**void findContours**//提取轮廓，用于提取图像的轮廓

(

InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset = Point()

)

①输入图像，必须是8位单通道图像，并且应该转化成二值的；

②检测到的轮廓，每个轮廓被表示成一个point向量；

③可选的输出向量，包含图像的拓扑信息。其中元素的个数和检测到的轮廓的数量相等；

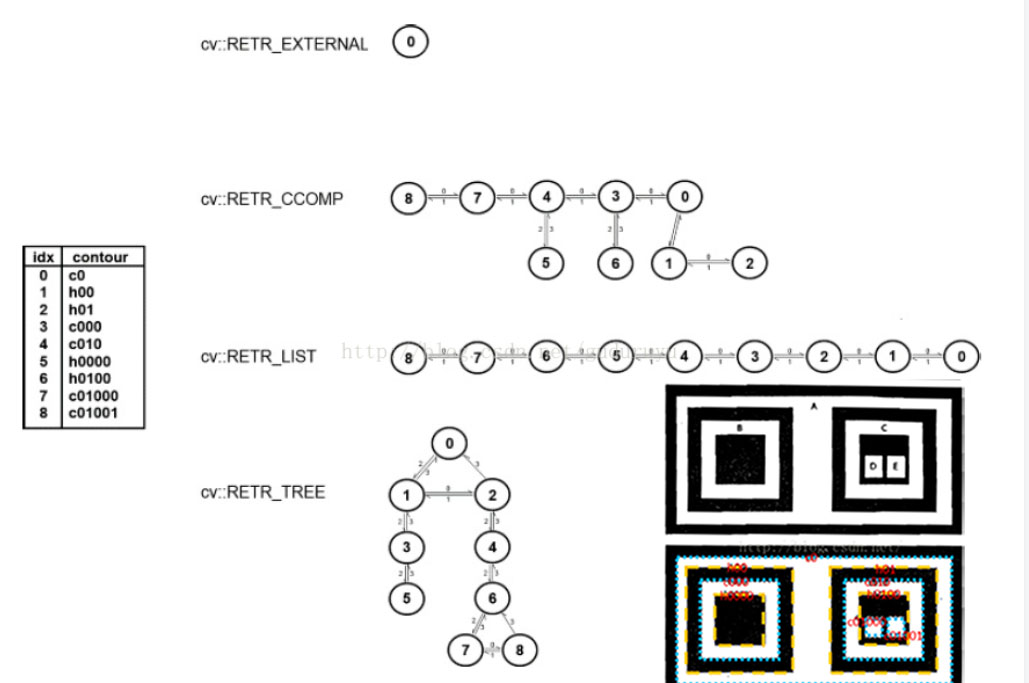
④说明需要的轮廓类型和希望的返回值方式

CV\_RETR\_EXTERNAL 只检测出最外轮廓即c0。图中第一个轮廓指向最外的序列，除此之外没有别的连接。

CV\_RETR\_LIST 检测出所有的轮廓并将他们保存到表(list)中，图中描绘了这个表，被找到的9条轮廓相互之间由h\_prev和h\_next连接。这里并没有表达出纵向的连接关系，没有使用v\_prev和v\_next.

CV\_RETR\_COMP 检测出所有的轮廓并将他们组织成双层的结构，第一层是外部轮廓边界，第二层边界是孔的边界。从图可以看到5个轮廓的边界，其中3个包含孔。最外层边界c0有两个孔，c0之间的所有孔相互间由h\_prev和h\_next指针连接。

CV\_RETR\_TREE 检测出所有轮廓并且重新建立网状的轮廓结构。图中，根节点是最外层的边界c0，c0之下是孔h00，在同一层中与另一个孔h01相连接。同理，每个孔都有子节点(相对于c000和c010)，这些子节点和父节点被垂直连接起来。这个步骤一直持续到图像最内层的轮廓，这些轮廓会成为树叶节点。



⑤轮廓近似方法

CV\_CHAIN\_CODE 用freeman链码输出轮廓，其他方法输出多边形(顶点的序列)。

CV\_CHAIN\_APPROX\_NONE将链码编码中的所有点转换为点。

CV\_CHAIN\_APPROX\_SIMPLE压缩水平，垂直或斜的部分，只保存最后一个点。CV\_CHAIN\_APPROX\_TC89\_L1，CV\_CHAIN\_QPPROX\_TC89\_KCOS使用Teh-Chin链逼近算法中的一个。

CV\_LINK\_RUNS与上述的算法完全不同，连接所有的水平层次的轮廓。

**void drawContours**//绘制轮廓，用于绘制找到的图像轮廓

(

InputOutputArray image, InputArrayOfArrays contours, int contourIdx, const Scalar& color, int thickness = 1, int lineType = 8, InputArray hierarchy = noArray(), int maxLevel = INT\_MAX, Point offset = Point()

)

①要绘制轮廓的图像；

②所有输入的轮廓，每个轮廓被保存成一个point向量；

③指定要绘制轮廓的编号，如果是负数，则绘制所有的轮廓；

④绘制轮廓所用的颜色；

⑤绘制轮廓的线的粗细，如果是负数，则轮廓内部被填充；

⑥绘制轮廓的线的连通性；

⑦关于层级的可选参数，只有绘制部分轮廓时才会用到；

⑧绘制轮廓的最高级别，这个参数只有hierarchy有效的时候才有效

maxLevel=0，绘制与输入轮廓属于同一等级的所有轮廓即输入轮廓和与其相邻的轮廓

maxLevel=1, 绘制与输入轮廓同一等级的所有轮廓与其子节点。

maxLevel=2，绘制与输入轮廓同一等级的所有轮廓与其子节点以及子节点的子节点

例程：

void main()

{

Mat image = imread("test.png");

Mat gray;

cvtColor(image, gray, CV\_BGR2GRAY);

GaussianBlur(gray, gray, Size(3, 3), 3, 3);

Mat img;

threshold(gray, img, 0, 255, THRESH\_OTSU);

Mat img1;

img.copyTo(img1);

vector<vector<Point>> contours;

vector<Vec4i> hierarchy;

findContours(img, contours, hierarchy, CV\_RETR\_CCOMP, CV\_CHAIN\_APPROX\_NONE);

Mat resultImage = Mat ::zeros(img.size(),CV\_8U);

drawContours(resultImage, contours, -1, Scalar(255, 0, 255));

return;

}