DirectX 12

异步图书
www.epubit.com.cn

Direct3D 12 编程指南

- 全面讲解了 Direct3D 12 新版本的流水线状态对象、根签名、捆绑包、命令列表、资源堆和屏障等新功能。
- 以实例程序演示了 Direct3D 12 的渲染流水线和计算流水线，帮助读者透彻掌握 Direct3D 12 的底层实现，为读者今后用 HLSL5.1 编写着色器打下扎实的基础。
- 综合应用部分将本书各章中的知识巧妙地结合在一起，便于学以致用。

张羽乔 编著

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

Direct3D 12 编程指南

张羽乔 编著

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

Direct 3D 12编程指南 / 张羽乔编著. -- 北京 : 人民邮电出版社, 2017.5
ISBN 978-7-115-45025-8

Ⅰ. ①D… Ⅱ. ①张… Ⅲ. ①DirectX软件－程序设计－指南 Ⅳ. ①TP317-62

中国版本图书馆CIP数据核字(2017)第066826号

内 容 提 要

本书系统介绍了 Direct3D 12 各方面的知识，包括开始前的准备工作，如何创建 DirectX 12 项目，编程后的步骤，以及关于多线程、命令队列、资源结构、图形流水线、计算流水线和 GPU 内部传参等内容，最后讲解了一个基于 Direct3D 12 实现的字体引擎。本书重点介绍 Direct3D 12 的知识，而且减少对计算机图形学中通用知识的介绍，因为读者完全可以在其他的书中得到这些知识。

本书的适用对象为面向 Windows 平台的 3D 开发人员。

◆ 编　　著　张羽乔
责任编辑　张　涛
执行编辑　张　爽
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000　1/16
印张：16
字数：387 千字　　2017 年 5 月第 1 版
印数：1－2 000 册　　2017 年 5 月河北第 1 次印刷

定价：59.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广字第 8052 号

前　言

Direct3D 12 是 Windows 10 新推出的 3D 图形库 API。本书将介绍 Direct3D 12 相关的各个方面的知识，主要内容如下。

第 1 章主要介绍开始前的准备工作，包括如何创建 Direct3D 12 项目和 COM 简介等内容。

第 2 章开始学习简单的 Direct3D 12 编程内容，包括设备、命令队列、交换链，以及渲染等内容。

第 3 章主要介绍多线程的相关内容，包括命令队列、命令分配器和命令列表等内容。

第 4 章主要介绍资源的结构、创建，以及 CPU、GPU 访问资源等内容。

第 5 章在第 4 章的基础上，更加深入地介绍图形流水线的相关内容。

第 6 章介绍计算流水线状态、启动等内容。

第 7 章简单介绍 GPU 内部传参的知识。

第 8 章结合本书所讲解的全部内容，完成一个基于 Direct3D 12 实现的字体引擎。

本书面向的读者

本书适合于希望使用 Direct3D 12 进行编程的读者，并且本书假定读者已经掌握了 C/C++ 编程语言的相关知识。本书并不要求读者掌握 Win32 And COM API 或之前版本的 Direct3D 的相关知识，但是已经掌握了这些知识的读者可以更轻松地阅读本书。

勘误

由于作者水平有限，本书中难免存在错漏之处。如果读者发现了本书中的错误，请反馈到作者的邮箱 D3D12CoreProgram@163.com 或本书编辑的邮箱 zhangshuang@ptpress.com.cn，作者将尽力对其进行更正。

进一步阅读

在完成 C++语法的学习后，应当进一步学习 C++的相关算法。与读者学习 C++的过程类似，学习完本书中的 Direct3D 12 语法后，读者可继续学习一些相关的算法。

如果读者需要学习渲染流水线的相关算法，那么《Real-Time Rendering》（ISBN：9781568814247）和《Physically Based Rendering》（ISBN：9780123750792）是两本不错的参考书。

如果读者需要学习计算流水线的相关算法，可以参考人民邮电出版社的《OpenCL 实战》（ISBN：9787115347343）一书。此外，Bullet 是一个基于 OpenCL 的开源物理引擎，AMD 也发布了很多基于 OpenCL 的开源项目，NVIDIA 也发布了很多基于 OpenCL 的示例程序，这些都可供读者自行参考学习。

目　录

第 1 章　开始前的准备

本章介绍在开始 Direct3D 12 编程前需要做的准备工作。

1.1 创建 DirectX 12 项目

下面介绍如何配置开发环境并创建一个贯穿本书使用的 DirectX 12 项目。

1.1.1　安装 Windows 10 和 Visual Studio 2015

读者可以从相应的官方网站上获取 Windows 10 和 Visual Studio 2015 的免费试用版。

在安装 Visual Studio 2015 时，应当确保安装了 Visual C++和 Windows 10 SDK 这两个功能，如图 1-1 所示。

图 1-1　配置 Visual Studio 2015 功能

1.1.2　新建解决方案和项目

安装完成后，打开 Visual Studio 2015，新建一个 Win32 空项目。

在菜单栏中选择“文件”→“新建”→“项目”，如图 1-2 所示。

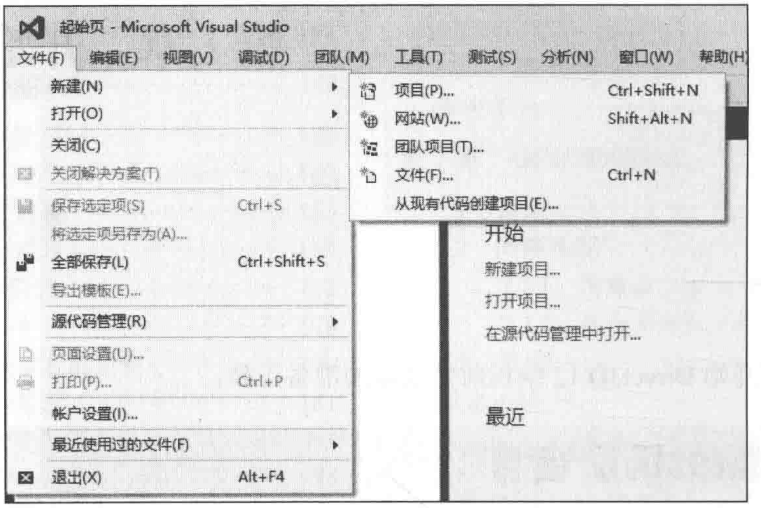

图 1-2　新建项目

在“Visual C++”→“Win32”中选择“Win32 项目”，如图 1-3 所示。

图 1-3　Win32 项目

在“附加选项”中勾选“空项目”，如图 1-4 所示。

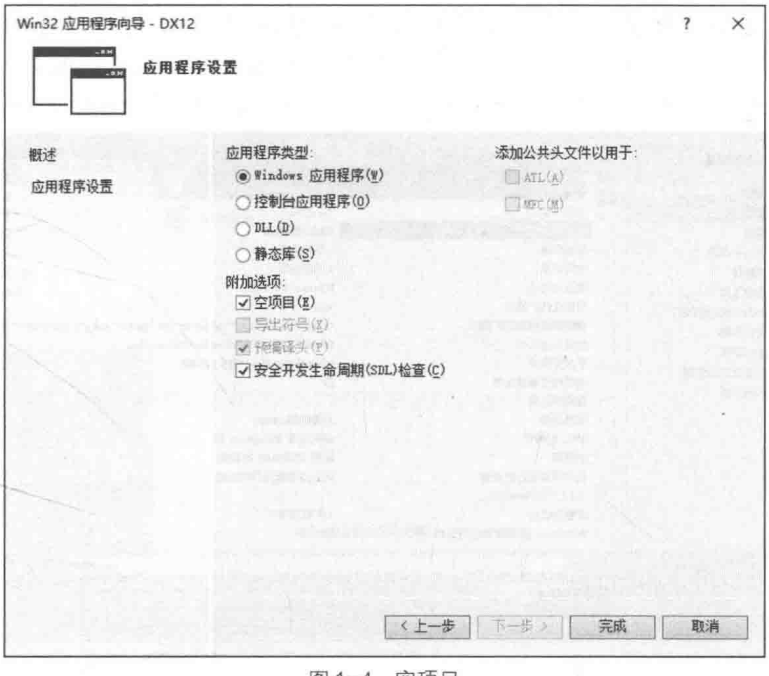

图 1-4 空项目

1.1.3 配置使用 Windows 10 SDK

在解决方案管理器中右击“DX12”项目，选择“属性”，如图 1-5 所示。

图 1-5 选择“属性”

选择所有配置和所有平台，在“配置属性”→“常规”中，将“目标平台”版本设置为“10.0.10586.0”，以使用 Windows 10 SDK 而不是默认的 Windows 8.1 SDK，如图 1-6 所示。

图 1-6　Windows 10 SDK

1.1.4　新建 main.cpp

在“解决方案管理器”中右击“DX12”项目，在下拉菜单栏中选择“添加”→“新建项”，如图 1-7 所示。

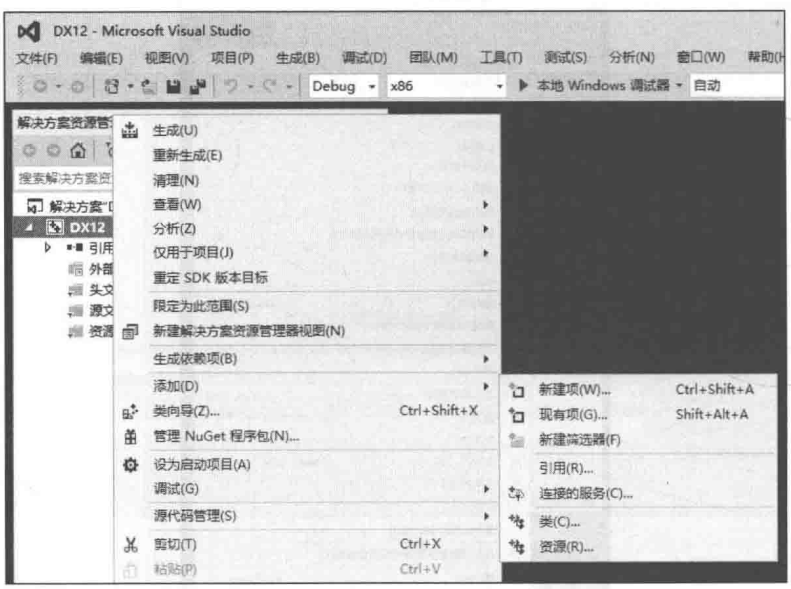

图 1-7　添加“新建项”

在“Visual C++”→“代码”中选择“C++文件（.cpp）”，并将名称设置为“main.cpp”，如图 1-8 所示。

图 1-8 C++文件（.cpp）

在 main.cpp 中添加以下代码，用于创建一个 Win32 窗口并创建一个渲染线程。代码中已添加了注释以帮助读者理解，如果读者希望更深入地了解相关知识，可以参阅《Windows 程序设计》（ISBN：9787302227397）和《Windows 核心编程》（ISBN：9787302184003）。

```
#include <sdkddkver.h>//表明当前使用的 SDK 的版本(即 Windows 10 SDK)
#define WIN32_LEAN_AND_MEAN
#include <Windows.h>
#include <process.h> //含有下文中用于创建线程的_beginthreadex 的定义

//_beginthreadex 的参数类型与 Windows 数据类型不匹配，定义了以下 BeginThread 辅助函数
inline HANDLE __stdcall BeginThread(
     LPSECURITY_ATTRIBUTES lpThreadAttributes,
     SIZE_T dwStackSize,
     LPTHREAD_START_ROUTINE lpStartAddress,
     LPVOID lpParameter,
     DWORD dwCreationFlags,
     LPDWORD lpThreadId
)
{
     return reinterpret_cast<HANDLE>(_beginthreadex(static_cast<void *>(lpThreadAttributes),
     dwStackSize,
     reinterpret_cast<unsigned(__stdcall *) (void *)>(lpStartAddress),
     lpParameter,
     dwCreationFlags,
     reinterpret_cast<unsigned *>(lpThreadId)));
}

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);//窗口过程的声明
DWORD WINAPI RenderThreadMain(LPVOID);//渲染线程的入口点函数的声明

int APIENTRY wWinMain(HINSTANCE hInstance, HINSTANCE, LPWSTR lpCmdLine, int nCmdShow)//进
程中主线程的入口点函数
{
     {
```

```
        //以下函数用于注册窗口类
        WNDCLASSEXW wcex;//用于描述窗口类的结构体
        wcex.cbSize = sizeof(WNDCLASSEX);
        wcex.style = CS_OWNDC;//缓存窗口的图形设备环境的状态，以提升性能
        wcex.lpfnWndProc = WndProc;//指定窗口过程
        wcex.cbClsExtra = 0;
        wcex.cbWndExtra = 0;
        wcex.hInstance = hInstance;//窗口类所在的模块
        wcex.hIcon = LoadIconW(hInstance, IDI_APPLICATION);//使用系统默认的图标
        wcex.hCursor = LoadCursorW(NULL, IDC_ARROW);//使用系统默认的鼠标指针
        wcex.hbrBackground = NULL;//指定空画刷，阻止 GDI 对窗口进行渲染，以提升性能
        wcex.lpszMenuName = NULL;//没有菜单栏
        wcex.lpszClassName = L"{640CB8AD-56CD-4328-B4D0-2A9DAA951494}";//窗口类名
        wcex.hIconSm = LoadIconW(hInstance, IDI_APPLICATION);//使用系统默认的图标
        RegisterClassExW(&wcex);//注册窗口类

        RECT windowrect;
        windowrect.left = 0;
        windowrect.top = 0;
        windowrect.right = 800;//窗口的客户区的宽度
        windowrect.bottom = 600; //窗口的客户区的高度
        AdjustWindowRect(&windowrect, WS_POPUP | WS_VISIBLE, FALSE);//根据窗口的客户区
大小计算窗口的大小

        //以下函数用于新建窗口
        HWND hWnd = CreateWindowExW(0, //不使用扩展的窗口风格
        L"{640CB8AD-56CD-4328-B4D0-2A9DAA951494}", //上文中注册的窗口类名
        L"《Direct3D 12 核心编程》", //窗口实例的名字
        WS_POPUP | WS_VISIBLE, //没有标题栏的窗口风格，并且显示
        GetSystemMetrics(SM_CXSCREEN) / 2 - windowrect.right / 2, //窗口左上角的横坐标，
GetSystemMetrics(SM_CXSCREEN)用于获取屏幕的宽度
        GetSystemMetrics(SM_CYSCREEN) / 2 - windowrect.bottom / 2, //窗口左上角的纵坐标，
GetSystemMetrics(SM_CXSCREEN)用于获取屏幕的高度
        windowrect.right, //窗口的宽度
        windowrect.bottom, //窗口的高度
        NULL, //没有父窗口
        NULL,//使用窗口类中指定的菜单栏
        hInstance,//上文中注册的窗口类所在的模块
        NULL//自定义参数为 NULL
        );

        //以下代码用于新建线程
        BeginThread(
        NULL,//使用默认的安全属性
        0,//使用默认的线程栈大小
        RenderThreadMain,//指向渲染线程的入口点函数
        static_cast<void *>(hWnd), //传入窗口句柄供渲染线程使用
        0,//没有创建标志
        NULL//对新建线程的线程 ID 不感兴趣
        );
    }

    //进入线程的消息循环（又称作消息泵）
    MSG msg;
    while(GetMessageW(&msg,NULL,0,0))//在 Windows 中，每个线程都有一个消息队列，窗口的消息会被
发送到创建该窗口的线程的消息队列中，GetMessageW 用于从消息队列中获取消息，当消息队列为空时（即队列中
没有任何消息），GetMessageW 会导致主调线程等待，直到消息队列不再为空（即有新的消息被加入到队列中）
        DispatchMessageW(&msg);//内部会使用 GetWindowLongPtrW 获取窗口所对应的窗口过程，并调
用该窗口过程
    return (int)msg.wParam;//收到 WM_QUIT 时消息循环会终止，按照惯例，消息的 wParam 成员表示线程的执行结果
}
```

```
//窗口过程的定义
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,LPARAM lParam)
{
     switch(message)
     {
     case WM_DESTROY://窗口销毁时收到该消息
          PostQuitMessage(0);//会将 WM_QUIT 消息添加到主调线程的消息队列中，结束上文中的消息循环
     break;
     default:
          return DefWindowProcW(hWnd, message, wParam, lParam);//调用 Windows 默认的窗口过程进行处理
     }
     return 0;
}
```

1.1.5 新建 rendermain.cpp

用同样的方式创建 rendermain.cpp，并添加以下代码。

```
#include <sdkddkver.h>
#define WIN32_LEAN_AND_MEAN
#include <Windows.h>
#include <dxgi.h>
#include <d3d12.h>

DWORD WINAPI RenderThreadMain(LPVOID lpThreadParameter)
{
     HWND hWnd=static_cast<HWND>(lpThreadParameter);//即我们在 main.cpp 中传入的窗口句柄
     //在后文中会添加代码
     return 0U;//表示成功返回
}
```

1.1.6 链接 dxgi.lib 和 d3d12.lib 库

在“解决方案管理器”中右击“DX12”项目，选择“属性”，如图 1-9 所示。

图 1-9 选择“属性”

在“配置”中选择“所有配置”和“平台”中选择“所有平台”，在“配置属性”→“链接器”→“输入”中，在“附加依赖项”中加入“dxgi.lib”和“d3d12.lib”，如图 1-10 所示。

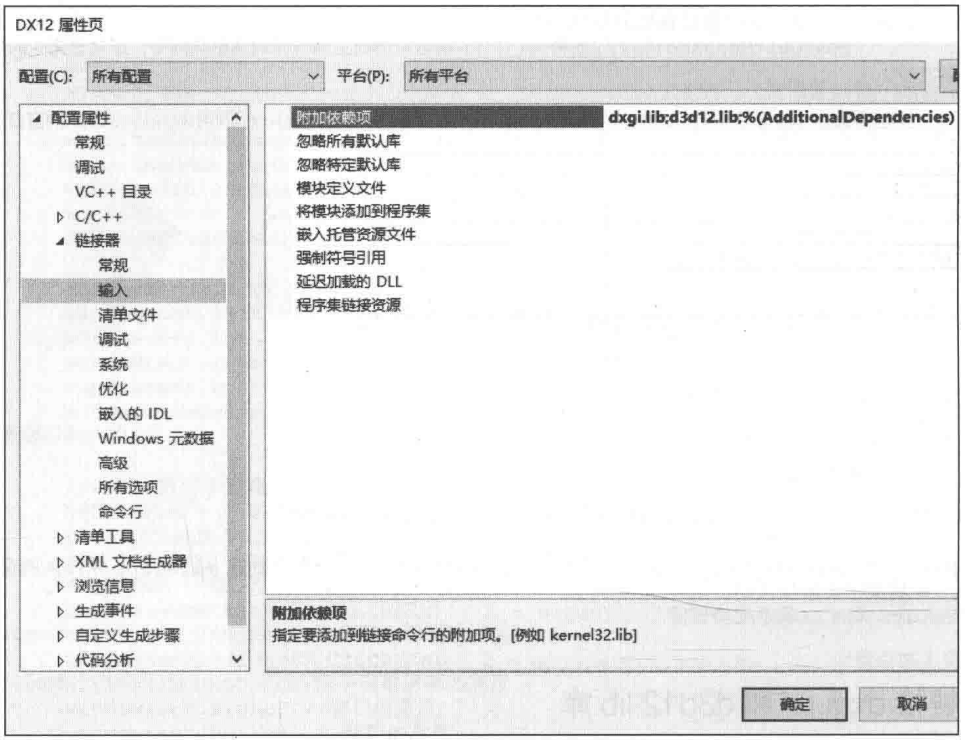

图 1-10　附加依赖项

1.1.7　生成并调试

选择工具栏中的“本地 Windows 调试器”生成并调试我们的程序，如图 1-11 所示。

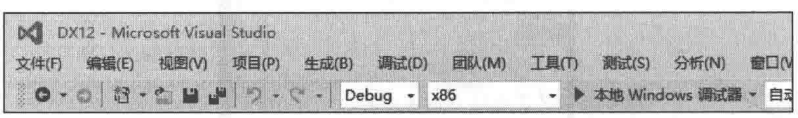

图 1-11　本地 Windows 调试器

可以在任务栏中看到我们创建的窗口，由于没有向窗口表面写入任何数据，因此窗口表面中没有任何内容可以显示，如图 1-12 所示。在接下来的章节中，我们将介绍如何用 Direct3D 12 渲染到窗口表面（严格意义上，Direct3D 12 并不直接与本地窗口系统交互，Direct3D 12 先渲染到 DXGI 交换链缓冲，然后 DXGI 再将交换链缓冲中的数据呈现到窗口表面）。

调试完毕，可以右击任务栏中的窗口图标，再单击关闭窗口，结束本次调试，如图 1-13 所示。

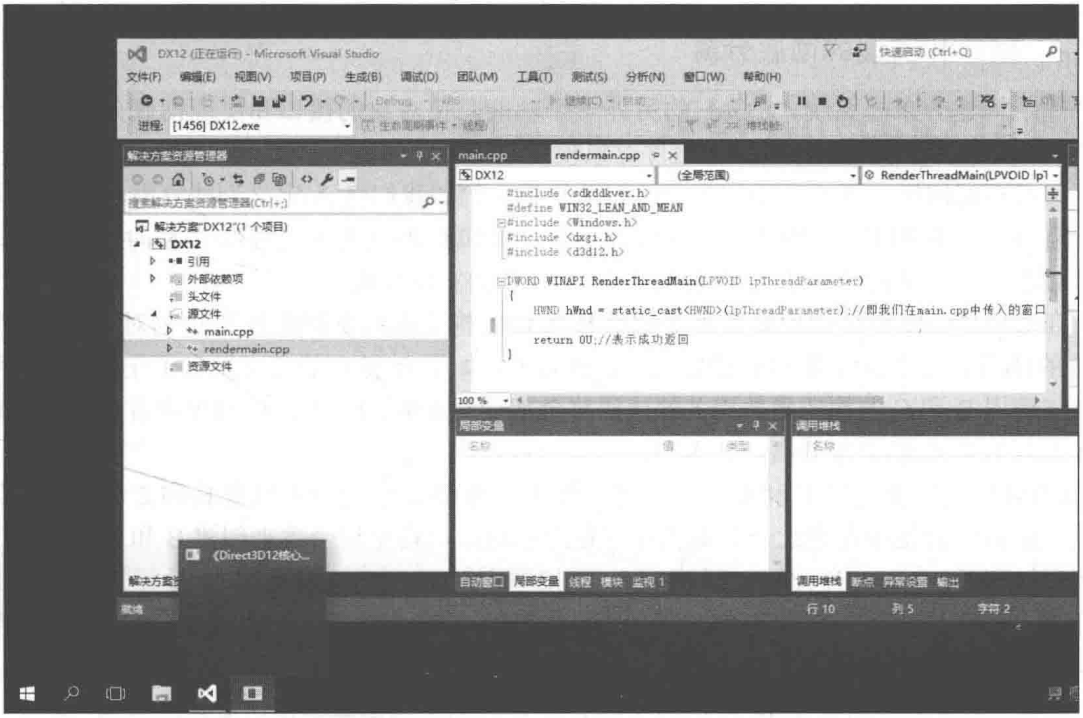

图 1-12 创建的窗口

图 1-13 结束程序运行的方式

1.2 COM 简介

由于 DXGI 和 Direct3D 都是基于 COM 构建的，因此在开始 Direct3D 12 编程前，简要介绍在接下来的编程中会用到的 COM 的相关知识，如果读者想要更深入地了解 COM，可以参阅《COM 本质论》（ISBN：9787508306117）。

1.2.1 构建分布式系统

COM 是为了构建分布式系统而设计的，可以认为是 CORBA 在 Windows 平台上的实现。

然而 DXGI 和 Direct3D 不需要用到构建分布式系统的相关功能，因此，DXGI 和 Direct3D 并没有与 COM 运行时层交互。一个很有力的证据就是，在调用 DXGI 和 Direct3D 的 API 时，并不要求主调线程事先调用 CoInitialize(Ex)进入到某个 COM 套间中。

1.2.2　接口和实现的彻底分离

除了构建分布式系统，COM 的另外一个功能是使对象的接口和实现彻底分离，这也是 COM（Component Object Model，组件对象模型）的名字的由来。

读者不妨回忆一下 C++中抽象类的相关知识。例如我们将 A 定义为抽象类，B 和 C 同时继承自 A（即 B 和 C 同时实现了接口 A），那么 B 和 C 的使用者就可以用 A 的指针统一访问 B 和 C 的对象，从而使 B 和 C 的使用者与 B 和 C 的实现分离。

在 C++中，B 和 C 的使用者需要调用 B 和 C 的构造函数来创建 B 和 C 的对象，从而使 B 和 C 的使用者必须包含 B 和 C 的定义。也就是说，如果 B 和 C 的定义发生了任何改变，那么所有与调用 B 和 C 的构造函数相关的代码将全部受到影响，即 B 和 C 的使用者并没有在真正意义上与 B 和 C 的实现分离。

COM 解决方案是由 B 和 C 的实现者，而不是 B 和 C 的使用者负责 B 和 C 的对象的创建过程。B 和 C 的使用者通过一个实现者事先约定的 C 风格全局函数来创建 B 和 C 的对象（在 DXGI 中是 CreateDXGIFactory，在 Direct3D 12 中是 D3D12CreateDevice，见后文），从而使 B 和 C 的使用者不再需要包含 B 和 C 的定义。如果 B 和 C 的定义发生任何改变，只要抽象类 A 不发生变化，B 和 C 的使用者就不会受到任何影响。

然而 COM 并没有规定 B 和 C 的实现者在 C 风格全局函数中创建对象的方式，并不一定是 new，因此 B 和 C 的使用者并不一定可以用 delete 来销毁对象。实现者需要事先约定某个函数，在其中定义了与对象的创建方法相兼容的销毁方式，供 B 和 C 的使用者调用。

为了统一，COM 对此进行了规定：所有的抽象类 A 都继承自 IUnknown 接口，B 和 C 的使用者通过 IUnknown 接口的 Release 方法来销毁对象。

关于 COM 的知识暂时就介绍这么多，如果在接下来的章节中需要用到关于 COM 的其他知识，那么会在相应的章节中进行介绍。

章末小结

本章我们创建了一个 DirectX 12 项目，该项目会贯穿本书使用。并且，由于 Direct3D 12 是基于 COM 架构的，因此，简单介绍了 COM 的相关知识。