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？

通过对文件名以下字符为分隔符进行拆分：

#define INV\_KAD\_KEYWORD\_CHARS " ()[]{}<>,.\_-!?:;\\/\""

LPCSTR g\_aszInvKadKeywordCharsA = INV\_KAD\_KEYWORD\_CHARS;

LPCTSTR g\_aszInvKadKeywordChars = \_T(INV\_KAD\_KEYWORD\_CHARS);

LPCWSTR g\_awszInvKadKeywordChars = L" ()[]{}<>,.\_-!?:;\\/\"";

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

KeywordID是关键字ID，根据文件名分解成各个关键字的HASH值（GetKadID()），SourceID是文件ID，根据对文件进行分块HASH，再总HASH得来的md4HASH值(GetFileHash())

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）

1、参见：文件发布过程 3.1、发布StoreKeyword

2、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;

}

3、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

4、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、启动

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构造函数

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

{

…

LoadSingleSharedFilesList();

FindSharedFiles();

}

### 3、LoadSingleSharedFilesList与AddSingleSharedFile

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、CheckAndAddSingleFile

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，参见文件Hash过程

###### 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、FileHashingFinished

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

}

}

5.2.3.1.3、SafeAddKFile()

调用AddFile(),后续同5.1、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、CKnownFileList::SafeAddKFile

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、OnHashFailed

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

{

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

return 0;

}

5.2.3.2.2、HashFailed

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

### 1、启动-- UploadTimer()

CUploadQueue::UploadTimer()中调用CSharedFileList::Process();

### 2、Process()函数

调用Publish()进行KAD发布，调用SendListToServer()将文件列表发布到emule服务器

void CSharedFileList::Process()

{

Publish();

if( !m\_lastPublishED2KFlag || ( ::GetTickCount() - m\_lastPublishED2K < ED2KREPUBLISHTIME ) )

{

return;

}

SendListToServer();

m\_lastPublishED2K = ::GetTickCount();

}

### 3、Publish()函数

发布三种类型的资源

void CSharedFileList::Publish()

{

//1、StoreKeyword

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

{

//pSearch was created. Which means no search was already being done with this HashID.

//This also means that it was checked to see if network load wasn't a factor.

//This sets the filename into the search object so we can show it in the gui.

pSearch->SetGUIName(pPubKw->GetKeyword());

//Add all file IDs which relate to the current keyword to be published

const CSimpleKnownFileArray& aFiles = pPubKw->GetReferences();

uint32 count = 0;

for (int f = 0; f < aFiles.GetSize(); f++)

{

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

ASSERT\_VALID( aFiles[f] );

// JOHNTODO - Why is this happening.. I think it may have to do with downloading a file that is already

// in the known file list..

// ASSERT( IsFilePtrInList(aFiles[f]) );

//Only publish complete files as someone else should have the full file to publish these keywords.

//As a side effect, this may help reduce people finding incomplete files in the network.

if( !aFiles[f]->IsPartFile() && IsFilePtrInList(aFiles[f]))

{

count++;

pSearch->AddFileID(Kademlia::CUInt128(aFiles[f]->GetFileHash()));///snow:将fileHash添加到m\_listFileIDs

if( count > 150 )

{

//We only publish up to 150 files per keyword publish then rotate the list.

pPubKw->RotateReferences(f);///snow:将m\_aFiles中的元素前后掉个个

break;

}

}

}

if( count )

{

//Start our keyword publish

pPubKw->SetNextPublishTime(tNow+(KADEMLIAREPUBLISHTIMEK));

pPubKw->IncPublishedCount();

Kademlia::CSearchManager::StartSearch(pSearch);

}

else

{

//There were no valid files to publish with this keyword.

delete pSearch;

}

}

}

}

m\_keywords->SetNextPublishTime(KADEMLIAPUBLISHTIME+tNow);

}

}

//2、发布StoreFile

if( Kademlia::CKademlia::GetTotalStoreSrc() < KADEMLIATOTALSTORESRC)

{

if(tNow >= m\_lastPublishKadSrc)

{

if(m\_currFileSrc > GetCount())

m\_currFileSrc = 0; ///snow:m\_Files\_map中的index

CKnownFile\* pCurKnownFile = GetFileByIndex(m\_currFileSrc);

if(pCurKnownFile)

{

if(pCurKnownFile->PublishSrc()) ///snow:允许publish

{

if(Kademlia::CSearchManager::PrepareLookup(Kademlia::CSearch::STOREFILE, true, Kademlia::CUInt128(pCurKnownFile->GetFileHash()))==NULL)

pCurKnownFile->SetLastPublishTimeKadSrc(0,0);

}

}

m\_currFileSrc++;

// even if we did not publish a source, reset the timer so that this list is processed

// only every KADEMLIAPUBLISHTIME seconds.

m\_lastPublishKadSrc = KADEMLIAPUBLISHTIME+tNow;

}

}

///3、发布StoreNotes

if( Kademlia::CKademlia::GetTotalStoreNotes() < KADEMLIATOTALSTORENOTES)

{

if(tNow >= m\_lastPublishKadNotes)

{

if(m\_currFileNotes > GetCount())

m\_currFileNotes = 0;

CKnownFile\* pCurKnownFile = GetFileByIndex(m\_currFileNotes);

if(pCurKnownFile)

{

if(pCurKnownFile->PublishNotes())

{

if(Kademlia::CSearchManager::PrepareLookup(Kademlia::CSearch::STORENOTES, true, Kademlia::CUInt128(pCurKnownFile->GetFileHash()))==NULL)

pCurKnownFile->SetLastPublishTimeKadNotes(0);

}

}

m\_currFileNotes++;

// even if we did not publish a source, reset the timer so that this list is processed

// only every KADEMLIAPUBLISHTIME seconds.

m\_lastPublishKadNotes = KADEMLIAPUBLISHTIME+tNow;

}

}

}

}

#### 3.1、发布StoreKeyword，主要动作是：

##### 1、构造CSearch对象

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

const CSimpleKnownFileArray& aFiles = pPubKw->GetReferences();

for (int f = 0; f < aFiles.GetSize(); f++)

{

pSearch->AddFileID(Kademlia::CUInt128(aFiles[f]->GetFileHash()));

}

Kademlia::CSearchManager::StartSearch(pSearch);

##### 2、PrepareLookup()

///snow:检查是否已存在相同搜索，当searchtype==STOREKEYWORD检查是否已存在相同节点，增加searchid，如果bStart为true，将uID添加到搜索列表m\_mapSearches，开始搜索

CSearch\* CSearchManager::PrepareLookup(uint32 uType, bool bStart, const CUInt128 &uID)

{

// Prepare a kad lookup.

// Make sure this target is not already in progress.

if(AlreadySearchingFor(uID))

return NULL;

// Create a new search.

///snow:以参数uType和uID构造一个CSearch对象

CSearch \*pSearch = new CSearch;

// Set type and target.

pSearch->SetSearchTypes(uType);

pSearch->m\_uTarget = uID;

try

{

switch(pSearch->m\_uType)

{

case CSearch::STOREKEYWORD: ///snow:CSharedFileList::Publish()中以此参数调用

if(!Kademlia::CKademlia::GetIndexed()->SendStoreRequest(uID)) ///m\_mapLoad中已存在这个节点，且没有过期

{

// Keyword Store was determined to be a overloaded node, abort store.

delete pSearch; ///snow:不再需要重复搜索，删除掉搜索节点，函数返回

return NULL;

}

break;

}

// Inc search ID.

pSearch->m\_uSearchID = ++m\_uNextID;

///snow:bStart参数决定是否开始搜索

if( bStart ) ///snow:在StoreKeyword时为false

{

…

}

}

…

return pSearch;

}

##### 3、搜索四部曲：

go()🡪SendFindValue()🡪JumpStart()🡪StorePacket()

void CSearch::StorePacket()

{

case STOREKEYWORD:

{

///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50个文件一个信息包，上一步AddFileID()时添加的

{

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50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0

byIO.Seek(current\_pos);

// Send packet

CUInt128 uClientID = pFromContact->GetClientID();

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

// Inc total request answers

m\_uTotalRequestAnswers++;

break;

}

…

}

##### 4、Process\_KADEMLIA2\_PUBLISH\_KEY\_REQ

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();

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;

CString sInfo;

uint16 uCount = byteIO.ReadUInt16();

uint8 uLoad = 0;

CUInt128 uTarget;

while( uCount > 0 )

{

sInfo.Empty();

byteIO.ReadUInt128(&uTarget);

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

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)

{

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

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( !pIndexed->AddKeyword(uFile, uTarget, pEntry, uLoad) ) ///snow: 每个Entry执行一次AddKeyword()，参数uFile，uTarget其实都已经存储在pEntry了，这里也返回了uLoad

{

…

}

uCount--;

}

CSafeMemFile fileIO2(17);

fileIO2.WriteUInt128(&uFile);

fileIO2.WriteUInt8(uLoad);

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

}

##### 5、AddKeyword()

添加keyword到m\_mapKeyword中，并传回uLoad值

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

{

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;

}

}

}

##### 6、发回KADEMLIA2\_PUBLISH\_RES 信息包

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

##### 7、Process\_KADEMLIA2\_PUBLISH\_RES

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;

}

…

}

#### 3.2、发布StoreFile

##### 1、构造CSearch对象

PrepareLookup()中参数bStart为true，表示马上开始搜索

if(Kademlia::CSearchManager::PrepareLookup(Kademlia::CSearch::STOREFILE, true, Kademlia::CUInt128(pCurKnownFile->GetFileHash()))==NULL)

##### 2、PrepareLookup()

///snow:检查是否已存在相同搜索，当searchtype==STOREKEYWORD检查是否已存在相同节点，增加searchid，如果bStart为true，将uID添加到搜索列表m\_mapSearches，开始搜索

CSearch\* CSearchManager::PrepareLookup(uint32 uType, bool bStart, const CUInt128 &uID)

{

// Prepare a kad lookup.

// Make sure this target is not already in progress.

if(AlreadySearchingFor(uID))

return NULL;

// Create a new search.

///snow:以参数uType和uID构造一个CSearch对象

CSearch \*pSearch = new CSearch;

// Set type and target.

pSearch->SetSearchTypes(uType);

pSearch->m\_uTarget = uID;

try

{

switch(pSearch->m\_uType)

{

case CSearch::STOREKEYWORD: ///snow:与StoreFile无关

break;

}

// Inc search ID.

pSearch->m\_uSearchID = ++m\_uNextID;

///snow:bStart参数决定是否开始搜索

if( bStart ) ///snow:本例中为true，马上开始搜索

{

// Auto start this search. ///snow:加上if(AlreadySearchingFor(uID))就是StartSearch

m\_mapSearches[pSearch->m\_uTarget] = pSearch;

pSearch->Go();

}

}

…

return pSearch;

}

##### 3、搜索四部曲：

go()🡪SendFindValue()🡪JumpStart()🡪StorePacket()

void CSearch::StorePacket()

{

case STOREFILE: ///snow:将自己的资源发布到网络节点上

{

// Try to store yourself as a source to a Node.

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

if( m\_uAnswers > SEARCHSTOREFILE\_TOTAL ) ///snow:最多10个

{

PrepareToStop();

break;

}

// Find the file we are trying to store as a source too.

uchar ucharFileid[16];

m\_uTarget.ToByteArray(ucharFileid);

CKnownFile\* pFile = theApp.sharedfiles->GetFileByID(ucharFileid); ///snow:共享的文件

if (pFile)

{

// We set this mostly for GUI resonse.

SetGUIName(pFile->GetFileName());

// Get our clientID for the packet.

CUInt128 uID(CKademlia::GetPrefs()->GetClientHash());

//We can use type for different types of sources. ///snow:SOURCETYPE类别，用-6表示(2保留）

//1 HighID Sources..

//2 cannot be used as older clients will not work.

//3 Firewalled Kad Source.

//4 >4GB file HighID Source.

//5 >4GB file Firewalled Kad source.

//6 Firewalled Source with Direct Callback (supports >4GB)

bool bDirectCallback = false;

TagList listTag;

if( theApp.IsFirewalled() ) ///snow:低ID用户，要求对方向自己发送回调请求连接对方

{

bDirectCallback = (Kademlia::CKademlia::IsRunning() && !Kademlia::CUDPFirewallTester::IsFirewalledUDP(true) && Kademlia::CUDPFirewallTester::IsVerified()); ///snow:本机UDP端口没有被墙

if (bDirectCallback){

// firewalled, but direct udp callback is possible so no need for buddies

// We are not firewalled..

listTag.push\_back(new CKadTagUInt(TAG\_SOURCETYPE, 6)); ///snow:Firewalled Source with Direct Callback (supports >4GB)

listTag.push\_back(new CKadTagUInt(TAG\_SOURCEPORT, thePrefs.GetPort()));

if (!CKademlia::GetPrefs()->GetUseExternKadPort())

listTag.push\_back(new CKadTagUInt16(TAG\_SOURCEUPORT, CKademlia::GetPrefs()->GetInternKadPort()));

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

{

listTag.push\_back(new CKadTagUInt(TAG\_FILESIZE, pFile->GetFileSize()));

}

}

else if( theApp.clientlist->GetBuddy() ) // We are firewalled, make sure we have a buddy.

{ ///snow:我们被墙了，需要一个Buddy中转连接

// We send the ID to our buddy so they can do a callback.

CUInt128 uBuddyID(true);

uBuddyID.Xor(CKademlia::GetPrefs()->GetKadID());

if(pFile->GetFileSize() > OLD\_MAX\_EMULE\_FILE\_SIZE) ///snow:大于GB

listTag.push\_back(new CKadTagUInt8(TAG\_SOURCETYPE, 5)); ///snow:>4GB file Firewalled Kad source.

else

listTag.push\_back(new CKadTagUInt8(TAG\_SOURCETYPE, 3));///snow:Firewalled Kad Source.

listTag.push\_back(new CKadTagUInt(TAG\_SERVERIP, theApp.clientlist->GetBuddy()->GetIP()));

listTag.push\_back(new CKadTagUInt(TAG\_SERVERPORT, theApp.clientlist->GetBuddy()->GetUDPPort()));

listTag.push\_back(new CKadTagStr(TAG\_BUDDYHASH, CStringW(md4str(uBuddyID.GetData()))));

listTag.push\_back(new CKadTagUInt(TAG\_SOURCEPORT, thePrefs.GetPort()));

if (!CKademlia::GetPrefs()->GetUseExternKadPort())

listTag.push\_back(new CKadTagUInt16(TAG\_SOURCEUPORT, CKademlia::GetPrefs()->GetInternKadPort()));

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

{

listTag.push\_back(new CKadTagUInt(TAG\_FILESIZE, pFile->GetFileSize()));

}

}

else

{

// We are firewalled, but lost our buddy.. Stop everything.

PrepareToStop();

break;

}

}

else ///snow:我们是高ID用户

{

// We are not firewalled..

if(pFile->GetFileSize() > OLD\_MAX\_EMULE\_FILE\_SIZE)

listTag.push\_back(new CKadTagUInt(TAG\_SOURCETYPE, 4)); ///snow:>4GB file HighID Source.

else

listTag.push\_back(new CKadTagUInt(TAG\_SOURCETYPE, 1)); ///snow:HighID Sources.

listTag.push\_back(new CKadTagUInt(TAG\_SOURCEPORT, thePrefs.GetPort()));

if (!CKademlia::GetPrefs()->GetUseExternKadPort())

listTag.push\_back(new CKadTagUInt16(TAG\_SOURCEUPORT, CKademlia::GetPrefs()->GetInternKadPort()));

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

{

listTag.push\_back(new CKadTagUInt(TAG\_FILESIZE, pFile->GetFileSize()));

}

}

listTag.push\_back(new CKadTagUInt8(TAG\_ENCRYPTION, CKademlia::GetPrefs()->GetMyConnectOptions(true, true)));

// Send packet

///snow:向回应的联系人发布源信息包，不是调用SendPacket()

CKademlia::GetUDPListener()->SendPublishSourcePacket(pFromContact, m\_uTarget, uID, listTag);

// Inc total request answers

m\_uTotalRequestAnswers++;

// Update search in the GUI

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

// Delete all tags.

for (TagList::const\_iterator itTagList = listTag.begin(); itTagList != listTag.end(); ++itTagList)

delete \*itTagList;

}

else

PrepareToStop();

break;

}

##### 4、SendPublishSourcePacket()

void CKademliaUDPListener::SendPublishSourcePacket(CContact\* pContact, const CUInt128 &uTargetID, const CUInt128 &uContactID, const TagList& tags)

{

//We need to get the tag lists working with CSafeMemFiles..

byte byPacket[1024];

CByteIO byteIO(byPacket, sizeof(byPacket));

byteIO.WriteByte(OP\_KADEMLIAHEADER);

if (pContact->GetVersion() >= 4/\*47c\*/)

{

byteIO.WriteByte(KADEMLIA2\_PUBLISH\_SOURCE\_REQ);

byteIO.WriteUInt128(uTargetID);

byteIO.WriteUInt128(uContactID);

byteIO.WriteTagList(tags);

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

{

DebugSend("KADEMLIA2\_PUBLISH\_SOURCE\_REQ", pContact->GetIPAddress(), pContact->GetUDPPort());

}

}

else

{

byteIO.WriteByte(KADEMLIA\_PUBLISH\_REQ);

byteIO.WriteUInt128(uTargetID);

//We only use this for publishing sources now.. So we always send one here..

byteIO.WriteUInt16(1);

byteIO.WriteUInt128(uContactID);

byteIO.WriteTagList(tags);

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

{

DebugSend("KADEMLIA\_PUBLISH\_REQ", pContact->GetIPAddress(), pContact->GetUDPPort());

}

}

uint32 uLen = sizeof(byPacket) - byteIO.GetAvailable();

if (pContact->GetVersion() >= 6/\*>48b\*/) // obfuscated?

{

CUInt128 uClientID = pContact->GetClientID();

SendPacket(byPacket, uLen, pContact->GetIPAddress(), pContact->GetUDPPort(), pContact->GetUDPKey(), &uClientID);

}

else

SendPacket(byPacket, uLen, pContact->GetIPAddress(), pContact->GetUDPPort(), 0, NULL);

}

##### 5、Process\_KADEMLIA2\_PUBLISH\_SOURCE\_REQ

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

{

//Used Pointers

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

CByteIO byteIO(pbyPacketData, uLenPacket);

CUInt128 uFile;

byteIO.ReadUInt128(&uFile);

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

uDistance.Xor(uFile);

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

return;

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

CString sInfo;

sInfo.Empty();

uint8 uLoad = 0;

bool bFlag = false;

CUInt128 uTarget;

byteIO.ReadUInt128(&uTarget);

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

try

{

pEntry->m\_uIP = uIP;

pEntry->m\_uUDPPort = uUDPPort;

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

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

pEntry->m\_bSource = false;

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

bool bAddUDPPortTag = true;

uint32 uTags = byteIO.ReadByte();

while(uTags > 0)

{

CKadTag\* pTag = byteIO.ReadTag();

if(pTag)

{

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

{

if( pEntry->m\_bSource == false )

{

pEntry->AddTag(new CKadTagUInt(TAG\_SOURCEIP, pEntry->m\_uIP));

pEntry->AddTag(pTag);

pEntry->m\_bSource = true;

}

else

{

//More then one sourcetype tag found.

delete pTag;

}

}

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;

}

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

{

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

{

pEntry->m\_uTCPPort = (uint16)pTag->GetInt();

pEntry->AddTag(pTag);

}

else

{

//More then one port tag found

delete pTag;

}

}

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

{

if(bAddUDPPortTag && pTag->IsInt() && pTag->GetInt() != 0)

{

pEntry->m\_uUDPPort = (uint16)pTag->GetInt();

pEntry->AddTag(pTag);

bAddUDPPortTag = false;

}

else

{

//More then one udp port tag found

delete pTag;

}

}

else

{

//TODO: Filter tags

pEntry->AddTag(pTag);

}

}

uTags--;

}

if (bAddUDPPortTag)

pEntry->AddTag(new CKadTagUInt(TAG\_SOURCEUPORT, pEntry->m\_uUDPPort));

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

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

}

catch(...)

{

delete pEntry;

throw;

}

if( pEntry->m\_bSource == true )

{

if( pIndexed->AddSources(uFile, uTarget, pEntry, uLoad ) ) ///snow:主要动作

bFlag = true;

else

{

delete pEntry;

pEntry = NULL;

}

}

else

{

delete pEntry;

pEntry = NULL;

}

if( bFlag )

{

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);

}

}

##### 6、AddSources()

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

{

…

SrcHash\* pCurrSrcHash;

if(!m\_mapSources.Lookup(CCKey(uKeyID.GetData()), pCurrSrcHash)) ///snow:不存在同ID的Source条目

{

Source\* pCurrSource = new Source;

pCurrSource->uSourceID.SetValue(uSourceID);

pCurrSource->ptrlEntryList.AddHead(pEntry);

pCurrSrcHash = new SrcHash;

pCurrSrcHash->uKeyID.SetValue(uKeyID);

pCurrSrcHash->ptrlistSource.AddHead(pCurrSource);

m\_mapSources.SetAt(CCKey(pCurrSrcHash->uKeyID.GetData()), pCurrSrcHash);

m\_uTotalIndexSource++;

uLoad = 1; ///snow:新文件，uLoad =1

return true;

}

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;

}

}

else

{

//This should never happen!

pCurrSource->ptrlEntryList.AddHead(pEntry);

ASSERT(0);

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

m\_uTotalIndexSource++;

return true;

}

}

///snow:IP、Port或UDPPort不相一致

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;

}

}

}

##### 7、发回KADEMLIA2\_PUBLISH\_RES 信息包

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

##### 8、Process\_KADEMLIA2\_PUBLISH\_RES

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;

…

// Inc the number of answers.

pSearch->m\_uAnswers++; ///snow:统计回应的联系人数，在JumpStart中用作判断是否继续搜索的依据

// Update the search for the GUI

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

}

#### 3.3、发布StoreNote

##### 1、构造CSearch对象

PrepareLookup()中参数bStart为true，表示马上开始搜索

if(Kademlia::CSearchManager::PrepareLookup(Kademlia::CSearch::STORENOTES, true, Kademlia::CUInt128(pCurKnownFile->GetFileHash()))==NULL)

##### 2、PrepareLookup()

处理方式同StoreFile.

##### 3、搜索四部曲：

go()🡪SendFindValue()🡪JumpStart()🡪StorePacket()

void CSearch::StorePacket()

{

case STORENOTES:

{

// Find file we are storing info about.

uchar ucharFileid[16];

m\_uTarget.ToByteArray(ucharFileid);

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

if (pFile)

{

byte byPacket[1024\*2];

CByteIO byIO(byPacket,sizeof(byPacket));

// Send the Hash of the file we are storing info about.

byIO.WriteUInt128(m\_uTarget);

// Send our ID with the info.

byIO.WriteUInt128(CKademlia::GetPrefs()->GetKadID());

// Create our taglist

TagList listTag;

listTag.push\_back(new CKadTagStr(TAG\_FILENAME, pFile->GetFileName()));

if(pFile->GetFileRating() != 0)

listTag.push\_back(new CKadTagUInt(TAG\_FILERATING, pFile->GetFileRating()));

if(pFile->GetFileComment() != \_T(""))

listTag.push\_back(new CKadTagStr(TAG\_DESCRIPTION, pFile->GetFileComment()));

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

listTag.push\_back(new CKadTagUInt(TAG\_FILESIZE, pFile->GetFileSize()));

byIO.WriteTagList(listTag);

// Send packet

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

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

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

CUInt128 uClientID = pFromContact->GetClientID();

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

}

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

{

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

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

CKademlia::GetUDPListener()->SendPacket( byPacket, sizeof(byPacket)-byIO.GetAvailable(), KADEMLIA2\_PUBLISH\_NOTES\_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);

// Delete all tags.

for (TagList::const\_iterator itTagList = listTag.begin(); itTagList != listTag.end(); ++itTagList)

delete \*itTagList;

}

else

PrepareToStop();

break;

}

…

}

##### 4、Process\_KADEMLIA2\_PUBLISH\_NOTES\_REQ

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

{

// 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 uTarget;

byteIO.ReadUInt128(&uTarget);

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

uDistance.Xor(uTarget);

// Shouldn't LAN IPs already be filtered?

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

return;

CUInt128 uSource;

byteIO.ReadUInt128(&uSource);

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

try

{

pEntry->m\_uIP = uIP;

pEntry->m\_uUDPPort = uUDPPort;

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

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

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());

}

delete pTag;

}

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

{

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

{

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

}

delete pTag;

}

else

{

//TODO: Filter tags

pEntry->AddTag(pTag);

}

}

uTags--;

}

}

catch(...)

{

delete pEntry;

pEntry = NULL;

throw;

}

uint8 uLoad = 0;

if( CKademlia::GetIndexed()->AddNotes(uTarget, uSource, pEntry, uLoad ) )

{

CSafeMemFile fileIO2(17);

fileIO2.WriteUInt128(&uTarget);

fileIO2.WriteUInt8(uLoad);

if (thePrefs.GetDebugClientKadUDPLevel() > 0)

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

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

}

else

delete pEntry;

}

##### 5、AddNotes()

bool CIndexed::AddNotes(const CUInt128& uKeyID, const CUInt128& uSourceID, Kademlia::CEntry\* 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->m\_uIP == 0 || pEntry->GetTagCount() == 0 )

return false;

SrcHash\* pCurrNoteHash;

if(!m\_mapNotes.Lookup(CCKey(uKeyID.GetData()), pCurrNoteHash))

{

Source\* pCurrNote = new Source;

pCurrNote->uSourceID.SetValue(uSourceID);

pCurrNote->ptrlEntryList.AddHead(pEntry);

SrcHash\* pCurrNoteHash = new SrcHash;

pCurrNoteHash->uKeyID.SetValue(uKeyID);

pCurrNoteHash->ptrlistSource.AddHead(pCurrNote);

m\_mapNotes.SetAt(CCKey(pCurrNoteHash->uKeyID.GetData()), pCurrNoteHash);

uLoad = 1;

m\_uTotalIndexNotes++;

return true;

}

else

{

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

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

{

Source\* pCurrNote = pCurrNoteHash->ptrlistSource.GetNext(pos1);

if( pCurrNote->ptrlEntryList.GetSize() )

{

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

if(pCurrEntry->m\_uIP == pEntry->m\_uIP || pCurrEntry->m\_uSourceID == pEntry->m\_uSourceID)

{

delete pCurrNote->ptrlEntryList.RemoveHead();

pCurrNote->ptrlEntryList.AddHead(pEntry);

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

return true;

}

}

else

{

//This should never happen!

pCurrNote->ptrlEntryList.AddHead(pEntry);

ASSERT(0);

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

m\_uTotalIndexKeyword++;

return true;

}

}

if( uSize > KADEMLIAMAXNOTESPERFILE )

{

Source\* pCurrNote = pCurrNoteHash->ptrlistSource.RemoveTail();

delete pCurrNote->ptrlEntryList.RemoveTail();

pCurrNote->uSourceID.SetValue(uSourceID);

pCurrNote->ptrlEntryList.AddHead(pEntry);

pCurrNoteHash->ptrlistSource.AddHead(pCurrNote);

uLoad = 100;

return true;

}

else

{

Source\* pCurrNote = new Source;

pCurrNote->uSourceID.SetValue(uSourceID);

pCurrNote->ptrlEntryList.AddHead(pEntry);

pCurrNoteHash->ptrlistSource.AddHead(pCurrNote);

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

m\_uTotalIndexKeyword++;

return true;

}

}

}

##### 6、发回KADEMLIA2\_PUBLISH\_RES 信息包

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

##### 7、Process\_KADEMLIA2\_PUBLISH\_RES

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;

}

…

}

# 文件Hash过程

## 1、线程CAddFileThread::Run()

启动，调用CknownFile:: CreateFromFile()对文件进行分块Hash

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

{

…

}

## 2、CreateFromFile

bool CKnownFile::CreateFromFile(LPCTSTR in\_directory, LPCTSTR in\_filename, LPVOID pvProgressParam)

{

SetPath(in\_directory);

SetFileName(in\_filename);

// open file

CString strFilePath;

if (!\_tmakepathlimit(strFilePath.GetBuffer(MAX\_PATH), NULL, in\_directory, in\_filename, NULL)){

LogError(GetResString(IDS\_ERR\_FILEOPEN), in\_filename, \_T(""));

return false;

}

strFilePath.ReleaseBuffer();

SetFilePath(strFilePath);

///snow:add by snow

FILE\* file = \_tfsopen(strFilePath, \_T("rbS"), \_SH\_DENYNO); // can not use \_SH\_DENYWR because we may access a completing part file

if (!file){

LogError(GetResString(IDS\_ERR\_FILEOPEN) + \_T(" - %s"), strFilePath, \_T(""), \_tcserror(errno));

return false;

}

// set filesize

\_\_int64 llFileSize = \_filelengthi64(\_fileno(file));

if ((uint64)llFileSize > MAX\_EMULE\_FILE\_SIZE){

if (llFileSize == -1i64)

LogError(\_T("Failed to hash file \"%s\" - %s"), strFilePath, \_tcserror(errno));

else

LogError(\_T("Skipped hashing of file \"%s\" - File size exceeds limit."), strFilePath);

fclose(file);

return false; // not supported by network

}

SetFileSize((uint64)llFileSize);

// we are reading the file data later in 8K blocks, adjust the internal file stream buffer accordingly

setvbuf(file, NULL, \_IOFBF, 1024\*8\*2);

m\_AvailPartFrequency.SetSize(GetPartCount());

for (UINT i = 0; i < GetPartCount();i++)

m\_AvailPartFrequency[i] = 0;

// create hashset

///snow:根据文件大小进行分块，每块大小为PARTSIZE（=9500KB)，然后分块进行Hash

///snow: 构造文件的主Hash二叉树，通过FindHash()生成各子树

CAICHRecoveryHashSet cAICHHashSet(this, m\_nFileSize); ///snow:构造总HashSet

uint64 togo = m\_nFileSize;

UINT hashcount;

///snow:当文件大小大于PARTSIZE时，每块大小为PARTSIZE，调用CreateHash，生成newhash，存入m\_FileIdentifier；文件大小小于PARTSIZE时，跳过此步骤

for (hashcount = 0; togo >= PARTSIZE; )

{

///snow:Hash二叉树

CAICHHashTree\* pBlockAICHHashTree = cAICHHashSet.m\_pHashTree.FindHash((uint64)hashcount\*PARTSIZE, PARTSIZE);

ASSERT( pBlockAICHHashTree != NULL );

uchar\* newhash = new uchar[16];

///snow:CreateHash生成两种Hash，一种是MD4Hash，通过传出参数存入newhash，后面将加入m\_FileIdentifier.m\_aMD4HashSet

///snow:另一种是AICHHAsh，存放在pBlockAICHHashTree中，而pBlockAICHHashTree通过FindHash在cAICHHashSet中生成并定位对象

///snow:在CreateHash中SetBlockHash()对pBlockAICHHashTree重新动作FindHash，生成新的Hash子树

if (!CreateHash(file, PARTSIZE, newhash, pBlockAICHHashTree)) { ///snow:对分块进行hash，应该追踪一下file：一是for循环时每次传递给CreateHash的是同一个file，在CreateHash中，CStdioFile pFile(file)，pFile是新对象，但file不是，file中的位置指针依然有效；二是pFile->Read()时指针的移动是否等同file指针的移动？是的！

LogError(\_T("Failed to hash file \"%s\" - %s"), strFilePath, \_tcserror(errno));

fclose(file);

delete[] newhash;

return false;

}

///snow:hash过程中关机或关闭程序了

if (theApp.emuledlg==NULL || !theApp.emuledlg->IsRunning()){ // in case of shutdown while still hashing

fclose(file);

delete[] newhash;

return false;

}

m\_FileIdentifier.GetRawMD4HashSet().Add(newhash); ///snow:往m\_aMD4HashSet中添加hash，最后CalculateMD4HashByHashSet时用到，生成m\_abyMD4Hash

togo -= PARTSIZE;

hashcount++;

if (pvProgressParam && theApp.emuledlg && theApp.emuledlg->IsRunning()){

ASSERT( ((CKnownFile\*)pvProgressParam)->IsKindOf(RUNTIME\_CLASS(CKnownFile)) );

ASSERT( ((CKnownFile\*)pvProgressParam)->GetFileSize() == GetFileSize() );

UINT uProgress = (UINT)(uint64)(((uint64)(GetFileSize() - togo) \* 100) / GetFileSize());

ASSERT( uProgress <= 100 );

VERIFY( PostMessage(theApp.emuledlg->GetSafeHwnd(), TM\_FILEOPPROGRESS, uProgress, (LPARAM)pvProgressParam) );

}

}

///snow:文件分块的最后一块，一般文件大小不会刚好是PARTSIZE的整数倍，所以最后一块的大小就会小于PARTSIZE

///snow:或文件大小小于PARTSIZE

CAICHHashTree\* pBlockAICHHashTree;

if (togo == 0)

pBlockAICHHashTree = NULL; // sha hashtree doesnt takes hash of 0-sized data

else{

pBlockAICHHashTree = cAICHHashSet.m\_pHashTree.FindHash((uint64)hashcount\*PARTSIZE, togo); ///snow:生成并定位分块所对应的CAICHHashTree对象

ASSERT( pBlockAICHHashTree != NULL );

}

uchar\* lasthash = new uchar[16];

md4clr(lasthash);

if (!CreateHash(file, togo, lasthash, pBlockAICHHashTree)) { ///snow:对最后一块生成HAsh

LogError(\_T("Failed to hash file \"%s\" - %s"), strFilePath, \_tcserror(errno));

fclose(file);

delete[] lasthash;

return false;

}

cAICHHashSet.ReCalculateHash(false); ///snow:生成m\_Hash

if (cAICHHashSet.VerifyHashTree(true)) ///snow:校验AICHHash

{

cAICHHashSet.SetStatus(AICH\_HASHSETCOMPLETE); ///snow:设置状态为Hash完成

m\_FileIdentifier.SetAICHHash(cAICHHashSet.GetMasterHash()); ///snow:return m\_pHashTree.m\_Hash，赋值给m\_AICHFileHash

if (!m\_FileIdentifier.SetAICHHashSet(cAICHHashSet))

{

ASSERT( false );

DebugLogError(\_T("CreateFromFile() - failed to create AICH PartHashSet out of RecoveryHashSet - %s"), GetFileName());

}

if (!cAICHHashSet.SaveHashSet()) ///snow:将计算出来的文件Hash写入Known2\_64.met

LogError(LOG\_STATUSBAR, GetResString(IDS\_SAVEACFAILED));

else

SetAICHRecoverHashSetAvailable(true); ///snow:m\_bAICHRecoverHashSetAvailable=true

}

else{

// now something went pretty wrong

DebugLogError(LOG\_STATUSBAR, \_T("Failed to calculate AICH Hashset from file %s"), GetFileName());

}

///snow:生成MD4Hash

if (!hashcount){ ///snow:文件大小小于PARTSIZE

m\_FileIdentifier.SetMD4Hash(lasthash);

delete[] lasthash;

}

else {

m\_FileIdentifier.GetRawMD4HashSet().Add(lasthash);

m\_FileIdentifier.CalculateMD4HashByHashSet(false);//snow:从m\_aMD4HashSet计算MD4Hash，参数bVerifyOnly=false，表示赋值给m\_abyMD4Hash

}

if (pvProgressParam && theApp.emuledlg && theApp.emuledlg->IsRunning()){

ASSERT( ((CKnownFile\*)pvProgressParam)->IsKindOf(RUNTIME\_CLASS(CKnownFile)) );

ASSERT( ((CKnownFile\*)pvProgressParam)->GetFileSize() == GetFileSize() );

UINT uProgress = 100;

ASSERT( uProgress <= 100 );

VERIFY( PostMessage(theApp.emuledlg->GetSafeHwnd(), TM\_FILEOPPROGRESS, uProgress, (LPARAM)pvProgressParam) );

}

// set lastwrite date

struct \_stat fileinfo;

if (\_fstat(file->\_file, &fileinfo) == 0){

m\_tUtcLastModified = fileinfo.st\_mtime;

AdjustNTFSDaylightFileTime(m\_tUtcLastModified, strFilePath);

}

fclose(file);

file = NULL;

// Add filetags

UpdateMetaDataTags();

UpdatePartsInfo();

return true;

}

## 3、FindHash

///snow:生成并定位分块所对应的CAICHHashTree对象，递归调用自己，构建一棵二叉树。构建的二叉树有两种规格，一种粒度为PARTSIZE(9500KB)大小，一种粒度为EMBLOCKSIZE(180K)大小

CAICHHashTree\* CAICHHashTree::FindHash(uint64 nStartPos, uint64 nSize, uint8\* nLevel){

(\*nLevel)++; ///snow:初值为，依次递增

if (\*nLevel > 22){ // sanity ///snow:22的值应该是根据最大文件大小(256G)计算出来的

ASSERT( false );

return false;

}

if (nStartPos + nSize > m\_nDataSize){ // sanity

ASSERT ( false );

return NULL;

}

if (nSize > m\_nDataSize){ // sanity

ASSERT ( false );

return NULL;

}

if (nStartPos == 0 && nSize == m\_nDataSize){

// this is the searched hash

return this;

}

else if (m\_nDataSize <= GetBaseSize()){ // sanity

// this is already the last level, cant go deeper

ASSERT( false );

return NULL;

}

else{

uint64 nBlocks = m\_nDataSize / GetBaseSize() + ((m\_nDataSize % GetBaseSize() != 0 )? 1:0); ///snow:计算分块数

uint64 nLeft = ( ((m\_bIsLeftBranch) ? nBlocks+1:nBlocks) / 2)\* GetBaseSize(); ///snow:m\_bIsLeftBranch在构造时置为true，计算左边子树的字节数

uint64 nRight = m\_nDataSize - nLeft;

if (nStartPos < nLeft){

if (nStartPos + nSize > nLeft){ // sanity ///snow:最多==nLeft

ASSERT ( false );

return NULL;

}

if (m\_pLeftTree == NULL)

{

m\_pLeftTree = new CAICHHashTree(nLeft, true, (nLeft <= PARTSIZE) ? EMBLOCKSIZE : PARTSIZE);///snow:构造左子树,m\_nDataSize=nLeft，m\_bIsLeftBranch为true，分块大小依nLeft而定

}

else{

ASSERT( m\_pLeftTree->m\_nDataSize == nLeft );

}

return m\_pLeftTree->FindHash(nStartPos, nSize, nLevel); ///snow:nLevel已经在函数开头+1了

}

else{

nStartPos -= nLeft; ///snow:在右子树，起始位置要减去nLeft

if (nStartPos + nSize > nRight){ // sanity ///snow:最多也是==nRight

ASSERT ( false );

return NULL;

}

if (m\_pRightTree == NULL)

{

m\_pRightTree = new CAICHHashTree(nRight, false, (nRight <= PARTSIZE) ? EMBLOCKSIZE : PARTSIZE);///snow:构造右子树,m\_nDataSize=nRight，m\_bIsLeftBranch为false，分块大小依nRight而定

}

else{

ASSERT( m\_pRightTree->m\_nDataSize == nRight );

}

return m\_pRightTree->FindHash(nStartPos, nSize, nLevel);///snow:nLevel已经在函数开头+1了

}

}

}

## 4、CreateHash

///snow:按块生成MD4Hash和SHAHash，pFile中应该有一个文件位置指针，指示当前读取位置

///snow:CreateHash生成两种Hash，一种是MD4Hash，通过传出参数存入newhash，后面将加入m\_FileIdentifier.m\_aMD4HashSet

///snow:另一种是AICHHAsh，存放在pBlockAICHHashTree中，而pBlockAICHHashTree通过FindHash在cAICHHashSet中生成并定位对象

///snow:pBlockAICHHashTree在SetBlockHash()中通过FindHash()还继续生成新的子树

void CKnownFile::CreateHash(CFile\* pFile, uint64 Length, uchar\* pMd4HashOut, CAICHHashTree\* pShaHashOut)

{

///theApp.QueueTraceLogLine(TRACE\_AICHHASHTREE,\_T("%hs"),\_\_FUNCTION\_\_);///snow:add by snow

ASSERT( pFile != NULL );

ASSERT( pMd4HashOut != NULL || pShaHashOut != NULL ); ///snow:传出参数必须已分配内存

uint64 Required = Length; ///snow:PARTSIZE 9728000

uchar X[64\*128];

uint64 posCurrentEMBlock = 0;

uint64 nIACHPos = 0;

CMD4 md4;

CAICHHashAlgo\* pHashAlg = NULL;

if (pShaHashOut != NULL)

pHashAlg = CAICHRecoveryHashSet::GetNewHashAlgo(); ///snow:new CSHA()

while (Required >= 64){ ///snow:最后剩下的不足字节，因为Required=Required-(Required/64)\*64

uint32 len;

if ((Required / 64) > sizeof(X)/(64 \* sizeof(X[0]))) ///snow:Required/64>128

len = sizeof(X)/(64 \* sizeof(X[0])); ///snow:len=128

else

len = (uint32)Required / 64; ///snow:当Required<8192时，假设Required=6314，len=6314/64=98，剩下字节

pFile->Read(X, len\*64); ///snow:读取\*128(8K)字节到X，当len<128时，读取len\*64

///snow:AICHHash的处理

// SHA hash needs 180KB blocks

///snow:SHA hash需要KB，所以当数据不足KB时，先添加到pHashAlg中，当到达KB时，调用SetBlockHash，

if (pShaHashOut != NULL && pHashAlg != NULL){

if (nIACHPos + len\*64 >= EMBLOCKSIZE){ ///snow:达到KB了，在最后一轮时Required=143360，已经不能满足KB的要求了

uint32 nToComplete = (uint32)(EMBLOCKSIZE - nIACHPos);

pHashAlg->Add(X, nToComplete); ///snow:只从X中读取nToComplete，剩下(len\*64) - nToComplete字节

ASSERT( nIACHPos + nToComplete == EMBLOCKSIZE );

pShaHashOut->SetBlockHash(EMBLOCKSIZE, posCurrentEMBlock, pHashAlg); ///snow:调用SetBlockHash生成BLOCKHash

posCurrentEMBlock += EMBLOCKSIZE;

pHashAlg->Reset(); ///snow:重置pHashAlg

pHashAlg->Add(X+nToComplete,(len\*64) - nToComplete); ///snow:读取X中剩下的(len\*64) - nToComplete字节

nIACHPos = (len\*64) - nToComplete;

theApp.QueueTraceLogLine(TRACE\_AICHHASHTREE,\_T("Function:%hs|Line:%i|Required:%I64d,len:%i,sizeof(X):%i,sizeof(X[0]):%i,nToComplete:%i,posCurrentEMBlock:%I64d,nIACHPos:%I64d"),\_\_FUNCTION\_\_,\_\_LINE\_\_,Required,len,sizeof(X),sizeof(X[0]),nToComplete,posCurrentEMBlock,nIACHPos);///snow:add by snow

}

else{

pHashAlg->Add(X, len\*64); ///snow:暂存在pHashAlg，满K时进行SHAHASH

nIACHPos += len\*64;

theApp.QueueTraceLogLine(TRACE\_AICHHASHTREE,\_T("Function:%hs|Line:%i|Required:%I64d,len:%i,sizeof(X):%i,sizeof(X[0]):%i,posCurrentEMBlock:%I64d,nIACHPos:%I64d"),\_\_FUNCTION\_\_,\_\_LINE\_\_,Required,len,sizeof(X),sizeof(X[0]),posCurrentEMBlock,nIACHPos);///snow:add by snow

}

}

///snow:MD4HAsh的处理

if (pMd4HashOut != NULL){

md4.Add(X, len\*64);

}

Required -= len\*64;

}

///snow:剩下不足字节的部分

///snow:1、这部分只处理AICHHASH

Required = Length % 64;

if (Required != 0){

pFile->Read(X, (uint32)Required);

if (pShaHashOut != NULL){

///snow:最后的部分如果能够构成一个EMBLOCKSIZE分块

if (nIACHPos + Required >= EMBLOCKSIZE){

uint32 nToComplete = (uint32)(EMBLOCKSIZE - nIACHPos);

pHashAlg->Add(X, nToComplete);

ASSERT( nIACHPos + nToComplete == EMBLOCKSIZE );

pShaHashOut->SetBlockHash(EMBLOCKSIZE, posCurrentEMBlock, pHashAlg);

posCurrentEMBlock += EMBLOCKSIZE;

pHashAlg->Reset();

pHashAlg->Add(X+nToComplete, (uint32)(Required - nToComplete));

nIACHPos = Required - nToComplete;

theApp.QueueTraceLogLine(TRACE\_AICHHASHTREE,\_T("Function:%hs|Line:%i|Required:%I64d,nToComplete:%i,posCurrentEMBlock:%I64d,nIACHPos:%I64d"),\_\_FUNCTION\_\_,\_\_LINE\_\_,Required,nToComplete,posCurrentEMBlock,nIACHPos);///snow:add by snow

}

else{

pHashAlg->Add(X, (uint32)Required);

nIACHPos += Required;

theApp.QueueTraceLogLine(TRACE\_AICHHASHTREE,\_T("Function:%hs|Line:%i|Required:%I64d,posCurrentEMBlock:%I64d,nIACHPos:%I64d"),\_\_FUNCTION\_\_,\_\_LINE\_\_,Required,posCurrentEMBlock,nIACHPos);///snow:add by snow

}

}

}

///snow:剩下不足形成一个EMBLOCKSIZE分块部分的Hash处理

if (pShaHashOut != NULL){

if(nIACHPos > 0){

pShaHashOut->SetBlockHash(nIACHPos, posCurrentEMBlock, pHashAlg);

posCurrentEMBlock += nIACHPos;

}

ASSERT( posCurrentEMBlock == Length );

VERIFY( pShaHashOut->ReCalculateHash(pHashAlg, false) );

}

///snow:2、通过md4生成MD4hash，赋值给pMd4HashOut

if (pMd4HashOut != NULL){

md4.Add(X, (uint32)Required);

md4.Finish();

md4cpy(pMd4HashOut, md4.GetHash());

}

delete pHashAlg;

}

## 5、SetBlockHash

pBlockAICHHashTree在SetBlockHash()中通过FindHash()还继续生成新的子树，以180K为单位

void CAICHHashTree::SetBlockHash(uint64 nSize, uint64 nStartPos, CAICHHashAlgo\* pHashAlg){

ASSERT ( nSize <= EMBLOCKSIZE );

CAICHHashTree\* pToInsert = FindHash(nStartPos, nSize);

if (pToInsert == NULL){ // sanity

ASSERT ( false );

theApp.QueueDebugLogLine(/\*DLP\_VERYHIGH,\*/ false, \_T("Critical Error: Failed to Insert SHA-HashBlock, FindHash() failed!"));

return;

}

//sanity

if (pToInsert->GetBaseSize() != EMBLOCKSIZE || pToInsert->m\_nDataSize != nSize){

ASSERT ( false );

theApp.QueueDebugLogLine(/\*DLP\_VERYHIGH,\*/ false, \_T("Critical Error: Logical error on values in SetBlockHashFromData"));

return;

}

pHashAlg->Finish(pToInsert->m\_Hash); ///snow:将生成的BlockHash添加到pHashAlg中

pToInsert->m\_bHashValid = true;

//DEBUG\_ONLY(theApp.QueueDebugLogLine(/\*DLP\_VERYLOW,\*/ false, \_T("Set ShaHash for block %u - %u (%u Bytes) to %s"), nStartPos, nStartPos + nSize, nSize, pToInsert->m\_Hash.GetString()) );

}

## 6、SaveHashSet

将计算出来的文件HashSet写入Known2\_64.met

bool CAICHRecoveryHashSet::SaveHashSet(){

if (m\_eStatus != AICH\_HASHSETCOMPLETE){

ASSERT( false );

return false;

}

if ( !m\_pHashTree.m\_bHashValid || m\_pHashTree.m\_nDataSize != m\_pOwner->GetFileSize()){

ASSERT( false );

return false;

}

CSingleLock lockKnown2Met(&m\_mutKnown2File, false);

if (!lockKnown2Met.Lock(5000)){

return false;

}

///snow:CAICHSyncThread::Run()中会新建KNOWN2\_MET\_FILENAME，并写入文件头信息

CString fullpath = thePrefs.GetMuleDirectory(EMULE\_CONFIGDIR);

fullpath.Append(KNOWN2\_MET\_FILENAME);

CSafeFile file;

CFileException fexp;

if (!file.Open(fullpath,CFile::modeCreate|CFile::modeReadWrite|CFile::modeNoTruncate|CFile::osSequentialScan|CFile::typeBinary|CFile::shareDenyNone, &fexp)){

if (fexp.m\_cause != CFileException::fileNotFound){

CString strError(\_T("Failed to load ") KNOWN2\_MET\_FILENAME \_T(" file"));

TCHAR szError[MAX\_CFEXP\_ERRORMSG];

if (fexp.GetErrorMessage(szError, ARRSIZE(szError))){

strError += \_T(" - ");

strError += szError;

}

theApp.QueueLogLine(true, \_T("%s"), strError);

}

return false;

}

try {

//setvbuf(file.m\_pStream, NULL, \_IOFBF, 16384);

uint8 header = file.ReadUInt8();

if (header != KNOWN2\_MET\_VERSION){

AfxThrowFileException(CFileException::endOfFile, 0, file.GetFileName());

}

///snow:AddStoredAICHHash()将为m\_liAICHHashsStored添加成员，下面的代码就将调用AddStoredAICHHash

// first we check if the hashset we want to write is already stored

if (m\_liAICHHashsStored.Find(m\_pHashTree.m\_Hash) != NULL)

{

theApp.QueueDebugLogLine(false, \_T("AICH Hashset to write should be already present in known2.met - %s"), m\_pHashTree.m\_Hash.GetString());

// this hashset if already available, no need to save it again

return true;

}

/\*CAICHHash CurrentHash;

while (file.GetPosition() < nExistingSize){

CurrentHash.Read(&file);

if (m\_pHashTree.m\_Hash == CurrentHash){

// this hashset if already available, no need to save it again

return true;

}

nHashCount = file.ReadUInt32();

if (file.GetPosition() + nHashCount\*HASHSIZE > nExistingSize){

AfxThrowFileException(CFileException::endOfFile, 0, file.GetFileName());

}

// skip the rest of this hashset

file.Seek(nHashCount\*HASHSIZE, CFile::current);

}\*/

// write hashset

uint32 nExistingSize = (UINT)file.GetLength(); ///snow:保存原文件长度，防止写入出错时恢复

file.SeekToEnd(); ///snow:定位到文件末尾

///snow:先写入文件总的Hash值m\_Hash，再根据文件长度计算出分块Hash数nHashCount，写入nHashCount，再调用WriteLowestLevelHashs写入分块Hash

m\_pHashTree.m\_Hash.Write(&file);

uint32 nHashCount = (uint32)((PARTSIZE/EMBLOCKSIZE + ((PARTSIZE % EMBLOCKSIZE != 0)? 1 : 0)) \* (m\_pHashTree.m\_nDataSize/PARTSIZE));

if (m\_pHashTree.m\_nDataSize % PARTSIZE != 0)

nHashCount += (uint32)((m\_pHashTree.m\_nDataSize % PARTSIZE)/EMBLOCKSIZE + (((m\_pHashTree.m\_nDataSize % PARTSIZE) % EMBLOCKSIZE != 0)? 1 : 0));

file.WriteUInt32(nHashCount);

///snow:所有的分块Hash均存放在二叉树的最低一层即叶子节点中

if (!m\_pHashTree.WriteLowestLevelHashs(&file, 0, true, true)){

// thats bad... really

file.SetLength(nExistingSize);

theApp.QueueDebugLogLine(true, \_T("Failed to save HashSet: WriteLowestLevelHashs() failed!"));

return false;

}

if (file.GetLength() != nExistingSize + (nHashCount+1)\*HASHSIZE + 4){

// thats even worse

file.SetLength(nExistingSize);

theApp.QueueDebugLogLine(true, \_T("Failed to save HashSet: Calculated and real size of hashset differ!"));

return false;

}

///snow:写入文件HAsh成功了，将m\_Hash存入m\_liAICHHashsStored

CAICHRecoveryHashSet::AddStoredAICHHash(m\_pHashTree.m\_Hash);

theApp.QueueDebugLogLine(false, \_T("Successfully saved eMuleAC Hashset, %u Hashs + 1 Masterhash written"), nHashCount);

file.Flush();

file.Close();

}

catch(CFileException\* error){

if (error->m\_cause == CFileException::endOfFile)

theApp.QueueLogLine(true, GetResString(IDS\_ERR\_MET\_BAD), KNOWN2\_MET\_FILENAME);

else{

TCHAR buffer[MAX\_CFEXP\_ERRORMSG];

error->GetErrorMessage(buffer, ARRSIZE(buffer));

theApp.QueueLogLine(true,GetResString(IDS\_ERR\_SERVERMET\_UNKNOWN),buffer);

}

error->Delete();

return false;

}

FreeHashSet();

return true;

}

# 文件搜索过程

## 1、OnBnClickedStart

void CSearchParamsWnd::OnBnClickedStart()

{

m\_ctlMore.EnableWindow(FALSE);

if (m\_ctlOpts.GetEditCtrl()->GetSafeHwnd())

m\_ctlOpts.CommitEditCtrl();

SSearchParams\* pParams = GetParameters();

if (pParams)

{

if (!pParams->strExpression.IsEmpty())

{

if (m\_pacSearchString && m\_pacSearchString->IsBound())

m\_pacSearchString->AddItem(pParams->strExpression, 0);

m\_searchdlg->StartSearch(pParams);

}

else

delete pParams;

}

}

## 2、StartSearch

void CSearchResultsWnd::StartSearch(SSearchParams\* pParams)

{

switch (pParams->eType)

{

case SearchTypeAutomatic:

case SearchTypeEd2kServer:

case SearchTypeEd2kGlobal:

case SearchTypeKademlia:

StartNewSearch(pParams);

break;

case SearchTypeFileDonkey:

ShellOpenFile(CreateWebQuery(pParams));

delete pParams;

return;

default:

ASSERT(0);

delete pParams;

}

}

## 3、StartNewSearch

bool CSearchResultsWnd::StartNewSearch(SSearchParams\* pParams)

{

if (pParams->eType == SearchTypeAutomatic){ ///snow:自动情况下的处理

// select between kad and server

// its easy if we are connected to one network only anyway

if (!theApp.serverconnect->IsConnected() && Kademlia::CKademlia::IsRunning() && Kademlia::CKademlia::IsConnected()) ///snow:Kad连接，server未连接

pParams->eType = SearchTypeKademlia;

else if (theApp.serverconnect->IsConnected() && (!Kademlia::CKademlia::IsRunning() || !Kademlia::CKademlia::IsConnected())) ///snow:server连接,Kad未连接

pParams->eType = SearchTypeEd2kServer;

else if (!theApp.serverconnect->IsConnected() && (!Kademlia::CKademlia::IsRunning() || !Kademlia::CKademlia::IsConnected())){ ///snow:都未连接

AfxMessageBox(GetResString(IDS\_NOTCONNECTEDANY), MB\_ICONWARNING);

delete pParams;

return false;

}

else { ///snow:KAD和SERVER两者都连接，当连接的服务器是静态的，或者服务器连接的客户端超过万少于万，文件数大于万，服务器列表的服务器少于个，使用server搜索；否则，进行KAD搜索

// connected to both

// We choose Kad, except

// - if we are connected to a static server

// - or a server with more than 40k and less than 2mio users connected, more than 5 mio files and if our serverlist contains less than

// 40 servers (otherwise we have assume that its polluted with fake server and we might just as well be connected to one)

// might be further optmized in the future

if (theApp.serverconnect->IsConnected() && theApp.serverconnect->GetCurrentServer() != NULL

&& (theApp.serverconnect->GetCurrentServer()->IsStaticMember()

|| (theApp.serverconnect->GetCurrentServer()->GetUsers() > 40000 && theApp.serverlist->GetServerCount() < 40

&& theApp.serverconnect->GetCurrentServer()->GetUsers() < 5000000

&& theApp.serverconnect->GetCurrentServer()->GetFiles() > 5000000)))

{

pParams->eType = SearchTypeEd2kServer;

}

else

pParams->eType = SearchTypeKademlia;

}

}

ESearchType eSearchType = pParams->eType;

if (eSearchType == SearchTypeEd2kServer || eSearchType == SearchTypeEd2kGlobal) ///snow:Ed2k服务器搜索或全局服务器搜索

{

if (!theApp.serverconnect->IsConnected()) {

AfxMessageBox(GetResString(IDS\_ERR\_NOTCONNECTED), MB\_ICONWARNING);

delete pParams;

//if (!theApp.serverconnect->IsConnecting() && !theApp.serverconnect->IsConnected())

// theApp.serverconnect->ConnectToAnyServer();

return false;

}

try

{

if (!DoNewEd2kSearch(pParams)) { ///snow:启动server搜索

delete pParams;

return false;

}

}

catch (CMsgBoxException\* ex)

{

AfxMessageBox(ex->m\_strMsg, ex->m\_uType, ex->m\_uHelpID);

ex->Delete();

delete pParams;

return false;

}

SearchStarted();

return true;

}

if (eSearchType == SearchTypeKademlia)

{

if (!Kademlia::CKademlia::IsRunning() || !Kademlia::CKademlia::IsConnected()) {

AfxMessageBox(GetResString(IDS\_ERR\_NOTCONNECTEDKAD), MB\_ICONWARNING);

delete pParams;

//if (!Kademlia::CKademlia::IsRunning())

// Kademlia::CKademlia::Start();

return false;

}

try

{

if (!DoNewKadSearch(pParams)) { ///snow:启动Kad搜索

delete pParams;

return false;

}

}

catch (CMsgBoxException\* ex)

{

AfxMessageBox(ex->m\_strMsg, ex->m\_uType, ex->m\_uHelpID);

ex->Delete();

delete pParams;

return false;

}

SearchStarted();///snow:更新界面

return true;

}

ASSERT(0);

delete pParams;

return false;

}