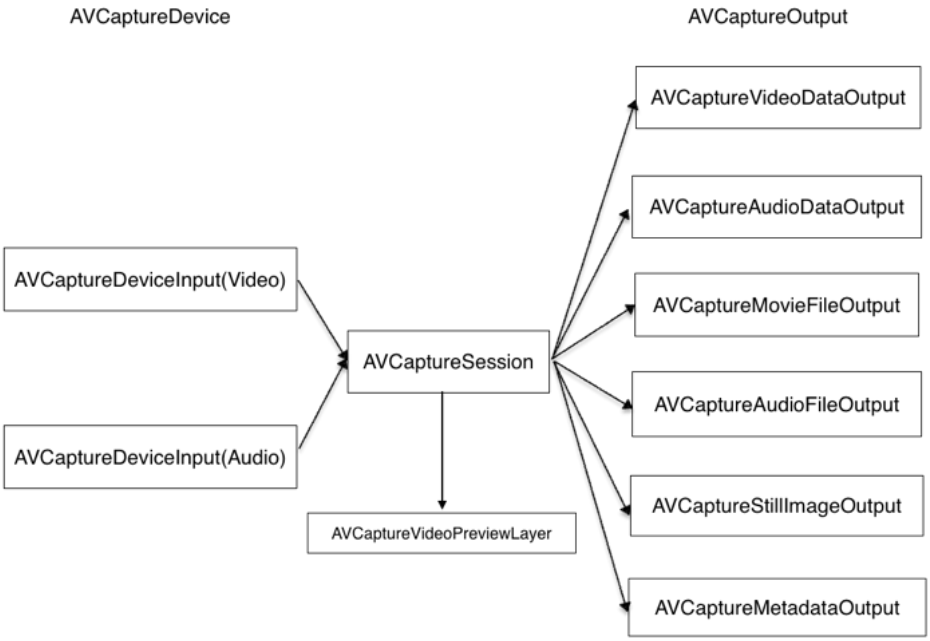
iOS-AVFoundation自定义相机详解

cdcyd (/u/2841ca1b2535) + 关注
2016.08.30 15:18* 字数 4561 阅读 7650 评论 21 喜欢 77 赞赏 1
(/u/2841ca1b2535)



AVFoundation 中关于视频主要的类

- 目录
  - 相机基本实现步骤
  - 捕捉会话——AVCaptureSession
  - 捕捉输入——AVCaptureDeviceInput
  - 捕捉预览——AVCaptureVideoPreviewLayer/OpenGL ES
  - 捕捉连接——AVCaptureConnection
  - 拍照——AVCaptureStillImageOutput
  - 音频——AVCaptureAudioDataOutput
  - 视频——AVCaptureVideoDataOutput
  - 生成视频文件——AVAssetWriter、AVAssetWriterInput
  - 写入相册——ALAssetsLibrary、PHPhotoLibrary
  - 操作相机
    - 转换摄像头
    - 补光
    - 闪光灯
    - 聚焦

        - 曝光
        - 自动聚焦/曝光
    - 视频重力——Video gravity
    - 方向问题——Orientation

- 项目地址

- **先看相机实现步骤，下面是对每一步需要做的事情详解**

> 1.创建session(捕捉会话)
> 2.创建device input(捕捉设备输入)
> 3.预览view
> 4.创建capture output(捕捉的输出)
> 5.拍照、录视频(元数据转成图片或文件)

- **捕捉会话——AVCaptureSession**

AVCaptureSession(捕捉会话管理)：它从物理设备得到数据流（比如摄像头和麦克风），输出到一个或多个目的地，它可以通过会话预设值(session preset)，来控制捕捉数据的格式和质量
下面是创建一个 session 的代码：

```
AVCaptureSession *captureSession = [[AVCaptureSession alloc]init];
[captureSession setSessionPreset:AVCaptureSessionPresetPhoto];
```

SessionPreset在iOS中大概有11个

> ```
> NSString *const AVCaptureSessionPresetPhoto;
> NSString *const AVCaptureSessionPresetHigh;
> NSString *const AVCaptureSessionPresetMedium;
> NSString *const AVCaptureSessionPresetLow;
> NSString *const AVCaptureSessionPreset352x288;
> NSString *const AVCaptureSessionPreset640x480;
> NSString *const AVCaptureSessionPreset1280x720;
> NSString *const AVCaptureSessionPreset1920x1080;
> NSString *const AVCaptureSessionPresetiFrame960x540;
> NSString *const AVCaptureSessionPresetiFrame1280x720;
> NSString *const AVCaptureSessionPresetInputPriority;
> ```
>
> 第一个代表高像素图片输出；接下来三种为相对预设(low, medium, high)，这些预设的编码配置会因设备不同而不同，如果选择high，那么你选定的相机会提供给你该设备所能支持的最高画质；再后面就是特定分辨率的预设(352x288 VGA, 1920x1080 VGA, 1280x720 VGA, 640x480 VGA, 960x540 iFrame, 1280x720 iFrame)；最后一个代表 capture session 不去控制音频与视频输出设置，而是通过已连接的捕获设备的 activeFormat 来反过来控制 capture session 的输出质量等级

注意：所有对 capture session 的调用都是阻塞的，因此建议将它们分配到后台串行队列中，不过这里为了简单，不考虑性能，所以省略了dispatch queue

- **捕捉输入——AVCaptureDeviceInput**

AVCaptureDeviceInput(捕捉设备)：它实际上是为摄像头和麦克风等物理设备定义的接口，我们可以通过它来访问或控制这些硬件设备。比如控制摄像头的对焦、曝光等。

```
  /**
    该方法会返回当前能够输入视频的全部设备，包括前后摄像头和外接设备
    NSArray *videoDevices = [AVCaptureDevice devicesWithMediaType:AVMediaTypeV
ideo];

    该方法会返回当前能够输入音频的全部设备
    NSArray *videoDevices = [AVCaptureDevice devicesWithMediaType:AVMediaTypeA
udio];
    */

  // 获取视频输入设备，该方法默认返回iPhone的后置摄像头
  AVCaptureDevice *videoDevice = [AVCaptureDevice defaultDeviceWithMediaType:A
VMediaTypeVideo];
  // 将捕捉设备加入到捕捉会话中
  AVCaptureDeviceInput *videoInput = [AVCaptureDeviceInput deviceInputWithDevi
ce:videoDevice error:error];
  if (videoInput) {
      if ([_captureSession canAddInput:videoInput]){
          [_captureSession addInput:videoInput];
      }
  }

  // 音频输入
  AVCaptureDevice *audioDevice = [AVCaptureDevice defaultDeviceWithMediaType:A
VMediaTypeAudio];
  AVCaptureDeviceInput *audioIn = [[AVCaptureDeviceInput alloc] initWithDevice
:audioDevice error:error];
  if ([_captureSession canAddInput:audioIn]){
      [_captureSession addInput:audioIn];
  }
```

- **捕捉预览——AVCaptureVideoPreviewLayer/OpenGL ES**

AVCaptureVideoPreviewLayer(捕捉预览)：它是CALayer的子类，可被用于自动显示相机产生的实时图像。previewLayer支持视频重力概念，可以控制视频内容渲染的缩放和拉效果(关于视频重力，将在后面进行详解)

```
// 创建一个previewLayer
AVCaptureVideoPreviewLayer  *previewLayer = [[AVCaptureVideoPreviewLayer alloc
] initpWithFrame:self.view.bounds]
[previewLayer.layer setVideoGravity:AVLayerVideoGravityResizeAspect];
[previewLayer.layer setSession:session];

// 将屏幕坐标系的点转换为previewLayer坐标系的点
- (CGPoint)captureDevicePointForPoint:(CGPoint)point {
  return [previewLayer.layer captureDevicePointOfInterestForPoint:point];
}
```

> 注意：
>
> 1. 它看起来有点像输出，但其实不是，它仅用来预览摄像头捕捉的画面。真正用于输出的是AVCaptureSession（previewLayer拥有session，session拥有outputs)；
> 2. 它的坐标系和屏幕的坐标系不同，如果点击某区域实现对焦时，我们需要将设备的坐标系转换为实时预览图的坐标；
> 3. 它的坐标原点永远都在右上角，这和我们手机的坐标系不同，手机坐标系的原点是不变的。因此拍照或录制视频时，要先得到设备方向(关于方向问题，后面会详解)，计算输出的旋转角度。

捕捉预览除了用AVCaptureVideoPreviewLayer外，还可以用OpenGL ES绘制，我们可以从输出数据流捕捉单一的图像帧，并使用 OpenGL ES手动地把它们显示在 view 上。如果我们想对预览视图进行操作，如使用滤镜，我们就必须这样做。这里不做深入研究，下面给出一段简单的实现代码：

```
  // 创建glview
  EAGLContext *context = [[EAGLContext alloc]initWithAPI:kEAGLRenderingAPIOpen
GLES2];
  GLKView *glView = [[GLKView alloc]initWithFrame:self.view.bounds context:con
text];
  [EAGLContext setCurrentContext:context];
  [self.view addSubview:glView];
  glView.transform = CGAffineTransformMakeRotation(M_PI_2);
  glView.frame = [UIApplication sharedApplication].keyWindow.bounds;

  // 在视频输出函数中绘制出来
-(void)captureOutput:(AVCaptureOutput *)captureOutput didOutputSampleBuffer:(C
MSampleBufferRef)sampleBuffer fromConnection:(AVCaptureConnection *)connection
{
  if (_glview.context != [EAGLContext currentContext]) {
      [EAGLContext setCurrentContext:_glview.context];
  }
  CVImageBufferRef imageRef = CMSampleBufferGetImageBuffer(sampleBuffer);
  CIImage *image = [CIImage imageWithCVImageBuffer:imageRef];
  [_glview bindDrawable];
  [_cicontext drawImage:image inRect:image.extent fromRect:image.extent];
  [_glview display];
}
```

- **捕捉连接——AVCaptureConnection**

捕捉连接负责将捕捉会话接收的媒体类型和输出连接起来，比如AVCaptureAudioDataOutput可以接受音频数据，AVCaptureVideoDataOutput可以接受视频数据。会话通过捕捉连接，确定哪些输入视频，那些输入音频。通过对捕捉连接的访问，可以对信号流进行底层控制，比如禁用某些特定的连接。

```
  // 设置视频捕捉连接
  _videoConnection = [videoOutput connectionWithMediaType:AVMediaTypeVideo];
  _videoConnection.videoOrientation = self.referenceOrientation;
  // 在视频元数据的输出函数中，如果捕捉连接是视频连接，则写入视频数据
  if (connection == _videoConnection){
      if ([self inputsReadyToRecord]){
          [self writeSampleBuffer:sampleBuffer ofType:AVMediaTypeVideo];
      }
  }

  // 设置音频捕捉连接
  _audioConnection = [audioOut connectionWithMediaType:AVMediaTypeAudio];
  // 在视频元数据的输出函数中，如果捕捉连接是音频连接，则写入音频数据
  if (connection == _audioConnection){
      if (_readyToRecordVideo && _readyToRecordAudio){
          [self writeSampleBuffer:sampleBuffer ofType:AVMediaTypeAudio];
      }
  }
```

- **拍照——AVCaptureStillImageOutput**

AVCaptureStillImageOutput会为我们捕捉高分辨率的图像，起设置如下：

```
  // 创建image output 代码
  AVCaptureStillImageOutput *imageOutput = [[AVCaptureStillImageOutput alloc]
init];
  imageOutput.outputSettings = @{AVVideoCodecKey:AVVideoCodecJPEG};
  if ([_captureSession canAddOutput:imageOutput]) {
      [_captureSession addOutput:imageOutput];
      _imageOutput = imageOutput;
  }

  // 输出图片
  AVCaptureConnection *connection = [_imageOutput connectionWithMediaType:AVMe
diaTypeVideo];
  if (connection.isVideoOrientationSupported) {
      connection.videoOrientation = [self currentVideoOrientation];
  }
  id takePictureSuccess = ^(CMSampleBufferRef sampleBuffer,NSError *error){
      if (sampleBuffer == NULL) {
          [self showError:error];
          return ;
      }
      NSData *imageData = [AVCaptureStillImageOutput jpegStillImageNSDataRepre
sentation:sampleBuffer];
      UIImage *image = [[UIImage alloc]initWithData:imageData];
  };
  [_imageOutput captureStillImageAsynchronouslyFromConnection:connection compl
etionHandler:takePictureSuccess];
```

- **音频——AVCaptureAudioDataOutput**

AVCaptureAudioDataOutput(音频数据输出)：它输出硬件实时捕捉的音频数字样本，还有一个音频输出类是AVCaptureAudioFileOutput，不过它只能在录制完成后输出完整的音频文件。

```
  // 音频输出
  AVCaptureAudioDataOutput *audioOut = [[AVCaptureAudioDataOutput alloc] init]
;
  [audioOut setSampleBufferDelegate:self queue:captureQueue];
  if ([_captureSession canAddOutput:audioOut]){
      [_captureSession addOutput:audioOut];
  }
```

- **视频——AVCaptureVideoDataOutput**

AVCaptureVideoDataOutput(视频数据输出)：它输出硬件实时捕捉的视频数字样本，

还有一个音频和视频输出类是AVCaptureMovieFileOutput，不过它只能在录制完成后输出完整的视频和音频文件。

```
  // 视频输出
  AVCaptureVideoDataOutput *videoOut = [[AVCaptureVideoDataOutput alloc] init]
;
  [videoOut setAlwaysDiscardsLateVideoFrames:YES];
  [videoOut setVideoSettings:@{(id)kCVPixelBufferPixelFormatTypeKey : [NSNumbe
r numberWithInt:kCVPixelFormatType_32BGRA]}];
  [videoOut setSampleBufferDelegate:self queue:captureQueue];
  if ([_captureSession canAddOutput:videoOut]){
      [_captureSession addOutput:videoOut];
      _videoOutput = videoOut;
  }
```

- **生成视频文件——AVAssetWriter、AVAssetWriterInput**

AVAssetWriter：用于对媒体资源进行编码并讲其写入到容器文件中，比如一个QuickTime文件。

AVAssetWriterInput：用于处理指定的媒体类型，比如音频和视频。

AVAssetWriterInputPixelBufferAdaptor:这个类在生成视频文件时提供最优性能，不过Demo没有使用该类，有兴趣的可以去研究一下

```
// 初始化一个assetWriter
NSError *error;
_assetWriter = [[AVAssetWriter alloc] initWithURL:_movieURL fileType:AVFileTyp
eQuickTimeMovie error:&error];
if (error){
   [self showError:error];
}

// 配置视频源数据输入
- (BOOL)setupAssetWriterVideoInput:(CMFormatDescriptionRef)currentFormatDescri
ption
{
  CGFloat bitsPerPixel;
  CMVideoDimensions dimensions = CMVideoFormatDescriptionGetDimensions(current
FormatDescription);
  NSUInteger numPixels = dimensions.width * dimensions.height;
  NSUInteger bitsPerSecond;

  if (numPixels < (640 * 480)){
      bitsPerPixel = 4.05;
  }
  else{
      bitsPerPixel = 11.4;
  }

  bitsPerSecond = numPixels * bitsPerPixel;
  NSDictionary *videoCompressionSettings = @{AVVideoCodecKey  : AVVideoCodecH2
64,
                                             AVVideoWidthKey  : [NSNumber numb
erWithInteger:dimensions.width],
                                             AVVideoHeightKey : [NSNumber numb
erWithInteger:dimensions.height],
                                             AVVideoCompressionPropertiesKey:@
{AVVideoAverageBitRateKey:[NSNumber numberWithInteger:bitsPerSecond],

 AVVideoMaxKeyFrameIntervalKey:[NSNumber numberWithInteger:30]}
                                             };
  if ([_assetWriter canApplyOutputSettings:videoCompressionSettings forMediaTy
pe:AVMediaTypeVideo])
  {
      _assetVideoInput = [AVAssetWriterInput assetWriterInputWithMediaType:AVM
ediaTypeVideo outputSettings:videoCompressionSettings];
      _assetVideoInput.expectsMediaDataInRealTime = YES;
      _assetVideoInput.transform = [self transformFromCurrentVideoOrientationT
oOrientation:self.referenceOrientation];
      if ([_assetWriter canAddInput:_assetVideoInput]){
          [_assetWriter addInput:_assetVideoInput];
      }
      else{
          [self showError:_assetWriter.error];
```

```
          return NO;
      }
  }
  else{
      [self showError:_assetWriter.error];
      return NO;
  }
  return YES;
}

// 配置音频源数据输入
- (BOOL)setupAssetWriterAudioInput:(CMFormatDescriptionRef)currentFormatDescri
ption
{
  size_t aclSize = 0;
  const AudioStreamBasicDescription *currentASBD = CMAudioFormatDescriptionGet
StreamBasicDescription(currentFormatDescription);
  const AudioChannelLayout *currentChannelLayout = CMAudioFormatDescriptionGet
ChannelLayout(currentFormatDescription, &aclSize);

  NSData *currentChannelLayoutData = nil;
  if (currentChannelLayout && aclSize > 0 ){
      currentChannelLayoutData = [NSData dataWithBytes:currentChannelLayout le
ngth:aclSize];
  }
  else{
      currentChannelLayoutData = [NSData data];
  }

  NSDictionary *audioCompressionSettings = @{AVFormatIDKey : [NSNumber numberW
ithInteger:kAudioFormatMPEG4AAC],
                                             AVSampleRateKey : [NSNumber numbe
rWithFloat:currentASBD->mSampleRate],
                                             AVEncoderBitRatePerChannelKey : [
NSNumber numberWithInt:64000],
                                             AVNumberOfChannelsKey : [NSNumber
 numberWithInteger:currentASBD->mChannelsPerFrame],
                                             AVChannelLayoutKey : currentChann
elLayoutData};

  if ([_assetWriter canApplyOutputSettings:audioCompressionSettings forMediaTy
pe:AVMediaTypeAudio])
  {
      _assetAudioInput = [AVAssetWriterInput assetWriterInputWithMediaType:AVM
ediaTypeAudio outputSettings:audioCompressionSettings];
      _assetAudioInput.expectsMediaDataInRealTime = YES;

      if ([_assetWriter canAddInput:_assetAudioInput]){
          [_assetWriter addInput:_assetAudioInput];
      }
      else{
          [self showError:_assetWriter.error];
          return NO;
      }
  }
  else{
      [self showError:_assetWriter.error];
      return NO;
  }

  return YES;
}
```

通过上面的代码，我们就准备好了一个AVAssetWriter了，就可以用它来生产视频文件，我们可以在视频源数据输出函数中写入

```
- (void)captureOutput:(AVCaptureOutput *)captureOutput didOutputSampleBuffer:(
CMSampleBufferRef)sampleBuffer fromConnection:(AVCaptureConnection *)connectio
n{
if (_recording) {
      CFRetain(sampleBuffer);
      dispatch_async(_movieWritingQueue, ^{
          if (_assetWriter)
          {
              if (connection == _videoConnection)
              {
                  if (!_readyToRecordVideo){
                      _readyToRecordVideo = [self setupAssetWriterVideoInput:C
MSampleBufferGetFormatDescription(sampleBuffer)];
                  }
                  if ([self inputsReadyToRecord]){
                      [self writeSampleBuffer:sampleBuffer ofType:AVMediaTypeV
ideo];
                  }
              }
              else if (connection == _audioConnection){
                  if (!_readyToRecordAudio){
                      _readyToRecordAudio = [self setupAssetWriterAudioInput:C
MSampleBufferGetFormatDescription(sampleBuffer)];
                  }
                  if ([self inputsReadyToRecord]){
                      [self writeSampleBuffer:sampleBuffer ofType:AVMediaTypeA
udio];
                  }
              }
          }
          CFRelease(sampleBuffer);
      });
  }
}
```

- **写入相册——ALAssetsLibrary、PHPhotoLibrary**

iOS9.0以前：

```
  ALAssetsLibrary *lab = [[ALAssetsLibrary alloc]init];
  // 保存视频
  [lab writeVideoAtPathToSavedPhotosAlbum:_movieURL completionBlock:^(NSURL *a
ssetURL, NSError *error) {
       if (error) {
           [self showError:error];
       }
   }];
```

iOS9.0以后

```
[PHPhotoLibrary requestAuthorization:^( PHAuthorizationStatus status ) {
     if (status == PHAuthorizationStatusAuthorized) {
         [[PHPhotoLibrary sharedPhotoLibrary] performChanges:^{
             // 保存视频
             PHAssetCreationRequest *videoRequest = [PHAssetCreationRequest cr
eationRequestForAsset];
             [videoRequest addResourceWithType:PHAssetResourceTypeVideo fileUR
L:_movieURL options:nil];
          } completionHandler:^( BOOL success, NSError * _Nullable error ) {
             if (!success) {
                 [self showError:error];
             }
         }];
     }
}];
```

- **操作相机**

相机的操作都是一些固定的代码，我就不多讲了，我们只需要注意以下几点：

1.闪光灯和手电筒不能同时开启

2.在前置摄像头时不能开启手电筒，所有在转换时，会被强制关闭

3.前后摄像头需要分别设置闪光灯的开关，所以我们必须记录当前闪光灯的设置状态，在转换完成之后，还需要重新设置一次
4.在转换摄像头时，你之前设置的视频输出就无效了，你需要删除原来的视频输出，再重新添加一个新的视频输出(我也不知道为什么会有这种情况，但是音频源数据是一直都有的，视频源数据每次转换摄像头都需要重新设置视频输出)

- **转换摄像头**

```
- (BOOL)switchCameras{
  if (![self canSwitchCameras]) {
      return NO;
  }
  NSError *error;
  AVCaptureDevice *videoDevice = [self inactiveCamera];
  AVCaptureDeviceInput *videoInput = [AVCaptureDeviceInput deviceInputWithDevi
ce:videoDevice error:&error];
  if (videoInput) {
      [_captureSession beginConfiguration];
      [_captureSession removeInput:_deviceInput];
      if ([_captureSession canAddInput:videoInput]) {
          [_captureSession addInput:videoInput];
          _deviceInput = videoInput;
      }
      else{
          [_captureSession addInput:_deviceInput];
      }
      [_captureSession commitConfiguration];
      // 如果从后置转前置，会关闭手电筒，如果之前打开的，需要通知camera更新UI
      if (videoDevice.position == AVCaptureDevicePositionFront) {
          [self.cameraView changeTorch:NO];
      }
      // 闪关灯，前后摄像头的闪光灯是不一样的，所以在转换摄像头后需要重新设置闪光灯
      [self changeFlash:_currentflashMode];
      // 转换摄像头时，视频输出就无效了，所以在转换回来时，需要把原来的删除了，在重新加一个
新的进去
      [self resetupVideoOutput];
  }
  else{
      [self showError:error];
      return NO;
  }
  return YES;
}

-(void)resetupVideoOutput{
  [_captureSession beginConfiguration];
  [_captureSession removeOutput:_videoOutput];

  AVCaptureVideoDataOutput *videoOut = [[AVCaptureVideoDataOutput alloc] init]
;
  [videoOut setAlwaysDiscardsLateVideoFrames:YES];
  [videoOut setVideoSettings:@{(id)kCVPixelBufferPixelFormatTypeKey : [NSNumbe
r numberWithInt:kCVPixelFormatType_32BGRA]}];
  dispatch_queue_t videoCaptureQueue = dispatch_queue_create("Video Capture Qu
eue", DISPATCH_QUEUE_SERIAL);
  [videoOut setSampleBufferDelegate:self queue:videoCaptureQueue];

  if ([_captureSession canAddOutput:videoOut]) {
      [_captureSession addOutput:videoOut];
      _videoOutput = videoOut;
  }
  _videoConnection = [videoOut connectionWithMediaType:AVMediaTypeVideo];
  [_captureSession commitConfiguration];
}
```

- **补光**

```
  AVCaptureDevice *device = [self activeCamera];
  if (device.torchMode != torchMode && [device isTorchModeSupported:torchMode]
) {
      NSError *error;
      if ([device lockForConfiguration:&error]) {
          device.torchMode = torchMode;
          [device unlockForConfiguration];
      }
      else{
          [self showError:error];
      }
  }
```

- **闪光灯**

```
  AVCaptureDevice *device = [self activeCamera];
  if (device.flashMode != flashMode && [device isFlashModeSupported:flashMode]
) {
      NSError *error;
      if ([device lockForConfiguration:&error]) {
          device.flashMode = flashMode;
          [device unlockForConfiguration];
      }
      else{
          [self showError:error];
      }
  }
```

- **聚焦**

```
- (void)focusAtPoint:(CGPoint)point {
  AVCaptureDevice *device = [self activeCamera];
  if ([self cameraSupportsTapToFocus] && [device isFocusModeSupported:AVCaptur
eFocusModeAutoFocus]) {
      NSError *error;
      if ([device lockForConfiguration:&error]) {
          device.focusPointOfInterest = point;
          device.focusMode = AVCaptureFocusModeAutoFocus;
          [device unlockForConfiguration];
      }
      else{
          [self showError:error];
      }
  }
}
```

- **曝光**

```
static const NSString *CameraAdjustingExposureContext;
- (void)exposeAtPoint:(CGPoint)point{
  AVCaptureDevice *device = [self activeCamera];
  if ([self cameraSupportsTapToExpose] && [device isExposureModeSupported:AVCa
ptureExposureModeContinuousAutoExposure]) {
      NSError *error;
      if ([device lockForConfiguration:&error]) {
          device.exposurePointOfInterest = point;
          device.exposureMode = AVCaptureExposureModeContinuousAutoExposure;
          if ([device isExposureModeSupported:AVCaptureExposureModeLocked]) {
              [device addObserver:self
                       forKeyPath:@"adjustingExposure"
                          options:NSKeyValueObservingOptionNew
                          context:&CameraAdjustingExposureContext];
          }
          [device unlockForConfiguration];
      }
      else{
          [self showError:error];
      }
  }
}

- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:
(NSDictionary *)change context:(void *)context {
  if (context == &CameraAdjustingExposureContext) {
      AVCaptureDevice *device = (AVCaptureDevice *)object;
      if (!device.isAdjustingExposure && [device isExposureModeSupported:AVCap
tureExposureModeLocked]) {
          [object removeObserver:self
                      forKeyPath:@"adjustingExposure"
                         context:&CameraAdjustingExposureContext];
          dispatch_async(dispatch_get_main_queue(), ^{
              NSError *error;
              if ([device lockForConfiguration:&error]) {
                  device.exposureMode = AVCaptureExposureModeLocked;
                  [device unlockForConfiguration];
              }
              else{
                  [self showError:error];
              }
          });
      }
  }
  else{
      [super observeValueForKeyPath:keyPath
                           ofObject:object
                             change:change
                            context:context];
  }
}
```

- **自动聚焦/曝光**

```
- (BOOL)resetFocusAndExposureModes{
  AVCaptureDevice *device = [self activeCamera];
  AVCaptureExposureMode exposureMode = AVCaptureExposureModeContinuousAutoExpo
sure;
  AVCaptureFocusMode focusMode = AVCaptureFocusModeContinuousAutoFocus;
  BOOL canResetFocus = [device isFocusPointOfInterestSupported] && [device isF
ocusModeSupported:focusMode];
  BOOL canResetExposure = [device isExposurePointOfInterestSupported] && [devi
ce isExposureModeSupported:exposureMode];
  CGPoint centerPoint = CGPointMake(0.5f, 0.5f);
  NSError *error;
  if ([device lockForConfiguration:&error]) {
      if (canResetFocus) {
          device.focusMode = focusMode;
          device.focusPointOfInterest = centerPoint;
      }
      if (canResetExposure) {
          device.exposureMode = exposureMode;
          device.exposurePointOfInterest = centerPoint;
      }
      [device unlockForConfiguration];
      return YES;
  }
  else{
      [self showError:error];
      return NO;
  }
}
```

◦ 视频重力——Video gravity

视频重力：控制视频内容渲染的缩放和拉伸效果。
举个例子，在我们设置会话时有一个参数session preset，它是用来控制捕捉数据格式和质量了。我的测试机是6s，当我选择参数AVCaptureSessionPresetPhoto时，输出图片大小如下：

```
Printing description of image:
<CIImage: 0x12c7bdad0 extent [0 0 750 1000]>
affine [1 0 0 -1 0 1000] extent=[0 0 750 1000]
  colormatch "QuickTime 'nclc' Video (1,1,6)"-to-workingspace extent=[0 0 750
1000]
  IOSurface 0x12da00008 BGRA8 extent=[0 0 750 1000]
```

当我选择参数AVCaptureSessionPresetHigh时，输出图片大小如下：

```
Printing description of image:
<CIImage: 0x15f851680 extent [0 0 1080 1920]>
affine [1 0 0 -1 0 1920] extent=[0 0 1080 1920]
  colormatch "QuickTime 'nclc' Video (1,1,6)"-to-workingspace extent=[0 0 1080
 1920]
  IOSurface 0x15f900008 BGRA8 extent=[0 0 1080 1920]
```

可以看出选择不同的session preset，会输出不同大小的图片，但是这些图片都是很大的，这么大的图片要显示在手机预览层，必须要缩放，而视频重力其实就是缩放参数。
AVLayerVideoGravityResizeAspect：在预览层区域内缩放视频，保持视频原始宽高比。这是默认值，同时适用大多数情况。使用该参数预览时，有可能不能铺满整个预览视图
AVLayerVideoGravityResizeAspectFill：按照视频的宽高比将视频拉伸填满整个图层。使用该参数时，很可能造成视频预览图片被裁剪，而拍摄输出没有被裁剪，这样就会使预览图和最终拍摄的图不一致。
AVLayerVideoGravityResize：拉伸视频内容以匹配预览层大小，这个是最不常用的，可能造成视频扭曲。

◦ 方向问题——Orientation

设备方向device orientation

```
// 设备方向
UIDevice *device = [UIDevice currentDevice] ;
switch (device.orientation) {
    case UIDeviceOrientationFaceUp:
    NSLog(@"屏幕朝上平躺");
        break;
    case UIDeviceOrientationFaceDown:
    NSLog(@"屏幕朝下平躺");
        break;
    case UIDeviceOrientationUnknown:
    NSLog(@"未知方向");
        break;
    case UIDeviceOrientationLandscapeLeft:
    NSLog(@"屏幕向左横置");
        break;
    case UIDeviceOrientationLandscapeRight:
    NSLog(@"屏幕向右横置");
        break;
    case UIDeviceOrientationPortrait:
    NSLog(@"屏幕直立");
        break;
    case UIDeviceOrientationPortraitUpsideDown:
    NSLog(@"屏幕直立，上下颠倒");
        break;
```

从上面可以看到所有的设备方向，而视频方向videoOrientation没有那么多分类，它分为：
AVCaptureVideoOrientationPortrait home健在下
AVCaptureVideoOrientationPortraitUpsideDown home健在上
AVCaptureVideoOrientationLandscapeRight home健在右
AVCaptureVideoOrientationLandscapeLeft home健在左
这些视频方向，是视频或拍照时的输入方向，而我们的数据输出时会跟具这些输入方向自动对图片或视频进行矩阵变换，以达到最佳的用户体验。
这里以拍照举个例子(视频同理)：
假如你横着手机拍了一张照片，第一次你在拍照前不传入视频方向，它默认为AVCaptureVideoOrientationPortrait，这是正常手机拿着的姿势，所以到输出时不会对图片进行矩阵变换，当你把图片存入相册时，你会发现，你要正确查看这张图，你也需要横着手机看。如果你是倒着手机拍的，就需要倒着手机看。但是如果你在拍照前传入视频方向，比如你横着手机拍，并且home健在右，就传入参数AVCaptureVideoOrientationLandscapeRight，这时你存入相册的照片就可以以正常拿手机的姿势查看它了。

```
 // 在拍照前通过会话连接，传入当前输入视频方向(视频同理也可以这样做)
AVCaptureConnection *connection = [_imageOutput connectionWithMediaType:AVMedi
aTypeVideo];
  if (connection.isVideoOrientationSupported) {
      connection.videoOrientation = [self currentVideoOrientation];
  }
```

苹果给出的类处理后都是默认正常拿手机的姿势观看，不管是图片还是视频，如果我们想拍出的所有图片或视频都需要横着手机看，我们这时可以不传入视频方向，这样视频到输出时就不会被变换，我们在视频输入类中，手动对视频进行transform变换，这样就可以实现我们想要的查看方式，在本例中，视频就是用的这种处理方式。

```
// 视频的播放方向，后面计算视频旋转角度使用
_referenceOrientation = AVCaptureVideoOrientationPortrait;

  // 这行代码在设置视频输入方向为默认输入方向
_videoConnection.videoOrientation = AVCaptureVideoOrientationPortrait;

  // 视频输入类中手动旋转视频方向
_assetVideoInput.transform = [self transformFromCurrentVideoOrientationToOrien
tation:self.referenceOrientation];

  // 旋转视频方向函数实现
  -  (CGAffineTransform)transformFromCurrentVideoOrientationToOrientation:(AVC
aptureVideoOrientation)orientation
{
      CGFloat orientationAngleOffset = [self angleOffsetFromPortraitOrientatio
nToOrientation:orientation];
      CGFloat videoOrientationAngleOffset = [self angleOffsetFromPortraitOrien
tationToOrientation:self.motionManager.videoOrientation];
      CGFloat angleOffset;
      if ([self activeCamera].position == AVCaptureDevicePositionBack) {
          angleOffset = orientationAngleOffset - videoOrientationAngleOffset;
      }
      else{
          angleOffset = videoOrientationAngleOffset - orientationAngleOffset +
 M_PI_2;
      }
      CGAffineTransform transform = CGAffineTransformMakeRotation(angleOffset)
;
      return transform;
}

  - (CGFloat)angleOffsetFromPortraitOrientationToOrientation:(AVCaptureVideoOr
ientation)orientation
{
  CGFloat angle = 0.0;
  switch (orientation)
  {
      case AVCaptureVideoOrientationPortrait:
          angle = 0.0;
          break;
      case AVCaptureVideoOrientationPortraitUpsideDown:
          angle = M_PI;
          break;
      case AVCaptureVideoOrientationLandscapeRight:
          angle = -M_PI_2;
          break;
      case AVCaptureVideoOrientationLandscapeLeft:
          angle = M_PI_2;
          break;
      default:
          break;
  }
  return angle;
}
```

- **项目地址**

https://github.com/cdcyd/CCCamera (https://github.com/cdcyd/CCCamera)

iOS (/nb/5597037) 举报文章 © 著作权归作者所有

cdcyd (/u/2841ca1b2535) ♂
写了 10924 字，被 158 人关注，获得了 215 个喜欢
(/u/2841ca1b2535)

+关注

♡ 喜欢 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-like-button) | 77

(http://cwb.assets.jianshu.io/notes/images/5473611/