# 采集工作站开发笔记

目录

[采集工作站开发笔记 1](#_Toc392787813)

[系统柜架图 3](#_Toc392787814)

[USB模块 3](#_Toc392787815)

[数据库 4](#_Toc392787816)

[插入媒体信息 5](#_Toc392787817)

[插入DVR日志 5](#_Toc392787818)

[插入终端日志 6](#_Toc392787819)

[上报终端状态 6](#_Toc392787820)

[根据ID实时查询姓名 7](#_Toc392787821)

[HTTP协议 7](#_Toc392787822)

[POST的方式 7](#_Toc392787823)

[GET方式 8](#_Toc392787824)

[注意 8](#_Toc392787825)

[转码操作 9](#_Toc392787826)

[思路 9](#_Toc392787827)

[转换命令 9](#_Toc392787828)

[步骤 9](#_Toc392787829)

[实现 9](#_Toc392787830)

## 系统柜架图

UI(Duilib库) ---- XML构建

逻辑层

类工厂及全局对象（单例）

配置文件

数据库逻辑（存储过程）

设备抽象层

公共库

凌锐

华安

Well

驱动

ADO

## USB模块

实现代码在USBCtrl文件中

此模块是实现的核心，以下列出开发的思路

// 1. 遍历所有USB设备，获取他拉ID

// 1.1 CM\_Get\_Child()和CM\_Get\_Device\_ID()获得其对应的USBSTOR的设备ID（形如USB\VID\_0595&PID\_2002\ZORAN01234567）

// 1.2 通过 USB\\VID\_0595&PID\_2002\\ 过滤掉不相关的设备

// 2. 遍历SYSTEM\\MountedDevices下带\\DosDevices\\X:的健，把他二进值存进来(盘符和值的应射表）

// //遍历A-Z

// for (n = 0; n < 26; ++n)

// {

// sprintf(strTmp, "\\DosDevices\\%c:", 'A' + n);

// dwLen = sizeof(strValue);

// if( (dwLen = GetRegKeyValue(strSKey1, strTmp, REG\_BINARY, strValue, dwLen)) != 0)

// {

// j = 0;

// for( i = 0; i < dwLen; ++i)

// {

// if(strValue[i] != 0)

// {

// //去掉每个字符中间00

// \*(strValue1[n] + j++) =strValue[i];

// }

// }

// }

// }

//

// 3. 打开SYSTEM\\CurrentControlSet\\Enum\\ + USBSTOR的设备ID 的键

// 如 SYSTEM\\CurrentControlSet\\Enum\\USB\\VID\_0595&PID\_2002\\ZORAN01234567

// 2.1 获取ParentIdPrefix的值(),如果获得，转到4, 如果没有ParentIdPrefix转到5

// 4. 把获取得到的ParentIdPrefix的值（8&16a2684b&0）,和第二步得到的值对比，找到则说明盘符找到

// //有ParentIdPrefix的，直接可以找到盘符

// //\_??\_USBSTOR#Disk&Ven\_ZOKOD\_DV&Prod\_R&Rev\_1.10#8&16a2684b&0&ZORAN01234567&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}

// 4.1 匹配到ParentIdPrefix则找到盘符

//

// 5. 没有ParentIdPrefix(每一个插入或枚举的盘不会产生ParentIdPrefix)

// //第一个枚举成U盘的，没有ParentIdPrefix的

// //\_??\_USBSTOR#Disk&Ven\_ZOKOD&Prod\_\_\_DVR&Rev\_1.10#ZORAN01234567&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}

// 5.1 提取查找串( # + USBSTOR的设备ID的最后一部分),如#ZORAN01234567

// 5.2 和第二步得到的值对比（查找），得到盘符

//

## 数据库

采集工作站需和后台管理系统进行交互，具体的交互方式通过数据库和http协议

数据库的操作类及存储过程的调用封装在DataBase文件中

以下是数据库的存储过程：

### 插入媒体信息

//////////////////////////////////////////////////////////////////////////

//描述：插入视频（媒体信息)

//时间：created by yjt 2014-07-03

//存储过程如下：

// P\_INSERT\_DVRMEDIA\_INFO (

// iDvrNumber in varchar2, -- DVR编号

// iFileName in varchar2, --文件名

// iFileSize in number, --文件大小

// iViewWidth in number, -- 宽

// iViewHeight in number, -- 高

// iMediaDuration in number, --时长

// iFrameRate in varchar2, --帧率

// iDataUploadTime in date, --上传时间

// iMediaStartTime in date, --开始时间

// iMediaEndTime in date, --结束时间

// iRecordTime in date, -- 录像时间

// iFileType in varchar2, -- 文件类型

// iVideoType in varchar2, -- 重点/高清/标清

// iLocateFilePath in varchar2, --本地文件路径

// iFtpFileUrl in varchar2, --ftp文件路径

// iIISFileUrl in varchar2, --iis文件路径

// iDataIp in varchar2, --数据存储ip

// iRemark in varchar2 --备注

// )

//////////////////////////////////////////////////////////////////////////

实现函数：

BOOL CDataBase::InsertMediaInfo(const DVRMEDIA\_INFO& info)

### 插入DVR日志

//////////////////////////////////////////////////////////////////////////

//描述：插入DVR日志

//时间：created by yjt 2014-07-03

//存储过程如下：

// P\_INSERT\_DVR\_DEVICE\_LOG (iDvrNumber in varchar2, --dvr编号

// iDeviceInsertTime in date, --设备插入时间

// iDeviceOperTime in date, --设备操作时间

// iDeviceOperUser in varchar2, --设备操作用户

// iLogContent in varchar2, --日志内容

// iRemark in varchar2 --备注

// )

//////////////////////////////////////////////////////////////////////////

实现函数：

BOOL CDataBase::InsertDvrLog(const DVR\_DEVICE\_LOG& ddLog)

### 插入终端日志

//////////////////////////////////////////////////////////////////////////

//描述：插入终端日志

//时间：created by yjt 2014-07-03

//存储过程如下：

// //终端日志

// P\_INSERT\_TERMINAL\_DEV\_LOG (

// iDeviceNumber IN VARCHAR2, --设备号

// iHarewareVersion IN VARCHAR2, --硬件版本

// iSoftwareVersion IN VARCHAR2, --软件版本

// iDiskSpace IN NUMBER, --磁盘空间

// iTerminalIp IN VARCHAR2, --IP

// iIsusing IN VARCHAR2, --是否启用 -Y是 -N否

// iDetachment IN VARCHAR2, --支队

// iBattalion IN VARCHAR2, --大队

// iSquadron IN VARCHAR2, --中队

// iDescribe IN VARCHAR2, --描述

// iRemark IN VARCHAR2 --备注

// )

//////////////////////////////////////////////////////////////////////////

实现函数：

BOOL CDataBase::InsertLog(const TERMINAL\_DEV\_LOG& tdLog)

### 上报终端状态

//////////////////////////////////////////////////////////////////////////

//描述：上报终端状态

//时间：created by yjt 2014-07-03

//存储过程如下：

//终端状态

// P\_INSERT\_TERMINAL\_STATE (

// iDeviceNumber IN VARCHAR2, --设备号

// iReportTime IN date, --报告时间

// iDeviceState IN VARCHAR2, --设备状态

// iRemark IN VARCHAR2 --备注

// )

//////////////////////////////////////////////////////////////////////////

实现函数：

BOOL CDataBase::InsertStatus(const TERMINAL\_STATE& ts)

### 根据ID实时查询姓名

select DVRNUM,REALNAME from mycar.DVR where DVRNUM='%s'

实现函数：

CString CDataBase::QueryNameFormID(const CString& id)

## HTTP协议

网络层在network文件中实现，由于http是基于TCP协议的，而我们只是上报状态，很简单，所以用TCP协议 + http头来实现 Http协议

具体的实现在NetHttpClient.cpp/h中

发送的状态地址如下：

<http://192.168.1.90:9090/hdvr/devHealthMonitor?deviceNumber=000001111&storeSpace=200&freeSpace=2000&runningState=normal&accessToken=B76269AC8F66BE563332ED094ED17AEB&actionType=reportHealth>

基中：api = [/hdvr/devHealthMonitor?deviceNumber=000001111&storeSpace=200&freeSpace=2000&runningState=normal&accessToken=B76269AC8F66BE563332ED094ED17AEB&actionType=reportHealth](http://192.168.1.90:9090/hdvr/devHealthMonitor?deviceNumber=000001111&storeSpace=200&freeSpace=2000&runningState=normal&accessToken=B76269AC8F66BE563332ED094ED17AEB&actionType=reportHealth)

hostname = 192.168.1.90:9090

### POST的方式

//初始化发送信息

//头信息

os << "POST " << api << " HTTP/1.1\r\n"

<< "Host: " << hostName << "\r\n"

<< "Connection: keep-alive\r\n"

<< "Content-Length: " << strlen(parameters) << "\r\n"

<< "Cache-Control: max-age=0\r\n"

<< "Origin: http://www.glwell-co.com/html/main.asp\r\n"

<< "User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/15.0.849.0 Safari/535.1\r\n"

<< "Content-Type: application/x-www-form-urlencoded\r\n"

<< "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8\r\n"

<< "Accept-Encoding: gzip,deflate,sdch\r\n"

<< "Accept-Language: zh-CN,zh;q=0.8\r\n";

### GET方式

os << "GET " << api << " HTTP/1.1\r\n"

<< "Host: " << hostName << "\r\n"

<< "Connection: keep-alive\r\n"

<< "Content-Length: " << strlen(parameters) << "\r\n"

<< "Cache-Control: max-age=0\r\n"

<< "Origin: http://www.glwell-co.com/html/main.asp\r\n"

<< "User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/15.0.849.0 Safari/535.1\r\n"

<< "Content-Type: application/x-www-form-urlencoded\r\n"

<< "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8\r\n"

<< "Accept-Encoding: gzip,deflate,sdch\r\n"

<< "Accept-Language: zh-CN,zh;q=0.8\r\n";

### 注意

由于每个终端都要上报，且有一定的时间间隔，为了不占用连接，以http短连接实现，即每次都要连接-》发送-》关闭

1. FTP协议

FTP是单机版中，多级存储中都用到，对FTP进行封装，实现文件在ftp文件夹中

主要是实现：

* 1. 三级FTP存储，有上传优先级
  2. 上传速度动态计算和显示
  3. FTP断点继传功能的实现

## 转码操作

### 思路

利用ffmpeg把avi转成flv, 通过配置不同的参数可以达到不同的效果

### 转换命令

//高质量: ffmpeg -i 20140625\_111916.avi -ab 128 -acodec libmp3lame -ac 1 -ar 22050 -r 29.97 -qscale 6 -y 2.flv

//低质量: ffmpeg -i 20140625\_111916.avi -ab 128 -acodec libmp3lame -ac 1 -ar 22050 -r 29.97 -b 360 -y 3.flv

### 步骤

1. 上传完成的文件进行判断，不是视频文件的拷贝完删除
2. 是视频文件的入队（先进先出）
3. 从队列头中取出一个视频进行转码操作
4. 循环2-3步骤

### 实现

bool CConvertVideo::Avi2Flv(const wstring& sAviFile, const wstring& sFlvFile, int width, int Height)

{

wstringstream os;

os << GetAppPath().GetString() << \_T("\\ffmpeg.exe -i ")

<< sAviFile

<< \_T(" -ab 128 -acodec libmp3lame -ac 1 -ar 22050 -r 29.97 -b 360 -s ")

<< width << \_T("x") << Height

<< \_T(" ")

<< sFlvFile

<< \_T("");

return Convert(os.str());

}

bool CConvertVideo::Convert(const wstring& strCmd)

{

SECURITY\_ATTRIBUTES sa;//创建一个安全属性的变量

HANDLE hRead,hWrite; //管道的读写句柄声明

sa.nLength = sizeof(SECURITY\_ATTRIBUTES); //获取安全属性的长度

sa.lpSecurityDescriptor = NULL; //使用系统默认的安全描述符

sa.bInheritHandle = TRUE; //创建的进程允许继承句柄

if (!CreatePipe(&hRead,&hWrite,&sa,0)) //创建匿名管道

{

//MessageBox(\_T("CreatePipe Failed!"), \_T("提示"), MB\_OK | MB\_ICONWARNING);

return false;

}

STARTUPINFO si; //启动信息结构体变量

PROCESS\_INFORMATION pi; //需要传入的进程信息的变量

ZeroMemory(&si,sizeof(STARTUPINFO)); //因为要传入参数，所以先清空该变量

si.cb = sizeof(STARTUPINFO); //结构体的长度

GetStartupInfo(&si);

si.hStdError = hWrite;

si.hStdOutput = hWrite; //新创建进程的标准输出连在写管道一端

si.wShowWindow = SW\_HIDE; //隐藏窗口

si.dwFlags = STARTF\_USESHOWWINDOW | STARTF\_USESTDHANDLES;

TCHAR cmdline[512] = {0};

CString tmp,stredit2;

//GetDlgItemText(IDC\_EDIT2,stredit2); //获取编辑框中输入的命令行

//tmp.Format(\_T("cmd /C %s"),stredit2);

tmp.Format(\_T("cmd /C %s"), strCmd.c\_str());

swprintf(cmdline,\_T("%s"),tmp);

if (!CreateProcess(\_T("C://Windows//System32//cmd.exe"),cmdline,NULL,NULL,TRUE,NULL,NULL,NULL,&si,&pi)) //创建子进程

{

DWORD dwErr = GetLastError();

//MessageBox(\_T("CreateProcess Failed!"), \_T("提示"),MB\_OK | MB\_ICONWARNING);

return false;

}

CloseHandle(hWrite); //关闭管道句柄

char buffer[4096] = {0};

CString strOutput;

DWORD bytesRead;

while (true)

{

if (ReadFile(hRead,buffer,4095,&bytesRead,NULL) == NULL) //读取管道内容到buffer中

break;

strOutput += buffer;

//m\_edtMsg.SetWindowText(strOutput);

//SetDlgItemText(IDC\_EDIT2,strOutput); //显示输出信息到编辑框,并刷新窗口

//UpdateWindow();

//Sleep(100);

}

CloseHandle(hRead); //关闭读取句柄

return true;

}