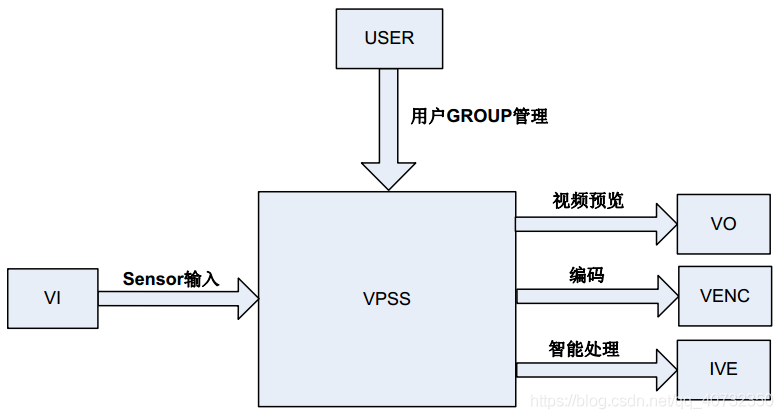
**一些git、c++、linux和解决bug的杂笔记**

海思芯片：

NNIE是Neural Network Inference Engine的简称，是海思媒体SoC中专门针对神经网络特别是深度学习卷积神经网络进行加速处理的硬件单元，支持现有大部分的公开网络，如Alexnet、VGG16、Googlenet、Resnet18、Resnet50等分类网络，Faster R-CNN、 YOLO、SSD、RFCN等检测网络，以及SegNet、FCN等场景分割网络。目前NNIE配套软件及工具链仅支持以Caffe框架，使用其他框架的网络模型需要转化为 Caffe框架下的模型。



连接使用磁盘阵列的问题：

1. 需要到git上去注册用户

2、若下载到虚拟机中，需要通过共享文件扩大内存。

**Vim常用编辑操作**

命令模式：

yy复制一行

dd删除一行

p粘贴

撤销u

复原ctrl+r

编辑模式：

ctrl+z撤销

ctrl+shift+z取消撤销

～代表无内容

**linux文件夹名称：**

opt:Optional application software packages添加的软件包 hisi放在这里

bin是可以执行文件

dev是驱动设备

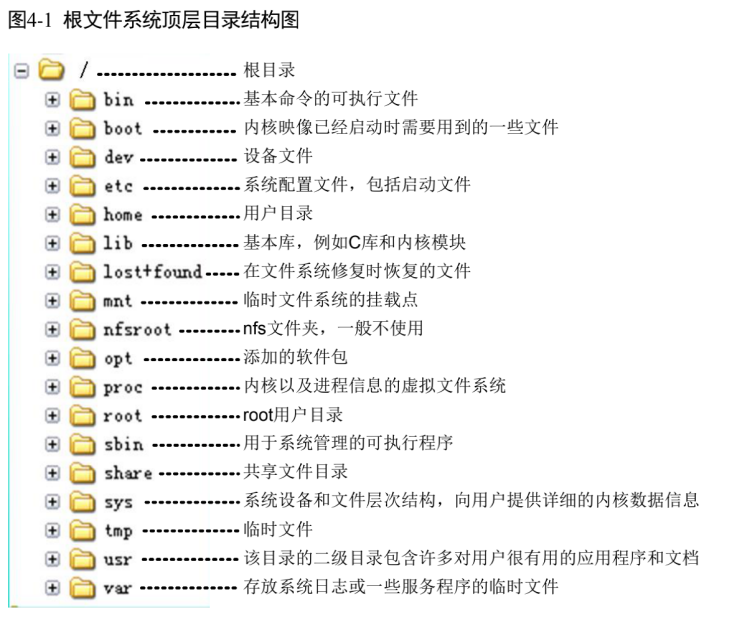
mnt是挂载点

etc:

初期：早期UNIX中，贝尔实验室的解释是：duetcetra directory 。

etc. 就是Et cetra。表示其他、等等什么的，英语里能常常看都这个缩写的。是用来放其他不能归类到其他目录中的内容。

后来FHS规定用来放配置文件，就解释为："Editable Text Configuration" 或者 "Extended Tool Chest"。



**cmakelist:**

add\_subdirectory命令，可以将指定的文件夹加到build任务列表中。

make用来执行Makefile，cmake用来执行CMakeLists.txt，qmake用来处理\*.pro工程文件。

Makefile的抽象层次最低，cmake和qmake在Linux等环境下最后还是会生成一个Makefile。

# 查找当前目录下的所有源文件

# 并将名称保存到 DIR\_SRCS 变量

aux\_source\_directory(. DIR\_SRCS)

# 添加链接库

target\_link\_libraries(Demo MathFunctions)

CMAKE\_BINARY\_DIR,PROJECT\_BINARY\_DIR,\_BINARY\_DIR：

这三个变量内容一致，如果是内部编译，就指的是工程的顶级目录，

如果是外部编译，指的就是工程编译发生的目录。

变量的引用方式是使用“${}”

自定义变量使用SET(OBJ\_NAME xxxx)，使用时${OBJ\_NAME}

在工程目录下，构建目录src,include,lib,bin。

#在src目录下存放源代码文件，

#include目录下存放头文件，

#lib目录用于存放生成的库（动态库或者静态库），

#bin目录存放最终生成的可执行文件。

#使用默认路径的g++指定编译器

set(CMAKE\_CXX\_COMPILER "g++")

#指定编译选项

set(CMAKE\_BUILD\_TYPE Debug )

#指定编译目录

set(PROJECT\_BINARY\_DIR ${PROJECT\_SOURCE\_DIR}/build)

#添加子目录,这样进入源码文件src目录可以继续构建

add\_subdirectory(${PROJECT\_SOURCE\_DIR}/src)

在使用git的过程中，有些时候我们只想要git服务器中的最新版本的项目，对于本地的项目中修改不做任何理会，就需要用到Git pull的强制覆盖，具体代码如下：

//git fetch从远程仓库获取更改，而不将它们合并到我的本地repo。

git fetch --all

//重置，抛弃我所有的分阶段和未分阶段的更改，忘记我当前的本地分支上的一切，使它与origin / master完全相同。通过使用与“硬复位”中相同的词暗示了破坏性本质。

git reset --hard origin/master

git pull

/\*\*

\* @brief 逐一以十六进制形式输出数组中的元素

\*

\* @param array 数组指针

\* @param length 数组长度

\* @param prefix 输出前缀

\*/

drwx---r-- 最前面的d表示这是一个文件夹,读写可执行

1.1 vector 说明

vector是向量类型，可以容纳许多类型的数据，因此也被称为容器

(可以理解为动态数组，是封装好了的类）

进行vector操作前应添加头文件#include <vector>

string s3 = R"(123\t123)";

OTABUG:

解决sring的乱码错误，发现问题在于库文件。

调整了login的位置，增加了线程之外的初次login，过了perform的问题，发现不是库的问题。

openssl: 多用途的命令行工具，包openssl，可以执行交互或批量命令。

libcrypto: 加密算法库，包openssl-libs。

libssl：加密模块应用库，实现了ssl及tls，包nss。

**#so（shared object）文件**

是Linux下的程bai序函数库,即编译du好的可以供其他程序使用的代码和数据。zhi

1、so文件就跟windows的.dll文件差不多。

windows中起名字更严肃点（链接文件，编译带上它，像一条链条一样），linux对它的起名更形象些(大家分享一起来用)，意思差不多。

2、一般来说，so文件就是常说的动态链接库, 都是C或C++编译出来的。与Java比较它通常是用的Class文件（字节码）。

3、Linux下的so文件时不能直接运行的,一般来讲,.so文件称为共享库。

4、so文件使用方法

(1)动态库的编译。这里有一个头文件：so\_test.h，三个.c文件：test\_a.c、test\_b.c、test\_c.c，我们将这几个文件编译成一个动态库：libtest.so。命令：$ gcc test\_a.c test\_b.c test\_c.c -fPIC -shared -o libtest.so 不用该标志外部程序无法连接。相当于一个可执行文件。

(2)动态库的链接这里有个程序源文件 test.c 与动态库 libtest.so 链接生成执行文件 test：命令：$ gcc test.c -L. -ltest -o test命令：$ ldd test执行test，可以看到它是如何调用动态库中的函数的。

**#SSL**

按照我的理解来解释下，为了让网络通信更安全，需要认证和加密，认证是说明你是要找的人，加密是为了让截获中间报文第三者无法得到消息内容。

为此有人设计了SSL，即套接字上的安全层，简单来说就是在TCP之上做一个安全通信层，HTTP on SSL 即是HTTPs，现在几乎所有的银行网站访问都是基于HTTPS协议的。

认证是通过证书+非对称加密算法来解决的，具体我也不搬迁了大家去看看

https交互过程

**#openssl库**

SSL交互过程还是很复杂的，牵扯到非对称加密和对称加密，以及复杂的交互过程，为此有人写了openssl库，这个库的使用非常广泛，本文就是阐明如何安装和基本使用这个库的。

有了这openssl库，就可以直接写和HTTPS的交互的代码了。

二、openssl 功能

openssl可以实现：秘钥证书管理、对称加密和非对称加密 。

#cmakelist 路径问题

在../..这个目录下可以正常 based on cmakelist

include\_directories(../../../3rdparty/Hi3559/jsoncpp/include)

基于ota的下一级目录,build

link\_directories(../../../../3rdparty/Hi3559/jsoncpp/lib)

设置新的3559板子。设置成200的。

现在有145、188、199、200。我在192.168.0.1网关下是145和200

修改ip主要是/etc/initd/s90network的inconfig th0啥的。

修改网关是route add default 192.168.0.1

都写到自启动项目里面。

如果telnet连接不上，可能是需要在自启动文件里面添加

telnetd &语句来启动telnet

有时候调bug，不一定是代码或者硬件的明确的错误。

最难受的是【不稳定】的错误。就是偶尔有，偶尔没有。没有错误不是因为代码或者接头，而是因为不稳定。又比如u盘的文件格式有问题。

这种很难受，没学到什么新知识，就是在不断地纠结。

自以为没有新知识会很轻松，反倒其实很痛苦。

mount -t nfs -o nolock 192.168.0.107:/home/ox/mount199 /opt pcgzcy

**Opencv中的Mat类**

使得Opencv的编程更加的简单，程序员不用过多的去关注内存管理，并且发现Opencv的编程有点像不需要太多编程技术的Matlab一样，甚至有些函数名字都是一样的。所以对Mat的了解是很有必要的。

  首先我们在处理一块数据的时候，如果使用Mat类，我们得到的好处是：

不需要手动申请一块内存；

在不需要时不用再手动释放内存；

可以通过类的封装，方便的获取到数据的相关信息。

**fopen中**

w 打开只写文件，若文件存在则文件长度清为0，即该文件内容会消失。若文件不存在则建立该文件。

w+ 打开可读写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。

int select(int n,fd\_set \* readfds,fd\_set \* writefds,fd\_set \* exceptfds,struct timeval \* timeout);

5. 具体参数说明：

1) int n:是一个整数值，是指集合中所有文件描述符的范围，即所有文件描述符的最大值加1，不能错！在Windows中这个参数的值无所谓，可以设置不正确。

2) fd\_set\*readfds是指向fd\_set结构的指针，这个集合中应该包括文件描述符，我们是要监视这些文件描述符的读变化的，即我们关心是否可以从这些文件中读取数据了，如果这个集合中有一个文件可读，select就会返回一个大于0的值，表示有文件可读，如果没有可读的文件，则根据timeout参数再判断是否超时，若超出timeout的时间，select返回0，若发生错误返回负值。可以传入NULL值，表示不关心任何文件的读变化。

3) fd\_set\*writefds是指向fd\_set结构的指针，这个集合中应该包括文件描述符，我们是要监视这些文件描述符的写变化的，即我们关心是否可以向这些文件中写入数据了，如果这个集合中有一个文件可写，select就会返回一个大于0的值，表示有文件可写，如果没有可写的文件，则根据timeout参数再判断是否超时，若超出timeout的时间，select返回0，若发生错误返回负值。可以传入NULL值，表示不关心任何文件的写变化。

4) fd\_set \*errorfds同上面两个参数的意图，用来监视文件错误异常。

5)struct timeval \*timeout是select的超时时间，这个参数至关重要，它可以使select处于三种状态，第一，若将NULL以形参传入，即不传入时间结构，就是将select置于阻塞状态，一定等到监视文件描述符集合中某个文件描述符发生变化为止；第二，若将时间值设为0秒0毫秒，就变成一个纯粹的非阻塞函数，不管文件描述符是否有变化，都立刻返回继续执行，文件无变化返回0，有变化返回一个正值；第三，timeout的值大于0，这就是等待的超时时间，即select在timeout时间内阻塞，超时时间之内有事件到来就返回了，否则在超时后不管怎样一定返回，返回值同上述。

Nvr的摄像头ip可以自己设置确定。

**环境变量：**

export LD\_LIBRARY\_PATH=../3rd/ffmpeg/:/opt/lib/hi3559/live555:/opt/lib/hi3559/openssl:/opt/lib/hi3559/x264::/opt/lib/hi3559/ffmpeg:../3rd/svp/:../3rd/curl/:$LD\_LIBRARY\_PATH

用include 引用头文件时，双引号和尖括号的区别：

1.双引号：引用非标准库的头文件，编译器首先在程序源文件所在目录查找，如果未找到，则去系统默认目录查找，通常用于引用用户自定义的头文件。

2.尖扩号：范围更小

只在系统默认目录（在Linux系统中通常为/usr/include目录）或者尖括号内的路径查找，通常用于引用标准库中自带的头文件。

综上，标准库自带的头文件既可以用双引号也可以用尖括号，不过习惯使用尖括号，用户自定义的头文件只能用双引号。

出现的好多问题，都是环境配置的问题。就很恼火。

库文件，环境变量和makefile等等

电子围栏，找不到libx264.so的问题

find / | grep libx264

发现在/usr/lib下面有3个文件：

libx264.a

libx264.so -> libx264.so.96

libx264.so.96

看来依赖的库是有了，只是版本不够高。估计pacman -Syu可以解决问题，不过现在先建个链接欺骗一下系统：

ln -s libx264.so libx264.so.98

gcc不成功

后来make成功之后，是程序有问题。可能要用新的souvif？

make 程序出错。dvpp

atlas的dvpp暂未成功。

电子围栏生成yuv成功。

下一步：实现libcurl上传图片

URL就是网址(域名）

HTTP是协议

比如http://www.baidu.com

那么就是通过HTTP协议访问URL是www.baidu.com的网址。

URL，中文意思是统一资源定位器，又称“域名”。

计算机只认IP地址，不认域名，但是IP地址太难记了，怎么办？于是就有了DNS（中文意思：域名服务器）来帮助用户将域名转换成IP地址。

DNS就像一本通讯录一样，它返回域名对应的IP地址（但用户是看不见的，因为地址栏的地址没有改变），浏览器再通过IP地址来连接网页服务器，最终达到“记忆域名就能打开对应网页”的目的。

大部分人是选择访问“www.baidu.com”来进入百度首页的，但也有那么一部分人喜欢访问“180.97.33.108”来进入百度首页，但记忆4段数字远没有记忆一个拼音效率高吧？由此可见URL对大部分人的重要性。

libcurl测试https://mail.qq.com，为什么make不出来。

gcc curl-new.c -lcrypto -lssl -lcurl

gcc curl-new.c -lcrypto -lssl -lcurl

gcc curl-new.c -lcrypto -lssl -lcurl

gcc curl-new.c -lcrypto -lssl -lcurl

gcc curl-new.c -lcrypto -lssl -lcurl

gcc curl-new.c -lcrypto -lssl -lcurl

gcc curl-new.c -lcrypto -lssl -lcurl

没有用cmake和make ，而是直接添加在后面的链接库！

-l库和在cmake里写链接库是一个作用呢。（link\_directories("/usr/lib32/")）

###但是最后还是没有解决link——directories的问题。经过峰哥指导，可能根本就不是路径的原因。

当搜索用于 ../lib/libcurl.so 时跳过不兼容的 -lcurl，可能是32位和64位的原因。

后来发现是编译器的问题。我成功的是gcc编译的。但是cmake里写的是交叉编译的编译器。峰哥说，gcc和交叉编译用的lib是不一样的。

curl访问https在3559上调试

./curlEXECUTABLE: relocation error: ../lib/libc.so.6: symbol \_dl\_exception\_create, version GLIBC\_PRIVATE not defined in file ld-linux-aarch64.so.1 with link time reference

/opt/pcgzcy # ./curlEXECUTABLE www.baidu.com

./curlEXECUTABLE: relocation error: ../lib/libc.so.6: symbol \_dl\_exception\_create, version GLIBC\_PRIVATE not defined in file ld-linux-aarch64.so.1 with link time reference

/opt/pcgzcy # ./curlEXECUTABLE www.baidu.com

./curlEXECUTABLE: relocation error: ../lib/libc.so.6: symbol \_dl\_exception\_create, version GLIBC\_PRIVATE not defined in file ld-linux-aarch64.so.1 with link time reference

/opt/pcgzcy # ./curlEXECUTABLE

./curlEXECUTABLE: relocation error: ../lib/libc.so.6: symbol \_dl\_exception\_create, version GLIBC\_PRIVATE not defined in file ld-linux-aarch64.so.1 with link time reference

##关于环境变量、makefile，cmakelist一知半解，库和头文件，还有c++，一无所知。

##关于atlas和3559一无所知。

交叉编译时候如何配置连接库的搜索路径

交叉编译的时候不能使用本地（i686机器，即PC机器，研发机器）机器上的库，但是在做编译链接的时候默认的是使用本地库，即/usr/lib,/lib两个目录。因此，在交叉编译的时候，要采取一些方法使得在编译链接的时候找到需要的库。

首先，要知道：

##编译的时候只需要头文档，真正实际的库文档在链接的时候用到。!!!

##头文件是用来声明的，库文件是放的函数！

（这是我的理解，假如有不对的地方，敬请网上各位大侠指教） 然后，讲讲如何在交叉编译链接的时候找到需要的库。

（1）、交叉编译时候直接使用-L和-I参数指定搜索非标准的库文档和头文档的路径。例如：

arm-linux-gcc test.c -L/usr/local/arm/2.95.3/arm-linux/lib -I/usr/local/arm/2.95.3/arm-linux/include

（2）、使用ld.so.conf文档，将用到的库所在文档目录添加到此文档中，然后使用ldconfig命令刷新缓存。

（3）、使用如下命令：

$ export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/arm/2.95.3/arm-linux-lib

参见《ld.so.conf 文档和PKG\_CONFIG\_PATH变量》这篇文章。

通过环境变量LD\_LIBRARY\_PATH指定动态库搜索路径（！）。

通过设定环境变量LD\_LIBRARY\_PATH也可以指定动态库搜索路径。当通过该环境变量指定多个动态库搜索路径时，路径之间用冒号"："分隔。

不过LD\_LIBRARY\_PATH的设定作用是全局的，过多的使用可能会影响到其他应用程序的运行，所以多用在调试。（LD\_LIBRARY\_PATH 的缺陷和使用准则，可以参考《Why LD\_LIBRARY\_PATH is bad》）。通常情况下推荐还是使用gcc的-R或-rpath选项来在编译时就指定库的查找路径，并且该库的路径信息保存在可执行文件中，运行时它会直接到该路径查找库，避免了使用LD\_LIBRARY\_PATH环境变量查找。

std::shared\_ptr,对象std域的**智能指针**

**###C++11 std::shared\_ptr总结与使用**

最近看代码，智能指针用的比较多，自己平时用的少，周末自己总结总结。方便后续使用。

std::shared\_ptr大概总结有以下几点:

(1) 智能指针主要的用途就是方便资源的管理，自动释放没有指针引用的资源。

(2) 使用引用计数来标识是否有多余指针指向该资源。(注意，shart\_ptr本身指针会占1个引用)

(3) 在赋值操作中, 原来资源的引用计数会减一，新指向的资源引用计数会加一。

std::shared\_ptr<Test> p1(new Test);

std::shared\_ptr<Test> p2(new Test);

p1 = p2;

(4) 引用计数加一/减一操作是原子性的，所以线程安全的。

(5) make\_shared要优于使用new，make\_shared可以一次将需要内存分配好。

std::shared\_ptr<Test> p = std::make\_shared<Test>();

std::shared\_ptr<Test> p(new Test);

(6) std::shared\_ptr的大小是原始指针的两倍，因为它的内部有一个原始指针指向资源，同时有个指针指向引用计数。

(7) 引用计数是分配在动态分配的，std::shared\_ptr支持拷贝，新的指针获可以获取前引用计数个数。

atlas编程：

在engine。cpp里修改engine、

在commom.h里修改路径。

BGR的输入给yolov4、

测时间

和soap—onvif：

soap自动生成onvif框架，soap开头的很多文件都是自动生成的。

onvif文件夹内的是主要需要改的地方。

live555是rtsp传输的。

ffmpeg和x264是编解码的，没用opencv。

指令tree -L 2，看目录结构。

lib的svp是海思相关的。

plugin（）建立插件

rtsp之后需要编解码

openurl，打开rul，进行传输，进入传输流

数据的存储，

grep -r xxxx

rtspclinentcount（）多线程，线程数加，多路摄像头？。。。

aftergettingframe（）每一帧的返回？？拿到r帧就返回，p帧和b帧屏蔽？264流。

decodeframe （）解码。yuv420，

yuv420toRGB（）可能要做成一个线程

getBoxBuffer（）是走算法。

saveyuvjpeg（）保存图片。由一个flag控制。

markimageflag（）保存数据

upload\_file（）上传

curl上传内容为form-data。

down\_file（）下载文件。（get？）

在httpFileTranssmission.cpp中

getresponsestatus（），可以解析

MQTT协议？

mount -t nfs -o nolock 192.168.31.152:/home/ox/3559mount /opt/pcgzcy/

mount -t nfs -o nolock 192.168.31.152:/home/ox/3559mount /opt/pcgzcy/

mount -t nfs -o nolock 192.168.31.152:/home/ox/3559mount /opt/pcgzcy/

mount -t nfs -o nolock 192.168.31.152:/home/ox/3559mount /opt/pcgzcy/

mount -t nfs -o nolock 192.168.31.152:/home/ox/3559mount /opt/pcgzcy/

mount -t nfs -o nolock 192.168.31.152:/home/ox/3559mount /opt/pcgzcy/

mount -t nfs -o nolock 192.168.31.152:/home/ox/3559mount /opt/pcgzcy/

mount -t nfs -o nolock 192.168.31.152:/home/ox/3559mount /opt/pcgzcy/

mount -t nfs -o nolock 192.168.31.152:/home/ox/3559mount /opt/pcgzcy/

少加一个nolock的选项都要失败。。

安装track3559的安装包，再替换新的iic的库文件。

两周半时间：

联调

怀礼哥程序：控制摄像头、登陆获取图像、

有一张图片上传下载，和一条json？。。。rgb？

环境搭起来一起跑。

怀礼直接访问智勇。url，设备管理，标定，报警。。。。。。

多设备采集，和什么什么的。

先完成功能、再求稳定。。

多路摄像头，多个频率。。内存值，保护数据，啥啥的

mmc空间的yuv存放位置

3559硬解码，到mmz的空间上。

rtsp之后解码之后，需要变换以一下大小。resize。

数据结构+软件设计这一块，

或可先跑一路。怀礼以及同了，

后面的移植性、

解码：硬件解码。

周三下午碰一下

--------------------

export LD\_LIBRARY\_PATH=/opt/wfjc/1.0.14/3rd/encry/:/opt/wfjc/1.0.14/lib/:/opt/wfjc/1.0.14/3rd/svp/:/opt/wfjc/lib/:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=/opt/wfjc/lib/:$LD\_LIBRARY\_PATH

1、测试3559新sdk

2、调试怀礼哥的电子围栏ipconvif

3、atlas跑yolo时间，yolo要通过git下载。

4、git上传程序。。

linux编译出现undefined reference to symbol 'pthread\_create错误解决方案

出现该问题的原因：pthread不是linux下的默认的库，也就是在链接的时候，无法找到phread库中线程函数的入口地址，于是链接会失败。

在该行最后添加-lpthread

单独可以成功的程序，放到线程就里不行！

死活调不同！原来是线程还没跑完，主程序就return 0 了！！

需要给线程sleep（1）！！！

##bug2

c语言中不允许在函数外部给全局变量赋值!比如在主函数外!全局变量在函数外部只能在定义的时候初始化!

2、string 转换成 char \*

如果要将string直接转换成const char \*类型。string有2个函数可以运用。

一个是.c\_str()，一个是data成员函数。

一：string s1 = "abcdeg";

const char \*k = s1.c\_str();

const char \*t = s1.data();

printf("%s%s",k,t);

cout<<k<<t<<endl;

如上，都可以输出。内容是一样的。但是只能转换成const char\*，如果去掉const编译不能通过。

二：那么，如果要转换成char\*，可以用string的一个成员函数copy实现。

string s1 = "abcdefg";

char \*data;

int len = s1.length();

data = (char \*)malloc((len+1)\*sizeof(char));

s1.copy(data,len,0);

printf("%s",data);

cout<<data;

这个函数可能是将char\*变成了char[]:

data = (char \*)malloc((len+1)\*sizeof(char));

##创建线程的时候，也不能直接传参数。要进行格式类型转换：

######： thread\_arg = (int)(\*((int\*)args));

void \*PrintHello(void \*args)

{

int thread\_arg;

sleep(1);

thread\_arg = (int)(\*((int\*)args));

printf("Hello from thread %d\n", thread\_arg);

return NULL;

}

int main(void)

{

int rc,t;

pthread\_t thread[NUM\_THREADS];

for( t = 0; t < NUM\_THREADS; t++)

{

printf("Creating thread %d\n", t);

//此处t变量的用法是方便大家调测代码的写法，实际使用会有问题，因为这个t是局部变量，

函数执行完后马上释放，大家传递参数时需要使用全局变量或malloc出来的变量。

rc = pthread\_create(&thread[t], NULL, PrintHello, &t);

if (rc)

{

printf("ERROR; return code is %d\n", rc);

return EXIT\_FAILURE;

}

}

sleep(5);

for( t = 0; t < NUM\_THREADS; t++)

pthread\_join(thread[t], NULL);

return EXIT\_SUCCESS;

}

#! 是个指示路径的表示符，/bin/bash和/bin/sh指定了脚本解析器的程序路径 bash是sh的完整版，bash完全兼容sh命令，反之不行

查询当前目录总大小可以使用du -sh

直接git reset --hard master，直接从仓库中拉出覆写暂存区和工作区。

学习C++：

创建对象的两种方法：

（1）在栈上创建。

ClassName ObjName1, ObjName2（parameter01）

ObjName1.xxx

在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束后在将这些局部变量的内存空间回收。在栈上分配内存空间效率很高，但是分配的内存容量有限。

stack栈里使用：

顾名思义，用这种方法创建的对象，内存分配到栈里（Stack）。使用 “.” 非 “->” 调用对象的方法。当程度离开对象的使用范围（如方法结束，一个程度块的最后{}），范围内的栈中的对象会自动删除，内存自动回收。这是创建对象最简单的方式，与“int x = 0;”是一样的。

（2）从堆上分配的。

ClassName \*obj1 = new ClassName();

ClassName \*obj2 = new ClassName(parameter);

delete obj1;

delete obj2;

obj2->xxx

程序在运行的时候用 malloc 或 new 申请任意多少的内存，程序员自己负责在何时用 free 或 delete 释放内存。

用这种方法创建的对象，内存分配到堆里（Heap）。一般使用“->” 调用对象的方法。箭头操作符”->"将解引用（dereferencing\*）和成员使用（member access.）结合起来，下例两个输出，效果等价：

1 int \_tmain(int argc, \_TCHAR\* argv[])

2 {

3 SpreadsheetCell \*myCellp = new SpreadsheetCell();

4

5 myCellp->setValue(3.7);

6

7 cout<<"cell 1: "<<myCellp->getValue()<<" "<<myCellp->getString()<<endl;

8

9 cout<<"cell 1: "<<(\*myCellp).getValue()<<" "<<(\*myCellp).getString()<<endl;

10

11 delete myCellp;

12 return 0;

13 }

构建函数：

#include <iostream>

using namespace std;

class time

{

public:

time()//constructor.构造函数

{

hour=0;

minute=0;

sec=0;

}

void set\_time();

void show\_time();

private:

int hour, minute, sec;

};

int main()

{

class time t1;

t1.show\_time();

t1.set\_time();

t1.show\_time();

return 0;

}

void time::set\_time()

{

cin>>hour>>minute>>sec;

}

void time::show\_time()

{

cout<<hour<<":"<<minute<<":"<<sec<<endl;

}

程序运行情况：

0：0：0

10 11 11 回车

10：11：11

overload可直译为重载

它是指我们可以定义一些名称相同的方法，通过定义不同的输入参数来区分这些方法，然后再调用时，VM就会根据不同的参数样式，来选择合适的方法执行。

与override的区别：

override可以翻译为覆盖，从字面就可以知道，它是覆盖了一个方法并且对其重写，以求达到不同的作用。

#构造函数 ，是一种特殊的方法。主要用来在创建对象时初始化对象， 即为对象成员变量赋初始值，总与new运算符一起使用在创建对象的语句中。特别的一个类可以有多个构造函数 ，可根据其参数个数的不同或参数类型的不同来区分它们 即构造函数的重载。

#与构造函数相反，当对象结束其生命周期，如对象所在的函数已调用完毕时，系统会自动执行析构函数。以C++语言为例：析构函数名也应与类名相同，只是在函数名前面加一个位取反符~，例如~stud( )，以区别于构造函数。它不能带任何参数，也没有返回值（包括void类型）。只能有一个析构函数，不能重载。

把类的声明放在main函数之前，它的作用域是全局的。这样做可以使main函数更简练一些。在main函数中定义了两个对象并且给出了初值，然后输出两个学生的数据。当主函数结束时调用析构函数，输出stud has been destructe!。值得注意的是，真正实用的析构函数一般是不含有输出信息的。

在本程序中，成员函数是在类中定义的，如果成员函数的数目很多以及函数的长度很长，类的声明就会占很大的篇幅，不利于阅读程序。而且为了隐藏实现，一般是有---//---必要将类的声明和实现（具体方法代码）分开编写的---//---，这也是一个良好的编程习惯。即可以在类的外面定义成员函数，而在类中只用函数的原型作声明。

**虚拟机**其实只是一个软件，它虚拟出CPU、显卡、内存、硬盘等硬件，“骗过”Windows操作系统，然后把系统文件安装在虚拟的硬盘中。而且这些虚拟硬件的配置参数都是可以调整的，只要你的电脑硬件配置足够强大。

basic\_string& insert( size\_type index, const CharT\* s );

index位置插入一个常量字符串

合并git上的版本，完成演示demo。半周都没成功，一直怀疑是代码的原因。实际原因是多方面的。

包括

1、代码原因

2、服务器设置原因

3、摄像头原因

4、155的3559的板卡本身有问题。

export LD\_LIBRARY\_PATH=/opt/wfjc\_test/1.0.15/3rd/enc

ry/:/opt/wfjc\_test/1.0.15/lib/:/opt/wfjc\_test/1.0.15/3rd/svp/:/opt/wfjc\_test/lib

/:/opt/wfjc\_test/1.0.15/3rd/opencvffmlib/lib/:$LD\_LIBRARY\_PATH

調試rtsp連接不穩定的結果是，追蹤到cffdecoder的構建函數中。未繼續往下。

10點任務：

嘗試楠姐的mqtt的demo。在linux上嘗試mqtt—fx終端。

平安雲sdk、使用說明，平安雲賬號daysend，zhinengyingjian1

並最終和勇哥完成聯調。我去訂閱他的，他還能發給我。

還需要把gcc 換成新的sdk？？

mqtt三元组

阿里物联网平台的连接注册也需要用到“三元组”：

ProductKey

DeviceName

DeviceSecret

这个三元组并不是直接拿来用的，而是需要按照官方设定的协议规则得到UserName、PassWord、ClientID、IP这几个参数，规则就是:

参数 阿里云物联网平台参数

UserName {DeviceName}&{ProductKey}

PassWord 通过下面的算法获取

ClientID {随便定}|securemode=3,signmethod=hmacsha1,timestamp=789|

IP {ProductKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com

13：30-15：00下載mqtt-fx。未成功。 在平安雲上調試了雲端的配置。再梳理一下流程吧。

mqttfx應該是模擬設備端。平安雲是服務器端。兩邊通信。

15-16：00完成mqtt box的下载。

ProductKey FJtkHLatd9fyjbdEWbG

DeviceSecret OIrCeReCNWYURxwkFOZNRrsfzDFFkAni

DeviceName enhss92929

${productKey}+"@@"+${deviceName}+"@@${signMethod}@@${timestamp}

FJtkHLatd9fyjbdEWbG@@enhss92929@@SHA-1@@

#nfs的重启指令：service nfs restart

cmake

clean不是一个文件，它只不过是一个【【【动作】】】

有点像C语言中的lable一样，其冒号后什么也没有，那么，make就不会自动去找文件的依赖性，也就不会自动执行其后所定义的命令。要执行其后的命令，就要在make命令后明显得指出这个lable的名字。

这样的方法非常有用，我们可以在一个makefile中定义不用的编译或是和编译无关的命令，比如程序的打包，程序的备份，等等。

###makefile中 $() ${}两种形式都可以读取变量的值

在Makefile中，.PHONY后面的target表示的也是一个伪造的target, 而不是真实存在的文件target，注意Makefile的target默认是文件。

所以.PHONY clean 可以忽略clean动作

av：audiovideo

libavformat：用于各种音视频【封装格式】的生成和解析，包括获取解码所需信息以生成解码上下文结构

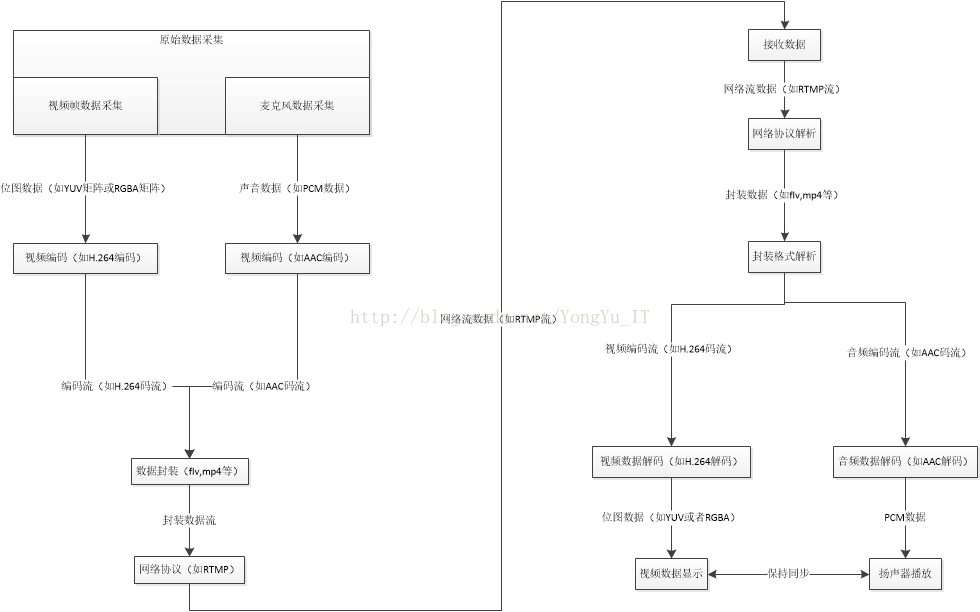
和读取音视频帧等功能；

libavcodec：用于各种类型声音/图像编解码；

为什么需要编码解码

采集的原始音视频信号体积都非常大。里面有很多相同的、眼看不到的、耳听不到的内容。为了减小体积便于存储、传输、交流，就有了压缩无用信息和不重要的信息的“压缩处理”，这就叫“编码”。

压缩（编码）过的内容无法直接使用，使用（观看）时必须解压缩，还原为原始的信号（比如视频中某个点的颜色等），这就是"解码"或者"解压缩"



编解码流程

下列代码：yuv通过RGB编码生成jpg。

其中编码jpg需要生成头header和尾tailer。

核心程序：avcodec\_encode\_video2(pCodecCtx, &pkt, pFrame, &got\_picture);//将pframe编码为packet中的jpeg

int CFFDecode::SaveYUVToJPEG(AVFrame\* pFrame, int width, int height)

{

// 输出文件路径

static int jpegIndex = 0;

char out\_file[1024] = {0};

//sprintf\_s(out\_file, sizeof(out\_file), "%s%d.jpg", "E:/QT/test\_ffmpegSavePic/ffmpeg/output/", jpegIndex);

//sprintf (out\_file, "/opt/sdcard/vision/jpegFrame%d.jpg", jpegIndex);

sprintf (out\_file, "./jpegFrame%d.jpg", jpegIndex);

jpegIndex++;

saveEvd(WebCamInfo[0].configInfo.str\_device\_id, WebCamInfo[0].url, out\_file);

// 分配AVFormatContext对象

AVFormatContext\* pFormatCtx = avformat\_alloc\_context();

// 设置输出文件格式

pFormatCtx->oformat = av\_guess\_format("mjpeg", NULL, NULL);

// 创建并初始化一个和该url相关的AVIOContext

if( avio\_open(&pFormatCtx->pb, out\_file, AVIO\_FLAG\_READ\_WRITE) < 0)

{

printf("Couldn't open output file.");

return -1;

}

// 构建一个新stream

AVStream\* pAVStream = avformat\_new\_stream(pFormatCtx, 0);

if( pAVStream == NULL )

{

return -1;

}

// 设置该stream的信息

AVCodecContext\* pCodecCtx = pAVStream->codec;

pCodecCtx->codec\_id = pFormatCtx->oformat->video\_codec;

pCodecCtx->codec\_type = AVMEDIA\_TYPE\_VIDEO;

pCodecCtx->pix\_fmt = AV\_PIX\_FMT\_YUVJ420P;

pCodecCtx->width = width;

pCodecCtx->height = height;

pCodecCtx->time\_base.num = 1;

pCodecCtx->time\_base.den = 25;

//打印输出相关信息

av\_dump\_format(pFormatCtx, 0, out\_file, 1);

//================================== 查找编码器 ==================================//

AVCodec\* pCodec = avcodec\_find\_encoder(pCodecCtx->codec\_id);

if( !pCodec )

{

printf("Codec not found.");

return -1;

}

// 设置pCodecCtx的解码器为pCodec

if( avcodec\_open2(pCodecCtx, pCodec, NULL) < 0 )

{

printf("Could not open codec.");

return -1;

}

//================================Write Header ===============================//

avformat\_write\_header(pFormatCtx, NULL);

int y\_size = pCodecCtx->width \* pCodecCtx->height;

//==================================== 编码 ==================================//

// 给AVPacket分配足够大的空间

AVPacket pkt;

av\_new\_packet(&pkt, y\_size \* 3);

//

int got\_picture = 0;

int ret = avcodec\_encode\_video2(pCodecCtx, &pkt, pFrame, &got\_picture);//编码为packet中的jpeg

if( ret < 0 )

{

printf("Encode Error.\n");

return -1;

}

if( got\_picture == 1 )

{

pkt.stream\_index = pAVStream->index;

ret = av\_write\_frame(pFormatCtx, &pkt);//Write a packet to an output media file？

}

av\_free\_packet(&pkt);

//Write Trailer

av\_write\_trailer(pFormatCtx);// Write the stream trailer to an output media file

if( pAVStream )

{

avcodec\_close(pAVStream->codec);

}

avio\_close(pFormatCtx->pb);

avformat\_free\_context(pFormatCtx);

//markImageFlag = false;

return 0;

}

---------------------------------------------------------

---------12.09-------------------------------------------

---------------------------------------------------------

昨天bug：

rectangle（）添加矩形之后，没有进行保存。因为以前的都是show（img），现在是在板卡上，没办法show就需要iwrite啦。

264--硬件解码可能可以避免这个问题.

m\_yuv420还是加了头部的东西的.

yuv的r帧保存啥啥的,所以有乱码.

opencv改为ffmpeg

linux查看yuv图像

ffplay -f rawvideo -video\_size 640x480 raw\_640x480.yuv

int bufLen = m\_yuv420->width \* m\_yuv420->height\*3/2;

cout << "++++m\_yuv420->width++++++" << m\_yuv420->width <<endl;

cout << "++++m\_yuv420->height++++++" << m\_yuv420->height <<endl;

saveYUVFrameToFile(m\_yuv420, m\_yuv420->width,m\_yuv420->height);

SaveYUVToJPEG(m\_yuv420, m\_yuv420->width, m\_yuv420->height);

Mat matImageYuv(m\_yuv420->height\*3/2, m\_yuv420->width, CV\_8UC1, (m\_yuv420->data+0));

cv::rectangle(matImageYuv,Point(10,10),Point(100,300),Scalar(0,0,255),5,1,0);

memcpy(m\_yuv420->data, matImageYuv.data, bufLen);

SaveYUVToJPEG(m\_yuv420, m\_yuv420->width, m\_yuv420->height);

cv::Mat matrgbImg;

matrgbImg.create(m\_yuv420->height\*3/2, m\_yuv420->width, CV\_8UC1);

cv::cvtColor(matImageYuv, matrgbImg, CV\_YUV2BGR\_I420);

cv::Mat matrgbImg;

cv::cvtColor(matImageYuv, matrgbImg, CV\_YUV2BGR\_I420);

cv::imwrite("testrgb.jpg", matrgbImg);

SaveYUVToJPEG(m\_yuv420, m\_yuv420->width, m\_yuv420->height);

AVFrame \*m\_yuv420\_2;

memcpy(m\_yuv420->data, matImageYuv.data, bufLen);

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

cout << "++++bufLen++++++" << bufLen <<endl;

// //cvtColor(matImageYuv, m\_yuv420->data,CV\_RGB2YUV\_I420);

// SaveYUVToJPEG(m\_yuv420\_2, m\_yuv420\_2->width, m\_yuv420\_2->height);

//AVFrame \*m\_yuv420\_2 = m\_yuv420;

//memcpy(m\_yuv420\_2->data, matImageYuv.data, bufLen\*sizeof(unsigned char));

//cvtColor(matImageYuv, m\_yuv420->data,CV\_RGB2YUV\_I420);

之所以yuv无法转回mat，是因为avFrame的data是有y、u、v三个分量。转给mat可以转data[0]，但是转回来就只剩下一个分量了。

最终策略，还是不在yuv这里进行加框操作了，直接给jpg添加框的操作。

任务完成的总结：

git未完成。

1、深入探究，发现cv的mat无法保存yuv格式的图片 。

2、最终采用在save evidence pengding之前，imread（ jpg）的图片并rectangle（），再进行imwrite（）

3、解决真实region的 bug:原来是webcam传过来的数据不对，以为足够大了，实际上很小。

后来进行了等比例转换就好多了。

遗留问题：box的边界和region的边界问题，有余力的时候进行面积判断，而不是点的判断。

4、yuv格式，和rgb格式，还有jpg格式。

###非常全的代码比较网站：

###Linux下9种优秀的代码比对工具推荐小结

###https://www.jb51.net/article/189697.htm

#####Git中.gitignore文件不起作用的解决以及Git中的忽略规则介绍

在Studio里使用Git管理代码的过程中，可以修改.gitignore文件中的标示的方法来忽略开发者想忽略掉的文件或目录，如果没有.gitignore文件，可以自己手工创建。在.gitignore文件中的每一行保存一个匹配的规则例如：

# 此为注释 – 将被 Git 忽略

\*.a # 忽略所有 .a 结尾的文件

!lib.a # 但 lib.a 除外

/TODO # 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO

build/ # 忽略 build/ 目录下的所有文件

doc/\*.txt # 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt

在填写忽略文件的过程中，我发现在Android Studio里面，.gitignore中已经标明忽略的文件目录下的文件，当我想git push的时候还会出现在push的目录中，原因是因为在Studio的git忽略目录中，新建的文件在git中会有缓存，如果某些文件已经被纳入了版本管理中，就算是在.gitignore中已经声明了忽略路径也是不起作用的，这时候我们就应该先把本地缓存删除，然后再进行git的push，这样就不会出现忽略的文件了。git清除本地缓存命令如下：

git rm -r --cached .

git add .

git commit -m 'update .gitignore'

```````````````````edge/appSoftware/main.cpp

修改： edge/deviceinfo/deviceInfo.cpp

修改： edge/evd\_upload/evd\_upload.cpp

修改： edge/include/live555/CFFDecode.h

修改： edge/live555/CFFDecode.cpp

修改： edge/live555/CRTSPDecoder.cpp

Git版本回退的最佳方式

使用git开发的过程中，存在误提交的时候怎么办呢？不用慌张，强大的git提供了两种版本回退的方式，可以让你恢复提交之前的内容：

方式一：reset（不推荐）

通过reset的方式，把head指针指向之前的某次提交，reset之后，后面的版本就找不到了

操作步骤如下：

1、在gitlab上找到要恢复的版本号，如：

139dcfaa558e3276b30b6b2e5cbbb9c00bbdca96

2、在客户端执行如下命令（执行前，先将本地代码切换到对应分支）：

git reset --hard 139dcfaa558e3276b30b6b2e5cbbb9c00bbdca96

3、强制push到对应的远程分支（如提交到develop分支）

git push -f -u origin develop

OK，现在到服务器上看到的代码就已经被还原回去了。这种操作存在一个问题，服务器上的代码虽然被还原了，但假如有多个人在使用，他们本地的版本依然是比服务器上的版本高的，所以，别人再重新提交代码的话，你撤销的操作又会被重新，你上面的操作也就白操作了。解决办法是，让别人把本地的分支先删掉，然后重新从服务器上拉取分支

方式二：revert（推荐）

操作步骤如下：

1、找到你误提交之前的版本号

2、git revert -n 版本号

3、git commit -m xxxx 提交

4、git push 推送到远程

OK，版本还原了，通知其他人更新一下代码吧

cmake的gtest测试套件：

add\_test(NAME craft\_test

COMMAND craft --gtest\_output='xml:report.xml')

在gcc编译时不会出错，但是用g++就会有问题，究其原因就是C语言编译器允许隐含性的将一个通用指针转换为任意类型的指针，而C＋＋不允许。

今天运行了一串代码，在VSCode中运行出错了，原因是它的参数不匹配。

invalid conversion from 'void\* ()()' to 'void ()(void)' [-fpermissive]

报错解决办法：

char\* a;

void\* b;

a=(char\*)b;

解决方法2：CXXFLAGS = -fpermissive

ld: 找不到 -lpaho-mqtt3c

调整了一下cmakelist的语句和顺序。。居然就成功了。

所以说明问题还是在于这个库没找到，而这个库本身是没有问题的。

后来编译成功之后，运行有库找不到。

error while loading shared libraries: libssl.so.1.1: cannot open shared object file: No such file or directory

用了两个方法：

1、find -name libssl.so.1.1;然后去核对其版本，怀疑主程序和mqtt使用的libssl.so的版本不对。虽然看到各个地方的libssl.so的大小不一样，但后来发现不是问题所在。

2、从cmakelist去看，编译时候链接的哪个库。

3、最后发现都不对，还是要用echo $PATH 去查看环境变量。发现都不在环境变量里。

回到run.sh里发现，之前export LD\_LIBRARY\_PATH没有添加尾巴:$LD\_LIBRARY\_PATH，进行环境变量地址的拼接。

export LD\_LIBRARY\_PATH=../lib/hi3559/openssl/:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/yolov4:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/json:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/mqtt:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/opencv-3.4.1/:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/x264/:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/encry:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/opencvffmlib/lib/:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/ffmpeg/:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/svp/:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=../lib/hi3559/curl/:$LD\_LIBRARY\_PATH

######################git下载地址################################

git clone wulianwang@192.168.31.153:/volume1/磁盘1/git/visionBox/.git

######################git下载地址################################

git失败的原因：

可能是global name和email的时候少了一个空格！！！

还有顺序，应该先设置config，再那啥啥。

git init

$ git config --global user.name "liguangxian"

$ git config --global user.email "2995183552@qq.com"

$ git remote add origin git@github.com:liguangxian/timing

git pull origin master

查看倉庫地址命令：git remote -v

倉庫地址：ssh://wulianwang@192.168.31.153/volume1/磁盘1/git/trafficBox