m\_mapKeyword: KeyID, SourceID, CKeyEntry

CKeyEntry:keyID,sourceID,LifeTime,AICH id,CKadTag

CKadTag:m\_name,m\_type

m\_mapSources: KeyID,SourceID,CEntry

CEntry: LiftTime ,CKadTag,KeyID,SourceID

搜索过程中使用到的Map

typedef std::map<CUInt128, CSearch\*> SearchMap;

SearchMap m\_mapSearches

///snow:在StartSearch、PrepareFindKeywords、PrepareLookup中赋值

typedef std::map<CUInt128, CContact\*> ContactMap;

ContactMap m\_mapPossible;

ContactMap m\_mapTried;

ContactMap m\_mapBest;

ContactMap m\_mapInUse;

std::map<Kademlia::CUInt128, bool> m\_mapResponded;

ContactList m\_listDelete;

问题1、文件名的关键字是怎么提取出来的？

2、KeywordID与SourceID是怎样分工的？ Sharefile中是否有相关信息？

3、Index中存储的关键字Map是怎样的？见PPT

4、Search的过程是？SendFindValue与StorePacket的分工

# Load\_index.dat的作用

变量：CSearch::m\_uTotolLoad 、CSearch::m\_uTotalLoadResponses

m\_mapLoads

AddSources()对uLoad的更新好象没起作用

当我们有源文件需要发布时，发出KADEMILA2\_PUBLISH\_SOURCE\_REQ信息包

对方接收到该信息包，调用Process\_KADEMLIA2\_PUBLISH\_SOURCE\_REQ过程进行处理，过程再调用pIndexed->AddSources()函数进行源的添加，在添加时通过与本机存储的Indexed数据进行比对，计算出uLoad值（新的源文件时uLoad=1，其它情况uLoad=EnrtySize\*100/1000，最大值100）

if(!m\_mapSources.Lookup(CCKey(uKeyID.GetData()), pCurrSrcHash))

{

uLoad = 1;

}

else ///snow:已存在同ID的Source条目

{

uint32 uSize = pCurrSrcHash->ptrlistSource.GetSize();

for(POSITION pos1 = pCurrSrcHash->ptrlistSource.GetHeadPosition(); pos1 != NULL; )

{

Source\* pCurrSource = pCurrSrcHash->ptrlistSource.GetNext(pos1);

if( pCurrSource->ptrlEntryList.GetSize() ) ///snow:ptrlEntryList中的Entry条目不为

{

CEntry\* pCurrEntry = pCurrSource->ptrlEntryList.GetHead();

ASSERT(pCurrEntry!=NULL);

if( pCurrEntry->m\_uIP == pEntry->m\_uIP && ( pCurrEntry->m\_uTCPPort == pEntry->m\_uTCPPort || pCurrEntry->m\_uUDPPort == pEntry->m\_uUDPPort )) ///snow:IP、Port或UDPPort相一致

{

delete pCurrSource->ptrlEntryList.RemoveHead();

pCurrSource->ptrlEntryList.AddHead(pEntry);

uLoad = (uint8)((uSize\*100)/KADEMLIAMAXSOUCEPERFILE); ///snow:每文件最多个源，uLoad=该文件源数\*0.1

return true;

}

}

}

if( uSize > KADEMLIAMAXSOUCEPERFILE )

{

Source\* pCurrSource = pCurrSrcHash->ptrlistSource.RemoveTail();

delete pCurrSource->ptrlEntryList.RemoveTail();

pCurrSource->uSourceID.SetValue(uSourceID);

pCurrSource->ptrlEntryList.AddHead(pEntry);

pCurrSrcHash->ptrlistSource.AddHead(pCurrSource);

uLoad = 100; ///snow:每文件最多个源，该文件源数>1000，uLoad=100（最大值）

return true;

}

else

{

Source\* pCurrSource = new Source;

pCurrSource->uSourceID.SetValue(uSourceID);

pCurrSource->ptrlEntryList.AddHead(pEntry);

pCurrSrcHash->ptrlistSource.AddHead(pCurrSource);

m\_uTotalIndexSource++;

uLoad = (uint8)((uSize\*100)/KADEMLIAMAXSOUCEPERFILE);

return true;

}

}

1、CSharedFileList::Publish()发起发布请求

void CSharedFileList::Publish()

{

if( Kademlia::CKademlia::IsConnected() && ( !isFirewalled || ( isFirewalled && theApp.clientlist->GetBuddyStatus() == Connected) || bDirectCallback) && GetCount() && Kademlia::CKademlia::GetPublish())

{

//We are connected to Kad. We are either open or have a buddy. And Kad is ready to start publishing.

if( Kademlia::CKademlia::GetTotalStoreKey() < KADEMLIATOTALSTOREKEY)///snow:为什么不超过两个hash?这里的totalStoreKey的值指的是m\_mapSearches中的各searchtype的统计数，CSearchManager::UpdateStats()调用SetTotalStoreKey(uTotalStoreKey)更新m\_uTotalStoreKey

///snow:这个判断条件的意思是当搜索列表中存在一个拟发布Keyword的搜索对象时，不再处理跟发布keyword有关的信息，必须等上个发布处理完毕

{

//We are not at the max simultaneous keyword publishes

if (tNow >= m\_keywords->GetNextPublishTime())

{

//Enough time has passed since last keyword publish

//Get the next keyword which has to be (re)-published

CPublishKeyword\* pPubKw = m\_keywords->GetNextKeyword();

if(pPubKw)

{

//We have the next keyword to check if it can be published

//Debug check to make sure things are going well.

ASSERT( pPubKw->GetRefCount() != 0 );///snow:pPubKw对象中m\_aFiles数组存储的条目数

if (tNow >= pPubKw->GetNextPublishTime())

{

//This keyword can be published.

///snow:PrepareLookup首先以传递进去的参数uType和uID构造一个CSearch对象，再检查m\_mapLoad中是否已有该keyword，如果没找到，则返回该对象；否则返回NULL

Kademlia::CSearch\* pSearch = Kademlia::CSearchManager::PrepareLookup(Kademlia::CSearch::STOREKEYWORD, false, pPubKw->GetKadID());

if (pSearch) ///snow:表示m\_mapLoad中没有该Keyword

{…

Kademlia::CSearchManager::StartSearch(pSearch);

}

…

}

2、搜索四部曲：go()🡪SendFindValue()🡪JumpStart()🡪StorePacket()

void CSearch::StorePacket()

{

case STOREKEYWORD:

{

// Try to store keywords to a Node.

// As a safe guard, check to see if we already stored to the Max Nodes

if( m\_uAnswers > SEARCHSTOREKEYWORD\_TOTAL ) ///snow:最多个

{

PrepareToStop();

break;

}

uint16 iCount = (uint16)m\_listFileIDs.size(); ///snow:AddFileID()中赋值

if(iCount == 0)

{

PrepareToStop();

break;

}

else if(iCount > 150) ///snow:最多个文件

iCount = 150;

UIntList::const\_iterator itListFileID = m\_listFileIDs.begin();

uchar ucharFileid[16];

///snow:在这里写入的数据在Process\_KADEMLIA2\_PUBLISH\_KEY\_REQ()中读取处理

while(iCount && (itListFileID != m\_listFileIDs.end())) ///snow:遍历m\_listFileIDs，每个文件一个信息包

{

uint16 iPacketCount = 0;

byte byPacket[1024\*50];

CByteIO byIO(byPacket,sizeof(byPacket));

byIO.WriteUInt128(m\_uTarget); ///snow:拟发送的Keyword，在PrepareLookup中通过参数传入

byIO.WriteUInt16(0); // Will be corrected before sending. ///snow:文件ID数，在后面更新

while((iPacketCount < 50) && (itListFileID != m\_listFileIDs.end())) ///snow:添加个文件ID及Tags

{

CUInt128 iID = \*itListFileID;

iID.ToByteArray(ucharFileid);

CKnownFile\* pFile = theApp.sharedfiles->GetFileByID(ucharFileid);

if(pFile)

{

iCount--;

iPacketCount++;

byIO.WriteUInt128(iID);

PreparePacketForTags( &byIO, pFile, pFromContact->GetVersion() ); ///snow:给每个文件添加Tags

}

++itListFileID;

}

// Correct file count.

uint32 current\_pos = byIO.GetUsed();

byIO.Seek(16);

byIO.WriteUInt16(iPacketCount); ///snow:以实际文件ID数更新之前写入的

byIO.Seek(current\_pos);

// Send packet

if (pFromContact->GetVersion() >= 6){ /\*48b\*/

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

DebugSend("KADEMLIA2\_PUBLISH\_KEY\_REQ", pFromContact->GetIPAddress(), pFromContact->GetUDPPort());

CUInt128 uClientID = pFromContact->GetClientID();

CKademlia::GetUDPListener()->SendPacket( byPacket, sizeof(byPacket)-byIO.GetAvailable(), KADEMLIA2\_PUBLISH\_KEY\_REQ, pFromContact->GetIPAddress(), pFromContact->GetUDPPort(), pFromContact->GetUDPKey(), &uClientID);

}

else if (pFromContact->GetVersion() >= 2/\*47a\*/)

{

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

DebugSend("KADEMLIA2\_PUBLISH\_KEY\_REQ", pFromContact->GetIPAddress(), pFromContact->GetUDPPort());

CKademlia::GetUDPListener()->SendPacket( byPacket, sizeof(byPacket)-byIO.GetAvailable(), KADEMLIA2\_PUBLISH\_KEY\_REQ, pFromContact->GetIPAddress(), pFromContact->GetUDPPort(), 0, NULL);

ASSERT( CKadUDPKey(0) == pFromContact->GetUDPKey() );

}

else

ASSERT( false );

}

// Inc total request answers

m\_uTotalRequestAnswers++;

// Update search in the GUI

theApp.emuledlg->kademliawnd->searchList->SearchRef(this);

break;

}

…

}

3、对方联系人收到发布请求，进行处理：

void CKademliaUDPListener::Process\_KADEMLIA2\_PUBLISH\_KEY\_REQ (const byte \*pbyPacketData, uint32 uLenPacket, uint32 uIP, uint16 uUDPPort, CKadUDPKey senderUDPKey)

{

//Used Pointers

CIndexed \*pIndexed = CKademlia::GetIndexed();

// check if we are UDP firewalled

if (CUDPFirewallTester::IsFirewalledUDP(true))

{

//We are firewalled. We should not index this entry and give publisher a false report.

return;

}

CByteIO byteIO(pbyPacketData, uLenPacket);

CUInt128 uFile;

byteIO.ReadUInt128(&uFile);

CUInt128 uDistance(CKademlia::GetPrefs()->GetKadID());

uDistance.Xor(uFile);

// Shouldn't LAN IPs already be filtered?

///snow:如果距离超过容忍值，则放弃

if(uDistance.Get32BitChunk(0) > SEARCHTOLERANCE && !::IsLANIP(ntohl(uIP)))

return;

bool bDbgInfo = (thePrefs.GetDebugClientKadUDPLevel() > 0);

CString sInfo;

uint16 uCount = byteIO.ReadUInt16();

uint8 uLoad = 0;

CUInt128 uTarget;

while( uCount > 0 )

{

sInfo.Empty();

byteIO.ReadUInt128(&uTarget);

CKeyEntry\* pEntry = new Kademlia::CKeyEntry();

try

{

pEntry->m\_uIP = uIP;

pEntry->m\_uUDPPort = uUDPPort;

pEntry->m\_uKeyID.SetValue(uFile);

pEntry->m\_uSourceID.SetValue(uTarget);

pEntry->m\_tLifetime = (uint32)time(NULL)+KADEMLIAREPUBLISHTIMEK;

pEntry->m\_bSource = false;

uint32 uTags = byteIO.ReadByte();

while(uTags > 0)

{

CKadTag\* pTag = byteIO.ReadTag();

if(pTag)

{

if (!pTag->m\_name.Compare(TAG\_FILENAME))

{

if ( pEntry->GetCommonFileName().IsEmpty() )

{

pEntry->SetFileName(pTag->GetStr());

if (bDbgInfo)

sInfo.AppendFormat(\_T(" Name=\"%ls\""), pEntry->GetCommonFileName());

}

delete pTag; // tag is no longer stored, but membervar is used

}

else if (!pTag->m\_name.Compare(TAG\_FILESIZE))

{

if( pEntry->m\_uSize == 0 )

{

if(pTag->IsBsob() && pTag->GetBsobSize() == 8)

{

pEntry->m\_uSize = \*((uint64\*)pTag->GetBsob());

}

else

pEntry->m\_uSize = pTag->GetInt();

if (bDbgInfo)

sInfo.AppendFormat(\_T(" Size=%u"), pEntry->m\_uSize);

}

delete pTag; // tag is no longer stored, but membervar is used

}

else if (!pTag->m\_name.Compare(TAG\_KADAICHHASHPUB))

{

if(pTag->IsBsob() && pTag->GetBsobSize() == CAICHHash::GetHashSize())

{

if (pEntry->GetAICHHashCount() == 0)

{

pEntry->AddRemoveAICHHash(CAICHHash((uchar\*)pTag->GetBsob()), true);

if (bDbgInfo)

sInfo.AppendFormat(\_T(" AICH Hash=%s"), CAICHHash((uchar\*)pTag->GetBsob()).GetString());

}

else

DebugLogWarning(\_T("Multiple TAG\_KADAICHHASHPUB tags received for single file from %s"), ipstr(ntohl(uIP)));

}

else

DEBUG\_ONLY( DebugLogWarning(\_T("Bad TAG\_KADAICHHASHPUB received from %s"), ipstr(ntohl(uIP))) );

delete pTag;

}

else

{

//TODO: Filter tags - we do some basic filtering already within this function, might want to do more at some point

pEntry->AddTag(pTag);

}

}

uTags--;

}

if (bDbgInfo && !sInfo.IsEmpty())

Debug(\_T("%s\n"), sInfo);

}

catch(...)

{

delete pEntry;

throw;

}

if( !pIndexed->AddKeyword(uFile, uTarget, pEntry, uLoad) ) ///snow:这里也返回了uLoad

{

//We already indexed the maximum number of keywords.

//We do not index anymore but we still send a success..

//Reason: Because if a VERY busy node tells the publisher it failed,

//this busy node will spread to all the surrounding nodes causing popular

//keywords to be stored on MANY nodes..

//So, once we are full, we will periodically clean our list until we can

//begin storing again..

delete pEntry;

pEntry = NULL;

}

uCount--;

}

CSafeMemFile fileIO2(17);

fileIO2.WriteUInt128(&uFile);

fileIO2.WriteUInt8(uLoad);

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

DebugSend("KADEMLIA2\_PUBLISH\_RES", uIP, uUDPPort);

SendPacket( &fileIO2, KADEMLIA2\_PUBLISH\_RES, uIP, uUDPPort, senderUDPKey, NULL);

}

4、调用AddKeyword，计算uLoad值，并回传

bool CIndexed::AddKeyword(const CUInt128& uKeyID, const CUInt128& uSourceID, Kademlia::CKeyEntry\* pEntry, uint8& uLoad, bool bIgnoreThreadLock)

{

// do not access any data while the loading thread is busy;

// bIgnoreThreadLock should be only used by CLoadDataThread itself

if (!bIgnoreThreadLock && !m\_bDataLoaded) {

DEBUG\_ONLY( DebugLogWarning(\_T("CIndexed Memberfunction call failed because the dataloading still in progress")) );

return false;

}

if( !pEntry )

return false;

if (!pEntry->IsKeyEntry()){

ASSERT( false );

return false;

}

if( m\_uTotalIndexKeyword > KADEMLIAMAXENTRIES ) ///snow:本机存储的关键字超过

{

uLoad = 100;

return false;

}

if( pEntry->m\_uSize == 0 || pEntry->GetCommonFileName().IsEmpty() || pEntry->GetTagCount() == 0 || pEntry->m\_tLifetime < time(NULL))

return false;

KeyHash\* pCurrKeyHash;

if(!m\_mapKeyword.Lookup(CCKey(uKeyID.GetData()), pCurrKeyHash)) ///snow:map中未发现相同key的条目

{

Source\* pCurrSource = new Source;

pCurrSource->uSourceID.SetValue(uSourceID);

pEntry->MergeIPsAndFilenames(NULL); //IpTracking init

pCurrSource->ptrlEntryList.AddHead(pEntry);

pCurrKeyHash = new KeyHash;

pCurrKeyHash->uKeyID.SetValue(uKeyID);

pCurrKeyHash->mapSource.SetAt(CCKey(pCurrSource->uSourceID.GetData()), pCurrSource);

m\_mapKeyword.SetAt(CCKey(pCurrKeyHash->uKeyID.GetData()), pCurrKeyHash);

uLoad = 1;

m\_uTotalIndexKeyword++;

return true;

}

else ///snow:已存在同一ID的Key条目

{

uint32 uIndexTotal = pCurrKeyHash->mapSource.GetCount();

if ( uIndexTotal > KADEMLIAMAXINDEX ) ///snow:当前关键字下存储的文件源超过

{

uLoad = 100;

//Too many entries for this Keyword..

return false;

}

Source\* pCurrSource;

if(pCurrKeyHash->mapSource.Lookup(CCKey(uSourceID.GetData()), pCurrSource)) ///snow:key中是否存在同一SourceID的条目

{

if (pCurrSource->ptrlEntryList.GetCount() > 0)

{

if( uIndexTotal > KADEMLIAMAXINDEX - 5000 ) ///snow:大于

{

uLoad = 100;

//We are in a hot node.. If we continued to update all the publishes

//while this index is full, popular files will be the only thing you index.

return false;

}

// also check for size match

CKeyEntry\* pOldEntry = NULL;

for (POSITION pos = pCurrSource->ptrlEntryList.GetHeadPosition(); pos != NULL; pCurrSource->ptrlEntryList.GetNext(pos)){

CKeyEntry\* pCurEntry = (CKeyEntry\*)pCurrSource->ptrlEntryList.GetAt(pos);

ASSERT( pCurEntry->IsKeyEntry() );

if (pCurEntry->m\_uSize == pEntry->m\_uSize){

pOldEntry = pCurEntry;

pCurrSource->ptrlEntryList.RemoveAt(pos);

break;

}

}

pEntry->MergeIPsAndFilenames(pOldEntry); // pOldEntry can be NULL, thats ok and we still need todo this call in this case

if (pOldEntry == NULL){

m\_uTotalIndexKeyword++;

DebugLogWarning(\_T("Kad: Indexing: Keywords: Multiple sizes published for file %s"), pEntry->m\_uSourceID.ToHexString());

}

DEBUG\_ONLY( AddDebugLogLine(DLP\_VERYLOW, false, \_T("Indexed file %s"), pEntry->m\_uSourceID.ToHexString()) );

delete pOldEntry;

pOldEntry = NULL;

}

else{

m\_uTotalIndexKeyword++;

pEntry->MergeIPsAndFilenames(NULL); //IpTracking init

}

uLoad = (uint8)((uIndexTotal\*100)/KADEMLIAMAXINDEX);

pCurrSource->ptrlEntryList.AddHead(pEntry);

return true;

}

else ///snow:不存在与SourceID相同的条目

{

pCurrSource = new Source;

pCurrSource->uSourceID.SetValue(uSourceID);

pEntry->MergeIPsAndFilenames(NULL); //IpTracking init

pCurrSource->ptrlEntryList.AddHead(pEntry);

pCurrKeyHash->mapSource.SetAt(CCKey(pCurrSource->uSourceID.GetData()), pCurrSource);

m\_uTotalIndexKeyword++;

uLoad = (uint8)((uIndexTotal\*100)/KADEMLIAMAXINDEX);

return true;

}

}

}

5、并发回KADEMLIA2\_PUBLISH\_RES 信息包

SendPacket( &fileIO2, KADEMLIA2\_PUBLISH\_RES, uIP, uUDPPort, senderUDPKey, NULL);

6、本机在接收到该信息包时，调用Process\_KADEMLIA2\_PUBLISH\_RES,读出uLoad值，并调用CSearchManager:: ProcessPublishResult(uFile, uLoad, true);传递uLoad值给ProcessPublishResult()进行处理。

void CSearchManager::ProcessPublishResult(const CUInt128 &uTarget, const uint8 uLoad, const bool bLoadResponse)

{

// We tried to publish some info and got a result.

CSearch \*pSearch = NULL;

SearchMap::const\_iterator itSearchMap = m\_mapSearches.find(uTarget);

if (itSearchMap != m\_mapSearches.end())

pSearch = itSearchMap->second;

// Result could be very late and store deleted, abort.

if (pSearch == NULL)

return;

switch(pSearch->GetSearchTypes())

{

case CSearch::STOREKEYWORD:

if( bLoadResponse )

pSearch->UpdateNodeLoad( uLoad ); ///snow:uLoad的值被添加到m\_uTotalLoad，m\_uTotalLoadResponses++

break;

case CSearch::STOREFILE:

case CSearch::STORENOTES:

break;

}

…

}

void CSearch::UpdateNodeLoad( uint8 uLoad )

{

// Since all nodes do not return a load value, keep track of total responses and total load.

m\_uTotalLoad += uLoad;

m\_uTotalLoadResponses++;

}

///snow:取加载的Load节点与有回应的Load节点的比值

uint32 CSearch::GetNodeLoad() const

{

// Node load is the average of all node load responses.

if( m\_uTotalLoadResponses == 0 )

{

return 0;

}

return m\_uTotalLoad/m\_uTotalLoadResponses;

}

在CSearch对象进行析构的时候，如果m\_uTotalLoad/m\_uTotalLoadResponses>20的时候，将该对象添加到m\_mapLoad

CSearch::~CSearch()

{

// Check if this search was contacting a overload node and adjust time of next time we use that node.

if(CKademlia::IsRunning() && GetNodeLoad() > 20)

{

switch(GetSearchTypes())

{

case CSearch::STOREKEYWORD:

Kademlia::CKademlia::GetIndexed()->AddLoad(GetTarget(), ((uint32)(DAY2S(7)\*((double)GetNodeLoad()/100.0))+(uint32)time(NULL)));

break;

}

}

CIndexed对象在析构的时候，将m\_mapLoads中的信息写入Load\_index.dat

CIndexed对象在初始化的时候，将Load\_index.dat读入m\_mapLoads

(CIndexed只有一个对象)

int CIndexed::CLoadDataThread::Run()

{

…

if (!m\_pOwner->m\_bAbortLoading)

{

///snow:加载load\_index.dat，示例：00 00 00 09 35 CA 58 01 00 00 00 17 7E D7 1A 51 A4 6C 77 CB 28 15 65 F9 B8 89 EA 2E 41 CB 58

CBufferedFileIO fileLoad;

if(fileLoad.Open(m\_sLoadFileName, CFile::modeRead | CFile::typeBinary | CFile::shareDenyWrite))

{

setvbuf(fileLoad.m\_pStream, NULL, \_IOFBF, 32768);

uint32 uVersion = fileLoad.ReadUInt32(); ///snow:前四个字节是版本号 01 00 00 00

if(uVersion<2) ///snow:版本号小于，只能是

{

/\*time\_t tSaveTime = \*/fileLoad.ReadUInt32(); ///snow:保存时间09 35 CA 58

uint32 uNumLoad = fileLoad.ReadUInt32(); ///snow:条目数 01 00 00 00

while(uNumLoad && !m\_pOwner->m\_bAbortLoading)

{

fileLoad.ReadUInt128(&uKeyID); ///snow:16个字节uKeyID:17 7E D7 1A 51 A4 6C 77 CB 28 15 65 F9 B8 89 EA

if(m\_pOwner->AddLoad(uKeyID, fileLoad.ReadUInt32(), true)) ///snow:4字节的加入时间 2E 41 CB 58

uTotalLoad++;

uNumLoad--;

}

}

fileLoad.Close();

}

else

DebugLogWarning(\_T("Unable to load Kad file: %s"), m\_sLoadFileName);

}

…

}

# 共享文件处理过程

## 一、共享文件的发现与添加

1、CemuleAPP对象的成员变量theApp.sharedfiles在CemuleApp::InitInstance()函数中构造CSharedFileList对象，调用CSharedFileList构造函数

sharedfiles = new CSharedFileList(serverconnect);

2、CSharedFileList::CSharedFileList(CServerConnect\* in\_server)

{

…

LoadSingleSharedFilesList();

FindSharedFiles();

}

3、void CSharedFileList::LoadSingleSharedFilesList()

{

CString strFullPath = thePrefs.GetMuleDirectory(EMULE\_CONFIGDIR) + SHAREDFILES\_FILE; ///sharedfiles.dat

CStdioFile\* sdirfile = new CStdioFile();

bool bIsUnicodeFile = IsUnicodeFile(strFullPath); // check for BOM

if (sdirfile->Open(strFullPath, CFile::modeRead | CFile::shareDenyWrite | (bIsUnicodeFile ? CFile::typeBinary : 0)))

{

…

AddSingleSharedFile(toadd, true);

…

}

3.1、bool CSharedFileList::AddSingleSharedFile(const CString& rstrFilePath, bool bNoUpdate)

{

…

return bNoUpdate || CheckAndAddSingleFile(rstrFilePath);

}

4、FindSharedFiles()从各个共享目录添加文件

void CSharedFileList::FindSharedFiles()

{

tempDir = thePrefs.GetMuleDirectory(EMULE\_INCOMINGDIR);

AddFilesFromDirectory(tempDir);

///snow:根据选项中设置的下载分类目录，逐个目录将文件添加到m\_Files\_map

for (int ix=1;ix<thePrefs.GetCatCount();ix++)

{

tempDir=CString( thePrefs.GetCatPath(ix) );

…

AddFilesFromDirectory(tempDir);

}

}

///snow:根据选项中设置的共享目录，逐个目录将文件添加到m\_Files\_map

for (POSITION pos = thePrefs.shareddir\_list.GetHeadPosition();pos != 0;)

{

tempDir = thePrefs.shareddir\_list.GetNext(pos);

…

AddFilesFromDirectory(tempDir);

}

}

// add all single shared files

///snow:添加单个共享文件

for (POSITION pos = m\_liSingleSharedFiles.GetHeadPosition(); pos != NULL; m\_liSingleSharedFiles.GetNext(pos))

CheckAndAddSingleFile(m\_liSingleSharedFiles.GetAt(pos));

…

}

4.1、AddFilesFromDirectory()遍历目录，逐个添加文件

void CSharedFileList::AddFilesFromDirectory(const CString& rstrDirectory)

{

CFileFind ff;

CString strSearchPath(rstrDirectory);

…

///snow:遍历目录，添加目录中的文件到m\_Files\_map

while (!end)

{

end = !ff.FindNextFile();

CheckAndAddSingleFile(ff);

}

ff.Close();

}

5、

void CSharedFileList::CheckAndAddSingleFile(const CFileFind& ff){

…

///snow:在CKnownFileList::m\_Files\_map中查找是否已存在该文件，非本类m\_Files\_map

CKnownFile\* toadd = theApp.knownfiles->FindKnownFile(strFoundFileName, fdate, ullFoundFileSize);

if (toadd)

{

CCKey key(toadd->GetFileHash()); ///snow:return toadd.m\_FileIdentifier.m\_abyMD4Hash

CKnownFile\* pFileInMap;

if (m\_Files\_map.Lookup(key, pFileInMap))

{

File already in shared file list

}

///snow:本类的m\_Files\_map中还没有该文件的条目

else

{

toadd->SetPath(strFoundDirectory);

toadd->SetFilePath(strFoundFilePath);

toadd->SetSharedDirectory(strShellLinkDir);

AddFile(toadd);

}

}

///snow:不在knownfilelist中，添加到waitingforhash\_list，等待Hashing

else

{

//not in knownfilelist - start adding thread to hash file if the hashing of this file isnt already waiting

// SLUGFILLER: SafeHash - don't double hash, MY way

if (!IsHashing(strFoundDirectory, strFoundFileName) && !thePrefs.IsTempFile(strFoundDirectory, strFoundFileName)){

UnknownFile\_Struct\* tohash = new UnknownFile\_Struct;

tohash->strDirectory = strFoundDirectory;

tohash->strName = strFoundFileName;

tohash->strSharedDirectory = strShellLinkDir;

waitingforhash\_list.AddTail(tohash);

}

}

5.1、已在Knownfilelist中的处理

5.1.1、AddFile() 将文件添加到m\_Files\_map，再根据需要决定是否AddKeywords()

bool CSharedFileList::AddFile(CKnownFile\* pFile)

{

CCKey key(pFile->GetFileHash()); ///snow:return toadd.m\_FileIdentifier.m\_abyMD4Hash

CKnownFile\* pFileInMap;

if (m\_Files\_map.Lookup(key, pFileInMap))

{

File already in shared file list

return false;

}

m\_UnsharedFiles\_map.RemoveKey(CSKey(pFile->GetFileHash()));

m\_Files\_map.SetAt(key, pFile);

bool bKeywordsNeedUpdated = true;

if(bKeywordsNeedUpdated)

m\_keywords->AddKeywords(pFile);

return true;

}

5.1.2、///根据文件名获取各关键字，将关键字对象添加到m\_lstKeywords，然后将文件源添加到各关键字对象的m\_aFiles中

void CPublishKeywordList::AddKeywords(CKnownFile\* pFile)

{

const Kademlia::WordList& wordlist = pFile->GetKadKeywords();

//ASSERT( wordlist.size() > 0 );

Kademlia::WordList::const\_iterator it;

for (it = wordlist.begin(); it != wordlist.end(); it++)

{

const CStringW& strKeyword = \*it;

CPublishKeyword\* pPubKw = FindKeyword(strKeyword);

if (pPubKw == NULL) ///snow:如果该keyword不存在，添加keyword到列表末尾

{

pPubKw = new CPublishKeyword(strKeyword);

m\_lstKeywords.AddTail(pPubKw);

SetNextPublishTime(0);

}

///snow:添加文件引用到该Keyword，在pPubKw对象的m\_aFiles数组中添加一个pFile的条目

if(pPubKw->AddRef(pFile) && pPubKw->GetNextPublishTime() > MIN2S(30))

{

// User may be adding and removing files, so if this is a keyword that

// has already been published, we reduce the time, but still give the user

// enough time to finish what they are doing.

// If this is a hot node, the Load list will prevent from republishing.

pPubKw->SetNextPublishTime(MIN2S(30));

}

}

}

5.2、未在Knownfilelist中的处理

在void CSharedFileList::FindSharedFiles()、CheckAndAddSingleFile()、AddSingleSharedDirectory()、SetOutputCtrl()、RemoveFromHashing()、HashFailed()等函数中均启动调用HashNextFile();

5.2.1、HashNextFile()启动线程CAddFileThread，对waitingforhash\_list中的待处理文件进行hash处理

void CSharedFileList::HashNextFile(){

///snow:将文件从waitingforhash\_list移动到currentlyhashing\_list

UnknownFile\_Struct\* nextfile = waitingforhash\_list.RemoveHead();

currentlyhashing\_list.AddTail(nextfile); // SLUGFILLER: SafeHash - keep track

CAddFileThread\* addfilethread = (CAddFileThread\*) AfxBeginThread(RUNTIME\_CLASS(CAddFileThread), THREAD\_PRIORITY\_BELOW\_NORMAL,0, CREATE\_SUSPENDED);

addfilethread->SetValues(this, nextfile->strDirectory, nextfile->strName, nextfile->strSharedDirectory);

addfilethread->ResumeThread();

// SLUGFILLER: SafeHash - nextfile deleting handled elsewhere

//delete nextfile;

}

5.2.2、线程CaddFileThread::Run()启动，调用CknownFile:: CreateFromFile()对文件进行分块Hash

int CAddFileThread::Run()

{

CoInitialize(NULL);

// locking that hashing thread is needed because we may create a couple of those threads at startup when rehashing

// potentially corrupted downloading part files. if all those hash threads would run concurrently, the io-system would be

// under very heavy load and slowly progressing

CSingleLock sLock1(&theApp.hashing\_mut); // only one filehash at a time

sLock1.Lock();

CString strFilePath;

\_tmakepathlimit(strFilePath.GetBuffer(MAX\_PATH), NULL, m\_strDirectory, m\_strFilename, NULL);

strFilePath.ReleaseBuffer();

CKnownFile\* newrecord = new CKnownFile();

///snow:对文件进行分块Hash

if (newrecord->CreateFromFile(m\_strDirectory, m\_strFilename, m\_partfile) && theApp.emuledlg && theApp.emuledlg->IsRunning()) // SLUGFILLER: SafeHash - in case of shutdown while still hashing

{

newrecord->SetSharedDirectory(m\_strSharedDir);

if (!PostMessage(theApp.emuledlg->m\_hWnd, TM\_FINISHEDHASHING, (m\_pOwner ? 0: (WPARAM)m\_partfile), (LPARAM)newrecord))

delete newrecord;

}

else

{

if (!PostMessage(theApp.emuledlg->m\_hWnd, TM\_HASHFAILED, 0, (LPARAM)hashed))

delete hashed;

}

// SLUGFILLER: SafeHash

delete newrecord;

}

sLock1.Unlock();

CoUninitialize();

return 0;

}

5.2.3、CreateFromFile()对文件进行分块Hash

5.2.3.1、Hash成功，发送TM\_FINISHEDHASHING信息给wnd窗口，emuledlg截获信息，调用函数进行处理：ON\_MESSAGE(TM\_FINISHEDHASHING, OnFileHashed)

5.2.3.1.1、OnFileHashed()根据参数wParam，分别调用CpartFile:: PartFileHashFinished()和Csharefilelist::FileHashingFinished()进行处理

LRESULT CemuleDlg::OnFileHashed(WPARAM wParam, LPARAM lParam)

{

CKnownFile\* result = (CKnownFile\*)lParam;

if (wParam)

{

// File hashing finished for a part file when:

// - part file just completed

// - part file was rehashed at startup because the file date of part.met did not match the part file date

CPartFile\* requester = (CPartFile\*)wParam;

if (theApp.downloadqueue->IsPartFile(requester))

requester->PartFileHashFinished(result);

}

else

{

// File hashing finished for a shared file (none partfile)

// - reading shared directories at startup and hashing files which were not found in known.met

// - reading shared directories during runtime (user hit Reload button, added a shared directory, ...)

theApp.sharedfiles->FileHashingFinished(result);

}

return TRUE;

}

5.2.3.1.2、void CSharedFileList::FileHashingFinished(CKnownFile\* file)

{

// File hashing finished for a shared file (none partfile)

// - reading shared directories at startup and hashing files which were not found in known.met

// - reading shared directories during runtime (user hit Reload button, added a shared directory, ...)

CKnownFile\* found\_file = GetFileByID(file->GetFileHash());

if (found\_file == NULL)

{

// check if we still want to actually share this file, the user might have unshared it while hashing

if (!ShouldBeShared(file->GetSharedDirectory(), file->GetFilePath(), false)){

RemoveFromHashing(file);

…

}

else

{

SafeAddKFile(file);

theApp.knownfiles->SafeAddKFile(file);

}

}

else

{

File already in shared file list

}

}

5.2.1.3.3、SafeAddKFile()调用AddFile(),后续同(一)、已在Knownfilelist中的处理

bool CSharedFileList::SafeAddKFile(CKnownFile\* toadd, bool bOnlyAdd)

{

bool bAdded = false;

RemoveFromHashing(toadd); // SLUGFILLER: SafeHash - hashed ok, remove from list, in case it was on the list

bAdded = AddFile(toadd);

…

m\_lastPublishED2KFlag = true;

return bAdded;

}

5.2.3.1.4、bool CKnownFileList::SafeAddKFile(CKnownFile\* toadd)

{

CCKey key(toadd->GetFileHash());

CKnownFile\* pFileInMap;

if (m\_Files\_map.Lookup(key, pFileInMap))

{

Already in known list

}

m\_Files\_map.SetAt(key, toadd);

… m\_mapKnownFilesByAICH.SetAt(toadd->GetFileIdentifier().GetAICHHash(), toadd);

return true;

}

5.2.3.2、Hash失败，发送TM\_HASHFAILED信息给wnd窗口，emuledlg截获信息，调用函数进行处理：ON\_MESSAGE(TM\_FINISHEDHASHING, OnHashFailed)

5.2.3.2.1、LRESULT CemuleDlg::OnHashFailed(WPARAM /\*wParam\*/, LPARAM lParam)

{

theApp.sharedfiles->HashFailed((UnknownFile\_Struct\*)lParam);

return 0;

}

5.2.3.2.2、void CSharedFileList::HashFailed(UnknownFile\_Struct\* hashed){

for (POSITION pos = currentlyhashing\_list.GetHeadPosition(); pos != 0; ){

POSITION posLast = pos;

const UnknownFile\_Struct\* pFile = currentlyhashing\_list.GetNext(pos);

if (!pFile->strName.CompareNoCase(hashed->strName) && !CompareDirectories(pFile->strDirectory, hashed->strDirectory)){

currentlyhashing\_list.RemoveAt(posLast);

delete pFile;

HashNextFile(); // start next hash if possible, but only if a previous hash finished

break;

}

}

delete hashed;

}

## 二、文件发布过程

# 文件Hash过程

1、