2021-01-05

AI人工智能Bonding异物检系统代码详细说明文档

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 说明 | 作者 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[引言 1](#_Toc53478279)

[1.1编写目的 1](#_Toc53478280)

[1.2背景 1](#_Toc53478281)

[1.3术语定义及说明 1](#_Toc53478282)

[2. 软件概述 2](#_Toc53478283)

[2.1需求概述 2](#_Toc53478284)

[2.2运行环境 2](#_Toc53478285)

[3. 系统结构设计 3](#_Toc53478286)

[3.1 软件介绍及准备工作 3](#_Toc53478287)

[3.1.1 库文件 3](#_Toc53478288)

[3.1.2 软件介绍 5](#_Toc53478289)

[4. 模块和功能需求的关系 11](#_Toc53478290)

[4.1 模块设计 11](#_Toc53478291)

[4.1.1算法模块 11](#_Toc53478292)

[4.1.2 Socket模块 12](#_Toc53478293)

[4.1.3自动检测线程模块 13](#_Toc53478294)

[4.1.4线程池模块 14](#_Toc53478295)

[4.1.5配方模块 15](#_Toc53478296)

[4.1.6 Ui模块 16](#_Toc53478297)

[5. 异常处理及维护 20](#_Toc53478298)

[5.1 日志相关 20](#_Toc53478299)

[5.2 安装CrashRpt及异常捕捉 20](#_Toc53478300)

# 引言

## 1.1编写目的

本文档给出了AI人工智能Bonding异物检系统软件代码的详细说明，重点介绍了软件的整体结构设计，为设计人员、编程人员熟悉代码提供一份说明文档。

## 1.2 背景

## 1.3术语定义及说明

|  |  |  |
| --- | --- | --- |
| 序号 | 术语或缩略语 | 说明性定义 |
|  |  |  |
|  |  |  |
|  |  |  |

# 软件概述

## 2.1需求概述

需求分析：

按照面向对象分析，需要编写的模块有：算法模块、ui模块、日志及信息记录模块、配方模块、ai模块。

利用上述模块来满足各种需求，自动检测的需求几乎涵盖了所有功能需求，所以设计好自动检测功能，项目基本完成。

## 2.2 运行环境

操作系统

Windows7以上

网络配置

1000M网卡

开发工具

Visual studio 2015

相关软件工具

GX Works3，网络调试助手

# 系统结构设计

## 3.1 软件介绍及准备工作

### 3.1.1 库文件

第三方库文件均在公司通用库build\_lib中。打开项目工程，包含的所有头文件如下：

wxWidgets-3.1.0-64为界面ui设计库的头文件；

Json和curl为ai通讯库的头文件；

opencv440\_2015是opencv库的头文件；

CrashRpt是捕捉异常库的头文件；

MIL是粒子算法所需头文件（已弃用）;

boost166是为C++提供拓展功能的头文件;

SuperDog是加密狗的头文件;

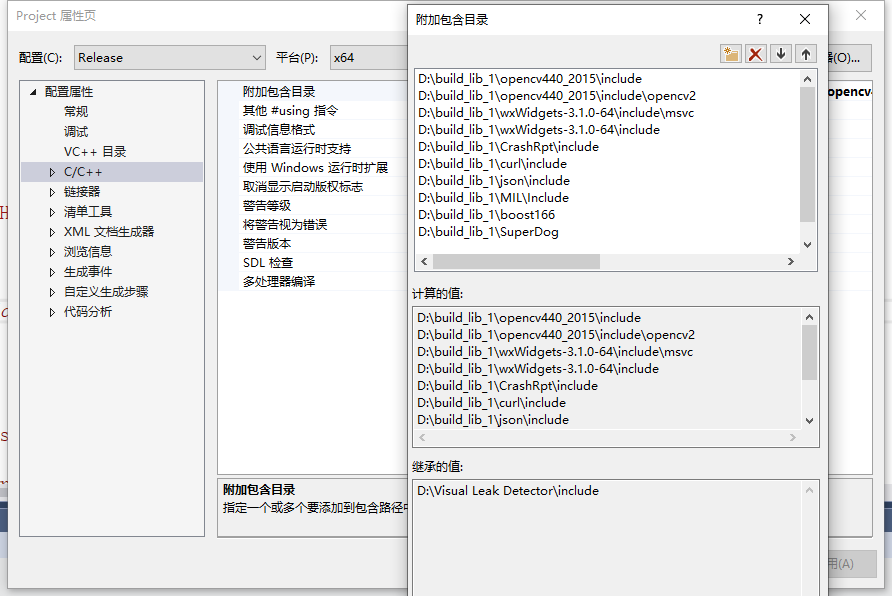


图3-1-1-1

加入头文件后，再加入的是对应相应的静态库静态库的需要加入库路径和附加依赖项，如图3-1-1-3和图3-1-1-4。

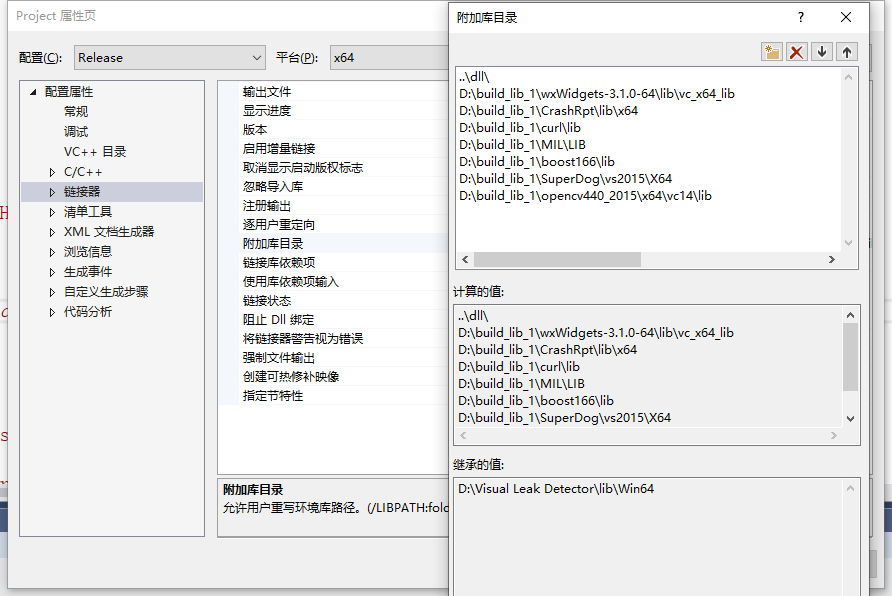


图3-1-1-3

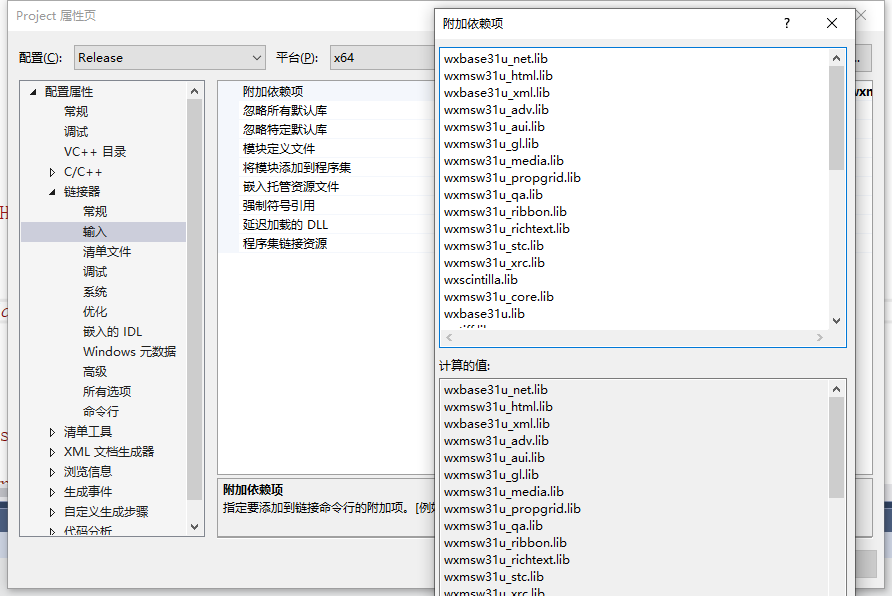


图3-1-1-4

### 3.1.2 软件介绍

打开软件，首先出现的是在线模式或者脱机模式的选择框；在线模式需要连接PLC后，在接收到信号后，才可进行自动检测。脱机模式则以轮循的方式进行自动检测。

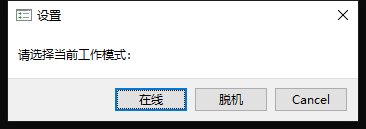


图3-1-2-1

如图3-1-2-2，显示的是主界面，从左往右，由上至下分为四个部分：检测信息窗口、数据统计窗口、操作页面，显示界面窗口。

检测信息窗口：显示每张图片检测后的信息，包括图片名，检测结果，检测结果数量，检测后图片所在路径及检测后信号发送的时间。

数据统计窗口：统计每张片子的检测结果，分为OK数，NG数和Ratio率。

操作页面：主要分为连接PLC，连接AI，检测，自动检测。

显示图片窗口：将检测图片按照长宽同等比例显示在此窗口。



图3-1-2-2

载入图像：点击菜单栏中的“载入图像”，可选择从外部载入一张图片至显示图片窗口中显示，主要用于制作配方使用。

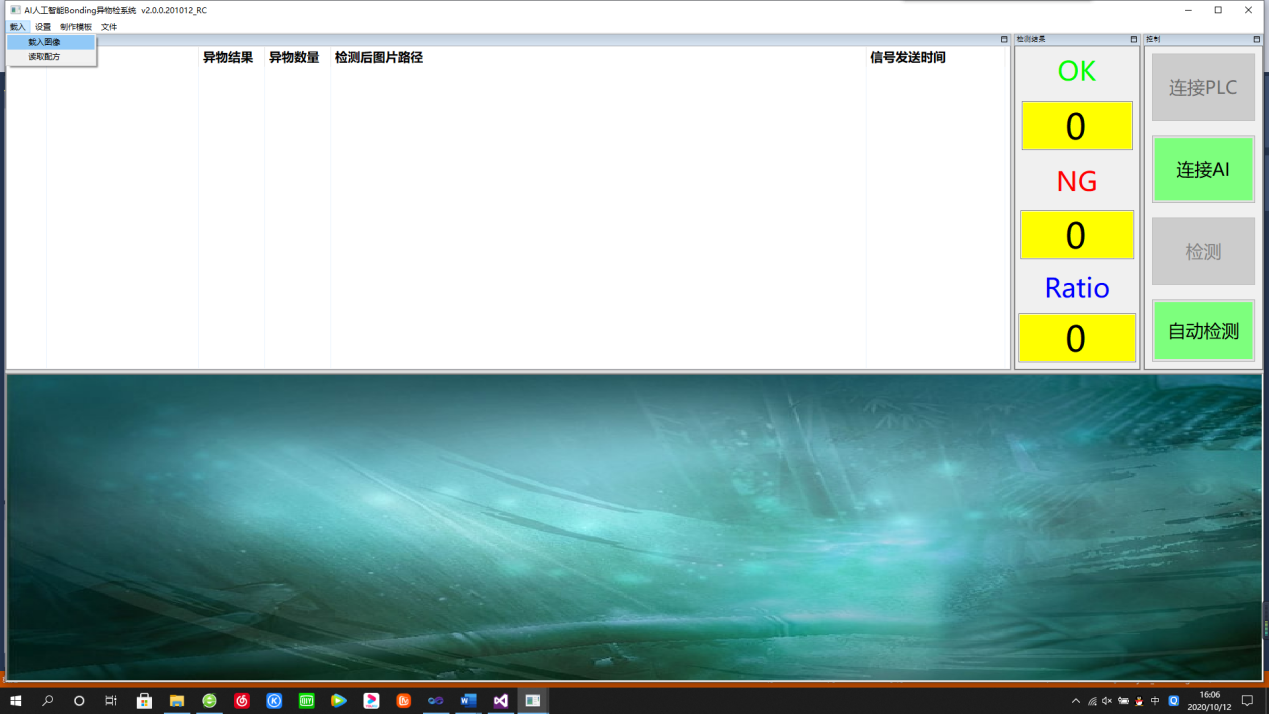


图3-1-2-3

读取配方：点击菜单栏中的“读取配方”，可选择读取之前的配方进行检测，方便使用。

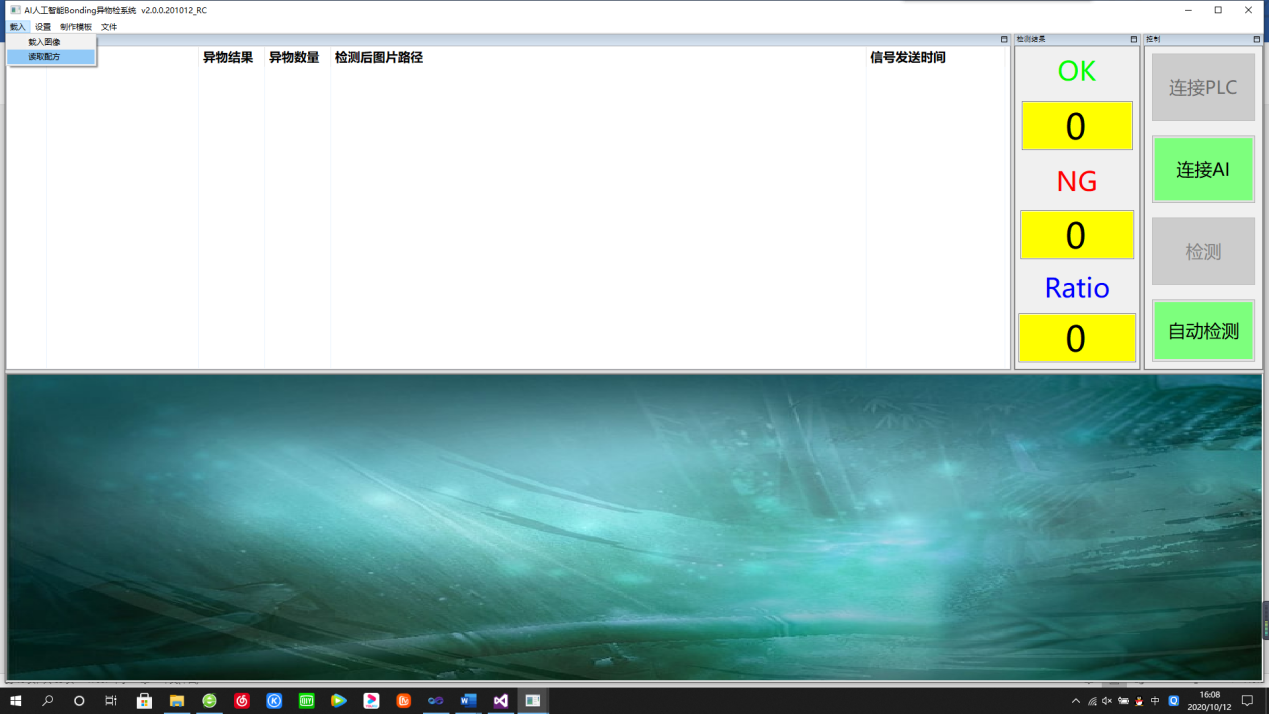


图3-1-2-4

自动检测设置：点击菜单栏中的“自动检测设置”，可界面中可对PLC的IP，端口，图片读取路径，csv文件路径，检测后的图片保存路径（算法及AI检测后的图片，大小为224\*224），Execl报表保存路径，所在载台信号等进行设置。

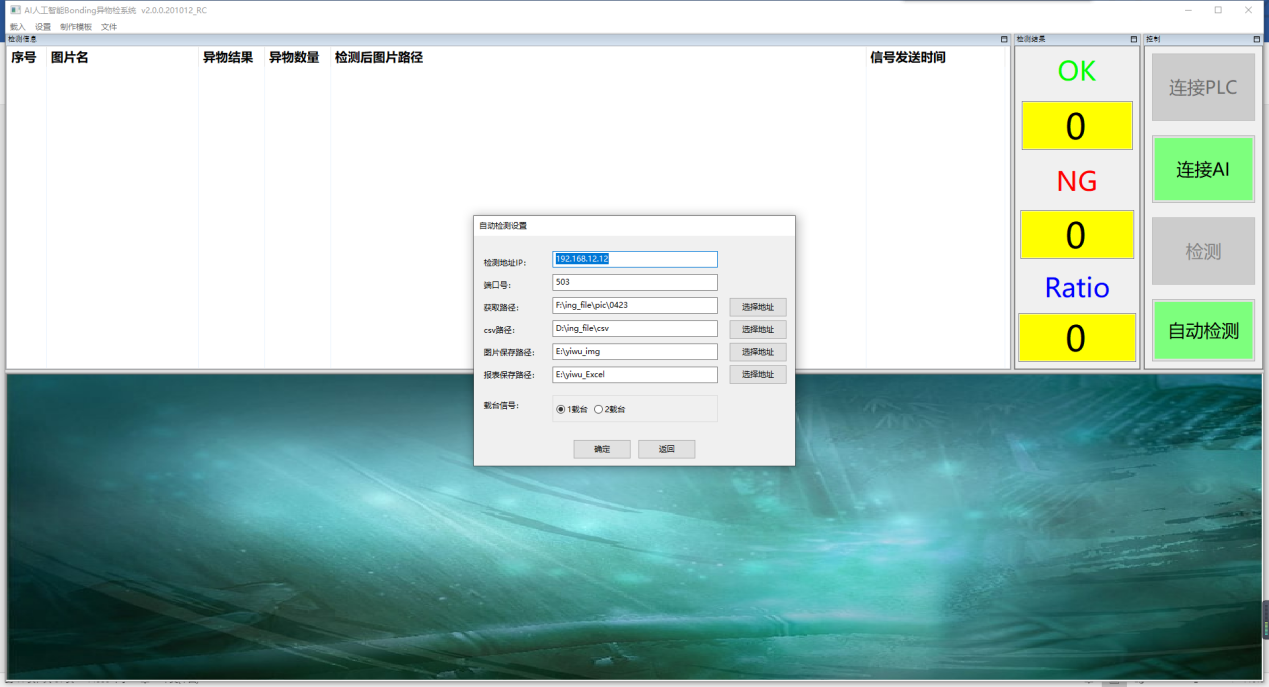


图3-1-2-5

设置算法参数：点击菜单栏中的“设置算法参数”，可对算法参数进行设置。

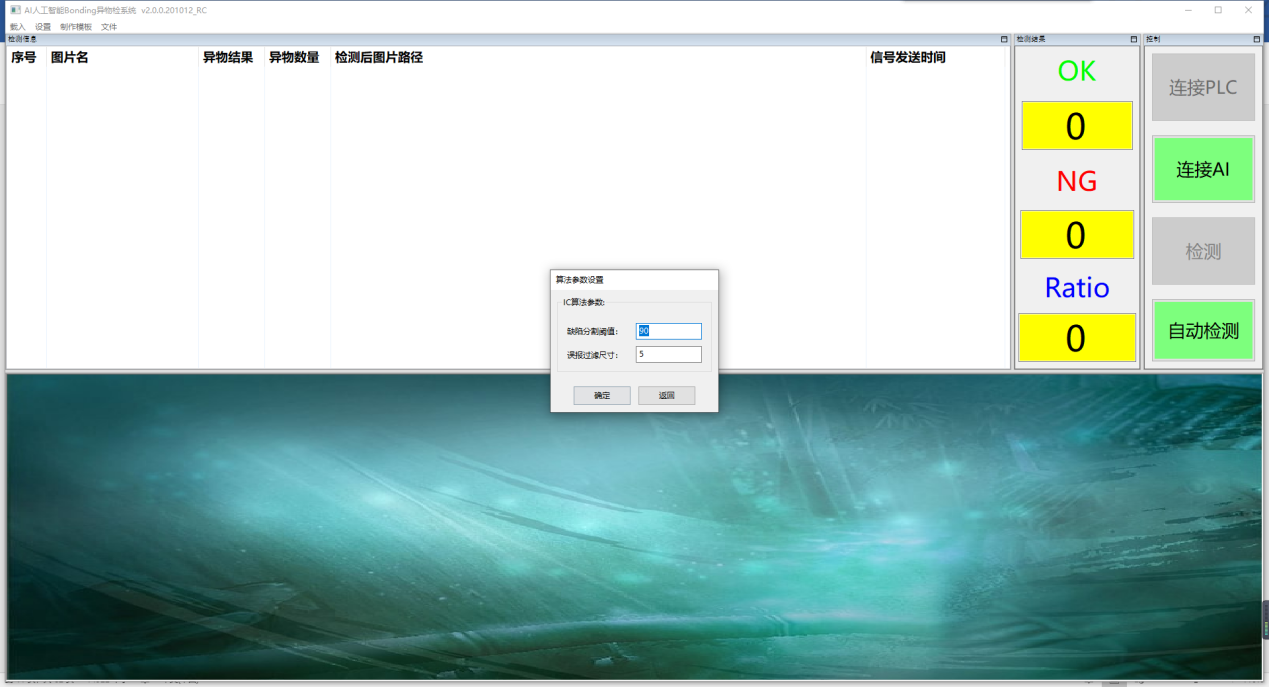


图3-1-2-6

设置读取/写入地址：点击菜单栏中的“设置读取/写入地址”，可对PLC读取/写入地址进行设置，OK信号写1，NG信号写2。



图3-1-2-7

制作模板：点击菜单栏中的“模板”或者“正方形模板”，在载入图片的基础中，可对图片进行画框操作，选择合适的地方画框，松开鼠标后将出现确认框，点击确认后，将用蓝色方框对该区域进行标记，同时该图片的模板类型将被记录在模板界面中。

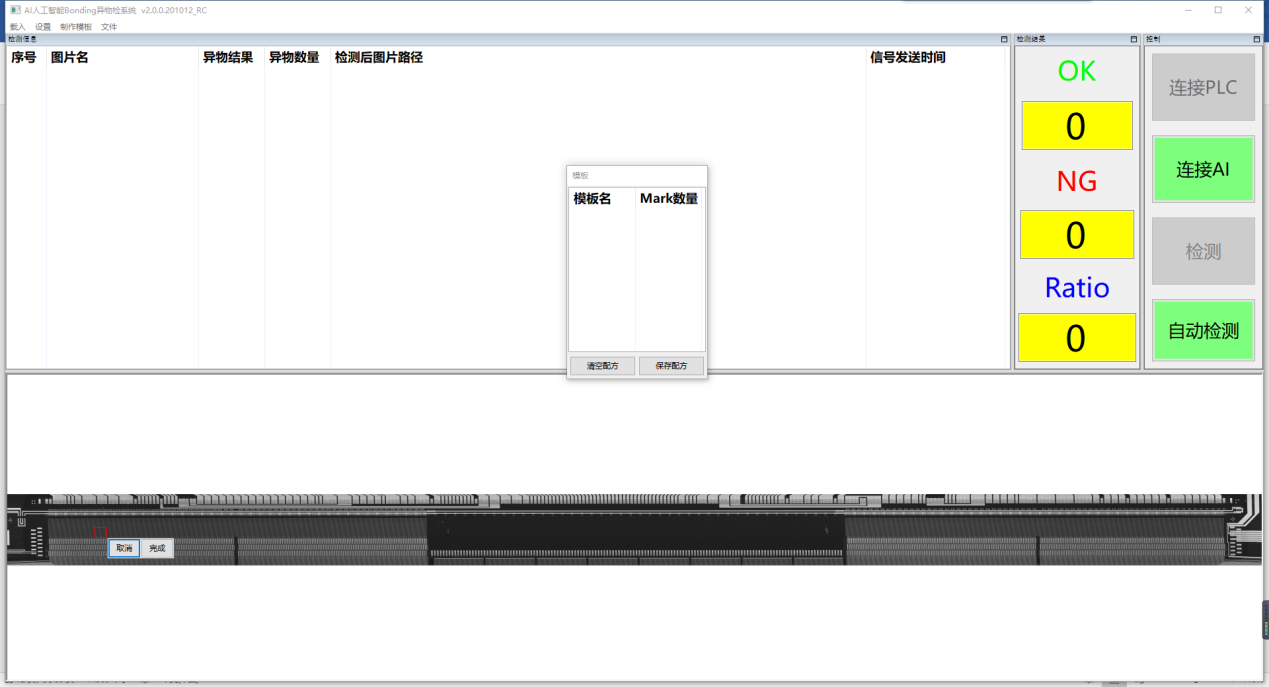


图3-1-2-8

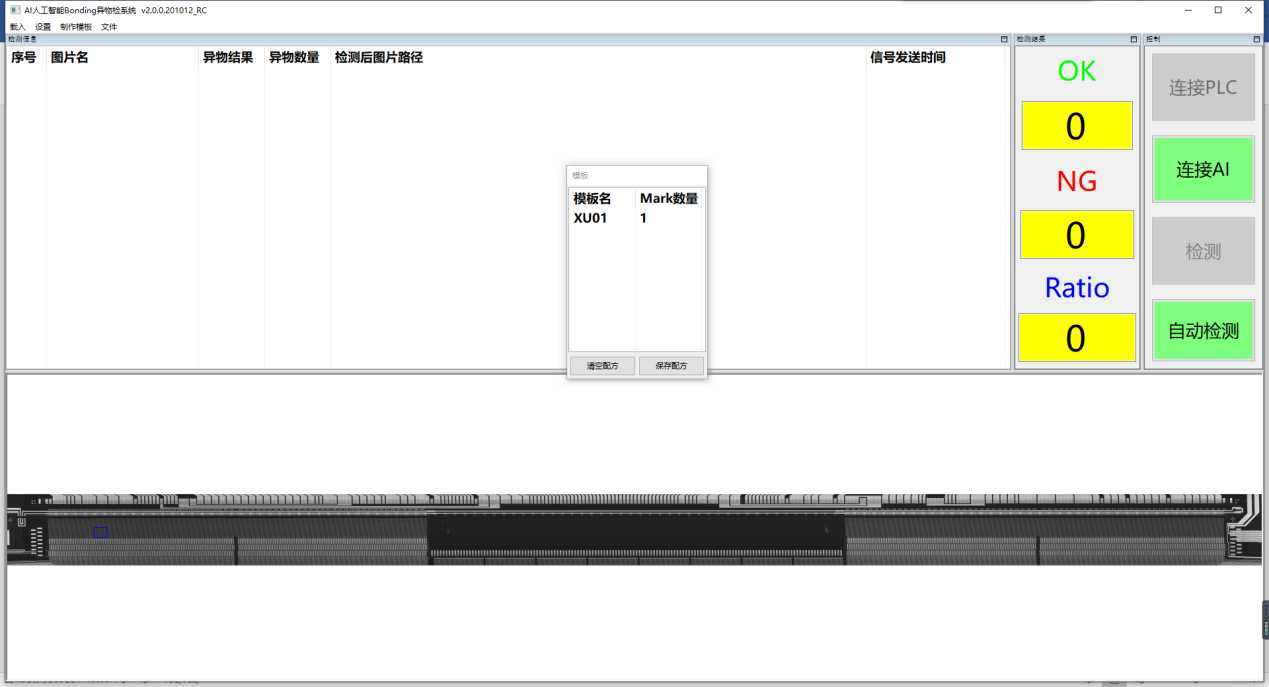


图3-1-2-9

打开算法检测结果目录：点击菜单栏中的“打开算法检测结果目录”，将使用window自带工具打开算法检测结果目录，方便人员查看。

打开AI检测结果目录：点击菜单栏中的“打开AI检测结果目录”，将使用window自带工具打开AI检测结果目录，方便人员查看。

打开报表目录：点击菜单栏中的“打开报表目录”，将使用window自带工具打开报表目录，方便人员查看。

打开当前时间段报表：点击菜单栏中的“打开当前时间段报表”，将使用Excel打开当前时间段的检测报表，方便人员查看。

打开汇总报表：点击菜单栏中的“打开汇总报表”，将使用Excel打开当前汇总报表，方便人员查看。

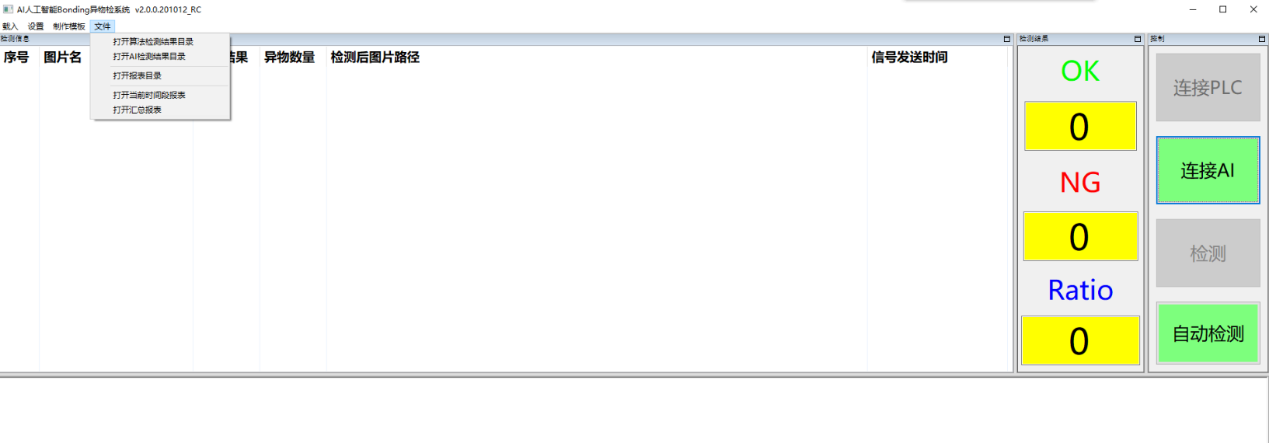


图3-1-2-10

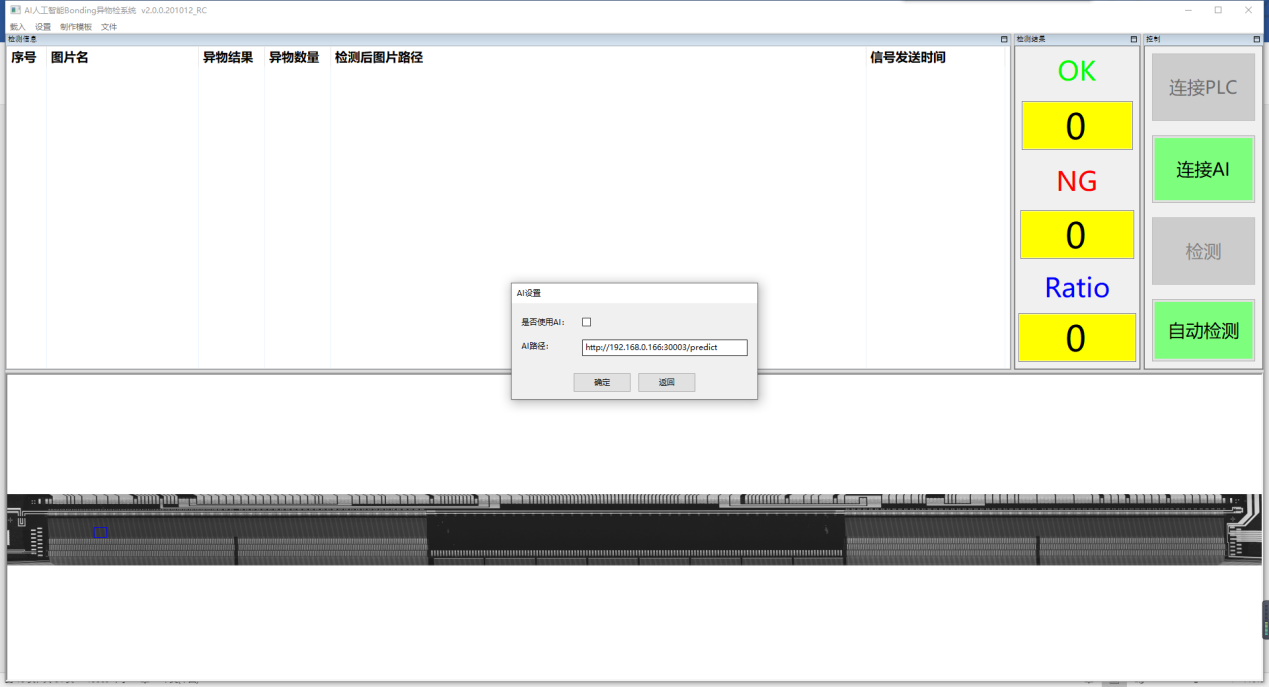
AI路径设置：点击操作页面中的“连接AI”，将打开AI设置界面，界面包括是否使用AI及AI路径设置。

图3-1-2-11

设置中的清空历史纪录目的是为了初始化检测信息窗口与检测结果窗口。

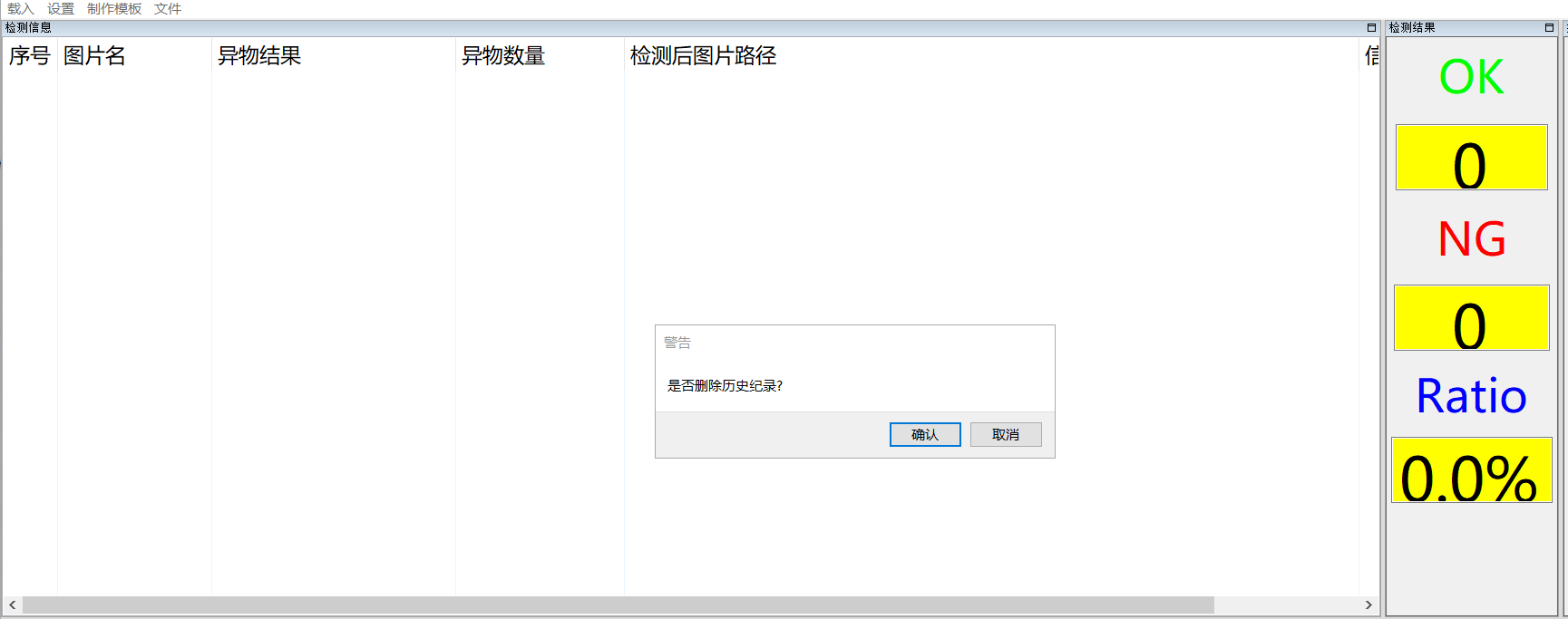


图3-1-2-12

制作模板中的不检测区域目的是为了过滤掉不检测区域的误报。



图3-1-2-13

# 模块和功能需求的关系

本项目软件主要功能模块有算法模块、socket模块、自动检测线程模块、线程池模块、配方模块、UI模块。

## 4.1 模块设计

下面将对各个模块的设计方式以及源代码进行介绍。

### 4.1.1算法模块

检测算法代码由算法工程师封装为动态库，无法查看源代码，下面将对其函数功能进行描述。

1. 检测算法

打开ForeignMaterialDetector.h，如图4-1-1-1.



图4-1-1-1

cv::Mat mark\_img：mark标记位置；

float mark\_x：mark左上角x坐标；

float mark\_y：mark左上角y坐标；

const cv::Mat src\_img：待检测图像；

cv::Mat template\_img：模板图像；

int binary\_value：缺陷分割阈值；

int median\_blur\_size：中值滤波大小（必须为奇数）；

std::vector<cv::Rect> &result\_rects: 缺陷位置；

### 4.1.2 Socket模块

Socket模块主要包括SocketCenter.cpp和SocketCenter.h文件，如图3-1-2-1。



图4-1-2-1

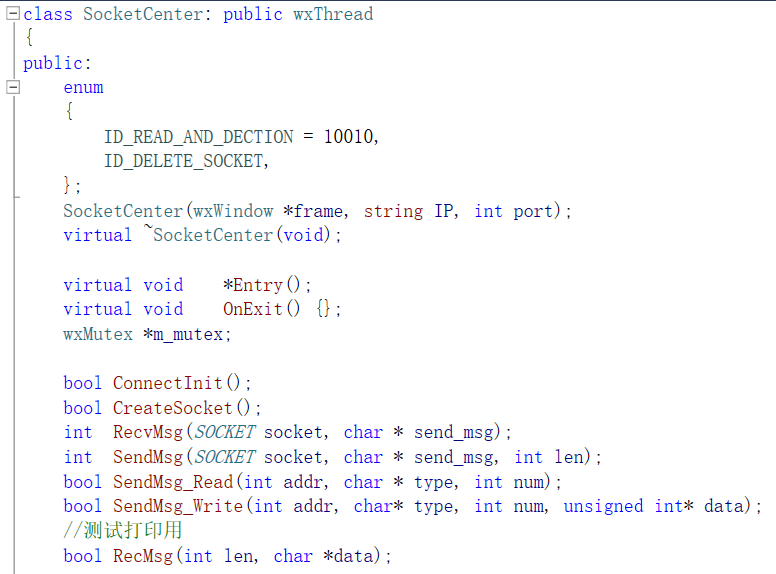
类设计概念为：实现连接、断开、发送信息，接收信息四个基本功能，然后在基础上再进行细分或者增加其他功能。

图4-1-2-2

图4-1-2-2为SocketCenter类的主要函数，函数功能如下：

bool a7500mg20\_config(map<string, void \*> \*pdc)：获取dc.dat的二次定位相机参数；

bool ConnectInit()：初始化连接。

bool CreateSocket()：创建Socket，并尝试Socket是否可以连接，成功连接后会自动断开，等待下次用户手动开启连接；

int RecvMsg(SOCKET socket, char \* send\_msg)：接收服务器发回来的信息；

int SendMsg(SOCKET socket, char \* send\_msg, int len)：发送所需要的信息；

bool modifyCameraExposureTime(double dbValue)：修改相机曝光时间；

bool SendMsg\_Read(int addr, char \* type, int num)：对PLC进行读取操作的封装函数；

bool SendMsg\_Write(int addr, char\* type, int num, unsigned int\* data)：对PLC进行写入操作的封装函数；

bool RecMsg(int len, char \*data)：用于打印发送或者接收后的信息，一般只在测试时用以确认发送接收信息是否正确时使用。

SocketCenter类的设计概念：当在线模式启动后，需要接收PLC信号后才可对图片进行抓取，因此该类需要存在于一个独立线程中，始终监控PLC信号是否产生变化，若产生变化则启动抓图功能，并在检测完成后向PLC发送检测结果信号。

### 4.1.3自动检测线程模块



图4-1-3-1

图4-1-3-1为自动检测线程类的主要函数，函数功能如下：

virtual void \*Entry()：线程的入口。

该线程在在线模式下的流程为：当接收到载台信号后，在抓图路径中找出生成时间距离当前时间最近的一个片子，将其中的各张图片送入检测函数中进行检测，检测后重新等待下一次信号的到来。

该线程在脱机模式下的流程为：在抓图路径中轮循取出各个片子，将其按照一定顺序排序，对其中的每张图片进行检测。

### 4.1.4线程池模块



图4-1-4-1



图4-1-4-2

图4-1-4-1和图4-1-4-2为线程池类的主要函数。

该项目中为了减少检测时间，需要同时对多张片子进行检测，因此引入线程池。线程池能够减少创建的线程个数，减少线程本身带来的开销。线程池采用预创建的技术，在应用程序启动之后，将立即创建一定数量的线程，放入空闲队列中。这些线程都是处于阻塞状态，不消耗CPU，但占用较小的内存空间。当任务到来后，缓冲池选择一个空闲线程，把任务传入此线程中运行。当N个线程都在处理任务后，缓冲池自动创建一定数量的新线程，用于处理更多的任务。在任务执行完毕后线程也不退出，而是继续保持在池中等待下一次的任务。当系统比较空闲时，大部分线程都一直处于暂停状态，线程池自动销毁一部分线程，回收系统资源。基于这种预创建技术，线程池将线程创建和销毁本身所带来的开销分摊到了各个具体的任务上，执行次数越多，每个任务所分担到的线程本身开销则越小。

### 4.1.5配方模块



图4-1-5-1

图4-1-5-1为配方模块，配方模块是本项目主要的模块之一，在制作配方模块时，考虑的顺序是如何建立配方的数据结构、怎样读写配方的数据、配方的数据是否保密。

建立配方的数据结构：综合对项目的研究，将配方属性归结为两个，配方名和器件类链表，其中器件类的属性为图片类型名称，图片mark点位置数据，图片mark点位置及图片原图数据。将数据结构确认后，就可以在相对应的位置存入相对应的数据。

怎样读写配方的数据及配方的数据是否保密：为了对配方进行读写，同时需要保证配方的数据不会在外部泄漏，在此引入Boost库。Boost库是为C++语言标准库提供扩展的一些C++程序库的总称，由Boost社区组织开发、维护。Boost库可以与C++标准库完美共同工作，并且为其提供扩展功能。引入Boost库的序列化和反序列化功能，该功能可以对数据写入进行序列化，数据序列化后会成为一段乱码，并被保存至某个文件中。在读取时，通过反序列化，将乱码转换成相对应的数据，并用相对应的变量进行接收，就能完成对数据的读取。

### 4.1.6 Ui模块



图4-1-6-1

界面模块则是各个窗口的绘制函数以及按钮功能的实现。

其中，ListPanel、ControlPanel、Refreshwindow及ResultDlgPanel组成了主界面（如图4-1-6-2），ListPanel为主界面的列表界面（如图4-1-6-3），ControlPanel为按钮界面（如图4-1-6-4），Refreshwindow为图片显示界面（如图4-1-6-5），ResultDlgPanel为结果统计界面（如图4-1-6-6）。

Ui是使用wxWidget库进行设计的，wxWidget的文档介绍的相当详细，根据文档学习此模块能够快速掌握。

说明的一点是wxWidget的入口函数为wxApp::OnInit函数，在该项目中LNIApp.cpp和LNIApp.h中的LNIApp类则是继承了wxApp，因此LNIApp类为本程序的入口函数。

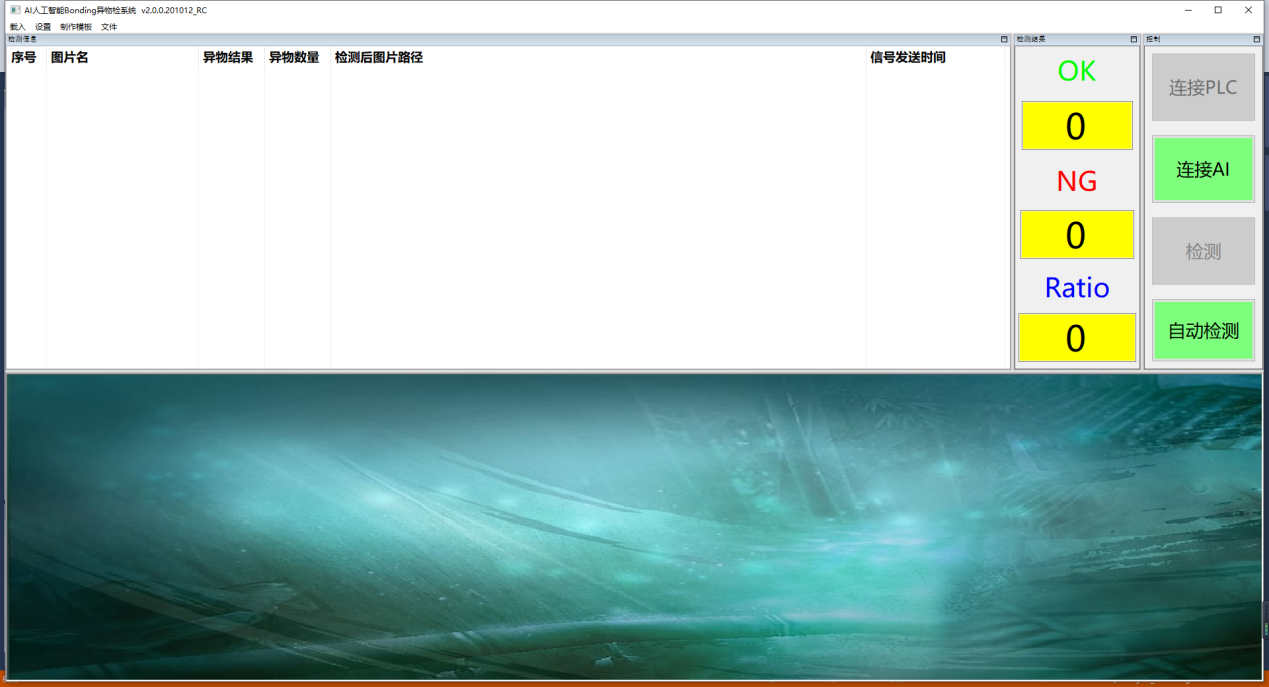


图4-1-6-2（主界面）

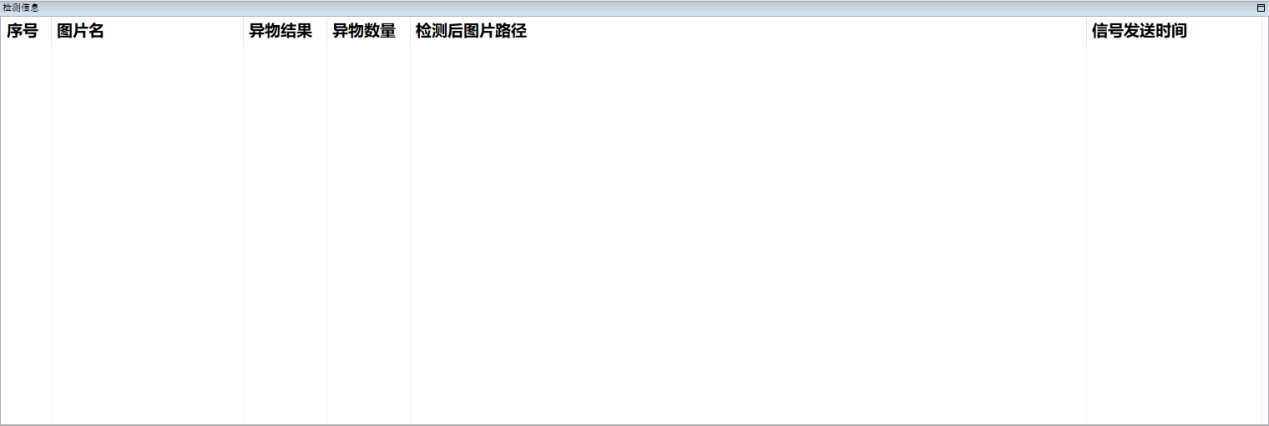


图4-1-6-3（列表界面）



图4-1-6-4（按钮界面）



图4-1-6-5（图片显示界面）



图4-1-6-6（结果统计界面）

# 异常处理及维护

## 5.1 日志相关

对于日志模块，当程序出现异常，比如闪退、卡死现象时，就需要查看相关日志查找问题（对于闪退，后面将介绍一个有用的插件crashrpt，可以帮助我们捕捉到异常，并定位到出错的代码），因此日志模块是相当重要的，在合适的地方打印合适的log，将会事半功倍。

如下图，在程序入口函数LNIApp::OnInit()中，使用MyLog::SetTargeFile()函数可以设置要进行日志显示的文件及文件打开方式，使用MyLog::SetTimeStamp()可以设置日志时间，使用MyLog::SetLogLevel()可以设置日志等级（状态），在自动检测过程中，打印的log越多越全，对我们查找bug的帮助就越大（注意额外一点是不要在循环里打印log，不仅耗时，而且查看log的难度加大）。

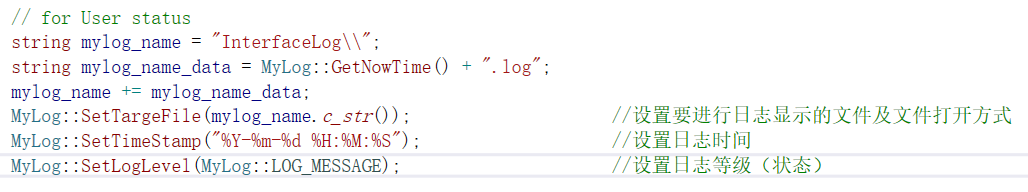


图5-1-1

## 5.2 安装CrashRpt及异常捕捉

安装此插件能够实现定位源代码的位置，下面说明一下使用步骤。

第一步，将下图的三个文件放入软件目录下。

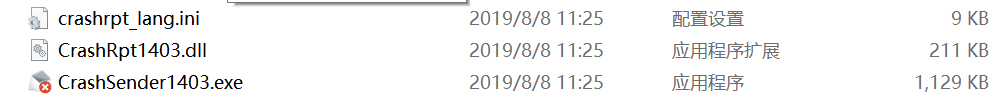


图5-1-2

第二步，在LNIApp中加入crashRpt的头文件和库文件后，在LNIApp::OnInit()加入如下图的#ifdef \_CRASH\_RPT\_ #endif的内容。

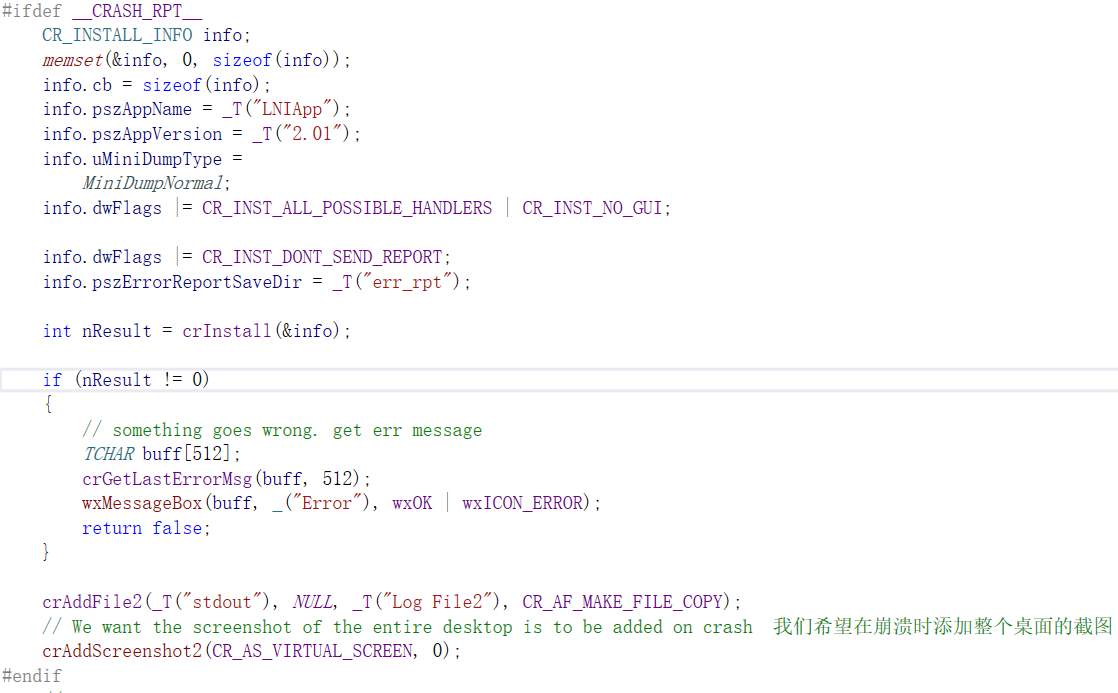


图5-1-3

第三步，编译完成项目后，找到x64下Release的Project.exe和Project.pdb，复制一份保存（注意：此文件只针对当前编译版本有效，如果改动代码再次编译，需要再另存Project.exe和Project.pdb）。

第四步，假设此时软件闪退，找到软件目录下的err\_rpt文件夹，找到最新时间额文件夹，打开crashdump.dmp，然后选择之前保存的Project.exe和Project.pdb，点击调试，就可以看到源代码的错误位置了。

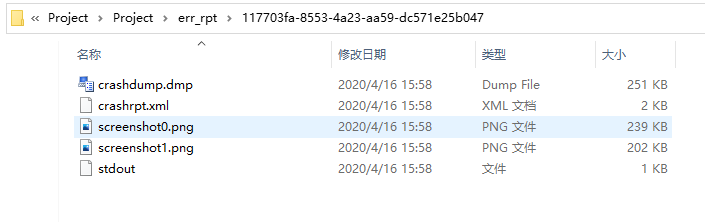


图5-1-4