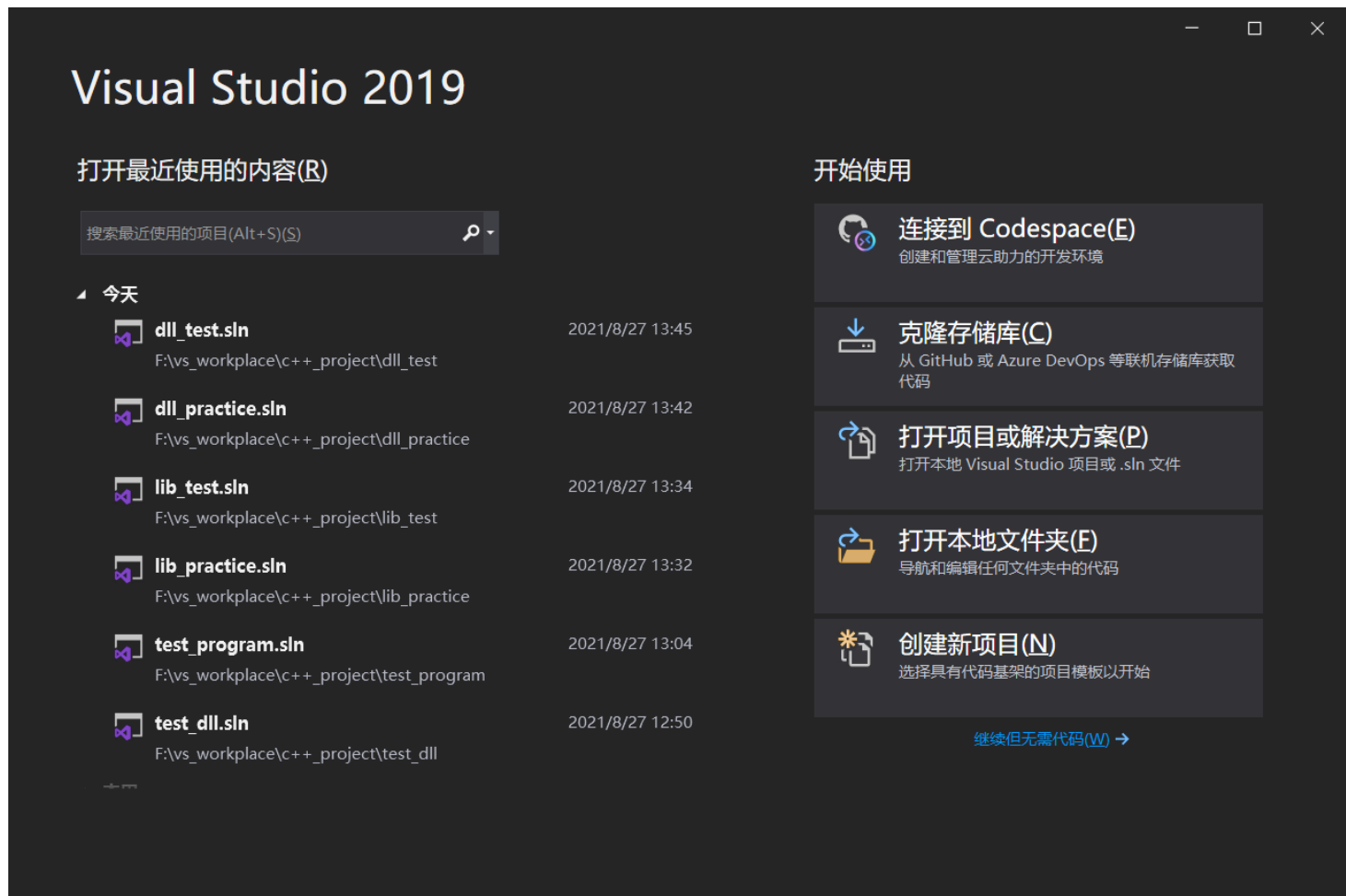
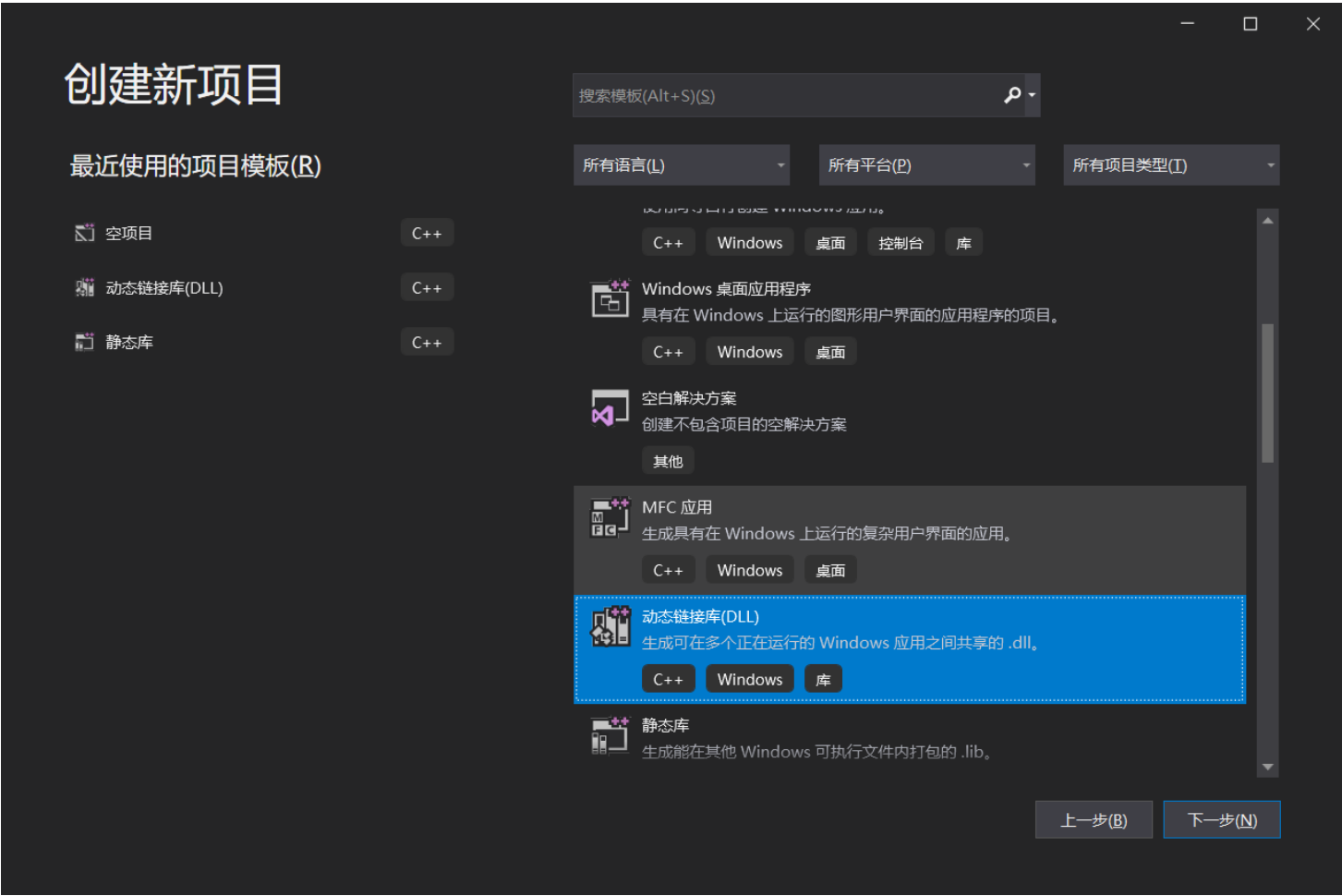


# vs2019编写dll（动态库）

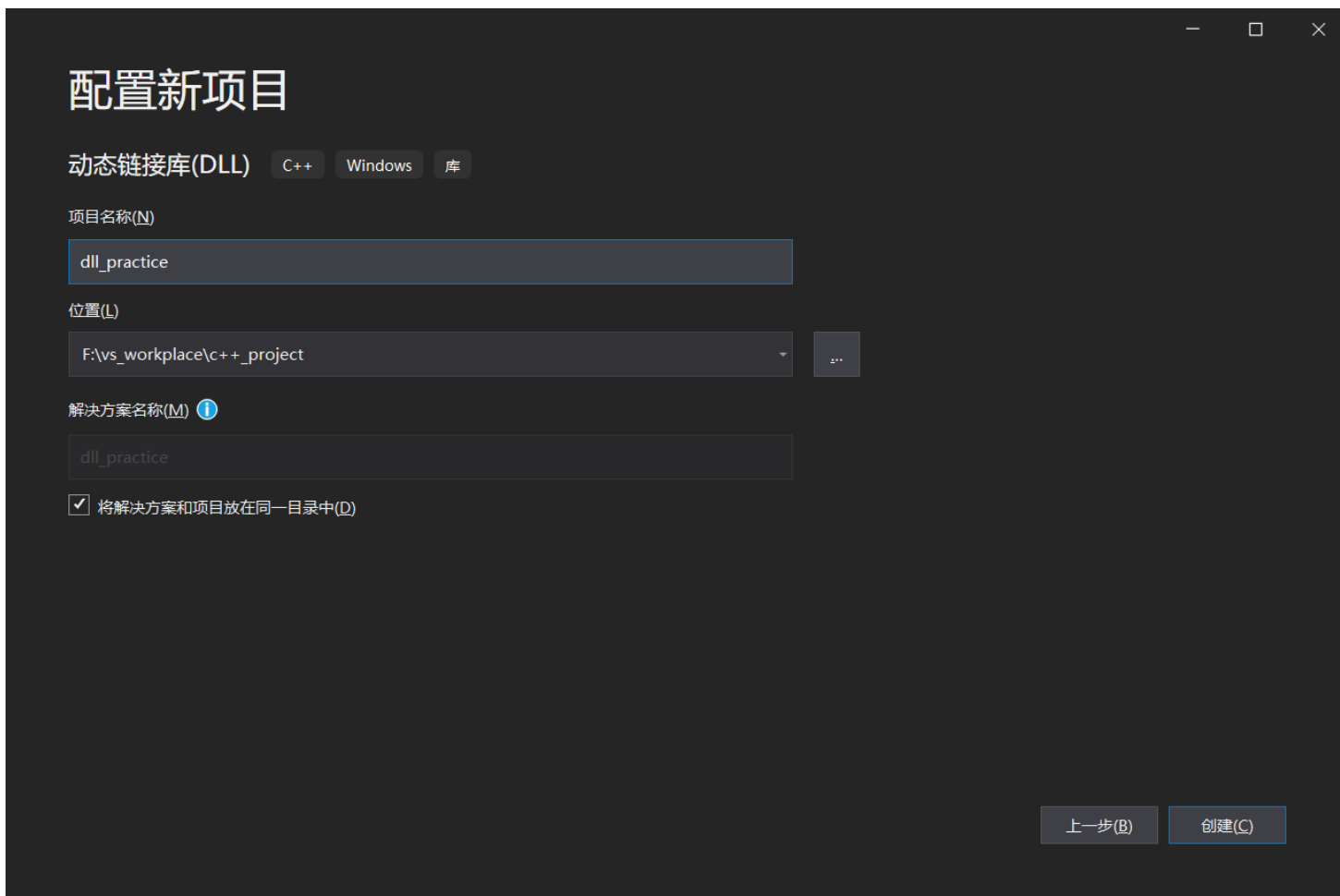
打开vs2019



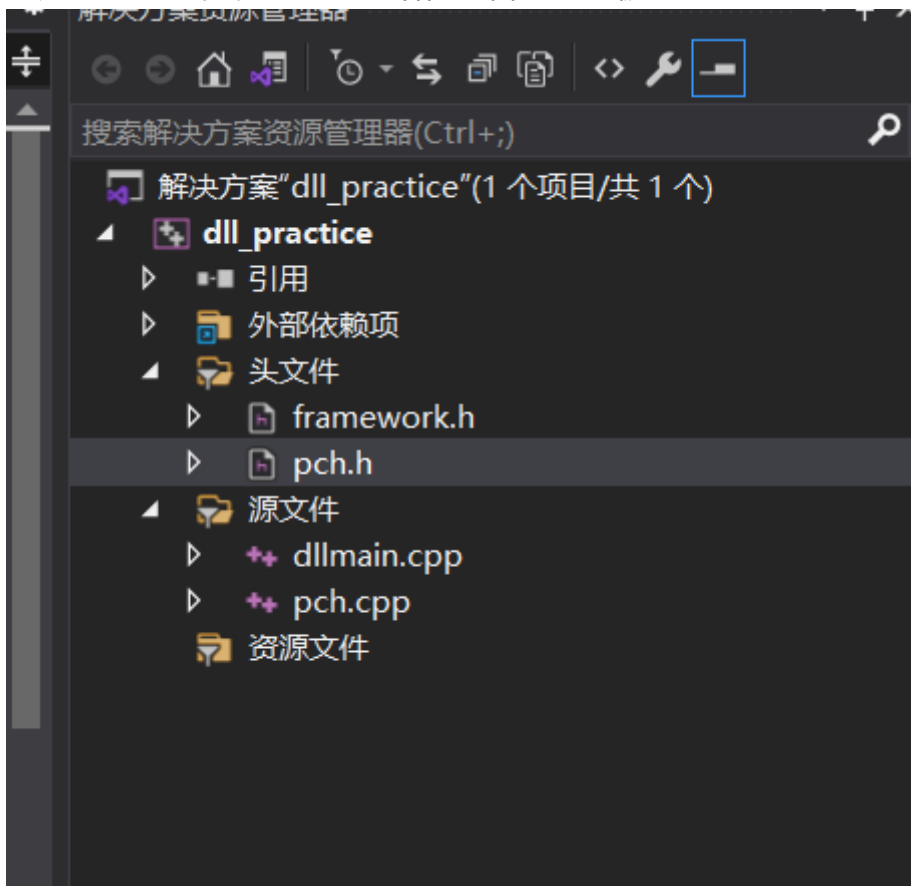
选择创建新项目，选择动态链接库



写入项目名称



创建好后进入，发现vs已经给你准备好了模板。



我们先看dllmain.cpp

```
// dllmain.cpp : 定义 DLL 应用程序的入口点。
#include "pch.h"
#include <stdio.h>
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD ul_reason_for_call,
                       LPVOID lpReserved
                       )
{
    switch (ul_reason_for_call)
    {
```

dllmain.cpp中是一个DllMain的函数（其实是dll的入口点）

我们知道 dll是在运行程序期间动态载入的。

每个dll在载入运行程序时都会自动执行dllmain函数。

这里我们可以暂时忽略它。

framework.h内容如下：

```
#pragma once

#define WIN32_LEAN_AND_MEAN // 从 Windows 头文件中排除极少使用的内容
// Windows 头文件
#include <windows.h>
```

其实对我们来说也没什么用，不管就是了。

主要要编写的是pch.h和pch.cpp两个文件

pch.h的文件格式如下

```
// 请勿在此处添加要频繁更新的文件，这将使得性能优势无效。

#ifndef PCH_H
#define PCH_H

// 添加要在此处预编译的标头
#include "framework.h"
extern "C" __declspec(dllexport) int add(int x, int y);
extern "C" __declspec(dllexport) int compare(int x, int y);
#endif //PCH_H
```

该文件的作用就是声明导出口。

(比如我们想要使用dll库中的函数，那么我们需要在pch.h中声明类似  
extern "C" \_declspec(dllexport) int add(int x, int y)  
格式的语句。  
其作用就是让dll中的函数能够被外界程序调用。

pch.cpp的作用就是编写函数功能了。

```
ice (全局范围)
// pch.cpp: 与预编译标头对应的源文件

#include "pch.h"

// 当使用预编译的头时，需要使用此源文件，编译才能成功。
int add(int x, int y) {
    return x + y;
}
int compare(int x, int y) {
    return x > y ? x : y;
}
```

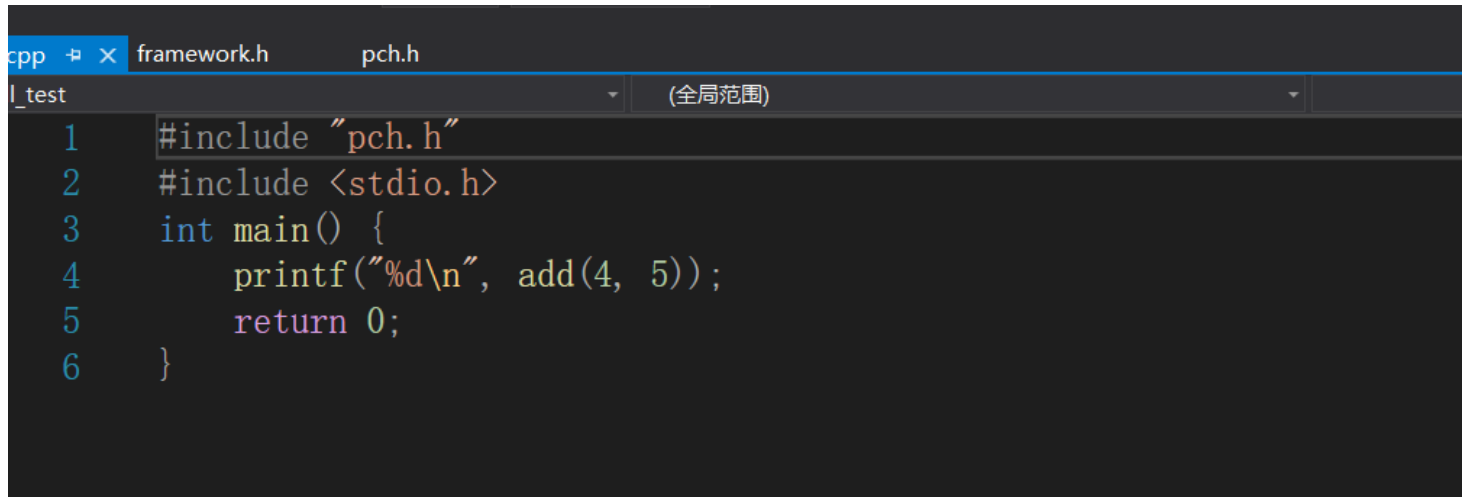
好了  
现在一个dll的大概内容就完成了  
然后生成解决方案



创建完成后会在项目文件夹中找的  
dll\_practice.dll  
dll\_practive.lib两个文件

## vs2019使用dll

用vs2019创建一个空项目  
将创建好的项目中添加pch.h,framwork.h (把2个头文件复制到空项目中)  
然后在资源文件中添加dll\_practive.lib文件  
然后编写程序代码



```
cpp  x framework.h pch.h
l_test
1  #include "pch.h"
2  #include <stdio.h>
3  int main() {
4      printf("%d\n", add(4, 5));
5      return 0;
6  }
```

生成解决方案即可。  
生成后，要把dll\_practive.dll放到生成的exe的同一文件夹下才可运行！

**lib编写和dll类似**