开始使用

PEView 是一款基于 C/C++ 开发的命令行交互式 Windows PE 文件解析器，主要用于病毒木马等样本的解包分析，经过多版本迭代，现已能解析 32/64 位可执行程序的绝大部分通用参数，还内置结构查询、格式转换等功能，为逆向分析提供基础支撑；而在其基础上升级延伸而来的 PEViewAI，则通过将 PEView 的功能接口化并对接 Ollama 大模型，让大模型具备分析磁盘静态文件的能力，既能协助逆向分析与反病毒工程师掌握文件结构，又能通过对话式交互提升分析效率、快速生成静态分析报告，成为生产力工具的重要补充。

- 独立版：保留 PEView 默认的命令行分析功能，专注基础解析需求，确保工具核心能力的稳定性。
- 接口版：提供可调用的功能接口，支持通过 Python 语言调用，方便开发者灵活集成到自定义分析流程中。
- MCP 版：集成 AI 的服务端版本，可对接本地 Ollama 大模型或其他任意大模型，重点支持对话方式分析恶意样本，最大化 AI 赋能的价值。

本次更新针对不同场景推出三个版本，其中独立版保留默认命令行分析功能以满足基础需求，接口版本支持通过 Python 调用接口便于灵活集成，MCP 版作为融入 AI 的服务端，可对接本地 Ollama 或其他大模型，实现对话式分析本地样本，最大化 AI 赋能价值。

- 版权局登记号：2023SR0490503

独立版

独立版不仅保留了核心的交互式分析能力，更通过丰富的可选参数覆盖 PE 文件逆向分析全流程。其参数体系涵盖文件操作、结构查看、地址转换、搜索计算等多个维度，既支持打开 PE 文件、显示 DOS 头 / NT 头 / 节表等基础结构信息，也能完成导入表 / 导出表查询、VA 与 RVA/FOA 地址互转、十六进制计算等专业操作，还具备反汇编、特征码搜索、保护方式检查等进阶功能。

打开待分析文件

在进行文件相关操作时，需首先通过Open命令完成文件的打开操作，且该操作需一次性执行到位，不支持分步骤或重复打开同一文件的冗余操作。执行Open命令时，必须严格按照 “Open --path [文件路径]” 的格式进行，其中 “[文件路径]” 需替换为待操作文件的具体存储路径（例如本地磁盘路径如 e://win32.exe 等）。只有在文件成功通过Open命令打开后，后续针对该文件的各类分析、查看、解析等操作才能够正常执行，若未执行此打开步骤或打开失败，所有其他操作都将无法进行。

```
__ __       _____       _     _     __       _____     _       _
 /_/\__/\   /\_____\    /_/\ /\_\   /\_\   /\_____\  /_/\  /\_\
 ) ) ) ) ) ( (_____/    ) ) ) ( (   \/_/  ( (_____/  ) ) )( ( (
_/_/ /_/ /   \ \__\     /_/ / \ \_\   /\_\  \ \__\   /_/ //\\ \_\
\ \ \_\/    / /__/_    \ \ \_/ / /  / / /  / /__/_  \ \ /  \ / /
 )_) )     ( (_____\    \ \   / /  ( (_(  ( (_____\  )_) /\ (_(
 \_\/       \/_____/     \_\_/_/    \/_/   \/_____/  \_\/  \/_/

[编译日期] Oct 19 2025
[解析格式] Windows x86 (PE32)
[当前版本] 4.0.0
[官方网站] peview.lyshark.com

[PEVIEW] # Open --path e://win32.exe
[+] 已读入文件
```

```
[PEVIEW] #
[PEVIEW] # Info
-------------------------------------------------------------------------------------------------
--------
文件基本信息
-------------------------------------------------------------------------------------------------
--------
[文件路径]: e://win32.exe
[文件大小]: 14.50 KB
[文件属性]: 归档;
[创建时间]: 2025-10-17 20:46:43
[修改时间]: 2025-10-17 20:46:43
[映射基址]: 0x01540000
-------------------------------------------------------------------------------
PE结构标识
-------------------------------------------------------------------------------
[DOS签名]: 0x5A4D (有效DOS签名(MZ))
[PE头偏移]: 0x00000100 (从文件开始的偏移)
[NT签名]: 0x00004550 (有效PE签名(PE00))
[机器类型]: 0x014C (x86 (32位))
[节区数量]: 5 个
[时间戳]: 0x68F23AB3 (1601-01-01 08:02:56)
[特性标记]: 0x0102 (可执行; )
-------------------------------------------------------------------------------
可选头关键信息
-------------------------------------------------------------------------------
[入口点RVA]: 0x000015BB (程序开始执行的相对虚拟地址)
[镜像基址]: 0x00400000 (加载到内存中的首选基地址)
[图像大小]: 0x00007000 字节 (加载到内存后的总大小)
[节区对齐]: 0x00001000 字节 (内存中节区的对齐粒度)
[文件对齐]: 0x00000200 字节 (磁盘上节区的对齐粒度)
[子系统]: 0x0002 (Windows GUI (图形界面))
[DLL特性]: 0x8140 (ASLR支持; DEP支持; )
[栈大小]: 保留 0x00100000 字节, 提交 0x00001000 字节
[堆大小]: 保留 0x00100000 字节, 提交 0x00001000 字节
-------------------------------------------------------------------------------------------------
--------
```

查询文件头部数据

本次通过Dos和Nt命令查询了目标 PE 文件的头部数据：Dos命令获取了 DOS 头部的完整字段信息，涵盖 DOS 标志、文件页面相关参数、初始寄存器值、重定位表偏移及 PE 头偏移指针等，其中 DOS 标志 0x00005A4D 验证了 DOS 头部有效性，PE 头偏移指针 0x00000100 指明了 PE 头在文件中的位置；Nt命令则详细展示了 NT 头部数据，包括验证 PE 有效性的 NT 标志、描述文件基础信息的文件头（如运行平台 x86、区段数目 5 个、时间日期标志对应 2025 年 10 月 17 日创建时间等），以及包含入口点、镜像基址、内存与文件对齐粒度、子系统类型（Windows GUI）、DLL 特性（支持 ASLR 和 DEP）等关键配置的可选头信息，完整呈现了文件头部的核心结构与参数。

```
[PEVIEW] # Dos
-----------------------------------------------------------------------
                          十六进制          十进制
-----------------------------------------------------------------------
DOS标志(MZ):              00005A4D        00023117
```

```
文件最后一页的字节数:      00000090        00000144
文件中的页面数:            00000003        00000003
重定位项数量:              00000000        00000000
头部占用的段落数:          00000004        00000004
所需最小额外段落数:        00000000        00000000
所需最大额外段落数:        0000FFFF        00065535
初始SS值(相对):            00000000        00000000
初始SP值:                  000000B8        00000184
校验和:                    00000000        00000000
初始IP值:                  00000000        00000000
初始CS值(相对):            00000000        00000000
重定位表偏移:              00000040        00000064
叠加层数:                  00000000        00000000
保留字段e_res[0]:          00000000        00000000
保留字段e_res[1]:          00000000        00000000
保留字段e_res[2]:          00000000        00000000
保留字段e_res[3]:          00000000        00000000
OEM标识符:                 00000000        00000000
OEM信息:                   00000000        00000000
保留字段e_res2[0]:         00000000        00000000
保留字段e_res2[1]:         00000000        00000000
保留字段e_res2[2]:         00000000        00000000
保留字段e_res2[3]:         00000000        00000000
保留字段e_res2[4]:         00000000        00000000
保留字段e_res2[5]:         00000000        00000000
保留字段e_res2[6]:         00000000        00000000
保留字段e_res2[7]:         00000000        00000000
保留字段e_res2[8]:         00000000        00000000
保留字段e_res2[9]:         00000000        00000000
PE头偏移指针:              00000100        00000256
-------------------------------------------------------------------

[PEVIEW] # Nt
-------------------------------------------------------------------------------------------------
--------
                              十六进制         十进制           描述
-------------------------------------------------------------------------------------------------
--------
NT标志:                    0x00004550        00017744        有效PE签名(PE00)

[文件头(IMAGE_FILE_HEADER)]
运行平台:                  0x0000014C        00000332        x86 (32位)
区段数目:                  0x00000005        00000005        PE文件包含的区段数量
时间日期标志:              0x68F23AB3        1760705203      Fri Oct 17 20:46:43 2025
特征值:                    0x00000102        00000258        可执行文件;
可选头部大小:              0x000000E0        00000224        可选头的字节大小
符号数量:                  0x00000000        00000000        符号表中的符号数量
符号表指针:                0x00000000        00000000        符号表在文件中的偏移

[可选头(IMAGE_OPTIONAL_HEADER32)]
入口点(RVA):               0x000015BB        00005563        入口点虚拟地址(VA: 0x004015BB)
镜像基址:                  0x00400000        04194304        加载到内存的首选基地址
镜像大小:                  0x00007000        00028672        内存中整个镜像的大小(字节)
```

```
代码基址(RVA):          0x00001000          00004096          代码段的起始相对虚拟地址
数据基址(RVA):          0x00002000          00008192          数据段的起始相对虚拟地址
代码大小:               0x00000C00          00003072          代码段的总大小(字节)
已初始化数据大小:       0x00002E00          00011776          已初始化数据段的大小(字节)
未初始化数据大小:       0x00000000          00000000          未初始化数据段的大小(字节)
内存对齐:               0x00001000          00004096          内存中区块的对齐粒度(字节)
文件对齐:               0x00000200          00000512          文件中区块的对齐粒度(字节)
子系统:                 0x00000002          00000002          Windows GUI(图形界面)
首部大小:               0x00000400          00001024          DOS头+NT头+区段表的总大小
校验和:                 0x00000000          00000000          用于验证文件完整性(通常为0)
数据目录数量:           0x00000010          00000016          数据目录项的数量(通常为16)
链接器主版本:           0x0000000C          00000012
链接器次版本:           0x00000000          00000000
操作系统版本:           6.0                 -         主版本.6.次版本.0
映像版本:               0.0                 -         主版本.0.次版本.0
子系统版本:             6.0                 -         主版本.6.次版本.0
Win32版本值:            0x00000000          00000000          通常为0(保留)
DLL特征:                0x00008140          00033088          支持ASLR; 支持DEP;
栈保留大小:             0x00100000          01048576          进程栈的保留大小
栈提交大小:             0x00001000          00004096          进程栈的初始提交大小
堆保留大小:             0x00100000          01048576          进程堆的保留大小
堆提交大小:             0x00001000          00004096          进程堆的初始提交大小
加载器标志:             0x00000000          00000000          已废弃(通常为0)
-------------------------------------------------------------------------------------------
--------
```

查询数据目录表

通过执行DataDirectory命令查询到目标 PE 文件的数据目录表信息，共包含 16 个目录项。其中有效目录项（标记“√”）有 6 个，分别是导入表（Import Table）、资源表（Resource Table）、基址重定位表（Base Relocation Table）、调试表（Debug Table）、加载配置表（Load Configuration Table）和导入地址表（Import Address Table），涵盖了程序运行所需的导入函数解析、资源索引、地址修正、调试信息及加载配置等关键数据；其余 10 个目录项（如导出表、异常表、安全表等）均无效（标记“×”），无对应数据内容，整体数据目录表结构符合 PE 文件规范，有效项可支撑程序正常加载与运行。

```
[PEVIEW] # DataDirectory
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
编号       标准名称                  目录RVA          目录VA           目录FOA          Size(十进制)
Size(十六进制)          有效      详细描述
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
001      Export Table     0x00000000      0x00000000      0xFFFFFFFF      0000000000
0x00000000              ×       包含导出函数和符号信息，供其他模块调用
002      Import Table     0x000022A4      0x004022A4      0x000012A4      0000000080
0x00000050              √       包含导入的DLL和函数信息，需要在加载时解析
003      Resource Table   0x00004000      0x00404000      0x00001A00      0000007600
0x00001DB0              √       包含程序资源（图标、字符串、对话框等）的索引信息
004      Exception Table  0x00000000      0x00000000      0xFFFFFFFF      0000000000
0x00000000              ×       包含异常处理相关结构，用于异常捕获和处理
005      Security Table   0x00000000      0x00000000      0xFFFFFFFF      0000000000
0x00000000              ×       包含数字签名等安全信息，用于验证文件完整性
```

```
006      Base Relocation Table 0x00006000        0x00406000      0x00003800      0000000392
     0x00000188              √       包含基地址重定位信息，用于模块加载地址与默认不同时修正
007      Debug Table      0x00002110      0x00402110      0x00001110      0000000056
0x00000038              √       包含调试信息（如调试符号路径、类型等）
008      Architecture     0x00000000      0x00000000      0xFFFFFFFF      0000000000
0x00000000              ×       版权信息字符串（通常为Unicode格式）
009      Global Pointer   0x00000000      0x00000000      0xFFFFFFFF      0000000000
0x00000000              ×       全局指针（用于某些架构的全局变量访问）
010      TLS Table        0x00000000      0x00000000      0xFFFFFFFF      0000000000
0x00000000              ×       线程本地存储信息，用于线程私有数据
011      Load Configuration Table 0x00002150      0x00402150      0x00001150      0000000064
     0x00000040              √       加载配置信息（如安全cookie、SEH验证等）
012      Bound Import Table 0x00000000   0x00000000      0xFFFFFFFF      0000000000
0x00000000              ×       绑定导入表，预解析的导入函数地址以加速加载
013      Import Address Table 0x00002000         0x00402000      0x00001000      0000000232
     0x000000E8              √       导入地址表，存储实际导入函数的内存地址
014      Delay Import Descriptor 0x00000000      0x00000000      0xFFFFFFFF      0000000000
     0x00000000              ×       延迟导入描述符，用于延迟加载DLL（运行时再加载）
015      COM Descriptor   0x00000000      0x00000000      0xFFFFFFFF      0000000000
0x00000000              ×       COM组件描述信息（如CLSID、接口信息等）
016      Reserved         0x00000000      0x00000000      0xFFFFFFFF      0000000000
0x00000000              ×       保留未使用
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

查询节表

通过Section命令查询到目标 PE 文件包含 5 个节区，各节区功能与属性明确：.text 为代码节（可执行、可读），存储程序核心执行代码；.rdata 和.data 为已初始化数据节（前者仅可读，后者可读可写），分别存储常量数据与变量数据；.rsrc 为资源节（可读），存放程序所需图标、字符串等资源；.reloc 为重定位节（可读），包含地址重定位信息。各节区均有明确的虚拟偏移（RVA）、实际偏移（FOA）及对应大小，节区属性符合各自功能定位，整体结构完整且规范。

```
[PEVIEW] # Section
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------
编号       节区名称         虚拟偏移(RVA)    虚拟大小         实际偏移(FOA)    实际大小         重定位偏移
    重定位数量       行号偏移         行号数量         节区属性(十六进制)       节区属性描述
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------
1      .text      0x00001000      0x00000B74      0x00000400      0x00000C00      0x00000000
     0          0x00000000   0          0x60000020    代码节; 可执行; 可读;
2      .rdata     0x00002000      0x000007BA      0x00001000      0x00000800      0x00000000
     0          0x00000000   0          0x40000040    已初始化数据; 可读;
3      .data      0x00003000      0x00000518      0x00001800      0x00000200      0x00000000
     0          0x00000000   0          0xC0000040    已初始化数据; 可读; 可写;
4      .rsrc      0x00004000      0x00001DB0      0x00001A00      0x00001E00      0x00000000
     0          0x00000000   0          0x40000040    已初始化数据; 可读;
5      .reloc     0x00006000      0x00000188      0x00003800      0x00000200      0x00000000
     0          0x00000000   0          0x42000040    已初始化数据; 可读;
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------
```

查询所有导入表

通过Import命令查询到目标 PE 文件的导入表信息，首先明确导入表全局数据：数据目录 RVA 为 0x000022A4、FOA 为 0x000012A4，模块加载基地址为 0x00400000。文件仅导入 1 个模块 USER32.dll，该模块的导入描述符 RVA 为 0x000012A4、FOA 为 0x000006A4，TimeDateStamp 为 0x00000000（未绑定），ForwarderChain 为 0x00000000（存在转发），INT（导入名称表）RVA 为 0x0000238C、FOA 为 0x0000138C，IAT（导入地址表）RVA 为 0x00002098、FOA 为 0x00001098。同时，从 USER32.dll 中共导入 19 个函数，均为名称导入方式，涵盖窗口操作（如 CreateWindowExW、DestroyWindow）、消息处理（如 DispatchMessageW、TranslateMessage）、资源加载（如 LoadCursorW、LoadIconW）等功能，且所有函数均存在函数转发状态。

```
[PEVIEW] # Import
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
----------------
导入表全局信息
导入表数据目录RVA: 0x000022A4, FOA: 0x000012A4
模块加载基地址: 0x00400000
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
----------------
[*] 导入模块 [1]:USER32.dll
  导入描述符信息:
  - 描述符RVA: 0x000012A4, FOA: 0x000006A4
  - TimeDateStamp: 0x00000000（未绑定）
  - ForwarderChain: 0x00000000（存在转发）
  - INT（导入名称表）RVA: 0x0000238C, FOA: 0x0000138C
  - IAT（导入地址表）RVA: 0x00002098, FOA: 0x00001098
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
----------------
函数序号          导入类型          Hint值          INT原始数据      IAT原始数据      INT VA
IAT VA          函数名称/序号    状态
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
----------------
[     1]          名称导入    231      0x0000250A          0x0000250A      0x0040238C
0x00402098      EndDialog               存在函数转发
[     2]          名称导入    625      0x000024F8          0x000024F8      0x00402390
0x0040209C      PostQuitMessage         存在函数转发
[     3]          名称导入    233      0x000024EC          0x000024EC      0x00402394
0x004020A0      EndPaint                存在函数转发
[     4]          名称导入    14       0x000024DE          0x000024DE      0x00402398
0x004020A4      BeginPaint              存在函数转发
[     5]          名称导入    161      0x000024CC          0x000024CC      0x0040239C
0x004020A8      DefWindowProcW          存在函数转发
[     6]          名称导入    173      0x000024BC          0x000024BC      0x004023A0
0x004020AC      DestroyWindow           存在函数转发
[     7]          名称导入    178      0x000024AA          0x000024AA      0x004023A4
0x004020B0      DialogBoxParamW         存在函数转发
```

```
[     8]          名称导入    855       0x0000249A          0x0000249A       0x004023A8
0x004020B4       UpdateWindow             存在函数转发
[     9]          名称导入    800       0x0000248C          0x0000248C       0x004023AC
0x004020B8       ShowWindow               存在函数转发
[    10]          名称导入    113       0x0000247A          0x0000247A       0x004023B0
0x004020BC       CreateWindowExW          存在函数转发
[    11]          名称导入    649       0x00002466          0x00002466       0x004023B4
0x004020C0       RegisterClassExW         存在函数转发
[    12]          名称导入    545       0x00002458          0x00002458       0x004023B8
0x004020C4       LoadCursorW              存在函数转发
[    13]          名称导入    547       0x0000244C          0x0000244C       0x004023BC
0x004020C8       LoadIconW                存在函数转发
[    14]          名称导入    181       0x00002438          0x00002438       0x004023C0
0x004020CC       DispatchMessageW         存在函数转发
[    15]          名称导入    831       0x00002424          0x00002424       0x004023C4
0x004020D0       TranslateMessage         存在函数转发
[    16]          名称导入    829       0x0000240C          0x0000240C       0x004023C8
0x004020D4       TranslateAcceleratorW    存在函数转发
[    17]          名称导入    371       0x000023FE          0x000023FE       0x004023CC
0x004020D8       GetMessageW              存在函数转发
[    18]          名称导入    539       0x000023EA          0x000023EA       0x004023D0
0x004020DC       LoadAcceleratorsW        存在函数转发
[    19]          名称导入    560       0x000023DC          0x000023DC       0x004023D4
0x004020E0       LoadStringW              存在函数转发
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
----------------
```

查询所有导入库

通过ImportDll命令查询到该程序共导入 3 个动态链接库，分别是 USER32.dll、MSVCR120.dll 和 KERNEL32.dll。每个 DLL 均包含完整的导入相关地址信息，包括 INT（导入名称表）和 IAT（导入地址表）的 RVA、FOA、VA，以及名称的 RVA 和 FOA。所有 DLL 的时间戳均为 0x00000000（未绑定），ForwarderChain 均为未设置（0），整体呈现了程序运行所依赖的动态链接库及其导入表相关地址配置。

```
[PEVIEW] # ImportDll
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------
序号      DLL名称                    INT RVA          INT FOA          INT VA           IAT RVA
    IAT FOA          IAT VA           时间戳(十六进制)         时间戳(UTC)      ForwarderChain
     名称RVA          名称FOA
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------
1      USER32.dll           0x0000238C   0x0000138C      0x0040238C      0x00002098
0x00001098      0x00402098      0x00000000              未绑定(0)            未设置(0)
0x00002516          0x00001516
2      MSVCR120.dll         0x00002318   0x00001318      0x00402318      0x00002024
0x00001024      0x00402024      0x00000000              未绑定(0)            未设置(0)
0x0000267A          0x0000167A
3      KERNEL32.dll         0x000022F4   0x000012F4      0x004022F4      0x00002000
0x00001000      0x00402000      0x00000000              未绑定(0)            未设置(0)
0x000027AC          0x000017AC
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------
```

查询特定库中导入表

通过ImportByName -dll KERNEL32.dll命令，查询到该程序从 KERNEL32.dll 中导入了 8 个函数，且均为名称导入类型。每个函数均包含完整的地址信息，包括 INT（导入名称表）和 IAT（导入地址表）的 RVA、FOA、VA，以及对应的 Hint 值与函数名。导入的函数涵盖内存指针操作（如 DecodePointer、EncodePointer）、系统信息获取（如 GetSystemTimeAsFileTime、GetCurrentThreadId、GetCurrentProcessId）、性能与调试相关（如 QueryPerformanceCounter、IsDebuggerPresent）及处理器特性检测（IsProcessorFeaturePresent）等功能，满足程序在系统交互、内存管理及运行状态监测等方面的需求。

```
[PEVIEW] # ImportByName -dll KERNEL32.dll
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
序号      导入类型         Hint值          INT RVA          INT FOA          INT VA          IAT
RVA          IAT FOA          IAT VA          函数名/序号      [当前模块: KERNEL32.dll]
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
1       名称导入    254       0x000022F4     0x000012F4       0x004022F4       0x00002000
0x00001000       0x00402000       DecodePointer
2       名称导入    726       0x000022F8     0x000012F8       0x004022F8       0x00002004
0x00001004       0x00402004       GetSystemTimeAsFileTime
3       名称导入    526       0x000022FC     0x000012FC       0x004022FC       0x00002008
0x00001008       0x00402008       GetCurrentThreadId
4       名称导入    522       0x00002300     0x00001300       0x00402300       0x0000200C
0x0000100C       0x0040200C       GetCurrentProcessId
5       名称导入    1069      0x00002304     0x00001304       0x00402304       0x00002010
0x00001010       0x00402010       QueryPerformanceCounter
6       名称导入    877       0x00002308     0x00001308       0x00402308       0x00002014
0x00001014       0x00402014       IsProcessorFeaturePresent
7       名称导入    871       0x0000230C     0x0000130C       0x0040230C       0x00002018
0x00001018       0x00402018       IsDebuggerPresent
8       名称导入    289       0x00002310     0x00001310       0x00402310       0x0000201C
0x0000101C       0x0040201C       EncodePointer
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
```

查询特定库中函数位置

通过ImportByFunction命令分别查询QueryPerformanceCounter和GetMessageW两个函数的导入位置，结果显示两个函数均在程序中被引入，且均为名称导入类型。其中，QueryPerformanceCounter函数来自 KERNEL32.dll，Hint 值为 1069，对应 INT（导入名称表）的 RVA、FOA、VA 分别为 0x00002304、0x00001304、0x00402304，IAT（导入地址表）的 RVA、FOA、VA 分别为 0x00002010、0x00001010、0x00402010；GetMessageW函数来自 USER32.dll，Hint 值为 371，INT 的 RVA、FOA、VA 分别为 0x000023CC、0x000013CC、0x004023CC，IAT 的 RVA、FOA、VA 分别为 0x000020D8、0x000010D8、0x004020D8，清晰呈现了两个函数的导入来源及相关地址信息。

```
[PEVIEW] # ImportByFunction --function QueryPerformanceCounter
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
匹配序号          导入类型         Hint值          INT RVA          INT FOA          INT VA
IAT RVA          IAT FOA          IAT VA          所在DLL
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
[     1]          名称导入    1069      0x00002304          0x00001304       0x00402304
0x00002010       0x00001010       0x00402010       KERNEL32.dll
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
[PEVIEW] #
[PEVIEW] # ImportByFunction --function GetMessageW
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
```

```
匹配序号          导入类型         Hint值          INT RVA          INT FOA          INT VA
IAT RVA          IAT FOA          IAT VA          所在DLL
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
[     1]          名称导入    371      0x000023CC          0x000013CC      0x004023CC
0x000020D8      0x000010D8      0x004020D8      USER32.dll
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------[PEVIEW] #
ImportByFunction --function QueryPerformanceCounter
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
匹配序号          导入类型         Hint值          INT RVA          INT FOA          INT VA
IAT RVA          IAT FOA          IAT VA          所在DLL
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
[     1]          名称导入    1069     0x00002304          0x00001304      0x00402304
0x00002010      0x00001010      0x00402010      KERNEL32.dll
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
[PEVIEW] #
[PEVIEW] # ImportByFunction --function GetMessageW
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
匹配序号          导入类型         Hint值          INT RVA          INT FOA          INT VA
IAT RVA          IAT FOA          IAT VA          所在DLL
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
[     1]          名称导入    371      0x000023CC          0x000013CC      0x004023CC
0x000020D8      0x000010D8      0x004020D8      USER32.dll
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
```

查询所有导出表

通过Export命令查询已打开的 e://win32.dll 文件导出表信息，首先明确导出表全局数据：数据目录 RVA 为 0x00016CD0、FOA 为 0x00005AD0，特征值 0x00000000，时间戳对应 1601-01-01 08:02:56（非实际时间），版本号 0.0，模块名为 win32.dll，起始序号 0x0001，函数总数与命名函数数均为 4。导出的 4 个函数均状态正常且无转发目标，序号 1-4 的函数 RVA 分别为 0x000110EB、0x0001114F、0x00011041、0x00017128，前 3 个函数位于.text 节，第 4 个位于.data 节，函数名称分别为？？0Cdddl@@QAE@XZ、??4Cdddl@@QAEAAV0@ABV0@@Z、?fnddd l@@YAHXZ、?ndddl@@3HA，涵盖类构造、赋值及普通功能函数等类型。

```
[PEVIEW] # Open --path e://win32.dll
[+] 已读入文件
[PEVIEW] #
[PEVIEW] # Export
-------------------------------------------------------------------------------------------------
-----------------------------
导出表全局信息
导出表数据目录RVA: 0x00016CD0, FOA: 0x00005AD0
特征值(Characteristics): 0x00000000
时间戳(TimeDateStamp): 0x68F24619 -> 1601-01-01 08:02:56
版本号: 0.0
```

```
模块名: win32.dll
起始序号(Base): 0x0001
函数总数(NumberOfFunctions): 4
命名函数数(NumberOfNames): 4
-------------------------------------------------------------------------------------------------------
----------------------------
导出序号          函数RVA          函数VA          函数FOA          所在节          函数名称
状态      转发目标
-------------------------------------------------------------------------------------------------------
----------------------------
       1          0x000110EB       0x100110EB       0x000004EB       .text           ??
0Cdddl@@QAE@XZ          正常              无
       2          0x0001114F       0x1001114F       0x0000054F       .text           ??
4Cdddl@@QAEAAV0@ABV0@@Z         正常              无
       3          0x00011041       0x10011041       0x00000441       .text           ?
fndddl@@YAHXZ           正常              无
       4          0x00017128       0x10017128       0x00005F28       .data           ?ndddl@@3HA
             正常              无
-------------------------------------------------------------------------------------------------------
----------------------------
```

查询重定位项

通过FixReloc命令查询到该程序的重定位表信息，首先明确全局配置：原始映像基地址为 0x10000000，模拟新基地址（示例）为 0x10010000，重定位表 RVA 为 0x0001A000、FOA 为 0x00006E00，总大小 0x00000324 字节，最终遍历出 5 个有效重定位块。仅第一个块有完整配置（起始 RVA 0x00011000、FOA 0x00006E00、VA 0x10011000、大小 0x00D4、项数量 102），后续块无实际配置；所有展示的 21 个重定位项均为 “HIGHLOW（32 位完整）” 类型，包含项偏移、项 RVA/FOA、原始 VA 及重定位后 VA，重定位后 VA 均在原始 VA 基础上增加 0x10000（因新基地址较原始基地址偏移 0x10000），整体可支撑模块加载地址变更时的地址修正需求。

```
[PEVIEW] # FixReloc
-------------------------------------------------------------------------------------------------------
----------------------------
重定位表全局信息
原始映像基地址: 0x10000000
模拟新基地址(示例): 0x10010000
重定位表RVA: 0x0001A000, FOA: 0x00006E00, 总大小: 0x00000324 字节
-------------------------------------------------------------------------------------------------------
----------------------------
块序号 | 块起始RVA  | 块FOA      | 块VA       | 块大小  | 项数量 | 项序号 | 类型 | 类型描述        | 项
偏移 | 项RVA     | 项FOA     | 原始VA    | 重定位后VA
-------------------------------------------------------------------------------------------------------
----------------------------
1      | 0x00011000 | 0x00006E00 | 0x10011000 | 0x00D4 | 102    | 1      | 3    | HIGHLOW
（32位完整） | 0x0442 | 0x00011442 | 0x00000842 | 0x10017130 | 0x10027130
0      | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0      | 2      | 3    | HIGHLOW
（32位完整） | 0x0455 | 0x00011455 | 0x00000855 | 0x10017130 | 0x10027130
0      | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0      | 3      | 3    | HIGHLOW
（32位完整） | 0x04AA | 0x000114AA | 0x000008AA | 0x1001565C | 0x1002565C
0      | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0      | 4      | 3    | HIGHLOW
（32位完整） | 0x04B7 | 0x000114B7 | 0x000008B7 | 0x100180D4 | 0x100280D4
```

```
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 5       | 3     | HIGHLOW
（32位完整） | 0x04C7 | 0x000114C7 | 0x000008C7 | 0x1001804C | 0x1002804C
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 6       | 3     | HIGHLOW
（32位完整） | 0x04CC | 0x000114CC | 0x000008CC | 0x10017544 | 0x10027544
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 7       | 3     | HIGHLOW
（32位完整） | 0x04D2 | 0x000114D2 | 0x000008D2 | 0x10017544 | 0x10027544
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 8       | 3     | HIGHLOW
（32位完整） | 0x04D8 | 0x000114D8 | 0x000008D8 | 0x10017534 | 0x10027534
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 9       | 3     | HIGHLOW
（32位完整） | 0x04F8 | 0x000114F8 | 0x000008F8 | 0x1001101E | 0x1002101E
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 10      | 3     | HIGHLOW
（32位完整） | 0x0505 | 0x00011505 | 0x00000905 | 0x10011136 | 0x10021136
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 11      | 3     | HIGHLOW
（32位完整） | 0x055E | 0x0001155E | 0x0000095E | 0x10017134 | 0x10027134
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 12      | 3     | HIGHLOW
（32位完整） | 0x0566 | 0x00011566 | 0x00000966 | 0x10017134 | 0x10027134
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 13      | 3     | HIGHLOW
（32位完整） | 0x056E | 0x0001156E | 0x0000096E | 0x10017134 | 0x10027134
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 14      | 3     | HIGHLOW
（32位完整） | 0x05A2 | 0x000115A2 | 0x000009A2 | 0x10017520 | 0x10027520
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 15      | 3     | HIGHLOW
（32位完整） | 0x05CA | 0x000115CA | 0x000009CA | 0x10017530 | 0x10027530
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 16      | 3     | HIGHLOW
（32位完整） | 0x05DF | 0x000115DF | 0x000009DF | 0x10017530 | 0x10027530
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 17      | 3     | HIGHLOW
（32位完整） | 0x05E8 | 0x000115E8 | 0x000009E8 | 0x10015514 | 0x10025514
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 18      | 3     | HIGHLOW
（32位完整） | 0x05ED | 0x000115ED | 0x000009ED | 0x10015208 | 0x10025208
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 19      | 3     | HIGHLOW
（32位完整） | 0x0605 | 0x00011605 | 0x00000A05 | 0x10015104 | 0x10025104
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 20      | 3     | HIGHLOW
（32位完整） | 0x060A | 0x0001160A | 0x00000A0A | 0x10015000 | 0x10025000
0       | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0       | 21      | 3     | HIGHLOW
（32位完整） | 0x0618 | 0x00011618 | 0x00000A18 | 0x10017530 | 0x10027530
-------------------------------------------------------------------------------------------------
----------------------------
重定位表遍历完成，共5个有效重定位块
-------------------------------------------------------------------------------------------------
----------------------------
```

查询重定位表分页

通过FixRelocPage命令查询到该程序重定位表的分页情况，先明确全局信息：原始映像基地址 0x10000000，模拟新基地址 0x10010000，重定位表数据目录 RVA 0x0001A000、FOA 0x00006E00，总大小 0x00000324 字节，共遍历出 5 个重定位块。此次展示了 2 个核心块的分页详情：块 1 起始 RVA 0x00011000、FOA 0x00006E00、内存起始 VA 0x10011000、长度 0x00D4，含 102 个重定位项；块 5 起始 RVA 0x00016000、FOA 0x000070F8、内存起始 VA 0x10016000、长度 0x002C，含 18 个重定位项。所有有效项（除块 5 第 18 项）均为 “HIGHLOW（32 位完整地址）” 类型，重定位后地址较原始地址偏移 0x10000（匹配新基地址变更），块 5 第 18 项为 “ABSOLUTE（无意义）” 类型，无实际地址修正作用，整体分页结构清晰呈现了不同内存块的重定位配置。

```
[PEVIEW] # FixRelocPage
```

```
--------------------------------------------------------------------------------------------------------------
----------------------------
重定位表全局信息
原始映像基地址: 0x10000000
模拟新基地址(示例): 0x10010000
重定位表数据目录RVA: 0x0001A000, FOA: 0x00006E00
重定位表总大小: 0x00000324 字节
--------------------------------------------------------------------------------------------------------------
----------------------------
块序号     块起始RVA         块FOA     块内存起始VA      块长度            重定位项数
--------------------------------------------------------------------------------------------------------------
----------------------------
1          0x00011000       0x00006E00       0x10011000       0x00D4           102
  ├─ 项序号       类型      类型描述          项偏移            完整RVA          原始地址          重定位后地
址
  ├─ 1    3        HIGHLOW（32位完整地址）          0x0442          0x00011442       0x10017130
   0x10027130
  ├─ 2    3        HIGHLOW（32位完整地址）          0x0455          0x00011455       0x10017130
   0x10027130
  ├─ 3    3        HIGHLOW（32位完整地址）          0x04AA          0x000114AA       0x1001565C
   0x1002565C
  ├─ 4    3        HIGHLOW（32位完整地址）          0x04B7          0x000114B7       0x100180D4
   0x100280D4
  ├─ 5    3        HIGHLOW（32位完整地址）          0x04C7          0x000114C7       0x1001804C
   0x1002804C
  ├─ 6    3        HIGHLOW（32位完整地址）          0x04CC          0x000114CC       0x10017544
   0x10027544
  ├─ 7    3        HIGHLOW（32位完整地址）          0x04D2          0x000114D2       0x10017544
   0x10027544
  ├─ 8    3        HIGHLOW（32位完整地址）          0x04D8          0x000114D8       0x10017534
   0x10027534
  ├─ 9    3        HIGHLOW（32位完整地址）          0x04F8          0x000114F8       0x1001101E
   0x1002101E
  ├─ 10           3        HIGHLOW（32位完整地址）          0x0505          0x00011505
0x10011136       0x10021136
  ├─ 11           3        HIGHLOW（32位完整地址）          0x055E          0x0001155E
0x10017134       0x10027134
  ├─ 12           3        HIGHLOW（32位完整地址）          0x0566          0x00011566
0x10017134       0x10027134
  ├─ 13           3        HIGHLOW（32位完整地址）          0x056E          0x0001156E
0x10017134       0x10027134
  5          0x00016000       0x000070F8       0x10016000       0x002C           18
  ├─ 项序号       类型      类型描述          项偏移            完整RVA          原始地址          重定位后地
址
  ├─ 1    3        HIGHLOW（32位完整地址）          0x01A8          0x000161A8       0x10017160
   0x10027160
  ├─ 2    3        HIGHLOW（32位完整地址）          0x01AC          0x000161AC       0x100171B0
   0x100271B0
  ├─ 3    3        HIGHLOW（32位完整地址）          0x01F4          0x000161F4       0x10017004
   0x10027004
  ├─ 4    3        HIGHLOW（32位完整地址）          0x05E4          0x000165E4       0x100110AF
   0x100210AF
```

```
  ├─ 5    3       HIGHLOW（32位完整地址）          0x05E8          0x000165E8      0x100110AF
   0x100210AF
  ├─ 6    3       HIGHLOW（32位完整地址）          0x08F8          0x000168F8      0x100110A5
   0x100210A5
  ├─ 7    3       HIGHLOW（32位完整地址）          0x08FC          0x000168FC      0x100110A5
   0x100210A5
  ├─ 8    3       HIGHLOW（32位完整地址）          0x0B20          0x00016B20      0x10011AB0
   0x10021AB0
  ├─ 9    3       HIGHLOW（32位完整地址）          0x0B28          0x00016B28      0x10011A76
   0x10021A76
  ├─ 10           3       HIGHLOW（32位完整地址）          0x0B2C          0x00016B2C
0x10011A91      0x10021A91
  ├─ 11           3       HIGHLOW（32位完整地址）          0x0B4C          0x00016B4C
0x10011DA5      0x10021DA5
  ├─ 12           3       HIGHLOW（32位完整地址）          0x0B50          0x00016B50
0x10011DCC      0x10021DCC
  ├─ 13           3       HIGHLOW（32位完整地址）          0x0B70          0x00016B70
0x10012088      0x10022088
  ├─ 14           3       HIGHLOW（32位完整地址）          0x0B8C          0x00016B8C
0x10012376      0x10022376
  ├─ 15           3       HIGHLOW（32位完整地址）          0x0B90          0x00016B90
0x10012389      0x10022389
  ├─ 16           3       HIGHLOW（32位完整地址）          0x0BAC          0x00016BAC
0x10012448      0x10022448
  ├─ 17           3       HIGHLOW（32位完整地址）          0x0BB0          0x00016BB0
0x1001245B      0x1002245B
  ├─ 18           0       ABSOLUTE（无意义）       0x0000          0x00016000      0x00000000
   0x00010000
  └─ （块结束）
-------------------------------------------------------------------------------------------------
----------------------------
重定位表遍历完成，共5个重定位块
-------------------------------------------------------------------------------------------------
----------------------------
```

查询重定位页内分页

通过FixRelocRVA --rva 00011000命令，查询指定 RVA（0x00011000）对应的重定位页内分页信息。首先明确重定位表全局配置：原始映像基地址 0x10000000，模拟新基地址 0x10010000，重定位表 RVA 0x0001A000、FOA 0x00006E00，总大小 0x00000324 字节，共 5 个有效重定位块。查询结果中，仅第 1 个块与指定 RVA 匹配（块起始 RVA 为 0x00011000），该块 FOA 0x00006E00、VA 0x10011000、大小 0x00D4、含 102 个重定位项；后续块无实际配置。展示的 19 个重定位项均为 “HIGHLOW（32 位完整）” 类型，包含项偏移、项 RVA/FOA、原始 VA 及重定位后 VA，重定位后 VA 均在原始 VA 基础上增加 0x10000（匹配新基地址偏移），清晰呈现了指定 RVA 页内的重定位细节。

```
[PEVIEW] # FixRelocRVA --rva 00011000
-------------------------------------------------------------------------------------------------
----------------------------
重定位表全局信息
原始映像基地址: 0x10000000
模拟新基地址(示例): 0x10010000
重定位表RVA: 0x0001A000, FOA: 0x00006E00, 总大小: 0x00000324 字节
```

```
---------------------------------------------------------------------------------------------------
----------------------------
块序号 | 块起始RVA | 块FOA | 块VA | 块大小 | 项数量 | 项序号 | 类型 | 类型描述 | 项
偏移 | 项RVA | 项FOA | 原始VA | 重定位后VA
---------------------------------------------------------------------------------------------------
----------------------------
1 | 0x00011000 | 0x00006E00 | 0x10011000 | 0x00D4 | 102 | 1 | 3 | HIGHLOW
（32位完整） | 0x0442 | 0x00011442 | 0x00000842 | 0x10017130 | 0x10027130
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 2 | 3 | HIGHLOW
（32位完整） | 0x0455 | 0x00011455 | 0x00000855 | 0x10017130 | 0x10027130
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 3 | 3 | HIGHLOW
（32位完整） | 0x04AA | 0x000114AA | 0x000008AA | 0x1001565C | 0x1002565C
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 4 | 3 | HIGHLOW
（32位完整） | 0x04B7 | 0x000114B7 | 0x000008B7 | 0x100180D4 | 0x100280D4
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 5 | 3 | HIGHLOW
（32位完整） | 0x04C7 | 0x000114C7 | 0x000008C7 | 0x1001804C | 0x1002804C
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 6 | 3 | HIGHLOW
（32位完整） | 0x04CC | 0x000114CC | 0x000008CC | 0x10017544 | 0x10027544
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 7 | 3 | HIGHLOW
（32位完整） | 0x04D2 | 0x000114D2 | 0x000008D2 | 0x10017544 | 0x10027544
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 8 | 3 | HIGHLOW
（32位完整） | 0x04D8 | 0x000114D8 | 0x000008D8 | 0x10017534 | 0x10027534
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 9 | 3 | HIGHLOW
（32位完整） | 0x04F8 | 0x000114F8 | 0x000008F8 | 0x1001101E | 0x1002101E
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 10 | 3 | HIGHLOW
（32位完整） | 0x0505 | 0x00011505 | 0x00000905 | 0x10011136 | 0x10021136
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 11 | 3 | HIGHLOW
（32位完整） | 0x055E | 0x0001155E | 0x0000095E | 0x10017134 | 0x10027134
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 12 | 3 | HIGHLOW
（32位完整） | 0x0566 | 0x00011566 | 0x00000966 | 0x10017134 | 0x10027134
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 13 | 3 | HIGHLOW
（32位完整） | 0x056E | 0x0001156E | 0x0000096E | 0x10017134 | 0x10027134
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 14 | 3 | HIGHLOW
（32位完整） | 0x05A2 | 0x000115A2 | 0x000009A2 | 0x10017520 | 0x10027520
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 15 | 3 | HIGHLOW
（32位完整） | 0x05CA | 0x000115CA | 0x000009CA | 0x10017530 | 0x10027530
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 16 | 3 | HIGHLOW
（32位完整） | 0x05DF | 0x000115DF | 0x000009DF | 0x10017530 | 0x10027530
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 17 | 3 | HIGHLOW
（32位完整） | 0x05E8 | 0x000115E8 | 0x000009E8 | 0x10015514 | 0x10025514
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 18 | 3 | HIGHLOW
（32位完整） | 0x05ED | 0x000115ED | 0x000009ED | 0x10015208 | 0x10025208
0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000 | 0 | 19 | 3 | HIGHLOW
（32位完整） | 0x0605 | 0x00011605 | 0x00000A05 | 0x10015104 | 0x10025104
---------------------------------------------------------------------------------------------------
----------------------------
重定位表遍历完成，共5个有效重定位块
---------------------------------------------------------------------------------------------------
----------------------------
```

查询资源表

通过Resource命令查询到该程序的资源表信息，首先明确全局配置：资源表数据目录 RVA 为 0x00019000、FOA 为 0x00006800，总大小 1084 字节（0x0000043C），在文件中的基地址为 0x01506800（由全局文件基地址与 FOA 计算得出）。资源表结构按 “类型目录→名称 / ID 目录→语言目录” 层级展开：类型目录含 1 个 ID 条目（类型 ID 0x0018，未知类型），指向名称 / ID 目录；名称 / ID 目录含 1 个 ID 条目（资源 ID 0x0002），指向语言目录；语言目录含 1 个 ID 条目（语言 ID 0x0409），对应资源数据，该数据 RVA 为 0x00019170、FOA 为 0x00006970，大小 381 字节（0x0000017D），代码页 0x0000（用于本地化编码），整体仅解析出一级资源，结构层级清晰但资源类型未明确。

```
[PEVIEW] # Resource
------------------------------------------------------------------------------------------------------------
----------------------------
资源表全局信息
资源表数据目录RVA: 0x00019000, FOA: 0x00006800
资源表总大小: 1084 字节 (0x0000043C)
资源表在文件中的基地址: 0x01506800 (GlobalFileBase + FOA)
------------------------------------------------------------------------------------------------------------
----------------------------

[ 类型目录 ] 信息:
- 目录RVA: 0x00018800, FOA: 0x00006800
- 命名条目数: 0, ID条目数: 1, 总条目数: 1

  |- [条目 1] ID条目 (类型ID: 0x0018, 类型: 未知类型):
  |- - 条目偏移: 0x00000018 (相对资源基地址)
  |- - 条目类型: 目录 (指向名称/ID目录)

  |- [ 名称/ID目录 ] 信息:
  |- - 目录RVA: 0x00019018, FOA: 0x00006818
  |- - 命名条目数: 0, ID条目数: 1, 总条目数: 1

  |-   |- [条目 1] ID条目 (资源ID: 0x0002):
  |-   |- - 条目偏移: 0x00000030 (相对资源基地址)
  |-   |- - 条目类型: 目录 (指向语言目录)

  |-   |- [ 语言目录 ] 信息:
  |-   |- - 目录RVA: 0x00019030, FOA: 0x00006830
  |-   |- - 命名条目数: 0, ID条目数: 1, 总条目数: 1

  |-   |-   |- [条目 1] ID条目 (语言ID: 0x0409):
  |-   |-   |- - 条目偏移: 0x00000048 (相对资源基地址)
  |-   |-   |- - 条目类型: 资源数据
  |-   |-   |- - 数据RVA: 0x00019170, FOA: 0x00006970
  |-   |-   |- - 数据大小: 381 字节 (0x0000017D)
  |-   |-   |- - 代码页: 0x0000 (用于本地化编码)
------------------------------------------------------------------------------------------------------------
----------------------------
资源表解析完成
------------------------------------------------------------------------------------------------------------
----------------------------
```

检查函数内存地址

通过GetProcAddr命令分别验证 user32.dll 中MessageBoxA和MessageBoxW两个函数的内存地址，结果显示：两个函数所在的 user32.dll 加载基地址均为 0x75660000；其中MessageBoxA函数的相对偏移（RVA）为 0x0008A7F0，绝对地址（VA）为 0x756EA7F0（由基地址加 RVA 计算得出）；MessageBoxW函数的相对偏移（RVA）为 0x0008AD10，绝对地址（VA）为 0x756EAD10，清晰呈现了两个同模块函数的内存地址配置，且符合基地址与 RVA 共同确定绝对地址的计算逻辑。

```
[PEVIEW] # GetProcAddr --dll user32.dll --function MessageBoxA
--------------------------------------------------------------------------------
[*] 尝试获取DLL函数地址:
[*] DLL名称:user32.dll
[*] 函数名称:MessageBoxA
--------------------------------------------------------------------------------
[+] DLL加载基地址:0x75660000
[+] 函数相对偏移（RVA）:0x0008A7F0
[+] 函数绝对地址（VA）:0x756EA7F0
--------------------------------------------------------------------------------
[PEVIEW] #
[PEVIEW] # GetProcAddr --dll user32.dll --function MessageBoxW
--------------------------------------------------------------------------------
[*] 尝试获取DLL函数地址:
[*] DLL名称:user32.dll
[*] 函数名称:MessageBoxW
--------------------------------------------------------------------------------
[+] DLL加载基地址:0x75660000
[+] 函数相对偏移（RVA）:0x0008AD10
[+] 函数绝对地址（VA）:0x756EAD10
--------------------------------------------------------------------------------
```

检查启用保护模式

通过Protection命令检查当前打开的 DLL 文件保护模式，结果显示：模块为 x86（32 位）图形界面（GUI）类型 DLL，链接器版本 0.0。安全特性方面，仅基址随机化（ASLR）启用、DEP/NX 保护兼容（数据页不可执行），高熵 ASLR、控制流保护（CFG）、强制完整性均禁用，支持 SEH 异常处理且无数字证书。其他特性上，终端服务感知为“否”，UAC 虚拟化为禁用（不虚拟化文件系统 / 注册表），隔离特性禁用，但存在调试信息（含调试符号目录），整体保护配置以基础安全防护为主，未启用高阶保护功能。

```
[PEVIEW] # Protection
--------------------------------------------------------------------------------
[模块基本属性]
--------------------------------------------------------------------------------
文件类型:        DLL文件
机器架构:        x86 (32位)
子系统类型:      图形界面应用程序 (GUI)
链接器版本:      0.0
--------------------------------------------------------------------------------
[安全特性]
--------------------------------------------------------------------------------
基址随机化(ASLR):  启用
高熵ASLR:          禁用
```

```
DEP/NX保护:          兼容 (数据页不可执行)
控制流保护(CFG):    禁用
强制完整性:          禁用
SEH异常处理:         允许 (支持结构化异常处理)
数字证书:            不存在
--------------------------------------------------------------------------------
[其他特性]
--------------------------------------------------------------------------------
终端服务感知:        否
UAC虚拟化:           禁用 (不虚拟化文件系统/注册表)
隔离特性:            禁用
调试信息:            存在 (包含调试符号目录)
--------------------------------------------------------------------------------
```

输出十六进制及字符串

通过HexAscii --offset 0x400 --len 200命令，获取了目标文件从偏移 0x400（十进制 1024）开始、长度 200 字节的十六进制机器码及对应 ASCII 字符。文件总大小为 14848 字节（0x3A00），本次读取范围为偏移 0x400 至 0x4C7（十进制 1024 至 1223）。输出内容以十六进制和 ASCII 字符对照形式呈现，左侧为偏移地址，中间两列是十六进制数据（每 16 字节为一组），右侧为可打印的 ASCII 字符（不可打印字符以 "." 表示），共输出 13 行数据，完整展示了该偏移区间内的二进制数据及字符映射情况，便于分析文件此部分的机器码或数据内容。

```
[PEVIEW] # HexAscii --offset 0x400 --len 200
--------------------------------------------------------------------------------------------------
----------------------------
文件总大小:14848 字节（0x0000000000003A00）
读取范围:起始偏移 = 1024（0x0000000000000400），结束偏移 = 1223（0x00000000000004C7），读取长度 =
200 字节
--------------------------------------------------------------------------------------------------
----------------------------
--------------------------------------------------------------------------------------------------
----
偏移         00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  |  ASCII
--------------------------------------------------------------------------------------------------
----
0x00000400 | 55 8B EC 83 EC 24 A1 00  30 40 00 33 C5 89 45 FC  | U....$..0@.3..E.
0x00000410 | 56 8B 35 E0 20 40 00 57  8B 7D 08 6A 64 68 38 34  | V.5. @.W.}.jdh84
0x00000420 | 40 00 6A 67 57 FF D6 6A  64 68 70 33 40 00 6A 6D  | @.jgW..jdhp3@.jm
0x00000430 | 57 FF D6 8B CF E8 D6 00  00 00 6A 00 57 6A 00 6A  | W.........j.Wj.j
0x00000440 | 00 6A 00 68 00 00 00 80  6A 00 68 00 00 00 80 68  | .j.h....j.h....h
0x00000450 | 00 00 CF 00 68 38 34 40  00 68 70 33 40 00 6A 00  | ....h84@.hp3@.j.
0x00000460 | 89 3D 00 35 40 00 FF 15  BC 20 40 00 8B F0 85 F6  | .=.5@.... @.....
0x00000470 | 0F 84 86 00 00 00 FF 75  14 56 FF 15 B8 20 40 00  | .......u.V... @.
0x00000480 | 56 FF 15 B4 20 40 00 6A  6D 57 FF 15 DC 20 40 00  | V... @.jmW... @.
0x00000490 | 8B 35 D8 20 40 00 6A 00  6A 00 89 45 DC 8D 45 E0  | .5. @.j.j..E..E.
0x000004A0 | 6A 00 50 FF D6 85 C0 74  3E 8B 3D D0 20 40 00 53  | j.P....t>.=. @.S
0x000004B0 | 8B 1D CC 20 40 00 8D 45  E0 50 FF 75 DC FF 75 E0  | ... @..E.P.u..u.
0x000004C0 | FF 15 D4 20 40 00 85 C0                           | ... @...
--------------------------------------------------------------------------------------------------
----------------------------
读取完成:共读取 200 字节，输出 13 行
```

```
--------------------------------------------------------------------------------------------
----------------------------
```

特定区域反汇编输出

通过Disassembly --offset 0x400 --len 50命令，对目标文件偏移 0x400 开始、长度 50 字节的区域进行反汇编，输出了该区域的机器码与对应汇编指令。结果按 “文件偏移 - 机器码 - 反汇编指令集” 三列呈现，共涵盖 18 条指令，内容以函数栈帧初始化（如push ebp、mov ebp, esp、sub esp, 0x24）、寄存器操作（如mov eax, dword ptr [0x403000]、xor eax, ebp）、寄存器入栈（如push esi、push edi）及函数调用准备（如多次push参数、call esi）为主，清晰展示了该特定区域的底层执行逻辑，可用于分析函数初始化或关键代码片段的指令流程。

```
[PEVIEW] # Disassembly --offset 0x400 --len 50
--------------------------------------------------------------------------------
文件偏移                    机器码                              反汇编指令集
--------------------------------------------------------------------------------
0x00000400 | 55                               | push ebp
0x00000401 | 8B EC                            | mov ebp, esp
0x00000403 | 83 EC 24                         | sub esp, 0x24
0x00000406 | A1 00 30 40 00                   | mov eax, dword ptr [0x403000]
0x0000040B | 33 C5                            | xor eax, ebp
0x0000040D | 89 45 FC                         | mov dword ptr [ebp - 4], eax
0x00000410 | 56                               | push esi
0x00000411 | 8B 35 E0 20 40 00                | mov esi, dword ptr [0x4020e0]
0x00000417 | 57                               | push edi
0x00000418 | 8B 7D 08                         | mov edi, dword ptr [ebp + 8]
0x0000041B | 6A 64                            | push 0x64
0x0000041D | 68 38 34 40 00                   | push 0x403438
0x00000422 | 6A 67                            | push 0x67
0x00000424 | 57                               | push edi
0x00000425 | FF D6                            | call esi
0x00000427 | 6A 64                            | push 0x64
0x00000429 | 68 70 33 40 00                   | push 0x403370
0x0000042E | 6A 6D                            | push 0x6d
0x00000430 | 57                               | push edi
--------------------------------------------------------------------------------
```

区域搜索机器码特征

通过SearchSig --offset 0x300 --len 1024 --sig 55 8B ??命令，在文件偏移 0x300（十进制 768）开始、长度 1024 字节的范围内，搜索特征码 “55 8B ??（含通配符，总长度 3 字节）”，共找到 4 个匹配结果。匹配位置的十进制偏移分别为 1024、1296、1440、1728，对应十六进制偏移 0x400、0x510、0x5A0、0x6C0，每个位置的十六进制数据均以 “55 8B” 开头，第三个字节符合通配符匹配规则（分别为 EC、EC、E4、EC），清晰呈现了特征码在指定区域内的分布情况，可用于定位符合特定指令模式（如函数开头的栈帧初始化指令）的代码片段。

```
[PEVIEW] # SearchSig --offset 0x300 --len 1024 --sig 55 8B ??
--------------------------------------------------------------------------------------------
----------------------------
搜索范围:起始偏移 = 768（0x0000000000000300），搜索长度 = 1024 字节
特征码:55 8B ??（长度:3字节，包含通配符）
--------------------------------------------------------------------------------------------
----------------------------
```

```
找到 4 个匹配结果:
-------------------------------------------------------------------------------------------------
----
偏移（十进制）           偏移（十六进制）           匹配位置的十六进制数据
-------------------------------------------------------------------------------------------------
----
1024                  0x0000000000000400  55 8B EC 83 EC 24 A1 00 30 40 00 33 C5 89 45 FC
1296                  0x0000000000000510  55 8B EC 83 EC 34 A1 00 30 40 00 33 C5 89 45 FC
1440                  0x00000000000005A0  55 8B EC 83 E4 F8 83 EC 4C A1 00 30 40 00 33 C4
1728                  0x00000000000006C0  55 8B EC 8B 45 0C 2D 10 01 00 00 74 1F 48 75 25
-------------------------------------------------------------------------------------------------
---------------------------
搜索完成
-------------------------------------------------------------------------------------------------
---------------------------
```

区域搜索内字符串特征

通过SearchString --offset 0x10 --len 1024 --str .text命令，在文件偏移 0x10（十进制 16）开始、长度 1024 字节的范围内，搜索 ASCII 字符串 ".text"（长度 5 字节），找到 1 个匹配结果。该结果位于十进制偏移 504（十六进制 0x1F8）处，对应的 VA 地址为 0x00400000，匹配内容为 ".text"，准确呈现了目标字符串在指定区域内的位置信息，可用于定位与.text 节区相关的字符串标识。

```
[PEVIEW] # SearchString --offset 0x10 --len 1024 --str .text
-------------------------------------------------------------------------------------------------
---------------------------
搜索范围:起始偏移 = 16（0x0000000000000010），搜索长度 = 1024 字节
目标字符串:".text"（长度:5字节，ASCII）
-------------------------------------------------------------------------------------------------
---------------------------
找到 1 个匹配结果:
-------------------------------------------------------------------------------------------------
----
偏移（十进制）           偏移（十六进制）           VA地址（十六进制）           匹配内容
-------------------------------------------------------------------------------------------------
----
504                   0x00000000000001F8  0x0000000000400000  ".text"
-------------------------------------------------------------------------------------------------
---------------------------
搜索完成
-------------------------------------------------------------------------------------------------
---------------------------
```

内置十六进制计算器

通过内置十六进制计算器的Add和Sub命令进行了运算：Add --x 0x100 --y 0x200计算得 0x300（十进制 768、八进制 1400、二进制 1100000000）； Sub --x 0x100 --y 0x10计算得 0xF0（十进制 240、八进制 360、二进制 11110000），结果均以十六进制、十进制、八进制、二进制四种形式呈现，清晰展示了十六进制加减法的运算结果。

```
[PEVIEW] # Add --x 0x100 --y 0x200
0x100 + 0x200 =>
        HEX= 00000300
        DEC= 768
        OCT= 1400
        BIN= 1100000000
[PEVIEW] #
[PEVIEW] # Sub --x 0x100 --y 0x10
0x100 - 0x10 =>
        HEX= 000000F0
        DEC= 240
        OCT= 360
        BIN= 11110000
```

文件地址转虚拟地址

通过FoaToVa --foa 420命令，将目标文件偏移（FOA）0x00000420 转换为虚拟地址（VA），转换过程如下：首先明确模块基地址为 0x00400000、节区总数 5 个，随后定位到 FOA 所属的.text 节区 —— 该节区文件偏移范围 0x00000400-0x00000FFF、虚拟地址（RVA 起始）0x00001000。通过公式计算：先算 RVA = 节区 RVA 起始 +（FOA - 节区文件偏移起始）=0x00001000+（0x00000420-0x00000400）=0x00001020，再算 VA = 基地址 + RVA=0x00400000+0x00001020=0x00401020，最终得到转换结果：FOA 0x00000420 对应 RVA 0x00001020、VA 0x00401020。

```
[PEVIEW] # FoaToVa --foa 420
--------------------------------------------------------------------------------
[*] FOA转VA转换开始:
[*] 目标FOA地址:0x00000420
[*] 模块基地址（ImageBase）:0x00400000
[*] 节区总数:5
--------------------------------------------------------------------------------
[节区 1] 名称:.text | 文件偏移范围:0x00000400 - 0x00000FFF | 节区虚拟地址:0x00001000
--------------------------------------------------------------------------------
[+] 找到FOA所属节区:
    节区名称:.text
    节区文件偏移范围:0x00000400 - 0x00000FFF（有效范围）
    节区虚拟地址（RVA起始）:0x00001000 | 节区文件大小:0x00000C00
------------------------------------------------------------
    转换过程:
    RVA = 节区虚拟地址 + (FOA - 节区文件偏移起始)
    RVA = 0x00001000 + (0x00000420 - 0x00000400) = 0x00001020
    VA = 基地址 + RVA
    VA = 0x00400000 + 0x00001020 = 0x00401020
------------------------------------------------------------
    最终结果:
    FOA:0x00000420 --> RVA:0x00001020 --> VA:0x00401020
--------------------------------------------------------------------------------
```

虚拟地址转文件地址

通过VaToFoa --va 00401020命令，将目标虚拟地址（VA）0x00401020 转换为文件偏移（FOA），转换过程如下：先明确模块基地址 0x00400000、节区总数 5 个，接着定位到 VA 所属的.text 节区 —— 该节区 VA 范围 0x00401000-0x00401B73、RVA 起始 0x00001000、文件偏移 0x00000400。按公式计算：先算 RVA=VA - 基地址 = 0x00401020-0x00400000=0x00001020，再算 FOA = 节区文件偏移 +（RVA - 节区 RVA 起始）=0x00000400+（0x00001020-0x00001000）=0x00000420，最终得到转换结果：VA 0x00401020 对应 RVA 0x00001020、FOA 0x00000420。

```
[PEVIEW] # VaToFoa --va 00401020
--------------------------------------------------------------------------------
[*] VA转FOA转换开始:
[*] 目标VA地址:0x00401020
[*] 模块基地址（ImageBase）:0x00400000
[*] 节区总数:5
--------------------------------------------------------------------------------
[节区 1] 名称:.text | VA范围:0x00401000 - 0x00401B73 | 节区RVA起始:0x00001000 | 内存大
小:0x00000B74
--------------------------------------------------------------------------------
[+] 找到VA所属节区:
    节区名称:.text
    节区VA范围:0x00401000 - 0x00401B73（有效范围）
    节区RVA起始:0x00001000 | 节区文件偏移:0x00000400 | 内存大小:0x00000B74
------------------------------------------------------------
    转换过程:
    RVA = VA - 基地址
    RVA = 0x00401020 - 0x00400000 = 0x00001020
    FOA = 节区文件偏移 + (RVA - 节区RVA起始)
    FOA = 0x00000400 + (0x00001020 - 0x00001000) = 0x00000420
------------------------------------------------------------
    最终结果:
    VA:0x00401020 --> RVA:0x00001020 --> FOA:0x00000420
--------------------------------------------------------------------------------
```

相对地址转文件地址

通过RvaToFoa --rva 00001020命令，将目标相对地址（RVA）0x00001020 转换为文件偏移（FOA），转换过程如下：先明确模块基地址 0x00400000、节区总数 5 个，随后定位到 RVA 所属的.text 节区 —— 该节区 RVA 范围 0x00001000-0x00001B74、文件偏移 0x00000400。按公式计算：FOA = 节区文件偏移 +（RVA - 节区 RVA 起始）=0x00000400+（0x00001020-0x00001000）=0x00000420，同时还推导得出对应虚拟地址（VA）为 0x00401020，最终转换结果为：RVA 0x00001020 对应 VA 0x00401020、FOA 0x00000420。

```
[PEVIEW] # RvaToFoa --rva 00001020
--------------------------------------------------------------------------------
[*] RVA转FOA转换开始:
[*] 目标RVA地址:0x00001020
[*] 模块基地址（ImageBase）:0x00400000
[*] 节区总数:5
--------------------------------------------------------------------------------
[节区 1] 名称:.text | RVA范围:0x00001000 - 0x00001B74 | 节区文件偏移:0x00000400
--------------------------------------------------------------------------------
[+] 找到RVA所属节区:
```

```
    节区名称:.text
    节区RVA起始:0x00001000 | 节区RVA结束:0x00001B74
    节区文件偏移:0x00000400 | 节区虚拟大小:0x00000B74 | 节区文件大小:0x00000C00
------------------------------------------------------------
    转换过程:
    FOA = 节区文件偏移 + (RVA - 节区RVA起始)
    FOA = 0x00000400 + (0x00001020 - 0x00001000) = 0x00000420
------------------------------------------------------------
    最终结果:
    RVA:0x00001020 --> VA:0x00401020 --> FOA:0x00000420
--------------------------------------------------------------------------------
```

虚拟地址转相对地址

通过VaToRva --va 00401020命令，将目标虚拟地址（VA）0x00401020 转换为相对地址（RVA），转换过程如下：明确模块基地址为 0x00400000、模块总大小 0x00007000，按核心公式 “RVA = VA - 基地址” 计算，即 0x00401020 - 0x00400000 = 0x00001020。最终结果显示，VA 0x00401020 对应的 RVA 为 0x00001020，且该 RVA 在模块范围内，转换有效。

```
[PEVIEW] # VaToRva --va 00401020
--------------------------------------------------------------------------------
[*] VA转RVA转换开始:
[*] 目标VA地址:0x00401020
[*] 模块基地址（ImageBase）:0x00400000
[*] 模块总大小（SizeOfImage）:0x00007000
------------------------------------------------------------
    转换过程:
    RVA = VA - 基地址
    RVA = 0x00401020 - 0x00400000 = 0x00001020
------------------------------------------------------------
    最终结果:
    VA:0x00401020 --> RVA:0x00001020（有效，在模块范围内）
--------------------------------------------------------------------------------
```

相对地址转虚拟地址

通过RvaToVa --rva 00001020命令，将目标相对地址（RVA）0x00001020 转换为虚拟地址（VA），转换过程如下：明确模块基地址为 0x00400000、模块总大小 0x00007000，依据核心公式 “VA = 基地址 + RVA” 计算，即 0x00400000 + 0x00001020 = 0x00401020。校验显示该 RVA 位于有效节区内，VA 有效，最终转换结果为：RVA 0x00001020 对应 VA 0x00401020。

```
[PEVIEW] # RvaToVa --rva 00001020
--------------------------------------------------------------------------------
[*] RVA转VA转换开始:
[*] 目标RVA地址:0x00001020
[*] 模块基地址（ImageBase）:0x00400000
[*] 模块总大小（SizeOfImage）:0x00007000
------------------------------------------------------------
    转换过程:
```

```
    VA = 基地址 + RVA
    VA = 0x00400000 + 0x00001020 = 0x00401020
------------------------------------------------------------
    校验结果:
    RVA在有效节区内，VA有效
    最终结果:
    RVA:0x00001020 --> VA:0x00401020
--------------------------------------------------------------------------------
```

接口版

PEVIEW 接口版运行后，将自动在本地主机的 8000 端口启动 HTTP 接口侦听服务，为外部工具或脚本提供标准化的 PE 文件分析功能调用入口。该接口支持多种调用方式，既可以通过 POSTMAN、Apifox 等可视化接口测试工具发起请求，也可使用 CURL 命令行工具执行快速调用，还能集成到 Python 等编程语言的脚本中，实现自定义分析流程的自动化。

使用CURL调用

接口版的所有后续分析功能（如头部查询、节表解析、反汇编等）均依赖前置操作，通过Open接口成功打开待分析的可执行文件（仅支持本地磁盘路径），只有完成文件加载后，后续调用其他功能接口才能返回有效数据，以CURL调用为例，如下是分别调用打开文件及获取文件基本信息的输出结果。

```
curl -X POST "http://localhost:8000" ^
-H "Content-Type: application/json" ^
-d "{\"class\":\"PE\",\"interface\":\"Open\",\"params\":[\"e:\\\\win32.exe\"]}"

{
  "status": "success",
  "result": {
    "message": "The PE file has been successfully opened",
    "file_path": "e:\\\\win32.exe",
    "file_size": 14848
  },
  "timestamp": 26924437
}

curl -X POST "http://localhost:8000" ^
-H "Content-Type: application/json" ^
-d "{\"class\":\"PE\",\"interface\":\"FileBasicInfo\",\"params\":[]}"

{
  "status": "success",
  "result": {
    "file_basic_info": {
      "file_path": "e://win32.exe",
      "file_size_bytes": 14848,
      "file_size_kb": 14.5,
      "file_size_mb": 0.01416015625,
      "file_attributes": "file; ",
      "create_time": "2025-10-17 20:46:43",
      "modify_time": "2025-10-17 20:46:43",
```

```
      "map_base_address": "0x01510000",
      "map_base_address_dec": 22085632
    },
    "pe_identifier": {
      "dos_signature_hex": "0x00005a4d",
      "dos_signature_dec": 23117,
      "dos_signature_desc": "Valid DOS signature (MZ)",
      "pe_header_offset_hex": "0x00000100",
      "pe_header_offset_dec": 256,
      "nt_signature_hex": "0x00004550",
      "nt_signature_dec": 17744,
      "nt_signature_desc": "Valid PE signature (pe00)",
      "machine_type_hex": "0x0000014c",
      "machine_type_dec": 332,
      "machine_type_desc": "x86 (32bit)",
      "section_count": 5,
      "nt_timestamp_hex": "0x68f23ab3",
      "nt_timestamp_dec": 1760705203,
      "nt_timestamp_desc": "1601-01-01 08:02:56",
      "nt_characteristics_hex": "0x00000102",
      "nt_characteristics_dec": 258,
      "nt_characteristics_desc": "Executable; "
    },
    "optional_header_info": {
      "entry_point_rva_hex": "0x000015bb",
      "entry_point_rva_dec": 5563,
      "image_base_hex": "0x00400000",
      "image_base_dec": 4194304,
      "image_size_hex": "0x00007000",
      "image_size_dec": 28672,
      "section_alignment_hex": "0x00001000",
      "section_alignment_dec": 4096,
      "file_alignment_hex": "0x00000200",
      "file_alignment_dec": 512,
      "subsystem_hex": "0x00000002",
      "subsystem_dec": 2,
      "subsystem_desc": "Windows GUI (Graphical interface)",
      "dll_characteristics_hex": "0x00008140",
      "dll_characteristics_dec": 33088,
      "dll_characteristics_desc": "ASLR support; DEP support; ",
      "stack_reserve_size_hex": "0x00100000",
      "stack_reserve_size_dec": 1048576,
      "stack_commit_size_hex": "0x00001000",
      "stack_commit_size_dec": 4096,
      "heap_reserve_size_hex": "0x00100000",
      "heap_reserve_size_dec": 1048576,
      "heap_commit_size_hex": "0x00001000",
      "heap_commit_size_dec": 4096
    },
    "message": "PE File basic information parsing succeeded"
  },
  "timestamp": 27166000
}
```

```
curl -X POST "http://localhost:8000" ^
-H "Content-Type: application/json" ^
-d "{\"class\":\"PE\",\"interface\":\"Close\",\"params\":[]}"

{
  "status": "success",
  "result": {
    "message": "PE file closed, resource released"
  },
  "timestamp": 27428375
}
```

使用PYTHON调用

通过使用`peview_client.py`脚本也可以快速与PE服务器建立通信，可在控制台执行pip命令安装此工具包。

```
pip install peview-client

Collecting peview-client
  Downloading peview_client-4.0.0-py3-none-any.whl.metadata (1.5 kB)
Downloading peview_client-4.0.0-py3-none-any.whl (19 kB)
Installing collected packages: peview-client
Successfully installed peview-client-4.0.0
```

该脚本是一个轻量级 Python 客户端工具，通过封装 HTTP 请求逻辑与接口调用细节，简化了与 PEVIEW 接口版服务端的通信流程，支持对 PE 文件进行各类解析和操作，包括文件基础信息查询、结构解析（DOS 头、NT 头、节区等）、导入 / 导出表分析、地址转换、数据读写与搜索等功能。

具体的函数名称及功能描述请参阅下表所示：

函数名	功能描述
open_file(file_path)	打开 PE 文件（所有其他操作的前置步骤），需传入文件绝对路径
close_file()	关闭 PE 文件并释放服务端资源
get_basic_info()	查询 PE 文件基础信息（文件属性、DOS/NT 头标识、可选头关键信息等）
show_dos_head()	查询 DOS 头完整信息（IMAGE_DOS_HEADER）
show_nt_head()	查询 NT 头完整信息（含 NT 签名、IMAGE_FILE_HEADER、IMAGE_OPTIONAL_HEADER32）
show_section()	查询所有节区信息（IMAGE_SECTION_HEADER）
show_optional_data_directory()	查询可选头数据目录表（16 个标准目录项，如导出表、导入表等）

函数名	功能描述
show_import_by_dll()	查询所有导入 DLL 列表（含每个 DLL 的 INT/IAT 地址、时间戳等）
show_import_by_name(dll_name)	查询指定 DLL 的导入函数列表（区分序号导入 / 名称导入）
show_import_by_function(target_func, case_sensitive=False, check_ordinal=True)	按函数名 / 序号匹配导入函数（支持模糊匹配）
show_import_all()	遍历所有导入模块和函数（全局导入表完整信息）
show_export()	查询导出表完整信息（含导出函数、序号、转发函数等）
show_fix_reloc_page()	查询重定位表分页情况（所有重定位块的基础信息）
show_fix_reloc(target_rva)	遍历指定 RVA 的重定位数据或所有重定位数据（支持 "all" 参数）
show_resource()	查询资源表完整信息（3 级目录结构：类型目录→名称 / ID 目录→语言目录）
va_to_foa(va)	VA（虚拟地址）转换为 FOA（文件偏移地址）
rva_to_foa(rva)	RVA（相对虚拟地址）转换为 FOA（文件偏移地址）
foa_to_va(foa)	FOA（文件偏移地址）转换为 VA（虚拟地址）
va_to_rva(va)	VA（虚拟地址）转换为 RVA（相对虚拟地址）
rva_to_va(rva)	RVA（相对虚拟地址）转换为 VA（虚拟地址）
get_hex_ascii(start_addr, addr_len)	读取文件指定范围的十六进制和 ASCII 数据（16 字节 / 行格式化）
search_signature(start_addr, search_len, sig_str)	特征码搜索（支持通配符??）
search_string(start_addr, search_len, target_str)	ASCII 字符串搜索（严格匹配）
get_module_status()	查询模块保护方式（ASLR/DEP/CFG 等安全特性）
get_process_address(dll_name, func_name)	获取指定 DLL 的导出函数地址（模拟 LoadLibrary+GetProcAddress）
disassemble_code(start_foa, disasm_len)	反汇编指定文件范围（x86及x64 架构）
add_calculator(x, y)	十六进制加法计算器（DWORD 无符号运算）
sub_calculator(x, y)	十六进制减法计算器（DWORD 无符号运算，负数自动模 2^32）

以打开PE文件及读取基本信息为例，其调用功能如下所示：

```python
from peview_client import *

if __name__ == "__main__":
    # 初始化配置
    config = Config(address="127.0.0.1", port=8000)

    # 验证服务器是否可用
    if not config.is_server_available():
        print("服务器未启动或端口不可达，请检查服务状态")
    else:
        print("服务器连接正常，可执行后续操作")

    # 创建PE接口实例
    pe = PE(config)

    # 1. 打开本地PE文件
    open_result = pe.open_file("e:\\win32.exe")
    print("文件打开结果：", open_result)

    # 2. 查询文件基础信息（需在open成功后调用）
    basic_info = pe.get_basic_info()
    print("\n文件基础信息：")
    print("文件路径：", basic_info["file_basic_info"]["file_path"])
    print("PE签名：", basic_info["pe_identifier"]["nt_signature_desc"])
    print("入口点RVA：", basic_info["optional_header_info"]["entry_point_rva_hex"])

    pe.close_file()
```

MCP Server

PEView 是一款常用的 PE（可执行文件）分析工具，主要用于查看 PE 文件结构、解析文件属性等。为拓展其功能边界，使其具备大模型调用能力（如借助大模型辅助 PE 文件分析、异常检测等），我们通过适配 MCP（Model Control Protocol，模型控制协议）接口开发了专属支持方案。

在 MCP 项目目录下，已存放适配文件 peview_mcp 该文件是专门为 PEView 工具开发的 AI 功能接口模块，核心作用是建立 PEView 与大模型服务的通信桥梁，实现工具与大模型的指令交互、数据传输，确保大模型调用功能稳定运行。

依赖库安装

使用前需先安装fast-mcp库（MCP 接口运行的核心依赖），确保本地已配置 Python 3.7 及以上环境，打开命令行执行以下命令即可完成安装：

```
pip install fastmcp
```

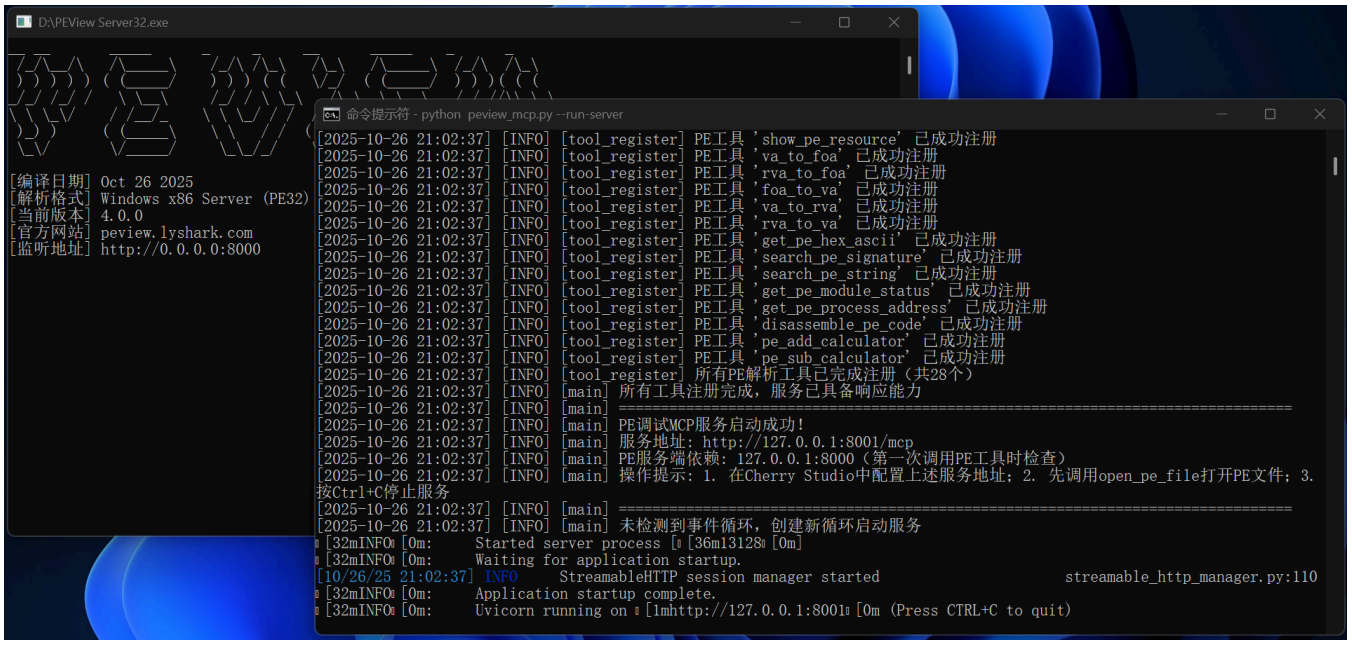
客户端工具准备

需搭配一款 MCP 协议兼容的客户端程序使用，推荐选用Cherry Studio（界面友好、操作便捷，适配 MCP 接口的常用工具）。若需替换其他客户端，只需选择支持 MCP 接口协议的工具即可，无强制限制。

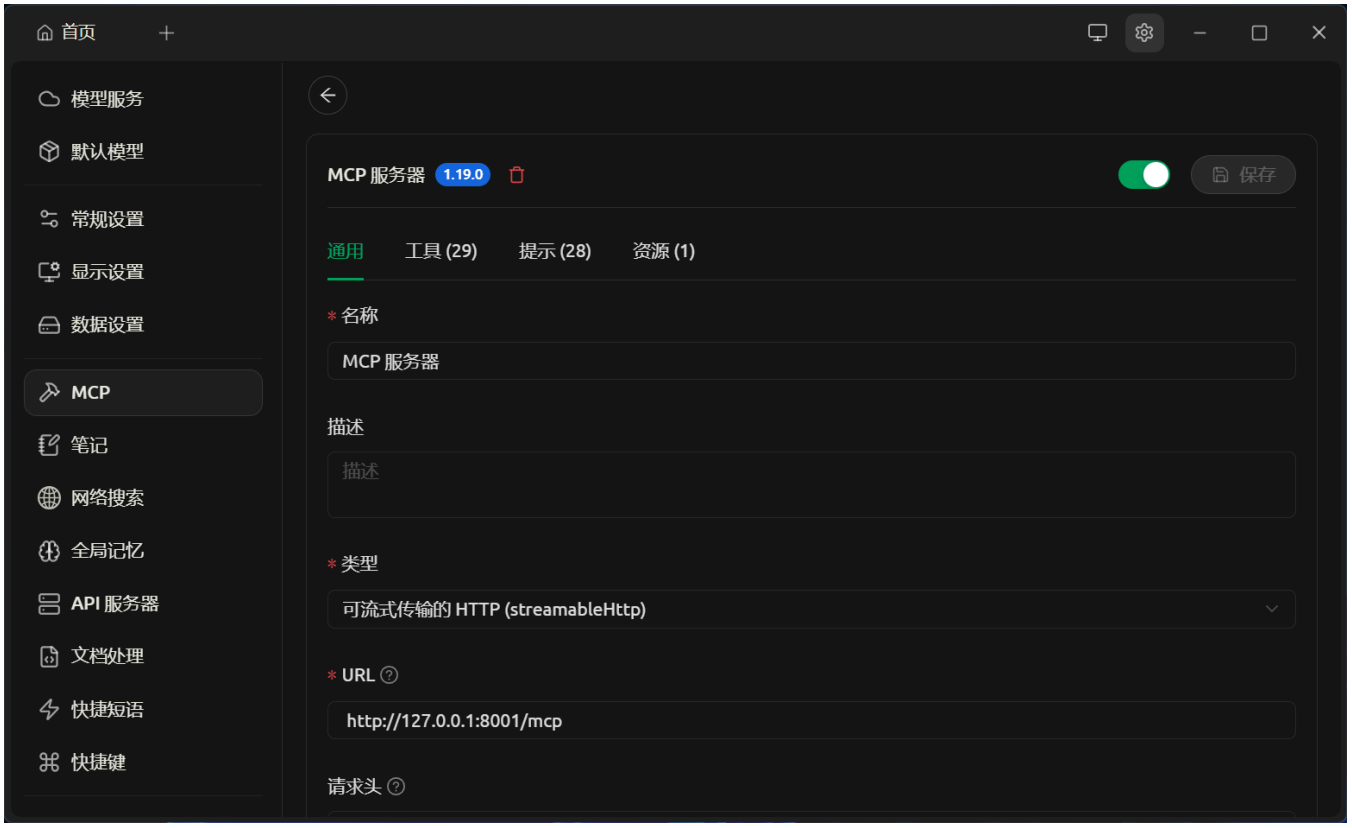
CherryStudio下载地址：https://www.cherry-ai.com/download

启动PEView及MCP

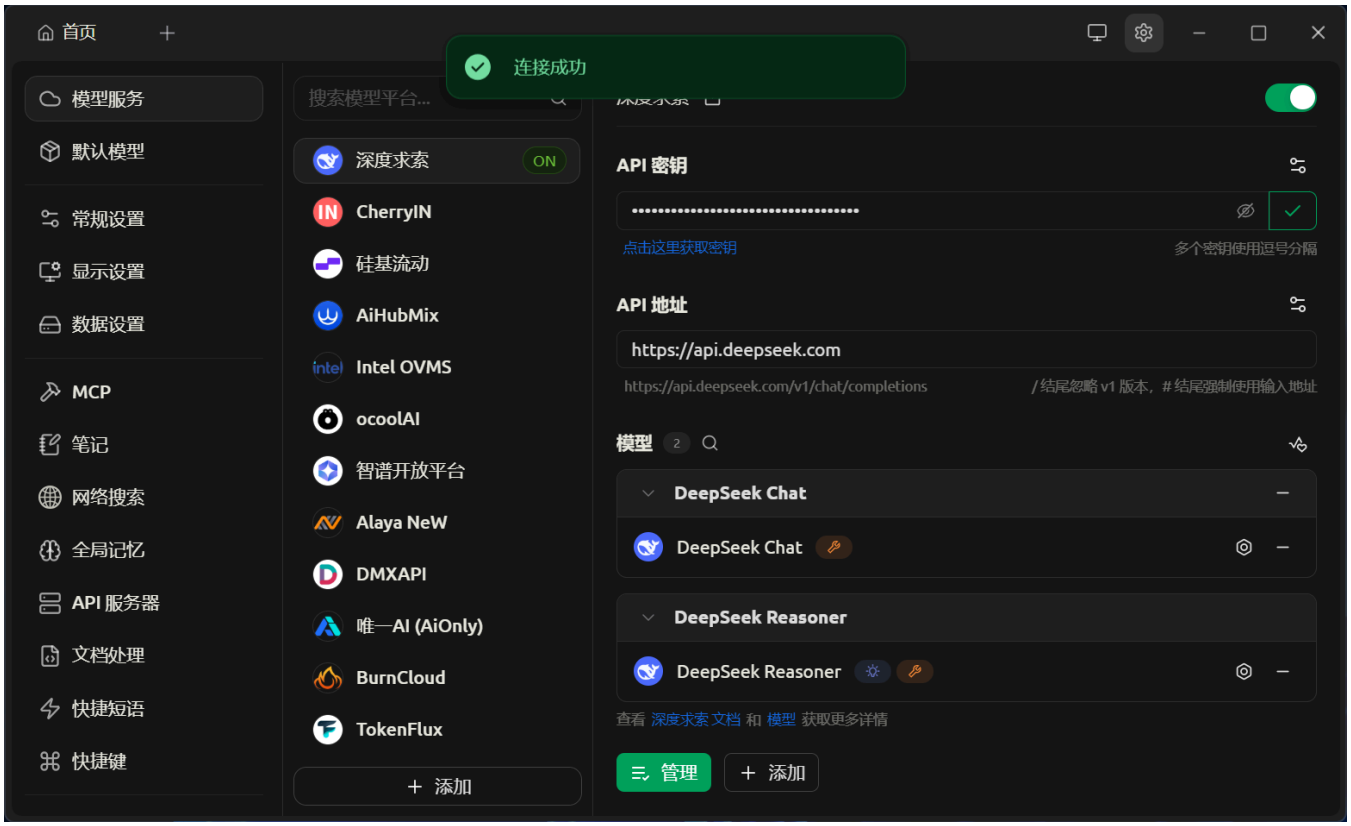
首先确保本地已配置 Python 运行环境，且已安装 PEView 相关依赖包（如 CherryPy 等）。打开命令行工具（如 Windows 的 CMD、PowerShell 或 Linux 的 Terminal），切换至 PEView 程序所在目录，执行命令python peview_mcp.py --run-server启动PEView Server与PEView MCP双服务器。执行成功后，命令行窗口会显示服务启动日志（如端口监听信息、服务状态 “running” 提示），对应截图中服务端程序运行界面。



打开 CherryStudio 工具后，在主界面找到顶部或右侧的 “设置” 按钮（通常为齿轮图标），点击后在弹出的设置窗口中选择 “MCP” 选项卡。在 “MCP 服务端地址” 输入框中，填写已启动的 MCP 服务器地址（格式通常为 “IP 地址：端口号”，本地测试可填 “127.0.0.1: 默认端口”，具体端口需参考服务端启动日志），输入完成后点击 “保存” 或 “应用”。此时 CherryStudio 客户端会自动与 MCP 服务端建立连接，成功后可在设置界面或主界面看到 “工具已识别” 的提示（如截图中工具信息显示栏出现对应设备 / 服务标识）。

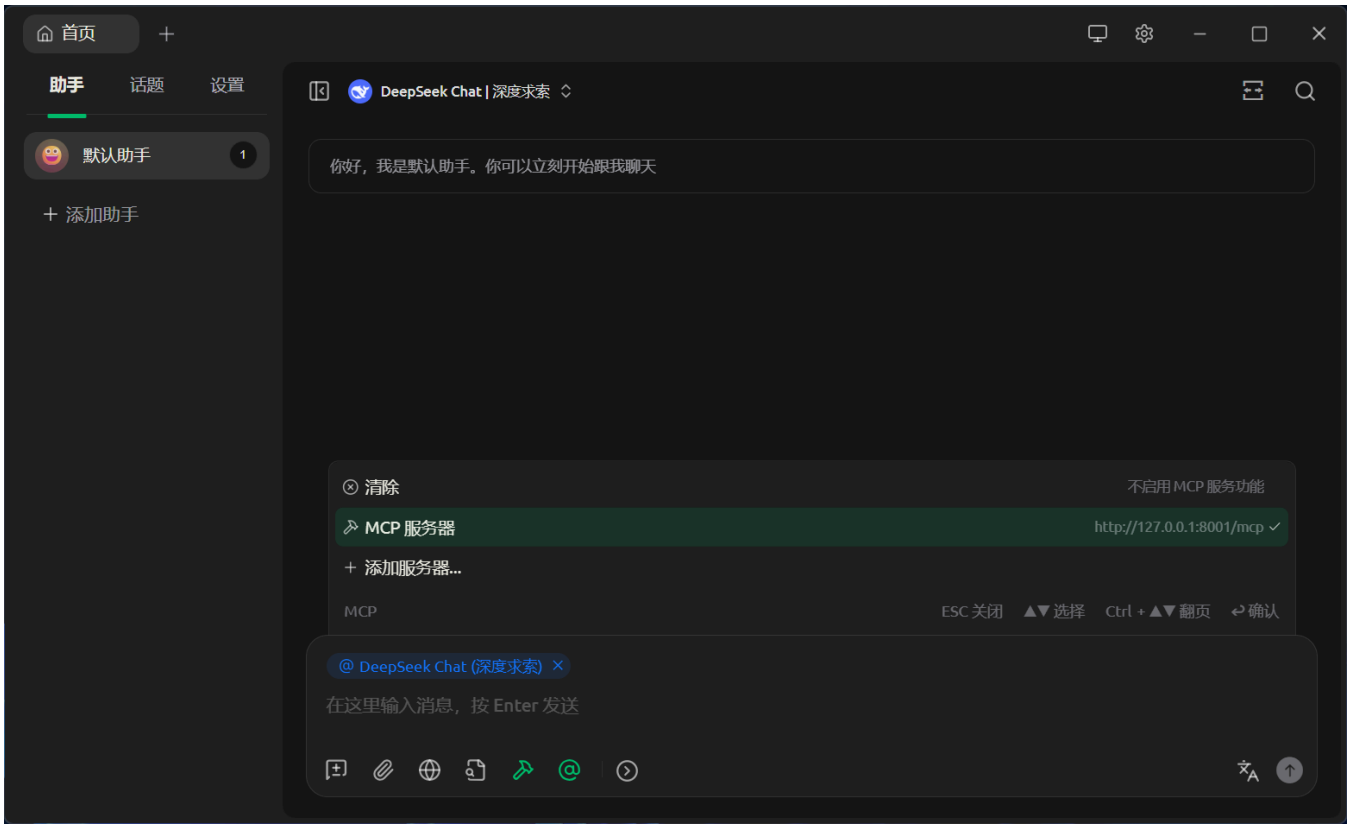


在 CherryStudio 设置界面，切换至 “模型服务” 列表页，找到 “深度求索” 模型选项（若列表中无该选项，需先确认已添加对应模型插件）。点击 “深度求索” 模型右侧的 “配置” 或 “连接” 按钮，在弹出的密钥输入框中，填入从深度求索官方平台申请的有效 API 密钥（需确保密钥未过期且具备模型调用权限），输入后点击 “确认”。系统会自动校验密钥有效性，校验通过后弹出 “链接成功” 弹窗（如截图中提示框），同时模型服务列表中 “深度求索” 的状态会变为 “已连接”。

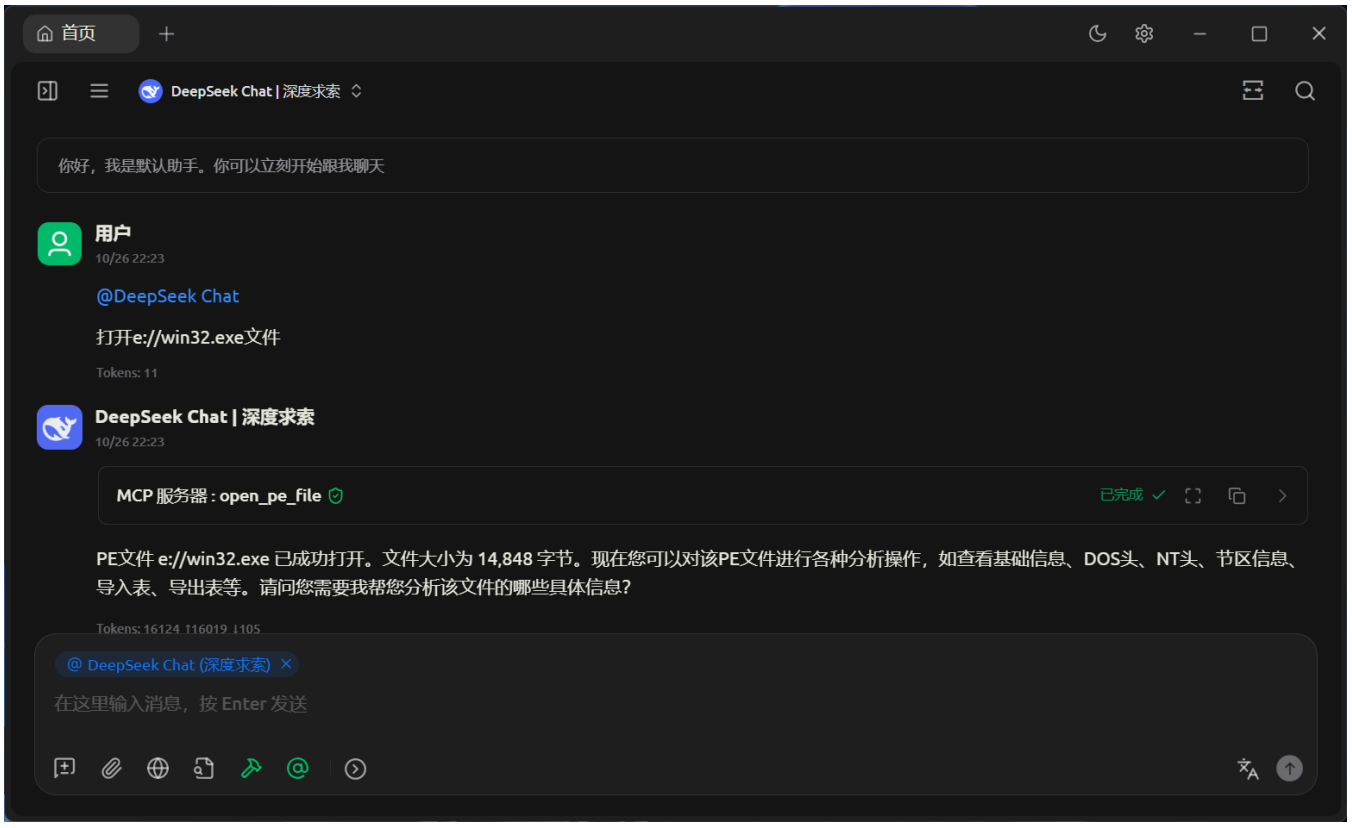


完成模型连接后，返回 CherryStudio 的 “默认模型” 配置页（通常在 “模型服务” 列表页下方或单独的 “模型设置” 选项卡中），在模型选择列表中找到 “深度求索” 模型，勾选其前方的复选框。勾选后需点击页面底部的 “保存设置” 按钮，确保默认模型配置生效（若未保存，后续对话可能仍使用原默认模型），对应截图中默认模型勾选界面。

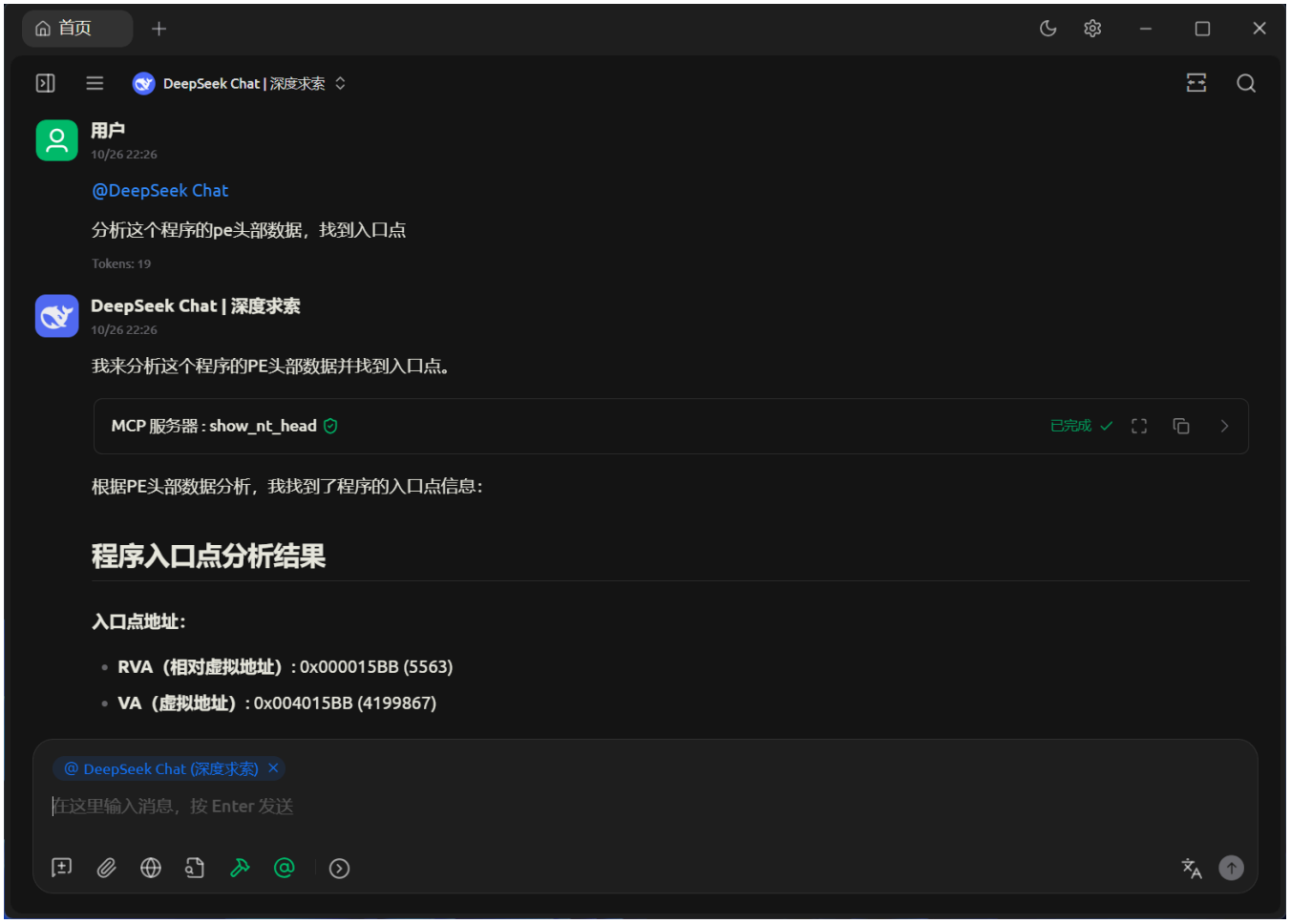
从 CherryStudio 主界面进入对话功能模块（如 “AI 对话”“工具交互” 窗口），在对话框顶部或右侧的 “服务配置” 区域，确认 “MCP 服务器” 已选择步骤 2 中配置的地址（若未选中需手动下拉选择），同时 “当前使用模型” 已自动切换为 “深度求索”（若未切换，可点击模型下拉菜单重新选中），确保对话框与服务端、模型的连接一致，对应截图中对话框的服务器与模型设置界面。



在对话框的输入框中，按照指令格式输入 “打开 E 盘下的 [具体文件名。后缀]”（如 “打开 E 盘下的 test.exe”），点击 “发送” 按钮提交指令。AI 接收指令后，会与 MCP 服务端交互并执行文件读取操作，执行完成后返回结果信息（如截图中显示 “已成功定位并打开 E 盘下目标文件，文件路径：E:\xxx.xxx”“文件基础信息：大小 xxKB、修改时间 xxxx-xx-xx” 等），需确认返回信息中无 “文件不存在”“权限不足” 等错误提示。



基于已打开的文件，在对话框中继续输入指令 “分析当前打开程序的 NT 头部数据，并提取其入口点地址”，提交指令后，AI 会调用 PEView 的程序解析功能，读取 NT 头部中的 IMAGE_OPTIONAL_HEADER 结构数据，筛选出 AddressOfEntryPoint 字段对应的数值。最终返回结果会包含 NT 头部关键数据（如签名、节表数量、时间戳）及入口点信息（通常以十六进制格式呈现，如 “入口点地址：0x00401000”），对应截图中 AI 返回的 NT 头部分析与入口点结果界面。



至此，我们封装了32个工具功能，用户可以依次提问并获取想要的结果。