SEH

结构化异常处理，与函数相关

`/EHa`编译选项开启异步异常处理（Debug版默认开启）

高级语言中异常是不可修复的，一般用来体面退出的

异常分派的顺序：SEH、筛选器、系统，之前会给调试器

SEH的分类

1. Per-Thread类型SEH（也称为**线程异常处理**），用来监视某线程代码是否发生异常。
2. Final类型SEH（也称为进程异常处理、筛选器或**顶层异常处理**），用于监视整个进程中所有线程是否发生异常。在整个进程中，该类型异常处理过程只有一个，可通过`SetUnhandledExceptionFilter`设置。

相关数据结构

TEB

```
struct _TEB // fs段存放_TEB
{
    struct _NT_TIB; // +0
    // ...
};
```

线程信息块TIB

```
typedef struct _NT_TIB {
     struct _EXCEPTION_REGISTRATION_RECORD *ExceptionList; // +0 异常处理的链表

     PVOID StackBase;
     PVOID StackLimit;
     PVOID SubSystemTib;

     union {
         PVOID FiberData;
         DWORD Version;
     };

     PVOID ArbitraryUserPointer;
     struct _NT_TIB *Self;
} NT_TIB;
```

`Fs:[0]`总是指向TEB，即总是指向当前线程的TIB，其中0偏移的指向线程的异常链表，即`ExceptionList`是指向异常处理链表（EXCEPTION_REGISTRATION结构）的一个指针。

EXCEPTION_REGISTRATION结构

```
typedef struct _EXCEPTION_REGISTRATION_RECORD {
     struct _EXCEPTION_REGISTRATION_RECORD *Prev; //指向前一个
EXCEPTION_REGISTRATION的指针
     PEXCEPTION_ROUTINE Handler; //当前异常处理回调函数的地址
} EXCEPTION_REGISTRATION_RECORD;
```

EXCEPTION_RECORD结构

```
typedef struct _EXCEPTION_RECORD {
    DWORD    ExceptionCode;  //异常码，以STATUS_或EXCEPTION_开头，可自定义。
（sehdef.inc）
    DWORD ExceptionFlags;    //异常标志。0可修复；1不可修复；2正在展开，不要试图修复
    struct _EXCEPTION_RECORD *ExceptionRecord; //指向嵌套的异常结构，通常是异常中又引
发异常
    PVOID ExceptionAddress;  //异常发生的地址
    DWORD NumberParameters;  //下面ExceptionInformation所含有的dword数目
    ULONG_PTR ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS]; //附加消息，如读
或写冲突
} EXCEPTION_RECORD;
```

CONTEXT结构

```
typedef struct _CONTEXT {
     DWORD ContextFlags; //用来表示该结构中的哪些域有效
     DWORD   Dr0, Dr2, Dr3, Dr4, Dr5, Dr6, Dr7; //调试寄存器
     FLOATING_SAVE_AREA FloatSave; //浮点寄存器区
     DWORD   SegGs, SegFs, SegEs, Seg Ds; //段寄存器
     DWORD   Edi, Esi, Ebx, Edx, Ecx, Eax; //通用寄存器组
     DWORD   Ebp, Eip, SegCs, EFlags, Esp, SegSs; //控制寄存器组

     //扩展寄存器，只有特定的处理器才有
     BYTE    ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION];
} CONTEXT;
```

线程异常处理

局部的，仅仅监视进程中某特定线程是否发生异常

异常处理回调函数

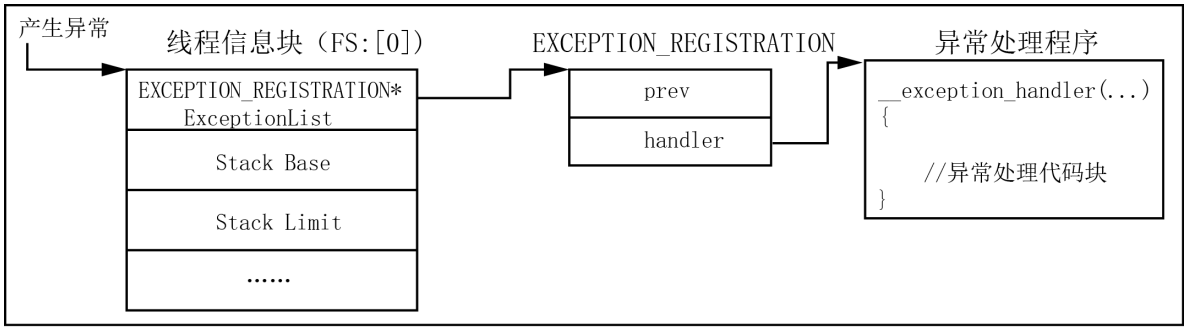
```
EXCEPTION_DISPOSITION __cdecl _except_handler(

     struct _EXCEPTION_RECORD *ExceptionRecord, // 指向包含异常信息的
EXCEPTION_RECORD结构
     void* EstablisherFrame,     // 指向该异常相关的EXCEPTION_REGISTRATION结构
     struct _CONTEXT *ContextRecord,     // 指向线程环境CONTEXT结构的指针
```

```
    void* DispatcherContext) {      // 该域暂无意义

    ...

    //4种返回值及含义
    //1.ExceptionContinueExecution(0)：回调函数处理了异常，可以从异常发生的指令处重新执行。
    //2.ExceptionContinueSearch(1)：回调函数不能处理该异常，需要要SEH链中的其他回调函数处理。
    //3.ExceptionNestedException(2)：回调函数在执行中又发生了新的异常，即发生了嵌套异常
    //4.ExceptionCollidedUnwind(3)：发生了嵌套的展开操作

    return ...
}
```

线程异常处理的注册和卸载

```
assume fs:nothings  ; fs不是任何指针

fun proc
    ; 开始压栈SEH结构（注册）
    push offset _exception_handler  ; 异常回调函数_exception_handler的地址，即handler
    push fs:[0]     ; 保存前一个异常回调函数的地址，即prev
    mov fs:[0], esp ; 安装新的EXCEPTION_REGISTRATION结构（两个成员:prev,handler）。
                    ; 此时栈顶分别是prev和handler，为新的EXCEPTION_REGISTRATION结构，
                    ; mov fs:[0],esp，就可以让fs:[0]指向该指构。

    ; ...

    ; 恢复（卸载）
    pop fs:[0]  ; 恢复前一个异常回调函数，即prev
    add esp, 4  ; 释放空间

    ret
fun endp
```

异常回调函数的调用过程



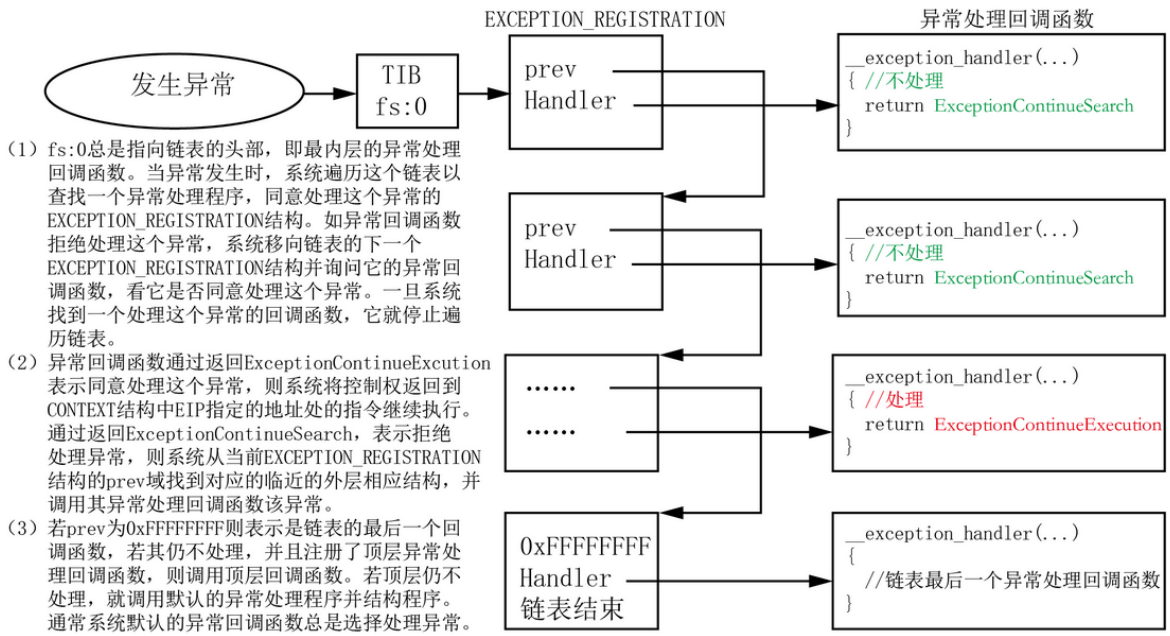
SEH异常处理程序入口地址的定义

1. 线程信息块（TIB），永远放在 `fs` 段选择器指定的数据段的0偏移处，即 `fs:[0]` 的地方就是TIB结构。对不同的线程 `fs` 寄存器的内容有所有不同，但 `fs:[0]` 都是指向当前线程的TIB结构体，所以

`fs:[0]`是一个`EXCEPTION_REGISTRATION`结构体的指针。

2. 当异常发生时，系统从`fs:[0]`指向的内存地址处取出`ExceptionList`字段，然后从`ExceptionList`字段指向的`EXCEPTION_REGISTRATION`结构中取出`handler`字段，并根据其中的地址去调用异常处理程序（回调函数）。

SEH链及异常的传递



异常嵌套形成的SEH链及异常的传递

1. 系统查看产生异常的进程是否被正在被调试，如果正在被调试，那么向调试器发送EXCEPTION_DEBUG_EVENT事件。
2. 如果进程没有没有被调试或者调试器不去处理这个异常，那么系统检查异常所处的线程并在这个线程环境中查看`fs:[0]`来确定是否安装SEH异常处理回调函数，如果有则调用它。
3. 回调函数尝试处理这个异常，如果可以正确处理的话，则修正错误并将返回值设置为`ExceptionContinueExecution`，这时系统将结束整个查找过程。
4. 如果回调函数返回`ExceptionContinueSearch`，相当于告诉系统它无法处理这个异常，系统将根据SEH链中的`prev`字段得到前一个回调函数地址并重复步骤3，直至链中的某个回调函数返回`ExceptionContinueExection`为止，查找结束。
5. 如果到了SEH链的尾部却没有一个回调函数愿意处理这个异常，那么系统会再被检查进程是否正在被调试，如果被调试的话，则再一次通知调试器。
6. 如果调试器还是不去处理这个异常或进程没有被调试，那么系统检查有没有**Final**型的异常处理回调函数，如果有，就去调用它，当这个回调函数返回时，系统会根据这个函数的返回值做相应的动作。
7. 如果没有安装**Final**型回调函数，系统直接调用默认的异常处理程序终止进程，但在终止之前，系统再次调用发生异常的线程中的所有异常处理过程，目的是让线程异常处理过程获得最后清理未释放资源的机会，其后程序终止。

异常的展开

> **（作为逆向来说知道有这么个东西就行）**

脱SEH链的操作叫做异常展开。

在异常处理程序中如果处理了异常则必须负责把Fs:[0]恢复到处理这个异常的`EXCEPTION_REGISTRATION`上，即展开操作导致堆栈上处理异常的帧以下的堆栈区域上的所有内容都被移除了，这个异常处理也就成了SEH链表的第1个节点。

```
NTSYSAPI VOID RtlUnwind(
  PVOID             TargetFrame,     // 当遍历到这个帧时就停止展开异常帧。为NULL时表示展开所有回调函数
  PVOID             TargetIp,    // 指向该函数返回的位置。如果指定为NULL，表示函数使用正常的方式返回
  PEXCEPTION_RECORD ExceptionRecord,     // 一般建议使用NULL
  PVOID             ReturnValue // 一般不被使用
);
```