实验：特定姿态轮廓匹配

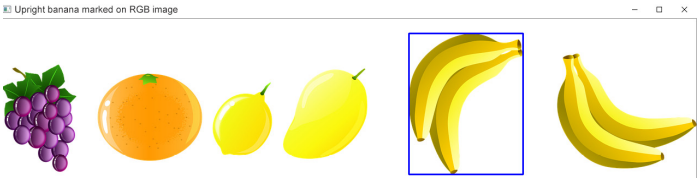
实验目标

在本实验中，您将继续以之前中所做的工作为基础。

此前，您在图像中检测到存在于多个水果间的两个香蕉束，但两者的姿态是不一致的：

- 一束处于水平位置
- 另一束处于垂直位置

在本实验中，您将检测到直立的香蕉束，如下所示：

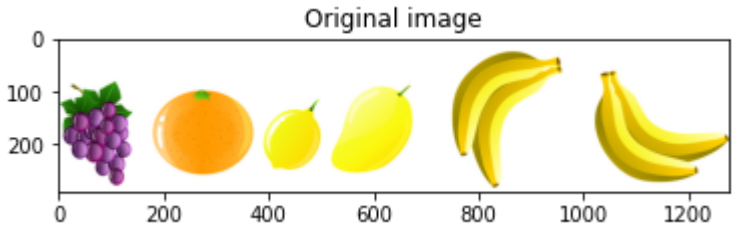


提示： 本实验的任务是在高度大于其宽度的香蕉束周围，绘制一个边界框。由于之前的实验扩展，因此，您首先需要恢复实验进度。

1. 恢复实验进度

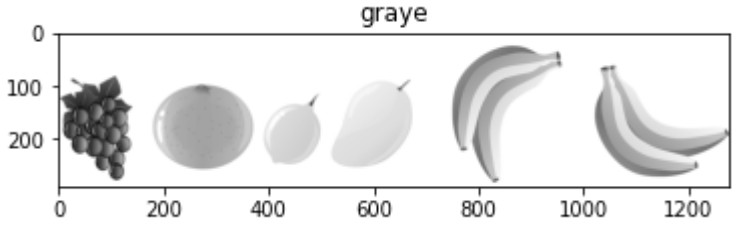
In [1]:

```
import cv2                          # 导入OpenCV
import matplotlib.pyplot as plt     # 导入matplotlib

# 魔法指令，使图像直接在Notebook中显示
%matplotlib inline

# 设置输入输出路径
import os
base_path = os.environ.get("BASE_PATH",'../data/')
data_path = os.path.join(base_path + "lab4/")
result_path = "result/"
os.makedirs(result_path, exist_ok=True)

# 读取本地图像
image = cv2.imread('./data/many fruits.png')
imagecopy= image.copy()                          # 拷贝原图像副本

# cv2.imshow( 'Original image' , image )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

plt.imshow(image[:,:,::-1])         # 将图像从 BGR 转换为 RGB
plt.title('Original image')         # 指定输出图像的标题
plt.show()                          # 显示图像
```



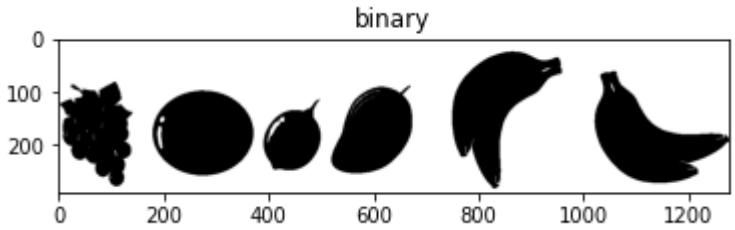
In [2]:

```
# 将图像转换为灰度图像
gray_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

# cv2.imshow( 'gray' , gray_image )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 使用灰色“喷涂”图像输出显示
plt.imshow(gray_image, cmap='gray')
# 指定输出图像的标题
plt.title('graye')
# 显示图像
plt.show()
```



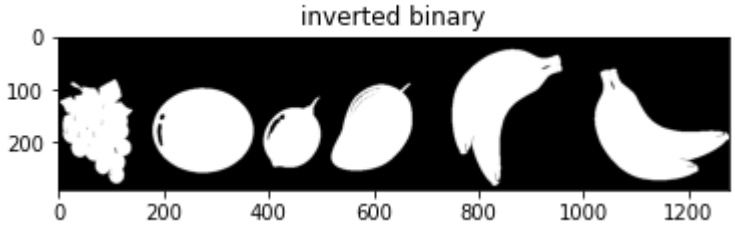
In [3]:

```
# 直接输入阈值与最大值，执行图像二值化
ret,binary_im = cv2.threshold(gray_image,245,255,cv2.THRESH_BINARY)

# cv2.imshow('binary', binary_im )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 使用灰色“喷涂”图像输出显示
plt.imshow(binary_im, cmap='gray')
# 指定输出图像的标题
plt.title('binary')
# 显示图像
plt.show()
```



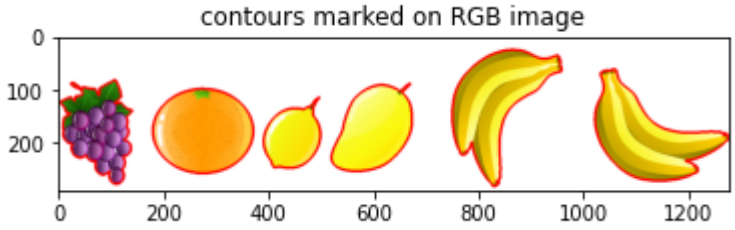
In [4]:

```
binary_im= ~binary_im # 反转图像（黑白像素互换）

# cv2.imshow('inverted binary', binary_im )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

 # 使用灰色“喷涂”图像输出显示
plt.imshow(binary_im, cmap='gray')
# 指定输出图像的标题
plt.title('inverted binary')
# 显示图像
plt.show()
```



In [5]:

```
# 从二值图像中找到外部轮廓
contours,hierarchy = cv2.findContours(binary_im,
                     cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

# 绘制轮廓
with_contours = cv2.drawContours(image,contours,-1,(0,0,255),3)

# cv2.imshow( 'contours marked on RGB image' , with_contours )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像从 BGR 转换为 RGB
plt.imshow(with_contours[:,:,::-1])
# 指定输出图像的标题
plt.title('contours marked on RGB image')
# 显示图像
plt.show()
```



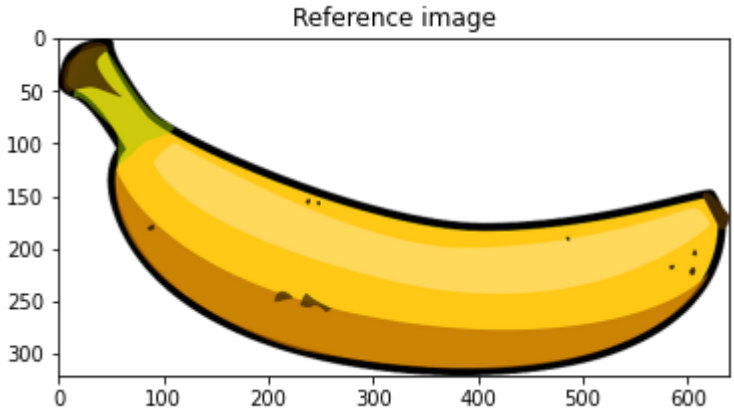
In [6]:

```
print('Total number of contours:')          # 设置输出提示
print(len(contours))                        # 设置输出提示
```

```
Total number of contours:
6
```

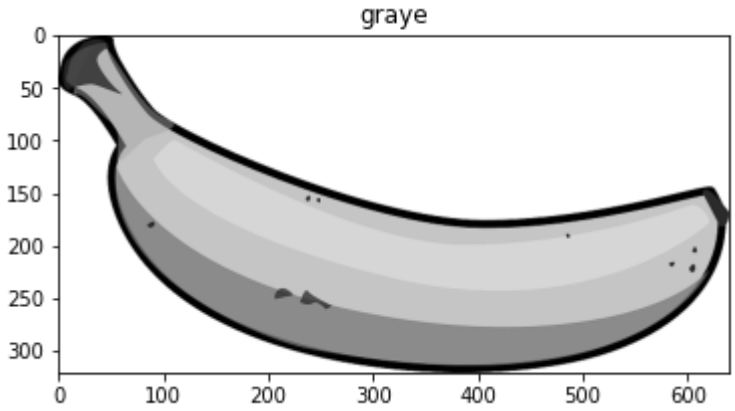
In [7]:

```
# 读取本地参考图像
ref_image = cv2.imread('./data/bananaref.png')

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'Reference image' , ref_image )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像从 BGR 转换为 RGB
plt.imshow(ref_image[:,:,::-1])
# 指定输出图像的标题
plt.title('Reference image')
# 显示图像
plt.show()
```



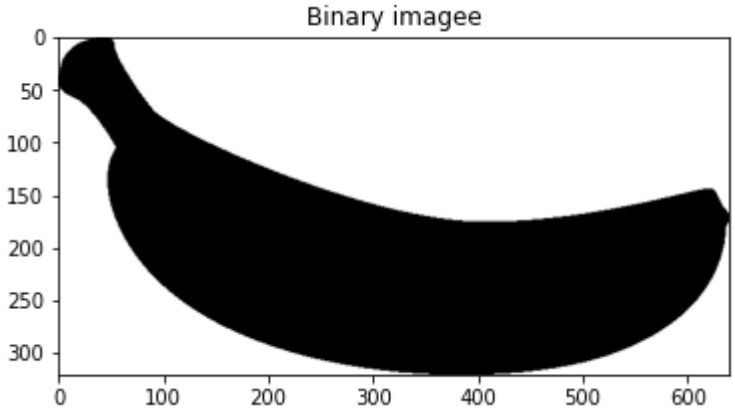
In [8]:

```
# 将图像转换为灰度图像
gray_image = cv2.cvtColor(ref_image,cv2.COLOR_BGR2GRAY)

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'Grayscale image' , gray_image )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 使用灰色“喷涂”图像输出显示
plt.imshow(gray_image, cmap='gray')
# 指定输出图像的标题
plt.title('graye')
# 显示图像
plt.show()
```



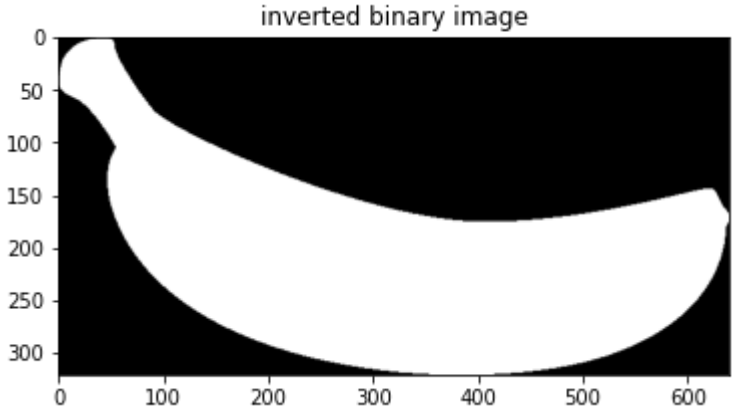
In [9]:

```
ret,binary_im = cv2.threshold(gray_image,245,255,cv2.THRESH_BINARY)

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'Binary image' , binary_im )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 使用灰色“喷涂”图像输出显示
plt.imshow(binary_im, cmap='gray')
# 指定输出图像的标题
plt.title('Binary imagee')
# 显示图像
plt.show()
```



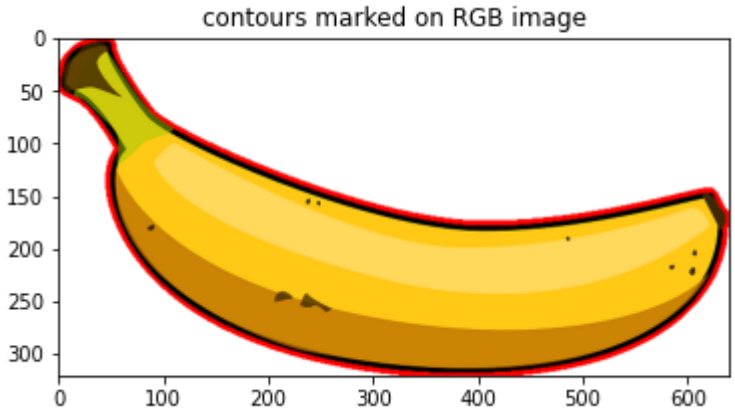
In [10]:

```
binary_im= ~binary_im  # 反转图像（黑白像素互换）

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'inverted binary image' , binary_im )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 使用灰色“喷涂”图像输出显示
plt.imshow(binary_im, cmap='gray')
# 指定输出图像的标题
plt.title('inverted binary image')
# 显示图像
plt.show()
```



In [11]:

```
# 从二值图像中找到外部轮廓
ref_contour_list,hierarchy = cv2.findContours(binary_im,
                                cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

# 绘制轮廓
with_contours = cv2.drawContours(ref_image,ref_contour_list,-1,(0,0,255),3)

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'contours marked on RGB image' , with_contours )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

plt.imshow(with_contours[:,:,::-1])          # 将图像从 BGR 转换为 RGB
plt.title('contours marked on RGB image')  # 指定输出图像的标题
plt.show()                                  # 显示图像
```



In [12]:

```
print('Total number of contours:')          # 设置输出提示
print(len(ref_contour_list))                # 设置输出提示
```

```
Total number of contours:
1
```

In [13]:

```
# 将ref_contour_list轮廓索引中的第一个轮廓（也是唯一的轮廓）
# 命名为reference_contour
# 在后面的轮廓匹配中，
# 用reference_contour与contours轮廓列表索引中的每一个轮廓进行匹配
reference_contour = ref_contour_list[0]

# 创建空列表，用于保存每次轮廓匹配后的数值差
dist_list= []
# 创建contours轮廓列表索引号码的for循环
for cnt in contours:
    # 按照contours轮廓列表索引号码提取轮廓，与reference_contour参考轮廓进行匹配
    # 其中，比较方法我们使用cv.CONTOURS_MATCH_I1
    # 而没用的parameter参数我们用0进行传递
    retval= cv2.matchShapes(cnt, reference_contour,1,0)

    # 将本次轮廓匹配数值差填入dist_list列表，之后执行下一次循环
    dist_list.append(retval)

print(dist_list) # 输出轮廓匹配数值差列表
```

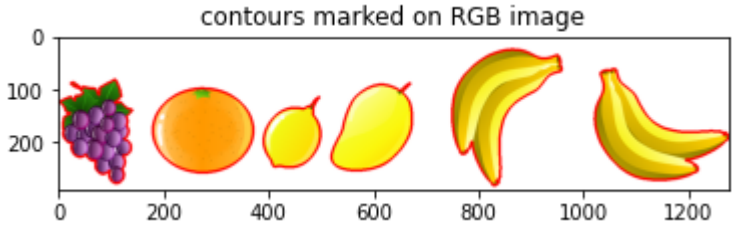
```
[1.9329380069362876, 1.6618014776874734, 1.771151523726344, 1.8150005423795166, 1.26
92130505335848, 1.269213050534173]
```

In [14]:

```
# 复制dist_list并将其存储在单独的变量中
sorted_list= dist_list.copy()
# 将列表从最小到最大排序
sorted_list.sort()

# 在原始 `dist_list` 列表中，找到存在最小距离和第二最小距离的索引编号
# 距离最小的索引号
ind1_dist= dist_list.index(sorted_list[0])
# 距离第二小的索引号
ind2_dist= dist_list.index(sorted_list[1])

# 初始化一个新的空列表，并将轮廓添加到以下两个索引处：
# 创建空列表
banana_cnts= []
# 将距离最小的索引号的轮廓附加到新建的空列表中
banana_cnts.append(contours[ind1_dist])
# 将距离第二小的索引号的轮廓附加到新建的空列表中
banana_cnts.append(contours[ind2_dist])
```

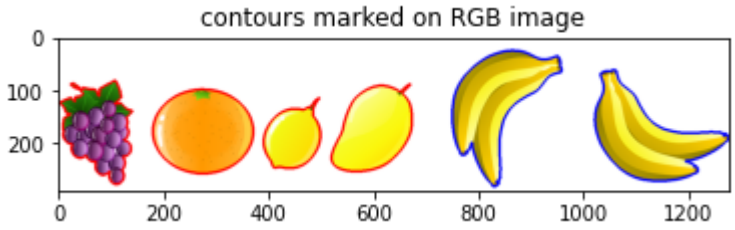
In [15]:

```
# 为了对比方便，先重新显示当前之前已经绘制过的 `image` 图像

# 将图像从 BGR 转换为 RGB
plt.imshow(image[:,:,::-1])
# 指定输出图像的标题
plt.title('contours marked on RGB image')
# 显示图像
plt.show()
```



In [16]:

```
# 在已绘制过一次红色轮廓的图像image上，
# 叠加绘制banana_cnts列表内的轮廓（只有香蕉），
# 使用蓝色轮廓线BGR(255,0,0)绘制：
with_contours = cv2.drawContours(image,banana_cnts,-1,(255,0,0),3)

# 在云实验环境下忽略以下代码，避免程序尝试打开系统窗口显示图片；
# 使用matplotlib替换，使图像直接在 Jupyter Notebook 中输出。

# cv2.imshow( 'contours marked on RGB image' , with_contours )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

plt.imshow(with_contours[:,:,::-1])         # 将图像从 BGR 转换为 RGB
plt.title('contours marked on RGB image')  # 指定输出图像的标题
plt.show()                                  # 显示图像
```



2. 检测直立姿态

这里我们通过轮廓的高度和宽度之比作为香蕉的姿态判断 ——

很明显：直立摆放的香蕉轮廓，**其高度大于宽度**；相反，水平摆置的香蕉轮廓，**其高度小于宽度。**

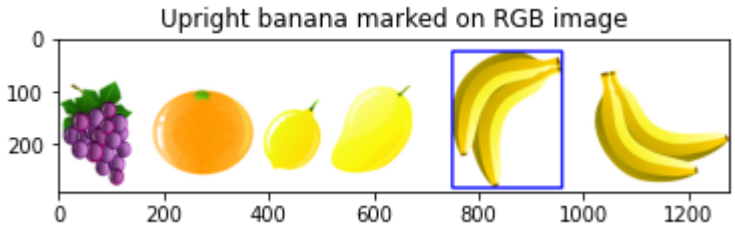
使用 `for` 循环逐个检查这两个轮廓。

嵌入 `if` 条件判断：如果检测到轮廓的 `高度` 大于 `宽度` ，可判断为直立摆放的香蕉。

那么，在之前保存的图像副本 `imagecopy` 上在其周围绘制一个蓝色边框：

In [17]:

```
# 使用for循环逐个轮询banana_cnts内的轮廓（实际只有两个）
for cnt in banana_cnts:
    # 将轮廓传递到cv2.boundingRect函数中来获取每一个轮廓的x/y/w/y参数
    x,y,w,h = cv2.boundingRect(cnt)
    # 如果高度大于宽度
    if h>w:
        # 在之前保存的原始彩色图像的副本上绘制此边界框
        cv2.rectangle(imagecopy,(x,y),(x+w,y+h),(255,0,0),3)

# cv2.imshow( 'Upright banana marked on RGB image' , imagecopy )
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# 将图像从 BGR 转换为 RGB
plt.imshow(imagecopy[:,:,::-1])
# 指定输出图像的标题
plt.title('Upright banana marked on RGB image')
# 显示图像
plt.show()
```



当您了解了我们如何利用轮廓特征来判断香蕉是直立摆置后，便很容易便能将水平放置的香蕉也找出来。

实验小结

在本实验中，我们对本章所有新获得的技能进行了练习 —— 执行了轮廓的检测，按层次结构访问它们，进行轮廓匹配，并使用轮廓的宽度和高度过滤轮廓。

通过本实验，您对轮廓检测和轮廓匹配的各种应用场景有了进一步的理解。