文件验证

MZ签名检查，检查文件开头两字节是否为4D 5A，失败则报错“Not a valid Win32 application”。

定位NT头，读取e\_lfanew值（IMAGE\_DOS\_HEADER -> e\_lfanew）并检验地址有效性，失败则报错“Invalid PE header offset”。

检验NT头签名（IMAGE\_NT\_HEADERS -> Signature）是否为00 00 45 50，失败则报错“Invalid PE signature”。

检验CPU架构，验证Machine值（IMAGE\_NT\_HEADERS -> IMAGE\_FILE\_HEADER -> Machine）是否和CPU架构兼容，失败则报错“Unsupported machine type”。

检验可选头大小，验证SizeOfOptionalHeader（IMAGE\_NT\_HEADERS -> IMAGE\_FILE\_HEADER -> SizeOfOptionalHeader）是否与IMAGE\_OPTIONAL\_HEADER长度匹配，失败则报错“Invalid optional header size”。

检验文件头魔术字，验证Magic值（IMAGE\_NT\_HEADERS -> IMAGE\_OPTIONAL\_HEADER -> Magic）有效性，失败则报错“Invalid PE format”。

检验入口点有效性，检查AddressOfEntryPoint（IMAGE\_NT\_HEADERS -> IMAGE\_OPTIONAL\_HEADER -> AddressOfEntryPoint是否落在任一节的VirtualAddress 范围内，失败则报错“Invalid entry point”。

检验内存对齐粒度，确认 SectionAlignment ≥ FileAlignment（IMAGE\_NT\_HEADERS -> IMAGE\_OPTIONAL\_HEADER -> SectionAlignment/FileAlignment），失败则报错“Invalid alignment values”。

校验映像大小，计算 SizeOfImage（IMAGE\_NT\_HEADERS -> IMAGE\_OPTIONAL\_HEADER -> SizeOfImage）是否 ≥ 所有头 + 节的内存总和，失败则报错“Corrupt image size”。

内存映射

遍历节表，按NumberOfSections（IMAGE\_NT\_HEADERS -> IMAGE\_FILE\_HEADER -> NumberOfSections）循环处理每个IMAGE\_SECTION\_HEADER，处理中检验PointerToRawData（IMAGE\_SECTION\_HEADER -> PointerToRawData）是否超出文件大小和Characteristics（IMAGE\_SECTION\_HEADER -> Characteristics）权限是否合法（如可执行节不可写），失败则报错 “Invalid section table”。

分配虚拟内存，调用VirtualAlloc（Windows API中用于操作虚拟内存的核心函数）按ImageBase（IMAGE\_NT\_HEADERS -> IMAGE\_OPTIONAL\_HEADER -> ImageBase）和SizeOfImage（IMAGE\_NT\_HEADERS -> IMAGE\_OPTIONAL\_HEADER -> SizeOfImage）保留地址空间，失败则报错“Memory allocation failed”。

映射头部数据，复制 DOS头 + NT头 + 节表到 ImageBase。

加载节数据，按PointerToRawData和SizeOfRawData（IMAGE\_SECTION\_HEADER -> PointerToRawData/SizeOfRawData）将文件内容写入VirtualAddress（IMAGE\_SECTION\_HEADER -> VirtualAddress）对应内存。

设置内存保护，根据Characteristics（IMAGE\_SECTION\_HEADER -> Characteristics）调用VirtualProtect（Windows API 中用于动态修改内存页保护属性的关键函数）。

动态链接处理

解析导入表，从 DataDirectory[1]定位IMAGE\_IMPORT\_DESCRIPTOR数组关键数据（IMAGE\_OPTIONAL\_HEADER -> IMAGE\_DATA\_DIRECTORY[1] -> VirtualAddress -> IMAGE\_IMPORT\_DESCRIPTOR -> Name/OriginalFirstThunk/FirstThunk），对每个DLL：按 Name 字段加载依赖库（LoadLibrary），遍历 OriginalFirstThunk 解析函数名/序号，填充 FirstThunk 指向的 IAT。

应用重定位（ASLR），若实际加载地址 ≠ ImageBase则从 DataDirectory[5] 读取 IMAGE\_BASE\_RELOCATION（IMAGE\_OPTIONAL\_HEADER -> IMAGE\_DATA\_DIRECTORY[5] -> VirtualAddress -> IMAGE\_BASE\_RELOCATION），按重定位项修正代码/数据中的绝对地址。

执行准备

初始化线程局部存储（TLS），若存在 DataDirectory[9]，调用 TLS 回调函数

调用入口点，跳转至 ImageBase + AddressOfEntryPoint，是EXE → 进入 mainCRTStartup，是DLL → 调用 DllMain（若存在）。