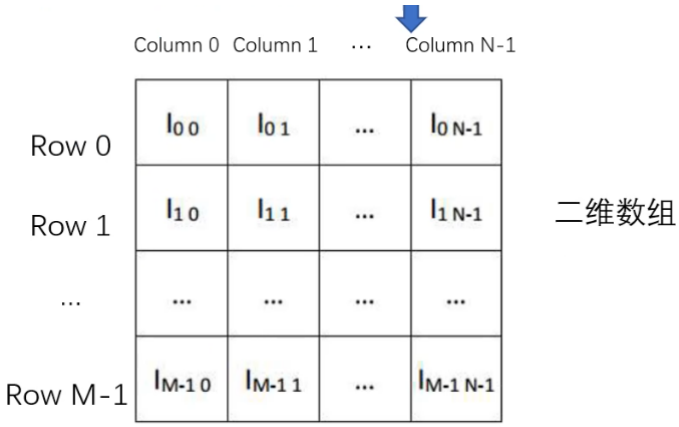
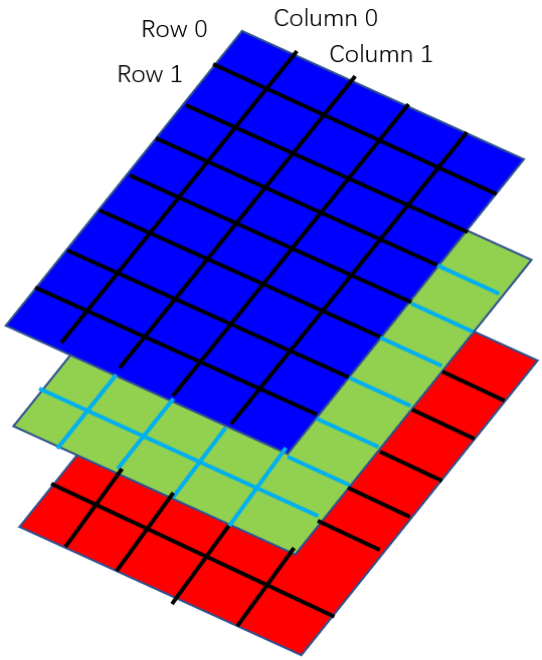
认识图像

灰度图



> 单通道：
>
> 255-白色
>
> 0-黑色
>
> 存储（二维数组）：
>
> 矩阵形式（M*N）—— M-行 N-列
>
> 从0开始索引

彩色图像



	Column 0			Column 1			Column …			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	…	…	…	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	…	…	…	1, m	1, m	1, m
Row …	…,0	…,0	…,0	…,1	…,1	…,1	…	…	…	…, m	…, m	…, m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,…	n,…	n,…	n, m	n, m	n, m

> 三个通道（蓝-B、绿-G、红-R）
>
> 存储形式：
>
> 三维数组
>
> 矩阵的每个位置含三个值

装载，显示和存储图像

三个常用头文件

> - #include <opencv2/core.hpp>
>
> **core.hpp 核心功能模块** 。主要包含了opencv基本数据结构，动态数据结构，绘图函数，数组操作相关函数，辅助功能与系统函数和宏。
>
> - #include <opencv2/imgproc.hpp>
>
> **imgproc.hpp 图像处理模块** 。主要包换了图像的变换，滤波直方图相关结构分析，形状描述 。
>
> - #include <opencv2/highgui.hpp>
>
> **highgui.hpp 高层**GUI图像交互模块** 。主要包换了图形交互界面，媒体I/O的输入输出，视频信息的捕捉和提取，图像视频编码等。

cv::Mat 结构

> cv::Mat 相当于一个数据类

```
cv::Mat image;   //image是图片变量，cv::Mat是数据类型

// 头部
int cols;       //图像列数
int rows;       //图像行数
int channels;   //图像通道数

uchar *data     // 指向存储图像数据块的指针
```

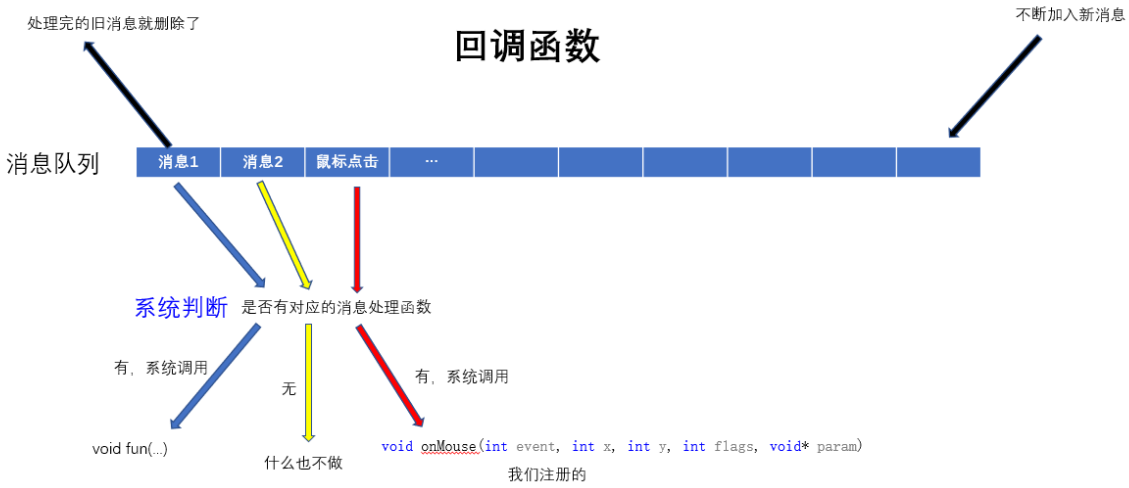
装载图像

```
// 创建一个空图像
    cv::Mat image;

// 检查图像尺寸
    image.cols
    image.cols

// 检查图像通道数
    image.channels()

// 读取输入图像
    image = cv::imread("1.JPG", cv::IMREAD_GRAYSCALE); // 单通道灰度图像
    image = cv::imread("1.JPG", cv::IMREAD_COLOR); // 三通道彩色图像
    image = cv::imread("1.JPG"); // 第二个参数为空，按图像的原本情况读入

// 错误处理
    image.empty() // 如果图像是空，返回1
```

显示，存储图像

```
// 定义窗口
    cv::namedWindow("original Image"); //窗口名，为了区分不同窗口

// 显示图像
    cv::imshow("original Image", image);

// 创建另一个空的图像（用作输出图像）
    cv::Mat result;

// 翻转图像
    cv::flip(image, result, 1); // image:输入图像；result:输出图像；int：翻转水平 正数-水平翻转，0-垂直翻转，负数-水平和垂直都翻转

// 保存图像到本地磁盘，默认路径即源代码所在位置
    cv::imwrite("output.JPG", result);

//控制台窗口，会在main函数结束时关闭（图像直接关闭），下面的函数可以等待用户按下任意键后再结束程序
    cv::waitKey(0); // 0--表示永远地等待按键（默认值），直到用户按下键盘任意键
                    // 键入正数表示等待的毫秒数
```

在图像上点击



> 原则：先进先出
>
> 消息：鼠标、键盘等的操作

```
// 注册鼠标点击的回调函数
    cv::setMouseCallback(
            "original Image",
            onMouse,
            reinterpret_cast<void*>(&image));

// 回调函数
    void* //作为指针表示万能指针，即任何类型指针都可以转化为void指针，转化后的指针也可以通过类型转换转换回去
    void* param //指向任意对象的指针，作为附加的参数发送给函数

    reinterpret_cast<cv::Mat*>(param) //强制类型转换符
```

```
    static_cast<new_type>(ecpression)
        // static_cast支持指向基类的指针和指向子类的指针之间的互相转换；
        // static_cast跟传统转换方式几乎一致

    cv::EVENT_LBUTTONDOWN: // 鼠标左键按下事件
    cv::EVENT_RBUTTONDOWN: // 鼠标右键按下事件
    cv::EVENT_MOUSEMOVE: // 鼠标移动事件

     -> // 重载箭头操作符

    at的用法
         image.at(i,j) // 灰度图 i行j列
         image.at(i,j)[k] // 彩色图 k通道 i行j列
```

在图像上绘图

```
// 画圆
    cv::circle(
        image,                  // 目标图像
        cv::Point(155, 110),    // 圆心坐标
        65,                     // 半径
        0,                      // 颜色（此处黑色）
        3                       // 厚度
    );

    // 图像上标注文本
    cv::putText(
        image,                  // 目标图像
        "This is a dog.",       // 文本
        cv::Point(40, 200),     // 文本位置（左上角）
        cv::FONT_HERSHEY_PLAIN, // 字体类型
        2.0,                    // 字体大小
        255,                    // 字体颜色（此处白色）
        2                       // 文本厚度
    );
```

创建图像

```
/* 灰度图像的创建 */
    // 创建一个240行 *320列的新灰度图像
    // 1
    cv::Mat image1(240, 320, CV_8U, 100); // 所有点像素都为100
    // 2
    cv::Mat image1(240, 320, CV_8U, cv::Scalar(100)); // scalar--定义最多四个元素的
向量值
    // 3
    cv::Mat image1(240, 320, CV_8U);
    image1 = 100;

/* 彩色图像的创建 */

    // 创建一个红色的彩色图像，用cv::size(320, 240)以提供图像的尺寸信息
    // 1
```

```
    cv::Mat image2(cv::Size(320, 240), CV_8UC3, cv::Scalar(0, 0, 255)); // 通道次序为BGR  Size-->可有两个参数(平面)也可有三个(三维立体) 常用
    // 2
    cv::Mat image2(cv::Size(320, 240), CV_8UC3);
    image2 = cv::Scalar(0, 0, 255);
```

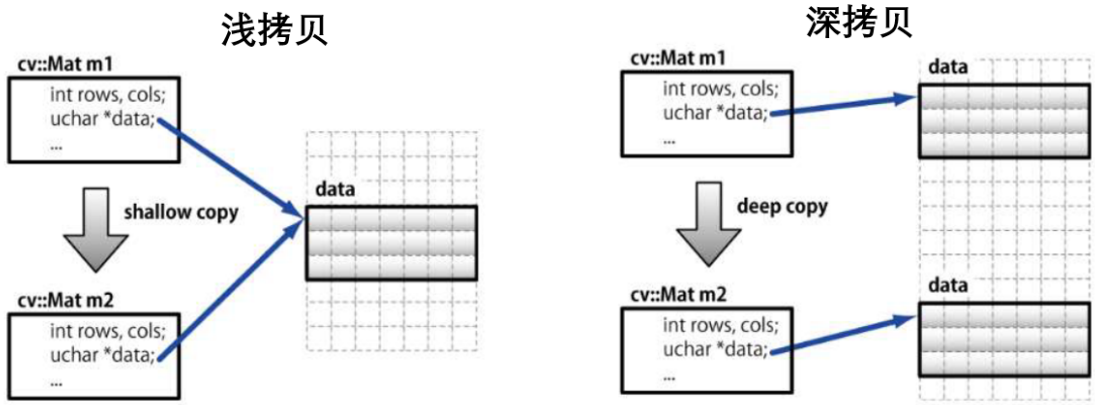
图像类型

```
// U: 表示无符号整数 unsigned int
// S: 表示有符号整数 signed int
// F: 表示浮点数 float

// C: 表示通道数 channel
// 无C/C1: 单通道，灰度图像
// C3: 三通道，彩色图像（BGR）

// 8bit(位) = 1Byte(字节)
// 8bit: char
// 16/32bit: int
// 32/64bit: float
```

图像的重分配

```
// 重新分配一个新的图像（原来的内容会先被释放）
// 如果新的尺寸和类型与原来相同，就不会重新分配内存
// 相当于修改原图像大小或类型
image1.create(200, 200, CV_8U);
image1 = 200; // 初始化图像
```

深拷贝和浅拷贝




```
cv::Mat image3 = cv::imread("1.JPG");

// 浅拷贝，所有图像都指向一个数据块
    cv::Mat image4(image3); // 浅拷贝1
    image1 = image3; // 浅拷贝2

// 深拷贝，这些图像是源图像的副本
    image3.copyTo(image2); // 深拷贝1
    cv::Mat image5 = image3.clone(); // 深拷贝2
```

从函数中获取图像

```
// 从函数中获取一个灰度图
    cv::Mat gray = getImg(); // 调用自己定义的函数

// 自己定义函数
    cv::Mat getImg()
    {
        // 创建图像
        cv::Mat ima(500, 500, CV_8U, 50);
        //返回图像
        return ima;
    }
```

图像的转换

```
//作为灰度图读入
    cv::Mat image6 = cv::imread("1.JPG", cv::IMREAD_GRAYSCALE);
    cv::Mat image7;

//转换成浮点型图像，像素强度范围[0, 1]，两幅图像的通道数量必须相同
    // 1
    image6.convertTo(
        image7, // 输出图像 如果是image6，则在原始图像上直接修改
        CV_32F, // 图像类型 rType
        1 / 255.0, // 缩放比例 α
        0.0 // 偏移量 β
    );
    // 2
    image7(x, y) = saturate_cast<rType>(α * image6(x, y) + β)   //
cv::saturate_cast函数在类型转换的同时，会把小于0的数值调整为0，大于255的值调整为255
```

处理小矩阵

```
// 3*3双精度型矩阵1
    cv::Matx33d matrix( // 并不是所有矩阵都可以这样定义
        3.0, 2.0, 1.0,
        2.0, 1.0, 3.0,
        1.0, 2.0, 3.0
    );

// 3*3双精度型矩阵2  最原始的定义方式
    cv::Matx<double, 3, 3> matrix1(
        3.0, 2.0, 1.0,
        2.0, 1.0, 3.0,
        1.0, 2.0, 3.0
    );

// 3*1矩阵1（即列向量）
    cv::Matx31d vector(5.0, 1.0, 3.0);
//3*1矩阵2
    cv::Matx<double, 3, 1>

// 矩阵相乘
    cv::Matx31d result = matrix * vector; //(3 x 3) * (3 x 1)
```

定义感兴趣的区域（ROI）

```
// 在图像的右下角定义一个ROI（初始化），放置logo的位置
// ROI 实际上就是一个cv::Mat对象，它与它的父图像指向同一个数据缓冲区，并在头部指明ROI的坐标

 // 定义ROI1
    // 初始化的时候直接赋值
    cv::Mat imageROI(
        image,                                                          // 读入的图像
        cv::Rect(image.cols - logo.cols, image.rows - logo.rows,    // ROI左上角
坐标
            logo.cols, logo.rows));                                     // ROI大小
（宽度和高度）
// 定义ROI2
    // 在图像的右下角定义一个ROI（赋值）
    imageROI = image(
        cv::Rect(image.cols - logo.cols, image.rows - logo.rows,    // ROI左上角
坐标
                 logo.cols, logo.rows));                                // ROI大小
（宽度和高度）

// 定义ROI3
    // ROI 还可以用行和列的值域来描述
    imageROI = image(cv::Range(image.rows - logo.rows, image.rows)    // ROI 行
（高度）范围
                         cv::Range(image.cols - logo.cols, image.cols));  // ROI
列（宽度）范围

// 插入logo，将logo这幅图像拷贝到image上定义的一个ROI（imageROI）中去
    logo.copyTo(imageROI);

// 使用图像掩码
    // 把标志作为掩码（必须是灰度图像）
    cv::Mat mask(logo); // 浅拷贝，mask和logo指向同一幅图像
    // 插入标志，只复制掩码mask中不为0的位置
    logo.copyTo(imageROI, mask);
    // logo/mask是同一幅图像，都是二值图像，即像素不是0就是255，只由黑白构成
    // 这里相当于只复制logo中值为255的像素
```