# Bug

## Xsh

Xsh(host,”top –p `pgrep SCService`)

## 配置文件

addConfString

delConfString

现在只能严格按格式,未解决空格等问题

[section]#....

Key = fwefwebebre这种的,查一下需不需解决..其实不需要解决,就这样按格式来

# *最新代码以及模板等*

## Java

Xlib最新在ws\_client中

## Qssh

### Remoteprocess远程执行命令

Windows最新在qlib中,增加了qxsh代替原来的xsh,支持多线程,qxsh使用的qssh中提取的remoteprocess(C:\Users\Administrator\Desktop\xlib\windows\projects\QSshT)作为远程执行指令,已测试每次200ms,可以接受已经比xsh快很多了, 使用在uitrans中,也测试了网上的libssh2,慢

之后考虑不要读取文件,直接读取输出—直接输出有问题,以后有时间再改

因为直接用代码多线程有问题,所以做成exe调用进程等待完成读取结果文件

目前qssh的remoteprocess两个版本,一个qxsh,结果存文件,目前xsh用的这个,没有问题,每次间隔200ms可以接受,一个qxshnofile,命令行没问题,但是用c++getcmdoutput不输出,qtqprocess也不行(xprocess),之后再调

### Sftp

与remoteprocess一样的处理

目前发现uploadfile 和uploaddir不能再对方没有目录的时候进行,针对这个问题也要做成exe

至于速度显示,可以另做一个线程定时读取对方文件大小速度(文件)

目录的话,待定

接口remotepath localpath type(uploadfile uploaddir downloadfile downloaddir)

因为其内部并不判断文件夹存不存在(已验证测试),所以做成通用exe后自己用remoteprocess判断输入参数正确性,比如不存在则创建

注意这个超时不能少于5秒,不然会有问题,显示断开连接啥的

Uploadfile dir downloadfile都有,但是没有downloaddir?

## Plib

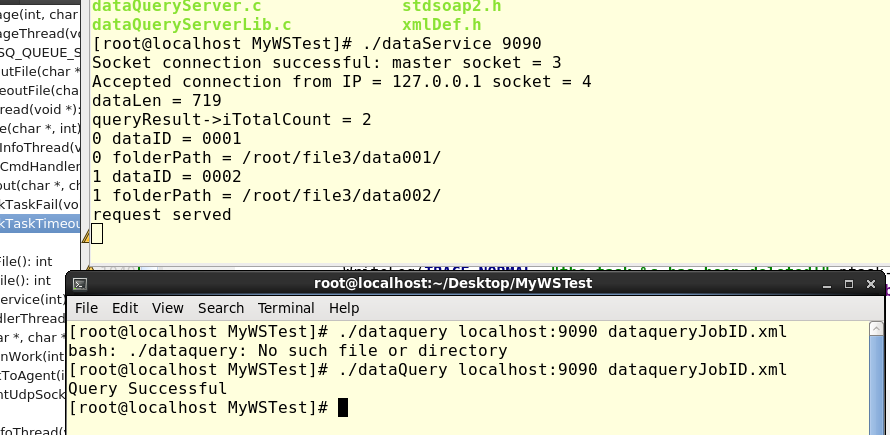
Linux/qlib/plib

所有东西都在一个文件夹,pmysql如果没装可以注释,去掉引用不用删就行

既有clion也有qt,全面使用

## Webservice

Linux centos 桌面上

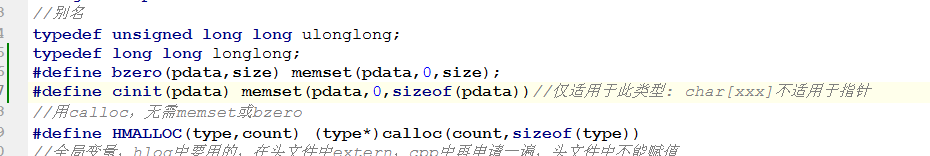


## Linux

Plib是非c++11 的库,

Plib11是基于plib的支持C++11的库,写的一些东西比如跨平台的库,如trans

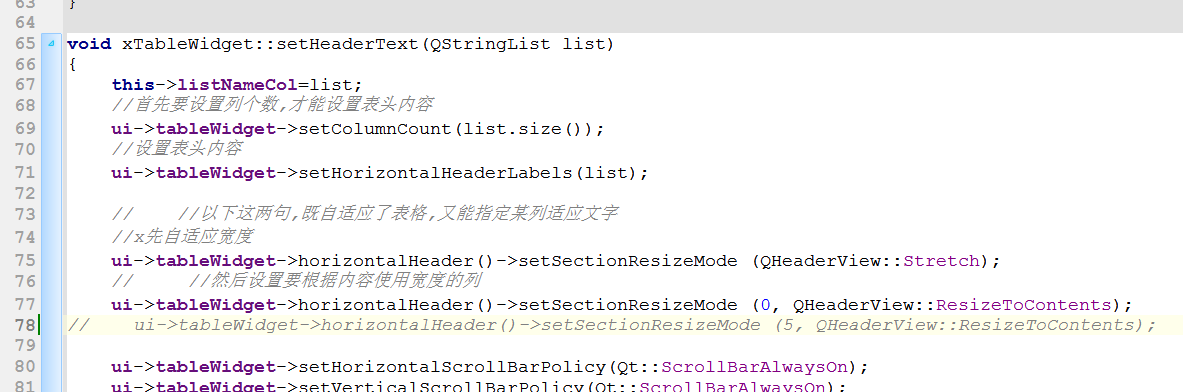
## *宏定义*

**

## *xTableWidget*

*目前最新是在C:\Users\Administrator\Desktop\xlib\windows\projects\testxTableWidget*

*已有排序追踪功能,已固定*font: 12pt 'DejaVu Sans Mono



*使用时注意这里,不然越界崩*

## xTreeWidget

*目前最新是在C:\Users\Administrator\Desktop\xlib\windows\projects\testxTableWidget*

## *QT*

### C++库

### Qlistwidget

在xlfd中

### Treewidget

在联通项目vedio-mmfpeg中,转换成了utf8代码

#### 备忘

增加了xsh getCMDoutput等,类似于linux下,要使用plink

增加xshbridge plink 和sshpass的完美结合使得跨平台成为可能

下载了自动接受key的plink版本

增加-batch,不然输错密码会一直卡主

增加了获取远程根目录以及桥接远程根目录

修改getCMDoutput兼容性,不弹黑框

增加了ping函数,只返回是否能ping通

增加了嵌入外部程序的完整解决方案xembedwidget,嵌入时不弹框,不弹黑框

增加qlibtoqstring(默带中文处理)

### Qss样式等

*C:\Users\Administrator\Desktop\xlib\windows\qt\projects\xlfd*

### 嵌入外部程序库

最新在embed中

并且集成了自动登录vpn的qt类加界面也在里面,发送到vpnmix中调用

直接拖控件就行,自动登录,断了自动重连

### Vpn自动登录后台

在vpnAutoOnly中,包括嵌入和vpn集成控件

## C#

### xlib

*目前最新在自动登录vpn中中*

## 自动登录vpn

最新就是这个,要注意vpn2改成网站模式后,webbrowser不支持多线程因此出现错误

Webbrowser必须前台,所有不用了,改用C++使用句柄实现后台,最近叫vpnAutoNoEmbed

# 近期要改

# 库结构

## Windows

Windows下如果写界面,直接拷贝qlib文件夹

如果调用c++,拷贝qlib下的plib文件夹.

以后就维护linux以及windows下的两个文件夹

# 要产品化的东西

## 内网穿透

## 单播转组播

Transform是最新

# 要改的东西

## 客户端中科院

查看任务队列,分配带宽,链路,磁盘任务数据,查看修改配置文件

非实时修改job号

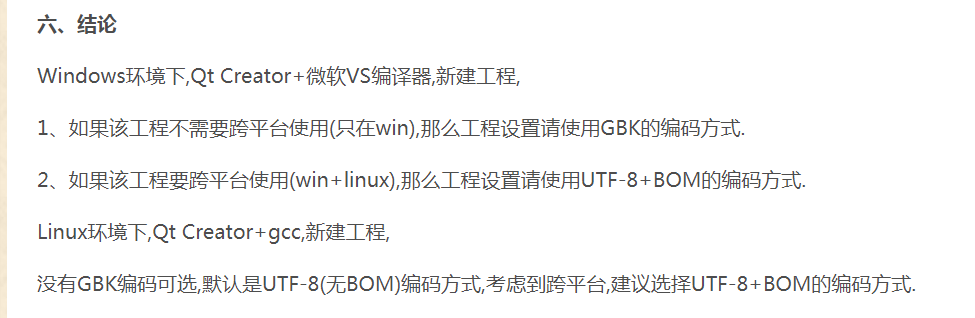
# *问题*

Udp组播同一端口,地址不一样(接近)也能收到?

## 考虑ptcp多线程接收的data,length type问题

这个没问题,因为多线程接收,每次都是来一个连接新建一个线程,新建一个socket,新建一个ptcp,所以不存在冲突问题

## 中文,常量中有换行符



UTF-8 BOM选择 如果编码是UTF-8则添加；

## *C++ windows下char[666666]申请一个没问题,申请两个就崩了?linux下没问题*

*Windows下静态分配的局部数组使用的是栈上的空间,因此是栈的大小为上限,一般是1M或2M,单个进程的最大栈空间一般是2M,实际比这小.亲测1M因为申请两个就崩了,所以不能超过两个,所以发送数据一般就用64k吧hlog时相当于复制另一个,因此也是增加的,一个序列化反序列化测试就四五个,算上,所以在上一个释放完了之后在用下一个.如果非要很大,请用全局变量*

*Linux栈比较大,但是也不能太多,就64k一包吧*

*写一个通用的c++多线程传输库*

# Vs2013

找到C:\ProgramData下的package cache删除 重新装

# Vncviewer

可登陆版卡,设置不用代理就可以了

# Windows cmd

窗口小设置一下

# 各种数据

Xshell数据 xlib下

# 关于vpn

自动登录vpn因为经常会前台化影响作业,因此一般放在微软分桌面的第二个桌面中即可

分桌面还有个用可以用来按键精灵

# Windows ssh远程执行命令

Plink\_32.exe –pw sjcsfwq [root@106.13.131.240](mailto:root@106.13.131.240) “pwd;ls”

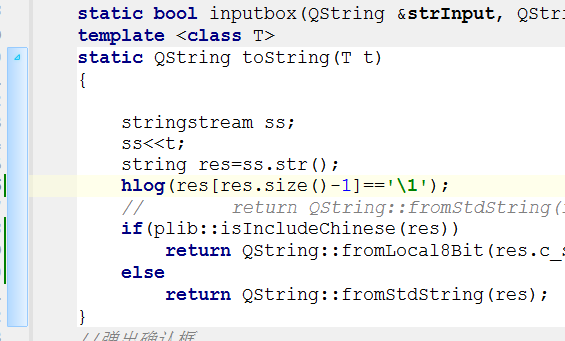
-batch是关闭所有交互提示

**hlog**(plib::getCmdOutput(**"plink.exe** **-pw** **sjcsfwq** **root@106.13.131.240** **\"pwd;ls\""**));

正常plink要求手动输入自动保存key, 需要这样echo y|plink…但是这样的话getCmdOutput不支持,所以需要特定版本的plink自动保存,下下来了别丢了

# Pstring问题

Pstring转qstring后,qstring会把\1也转进来,考虑解决方法:



是因为在此转换时出问题,要封装,pstring或者string不能直接转要跳过,否则就会出现加\1的问题

另外要增加qlib::tostring自身就是QSring的问题

要修改转换,判断等相关函数,记得更新到linux中

用\0的时候发现用cout就没问题,原因是最后写入的时候用strlen等,没有指定长度修改这个bug

Linux同步修改之后pstring就可以用\0了

# Pstring recvx改进

## 跨windowslinux平台处理中文乱码问题

类似于QString::fromlocal8bit,不要在pstring中封装,也不要在recvx中封装,会导致各种问题,在需要的时候,觉得哪里可能有中文的时候用转换就行,比如跨平台传输linux ut8到windows接收时用一下,跨平台解决方案,代码一样

static pstring toChinese(const char\* str)

{

bool isHaveChinese=plib::isIncludeChinese(str);

hlog(isHaveChinese);

//如果不含有中文,直接返回

if(!isHaveChinese)

return pstring(str);

pstring strSystem=plib::getSystem();

bool iutf8=plib::isUTF8(str);

hlog(strSystem,iutf8);

//如果是windows,且含有中文,且字符串不是GBK而是UTF8,则要转换成GBK

if(strSystem=="windows"&&iutf8)

{

return plib::toGBK(str);

}

else if(strSystem=="linux"&&!iutf8)

{

//如果是linux,且含有中文,且不是UTF8,要转成UTF8

return plib::toUTF8(str);

}

return pstring(str);

}

# 要改进的地方

## Pmap根据索引读取

## 传输程序内存增长的问题

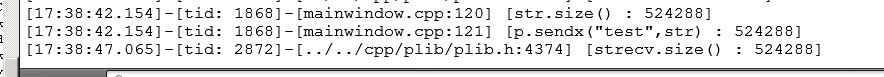
# 抓包

正常情况下,服务端开启的情况下是不可能出现win=0的,关闭服务端会出现win=0,如果服务端没关出现了win=0说明不能接受数据了

# 文件传输问题

## 传不过去,一直断点续传(百度云服务器)

Windows下这是因为发送512k包时,广域网时延太大,发过去之后服务端很久才收到,而这时已经在断点续传了,所以第二包就发不过去,就一直断点续传



5秒才收到

而连续发两包大包会导致第二包发送失败,返回-1

// while ((iRet = send(iSocket, pBuffer + iSend, iRet, 0)) < 0 && (errno == EINTR || errno ==EAGAIN))

while ((iRet = send(iSocket, pBuffer + iSend, iRet, 0)) < 0)

将那个两个错误处理去掉后,send会阻塞到接收端处理完之后再返回,改之前是非阻塞的所以第二包直接失败,所以应该改为这样,send和

Recv不要改否则会导致连续发送大包接收错误

将发送接收缓冲区调大(666k)后,发送两包没问题,但是发送多包超过666就有问题了

发送大包后,recv端只接收了一部分,所以要判断包长和接收的长度是否相同!!!!要判断

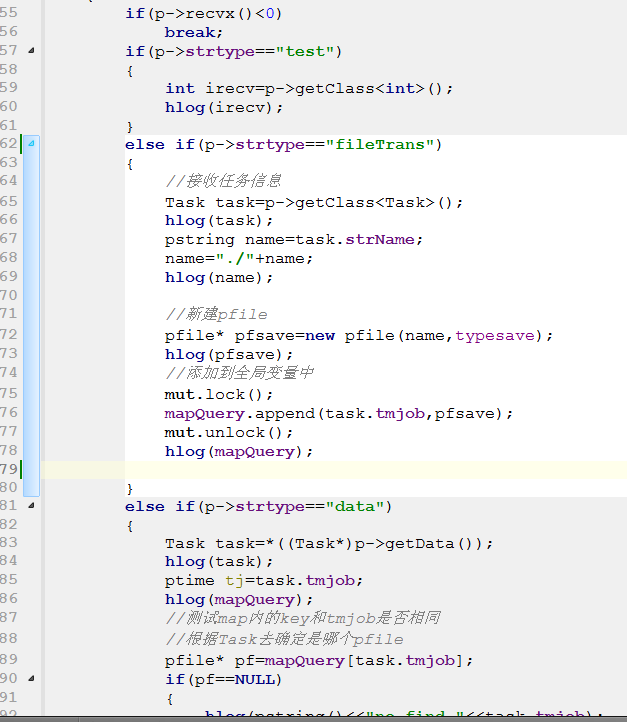
所以recvx和sendx要统一,有数据的话是一致的,要改没数据的只发包头的,这个length和包头长度不一致,都改成0

又字节数不一致的问题

再网路不好的情况下,传输发送失败,断线重连,这个时候接收端原来线程半天才反应过来收到包了,但是这时候重连新的线程又开始发了,所以要同步多线程的问题,还有就是发送完成帧时还有其他线程在写文件,都要考虑

最简便的方法就是发完数据等待回执,收到回执后再发下一包,不然不发,这样最保险

# *文件传输逻辑(不要分开)*

**

*不用使用这种模式,增加难度,只要一个filetrans后边接着收发就行了!!!*

## 断点续传

Fseek只能最大2G,所以会失败,要用windows下的\_fseeki64

## 百度云服务器上fwrite错误的问题

**EINVAL**

Invalid argument. An invalid value was given for one of the arguments to a function. For example, the value given for the origin when positioning a file pointer (by means of a call to**fseek**) is before the beginning of the file.

在网上查找，终于找到一个外国朋友写的帖子，问题描述基本一样，并且他们做了很多的实验，发现在低转数的硬盘上容易（不是一定）出现。而且和写入的数据大小也有关，当写入的数据较大，容易出现，当降低写入的数据大小，出现的概率随之降低。据此，我将要写入文件的数据分成小块儿分次使用fwrite写入文件。最终问题得到了解决！发帖以供以后遇到此问题的朋友参考。

测试一下,然后要加回执,每一包加个回执测试写入是否成功,影响效率的话,重新写入直到成功

要加上磁盘剩余空间,以及内存判断

直接在百度云上256k写入试试

# Lambda函数

**auto** func = [] (pstring data)

{

**hlog**(data);

};

func(**"xlfd"**); *//* *now* *call* *the* *function*

# *关于plist效率低的问题*

*//接收数据缓冲区，不再用char\*，提供专门的转换接口*

*plist<****char****>* ***data****;*

*//接收一次后使用，根据使用的结构体转换*

***char*** *\*ptcp::getData()*

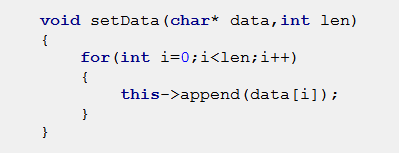
*{*

***return*** *(****char****\*)(&(\*(****this****->****data****.begin())));*

*}*

*这样能提高效率?*

## 提高plistbyte效率测试

事实证明

最耗时间的是这个,不要这么干!

vector里很重要的一个函数 vector::assign() //用来构造一个vector的函数，类似于copy函数

也可以用copy函数

函数原型是：   
1:void assign(const\_iterator first,const\_iterator last);  //两个指针，分别指向复制开始和结束的地方!

要使用assign

不过即使增加了效率,2G文件assign依然要50多秒,而直接使用char数组不到1秒,因此发送大文件还是用char数组吧,别折腾了plistbyte观赏用就行了

有时间可以测试一下qbytearray看看效率

另一个思路,不要用plistchar

pstring,pstring证明可以当成字符数组来看,而且效率高,但是不能当做发送的数据,因为序列化反序列化是根据字符串来分解的,所以会乱,sendb内部有一块

研究直接发送字符串,速度也很快

String 本质也是plist<char>的封装,因此只要不是一个一个字符复制都可以

修改了sendx,现在可以直接发送字符串了,包括如下形式:

Sendx(“type”,”string”)

String data; sendx(“type”,data)

Pstring data;sendx(“type”,data);

也就是改完之后可以直接读取到pstring发送数据了

都直接用recvx接收,且用getDataToString接收

将plistbyte改造,不用plistchar了,用pstring

# 测试读写性能

2.6Gvs2010.iso

## 直接调用c语言fread

string path=**"D:/vs2010ultimate.iso"**;

**int** len=256\*1024;

pfile file(path,typeload);

**int** iread=-1;

**char** data[333333];**cinit**(data);

ptime ts;

**while**((iread=file.load(data,len))>0)

{

*//* *hlog(iread);*

}

ptime te;

**hlog**(te-ts);

file.close();

0.59秒

## 调用封装pstirng(局部变量)(最快的方式)

string path=**"D:/vs2010ultimate.iso"**;

**int** len=256\*1024;

pfile file(path,typeload);

**int** iread=-1;

**char** data[333333];**cinit**(data);

ptime ts;

**while**(1)

{

pstring dr=file.load(len);

**if**(dr.size()==0)

**break**;

}

ptime te;

**hlog**(te-ts);

file.close();

1.157秒

## 调用封装pstirng(上层变量)

string path=**"D:/vs2010ultimate.iso"**;

**int** len=256\*1024;

pfile file(path,typeload);

**int** iread=-1;

**char** data[333333];**cinit**(data);

ptime ts;

pstring dr;

**while**(1)

{

dr=file.load(len);

**if**(dr.size()==0)

**break**;

}

ptime te;

**hlog**(te-ts);

file.close();

1.7秒

# 关于string:

## **c\_str()：生成一个const char\*指针，指向以空字符终止的数组**

## **data():与c\_str()类似，但是返回的数组不以空字符终止。**

## 直接写入到string中

需要先分配空间

**char** ad[5];**cinit**(ad);

strcpy(ad,**"xlfd"**);

pstring str;

str.resize(5);

copy(ad,ad+5,str.begin());

**hlog**(str);

因此可以解决调用c时直接写入char\*的问题如file.load

说明可以直接对str.begin()进行操作

*//* *读取到pstirng中,如果size==0表示读不到了*

*//用来读一包的，一般最大512k*

pstring load(**int** len)

{

pstring strdata;

strdata.resize(len);

**int** iread=**this**->load((**char**\*)(&(\*(strdata.begin()))),len);

**if**(iread<0)

{

strdata.resize(0);

**return** strdata;

}

strdata.resize(iread);

**return** strdata;

}

# *类成员函数做线程函数*

## 目前可用的方法

*//类的成员函数回调,该函数兼容普通函数回调,直接fun带入普通函数指针即可*

*//* *class* *Test*

*//* *{*

*//* *public:*

*//* *void* *show(pstring* *str)*

*//* *{*

*//* *hlog(str);*

*//* *}*

*//* *};*

*//* *void* *xshow(pstring* *str)*

*//* *{*

*//* *hlog(str);*

*//* *}*

*//使用时默认两参数,不回调*

*//* *hlog(trans.uploadFileCallback(host,"D:\\qt-opensource-windows-x86-mingw492-5.5.0.exe");*

*//* *Test* *t;*

*//* *hlog(trans.uploadFileCallback(host,"D:\\qt-opensource-windows-x86-mingw492-5.5.0.exe",*

*//* *std::bind(&Test::show,&t,placeholders::\_1)));*

*//或者*

*//* *hlog(trans.uploadFileCallback(host,"D:\\qt-opensource-windows-x86-mingw492-5.5.0.exe",xshow);*

**int** uploadFileCallback(pstring strhost,pstring strPathFull,

std::function<**void**(pstring)> fun=printString)

{

std::thread\* pthUploadFile=**new** std::thread(

&Trans::uploadFileThreadCallback,**this**,strhost,strPathFull,fun);

pthUploadFile->detach();

**return** 0;

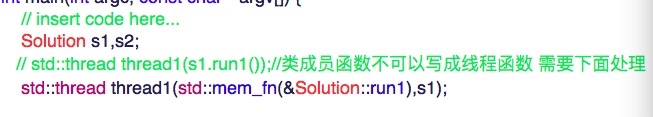
}

该方法同时支持普通函数以及类成员函数

默认是全局函数,也可用类中的函数,但是不能以默认参数的形式,因为this不能再参数中使用,只能在函数 内使用 **this**->**funRecv**=std::bind(&Trans::show,**this**,placeholders::\_1);

## 第一种(要测试)

当我们创建线程的时候 线程处理函数如果是类里面的成员函数会报错，需要以下处理。



当需要利用类成员函数( MyClass::thread\_func )来创建子线程时，需如下码码：

t[i] = thread(std::mem\_fn(&MyClass::thread\_func), Object, args..);

如果thread\_func为static，则不用写object。否则需要，如主进程所调函数也为该类成员，则传入this指回自己。

menfn这个相当于STL中内置的仿函数，可以使用调取STL容器内对象的内置函数；

mem\_fn最为人所熟知的作用是，将一个成员函数作用在一个容器上，就像这样std::for\_each(v.begin(), v.end(), boost::mem\_fn(&Shape::draw))就可以让容器vector中的每一个元素都执行一遍draw方法。

第二个用法是，它可以帮助把一个函数指针模拟得像一个函数实体(function object)。

————————————————

版权声明：本文为CSDN博主「HELLO\_蓝猫」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/BJUT_bluecat/article/details/85201576>

## 第二种(需测试)

友元函数：

    把线程函数设置成类的友元函数，这样线性函数就能访问类中的成员。

例程：

#include "windows.h"

#include <process.h>

class ExampleTask

{

public:

friend void taskmain(LPVOID param);

void StartTask();

int value;

};

void taskmain(LPVOID param)

{

ExampleTask \* pTaskMain = (ExampleTask \*) param;

//通过pTaskMain指针引用

}

void ExampleTask::StartTask()

{

\_beginthread(taskmain,0,this);

}

int main(int argc, char\* argv[])

{

ExampleTask realTimeTask;

realTimeTask.StartTask();

return 0;

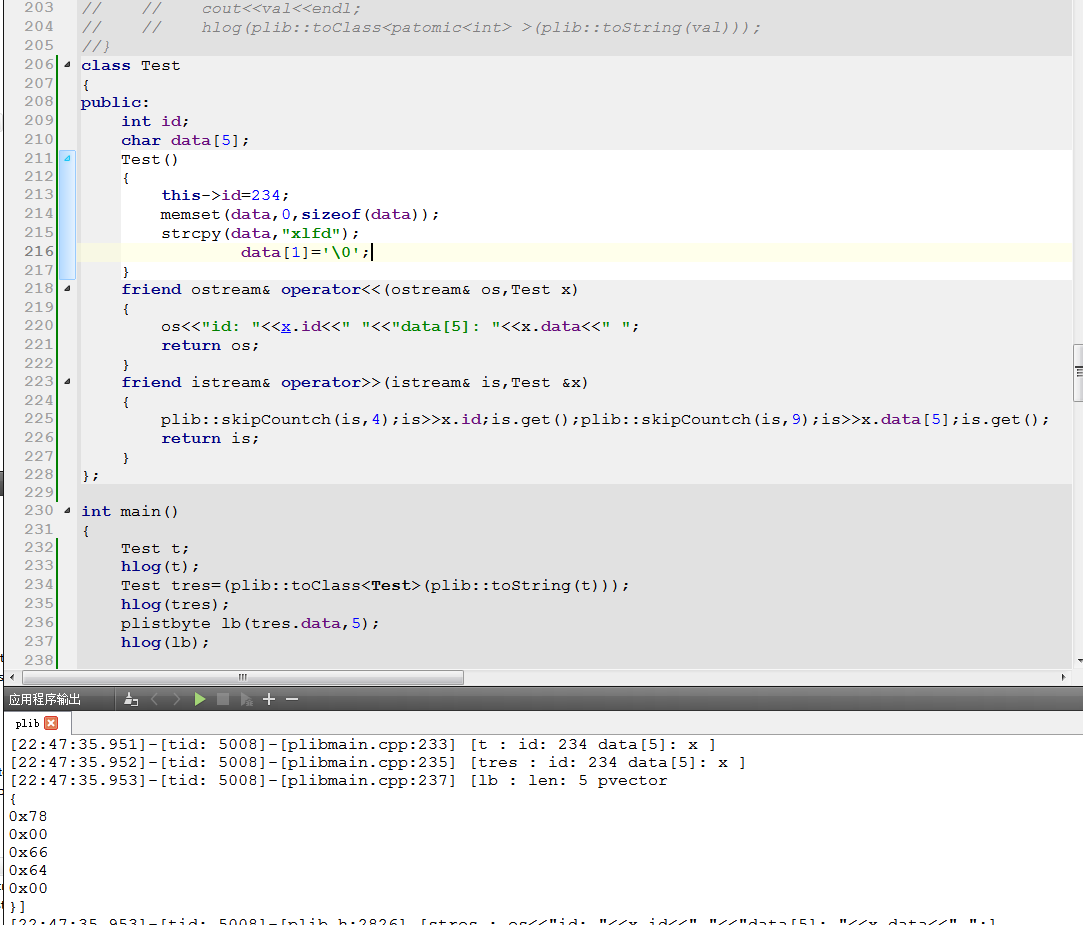
# *要同步到linux的*

*Ptime(支持ms)pbyte plistbyte*

*plistbyte好像速度特别慢,如果是发送数据,读取数据不如直接用char\*,所以不要封装了,plistbyte没啥用,可以用来查看16进制*

*直接序列化反序列化char数组(在结构体里面),但此时又有另一问题就是data数组是不能打印的,不然太长..一般都是数据*

# *序列化与反序列化*

**

在进行序列化时,虽然对数组中的第二元素进行了赋值\0的操作,但是从结果看依然是成功导入导出了,因此可以信赖c++中的直接对字符串数组序列化,反序列化操作,又因为字符串数组效率最高,因此发数据或者其他时候就不要封装了,直接声明一个类(包头加char数组)进行序列化就可以适应所有场景

直接使用>> <<并不能解决而问题,因为>>是使用的字符串,因此如果data[66666]全是0的话就不会计算长度,因此要重新写这个的序列化,挨个输入进去,但是挨个输入进去的话又会导致效率降低,因此char数组不适合进行序列化/反序列化,直接使用就好