# 《计算机视觉-openCV应用技术》

## 第一章 安装 OpenCV

## 第二章 处理文件、摄像头和图像用户界面

李策

中国矿业大学（北京）计算机科学与技术系

E-mail: celi@cumtb.edu.cn

# 提纲

## 第一章 安装 OpenCV

- 1.1 选择和使用合适的安装工具
- 1.2 安装Contrib模块
- 1.3 运行示例
- 1.4 查找文档、帮助及更新
- 1.5 总结

# 1.1 选择和使用合适的安装工具

### 1.1.1 在Windows上安装

1.使用二进制安装程序

2.使用Cmake和编译器

### 1.1.2 在OS X系统中安装

1.使用MacPorts的现成包

2.在MacPorts上使用自定义软件包

3.使用Homebrew的现成包

4.在Homebrew上自定义软件包

### 1.1.3 在Ubuntu及其衍生版本中安装

1.使用Ubuntu的资源库

2.从源代码构建OpenCV

### 1.1.4 在其他类Unix系统中安装

# 1.1.1 在Windows上安装

前期准备：预编译好的Python、NumPy、SciPy和OpenCV

修改PATH变量步骤：

‘控制面板’——‘环境变量’

1）Windows Vista / Windows 7 / Windows 8：

- 单击“开始”菜单
- 打开“控制面板”
- 选择“系统”和“安全|系统|高级系统设置”
- 单击“环境变量...”按钮

2）Windows XP：

- 单击“开始”菜单
- 打开“控制面板|系统”
- 选择“高级”选项卡
- 单击“环境变量...”按钮

# 1.1.1 在Windows上安装

3）在“系统变量”中选择“路径”，然后单击“编辑”按钮

4）按规定进行修改

5）选择“确定”按钮

6）注销并重新登录

# 1.1.2 在OS X系统中安装

前期准备：确保Xcode开发工具已正确安装

1）从Mac App Store或https://developer.apple.com/xcode/downloads/

下载并安装Xcode 。出现“命令行工具（Command Line Tools)”时，选择它

2）打开并接受许可协议

3）如果没出现“命令行工具”选项，执行以下步骤：

选择Xcode|Preferences|Downloads

单击“命令行工具”安装

还可以终端执行命令:

$ xcode-select -install

# 1.1.3 在Ubuntu及衍生版本中安装

1. 使用Ubuntu的资源库（不支持深度摄像头）

2. 从源代码构建OpenCV

# 1.1.4 在其他类Unix系统中安装

- 在Debian Linux中：软件包可能不同，使用方式与Ubuntu中一样
- 在Gentoo Linux中：软件包管理器称为Portage，与MacPorts相似，软件包不同
- 在FreeBSD中：安装过程与MacPorts相似，采用ports来安装系统
- 在其他类Unix中：查阅包管理器的说明文档

# 1.2 安装Contrib模块

- 有些模块包含在称为opencv_contrib的资源库中，可从github中下载
- 下载成功后，需重新运行cmake命令，该命令可生成带opencv_contrib模块的OpenCV项目
- 在下载OpenCV的文件夹下创建一个build文件夹，运行命令

# 提纲

**第二章**

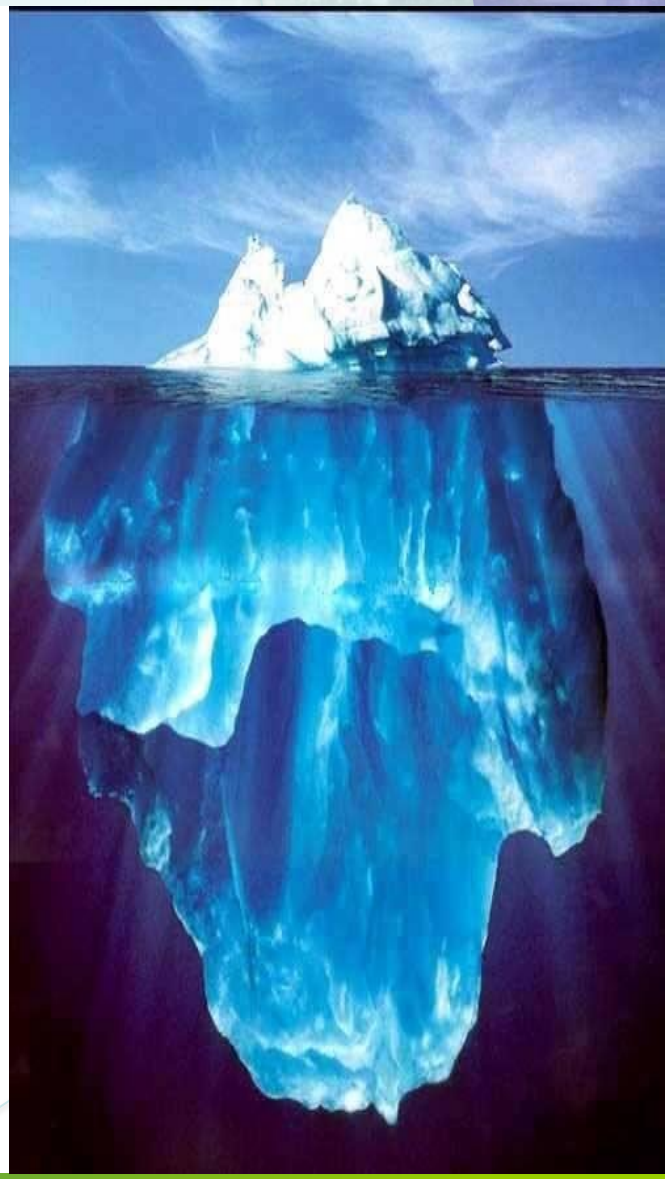
**处理文件、摄像头和图像用户界面**

**2.1 基本I/O脚本**

**2.2 Cameo 项目（人脸跟踪和图像处理）**

**2.3 Cameo—面向对象的设计**

**2.4 总结**



# 2.1.1 读 / 写图像文件

OpenCV的**imread()函数**和**imwrite()函数**能支持各种静态图像文件格式。

不同系统支持的文件格式不一样，但都支持BMP格式，通常还应该支持PNG、JPEG和TIFF格式。

无论哪种格式，每个像素都会有一个值，但不同格式表示像素的方式有所不同。如：

# 2.1.1 读 / 写图像文件

```
import cv2
import numpy as np

img = np.zeros((3, 3), dtype = np.uint8)    # 通过二维NumPy数组来简单创建一个黑色的正方形图像
print(img)                                  # 在控制台打印该图像
print(img.shape)                            # 通过shape属性来查看图像的结构，返回行和列，如果有一个以上的通道，还会返回通道数
img = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)  # 利用cv2.cvtColor函数将该图像转换成BGR格式
print(img)
print(img.shape)
cv2.namedWindow("Image")                    # 显示该图像
cv2.imshow("Image", img)
cv2.waitKey (0)
```

# 2.1.1 读 / 写图像文件

结果为：

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
# 每个像素都由一个8位整数来表示，即每个像素值的范围是0~255
(3, 3)
[[[0 0 0]
  [0 0 0]
  [0 0 0]]
 [[0 0 0]
  [0 0 0]
  [0 0 0]]
 [[0 0 0]
  [0 0 0]
  [0 0 0]]]
```

# 2.1.1 读 / 写图像文件

# 现在每个像素都由一个三元数组表示，并且每个整型向量分别表示一个B,G和R通道。其他色彩空间（如HSV）也以同样的方式来表示像素，只是取值范围和通道数目不同（例如，HSV色彩空间的色度值范围为0~180）

(3, 3, 3)

# 此时每个像素有3通道

# 2.1.1 读 / 写图像文件

可读取一种格式的图像文件，然后将其保存为另一种格式。如

image = cv2.imread(‘dog.jpg’) # 将 ‘dog.jpg’的图片与.py文件放在同一目录下，或者使用绝对路径
cv2.imwrite(' dog.png',image)
程序运行后，可看到文件夹中多了’dog.png'的图片

dog.jpg

dog.png

# 2.1.1 读 / 写图像文件

在默认情况下，即使图像文件为灰度格式，imread()函数也会返回BGR格式的图像，BGR与RGB表示的色彩空间相同，但字节顺序相反。

下面列出的选项可作为imread()函数的参数：

IMREAD_ANYCOLOR = 4
IMREAD_ANYDEPTH = 2
IMREAD_COLOR = 1
IMREAD_GRAYSCALE = 0
IMREAD_LOAD_GDAL = 8
IMREAD_UNCHANGED = -1

cv2.imread(‘图像名称’，’可选参数’)

可选参数决定读入图像的模式：

**0：读入的为灰度图像**（即使图像为彩色的）

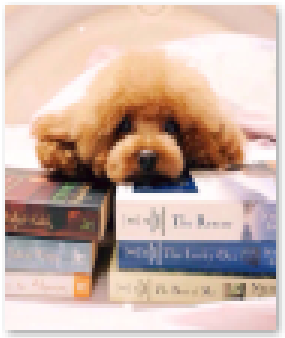
**1：读入的图像为彩色的**（默认）；

# 2.1.1 读 / 写图像文件

如下面的例子将加载的PNG文件作为灰度图像（在这个过程中会丢失所有的颜色信息），然后又将其保存为灰度的PNG图像。

```
garyImage = cv2.imread(' dog.jpg',cv2.IMREAD_GRAYSCALE)
cv2.imwrite(' doggray.png',garyImage)
```

程序运行后，文件夹中出现了灰度图像



dog.png

doggray.png

# 2.1.1 读 / 写图像文件

无论采用哪种模式，imread()函数会删除所有alpha通道的信息（透明度）。imwrite()函数要求图像为BGR或灰度格式，并且每个通道要有一定的位（bit）

输出格式要支持这些通道。例如，bmp格式要求每个通道有8位，而PNG允许每个通道有8位或16位

# 2.1.2 图像与原始字节之间的转换

一个OpenCV图像是.array类型的二维或三维数组。8位的灰度图像是一个含有字节值的二维数组。

可使用表达式访问这些值：如image[0,0]或image[0,0,0]。
第一个值代表像素的y坐标或行，0表示顶部；
第二个值是像素的x坐标或列，0表示最左边；
第三个值（如果可用的话）表示颜色通道。

若一幅图像的每个通道为8位，则可将其显式转换为标准的一维Python bytearray格式：byteArray = bytearray(image)

反之，bytearray含有恰当顺序的字节，可以通过显式转换和重构，得到numpy.array形式的图像：

```
garyImage = numpy.array(garyByteArray ).reshape(height, width)
bgrImage = numpy.array(bgrByteArray ).reshape(height, width, 3)
```

# 2.1.2 图像与原始字节之间的转换

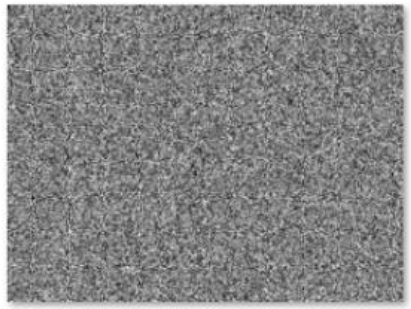
```
# 创建一个120000个随机字节的数组
randomByteArray = bytearray(os.urandom(120000))  #os.urandom(n) 返回n个随机byte值的string，作为加密使用
flatNumpyArray = np.array(randomByteArray)

# 将数组转换为400 x 300的灰度图像
garyImage = flatNumpyArray.reshape(300, 400)
cv2.imwrite('randomGary.png', garyImage)

# 将数组转换为400 x 100的彩色图像
bgrImage = flatNumpyArray.reshape(100, 400, 3)
cv2.imwrite('randomColor.png', bgrImage)
```

运行该程序，将会在程序所在目录中生成两张图像，尺寸分别为400 x 100，400 x 300

randomColor.png



randomGary.png

# 2.1.3 使用numpy.array访问图像数据

一个OpenCV图像是.array类型的二维或三维数组。8位的灰度图像是一个含有字节值的二维数组。

可使用表达式访问这些值：如image[0,0]或image[0,0,0]。
第一个值代表像素的y坐标或行，0表示顶部；
第二个值是像素的x坐标或列，0表示最左边；
第三个值（如果可用的话）表示颜色通道。

若一幅图像的每个通道为8位，则可将其显式转换为标准的一维Python bytearray格式：byteArray = bytearray(image)

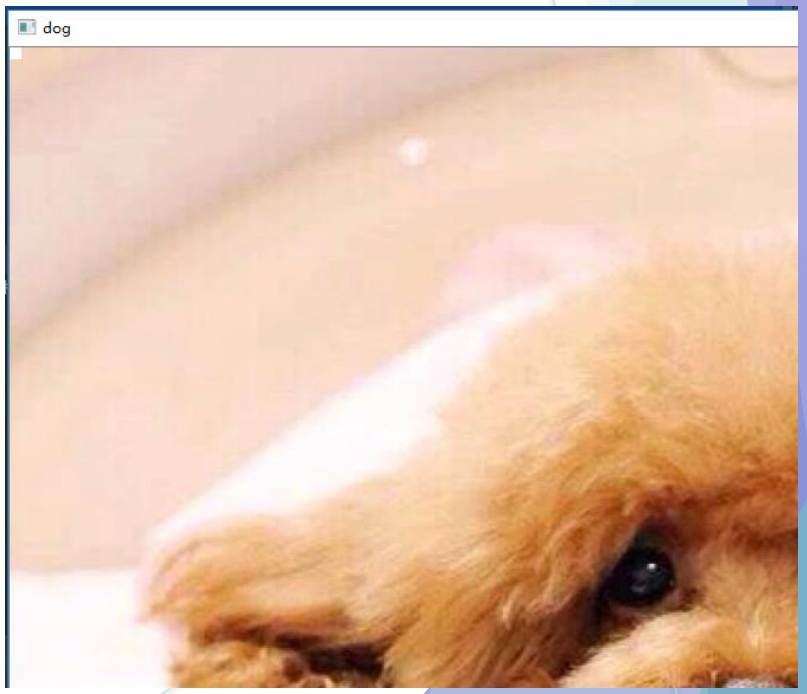
反之，bytearray含有恰当顺序的字节，可以通过显式转换和重构，得到numpy.array形式的图像：

```
garyImage = numpy.array(garyByteArray ).reshape(height, width)
bgrImage = numpy.array(bgrByteArray ).reshape(height, width, 3)
```

# 2.1.3 使用numpy.array访问图像数据

```
img = cv2.imread(‘dog.png')
img[0,0] = [255, 255, 255]
cv2.imshow('flower', img)
cv2.waitKey(0)
```

在图像左上方会出现一个白点。



# 2.1.3 使用numpy.array访问图像数据

假设想要改变一个特定像素的蓝色值，numpy.array提供了item()方法。

该函数有3个参数：x（或左）位置、y（或顶部）位置以及（x,y）位置的数组索引

（注意，在BGR图像中，某一位置的数据是按B,G,R的顺序保存的三元数组）

该函数能返回索引位置的值。另一个方法是通过itemset()函数可设置指定像素在指定通道的值

itemset()有两个参数：一个三元组(x、y和索引）和要设定的值

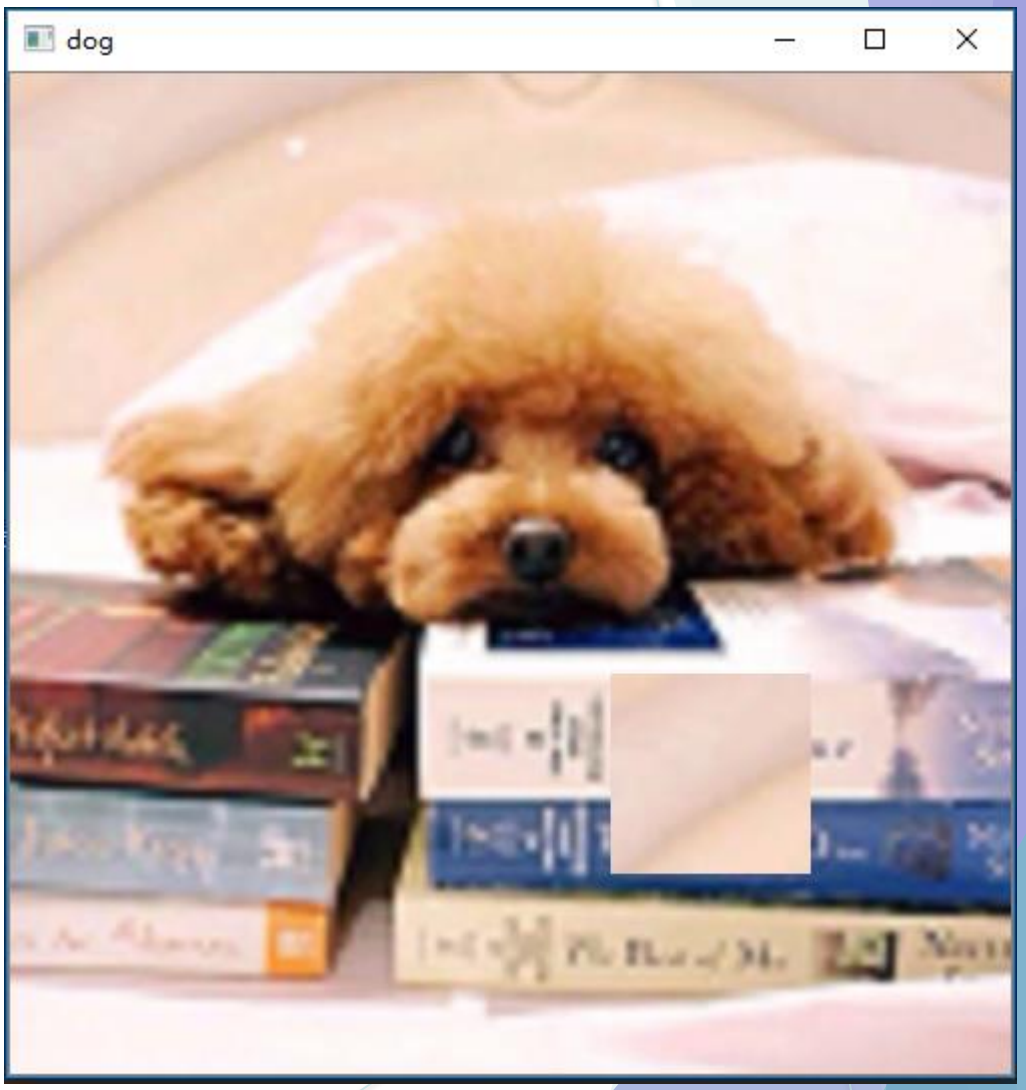
如下例子将坐标（150,120）的当前蓝色值127变为255

# 2.1.3 使用numpy.array访问图像数据

```
img = cv2.imread(‘dog.png')
print(img.item(150, 120, 0))  # 打印当前坐标点的蓝色值
img.itemset((150, 120, 0), 255)
print(img.item(150, 120, 0)) #255
```

# 2.1.3 使用numpy.array访问图像数据

```
img = cv2.imread("dog.png")
my_roi = img[0:100, 0:100]
img[300:400, 300:400] = my_roi
cv2.imshow('dog', img)
cv2.waitKey()
```



# 2.1.3 使用numpy.array访问图像数据

还可使用numpy.array来获得图像其他属性:

- shape：NumPy返回包含宽度、高度和通道数的数组
- size：该属性是指图像像素的大小
- datatype：该属性会得到图像的数据类型

```
img = cv2.imread('dog.png')
print(img.shape)
print(img.size)
print(img.dtype)
```

运行结果：

```
(500, 500, 3)
750000
uint8
```

# 2.1.4 视频文件的读 / 写

视频文件的读：

OpenCV提供了 VideoCapture和 VideoWrite类支持视频文件

```
Import cv2
videoCapture = cv2.VideoCapture('MyInputVid.avi')
```

# 2.1.4 视频文件的读 / 写

视频文件的写：

```
videoWriter = cv2.VideoWriter(
    'MyOutputVid.avi',
    cv2.VideoWriter_fourcc('I', '4', '2', '0'),
    fps,
    size
    )
```

# 2.1.4 视频文件的读 / 写

必须为 VideoWriter 类的构造函数指定视频文件名，若文件已存在，则被覆盖；必须为视频指定编解码器，编解码器的可用性根据系统的不同而不同。

- cv2.VideoWriter_fourcc('I', '4', '2', '0')
- cv2.VideoWriter_fourcc('P', 'I', 'M', '1')
- cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')
- cv2.VideoWriter_fourcc('T', 'H', 'E', 'O')
- cv2.VideoWriter_fourcc('F', 'L', 'V', 'I')

# 2.1.5 捕获摄像头的帧

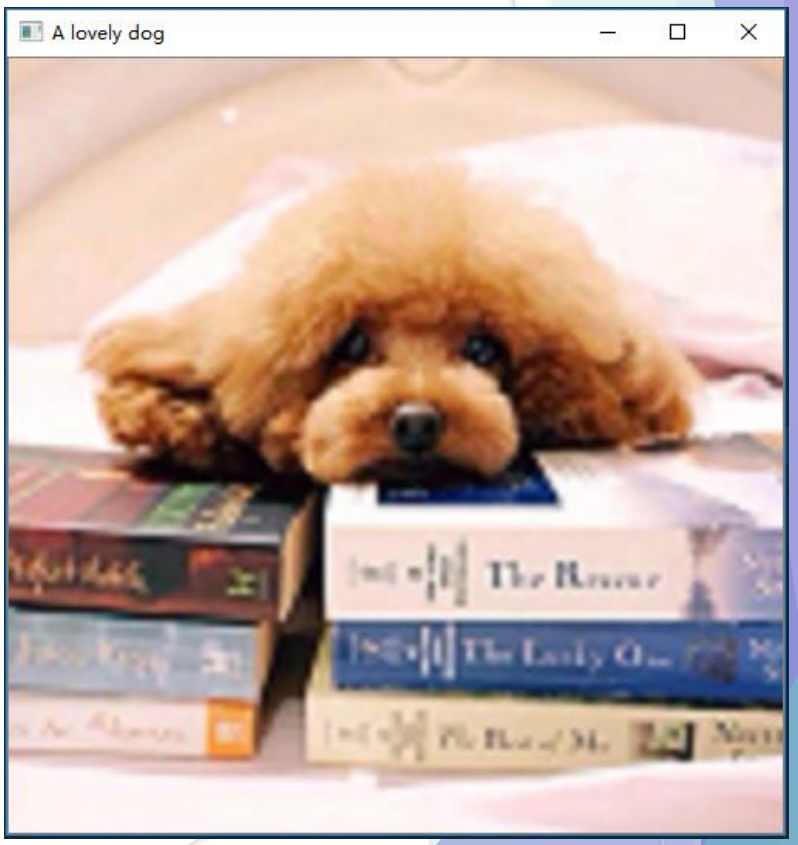
VideoCapture 类可以获得摄像头的帧流，参数则为摄像头的设备索引（device index），下段代码为捕捉摄像头 10 秒的视频信息，并将其写入一个avi文件中。

```
success, frame = cameraCapture.read()
numFrameRemaining = 10 * fps - 1
while success and numFrameRemaining > 0:
    videoWriter.write(frame)
    success, frame = cameraCapture.read()
    numFrameRemaining -= 1
cameraCapture.release()
```

# 2.1.6 在窗口显示图像

imshow(winname, mat)
winname - 窗口的名称
mat - 要显示的图像。

cv2.imshow('A lovely dog', img)



# 2.1.7 在窗口显示摄像头帧

- nameWindow() - 创建窗口
- imshow() - 显示窗口
- DestroyWindow() 销毁窗口
- waitKey() 获取键盘输入
- setMouseCallback() 鼠标输入

```
cv2.namedWindow('MyWindow')
cv2.setMouseCallback('MyWindow', clickMouse)
cv2.imshow('MyWindow', frame)
cv2.destroyWindow('MyWindow')
```

# 2.1.7 在窗口显示摄像头帧

对鼠标操作函数 setMouseCallback()

setMouseCallback(windowName, onMouse [, param])

windowName - 目标窗口名称

onMouse - 鼠标响应函数，回调函数名

param - 回调函数的参数，可选参数

回调函数中的参数可以取如下值(见教材p26)，它们对应不同的鼠标事件。

# 2.2 Cameo项目 (人脸跟踪与图像处理)

为了开发人脸跟踪和图像处理的交互式作用，需要实时获取摄像头的输入；该类的应用覆盖了 OpenCV 大部分的功能。

该应用：

实时进行脸部合并(facial merging)

滤波器和扭曲（distortion）技术

传统成像和深度成像的方法

应用程序命名为Cameo

# 2.3 Cameo —— 面向对象的设计

OpenCV 的图像 I/O 功能相似

CaptureManager 类和 WindowManager 类作为高级的 I/O 流接口

CaptureManager 来读取新的帧，并将帧分派到一个或多个输出中，这些输出包括静止的图像文件、视频文件及窗口

WindowManager 类使应用程序代码能以面向对象的形式处理窗口和事件

可扩展性，实现时可以不依赖OpenCV 的I/O。

# 2.3 Cameo —— 面向对象的设计

1.使用managers.CaptureManager提取视频流

2.使用managers.WindowManager抽象窗口和键盘

3.cameo.Cameo的实现

# 2.4 总结

1. 设计一个应用程序，基本功能包括：显示摄像头帧、监听键盘输入、记录截图或截屏。
2. 扩展应用程序：在视频帧开始与结束之间，插入图像滤波代码。
3. 应用程序集成：OpenCV支持的摄像头驱动程序，或其他摄像头驱动程序