探索 EVERYTHING 背后的技术

BY J Zhao

WHY EVERYTHING FAST ?

- 获取硬盘上所有文件的方式（想想以前的方式）
- 监视文件系统变化的方式
- 并不需要优秀的搜索算法

REFERENCE

- Everything 研究之读取NTFS下的USN日志文件
- Everything 桌面搜索原理揭秘
- Reading MFT @ Microsoft Develop Network
- Fsutil usn 命令使用帮助
- 从 frn 到 full path 的转换, 非公开的 win32 api
- Keeping an Eye on Your NTFS Drives: the Windows 2000 Change Journal Explained
- Keeping an Eye on Your NTFS Drives, Part II: Building a Change Journal Application

CHANGE JOURNAL ON NTFS 5.0

① On an NTFS volume, file and directory information is stored in the Master File Table (**MFT**).

② With NTFS 5.0, each file's MFT entry records the Last **USN** generated for that file.

③ Whenever the file system makes a change to a file or directory, it appends a record to the journal. The record identifies the file name, the time of the change, and what type of change occurred.

INTRODUCE USN

- The Change Journal is initially an empty file on the disk volume. As changes occur to the volume, records are appended to the end of this file. Each record is assigned a 64-bit identifier called an Update Sequence Number (USN).

JOURNAL 和 USN 的结构

```
typedef struct {
  DWORDLONG UsnJournalID;
  USN FirstUsn;
  USN NextUsn;
  USN LowestValidUsn;
  USN MaxUsn;
  DWORDLONG MaximumSize;
  DWORDLONG AllocationDelta;
} USN_JOURNAL_DATA, *PUSN_JOURNAL_DATA;
```

```
typedef struct {
DWORD         RecordLength;
WORD          MajorVersion;
WORD          MinorVersion;
DWORDLONG     FileReferenceNumber;
DWORDLONG     ParentFileReferenceNumber;
USN Usn;
LARGE_INTEGER TimeStamp;
DWORD         Reason;
DWORD         SourceInfo;
DWORD         SecurityId;
DWORD         FileAttributes;
WORD          FileNameLength;
WORD          FileNameOffset;
WCHAR         FileName[1];
} USN_RECORD, *PUSN_RECORD;
```

怎么看待MFT的CHANGE JOURNAL(个人观点)

本质: MFT 的 Change Journal 数据结构仅仅是在MFT结构上的每条数据增加了一个USN字段而已.

实际: 基于Change Journal的Win32 API为我们带来了两种新功能:

① 顺序遍历MFT上的每条数据, 从而快速获得某个Volume上的所有文件和文件夹的名字

② 通过等待Change Journal的下一条记录来做到对NTFS文件系统变化的监视

CODE FRAMEWORK

- 得到 Volume 的句柄 (via CreateFile 函数)
- 打开 Volume 上的 Change Journal
- 针对不同的操作(遍历, 筛选, 等待新记录等等)构造输入参数的结构体
- 调用 DeviceIoControl 函数
- 分析输出结果(通常是一片内存块)

DEVICEIOCONTROL 函数

```
BOOL DeviceIoControl(
  HANDLE hDevice,          // handle to device or file or directory
  DWORD dwIoControlCode,   // control code of operation to perform
  LPVOID lpInBuffer,       // pointer to buffer of input data
  DWORD nInBufferSize,     // size, in bytes, of input buffer
  LPVOID lpOutBuffer,      // pointer to buffer for output data
  DWORD nOutBufferSize,    // size, in bytes, of output buffer
  LPDWORD lpBytesReturned, // receives number of bytes written to lpOutBuffer
  LPOVERLAPPED lpOverlapped// for asynchronous operation
);
```

操作控制码举例

- FSCTL_CREATE_USN_JOURNAL // 打开日志
- FSCTL_READ_FILE_USN_DATA
  FSCTL_WRITE_USN_CLOSE_RECORD // 关闭日志
- FSCTL_QUERY_USN_JOURNAL // 查询日志的状态
- FSCTL_DELETE_USN_JOURNAL
- FSCTL_ENUM_USN_DATA // 遍历MFT上的每条记录
- FSCTL_READ_USN_JOURNAL // 读取指定编号的记录

```
eg: DeviceIoControl( hVol, FSCTL_QUERY_USN_JOURNAL, NULL,
                     0, &qujd, sizeof(qujd), &br, NULL );
```

遍历指定 VOLUME 上的所有文件(夹)名

```
unsafe private void EnumerateFiles(IntPtr pVolume, IntPtr medBuffer, ref List<JRecord> files, ref List<JRecord> dirs) {
    IntPtr pData = Marshal.AllocHGlobal(sizeof(UInt64) + 0x10000);
    PInvokeWin32.ZeroMemory(pData, sizeof(UInt64) + 0x10000);
    uint outBytesReturned = 0;

    while (false != PInvokeWin32.DeviceIoControl(pVolume, PInvokeWin32.FSCTL_ENUM_USN_DATA, medBuffer,
                        sizeof(PInvokeWin32.MFT_ENUM_DATA), pData, sizeof(UInt64) + 0x10000, out outBytesReturned,
                        IntPtr.Zero)) {
        IntPtr pUsnRecord = new IntPtr(pData.ToInt32() + sizeof(Int64));
        while (outBytesReturned > 60) {
            PInvokeWin32.USN_RECORD usn = new PInvokeWin32.USN_RECORD(pUsnRecord);

            if (0 != (usn.FileAttributes & PInvokeWin32.FILE_ATTRIBUTE_DIRECTORY)) {
                dirs.Add(new JRecord {
                    Name = usn.FileName,
                    ParentFrn = usn.ParentFileReferenceNumber,
                    Frn = usn.FileReferenceNumber
                });
            } else {
                files.Add(new JRecord {
                    Name = usn.FileName,
                    ParentFrn = usn.ParentFileReferenceNumber,
                    Frn = usn.FileReferenceNumber
                });
            }

            pUsnRecord = new IntPtr(pUsnRecord.ToInt32() + usn.RecordLength);
            outBytesReturned -= usn.RecordLength;
        }
        Marshal.WriteInt64(medBuffer, Marshal.ReadInt64(pData, 0));
    }
    Marshal.FreeHGlobal(pData);
}
```

MISS FULL PATH !

- 仔细观察 USN_RECORD 结构体

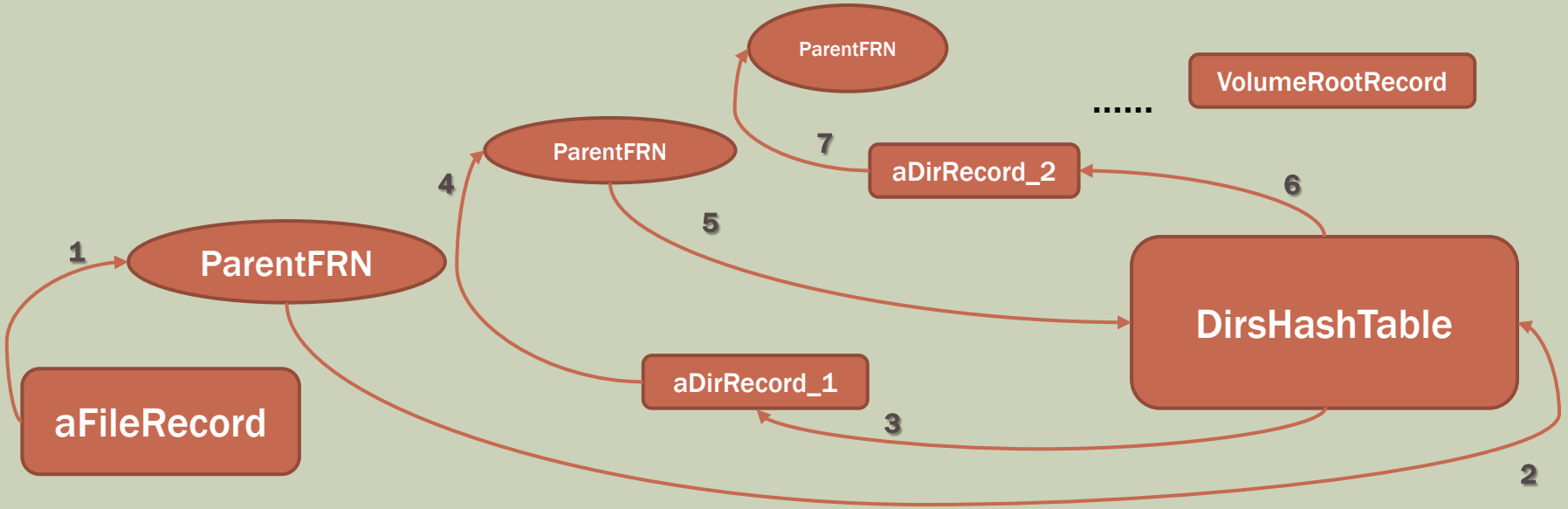
```
typedef struct {
DWORD         RecordLength;
WORD          MajorVersion;
WORD          MinorVersion;
DWORDLONG     FileReferenceNumber;
DWORDLONG     ParentFileReferenceNumber;
USN Usn;
LARGE_INTEGER TimeStamp;
DWORD         Reason;
DWORD         SourceInfo;
DWORD         SecurityId;
DWORD         FileAttributes;
WORD          FileNameLength;
WORD          FileNameOffset;
WCHAR         FileName[1];
} USN_RECORD, *PUSN_RECORD;
```

- The full path of a record is not stored in the record itself.

CONVERT TO FULL PATH

① 利用 ntdll.dll 中非公开的方法

② 利用ParentFileReferenceNumber递归出 fullpath



Full Path = VolumeRootRecord + ... + aDirRecord_2.Name + aDirRecord_1.Name + aFileRecord.Name

监视 NTFS 文件系统的变化

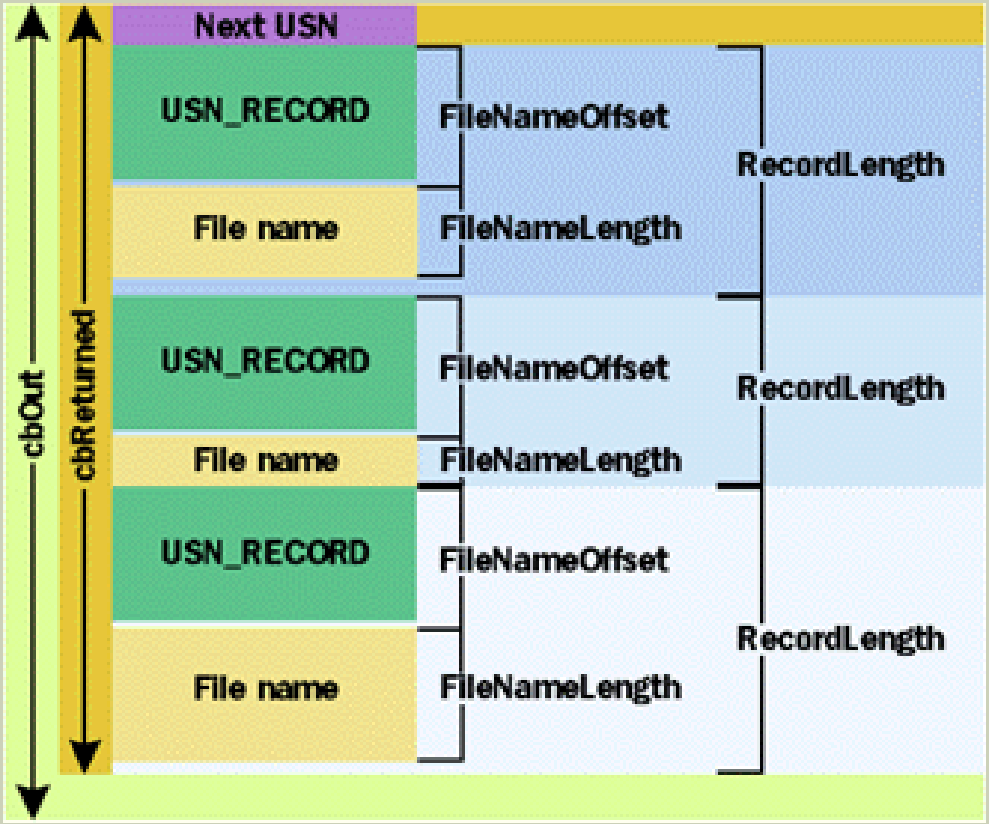
① 通过FSCTL_QUERY_USN_JOURNAL操作得到NextUsn
② 构造输入参数

```
typedef struct {
 USN StartUsn; // 使用NextUsn填充
 DWORD ReasonMask; // 填充你关心的 Reason
 DWORD ReturnOnlyOnClose;
 DWORDLONG Timeout;
 DWORDLONG BytesToWaitFor; // 设置为非零成为同步调用模式
 DWORDLONG UsnJournalID;
} READ_USN_JOURNAL_DATA, *PREAD_USN_JOURNAL_DATA;
```

③ 调用 DeviceIoControl 函数
④ DeviceIoControl会以阻塞，超时或不阻塞等各种方式来工作
⑤ 获得返回值中的NextUSN的值充当 StartUsn 构建输入参数
⑥ 重复步骤4和5

监视 NTFS 文件系统的变化 (续)

- 输出参数的结构:



持续运行

① 将硬盘上的文件和文件夹的名字全部获取
② 将文件和文件夹的名字转换成 full path 以进一步获取信息
③ 保存上述信息至自定义 db
④ 监视 volumes 上的变化
⑤ 根据变化类型(Reason)增删改 db 中对应的数据
⑥ 重复步骤4和步骤5

- 那如果程序被关闭了呢?