

# Visual Studio libtorch 使用教程

## 预备软件

- **libtorch**: cpu 或 gpu 版本均可, 根据本机环境选择, 但要注意下载 nightly build 版, 因 stable 版头文件引用有错误。
- **visual studio**: 本教程在 vs2015 与 2019 版本下均调试通过, 根据本机环境选择。
- **\*cmake**: 可选, 当教程中其他方法均失败情况下尝试。

## 使用流程

1. 下载对应版本 libtorch 并解压到对应目录, 如在本机为 C:\Users\chenjz\Downloads\libtorchGPU。
2. 打开 visual studio 创建新项目, 添加源代码 test.cpp

```
#include <iostream>
#include <torch/torch.h>

using namespace std;

int main()
{
    torch::Tensor tensor = torch::eye(3);
    std::cout << tensor.cuda() << std::endl;
    return 0;
}
```

3. 右键项目属性, 总共有三项需要配置:

```
C:\Users\chenjz\Downloads\libtorchGPU\include;C:\Users\chenjz\Downloads\libtorchGPU\include  
C:\Users\chenjz\Downloads\libtorchGPU\lib\torch.lib  
C:\Users\chenjz\Downloads\libtorchGPU\lib\c10.lib  
C:\Program Files\NVIDIA Corporation\NvToolsExt\lib\x64\nvToolsExt64_1.lib  
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\lib\x64\cudart.lib  
C:\Users\chenjz\Downloads\libtorchGPU\lib\c10_cuda.lib  
C:\Users\chenjz\Downloads\libtorchGPU\lib\caffe2_gpu.lib  
C:\Users\chenjz\Downloads\libtorchGPU\lib\caffe2.lib  
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\lib\x64\cufft.lib  
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\lib\x64\curand.lib  
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\lib\x64\cudnn.lib  
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\lib\x64\cublas.lib  
PATH=%PATH%;C:\Users\chenjz\Downloads\libtorchGPU\lib
```

配置 1 设置引用目录, 配置 2 设置引用静态库, 配置 3 设置 dll 引用路径, 若使用的是 cpu 版本可以将配置中有关 cuda 的部分全部删除。

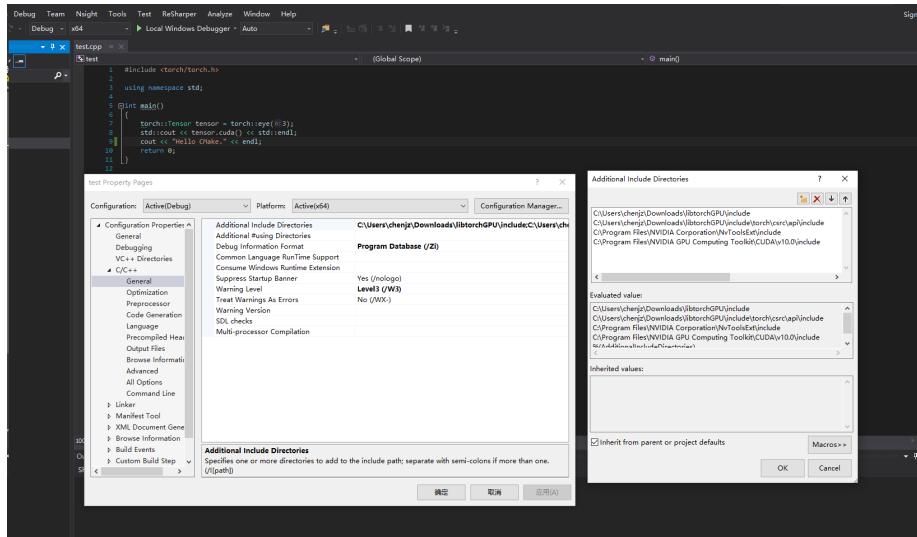


Figure 1: pic1

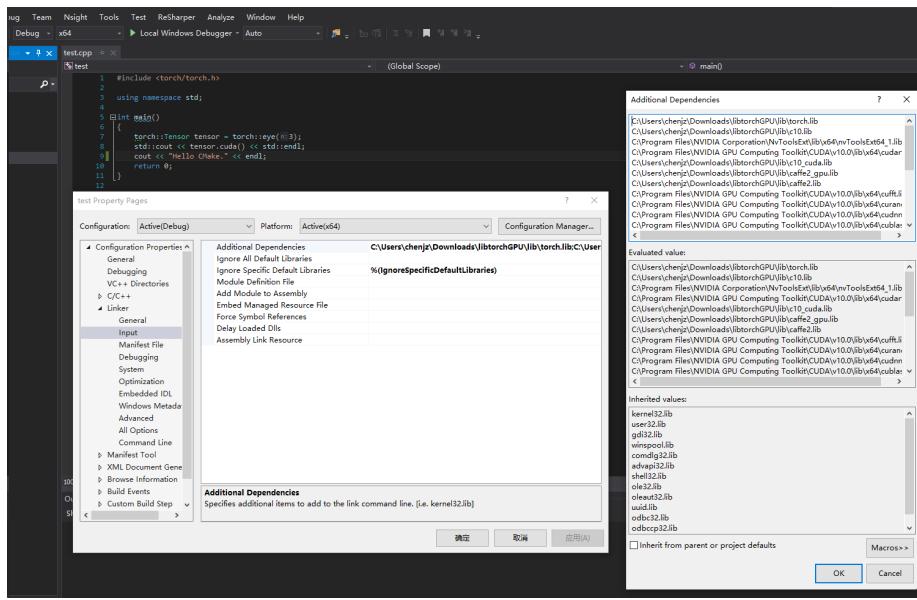


Figure 2: pic2

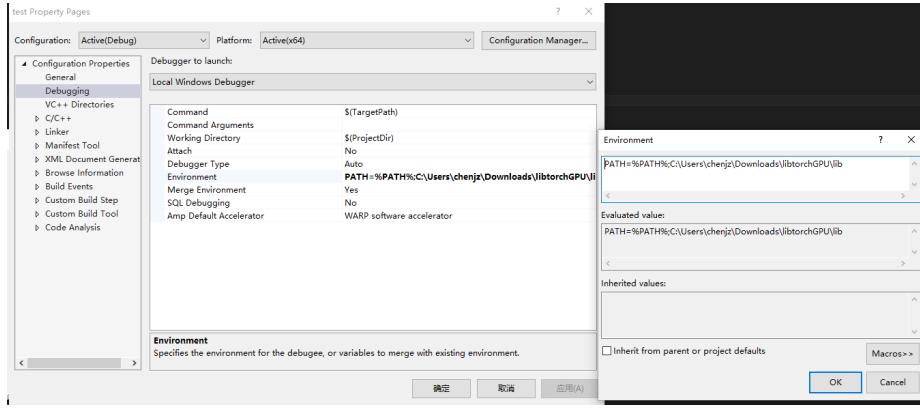


Figure 3: pic3

4. 编译并链接程序：有人 build 项目时可能会报错 C4996，这是 msrv 为了安全所引用的特性，可以在项目配置->C/C++->Advanced->Disable Specific Warnings 里添加 4996。

\* 在已有项目里添加 libtorch 同样需要以上三步。

运行成功！

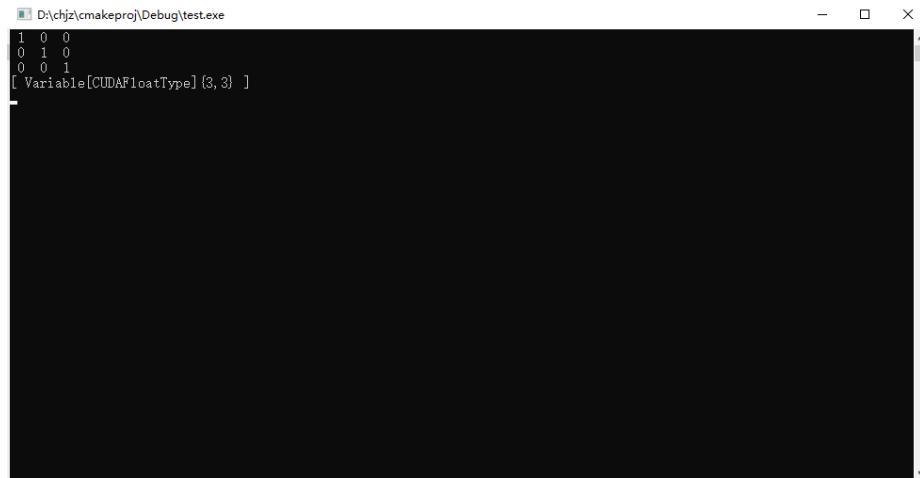
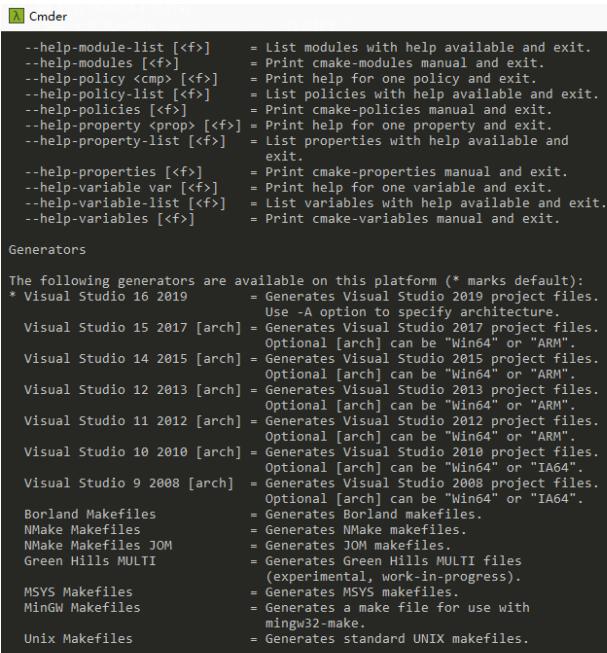


Figure 4: pic4

## 在上述方法失效的情况下从 cmake 重新创建工程

以上方法依赖于 libtorch 现有的项目结构，故可能不适用于未来版本。在这种时候需要从 cmake 项目开始重新生成 sln 项目。

### 检查 cmake 版本与功能



```
cmder
--help-module-list [<f>] = List modules with help available and exit.
--help-modules [<f>] = Print cmake-modules manual and exit.
--help-policy <cmp> [<f>] = Print help for one policy and exit.
--help-policy-list [<f>] = List policies with help available and exit.
--help-policies [<f>] = Print cmake-policies manual and exit.
--help-property <prop> [<f>] = Print help for one property and exit.
--help-property-list [<f>] = List properties with help available and exit.
--help-properties [<f>] = Print cmake-properties manual and exit.
--help-variable var [<f>] = Print help for one variable and exit.
--help-variable-list [<f>] = List variables with help available and exit.
--help-variables [<f>] = Print cmake-variables manual and exit.

Generators
The following generators are available on this platform (* marks default).
* Visual Studio 16 2019 = Generates Visual Studio 2019 project files.
  Use -A option to specify architecture.
Visual Studio 15 2017 [arch] = Generates Visual Studio 2017 project files.
  Optional [arch] can be "Win64" or "ARM".
Visual Studio 14 2015 [arch] = Generates Visual Studio 2015 project files.
  Optional [arch] can be "Win64" or "ARM".
Visual Studio 12 2013 [arch] = Generates Visual Studio 2013 project files.
  Optional [arch] can be "Win64" or "ARM".
Visual Studio 11 2012 [arch] = Generates Visual Studio 2012 project files.
  Optional [arch] can be "Win64" or "ARM".
Visual Studio 10 2010 [arch] = Generates Visual Studio 2010 project files.
  Optional [arch] can be "Win64" or "IA64".
Visual Studio 9 2008 [arch] = Generates Visual Studio 2008 project files.
  Optional [arch] can be "Win64" or "IA64".
Borland Makefiles = Generates Borland makefiles.
NMake Makefiles = Generates NMake makefiles.
NMake Makefiles JOM = Generates JOM makefiles.
Green Hills MULTI = Generates Green Hills MULTI files
  (experimental, work-in-progress).
MSYS Makefiles = Generates MSYS makefiles.
MinGW Makefiles = Generates a make file for use with
  mingw32-make.
Unix Makefiles = Generates standard UNIX makefiles.
```

输入命令 `cmake -h`, 检查 generator 选项, 样例输出如下图

### 手动编写 CMakeLists.txt 文件, 示例如下

```
cmake_minimum_required (VERSION 3.8)
set(CMAKE_CXX_STANDARD 11)
find_package(Torch REQUIRED)
add_executable (CMakeProject "CMakeProject.cpp" "CMakeProject.h")
target_link_libraries(CMakeProject ${TORCH_LIBRARIES})
```

使用 cmake 自带的 generator 生成 vs 项目，以 vs2015 为例

```
cmake -DCMAKE_PREFIX_PATH=C:\Users\chenjz\Downloads\libtorchGPU  
-DCMAKE_BUILD_TYPE=Debug -G"Visual Studio 14 2015 Win 64" .
```

若上述操作无误项目文件夹应该如下图，Project.sln 为生成项目。

名称	修改日期	类型	大小
CMakelists.txt	2019/4/5 20:46	文本文件	15 KB
ALL_BUILD.vcxproj	2019/4/5 20:46	VC++ Project	45 KB
ALL_BUILD.vcxproj.filters	2019/4/5 20:46	VC++ Project Fil...	1 KB
cmake_install.cmake	2019/4/5 20:46	CMAKE 文件	2 KB
CMakeCache.txt	2019/4/5 20:46	文本文档	15 KB
CMakeLists.txt	2019/4/4 20:59	文本文档	1 KB
CMakeProject.cpp	2019/4/5 9:28	C++ 源文件	1 KB
CMakeProject.h	2019/4/4 19:42	C Header 源文件	1 KB
CMakeProject.vcxproj	2019/4/5 20:46	VC++ Project	57 KB
CMakeProject.vcxproj.filters	2019/4/5 20:46	VC++ Project Fil...	1 KB
Project.sln	2019/4/5 20:46	Visual Studio Sol...	4 KB
ZERO_CHECK.vcxproj	2019/4/5 20:46	VC++ Project	44 KB
ZERO_CHECK.vcxproj.filters	2019/4/5 20:46	VC++ Project Fil...	1 KB

Figure 5: result

## 项目说明

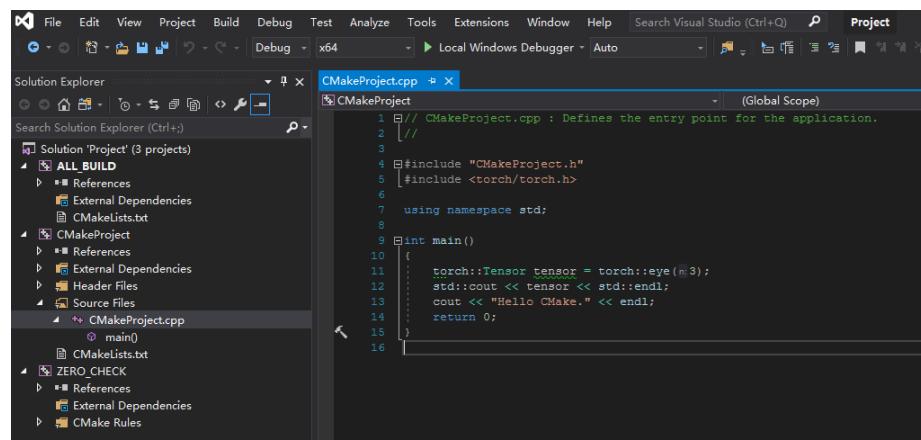
Project.sln 共包含三个项目： ALL\_BUILD, CMakeProject, ZERO\_CHECK，其中 CMakeProject 是我们想要的。

CMakeProject 下的 External Dependencies 里包含了所有与 libtorch 有关的头文件，配置好 dll 路径后若无错误就可以直接编译运行了。

## 在 HM 中调用代码

我在 github 上提供了一个 HM16.20 的 cmake 项目，大家可以在对应位置修改 CMakeLists.txt 来在指定项目调用 libtorch。

- 在 C++ 中加载 pytorch 模型教程 [https://pytorch.org/tutorials/advanced/cpp\\_export.html](https://pytorch.org/tutorials/advanced/cpp_export.html)
- HM cmake 工程 [https://github.com/chjz1024/HM\\_vc2015](https://github.com/chjz1024/HM_vc2015)



The screenshot shows the Visual Studio IDE interface. The menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search Visual Studio (Ctrl+Q). The toolbar has icons for file operations like Open, Save, and Build. The status bar indicates "x64" and "Local Windows Debugger - Auto".

The Solution Explorer window on the left lists three projects: ALL\_BUILD, CMakeProject, and ZERO\_CHECK. The CMakeProject node is expanded, showing Source Files, Header Files, External Dependencies, References, and CMake Rules. A file named "CMakeProject.cpp" is selected in the Source Files list.

The code editor window on the right displays the contents of "CMakeProject.cpp". The code is as follows:

```
1 // CMakeProject.cpp : Defines the entry point for the application.
2 //
3
4 #include "CMakeProject.h"
5 [#include <torch/torch.h>
6
7 using namespace std;
8
9 int main()
10 {
11     torch::Tensor tensor = torch::eye(3);
12     std::cout << tensor << std::endl;
13     cout << "Hello CMake." << endl;
14     return 0;
15 }
16
```

Figure 6: result