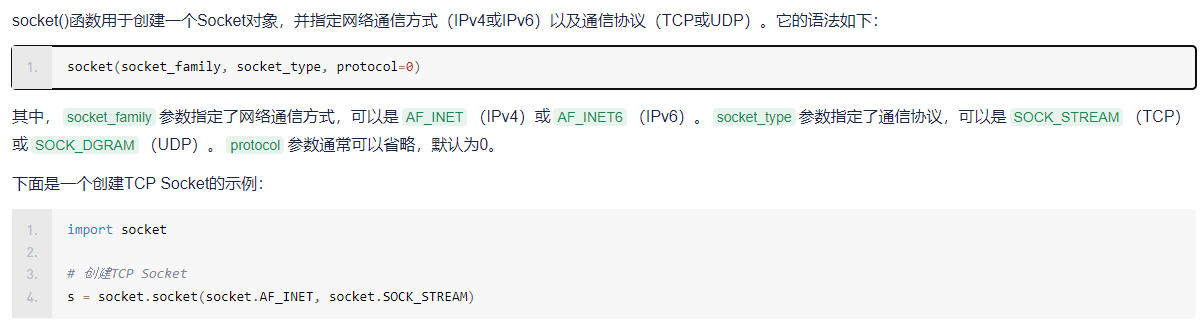
**Python实现**

**发送数据：**

RTSP：实时流传输协议（Real Time Streaming Protocol，RTSP），是TCP/IP协议体系中的一个应用层协议。该协议定义了一对多应用程序如何有效地通过IP网络传送多媒体数据。RTSP在体系结构上位于RTP和RTCP之上，它使用TCP或UDP完成数据传输。HTTP与RTSP相比，HTTP请求由客户机发出，服务器作出响应；使用RTSP时，客户机和服务器都可以发出请求，即RTSP可以是双向的。RTSP是用来控制声音或影像的多媒体串流协议，并允许同时多个串流需求控制，传输时所用的网络通讯协定并不在其定义的范围内，服务器端可以自行选择使用TCP或UDP来传送串流内容，它的语法和运作跟HTTP 1.1类似，但并不特别强调时间同步，所以比较能容忍网络延迟。

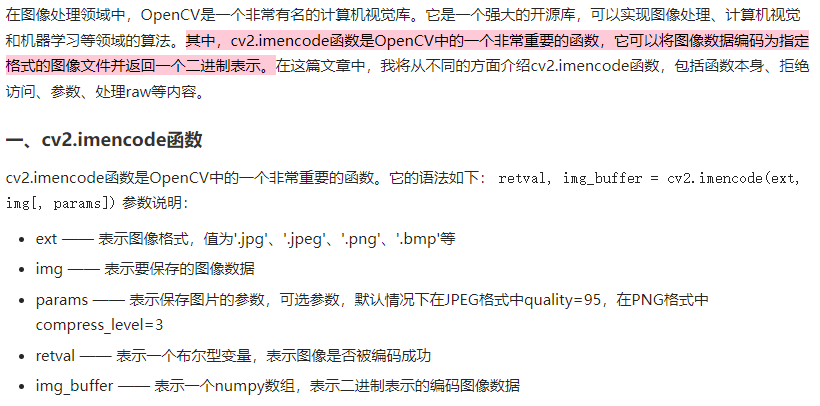
在网络编程中，Socket是一种通信方式，它可以在不同的计算机之间建立连接，进行数据传输。

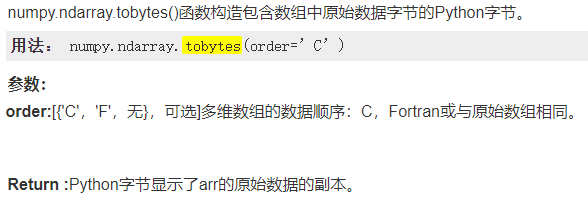




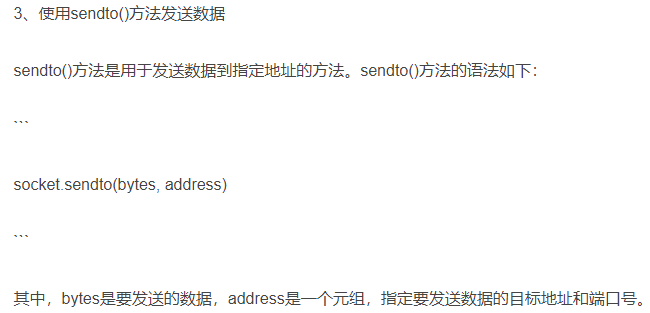


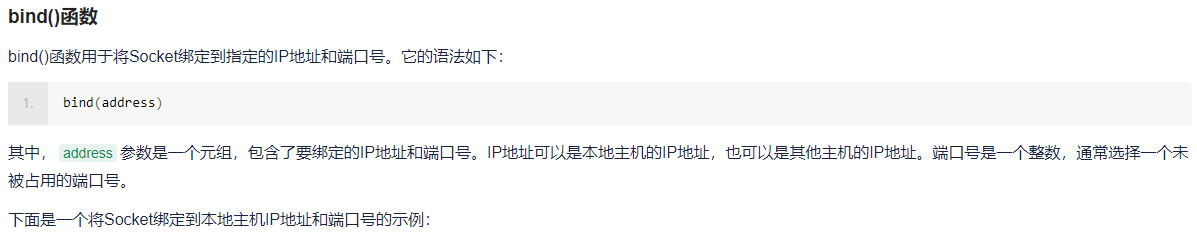




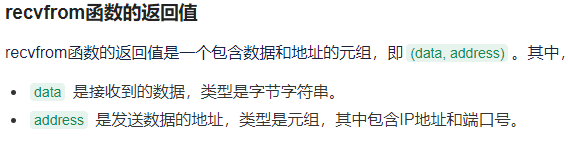


struct.pack则是将输入的值根据对应的格式进行压缩，并返回对应压缩后的二进制串。

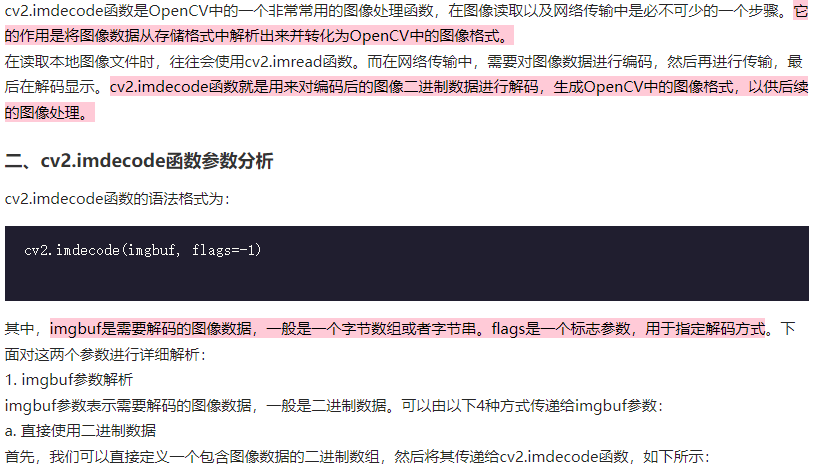


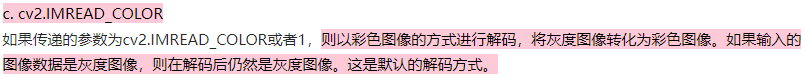




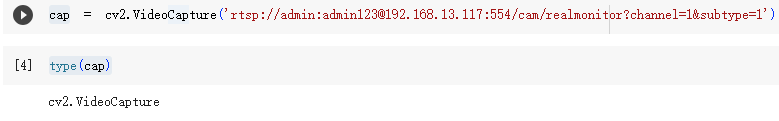








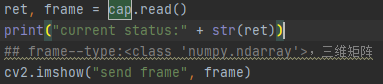
**连接摄像头**



**配置UDP服务器**

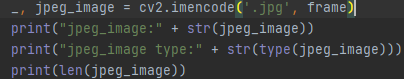


**获取帧并展示**



其中frame是返回的捕捉到的帧，frame的类型为<class 'numpy.ndarray'>，frame是一个三维矩阵，其维度为print(frame.shape) = (288, 352, 3)

**将图像数据（frame--三位矩阵）编码为指定格式的图像文件并返回一个二进制表示的numpy一维数组**





注意：每个图片的len(jpeg\_image)是不同的

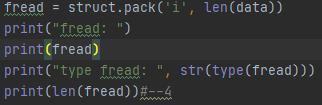
**将二进制的一维数组转换为原一维数组副本的python字节表示**

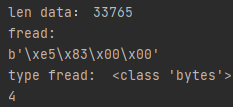




data的数据类型为<class 'bytes'>，展示的是字节码的长度

**定义文件头、数据**







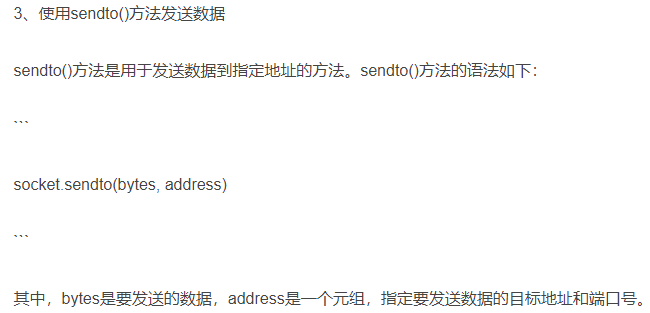
struct.pack则是将输入的值（len(data)）根据对应的格式进行压缩，并返回对应压缩后的二进制串。

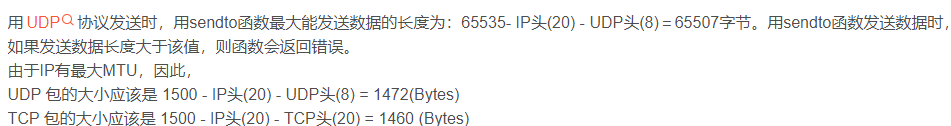
注意：同一图片的print(len(jpeg\_image))与print("len data： " + str(len(data)))的数值应该是相同的

此时fread只有四个字节

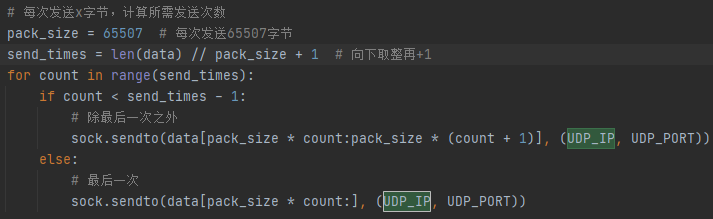
**发送文件头**



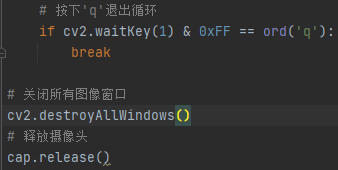


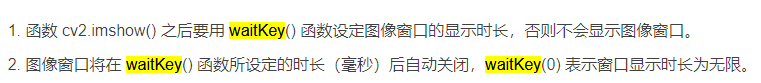


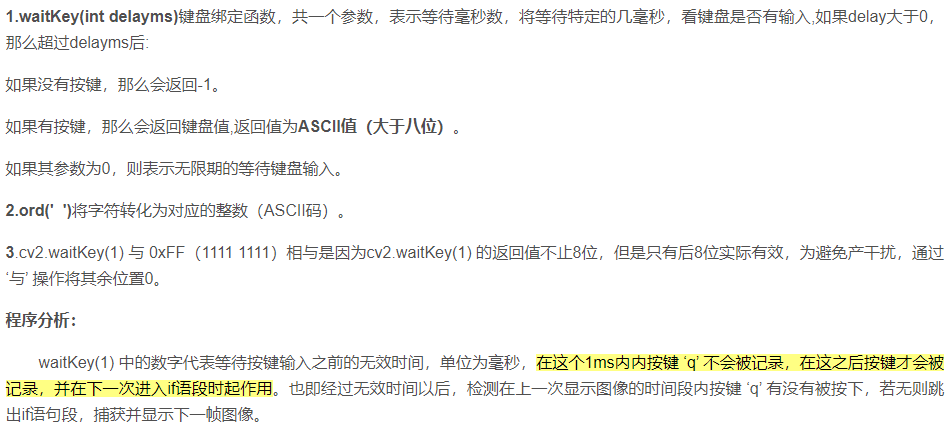
**发送数据**



**停止播放视频流：**



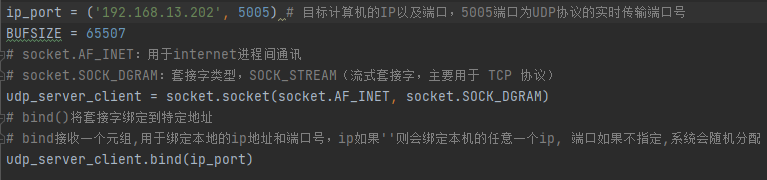






**接收数据：**

**配置UDP服务器**



创建socket对象，并指定网络通信方式和通信协议，socket是一种通信方式，socket()函数用于创建一个socket对象

用bind()函数用于将Socket绑定到指定的IP地址和端口号





**获取数据头信息**



其中，recvfrom函数是用于从套接字接收数据的方法。

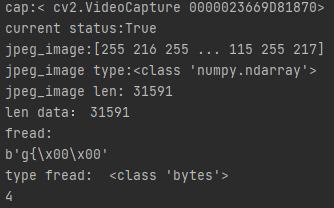
recvfrom函数接收指定大小的数据，并返回一个包含数据和地址的元组。其中数据是一个字节字符串，地址是发送数据的地址。

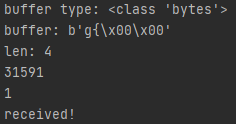
recvfrom函数的返回值是一个包含数据和地址的元组，即(data, address)。其中，

data 是接收到的数据，类型是字节字符串。

address 是发送数据的地址，类型是元组，其中包含IP地址和端口号。

通过查看结果：





可以发现buffer, \_ = udp\_server\_client.recvfrom(BUFSIZE)接收到的数据buffer与发送端的fread相同，说明此时接收到了文件头。

buffer = b’g{\x00\x00’

fread = b’g{\x00\x00’

与此同时，可以发现，data\_size = struct.unpack('i', buffer)[0]中的：

jpeg\_image，data，data\_size的大小相同

jpeg\_image：len--31591 type:<class 'numpy.ndarray'>

data : len--31591 type:<class 'bytes'>

data\_size: len--31591 type:<class 'int'>

关于struct.unpack函数：它的作用是将以指定格式编码的二进制数据解码成Python中的元组或其他合适的数据类型。

**接收完文件头之后需要接收发送的数据**



b" "的作用是：b后面””中的字符串是bytes 类型

data就是发送端发送的data，都是字节字符串，由于len(data)<BUFSIZE，所以一张图片只需要发送一次接收一次

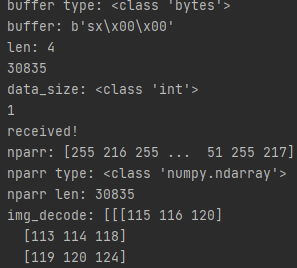
由于一张图片只需要发送一次接受一次，所以data\_total就是data

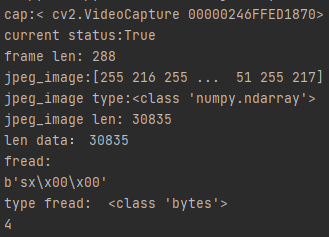


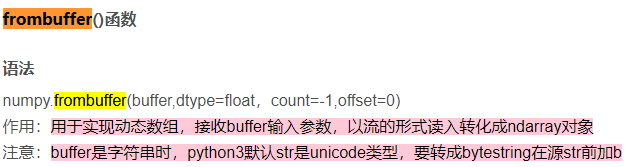
此处判断接收到的数据的字节长度是否与发送的数据的字节长度相等

**解码并显示接收的图像**









首先，通过np.frombuffer()将data\_total（字节字符串）进行解码，并将其转化为一维数组，可以发现：

len(nparr) = len(jpeg\_image)，也就是说此时把字节字符串返回到了一维数组表示

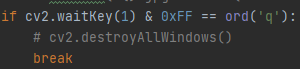
其次，通过cv2.imdecode将图像数据从存储格式中解析出来并转化为OpenCV中的图像格式。

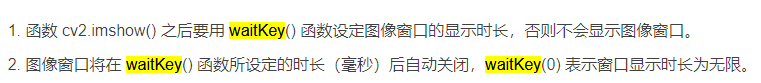
可以发现：

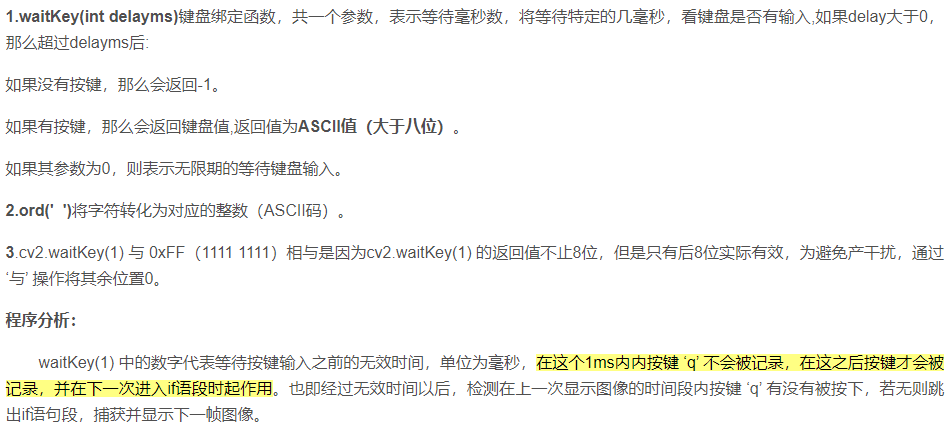
len(img\_decode) = len(frame)，也就是说此时把一维的数组返回到了三维数组表示

最后再通过cv2.imshow()函数将图像展示

**停止播放视频流：**



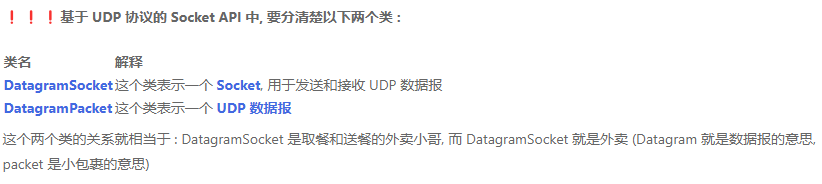




**Java接收视频数据**

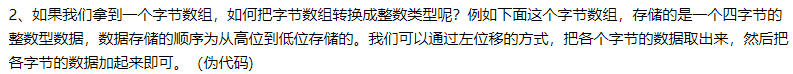
将python接收视频流移植到java端的时候需要参考以下链接

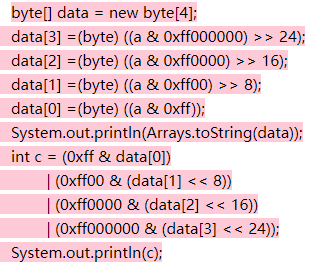
1. <https://blog.csdn.net/qq_38628046/article/details/130756837>
2. <https://blog.csdn.net/swadian2008/article/details/123273552>
3. <https://blog.51cto.com/u_16175447/6752069>

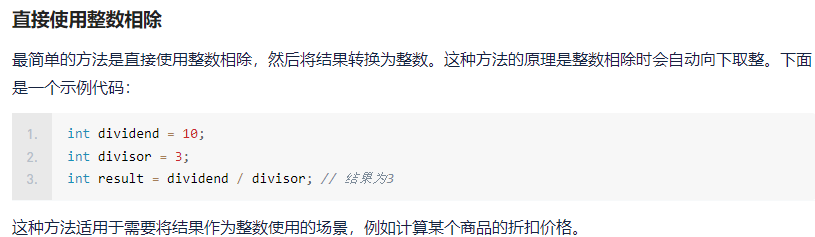






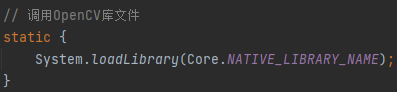




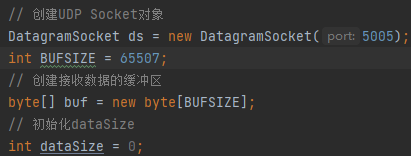


Java实现图像流数据接收

**调用OpenCV库的文件**



**创建UDP socket对象并做一些初始化**



创建一个UDP socket对象，用于接收UDP数据报，同时也需要端口号为5005.

创建一个接收数据的缓冲区，大小为65507.

初始化dataSize

**接收数据头并获取数据的字节大小**



创建接收数据头的数据包，接收到的数据被存储在buf（第一个参数：字节数组）中，需要指定length（第二个参数，指定缓冲区的大小）

ds.receive(receiveHead)为接收数据报

由于在在python中实际先发送一次数据头，因此先接收数据头（四个字节大小），所以getLength为获取的数据报中的数据的实际长度（长度为4）

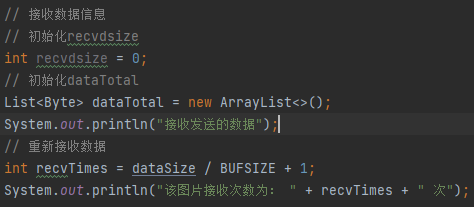
接收到的数据被存储在buf中，虽然缓冲区的大小被设置为65507，但是实际接收到的数据只有4个字节，因此根据切片方法只获取缓冲区中的前4个字节的数据。

此时的buffer变量为包含4个字节的字节数组

我们需要将这个包含4个字节的字节数组转化为int整数，才能知道发送来的图片的大小为多少字节，所以我们得到的应该是一个代表图片字节数量的int整数

如果接收到的数据的长度不为4个字节，那么说明数据传输有误。

**初始化recvdsize和dataTotal，并且获取了单张图片的接收次数**



初始化recvdsize

初始化dataTotal集合，其中的每个元素代表Byte类型的对象

int recvTimes = dataSize / BUFSIZE + 1：将传输的数据的字节数除以缓冲区大小（相除向下取整），这样就得到了一张图片需要的接收次数

**根据得到的接收次数对数据进行接收**



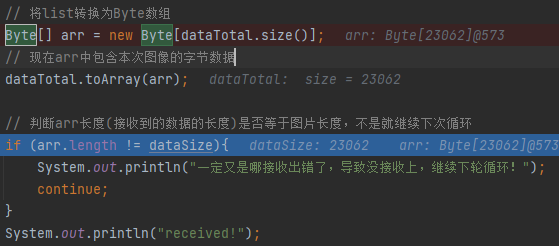
**DatagramPacket reciveData = new DatagramPacket(buf, BUFSIZE);**

**ds.receive(reciveData);**

每一次接收都需要创建接收数据的数据包，并且通过**ds.receive(reciveData)**接收数据报

获取数据报中数据的实际长度并遍历其中的每一个字节，将这些字节存储到dataTotal中，这样就把全部的数据字节存入了dataTotal中，recvdsize就是单张图片的全部字节长度

**将Byte集合转化为Byte[]，判断接收的数据长度与文件头接收的数据长度是否相等**

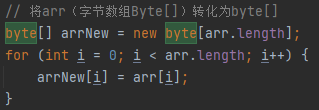


将dataTotal（一个包含Byte对象的集合）转化为Byte[]

现在arr中包含的是本次传输的图片的所有字节数据

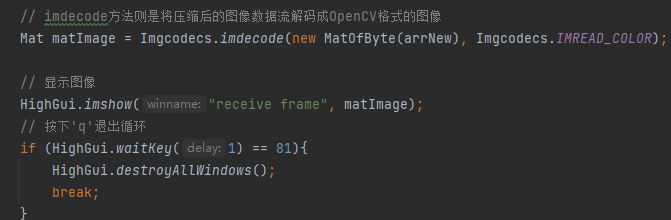
判断arr中字节的长度与文件头传来的字节的长度是否相等

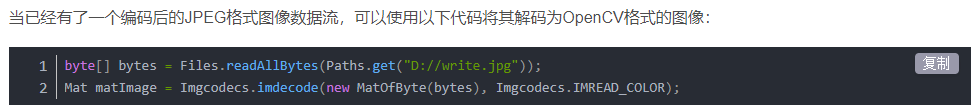
**将arr（Byte[]转化为byte[]）**



将arr（Byte[]转化为byte[]）

**解码并显示接收的图像**





imdecode方法则是将压缩后的图像数据流解码成OpenCV格式的图像。

