Reader.py：

实现Reader类与测试

img属性存放读取的图片（Image对象）

result属性存放检测结果

import cv2  
import time  
  
import Image  
  
class Reader(object):  
  
 def \_\_init\_\_(self):  
  
 self.img = None  
 self.result = None  
  
 def read\_from\_file(self**,** path):  
 #指定路径读取图片  
 #input:  
 # path:图片路径  
 #读取成功则返回1，失败返回0  
  
 try:  
 pic = cv2.imread(path)  
 self.img = Image.Image(f'path:{path}'**,** pic)  
 return True  
 except:  
 return False  
  
 def read\_from\_camera(self):  
 #调用摄像头获取图片  
 #读取成功则返回1，失败返回0  
 try:  
 cap = cv2.VideoCapture(**0**)  
 \_**,** pic = cap.read()  
 self.img = Image.Image(f'camera:{time.time()}'**,** pic)  
 return True  
 except:  
 return False  
  
if \_\_name\_\_=="\_\_main\_\_":  
 #测试  
 #分别用两个方法获取图片并显示  
 reader1 = Reader()  
 reader2 = Reader()  
  
 ret1 = reader1.read\_from\_file(r"..\HP65W\_4.png")  
 ret2 = reader2.read\_from\_camera()  
  
 if ret1:  
 cv2.imshow("read\_from\_file"**,** reader1.img.binary)  
 else:  
 print("Cannot read image from file")  
  
 if ret2:  
 cv2.imshow("read\_from\_camera"**,** reader2.img.binary)  
 else:  
 print("Cannot read video from camera")  
  
 cv2.waitKey(**0**)  
 cv2.destroyAllWindows()

Image.py:

import cv2  
import numpy as np  
  
  
class Image(object):  
 def \_\_init\_\_(self**,** name**,** raw):  
 self.name = name  
 self.raw = raw # 原图  
 self.binary = None # 二值化增强的图片  
 self.is\_qualified = None # 是否合格  
 self.qrcode\_segmentation = [] # 二维码的分割  
 self.character\_segmentation = [] # 文字的分割  
 self.barcode\_segmentation=[] # 条形码的分割  
 self.\_to\_binary()  
 self.find\_square()  
  
 def \_to\_binary(self):  
 img = cv2.cvtColor(self.raw**,** cv2.COLOR\_BGR2GRAY)  
 # 局部二值化  
 block\_size = **25** const\_value = **10** local\_binary = cv2.adaptiveThreshold(img**, 255,** cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C**,** cv2.THRESH\_BINARY\_INV**,** block\_size**,** const\_value)  
 # binary = cv2.GaussianBlur(local\_binary, (3, 3), 0)  
 # binary = cv2.Canny(local\_binary, 100, 150)  
 self.binary = local\_binary  
  
 def find\_square(self**,** img=None**,** lmin=**300,** lmax=**2000**):  
 # edges = cv2.Canny(img, 100, 200)  
 if img is None:  
 img = self.binary.copy()  
 contours**,** hierarchy = cv2.findContours(img**,** cv2.RETR\_TREE**,** cv2.CHAIN\_APPROX\_SIMPLE)  
 hierarchy = hierarchy[**0**]  
 found = []  
  
 for i in range(len(contours)):  
 # find bounding box coordinates  
 k = i  
 c = **0** while hierarchy[k][**2**] != -**1**:  
 k = hierarchy[k][**2**]  
 c = c + **1** if c >= **1**:  
 found.append(i)  
 boxes = [] # 包围盒组  
  
 for i in found:  
 x**,** y**,** w**,** h = cv2.boundingRect(contours[i])  
  
 ratio = w / h  
 ratio\_grade = ratio  
 if not (**0.96** < ratio\_grade < **1.04**): # 筛去长宽比不合格的  
 continue  
 rect = cv2.minAreaRect(contours[i]) # 获得轮廓的最小外接矩形  
 box = cv2.boxPoints(rect)  
  
 length = lambda x: abs(x[**0**][**0**] - x[**2**][**0**]) + abs(x[**0**][**1**] - x[**2**][**1**]) # 以曼哈顿距离代替边长  
  
 if length(box) < lmin or length(box) > lmax: # 筛去边长太短的点  
 continue  
 box = np.int0(box)  
 boxes.append(box)  
 self.qrcode\_segmentation = boxes  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 #cv2.namedWindow("test", 0)  
 cv2.namedWindow("raw"**, 0**)  
 raw = cv2.imread(r'..\3-2.png')  
 image = Image('test'**,** raw)  
 print(image.qrcode\_segmentation)  
 cv2.imshow('raw'**,** image.raw)  
 # cv2.imshow('test', image.binary)  
 cv2.waitKey(**0**)