## mtkPack C++开发库参数手册

目录

[mtkPack C++开发库参数手册 1](#_Toc446680691)

[1.概要 3](#_Toc446680692)

[2.类参考 3](#_Toc446680693)

[2.1 Cabinet.h 3](#_Toc446680694)

[2.1.1函数说明 3](#_Toc446680695)

[2.1.2使用例 4](#_Toc446680696)

[2.2 Codecs.h 4](#_Toc446680697)

[2.2.1函数说明 4](#_Toc446680698)

[2.2.2使用例 5](#_Toc446680699)

[2.3 ComputerInfo.h 5](#_Toc446680700)

[2.3.1 函数说明 5](#_Toc446680701)

[2.3.2使用例 5](#_Toc446680702)

[2.4 Engine.h 6](#_Toc446680703)

[2.4.1函数说明 6](#_Toc446680704)

[2.4.2使用例 6](#_Toc446680705)

[2.5 File.h 7](#_Toc446680706)

[2.5.1函数说明 7](#_Toc446680707)

[File类 7](#_Toc446680708)

[File::Iterator类 9](#_Toc446680709)

[2.5.2使用例 10](#_Toc446680710)

[2.6 IniFile.h 10](#_Toc446680711)

[2.6.1函数说明 11](#_Toc446680712)

[2.6.2使用例 13](#_Toc446680713)

[2.7 MRUList.h 13](#_Toc446680714)

[2.7.1函数说明 14](#_Toc446680715)

[2.7.2使用例 14](#_Toc446680716)

[2.8 Process.h 15](#_Toc446680717)

[2.8.1函数说明 15](#_Toc446680718)

[2.8.2使用例 16](#_Toc446680719)

[2.9 SecureDelete.h 16](#_Toc446680720)

[2.9.1函数说明 16](#_Toc446680721)

[2.9.2使用例 17](#_Toc446680722)

[2.10 StdConvs.h 17](#_Toc446680723)

[2.10.1函数说明 17](#_Toc446680724)

[2.10.2使用例 18](#_Toc446680725)

[2.11 StrUtils.h 18](#_Toc446680726)

[2.11.1函数说明 18](#_Toc446680727)

[2.11.2使用例 20](#_Toc446680728)

[2.12 Synchronization.h 21](#_Toc446680729)

[2.12.1函数说明 21](#_Toc446680730)

[Synchronization类 21](#_Toc446680731)

[AutoLoad类 21](#_Toc446680732)

[2.12.2使用例 21](#_Toc446680733)

[2.13 TcpSocket.h 22](#_Toc446680734)

[2.13.1函数说明 22](#_Toc446680735)

[TcpSocket类 22](#_Toc446680736)

[TcpClient类 25](#_Toc446680737)

[2.13.2使用例 26](#_Toc446680738)

[2.14 Watcher.h 26](#_Toc446680739)

[2.14.1函数说明 26](#_Toc446680740)

[2.14.2使用例 27](#_Toc446680741)

[3.静态库的使用方法 27](#_Toc446680742)

[3.1 生成 27](#_Toc446680743)

[3.2 方法1（副本使用） 27](#_Toc446680744)

[3.3 方法2（直接使用） 28](#_Toc446680745)

[4.备注 28](#_Toc446680746)

## 1.概要

mtkPack中的类都包含于namespace mtkPack中。

从SVN下载代码：

<https://svn:18080/svn/repo/trunk/mtkPack>

## 2.类参考

### 2.1 Cabinet.h

操作.cab格式的压缩文件。支持压缩、解压和获取压缩文件列表操作。

#### 2.1.1函数说明

void SetFileName(

const *std*::*string*& sCabFile

);

设置.cab文件的完整名称。

*INT* GetFileList(

*std*::*vector*<*std*::*string*>& svFileList

);

获取.cab文件中包含的文件列表，结果写到svFileList中。返回值是文件数。

*INT* Expand(

const *std*::*string*& sWildName,

const *std*::*string*& sDest

);

解压缩.cab文件到指定目录。sWildName指定文件名（可包含通配符），sDest指定目标目录。

*INT* Compress(

const *std*::*string*& sWildName,

const *BOOL* bPreservePath = *TRUE*,

const *BOOL* bRecursive = *TRUE*

);

压缩指定的文件。sWildName指定文件名（可包含通配符），bPreservePath指定在压缩时是否需要保留目录；bRecursive指定是否遍历sWildName中指定的目录及所有子目录，并把所有符合该名称的文件全部压缩。

#### 2.1.2使用例

Cabinet cab1("D:\\temp\\test.cab");

*vector*<*string*> filelist;

cab1.GetFileList(filelist);

cab1.Expand("D:\\temp");

### 2.2 Codecs.h

抽象基类CodecBase，为编码、解码类定义基本接口。

virtual void SetData(

const *std*::*string* &sData

);

设置输入数据。

virtual *std*::*string* Encode() = 0;

virtual *std*::*string* Decode() = 0;

纯虚函数，由子类根据编码方案实现编码、解码。

#### 2.2.1函数说明

Base64类，实现base64方案的编码、解码。

virtual *std*::*string* Encode();

virtual *std*::*string* Decode();

编码、解码的结果通过返回值传递。

static *std*::*string* Encode(

const *std*::*string* &sData

);

static *std*::*string* Decode(

const *std*::*string* &sData

);

静态函数。

#### 2.2.2使用例

Base64 base64;

base64.SetData("YWJj");

*string* dec = base64.Decode();

dec.*clear*();

dec = Base64::Decode("YWJj");

dec = Base64::Encode("abc");

### 2.3 ComputerInfo.h

ComputerInfo类，获取计算机的软硬件信息。待扩充，当前只实现了CPUInfo类的部分功能。

#### 2.3.1 函数说明

CPUInfo类

long GetFreqMHz(void);

获取当前工作频率，单位是MHz。

const *std*::*string* ID;

CPU标识符。

const *std*::*string* Name;

CPU名称。

const *std*::*string* SpecFreq;

CPU的标称频率。

const unsigned int CoreNum;

CPU的核心数。

#### 2.3.2使用例

略。

### 2.4 Engine.h

为调用其他程序提供支持。

#### 2.4.1函数说明

*INT* SetAppCmdLine(

const *std*::*string*& sAppName,

const *std*::*string*& sCmdLine

);

设置执行程序的完整的名称和命令行参数。

*INT* SetAppName(

const *std*::*string*& sAppName

);

void SetCmdLine(

const *std*::*string*& sCmdLine

);

*INT* Exec(

*LPSTR* lpAdditionalParams = *NULL*,

EngineExecMode mode = EEM\_BLOCKING

);

开始执行程序。

可通过lpAdditionalParams指定额外的命令行参数。

可通过mode指定执行方式，阻塞和非阻塞两种方式

*BOOL* IsRunning(void);

#### 2.4.2使用例

略。

### 2.5 File.h

文件操作类，支持搜索、分割与合并。

#### 2.5.1函数说明

##### File类

静态：

static const int Exist(

const char\* cFilename

);

static const int Exist(

const wchar\_t\* wcFilename

);

检查文件是否存在，返回值等于1表示存在，0表示不存在。

static const int IsDirectory(

const char\* cFilename

);

static const int IsDirectory(

const wchar\_t\* wcFilename

);

检查指定的文件是否是目录。

static *std*::*string* ExtractFileExt(

const *std*::*string*& sFileName

);

返回文件名中的扩展名。

static *std*::*string* ExtractFileName(

const *std*::*string*& sFullName,

const bool bWithExt = true,

const bool bWithPath = false

);

从完整文件路径名中截取文件名的部分名称。

bWithExt指定是否需要包含扩展名。

bWithPath指定是否需要包含路径。

static *std*::*string* ExtractFilePath(

const *std*::*string*& sFullName

);

从完整文件路径名中截取文件所在的目录名。

static *std*::*string* TrName(

const *std*::*string*& sExpandableName

);

返回完整的文件路径名。

sExpandableName中可以包含以$开头的环境变量。详见使用例。

static *size\_t* Size(

const char\* cFilename

);

static *size\_t* Size(

const wchar\_t\* wcFilename

);

返回指定文件的大小。

static int Split(

const *std*::*string*& sFileName,

const SplitMode mode,

const *size\_t* nSizeOrCount

);

分割由sFileName指定的文件。

mode指定分割方式：

SM\_FILE\_SIZE表示按大小分割，此时nSizeOrCount指定每个文件的大小。

SM\_FILE\_COUNT表示按个数分割，此时nSizeOrCount指定文件分割后的个数。

static int Join(

const *std*::*string*& sFileName,

const *size\_t* nCnt,

const *std*::*string*& sNewFileName

);

合并被分割的文件。

sFileName指定要合并的第一个文件名。

nCnt指定合并文件的个数。

sNewFileName指定合并后的文件名。

static int Join(

const *std*::*string*& sFirstFileName,

const *std*::*string*& sLastFileName,

const *std*::*string*& sNewFileName

);

合并被分割的文件。

sFileName指定要合并的第一个文件名。

sLastFileName指定要合并的最后一个文件名。

sNewFileName指定合并后的文件名。

##### File::Iterator类

非静态：

File::Iterator，文件搜索类

void SetCompatibleMode(

const int bOnOff

);

设置与系统函数兼容的搜索方式。设置为1后，不支持通配符搜索，不支持搜索子目录。仅当确实有需要时才使用。

virtual void Find(

const *std*::*string*& sSearchDir,

const *std*::*string*& sFilePatterns

);

设置搜索条件。

sSearchDir指定要搜索的目录。

sFilePatterns指定要搜索的文件名集合。支持通配符和分割符，如："\*.txt;\*.ini|\*.cfg"

virtual int Next(void);

virtual int operator ++();

搜索文件。每次调用，返回一个符合条件的文件。当返回值等于0，表示没有符合条件的结果。详见使用例。

virtual int Close(void);

结束当前的搜索，通常用于下一次不同的搜索。即：除第一次调用Find外，以后每次调用Find函数以前，都需要先调用此函数来结束上一次搜索。

const *std*::*string* FilePath(void);

返回完整的文件名称。兼容方式设置为1时，只返回不包含目录的文件名。

const *std*::*string* FileName(void);

返回不包含目录的文件名。

const int IsDirectory(void);

检查当前搜索的结果是否是目录 。

#### 2.5.2使用例

*string* sNoExtName = File::ExtractFileName(sFileName, false, true);

*string* sExt = File::ExtractFileExt(sFileName);

// C:\Users\ADMINI~1\AppData\Local\Temp\~cab.tmp

*string* sTempfile = File::TrName("$TEMP\\~cab.tmp");

//把目标文件平均分割成N个小文件

*string* fn("D:\\temp\\ bigfile.dat ");

int filecnt = File::Split(fn, SM\_FILE\_COUNT, 8);

//把目标文件割成N个10MB的小文件

*string* fn("D:\\temp\\bigfile.dat");

int filecnt = File::Split(fn, SM\_FILE\_SIZE, 10 \* 1024 \* 1024);

//合并从 bigfile.dat.0 到 bigfile.dat.7 的文件

filecnt = File::Join(fn + ".0", fn + ".7", fn + ".dat");

//搜索指定目录中的所有jpg和png文件

File::Iterator fitr;

fitr.Find(File::TrName("$TEMP"), "\*.jpg;\*.png");

int filecnt = 0;

while(fitr.Next()) {

IniFile inif(fitr.FilePath());

filecnt ++;

}

//使用fitr进行二次搜索前，必须调用Close来结束前一次搜索

fitr.Close();

fitr.Find(File::TrName("$TEMP"), "\*.txt;\*.html");

filecnt = 0;

while(fitr.Next()) {

IniFile inif(fitr.FilePath());

filecnt ++;

}

### 2.6 IniFile.h

Ini文件管理类。除Load()和Store()函数会读写文件，其他函数都是对内存中数据进行读写。

#### 2.6.1函数说明

int Load(void);

int Load(

const *std*::*string*& sFileName

);

读取Ini文件，保存所有数据到内存。

对于构造时已指定文件名的实例对象，此函数会被自动调用 。

对于以无参构造建立的实例对象，使用sFileName指定文件名。

int Store(void);

把内存中的Ini数据写入文件，此函数在实例对象被析构时会被自动调用 。

void Clear(void);

清除已经载入内存的所有数据。

*std*::*string* & GetFileName(void);

取得已经载入的Ini文件名。

int GetInt(

const *std*::*string*& sSection,

const *std*::*string*& sKey,

const int iDefault = 0

);

取得Ini文件中指定的Section和Key对应的数据的整数值。

sSection指定Section名。

sKey指定Key名。

iDefault，当指定的Section或Key不存在时，返回该值。

*std*::*string* GetString(

const *std*::*string*& sSection,

const *std*::*string*& sKey,

const *std*::*string*& sDefault = ""

);

返回字符型，其他同GetInt函数。

void SetInt(

const *std*::*string*& sSection,

const *std*::*string*& sKey,

const int iVal

);

设置指定的Section和Key的整数值。

void SetString(

const *std*::*string*& sSection,

const *std*::*string*& sKey,

const *std*::*string*& sVal

);

设置指定的Section和Key的字符串值。

int GetGlobalInt(

const *std*::*string*& sKey,

const int iDefault = 0

);

取得全局Key的整数值。

iDefault,全局Key不存在时，返回此值。

*std*::*string* GetGlobalString(

const *std*::*string*& sKey,

const *std*::*string*& sDefault = ""

);

取得全局Key的字符串值。

void SetGlobalInt(

const *std*::*string*& sKey,

const int iVal

);

void SetGlobalString(

const *std*::*string*& sKey,

const *std*::*string*& sVal

);

设置全局Key的值。

int SectionExist(const *std*::*string*& sSection) const;

int IsSectionEmpty(const *std*::*string*& sSection) const;

检查指定的Section是否存在，是否为空（存在但不含有任何Key=Value）。

int RemoveKey(const *std*::*string*& sSection, const *std*::*string*& sKey);

int RemoveSection(const *std*::*string*& sSection);

删除指定的Key或Section。

int CommentKey(

const *std*::*string*& sSection,

const *std*::*string*& sKey

);

注释指定的Key。不删除该Key和Value，只在左侧增加分号，使该Key失效。如：Key1=Value1 注释后变成 ;Key1=Value1。此时程序无法找到该Key，但数据不会丢失。可以通过UncommentKey函数来恢复为有效的Key=Value。

int UncommentKey(

const *std*::*string*& sSection,

const *std*::*string*& sKey

);

取消对指定Key的注释，恢复为有效状态。

int CommentGlobalKey(const *std*::*string*& sKey);

int UncommentGlobalKey(const *std*::*string*& sKey);

注释、恢复全局的Key。

int GetSectionInfo(SectionInfo& siSectionInfo);

取得Section信息，可通过Iterator查找内容。

int GetSectionCount(const *std*::*string*& sSection);

取得Section总数。

const GlobalCommentMap & GetGlobalComment(void);

const GlobalDataStrMap & GetGlobalData(void);

const DataStrMap & GetData(void);

只读数据，便于类的内部调试。

#### 2.6.2使用例

IniFile inif("D:\\temp\\config.ini");

int regtries = inif.GetInt("DEFAULT", "regtries", -1);

if (regtries == -1) {

inif.SetInt("DEFAULT", "regtries", 3);

}

*string* appname = inif.GetString("DEFAULT", "AppName", "error");

*string* gvalue = inif.GetGlobalString("gkey","error");

### 2.7 MRUList.h

此为模板类，建立Most Recently Used列表。

#### 2.7.1函数说明

void Add(

const *T* &Val

)

增加一个T类型的元素到MRUList中。

void Resize(

const *size\_t* &nSize

)

重新调整MRUList的大小。

nSize，新的大小。如果小于设置前的大小，原列表中的内容会被删除。

void Remove(

const *T* &Val

)

删除一个指定的元素。

void Clear()

清空MRUList。

const *std*::*list*<*T*> &DataList()

取得当前的所有数据。

const *size\_t* MaxSize()

const *size\_t* Size()

取得MRUList的最大容量，当前大小。

const *T* &First()

const *T* &Last()

取得头、尾元素。

#### 2.7.2使用例

MRUList<*string*> mruFileList;

mruFileList.Add("abc.txt");

mruFileList.Add("xyz.txt");

mruFileList.Add("123.txt");

*list*<*string*>::*const\_iterator* mruItr = mruFileList.DataList().*begin*();

for (; mruItr != mruFileList.DataList().*end*(); mruItr++) {

*cout*<< \*mruItr << *endl*;

}

*cout*<<*endl*;

mruFileList.Add("xyz.txt");

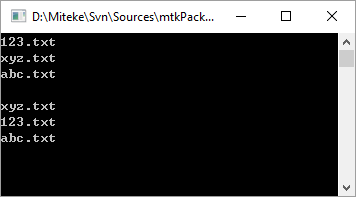
mruItr = mruFileList.DataList().*begin*();

for (; mruItr != mruFileList.DataList().*end*(); mruItr++) {

*cout*<< \*mruItr << *endl*;

}

输出



### 2.8 Process.h

进程操作，为调用外部程序提供基本支持。可作为其他引擎类型的成员（参照Engine.h）。

#### 2.8.1函数说明

*BOOL* Prepare(

const *CHAR*\* pAppNameA,

const *CHAR*\* pCmdLineA

);

*BOOL* Prepare(

const *WCHAR*\* pAppNameW,

const *WCHAR*\* pCmdLineW

);

输入外部程序名称及命令行参数，以*CREATE\_SUSPENDED*方式创建进程。

*DWORD* Run(void);

*BOOL* IsRunning(void) const;

*BOOL* IsCreated(void) const;

*DWORD* GetExitCode(void) const;

取得进程返回值。

*DWORD* GetPID(void) const;

取得进程ID。

*INT* GetPriority(void) const;

*BOOL* SetPriority(*INT* nPriority);

取得、设置进程优先级。

static *INT* StartAndWait(const *CHAR*\* pCmdLineA);

static *INT* StartAndWait(const *WCHAR*\* pCmdLineW);

静态成员，直接调用外部程序，由参数指定程序名和命令行参数。

#### 2.8.2使用例

Process myProc(APPNAME, CMDLINE);

myProc.Run();

### 2.9 SecureDelete.h

此为模板类，安全删除程序中动态声明的已经分配内存的指针对象。

#### 2.9.1函数说明

void Bind(

*T*\*\* pObjPtr,

const int bArrayFlg

)

pObjPtr，指定待释放的指针对象的地址（二次地址）。

bArrayFlg，指定该指针对象是否是数组类型。

注意：仅当使用无参构造函数时，才需要调用此函数。详见使用例。

#### 2.9.2使用例

以下两种方式中，仅当SecureDelete的实例对象被销毁时，与其绑定的指针对象才会被释放，同时，指针被置为NULL。

同时 ，对已被释放或已被设置为NULL的指针进行绑定时，不会由于二次删除导致错误。

//方式一，实例对象在构造的同时完成Bind

char \*pszDecoded = new(*std*::*nothrow*) char[cbSize + 1];

SecureDelete<char> sd1(&pszDecoded, 1);//指针对象以数组方式申请内存

//此时，pszDecoded指针对象并不会被释放，它在sd1被销毁前将一直有效

//方式二，通过无参构造函数创建实例对象后，显示调用Bind

Process\* myProc= new Process(APPNAME, CMDLINE);

SecureDelete<Process> autodel1;

autodel1.Bind(&myProc, 0); //指针对象未以数组方式申请内存

//此时，myProc指针对象也不会被释放，它在autodel1被销毁前也一直有效

### 2.10 StdConvs.h

数值转换类，待扩充。

#### 2.10.1函数说明

static double FahrenheitToCelsius(

const double& AValue

);

static double CelsiusToFahrenheit(

const double& AValue

);

#### 2.10.2使用例

略

### 2.11 StrUtils.h

字符串处理类。当前全部是静态函数。

#### 2.11.1函数说明

static char\* ToChar(

const wchar\_t\* pWChar

);

static wchar\_t\* ToWChar(

const char\* pChar

);

char与wchar\_t的相互转换。

函数内都申请了内存，需要在外部释放。可使用SecureDelete，详见使用例。

static *std*::*string* ToString(const wchar\_t\* pWChar);

static *std*::*string* ToString(const *std*::*wstring*& wsStr);

将wchar\_t和wstring转换为string。

函数内都申请并释放了内存，不需要维护内存。

static *std*::*wstring* ToWString(const char\* pChar);

static *std*::*wstring* ToWString(const *std*::*string*& sStr);

将char和string转换为wstring。

函数内都申请并释放了内存，不需要维护内存。

static void ToLower(char\* pStr, const int nSize = 0);

static char\* ToLower(const char\* pStr, const int nSize = 0);

static void ToLower(*std*::*string*& sStr);

static *std*::*string* ToLower(const *std*::*string*& sStr);

static void ToUpper(char\* pStr, const int nSize = 0);

static char\* ToUpper(const char\* pStr, const int nSize = 0);

static void ToUpper(*std*::*string*& sStr);

static *std*::*string* ToUpper(const *std*::*string*& sStr);

大小写转换函数。

static int SplitToken(

const *std*::*string*& sStr,

const *std*::*string*& sTokenSet,

*std*::*vector*<*std*::*string*>& sResult

);

字符串拆分。

sStr指定要拆分的字符串。

sTokenSet指定分割符集合。

sResult按顺序保存分割后的结果。

static int SplitToken(

const *std*::*string*& sStr,

const *std*::*string*& sTokenSet,

const char& cDelimiter,

*std*::*map*<*std*::*string*, *std*::*string*>& sResult

);

字符串拆分，并解析为Key和Value的配对。

sStr指定要拆分的字符串。

sTokenSet指定分割符集合。

cDelimiter指定Key和Value的分割字符。

sResult保存分割分析后的结果。

static *std*::*string*& LTrim(

*std*::*string*& sStr,

const *std*::*string*& sTrimSet = " "

);

static *std*::*string*& RTrim(

*std*::*string*& sStr,

const *std*::*string*& sTrimSet = " "

);

static *std*::*string*& Trim(

*std*::*string*& sStr,

const *std*::*string*& sTrimSet = " "

);

删除字符串首、尾的空格。

static int ToInt(const char\* pChar);

static int ToInt(const wchar\_t\* pWChar);

static int ToInt(const *std*::*string*& sStr);

static int ToInt(const *std*::*wstring*& wsStr);

将各字符串转换为整数值。

static *std*::*string* ToStr(const int nVal);

将整数值转换为string。

#### 2.11.2使用例

wchar\_t pWChar[] = L"abc";

char\* pChar = ToChar(pWChar);

SecureDelete<char> sd1(&pChar, 1);

//生成大小写交错的字符串AaBbCc…Zz

char strTable[52 + 1] = {0};

for (int i = 0; i < 52; i+=2) {

if (i == 0) {

strTable[0] = 'A';

strTable[1] = 'a';

} else {

strTable[i] = strTable[i-2] +1;

strTable[i+1] = strTable[i-1] + 1;

}

}

char sepTable1[27] = {0};//ABC…XYZ

char sepTable2[27] = {0};//abc…xyz

for(int i = 0; i < 26; i++){

sepTable1[i] = 'A' + i;

sepTable2[i] = 'a' + i;

}

*vector*<*string*> result1;

*vector*<*string*> result2;

//result1 = ['a'…'z']

int cnt1 = StrUtils::SplitToken(strTable, sepTable1, result1);

//result2 = ['A'…'Z']

int cnt2 = StrUtils::SplitToken(strTable, sepTable2, result2);

//解析参数：param1=val1&param2=val2&param3=val3

*map*<*string*, *string*> sParamList;

StrUtils::SplitToken(pData, "&", '=', sParamList);

### 2.12 Synchronization.h

同步操作，支持相关临界区或互斥对象，及自动加、解锁。

#### 2.12.1函数说明

##### Synchronization类

void SetType(

const SyncType stType

);

设置使用同步的种类：

SYNC\_CS：使用相关临界区

SYNC\_MUTEX：使用互斥对象

virtual void Lock(void);

virtual void Unlock(void);

加、解锁函数。

##### AutoLoad类

AutoLock(

Synchronization& pSync

);

指定同步对象，构造时执行加锁，析构时执行解锁。

#### 2.12.2使用例

extern Synchronization gSync;

{

AutoLock AL(gSync);

//需要保护的操作

}

### 2.13 TcpSocket.h

Socket通信类，抽象类TcpSocket派生TcpServer和TcpClient两个类。

#### 2.13.1函数说明

##### TcpSocket类

静态：

static *DWORD* IPv4StrToDWord(

const *std*::*string* &sIP

);

static *std*::*string* IPv4DWordToStr(

const *DWORD* &dwIP

);

DWORD型和string型IPv4地址的互相转换函数。

static *std*::*string* GetIPv4(

const *SOCKET* &socket, IPDirection ipDir

);

获取和socket连接的IPv4地址。

ipDir等于CLIENT\_IP时，获取客户端IP；

等于SERVER\_IP时，获取服务器IP。

static const bool IsValidMask(const *std*::*string*& sMask);

static const bool IsValidMask(const *DWORD*& dwMask);

检查指定的子网掩码是否有效。

TcpServer类

typedef int(\*RecvCallback)(

const TcpClient \*pClient

);

接收数据的回调函数。

typedef int(\*AcceptCallback)(

const TcpServer &Server,

const *SOCKET* &ClientSock

);

应答回调函数，返回1时，建立连接；返回0时，忽略请求，不建立连接。

typedef void(\*BanCallback)(

const *std*::*string* &sIPv4

);

屏蔽客户端回调函数。如果客户端IP在黑名单中，则屏蔽该客户端，并触发此回调函数。

int Setup(

const *std*::*string* &sHost,

const int iPort,

AcceptCallback pAcceptCallback,

RecvCallback pRecvCallback,

BanCallback pBanCallback

);

设置Server端的必要参数。

sHost指定主机地址。iPort指定端口号。

pAcceptCallBack，指定应答回调函数。

注意：仅当此回调函数返回1时，才能成功建立与客户端的连接。

pRecvCallback，指定接收数据回调函数。

pBanCallback，指定屏蔽客户端回调函数。

virtual int Start(void);

virtual int Stop(void);

启动，关闭Server。

virtual const *std*::*string* GetIPv4(void);

获取Server的IPv4地址。

const TcpServerParam & GetParam(void)

获取Server参数数据。

void SetMaxClients(

*size\_t* nMaxClients

);

设置客户端的最大同时连接数量。

nMaxClients，指定数量，取值范围从10～10000。默认值等于10。

void SetClientTimeoutSecond(

unsigned long nSecond

);

设置客户端超时时间。超时后，Server会主动关闭与该客户端的连接。

nSecond指定超时秒数。默认等于0，即Server端从不主动断开连接。

int AddToIPList(

IP\_TYPE type,

const *std*::*string*& sIPv4

);

增加IP地址到黑、白名单。

type等于WHITE\_IP时，对白名单操作；等于BLACK\_IP时，对黑名单操作。

int DeleteFromIPList(

IP\_TYPE type,

const *std*::*string*& sIPv4

);

从黑、白名单中删除IP地址。

type等于WHITE\_IP时，对白名单操作；等于BLACK\_IP时，对黑名单操作。

int AddToIPMap(

IP\_TYPE type,

const *std*::*string*& sIPv4,

const *std*::*string*& sMask

);

int DeleteFromIPMap(

IP\_TYPE type,

const *std*::*string*& sIPv4,

const *std*::*string*& sMask

);

以IP地址加子网掩码组合的方式，增加、删除黑白名单。

注意：此二函数尚未实现，需要计算各个子网的起、止地址。

const bool IsIPBanned(

const *std*::*string*& sIPv4

);

检查指定IP是否被Server端屏蔽。

int AddToMsgRuleList(

const *std*::*string*& sMsg

);

增加指定字符串到规则列表。

const bool CheckMsgByRules(

const *std*::*string*& sMsg

);

检查指定的字符串是否符合现有的规则。

可以根据结果，增加IP到黑、白名单中；或从名单中删除IP。

##### TcpClient类

void Setup(

const *std*::*string* &sRemoteHost,

const int iRemotePort,

RecvCallback pRecvCallback

);

设置客户端的连接参数。

sRemoteHost指定Server的IP。

iRemotePort指定Server的端口。

pRecvCallback指定接收数据回调函数。

bool HasReceivedData(void) const;

检查是否收到数据。

const char\* GetData(void);

获取已收到的数据。

virtual int Start(void);

virtual int Stop(void);

启动、停止客户端。

virtual const *std*::*string* GetIPv4(void);

获取客户端IP。

virtual int Send(

const char \*pData

);

发送数据。

const TcpClientParam & GetParam(void)

获取客户端参数数据。

void SetClosingKeyword(

const *std*::*string* &sKeyword

);

设置自动关闭连接的关键字。当Server端收到此关键字后，会断开与客户端的连接。

bool IsValid(

const ClientChkMask& chkmask

);

检查客户端对象是否仍然有效。

chkmask的取值

CHK\_STOPEVENT,检查停止的Event是否有效（内部使用）；

CHK\_SELECTEVENT, 检查Select的Event是否有效（内部使用）；

CHK\_SOCKET, 检查自身Socket是否有效；

CHK\_DATA,检查接收数据缓冲区是否有效；

CHK\_ALL，检查以上所有内容。

#### 2.13.2使用例

A.参考测试工程，Test\_TcpServer和Test\_TcpClient。

B.参考IFServer工程（查找myServer，及其各回调函数）。

### 2.14 Watcher.h

监视类，以线程方式定时处理任务。使用互斥对象（每个实例使用一个随机的互斥对象）保证线程安全。具体操作写在回调函数中，在设置的间隔自动被调用。

#### 2.14.1函数说明

void(\*WatcherCallback)(void\* pParam);

回调函数，通过pParam指定传递所需参数。

WatcherInfo(

unsigned int nInterval,

WatcherCallback cbOnWatch,

void\* lpParam

);

构造一个WatcherInfo参数，传递给Watcher类的构造函数或Setup函数。

void Setup(

const WatcherInfo& Info

);

通过无参构造创建Watcher时，必须调用Setup传递参数，才能正确运行。

void Start(void);

启动Watcher。按照WatcherInfo.nInterval的间隔，执行定时任务。

void Stop(void);

停止Watcher。结束定时任务。正在执行的定时任务，不会被打断，只会在执行完成后停止。

#### 2.14.2使用例

//回调函数

void OnWatch(void\* pParam) {

//定时任务

}

//每一秒执行一次

WatcherInfo wi(1000, OnWatch, *NULL*);

Watcher myWatcher;

myWatcher.Setup(wi);

myWatcher.Start();

Sleep(1000 \* 3);

myWatcher.Stop();

## 3.静态库的使用方法

### 3.1 生成

成功编译mtkPack后，运行make\_DevPack.cmd，在DevPack目录内生成.h文件和.lib文件。

### 3.2 方法1（副本使用）

复制DevPack目录到需要使用的工程中，在工程的主要.h文件中增加如下代码后，即可使用。

#include "..\mtkPack\include\mtkPack.h"

#ifdef \_DEBUG

#pragma comment(lib, "..\\mtkPack\\lib\\debug\\mtkPack.lib")

#else

#pragma comment(lib, "..\\mtkPack\\lib\\release\\mtkPack.lib")

#endif

注意：相对文件名需要根据实际情况修改。

特点：与mtkPack工程无耦合，互不影响。适用于，使用稳定版的mtkPack开发新工程。

### 3.3 方法2（直接使用）

不必复制DevPack目录，直接在工程的主要.h文件中增加如方法1中的码。

注意：相对文件名需要设置为mtkPack工程的相应目录。

特点：直接使用并依赖于mtkPack工程。适用于，开发新工程的同时，需要调试修改mtkPack。

## 4.备注

无