2024.4.22

1.sourceTree：开源的代码仓库客户端工具 2.gitLab:是一个基于Git的版本控制工具，并在此基础上搭建起来的Web服务

4.23

1.C++编码规范看一下:

注意一下

/\*! \file LeeperCameraController.h

. Copyright (C) 2014 杭州利珀科技有限公司. . . . . Created by 金秉文 at 2014/08/23.

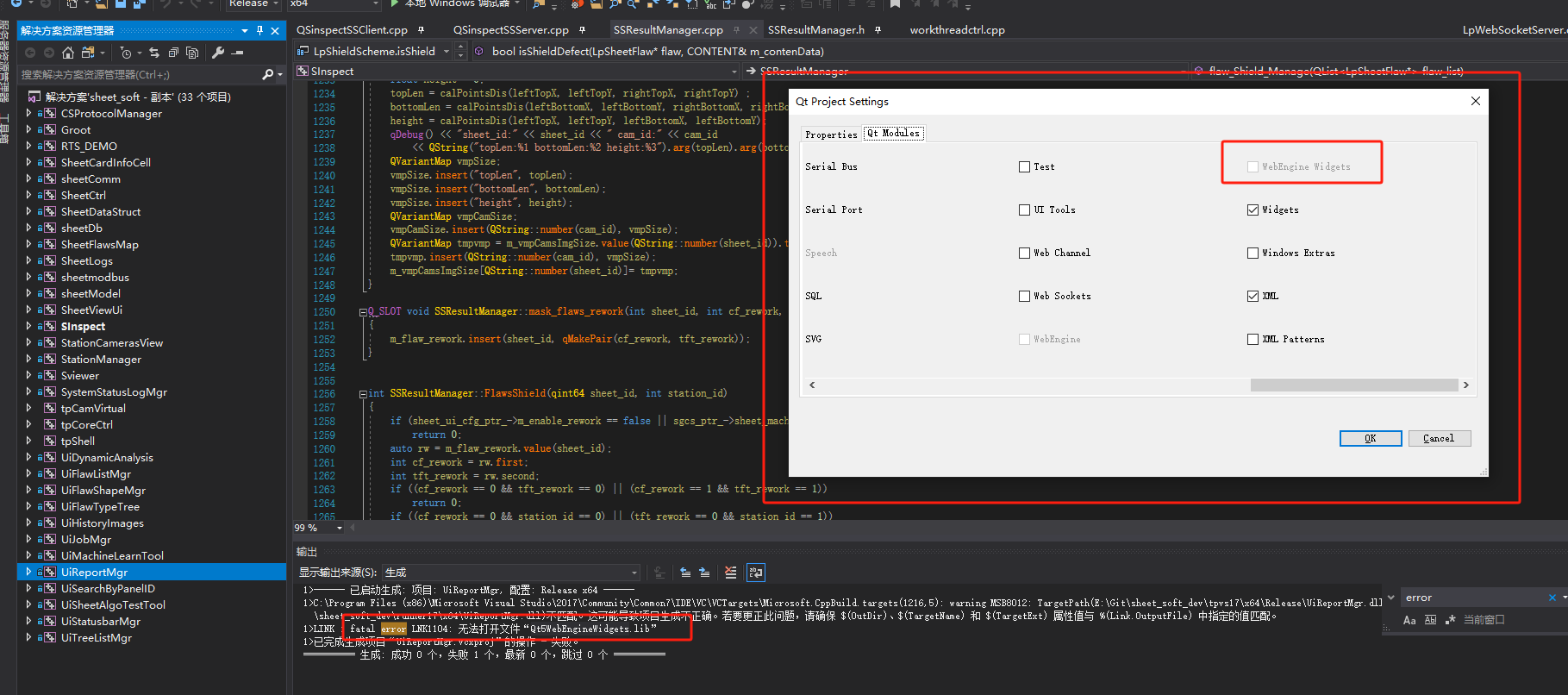
\*/ #ifndef LeeperCameraController\_h #define LeeperCameraController\_h

2.环境配一下

3.sheet\_tadpole-副本.sln 是基础框架（先编译这个），再编译sheet\_soft - 副本.sln

4.24

一. 部署环境过程中遇到的一些问题

1.在编译过程中 遇到了缺少Qt5的库

解决方案：卸载原来的Qt版本，因为此时版本的Qt没有Qt5WebEngineWidgets模块，重新下载Qt5.9.4版本配置，添加这模块就ok了。

2.有两个工程，要先编译底层的工程，再编译上层的。（sheet\_tadpole-副本.sln 是基础框架（先编译这个），再编译sheet\_soft - 副本.sln）。

3.配置数据库的步骤。。。

二、看代码部分

1.std::atomic 是 C++11 标准引入的一个模板类，用于提供原子操作。原子操作是不可中断的操作，要么全部执行，要么完全不执行，不存在执行过程中被中断的情况。用于多线程编程，用来保证对共享变量的操作是原子的，从而避免了竞态条件（race condition）和数据竞争（data race）的问题。作用：保证线程安全。

2.内存顺序标记

std::memory\_order\_relaxed:松散的内存顺序  
std::memory\_order\_consume:消费者内存顺序  
std::memory\_order\_acquire:获取内存顺序  
std::memory\_order\_release:释放内存顺序  
std::memory\_order\_acq\_rel:获取-释放内存顺序  
std::memory\_order\_seq\_cst:顺序一致内存顺序,提供了最强的内存顺序保证

3.Q\_ASSERT是 Qt 框架中提供的一个用于断言的宏,用于在运行时检查条件是否为真，如果条件为假，则输出错误消息并终止程序的执行。注意：只在调试版本中起作用，在发布版本中会被忽略。

4.Q\_DECLARE\_METATYPE用于在元对象系统（Meta Object System）中注册自定义的数据类型,以便这些类型可以被 QVariant 类型所支持，从而能够在信号和槽之间传递。

4.25

1.片料机逻辑图学习。

2.新建工单代码部分熟悉。

3.自动切换工单需求（先想想思路），还有协议部分。

4.qna3E部分代码熟悉。

5.加密狗？

也称为加密锁、硬件锁或安全锁，是一种用于保护软件应用程序安全的硬件设备。它通常是一个小型的硬件设备，类似于 USB 闪存驱动器，与计算机通过 USB 接口连接。

工作原理涉及以下几个方面：

1.加密算法 2.授权管理 3.硬件特征 4.安全通信

6.QTcpSocket？

5.28

示例代码

void qna3e\_tcp::send\_binary\_data(const QByteArray& data) { if (socket\_ptr\_->state() != QAbstractSocket::SocketState::ConnectedState) return;

if (m\_pTimer\_send) {  
 // 开启了发送时钟功能  
 m\_ls\_send << data;  
 return;  
}  
if (m\_bPrint)  
 qDebug() << "data=" << data << \_\_FUNCTION\_\_;  
socket\_ptr\_->write(data.constData(), data.size());  
socket\_ptr\_->flush();

}

以下是代码解释：

这段代码是Qt中使用TCP套接字进行网络通信的一部分，封装在qna3e\_tcp类的一个成员函数send\_binary\_data中，目的是向已建立连接的TCP套接字发送二进制数据。下面逐行解释代码的功能：

void qna3e\_tcp::send\_binary\_data(const QByteArray& data)：这是qna3e\_tcp类的一个成员函数，名为send\_binary\_data，用于发送二进制数据。它接受一个QByteArray类型的参数data，这个参数包含了要发送的二进制数据。

if (socket\_ptr\_->state() != QAbstractSocket::SocketState::ConnectedState) return;：这一行检查TCP套接字（通过socket\_ptr\_指针访问）的状态是否为ConnectedState。如果套接字没有处于已连接状态，函数将直接返回，不进行数据发送。这是为了避免在无连接状态下尝试发送数据导致的错误。

if (m\_pTimer\_send) {：检查成员变量m\_pTimer\_send是否非空。这个变量似乎用于指示是否启用了某种定时发送功能。

m\_ls\_send << data;：如果启用了定时发送（m\_pTimer\_send非空），则将待发送的数据data添加到成员变量m\_ls\_send中。m\_ls\_send可能是一个队列或列表，用于收集待发送的数据。

return;：在定时发送模式下，函数在将数据添加到待发送列表后直接返回，不立即执行发送操作。

if (m\_bPrint) qDebug() << "data=" << data << \_\_FUNCTION\_\_;：如果成员变量m\_bPrint为true，这行代码将使用qDebug()打印出待发送的数据和函数名。这可能是用于调试目的。

socket\_ptr\_->write(data.constData(), data.size());：使用socket\_ptr\_指针（指向TCP套接字）调用write方法，将data中的内容发送出去。data.constData()返回指向数据缓冲区的指针，data.size()返回数据的大小，单位是字节。

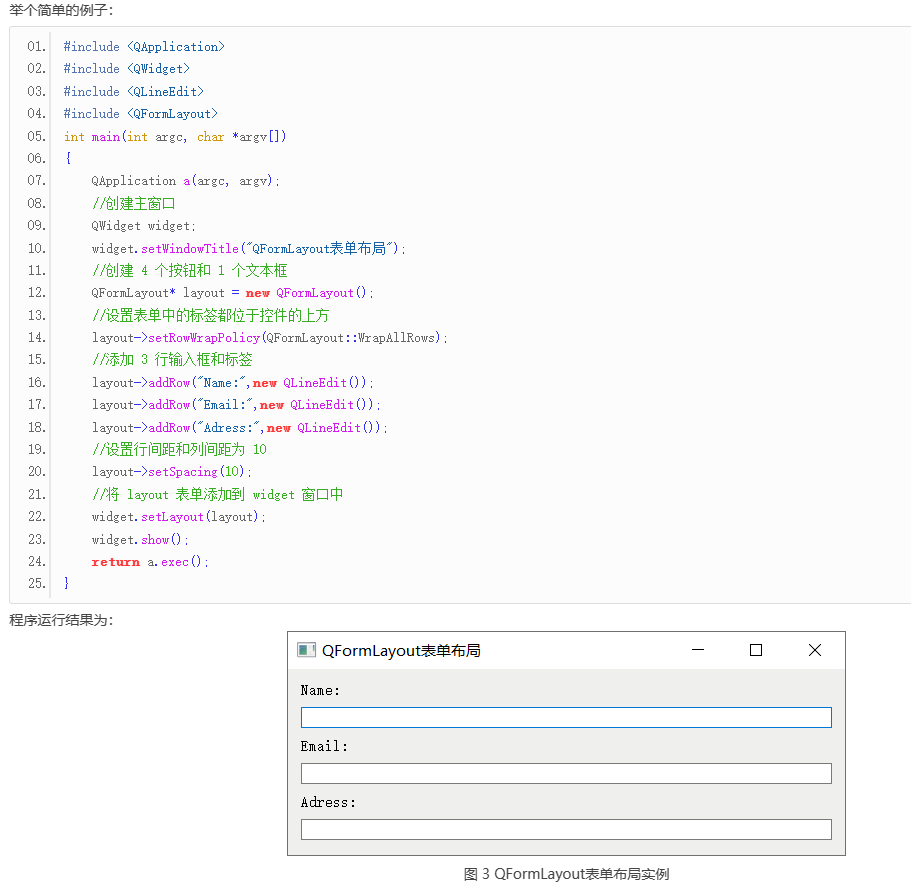
socket\_ptr\_->flush();：调用flush方法以确保所有发送的数据都被立即写入底层网络缓冲区，并尝试发送出去。这可以确保数据不会在套接字的输出缓冲区中延迟。

总结来说，这段代码实现了向TCP连接发送二进制数据的功能，并提供了定时发送和调试打印的选项。在发送数据前，它会检查TCP连接的状态，以确保通信通道正常。如果启用了定时发送，则将数据暂存而不是立即发送。

7.QFormLayout（表单布局）？

QFormLayout 是 Qt 提供的一种布局工具（或者理解为一种布局控件）

举个例子：



8.QCompleter

基于模型的补全功能，能实现自动填充功能,方便用户输入,提升用户的体验,一般和QLineEdit与QComboBox搭配起来使用。

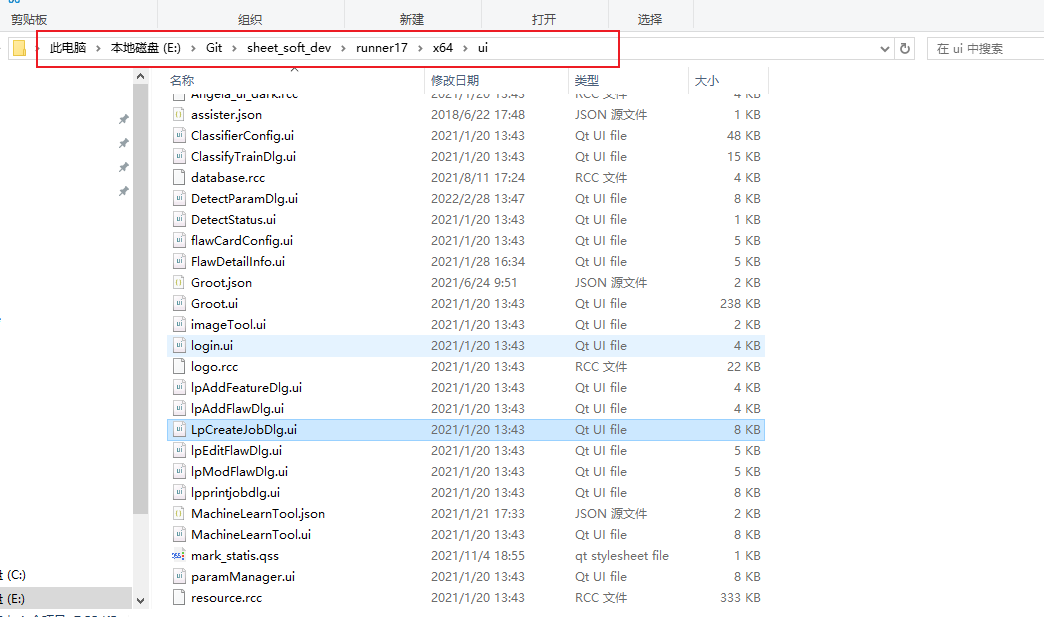
示例：QStringList word\_list; word\_list<<"XiaoTu"<<"xiaomi"<<"Huawei"<<"huafei"<<"Shanghai"<<"shangshan"<<"abc";

QCompleter \*completer = new QCompleter(word\_list, this); ui->lineEdit\_Simple->setCompleter(completer);

原文链接：<https://blog.csdn.net/sabcdefg/article/details/112257474>

4.26

1.找ui文件 ---> 打开ui文件夹





根据ui文件的变量名去 代码中搜索。

4.28

1.了解上位机开发的基本流程。

2.学习上位机想要快速突破：1.编程基础 2.通信方式 3.项目经验。

3.异常调试分析方法：使用windbg工具，结合dump文件跟pdb文件分析使用。

4.AOI同步启动逻辑图熟悉：1.lpUtils是干什么用的？

5.PagerDoll工程文件说明图熟悉。

6.看代码，先看头文件，了解大概的一个功能流程，遇到问题了再去细看代码。

7.sys日志工具

8.QCache:是 Qt 框架中用于缓存数据的类。

QCache 可以存储键值对数据，并提供了快速的查找和访问方法。它内部使用哈希表来组织数据，以保证数据的快速查找和访问。

以下是 QCache 类的一些常用功能和方法：

插入数据：insert(key, value) 方法用于向缓存中插入一个键值对数据。

查找数据：value(key) 方法用于查找指定键对应的值。

删除数据：remove(key) 方法用于删除指定键对应的数据。

清空缓存：clear() 方法用于清空整个缓存。

设置最大容量：setMaxCost(cost) 方法用于设置缓存的最大容量，当缓存中的数据数量超过最大容量时，会自动删除一些数据。

获取缓存大小：size() 方法用于获取缓存中数据的数量。

获取缓存容量：maxCost() 方法用于获取缓存的最大容量。

缓存状态查询：contains(key) 方法用于判断缓存中是否包含指定键对应的数据。

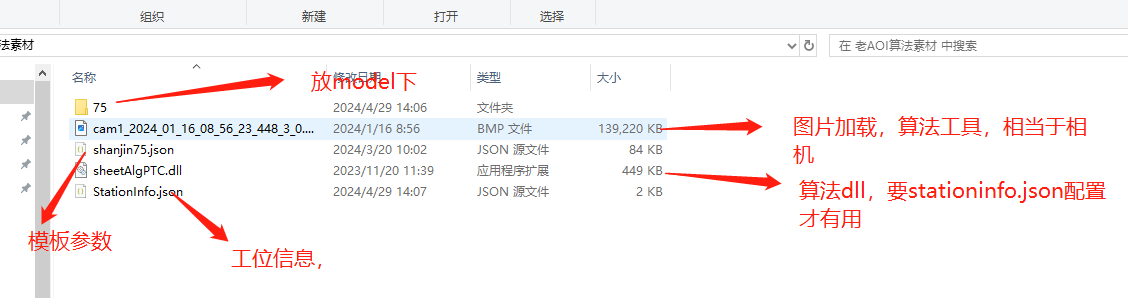
4.29

1、任务一：片料老软件，服务端创建缺陷流程图或文档。

1.1、void on\_depiler\_sheet（）；//主要负责图像检测结果：异常处理，缺陷屏蔽，尺寸检测，并将结果显示到界面上。

软件操作：

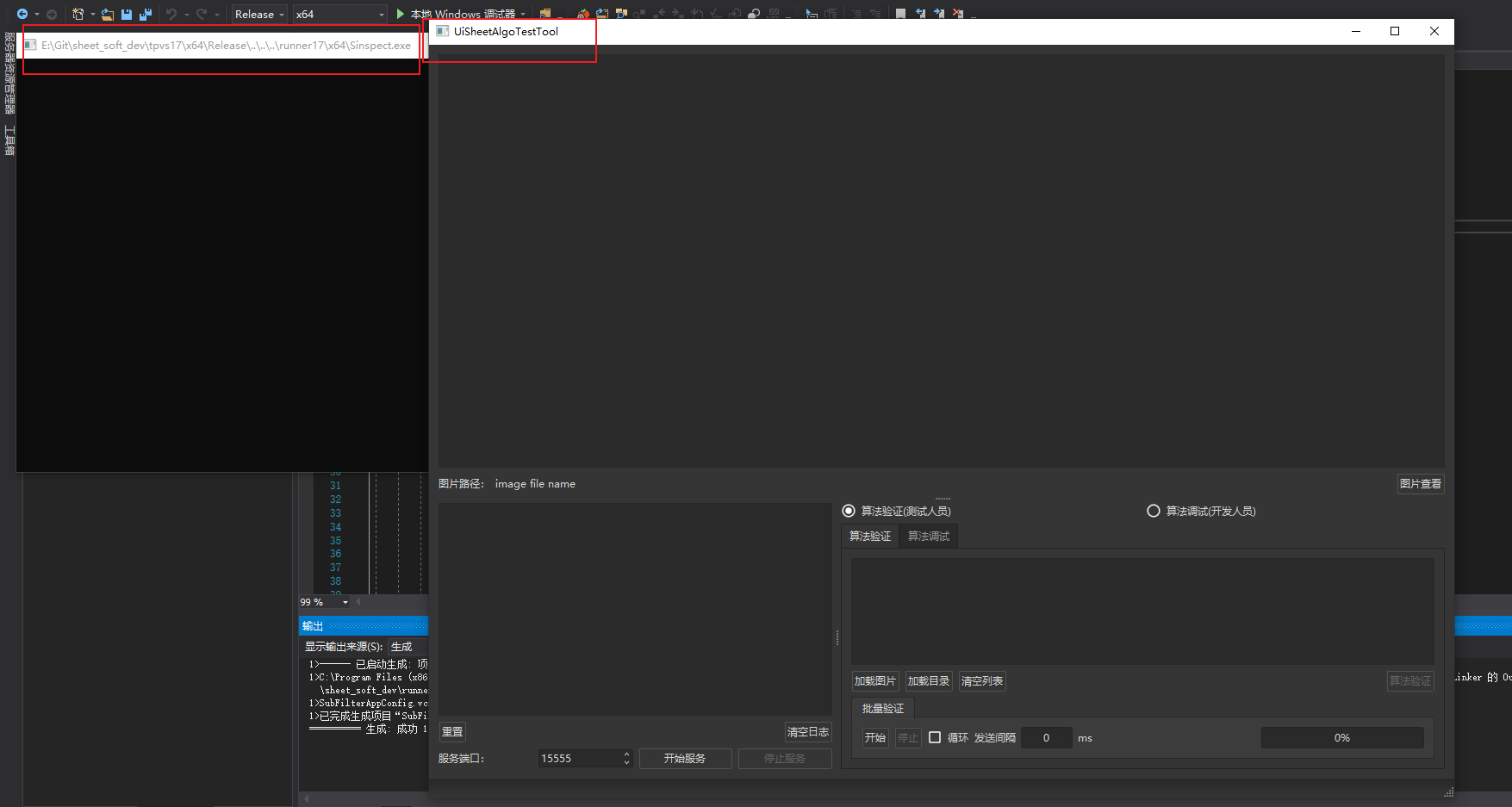
1.打开软件-->新建工单-->

-->

4.30

1.遇到的编译问题



原因：原来软件的两个exe没有关闭

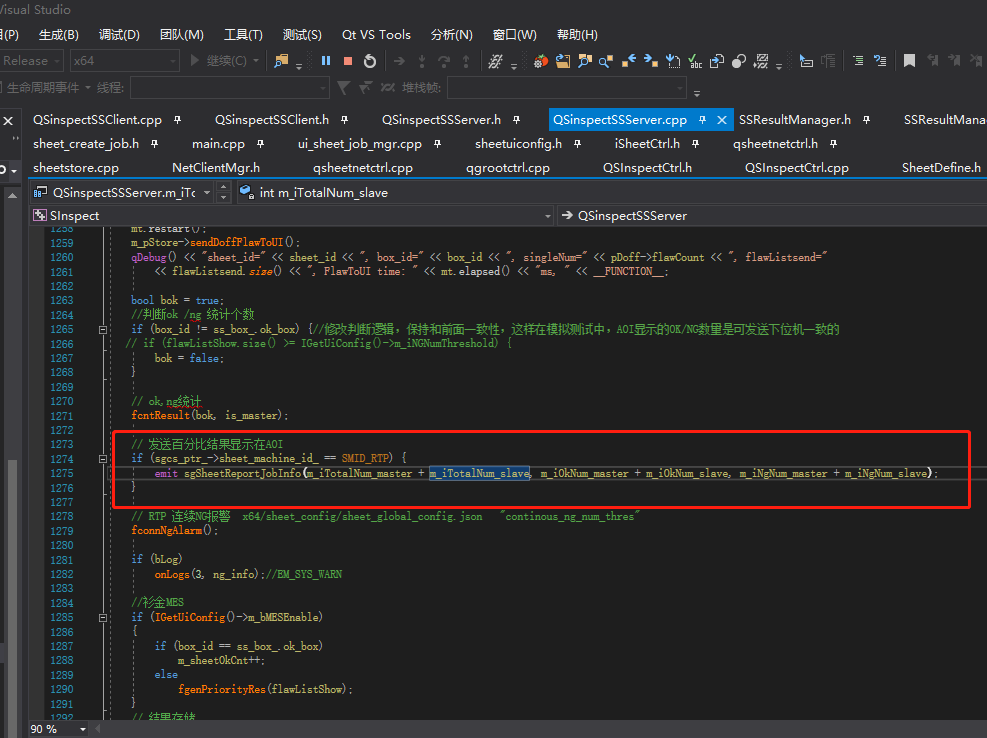
导致编译失败，进程占用了，会涉及操作文件，得关闭，问题得到解决。

0506

1.大小尺寸产品屏蔽缺陷逻辑图。

2.AOI：检测缺陷的软件。

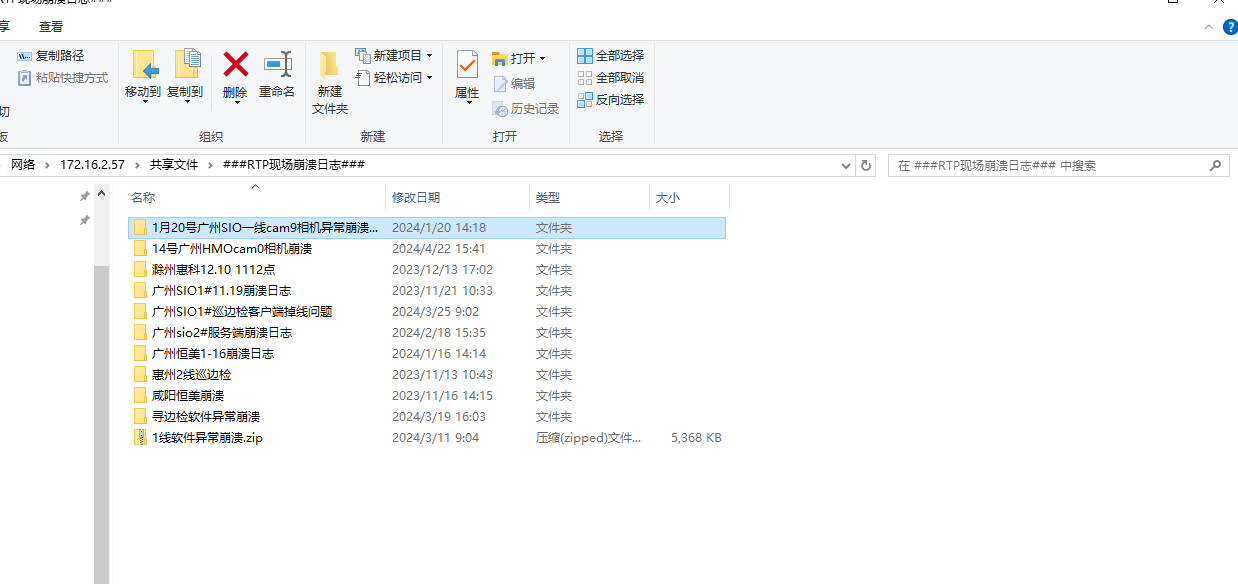
3.RTP中panel\_id：



0507

1.RTP的panel\_id 完整流程图

2.void qna3e\_tcp::process\_read\_rtp\_panel\_id(QString& data\_recv, int head\_len),发出panel\_id信号的地方（读取panel\_id的地方）

3.

4.RTP是什么？

实时传输协议，是用于在互联网上传输多媒体数据的网络协议。它通常用于音频和视频的实时传输，如VoIP电话、视频会议等。

0508:

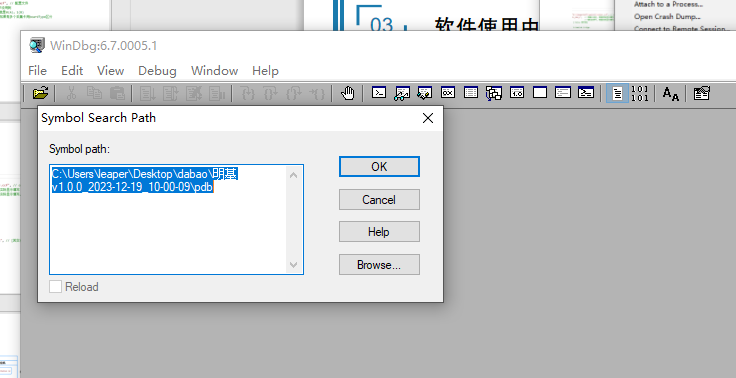
1.WinDbg：是一个调试器，可用于分析故障转储、调试实时用户模式和内核模式代码，以及检查 CPU 寄存器和内存。（非常重要！！！以前用的不多，这个如果有需要 可分享）

WinDbg是在windows平台下，强大的用户态和内核态调试工具。它能够通过dmp文件轻松的定位到问题根源，可用于分析蓝屏、程序崩溃（IE崩溃）原因，是我们日常工作中必不可少的一个有力工具，学会使用它，将有效提升我们的问题解决效率和准确率。

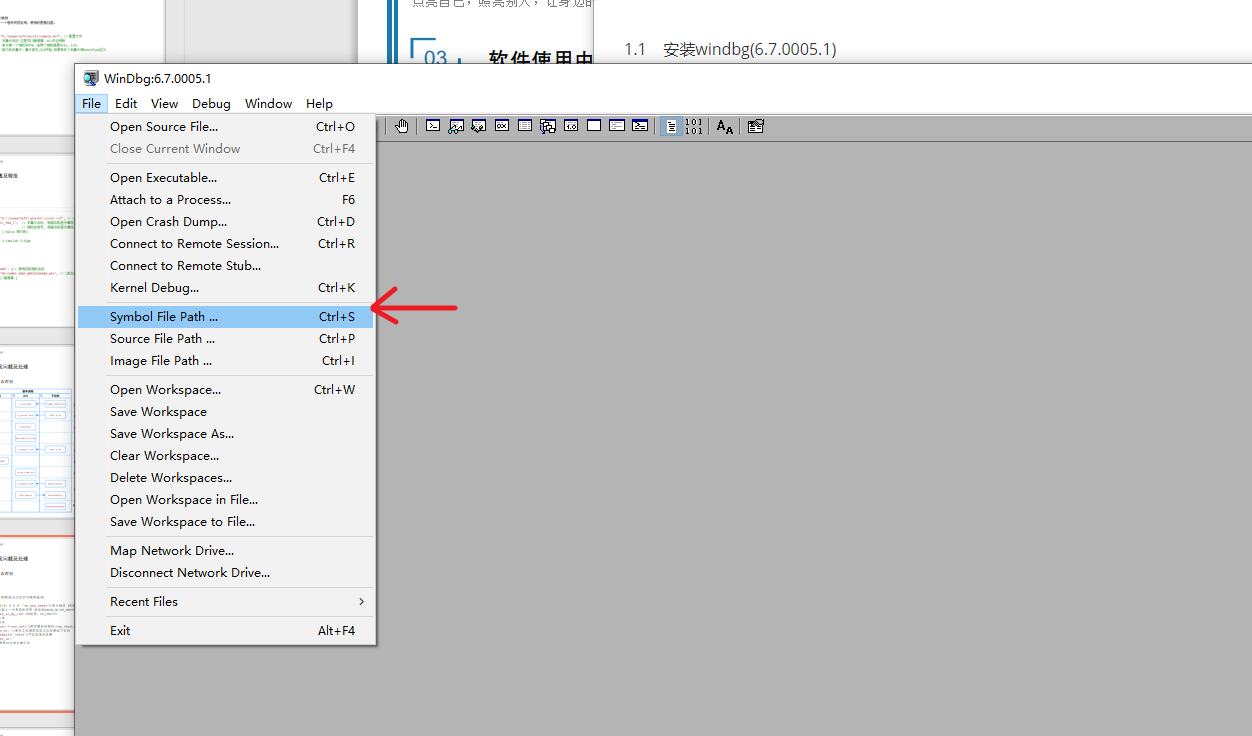
1.1 安装windbg(6.7.0005.1)

1.2 dmp文件 时间戳转换，定位问题时间点

1.3 需要注意点 把.dmp文件



放到该目录下，并且保持版本号一致才能定位 查找问题



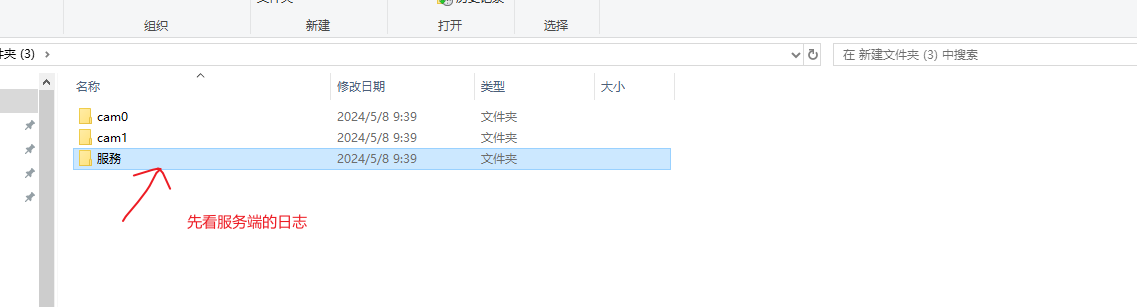
2.分析崩溃日志

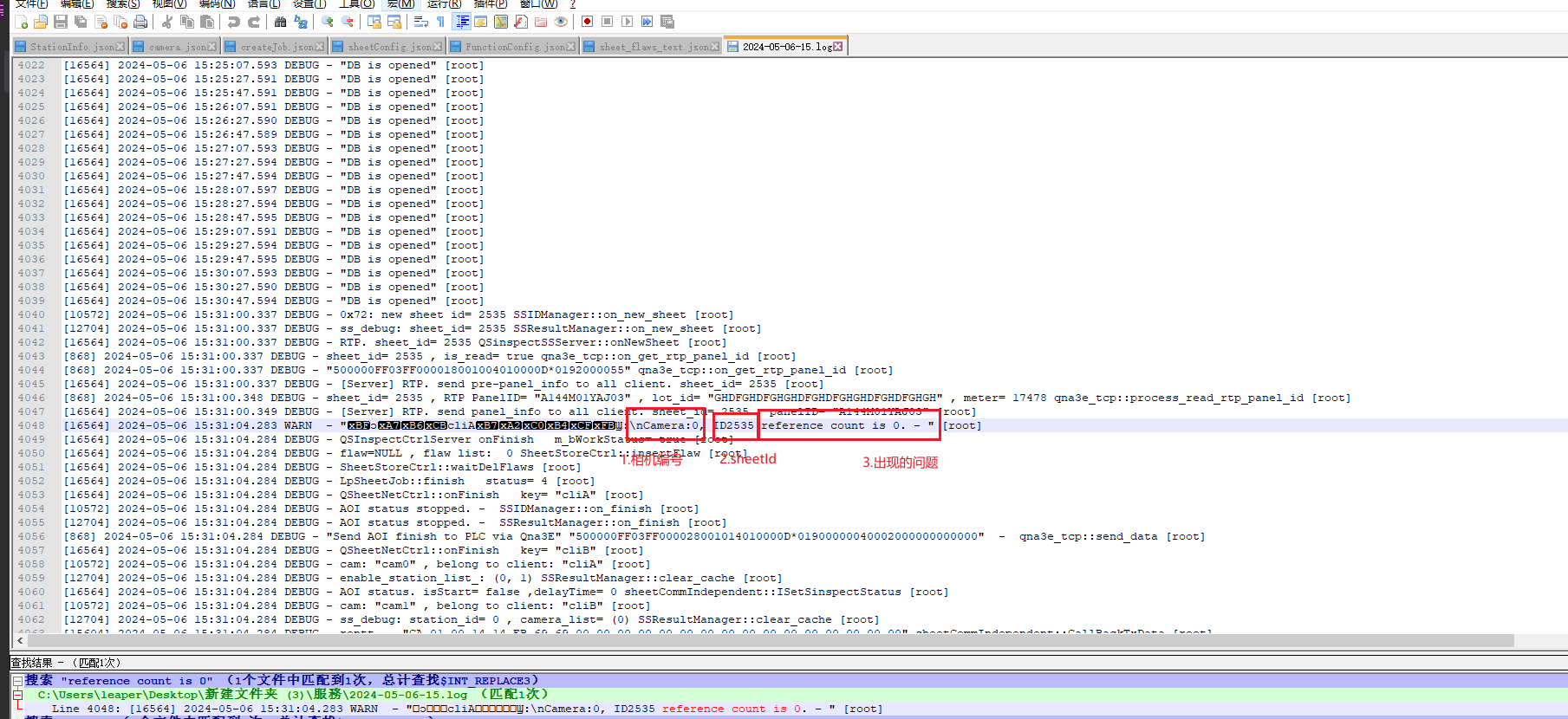


2.1 先看版本号！！！

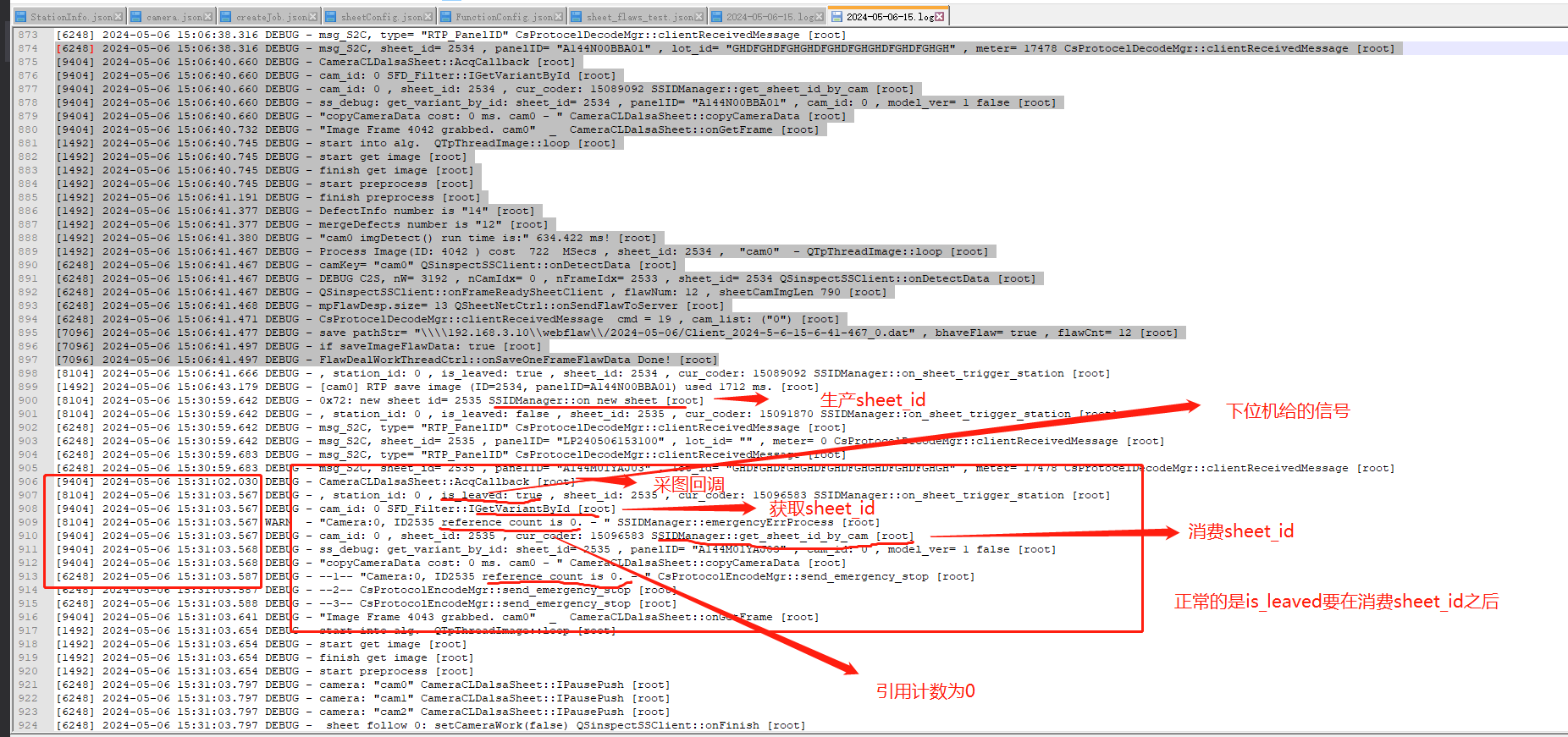
app v(中间有个空格！！！)

2.3 分析步骤梳理

1.

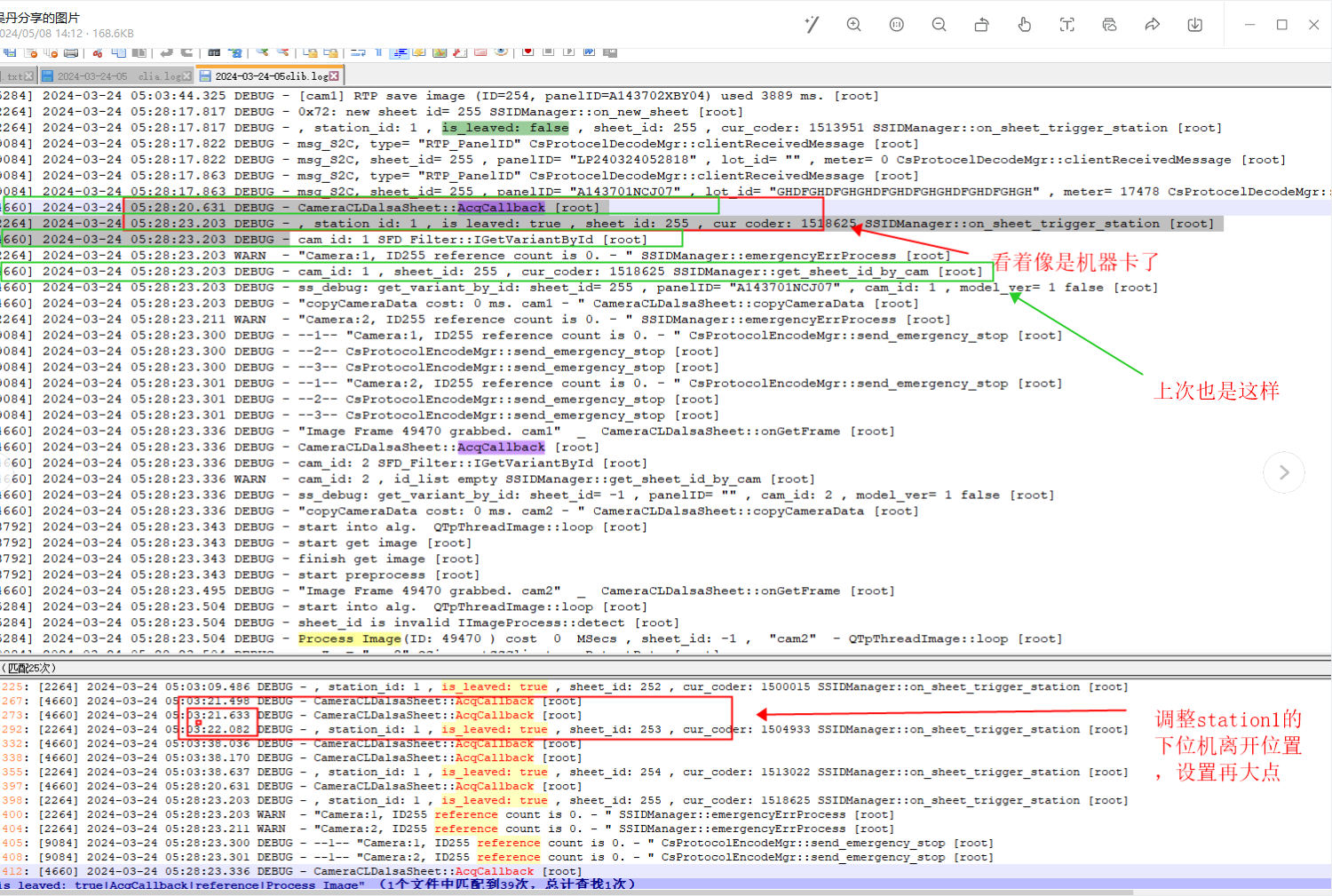
2

从上图可以发现是编号为0的相机出现了异常，我们这时候可以直接去查看cam0相机的日志，搜索引用计数为0的地方（引用计数为0不是奔溃，是软件异常保护机制）

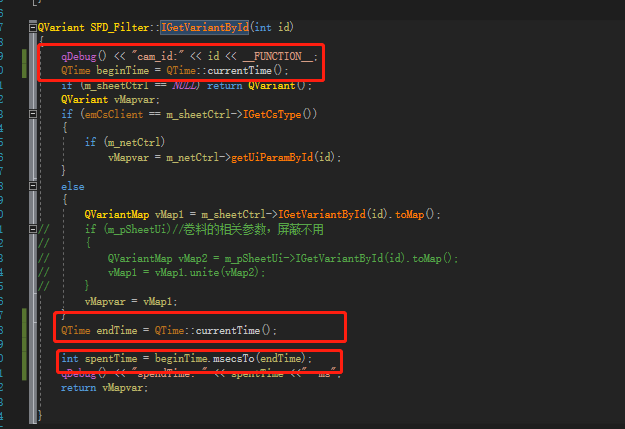


sheet\_id是跟相机拍的片子绑定的。(原本是callBack之后就直接绑定，不应该有1s多的延迟)中间时间间隔有1.5ms，查一下到底是什么原因？关注线程ID为９４０４

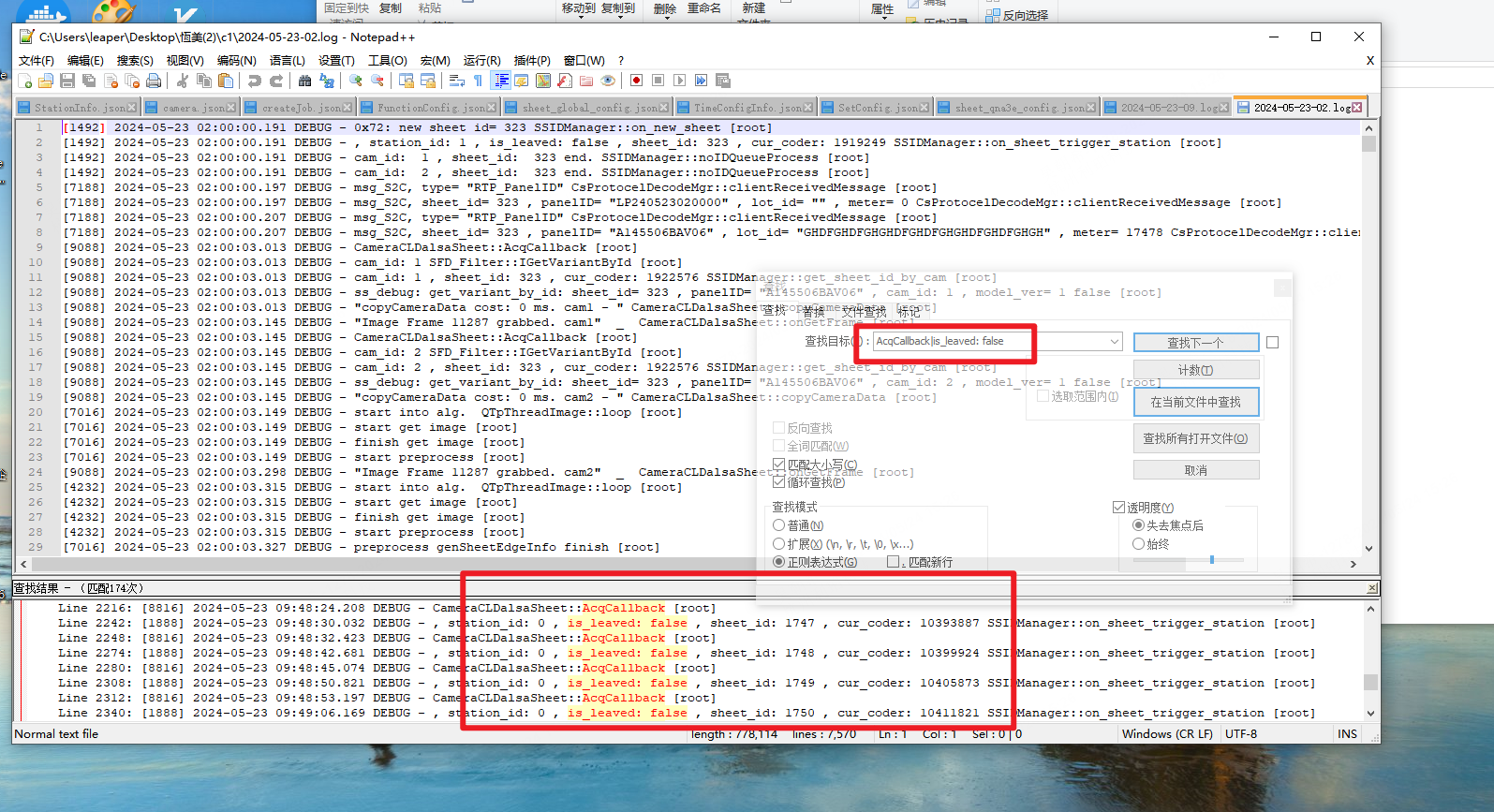
上次出现的问题：



３、ＱＴｉｍｅ，打印一下　函数运行耗时？如何使用？



5.24更改非队列模式后排查日志



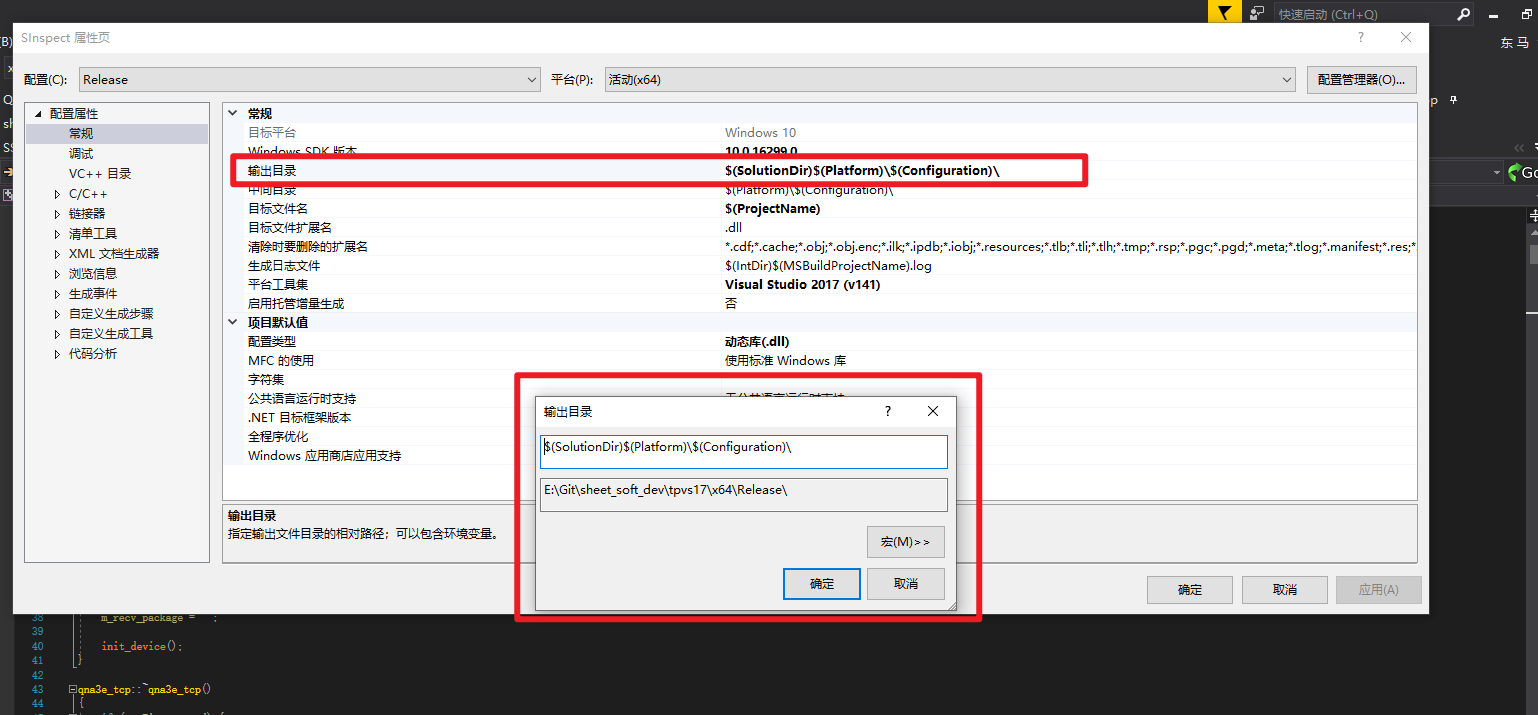
排查异常点，不是引用计数为0的问题。

看is\_leaved:false是否在AcqCallback之前（正常是在之前的）。如果is\_leaved:false跟两个AcqCallback，证明是双相机。

0509:

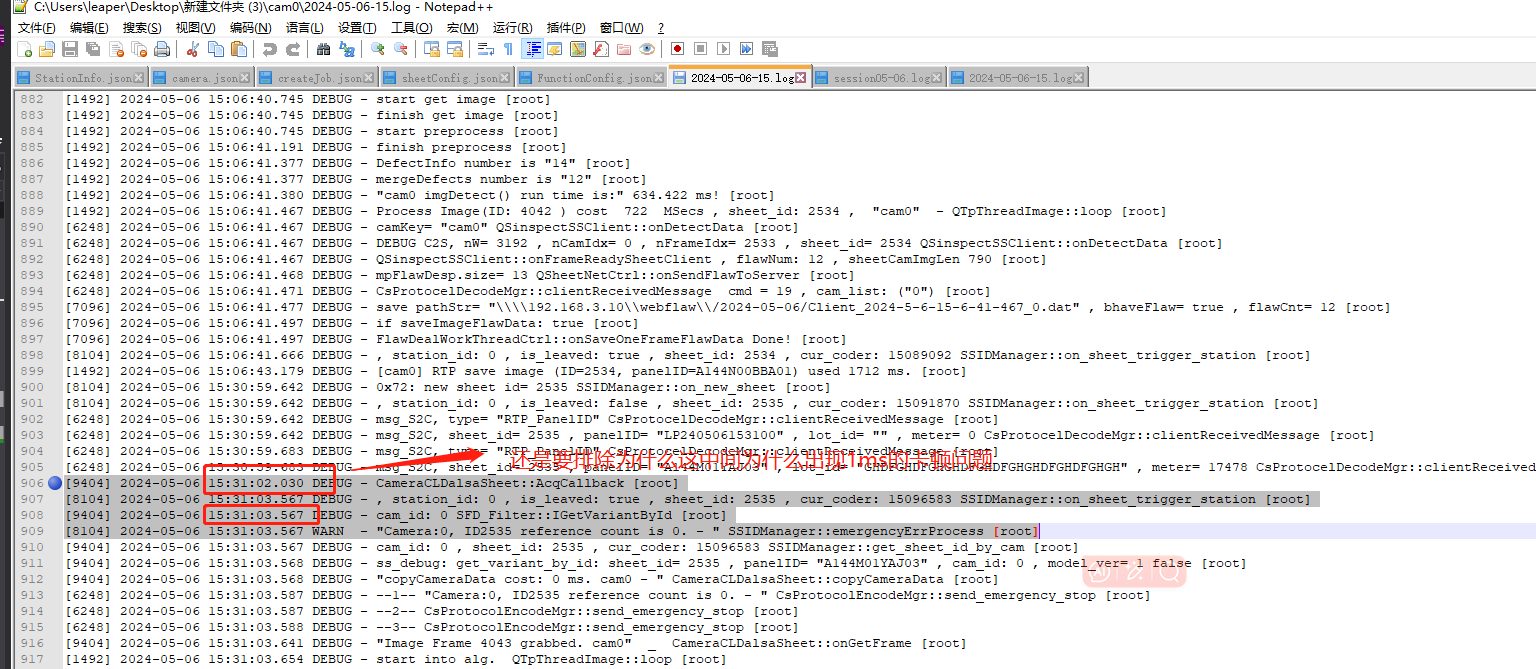
1.片料重构软件环境搭建

工作运行目录

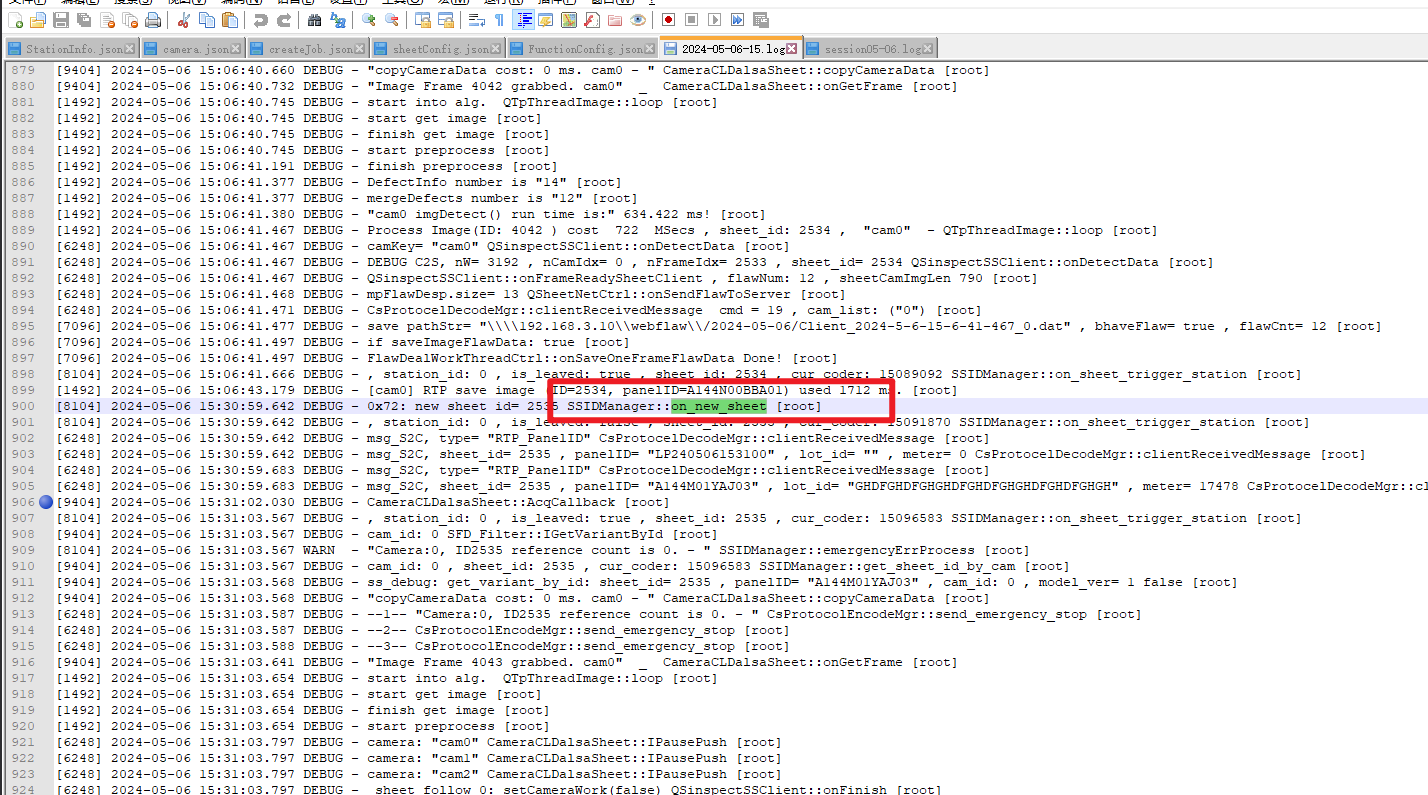


0510:

1.查现场日志 引用计数为0的问题



看客户端和服务端的日志



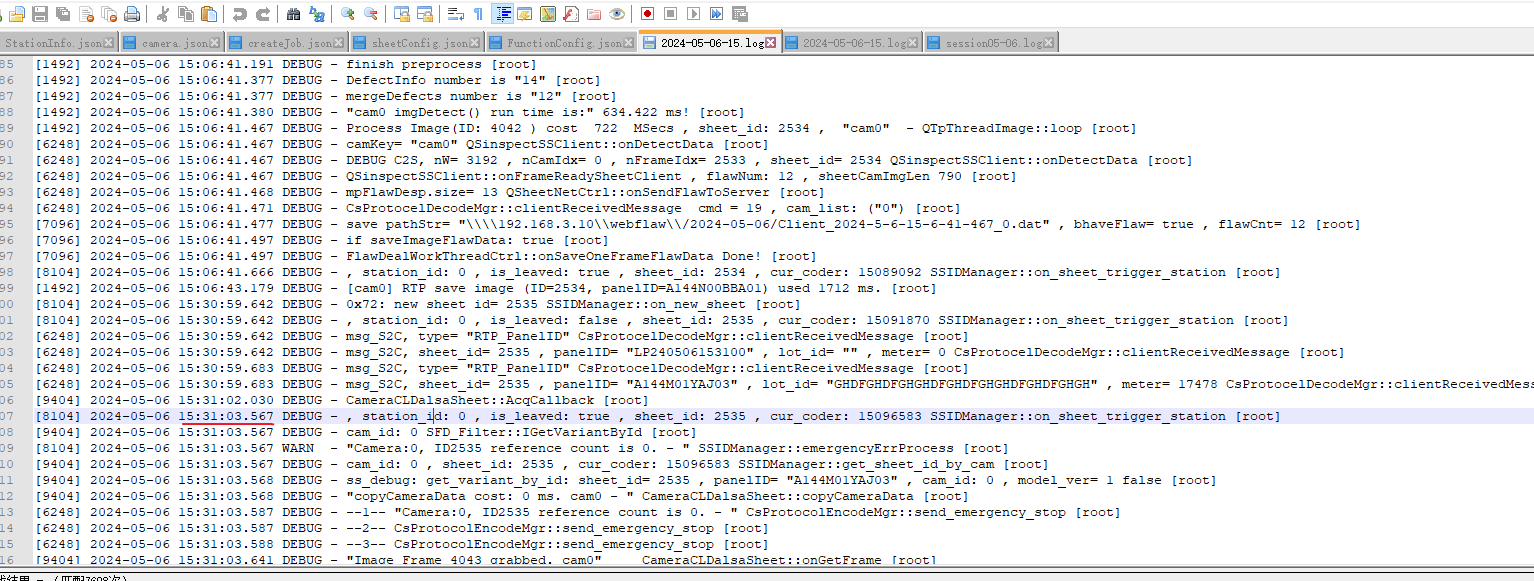
比较一下时间

1.先用on new sheet 对应出 服务端和客户端的时间差

2.看下位机的After\_Up\_ID的发出时间和软件的is\_leaved: true 的时间

客户端 服务端

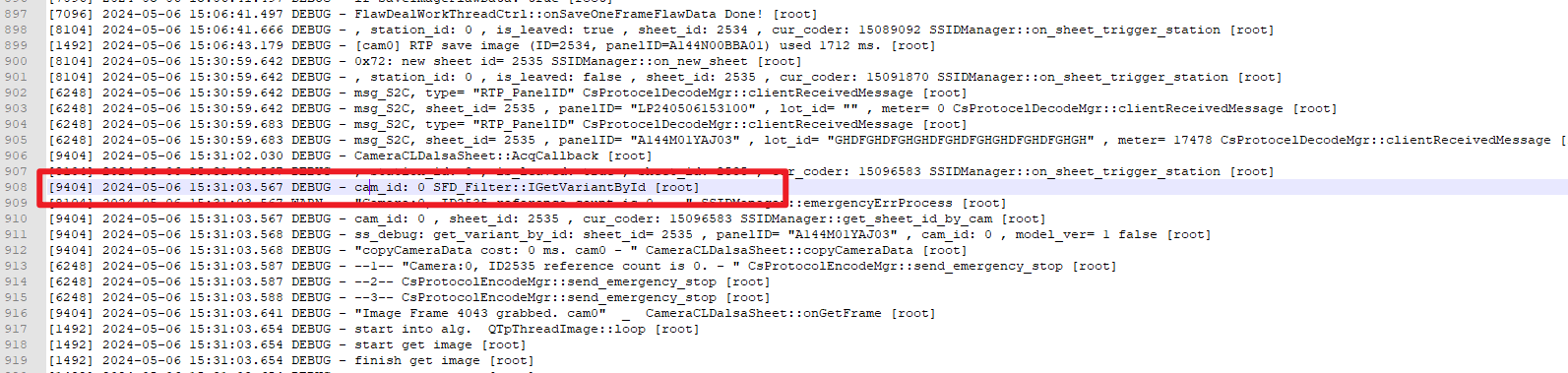
15:30:59.642 15:31:00.337 服务端-客户端时间 695ms



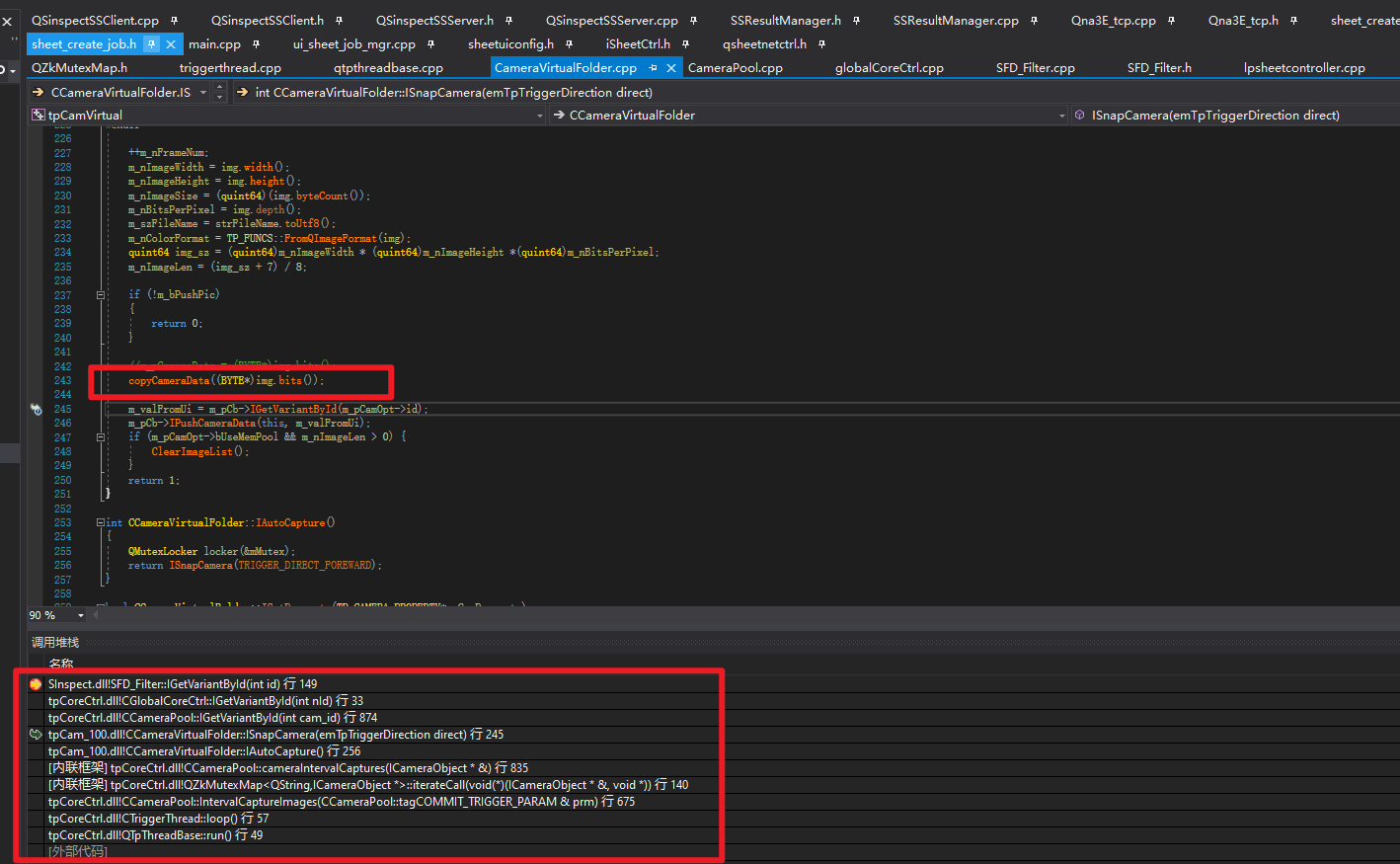
软件发出额is\_leaved:true时间：15:31:03.567

下位机的After\_Up\_ID的发出时间：15:31:03.718

此时时间间隔比较短，认为正常误差内



看日志在哪打印的 ，打印之前 发生了什么，有什么耗时的操作？



貌似跟底层逻辑有关，先不管了。

2.自动切换工单需求理解

5.11

1、RTP广州恒美PLC切型软件自动切换工单 开发文档编写（wiki）

2、

5.13

1、自动切换工单功能开发

2、引用计数为0的问题，队列模式换为非队列模式？

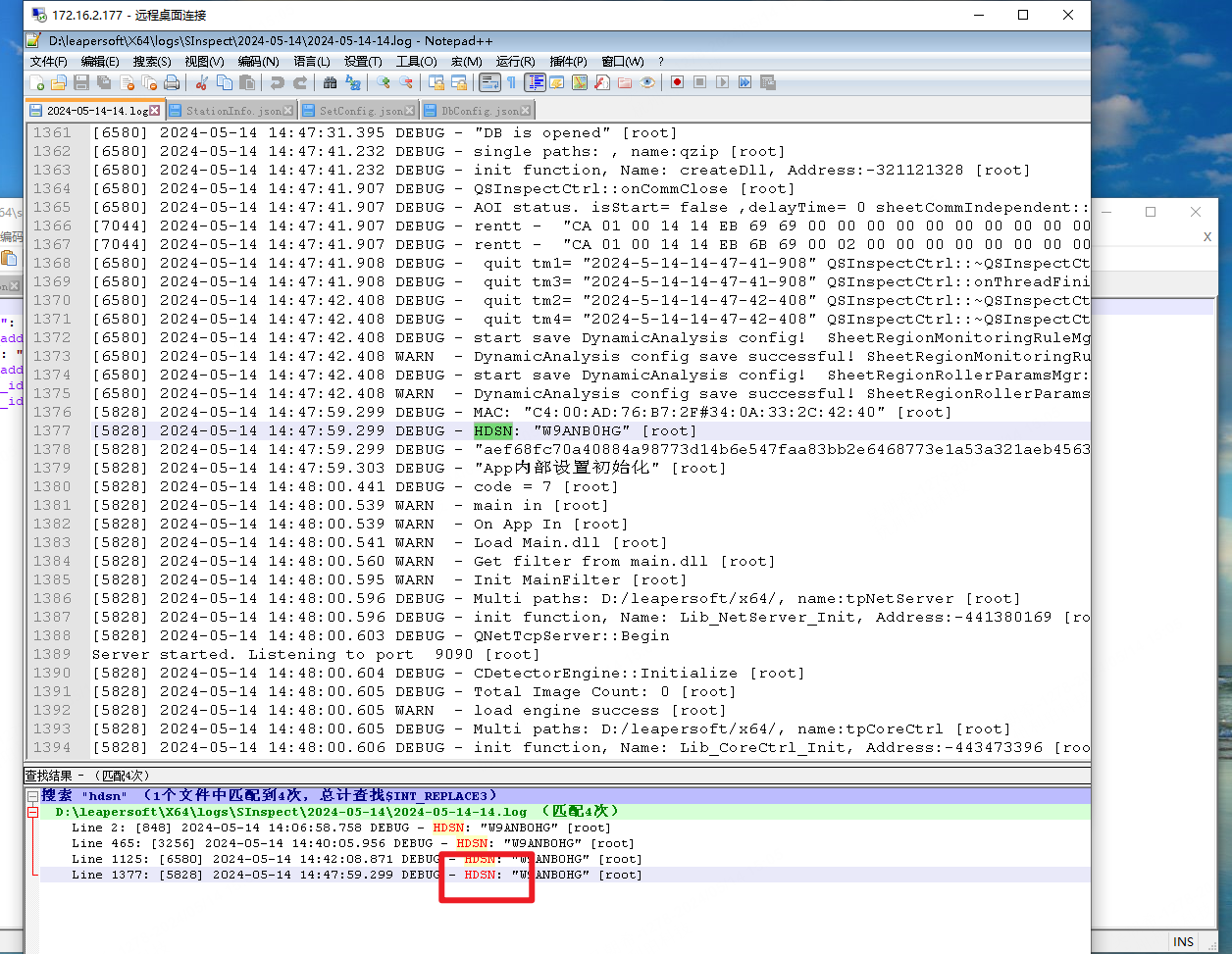
出问题的是，不拼接的图。

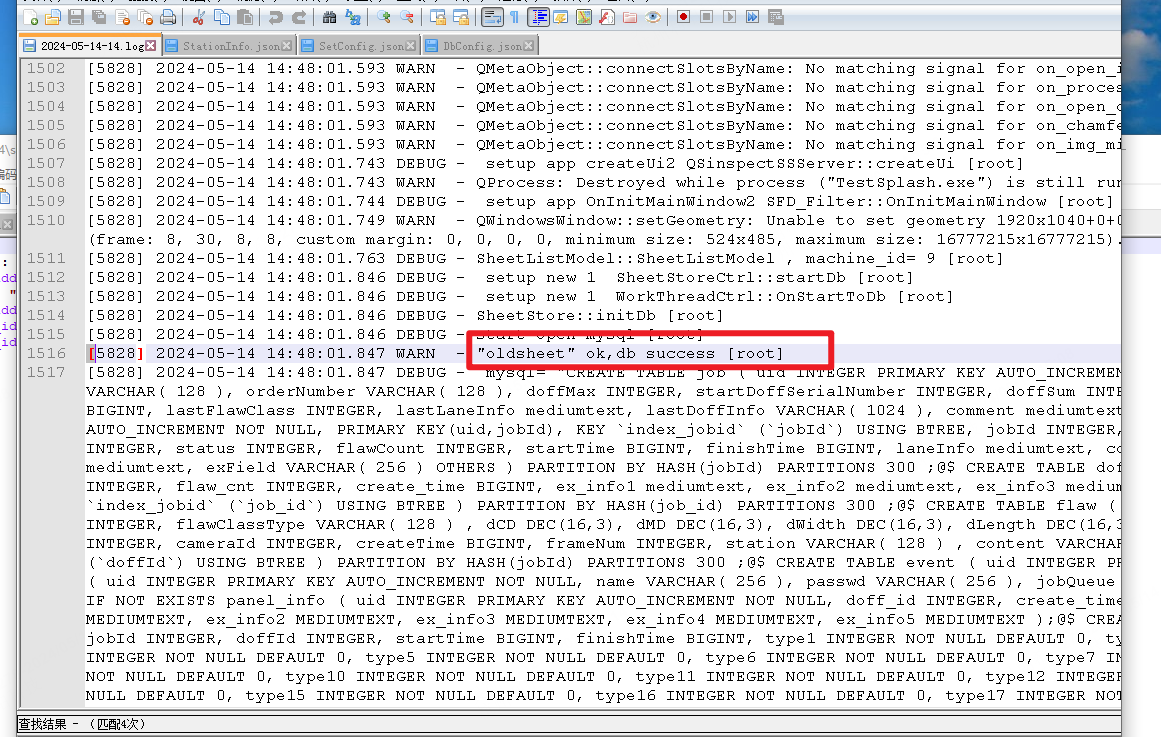
方案1：在获取一些指针之前，打印一下时间。

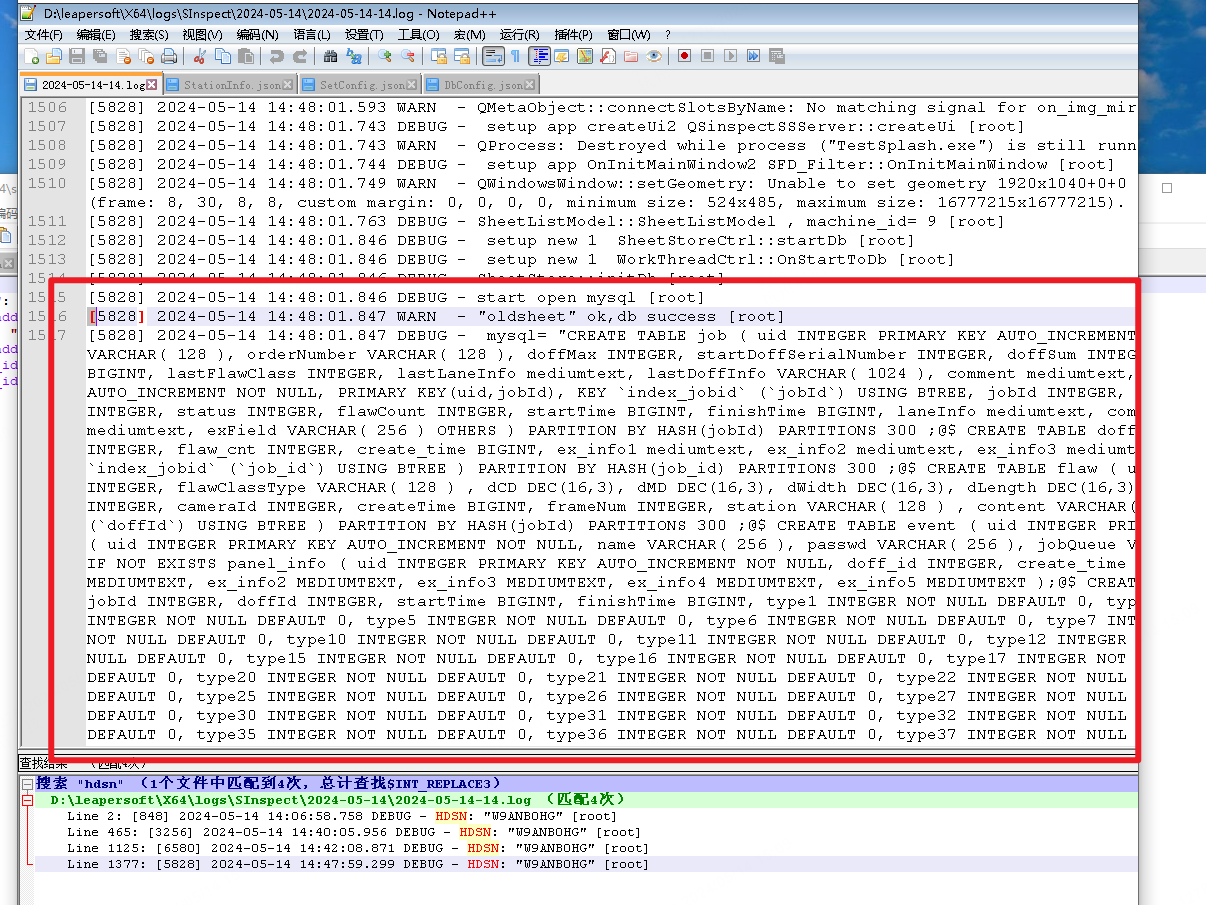
5.14

1、通信协议确定后，如何解析？

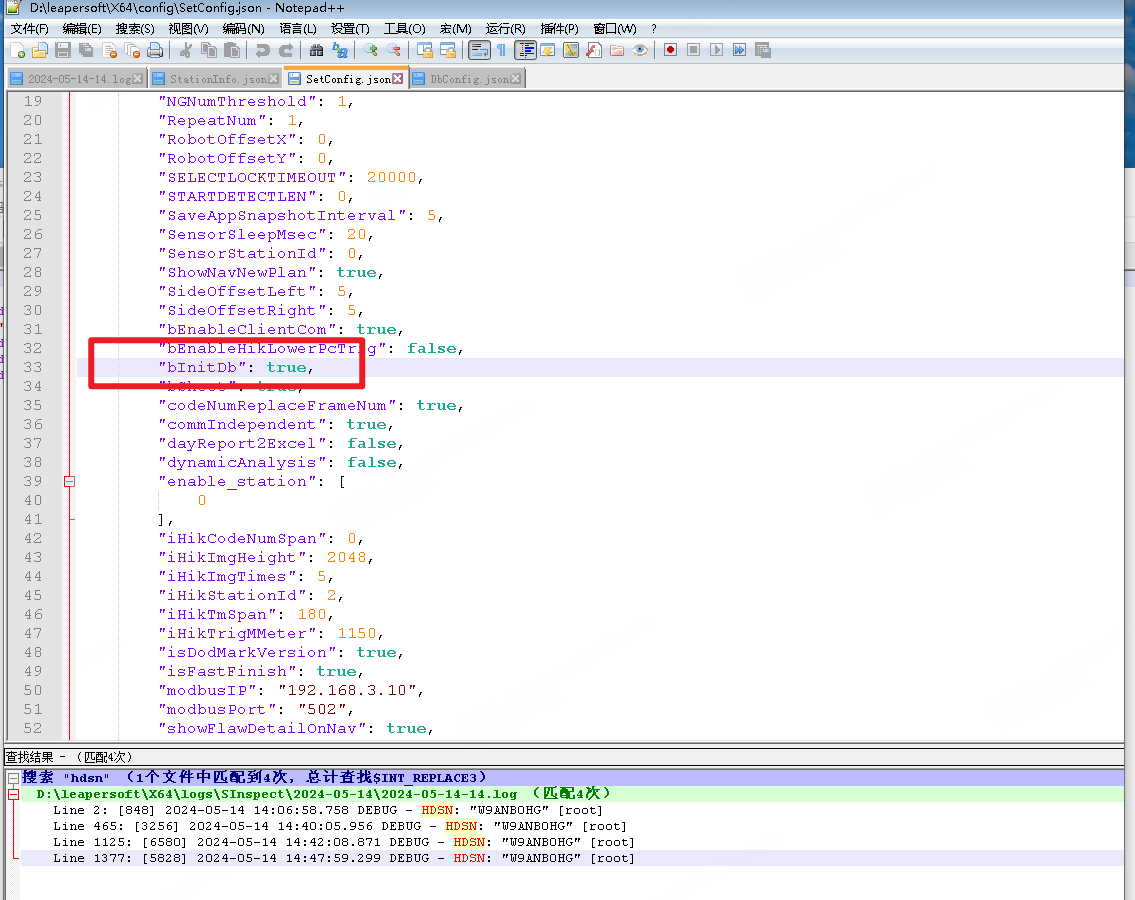
2、排查软件数据库的问题？

2.1看日志

软件刚开始的时候输出的日志-->查找数据库的

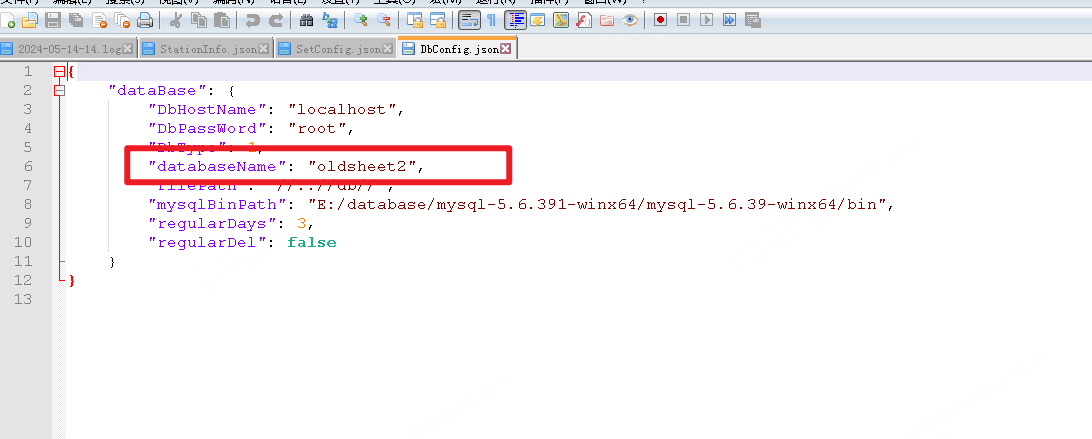


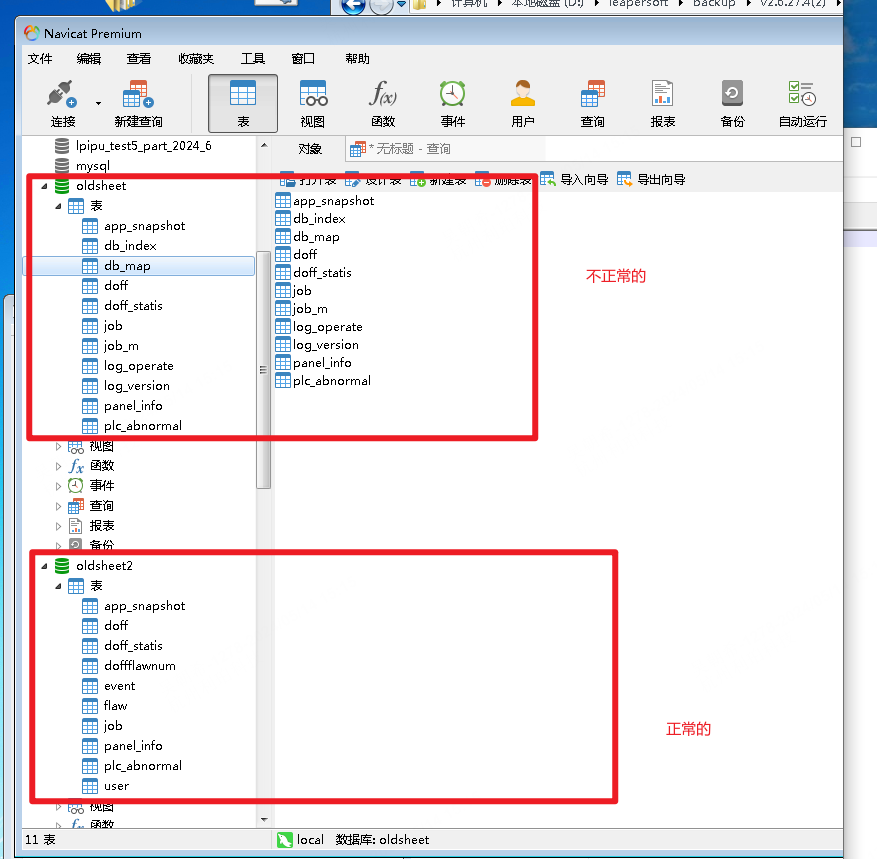
再查

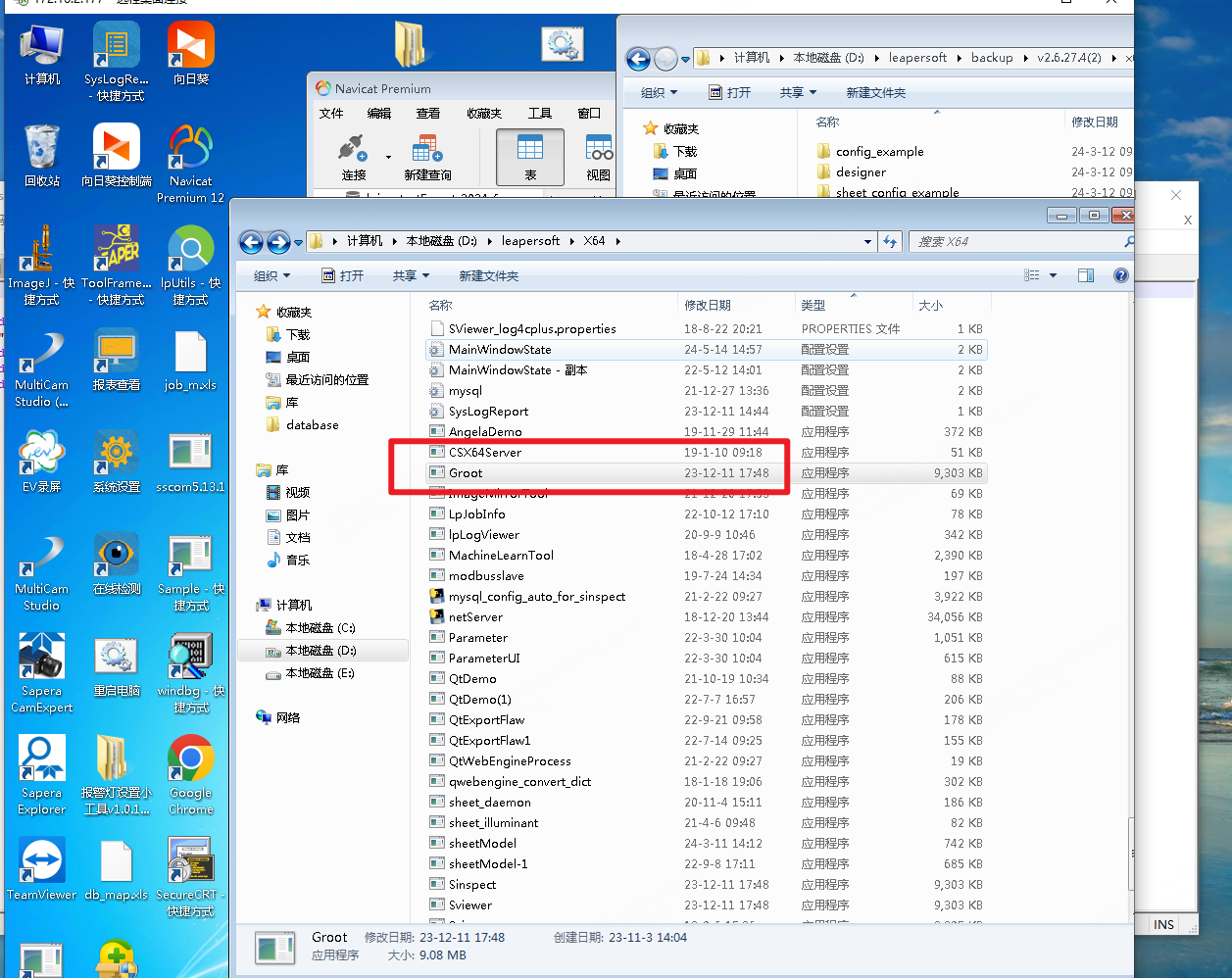


该字段初始化数据库

再看

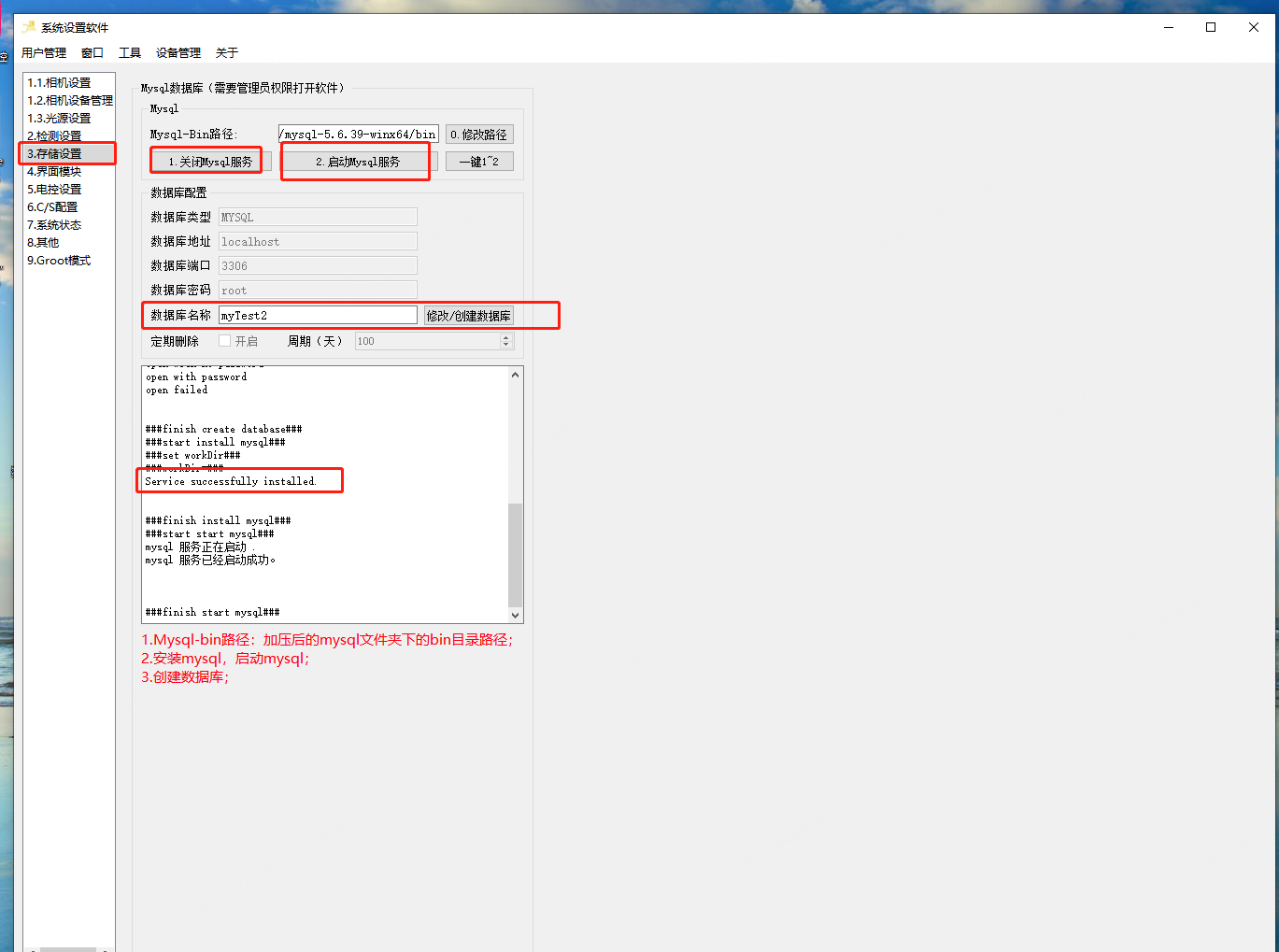


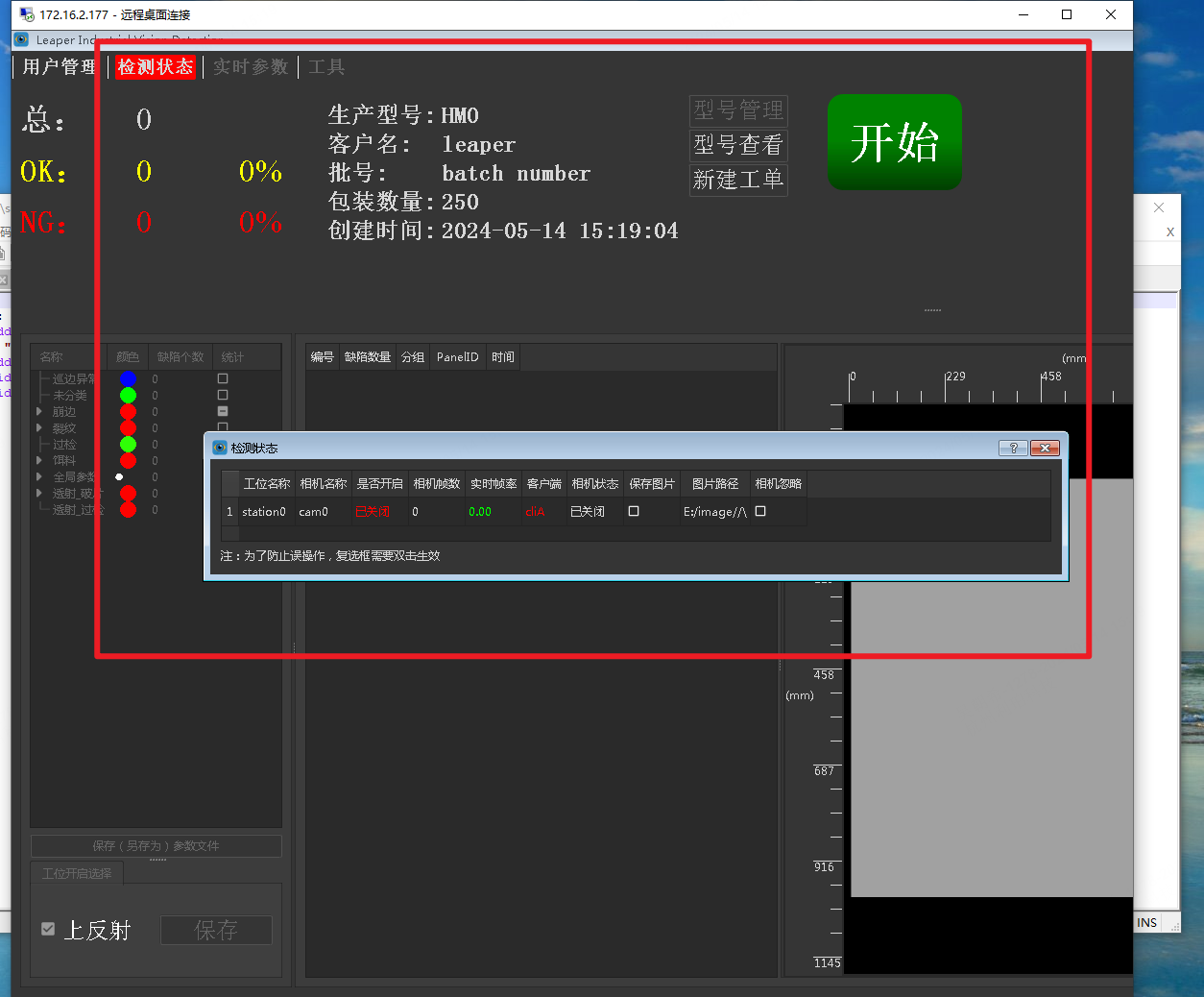




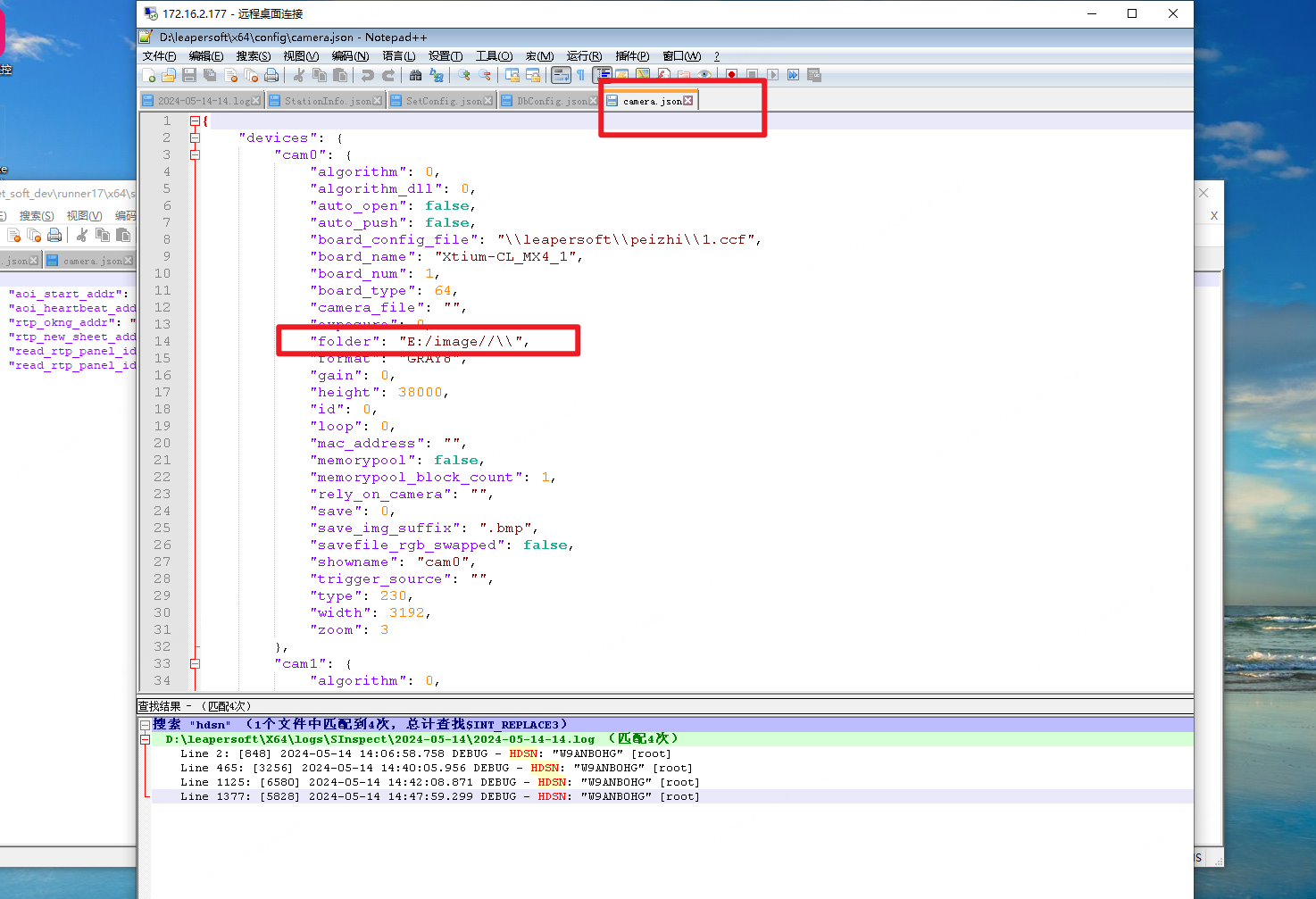
新建数据库

5.24





此时相机有问题



看路径下是否有image这个文件夹

3、图像异常问题排查

3.1 先看服务端 stationInfo的算法，客户端可能没有算法dll,也有可能是dll有问题

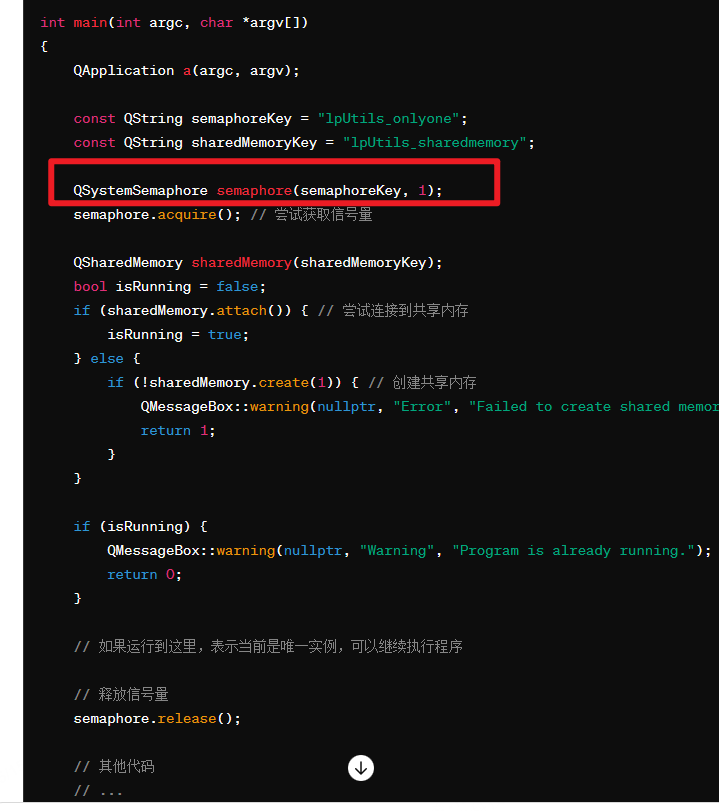
5.15

1、RTP 偏贴后自动删图/部署小工具lpUtils多次启动bug?

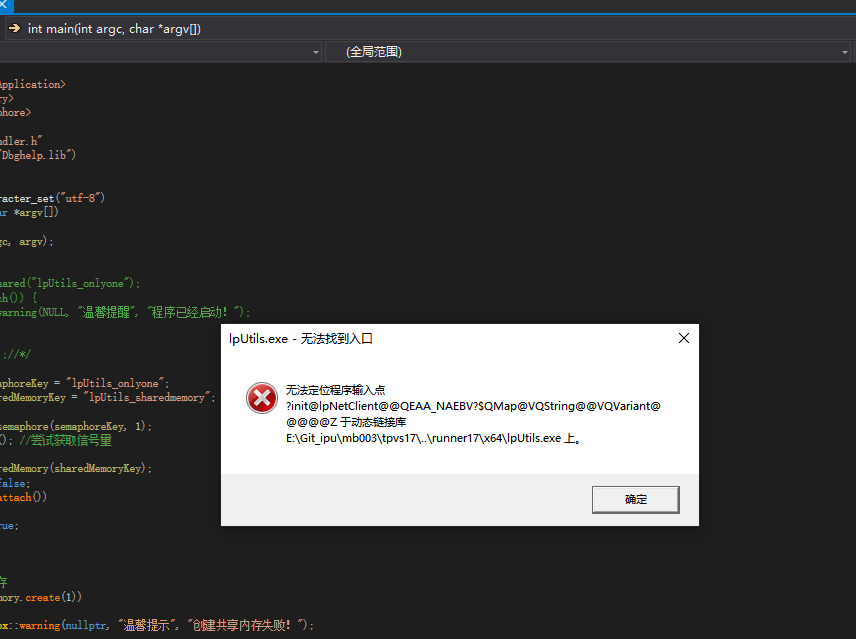
分析问题：

QSharedMemory创建了一个共享内存对象，目的是为了确保只有一个实例运行。通过共享内存来实现进程间的通信，如果有一个实例在运行，那么第二个实例尝试创建共享内存时就会失败，从而可以判断出已经有一个实例在运行，此时就会执行弹出警告对话框并退出。但是，如果第一个实例意外崩溃或者退出时没有正确释放共享内存，就可能会导致第二个实例无法检测到第一个实例的存在，就会启动第二个实例。

解决：尝试获取一个系统信号量，然后尝试连接到共享内存，如果连接失败，就创建一个共享内存。如果连接到共享内存成功，则表示已经有一个实例在运行了，此时就会弹出警告对话框并退出。如果能够成功创建共享内存，则表示当前是唯一实例，可以继续执行程序。在程序退出时，释放系统信号量。

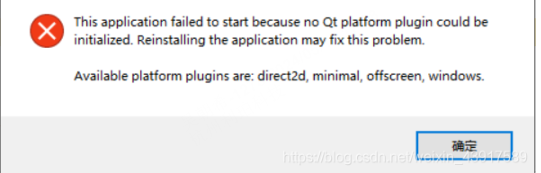


2、本地缺少依赖库，补齐后出现截图问题



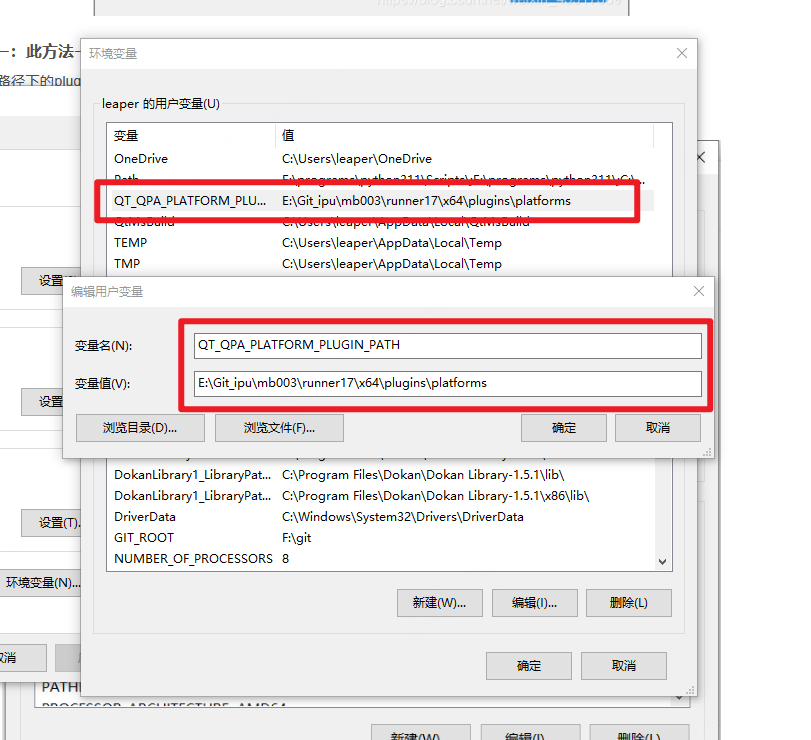
出现的原因：可能是缺少依赖的dll,也有可能是依赖的dll版本不一致。

解决方案：把能够运行程序的运行目录下的东西全拷到这个工程下。

出现了

解决方案1：

路径下的plugins\platforms添加到系统环境变量中用户变量里：如我的路径：E:\Git\_ipu\mb003\runner17\x64\plugins\platforms



变量名：QT\_QPA\_PLATFORM\_PLUGIN\_PATH 变量值：D:\soft\Anaconda\Lib\site-packages\PySide2\plugins\platforms 不同用户，变量名一致，变量值随文件位置变。 最后，保存变量后，一定要重启电脑，因为新的环境变量要重启后才能被系统识别！

原文链接：<https://blog.csdn.net/weixin_43917589/article/details/106137952>

5.16

1、RTP 偏贴后自动删图/部署小工具lpUtils多次启动bug?问题尚未解决，还得定位

2、QProcess?

在Qt中，我们可以使用QProcess类来操作进程。这个类包含一系列功能丰富的函数，用于启动并控制外部进程。

2.1 QProcess注意事项

（1）、关于start和startDetached

在QProcess中，start()和startDetached()函数提供了两种不同的启动进程的方式。虽然他们在大部分情况下能达到相同的效果，但在某些方面有着重要的不同。具体如下：

生命周期：start()函数创建的进程的生命周期与创建它的QProcess对象的生命周期相绑定。也就是说，当QProcess对象被销毁时，由它启动的进程也会被销毁。相比之下，startDetached()函数则会创建一个独立的进程，即使创建它的QProcess对象被销毁，启动的进程也会继续运行。

输入/输出：只有start()启动的进程可以使用QProcess类的一些重要功能。例如，使用setStandardInputFile()设置进程的标准输入，或者用readAllStandardOutput()和readAllStandardError()来获取进程的标准输出和标准错误。而startDetached()启动的进程则没有这样的功能，它的输入/输出不能由创建它的QProcess对象所控制。

进程间通信：start()启动的进程可以利用QProcess的信号/槽机制进行通信，例如用readyReadStandardOutput()和readyReadStandardError()信号获取到输出信息，并进行处理。而startDetached()则无法实现此类通信。

所以说，从上面的说明中，如果你需要对创建的进程进行控制的话，推荐使用start()函数。而如果你只是需要创建一个独立的进程，那么startDetached()可能会是更好的选择。

原文链接：<https://blog.csdn.net/lizhong2008/article/details/136840521>

5.17

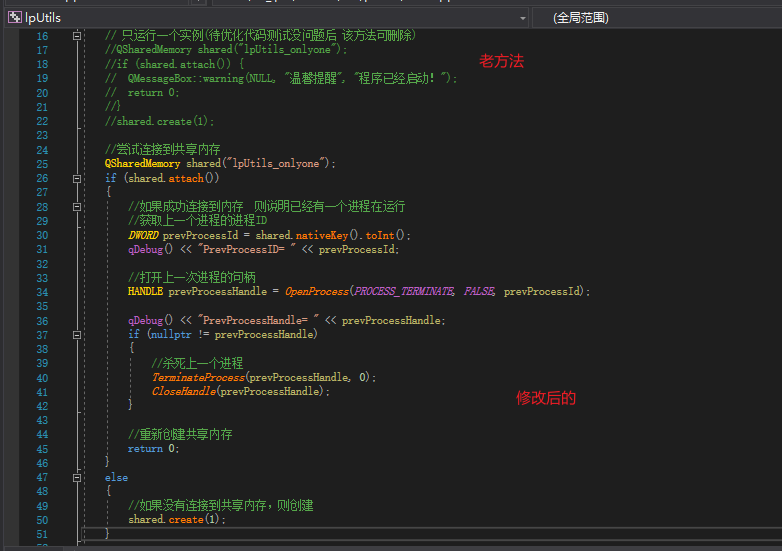
1、流程图添加Wiki文档(√)

2、用SingleApplication替代QSingleApplication方案，刚开始编译不过，[如何解决 “fatal error C1083: ”无法打开包括文件\_fatal error c1083: 无法打开包括文件: “jpeglib.h”: no such -CSDN博客](https://blog.csdn.net/qq_33266987/article/details/51878975)出现该情况。

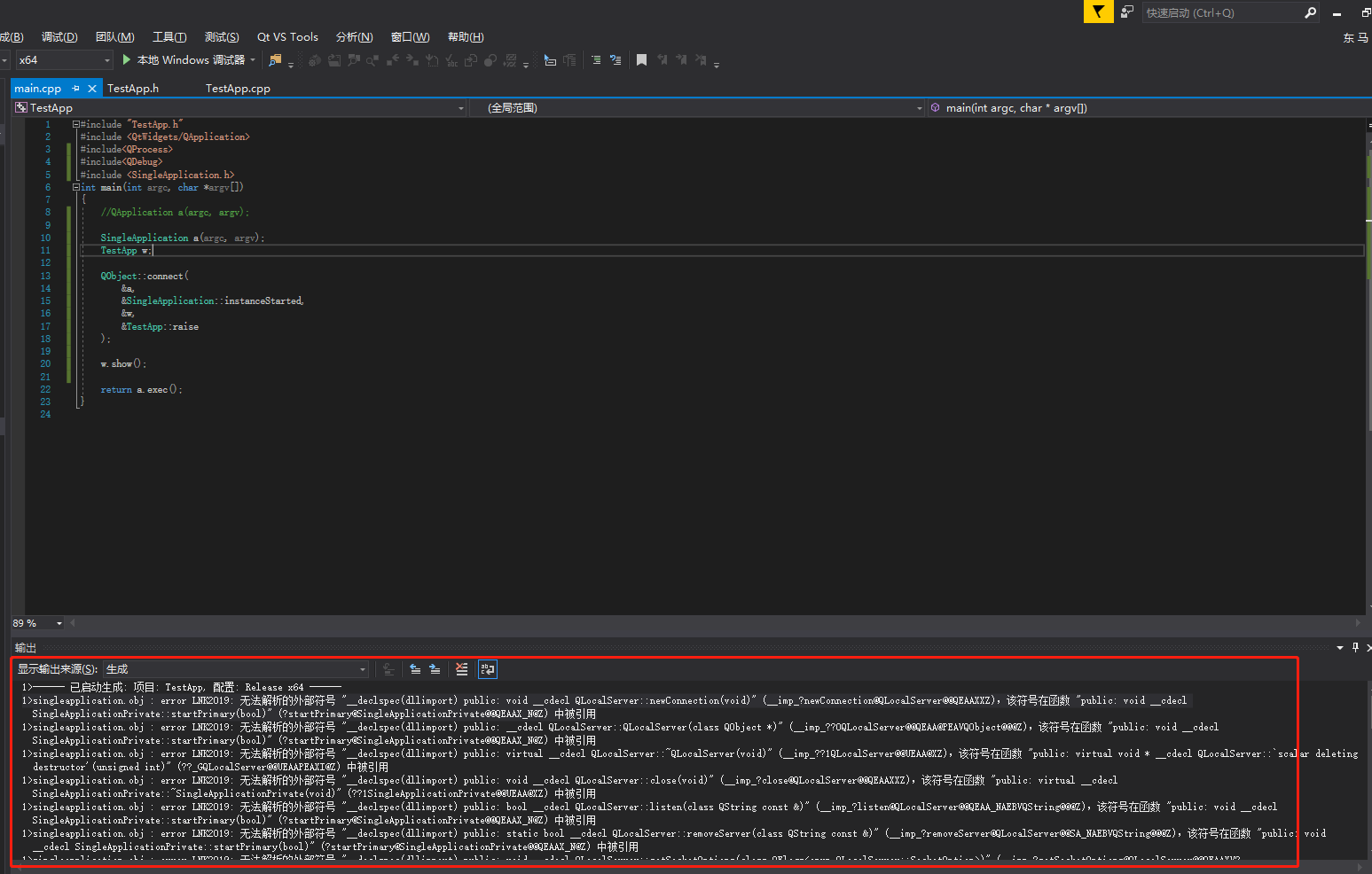
能编译过之后，但是出现了崩溃的情况，如图

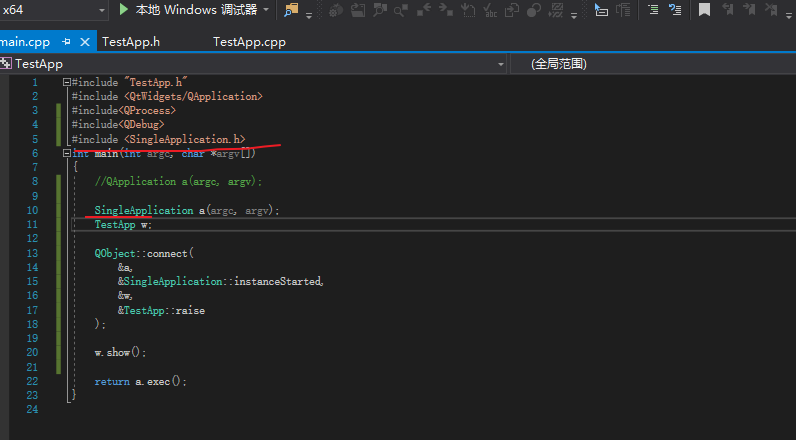


3、exe多开问题，问题可能是上一次的线程没有结束掉。



4、在一个新建的exe中，添加SingleApplication，文件都能编译过，但是运行却显示





报这个错误一般是因为使用了一些未定义或者没找到定义的函数或者变量等等。

5.20

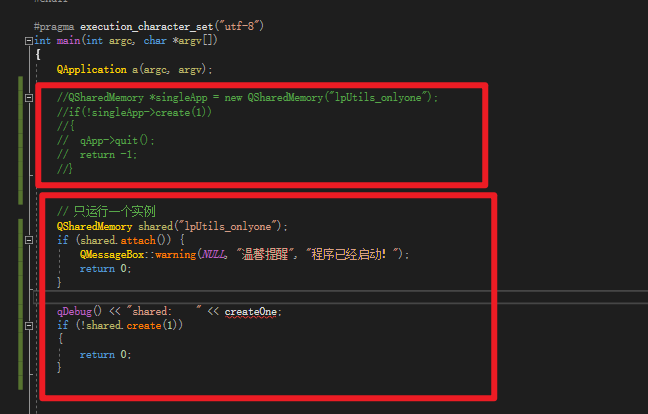
1、内存&虚拟内存方面的资料查询

2、多开问题修改后的代码的exe已发测试

3、工业通信协议？

5.21

1、修改多开问题



两种方式都可行

5.21

自动切换工单20%

5.22

自动切换工单

发现的问题：在开发过程中，工作被打断如何处理？

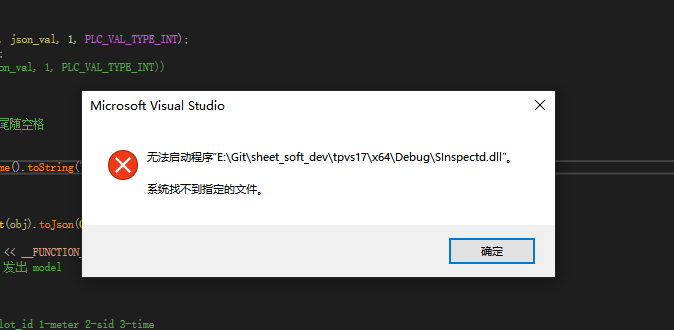
5.23

学习一下协议，解析数据跟下位机通信

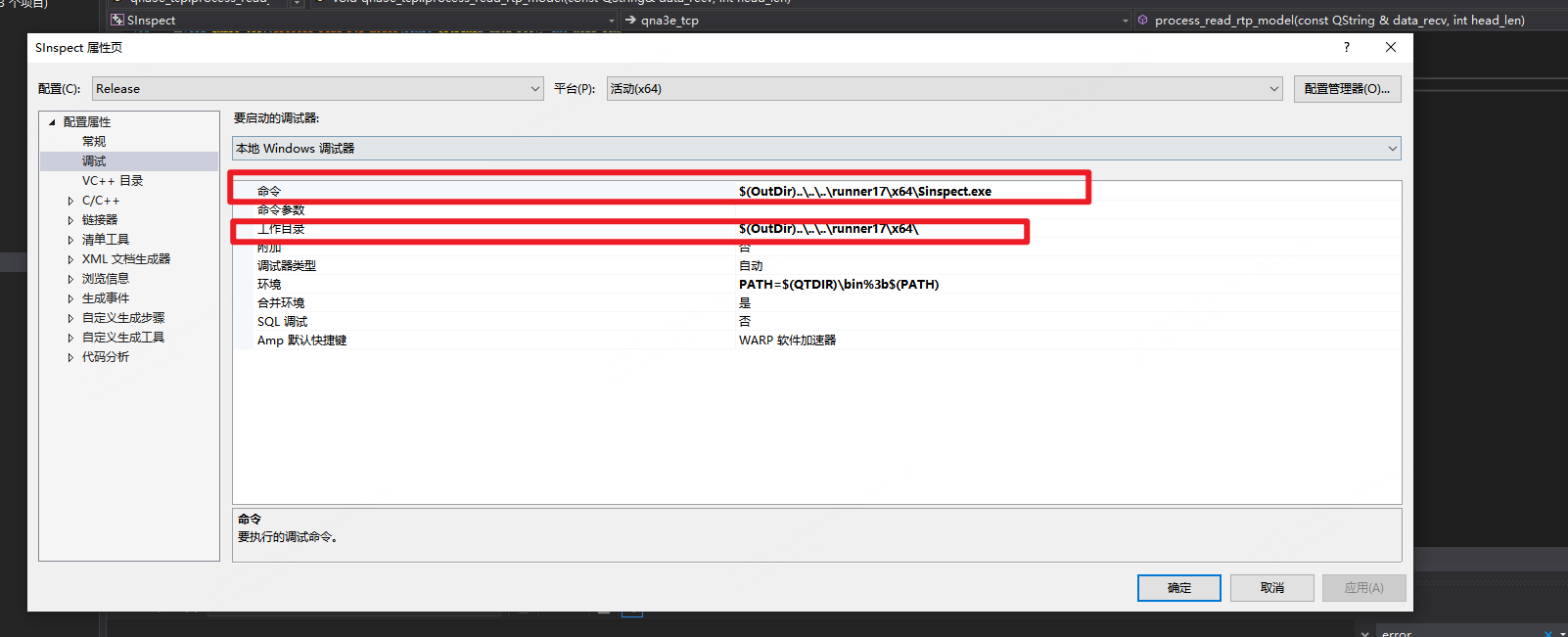


D\*固定的，这是地址，后面那段字符串是需要配置的地址。

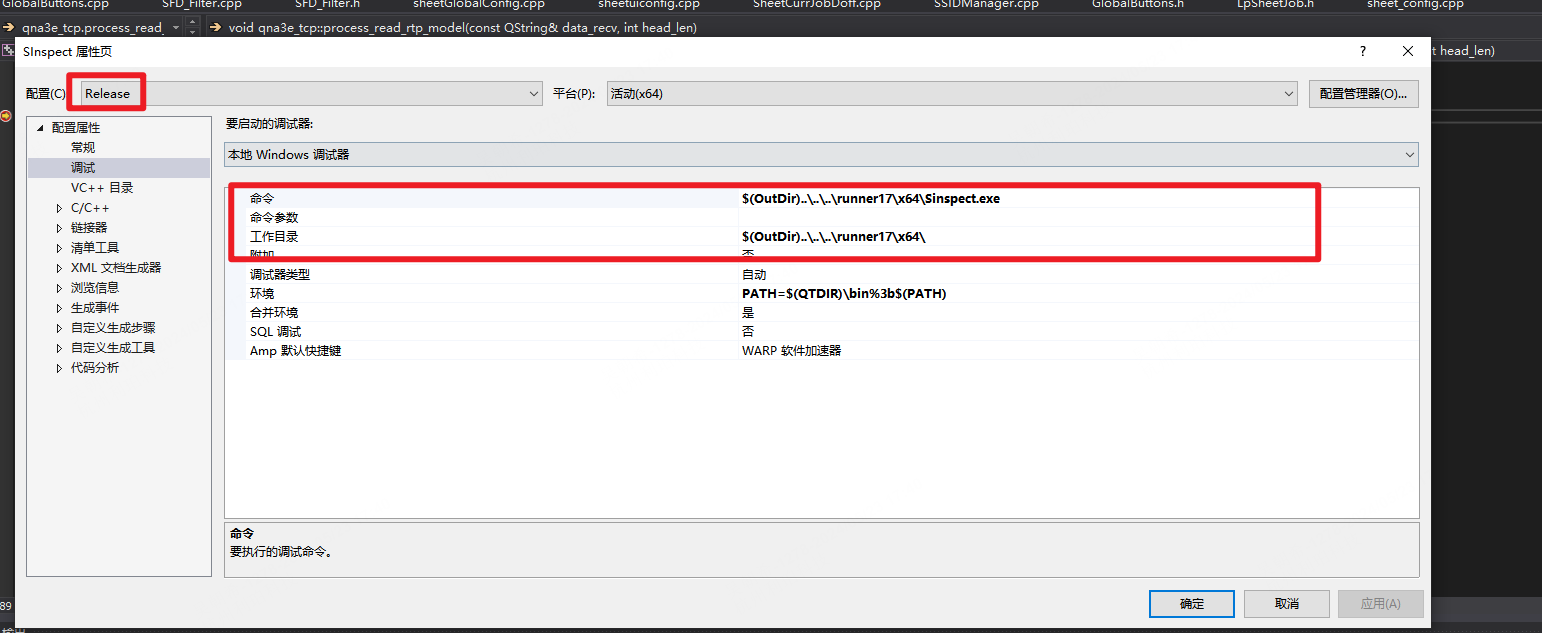
2、问题



解决方案



复制另一个版本的



5.25

1、恒美日志异常排查

2、

5.27

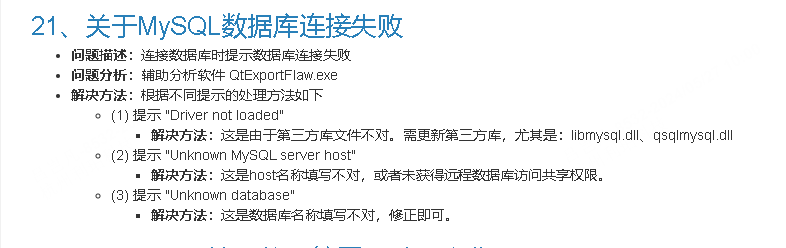
1、项目工程配置

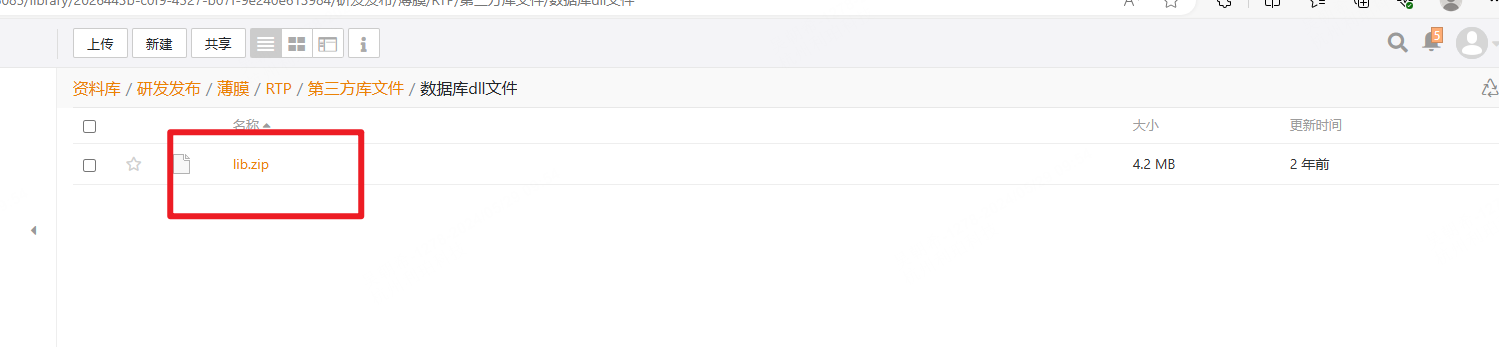
// 工程配置 $(SolutionDir)..\runner17$(Platform)\ // 输出目录 $(OutDir) // 调试 $(OutDir)$(TargetName)$(TargetExt) // 输出文件

$(GIT\_ROOT)\tb002\include // C/C++ 常规 $(ProjectDir)..\sdk\ // C/C++ 常规 , 链接器 常规 $(OutDir) // 链接器 常规

2、数据库驱动问题

根据日志提示



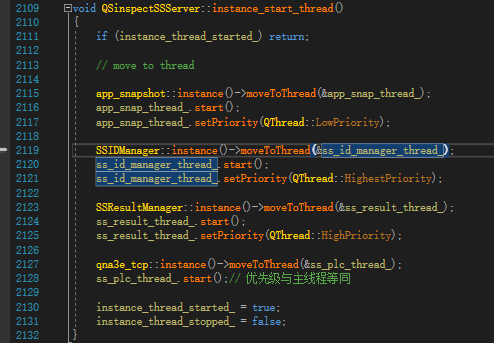
换第三方库问题得到了解决

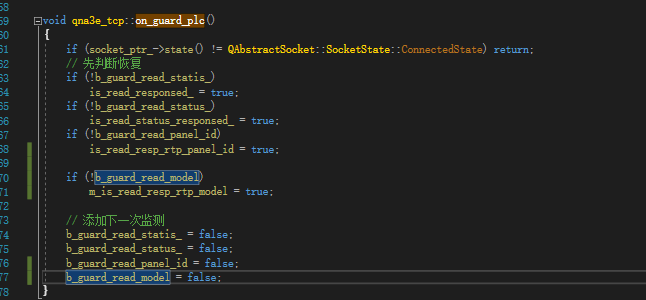
5.28

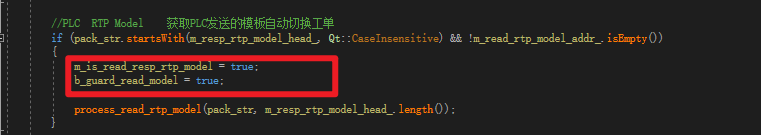
自动切换工单开发大致完成

5.29

1、线程，线程优先级代码示例

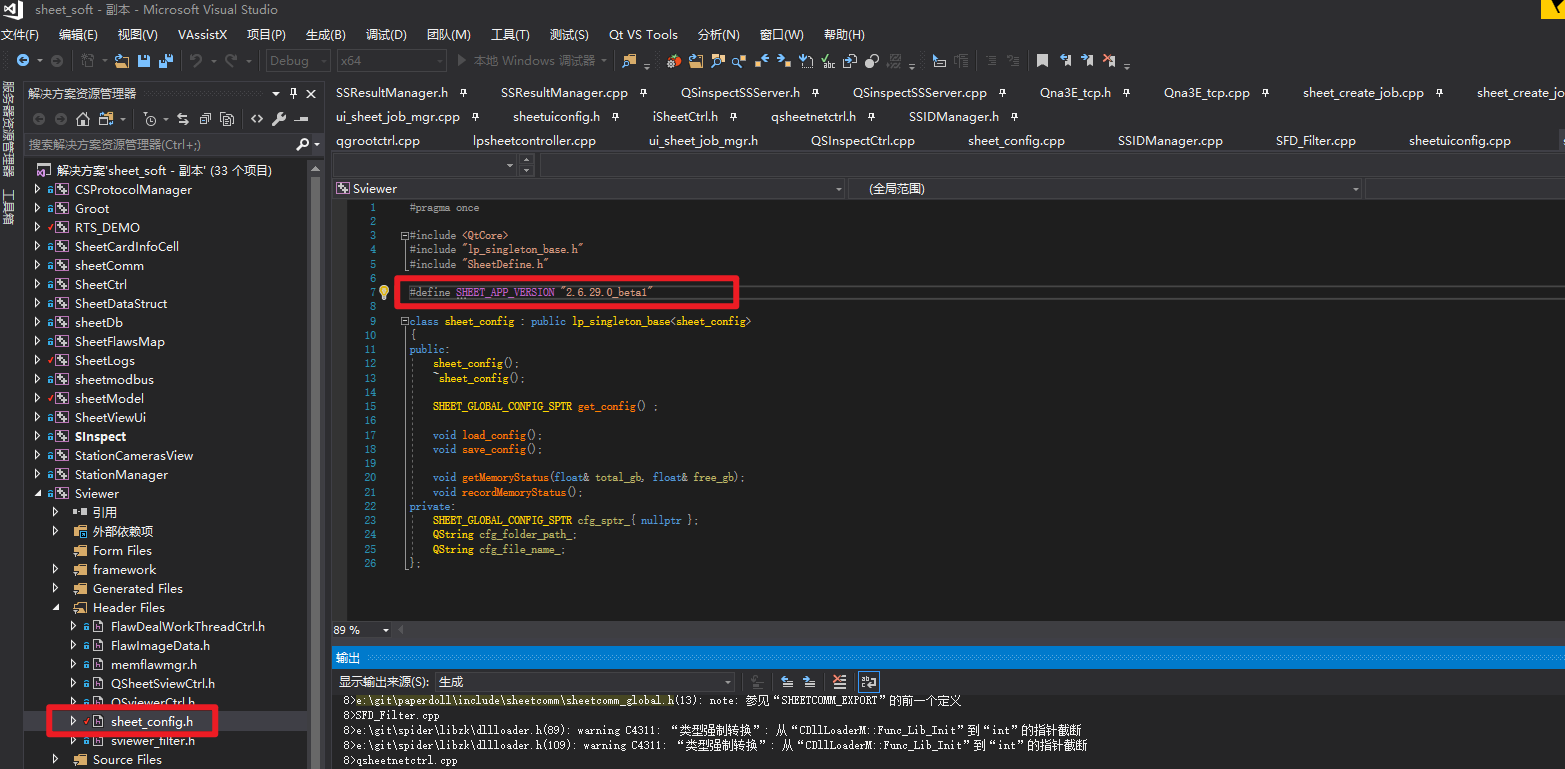


1



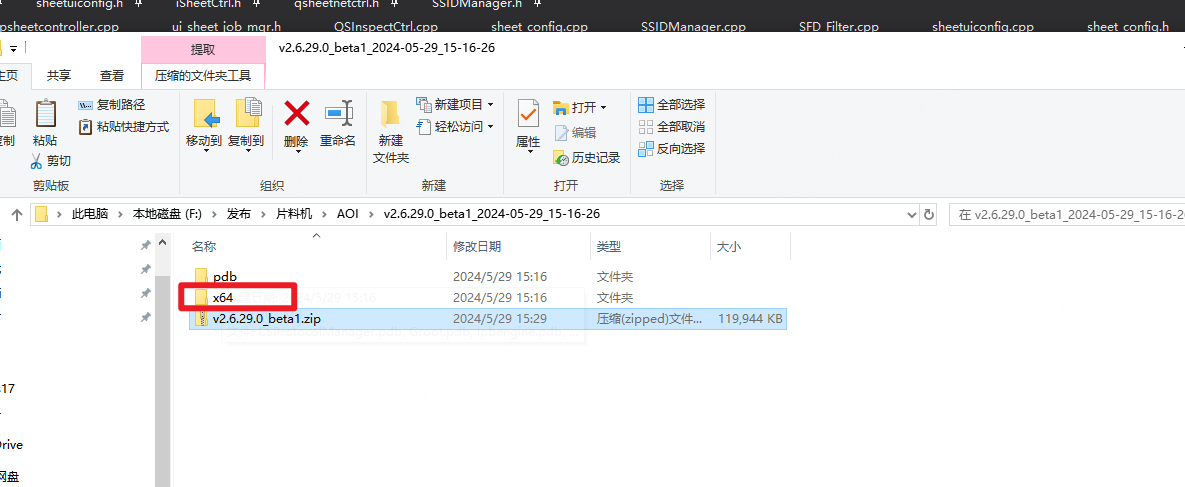
定时向plc发送请求，确保能够一应一答

打包版本号

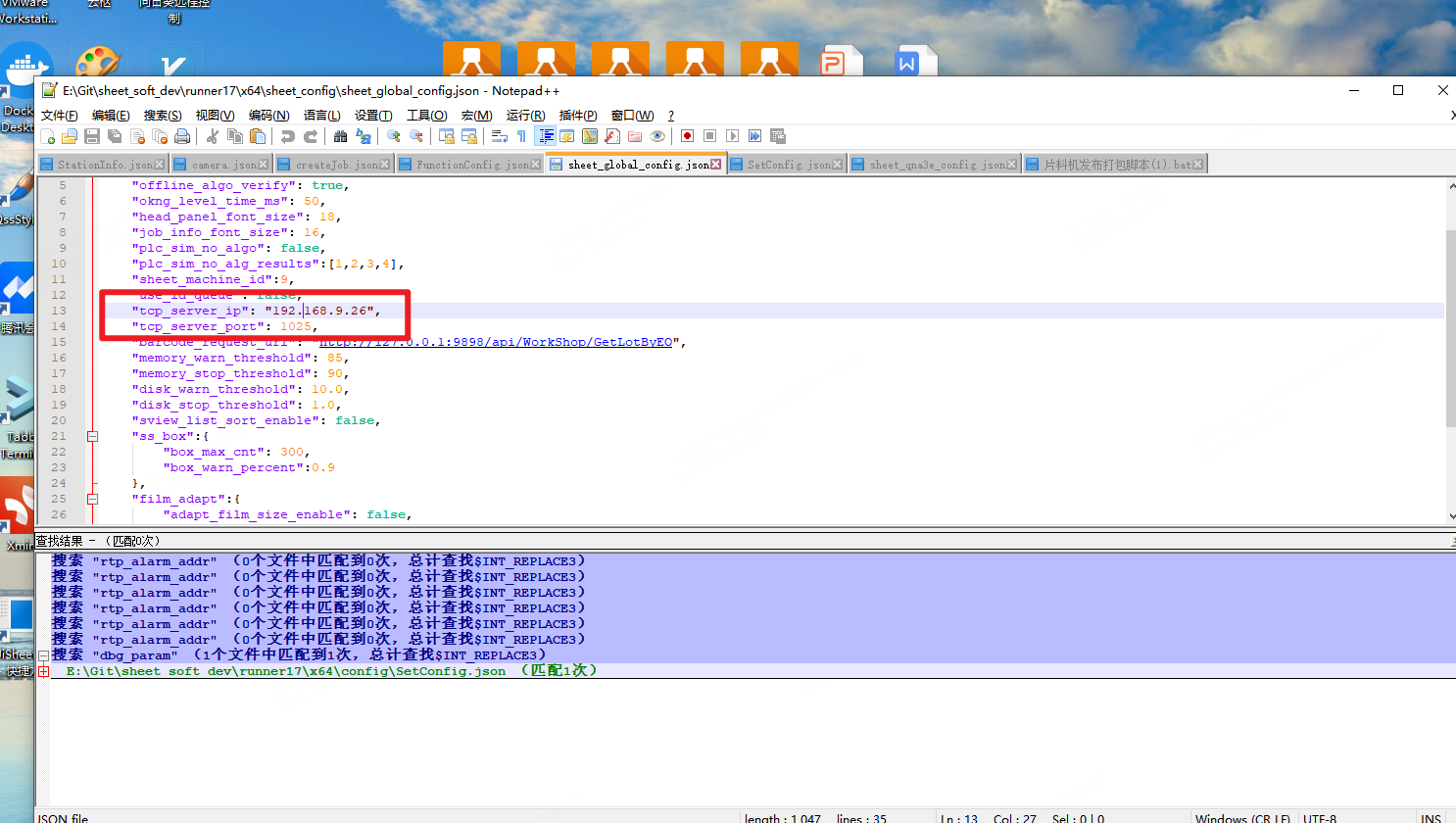


需要做一些修改

打包只要x64的就行



打包用Release版本,配置文件需要在服务端客户端配置，不会跟随本地打包的过去。

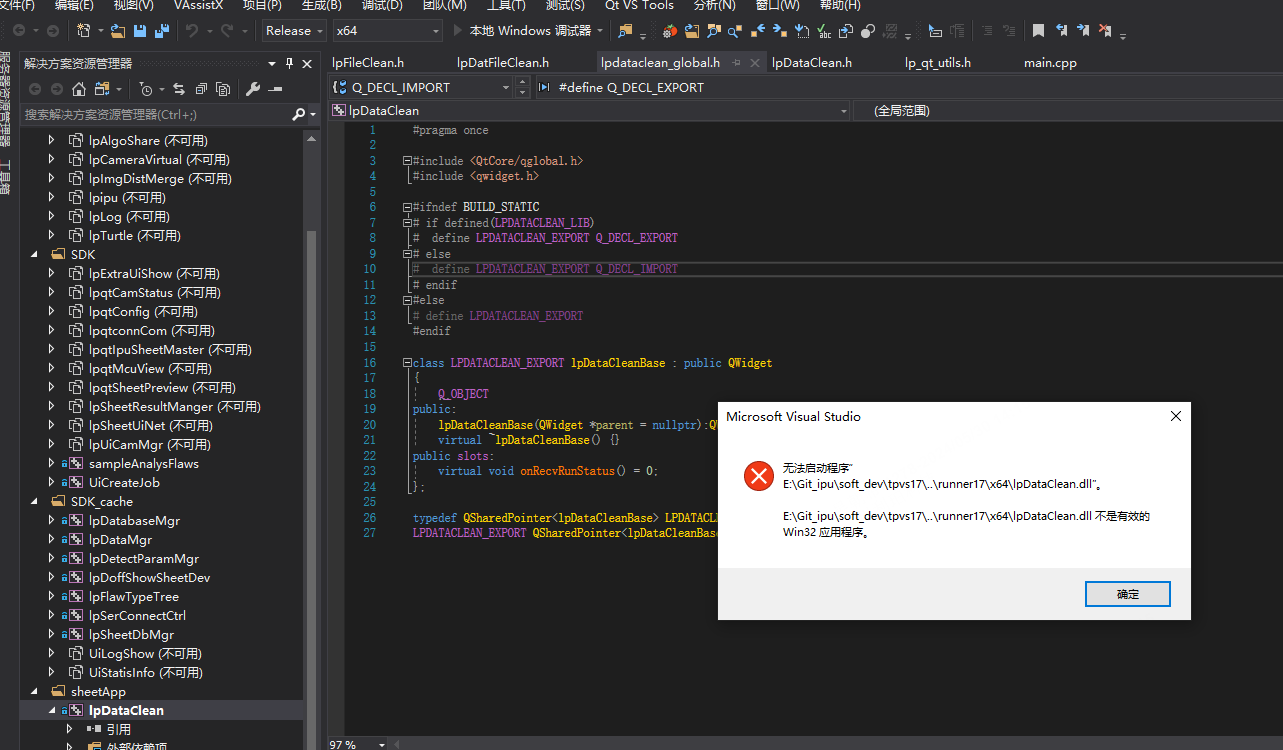


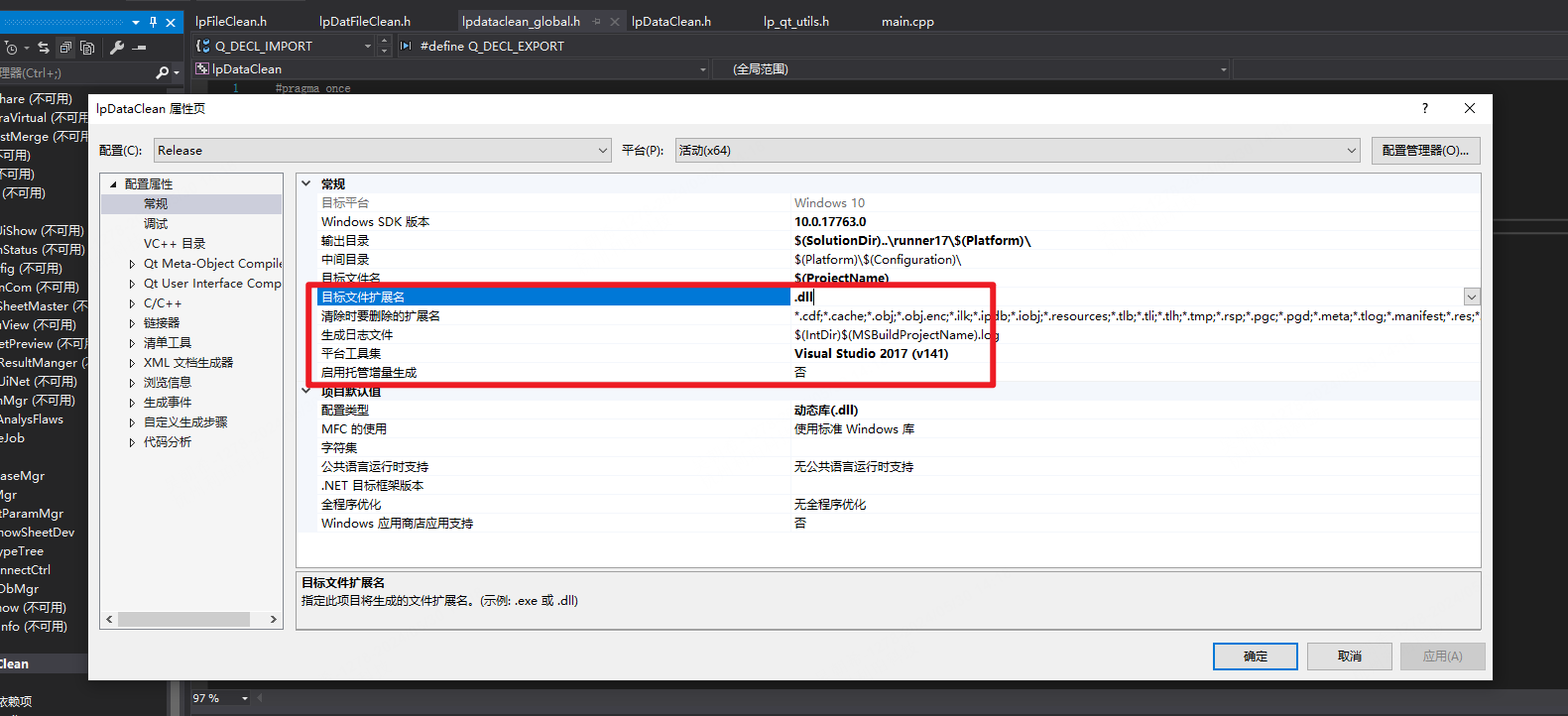
plc的配置文件

5.30

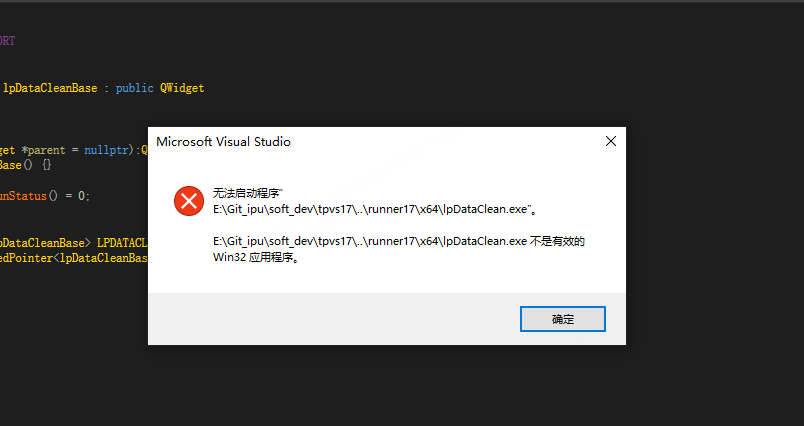
1、自动切工单自测后提测

2、数据清理模块完善，并用到重构软件中

3、



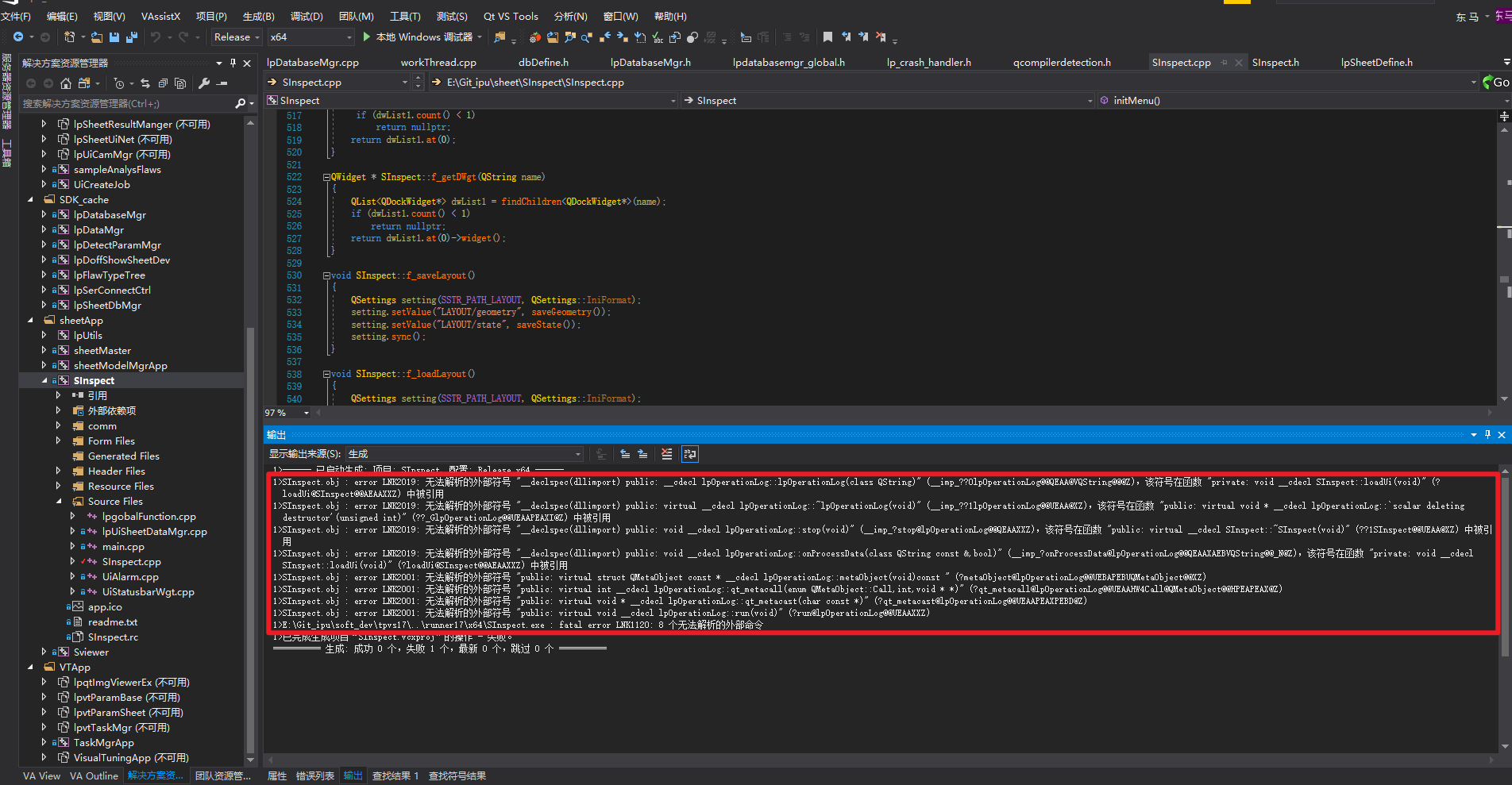
改成exe，



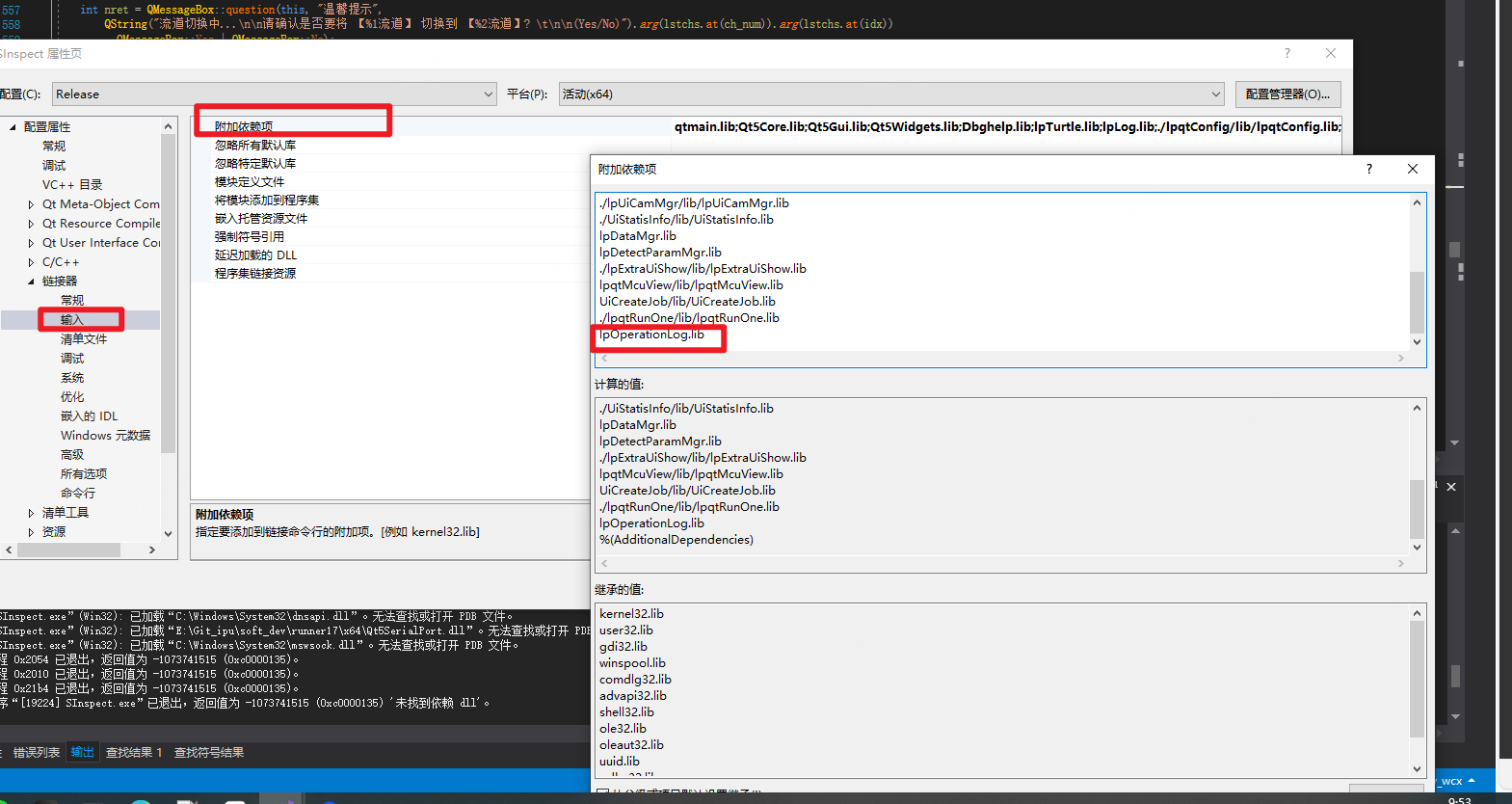
又出现新问题。

5.31

1、编译时遇到的无法解析的外部命令



添加依赖



2、自动切工单根据测试提出修改意见修改

3、

6.3

1、

6.4

1、片料重构功能配置界面模块 代码优化

2、QTreeView实时刷新界面

有的时候我们QTreeView的里面数据已经改变，但是界面却还是显示原来的数据，有的人会说用update函数，但是这个函数没啥用，

经过半天的摸索，发现了这三种方法。

2.1：doItemsLayout()，但是这个函数最好是在主线程中调用，因为这个函数里面是哟你定时器，在非主线程中调用会报警告。

2.2：使用模型/视图框架，继承QAbstractItemModel，然后就可以使用下面两个函数来刷新， beginResetModel(); endResetModel();

2.3：这个方法是最好的：QAbstractItemView::viewport()->update();

3、智能指针使用：

std::unique\_ptr<LpSheetJob> m\_pCurrentJob;

m\_pCurrentJob=std::make\_unique<LpSheetJob>();

如果不希望改变函数参数类型，函数内部也不需要智能指针的特性（比如引用计数），只是简单地使用对象，可以在调用函数时使用智能指针的.get()方法获取裸指针。