# 异常处理

`SetUnhandledExceptionFilter`当发生异常时，此函数的回调函数会被触发接管异常

```
LPTOP_LEVEL_EXCEPTION_FILTER SetUnhandledExceptionFilter(
  LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter    // 异常处理回调函数
);
```

回调函数

```
LONG UnhandledExceptionFilter(
  _EXCEPTION_POINTERS *ExceptionInfo
);
// 返回值
// EXCEPTION_EXECUTE_HANDLER = 1，异常已经处理
// EXCEPTION_CONTINUE_SEARCH = 0，异常要交给上级处理
// EXCEPTION_CONTINUE_EXECUTION = -1，异常已经处理了且可以继续执行
```

参数结构

```
typedef struct _EXCEPTION_POINTERS {
  PEXCEPTION_RECORD ExceptionRecord;    // 异常记录
  PCONTEXT          ContextRecord;      // context记录
} EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;
```

# 关于exception

```
typedef struct _EXCEPTION_RECORD {
  DWORD                    ExceptionCode;   // 异常编码
  DWORD                    ExceptionFlags;
  struct _EXCEPTION_RECORD *ExceptionRecord;    // 构成链表，后续异常记录
  PVOID                    ExceptionAddress;    // 异常发生的地址
  DWORD                    NumberParameters;    // 后续ExceptionInformation有几
项，柔性数组
  ULONG_PTR                ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
// 只对访问类异常才有意义，第一个元素标明读/写（0/1），第二个元素标明访问的目标地址
} EXCEPTION_RECORD;
```

# 关于context

```
typedef struct _CONTEXT {

    //
    // 这个标志值控制上下文记录的内容
    //
```

```
    // 如果上下文记录作为输入参数使用，
    // 那么每个上下文记录的部分被设置的标志值控制，
    // 它假定上下文记录部分包含有效的上下文。
    // 如果该上下文记录正在用来修改线程上下文，
    // 那么仅有该线程上下文的部分将被修改。
    //
    // 如果上下文记录作为输入输出参数来捕获线程上下文，
    // 那么只有那些与设置的标志相对应的线程的上下文部分将会被返回。
    //
    // 上下文记录从来都不仅作为输出参数使用。
    //

    DWORD ContextFlags;

    //
    // 如果 ContextFlags 设置为 CONTEXT_DEBUG_REGISTERS，
    // 这个部分被指定或返回。
    // 注意：CONTEXT_DEBUG_REGISTERS 没有被包含在 CONTEXT_FULL 中。
    //

    DWORD   Dr0;
    DWORD   Dr1;
    DWORD   Dr2;
    DWORD   Dr3;
    DWORD   Dr6;
    DWORD   Dr7;

    //
    // 如果 ContextFlags 设置为 CONTEXT_FLOATING_POINT，
    // 这个部分被指定/返回。
    //

    FLOATING_SAVE_AREA FloatSave;

    //
    // 如果 ContextFlags 字包含 CONTEXT_SEGMENTS，
    // 这个部分被指定/返回。
    //

    DWORD   SegGs;
    DWORD   SegFs;
    DWORD   SegEs;
    DWORD   SegDs;

    //
    // 如果 ContextFlags 字包含 CONTEXT_INTEGER，
    // 这个部分被指定/返回。
    //

    DWORD   Edi;
    DWORD   Esi;
    DWORD   Ebx;
    DWORD   Edx;
    DWORD   Ecx;
    DWORD   Eax;

    //
    // 如果 ContextFlags 字包含 CONTEXT_CONTROL，
```

```
    // 这个部分被指定/返回。
    //

    DWORD   Ebp;
    DWORD   Eip;
    DWORD   SegCs;              // MUST BE SANITIZED
    DWORD   EFlags;             // MUST BE SANITIZED
    DWORD   Esp;
    DWORD   SegSs;

    //
    // 如果 ContextFlags 字包含 CONTEXT_EXTENDED_REGISTERS，
    // 这个部分被指定/返回，且这个格式和上下文是特殊处理器。
    //

    BYTE    ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION];

} CONTEXT;
```

`GetThreadContext`、`SetThreadContext`可以获取/设置context环境

```
BOOL SetThreadContext(
HANDLE hThread, //目标线程的句柄
CONST CONTEXT * lpContext //Context指针);

BOOL GetThreadContext(
HANDLE hThread, //目标线程的句柄
CONST CONTEXT * lpContext //Context指针);
```

不过在获取和设置时需要记得`SuspendThread`挂起线程，设置完成后才`ResumeThread`激活线程