动态加载

`LoadLibrary`用于动态加载模块，得到的句柄为**DLL**在进程的内存中的首地址，`GetModuleHandle`用于获取模块的句柄

`GetProcAddress`用于获取模块中的函数（变量）的地址

动态加载拿不到类，只能拿到函数、变量对象

`FreeLibrary`用于释放指定模块

def导出

跨编译器使用而存在的，使得导出后的名称不进行名称粉碎，而以指定的名称导出

语法：

- `entryname[=internalname]`
  - `MyAdd=Add`： `Add`函数以名字`MyAdd`导出
- `[@ordinal]`
  - 序号 `WORD`类型
- `NONAME`
  - 只以序号导出，不导出名字
- `DATA`
  - 导出的是数据
- `PRIVATE`
  - 私有的，只能显示使用，不能隐式使用

最小的 DEF 文件必须包含以下模块定义语句：

- 在文件中的第一个语句必须是 LIBRARY 语句。 此语句将 DEF 文件标识为属于 DLL。 LIBRARY 语句后跟的 DLL 的名称。 链接器将此名称放在 DLL 的导入库中
- EXPORTS 语句列出名称和 DLL 导出的函数的序号值 （可选）

```
LIBRARY   BTREE
EXPORTS
   Insert   @1
   Delete   @2
   Member   @3
   Min   @4
```

DllMain

DLL的加载和卸载的时候调用，用于**DLL**的初始化和资源的释放

- `DllMain`是可选的

- `DLL_PROCESS_ATTACH` 在进程启动时调用
- `DLL_PROCESS_DETACH` 在进程结束时调用

返回值：

> 当系统使用DLL_PROCESS_ATTACH值调用DllMain函数时，如果成功，该函数将返回TRUE，如果初始化失败，则返回FALSE。
>
> 如果由于进程使用LoadLibrary函数而调用DllMain时返回值为false，则LoadLibrary将返回NULL。(系统立即使用DLL_PROCESS_DETACH调用入口点函数并卸载DLL。)。
>
> 如果在进程初始化过程中调用DllMain时返回值为false，则进程将终止并返回错误。若要获取扩展错误信息，请调用GetLastError。
>
> 静态加载会弹出142错误。

```
BOOL WINAPI DllMain(
    HINSTANCE hinstDLL,  // handle to DLL module
    DWORD fdwReason,     // reason for calling function
    LPVOID lpReserved )  // reserved
{
    // Perform actions based on the reason for calling.
    switch( fdwReason )
    {
        case DLL_PROCESS_ATTACH:
            // Initialize once for each new process.
            // Return FALSE to fail DLL load.
            break;

        case DLL_THREAD_ATTACH:
            // Do thread-specific initialization.
            break;

        case DLL_THREAD_DETACH:
            // Do thread-specific cleanup.
            break;

        case DLL_PROCESS_DETACH:
            // Perform any necessary cleanup.
            break;
    }
    return TRUE;  // Successful DLL_PROCESS_ATTACH.
}
```

在main函数之前执行代码：

1. DllMain
2. 全局对象的构造
3. 全局变量使用函数赋值

DLL劫持

函数转发

`a.dll`导出声明，`b.dll`里面时实现，`a.dll`通过`#pragma comment(libker, "/EXPORT:函数名=b.函数名")`